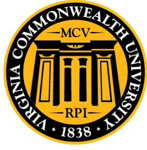


Virginia Commonwealth University
VCU Scholars Compass

Theses and Dissertations | Graduate School

2005

Attribute-Level Versioning: A Relational Mechanism for Version Storage and Retrieval

Charles Andrew Bell
Virginia Commonwealth University

Follow this and additional works at: https://scholarscompass.vcu.edu/etd

Part of the Computer Sciences Commons

© The Author

Downloaded from
https://scholarscompass.vcu.edu/etd/988

This Dissertation is brought to you for free and open access by the Graduate School at VCU Scholars Compass. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of VCU Scholars Compass. For more information, please contact libcompass@vcu.edu.

المنارة للاستشارات

www.manaraa.com

School of Engineering
Virginia Commonwealth University

This is to certify that the dissertation prepared by Charles Andrew Bell entitled "Attribute-Level Versioning: A Relational Mechanism for Version Storage and Retrieval" has been approved by his committee as satisfactory completion of the dissertation requirement for the degree of Doctor of Philosophy.

James E. Ames IV, Ph.D., Associate Professor, School of Engineering

---

Lorraine M. Parker, Ph.D., Associate Professor, School of Engineering

---

Susan Brilliant, Ph.D., Associate Professor, School of Engineering

---

David Primeaux, Ph.D., Associate Professor, Interim Chairperson of the Computer Science Department, School of Engineering

---

Jason Merrick, Ph.D., Associate Professor, School of Humanities and Sciences

---

Thomas Overby, Ph.D., Assistant Dean of Graduate Affairs, School of Engineering

---

Robert Mattauch, Ph.D., Dean, School of Engineering

F. Douglas Boudinot, Ph.D., Dean, School of Graduate Studies

Date

المنارة للاستشارات
www.manaraa.com

© Charles Andrew Bell 2005
All Rights Reserved[†]

[†]The government of the United States of America retains certain rights to the use of this work. All other rights reserved.

المنارة للاستشارات

www.manaraa.com

Attribute-Level Versioning:
A Relational Mechanism for Version Storage and Retrieval

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Engineering at Virginia Commonwealth University

by

Charles A. Bell
Bachelor of Science, Computer Science, Virginia Commonwealth University, 1995
Master of Science, Computer Science, Virginia Commonwealth University, 1997
Doctor of Philosophy, Virginia Commonwealth University, 2005

Directors: James E. Ames IV, Ph.D., Associate Professor, School of Engineering
and Lorraine M. Parker, Ph.D., Associate Professor, School of Engineering

Virginia Commonwealth University
Richmond, Virginia
December 2005

المنارة للاستشارات
www.manaraa.com

Acknowledgement

It is by the grace of God alone that this work has been completed – I couldn't do it without you, Lord. I am also very grateful for the encouragement of my wife Annette, my family, and my coworkers. With special thanks to Grandmother Bell; you were right. Special regards to my Middle School Principal; where I am glad to say you were wrong.

المنارة للاستشارات

www.manaraa.com

Table of Contents

Chapter One – Introduction 1

1.1 Background 2

1.2 What is Data Versioning? 6

1.2.1 Definition of Versioning 6

1.2.2 Versioning Explained 7

1.2.3 Ship of Theseus – Version ad Infinitum 9

1.3 What is Attribute-Level Versioning? 11

1.3.1 Definitions 12

1.3.2 ALV Requirements 13

1.3.3 Goals 15

1.5 Dissertation Overview 17

Chapter Two – Existing Solutions, Technologies and Theories 19

2.1 Document Management Systems 19

2.1.1 Technologies 21

2.1.2 Applications 24

2.1.3 Comparison with ALV 26

2.2 Relational Database Systems 27

2.2.1 Technologies 29

المنارة للاستشارات

www.manaraa.com

2.2.2 Applications 31
2.2.3 Comparison with ALV 32
2.3 Temporal Database Systems 33
2.3.1 Technologies 35
2.3.2 Applications 39
2.3.3 Comparison with ALV 40
2.4 Object-Oriented Database Systems 41
2.4.1 Technologies 43
2.4.2 Applications 46
2.4.3 Comparison with ALV 46
2.5 Object Relational Database Systems 47
2.5.1 Technologies 48
2.5.2 Applications 49
2.5.3 Comparison with ALV 51
2.6 Other Considerations 51
2.6.2 Legacy Application Support 53
2.6.3 Graph Stores 55
2.6.4 Spatial Data and Temporal Databases 56
2.6.5 Long Transactions 57
2.6.6 Versioning Requirements 58
2.6.6.1 Transparency 58
2.6.6.2 Multiple Stores 59

المنارة للاستشارات
www.manaraa.com

2.6.6.3 Implemented as an Extension 60
2.6.6.4 Support for Recovery 61
2.6.7 Concept versus Form 61
2.7 Conclusion 63
Chapter Three – Introduction to ALV Technologies 65
3.2 Advanced Storage and Retrieval Mechanism 70
3.3 Advanced Query Mechanism 75
3.4 Advanced Index Mechanism 79
3.5 Data Mining Algorithm Applications 81
3.6 Application of Emerging Technologies in Conjunction with ALV 84
3.8 To be continued… 90
Chapter Four - A Clustered Storage Mechanism for Versioning 92
4.1 Introduction 92
4.2 Background 93
4.2.1 File Organization Techniques 96
4.2.1.1 Access Methods 97
4.2.1.2 Extended Blocks 99
4.2.1.3 Free Blocks 100
4.2.1.4 A Comment about Data Independence 100
4.2.1.5 Buffer Manager 101
4.2.1.6 Shadow Paging 108
4.2.1.7 Sparse Files 110

المنارة للاستشارات
www.manaraa.com

4.2.1.8 Transposed Files 110
4.2.2 Version Store Implementations 111
4.2.2.1 Horizontal 112
4.2.2.2 Vertical 115
4.2.2.3 Are There Alternative Implementations for a Version Store? 117
4.2.2.4 The Use of Superkeys 122
4.2.3 Clustered File Organization 122
4.3 Clustered Version Store 126
4.3.1 Technology Descriptions 128
4.3.1.1 Physical Design Goals 128
4.3.1.2 Attribute Chains 130
4.3.1.3 Secondary Representation 132
4.3.2 Execution Sequence 133
4.3.3 Class Descriptions 136
4.3.4 The ALV Buffer Manager 138
4.3.4.1 Integration with Physical Data Store 139
4.3.4.2 Concurrency Support 143
4.4 Analysis 145
4.4.1 Caching and Performance 146
4.4.2 Blocksize Experiment 147
4.4.3 Real World Performance 149
4.5 Conclusion 154

المنارة للاستشارات
www.manaraa.com

4.6 Future Work ... 154
Chapter Five – An Indexing Mechanism for fast Version Retrieval ... 156
5.1 Introduction ... 156
5.2 Background ... 157
5.2.1 Indexing Methods ... 160
5.2.1.1 Indexed Sequential Files ... 162
5.2.1.2 Hashing ... 163
5.2.1.3 B Trees ... 166
5.2.1.4 B+ Trees ... 171
5.2.2 Concurrency Issues ... 175
5.2.3 Transaction Processing Issues ... 178
5.2.4 Performance Issues ... 179
5.3 Version Indexing ... 180
5.3.1 Technology Description ... 183
5.3.1.1 B^2+ Tree ... 184
5.3.1.2 mB^2+ Tree ... 187
5.3.2 Execution Sequence ... 190
5.3.3 Class Descriptions ... 192
5.3.3.1 bptNode ... 192
5.3.3.2 bptIndex ... 194
5.3.3.3 ALVDataFile ... 194
5.3.4 Buffering and Transactions ... 196

المنارة للاستشارات
www.manaraa.com

5.3.4.1 The role of the ALV Buffer Manager 196

5.3.4.2 Transactions in ALV 197

5.3.4.3 Deadlock Prevention 200

5.4 Analysis..201

5.4.1 Index Experiments 201

5.4.2 Real World Performance 207

5.5 Conclusion..207

5.6 Future Work 208

Chapter Six - A Query Optimizer and Execution Engine for Versioning......................210

6.1 Introduction 210

6.2 Background 211

6.2.1 Query Language...215

6.2.2 Query Optimization Strategies 217

6.2.3.1 Relational Calculus and Relational Algebra..224

6.2.3.2 Query Trees...228

6.2.4 Optimizers 230

6.2.4.1 Cost-based Optimizers 233

6.2.4.2 Heuristic Optimizers 238

6.2.4.3 Semantic Optimizers 239

6.2.4.4 Parametric Optimizers 240

6.2.5 Query Execution 241

6.3 ALV Query Optimizer and Execution Engine 245

المنارة للاستشارات

www.manaraa.com

6.3.1 Technology Descriptions....................245
6.3.1.1 ALV Query Tree....................246
6.3.1.2 ALV Query Optimizer....................248
6.3.1.3 Rules for Query Tree Optimizations....................250
6.3.2 ALV Query Execution....................252
6.3.3 Class Descriptions....................252
6.3.3.1 Query Transformation....................253
6.3.3.2 ALV QueryTree....................254
6.3.3.3 ALVExecute....................258
6.3.4 SQL_{ALV} Commands....................260
6.3.4.1 Select Command....................261
6.3.4.2 Create Command....................262
6.3.4.3 Drop Table Command....................263
6.3.4.4 Drop Database Command....................264
6.3.4.5 Insert Command....................264
6.3.4.6 Delete Command....................265
6.3.4.7 Update Command....................266
6.3.4.8 Show Tables Command....................267
6.3.4.9 Show Databases Command....................267
6.3.4.10 Explain Table Command....................267
6.3.4.11 Explain Query Command....................268
6.3.4.12 Backup Command....................269

المنارة للاستشارات
www.manaraa.com

6.3.4.13 Restore Command 270

6.3.4.14 Version() Function 270

Chapter Seven - Data Mining for Version Analysis 278

7.1 Introduction 278

7.2 Background 279

7.2.1 Knowledge Discovery in Databases 281

7.2.2 Machine Learning 283

7.2.3 Data Mining 285

7.2.3.1 Applications of Data Mining 287

7.2.3.2 Data Mining Algorithms 288

7.2.3.3 The Data Mining Process 291

7.2.3.4 Analyzing the Results of Data Mining 296

7.2.3.5 What about privacy? 298

7.2.3.6 Data Mining Tools and Environments 299

7.2.3.7 Machine Learning versus Statistics: What's the difference? 301

7.3 Data Mining in a Versioning Environment 301

7.3.1 How are Attribute Versions Created? 302

7.3.2 ALV Metadata Preparation 303

7.3.3 Algorithm Choice 306

7.4 Analysis 308

7.5 Conclusion 314

7.6 Future Work 314

المنارة للاستشارات

www.manaraa.com

Chapter Eight - Conclusion....316

8.1 Analysis of the ALV Experiment....316

8.2 Conclusion....317

8.3 Future Work....320

8.3.1 Clustered Version Store....321

8.3.2 Version Indexing....322

8.3.3 ALV Query Optimizer and Execution Engine....324

8.3.4 Data Mining in a Versioning Environment....325

8.3.5 Other Areas....326

Bibliography....328

Appendix A – Dataset Descriptions....345

Appendix B – Modifying MySQL for use with ALV....359

المنارة للاستشارات

www.manaraa.com

List of Tables

Table 1-1: An Example Set of Versions 7

Table 2-1: A multi-valued attribute temporal “employee" relation 37

Table 3-1: Sample SQLALV Commands 77

Table 4-1: Performance Tradeoffs 95

Table 4-2: File Retrieval Experiment Data 153

Table 5-1: Results of Indexing Experiments 203

Table 5-2: Description of Data for Indexing Experiments 206

Table 6-1:The Logical and Physical Models of Database Design 212

Table 6-2: Examples of Relational Algebra and Tuple Relational Algebra Expressions227

Table 6-3: Internal Representation Requirements.. 230

Table 6-4: Query Transformation Data 272

Table 6-5: Query Optimization Data... 273

Table 6-6: Query Execution Data 275

Table 7-1: Data Mining Functions 290

Table 7-2: Data Mining Products.. 300

Table 7-3: Example Suppositions for Mining Version Metadata 302

المنارة للاستشارات

www.manaraa.com

List of Figures

Figure 3-1: The ALV File Storage Layout 74

Figure 4-1: Simplified Horizontal Database Example 113

Figure 4-2: Horizontal SQL Statement 114

Figure 4-3: Simplified Vertical Database Example 115

Figure 4-4: Vertical SQL Statement 116

Figure 4-5: Sample Transaction SQL Statements 121

Figure 4-6: Extents Addressing 129

Figure 4-7: Attribute Chains 130

Figure 4-8: Attribute Chain Layout with Block Headers 131

Figure 4-9: ALV Execution Sequence 135

Figure 4-10: Major ALV C++ Classes 137

Figure 4-11: ALV Data Structures 140

Figure 4-12: ALV Binary Data File Format 140

Figure 4-13: The ALV Physical Data Store Header and Block Diagram 141

Figure 4-14: File Access Experiment 148

Figure 4-15: Walk by Cluster Experiment 149

Figure 4-16: Results of File Retrieval Experiments - Small Table 151

المنارة للاستشارات

www.manaraa.com

Figure 4-17: Results of File Retrieval Experiments - Medium Table 151
Figure 4-18: Results of File Retrieval Experiments - Large Table 152
Figure 5-1: Multiway Tree Structure 160
Figure 5-2: Indexed Sequential File Structure 162
Figure 5-3: Conceptual B Tree with Block Addresses 169
Figure 5-4: B+ Tree Configuration 170
Figure 5-5: Conceptual B+ Tree with Block Addresses 174
Figure 5-6: B2+ Class Diagram 184
Figure 5-7: mB2+ Tree Node View 189
Figure 5-8: Execution Sequence for the B2+ Tree 191
Figure 5-9: Version Indexing File Format 195
Figure 5-10: Results of Indexing Experiment - Small Table 203
Figure 5-11: Results of Indexing Experiment - Medium Table 204
Figure 5-12: Results of Indexing Experiment - Large Table 204
Figure 6-1: Query Processing Steps 214
Figure 6-2: Typical Database System Implementation 220
Figure 6-3: Query Tree Example 229
Figure 6-4: The MySQL Query Processing Methodology 243
Figure 6-5: The ALV Query Tree Node Structure 254
Figure 6-6: Simplified ALV Query Execution Sequence 259
Figure 7-1: The Knowledge Discovery Process 282
Figure 7-2: The Data Mining Process 291

المنارة للاستشارات
www.manaraa.com

Figure 7-3: A Data Mining Cube 294

Figure 7-4: Results of Clustering - Source versus Reliability, Clusters Highlighted 310

Figure 7-5: Results of Clustering - Source versus Reliability, Sources Highlighted...... 311

Figure 7-6: Results of Clustering - Source versus Confidence, Sources Highlighted 312

Figure 7-7: Results of Clustering - Source versus Sensitivity, Sources Highlighted...... 313

للاستشارات المنارة

www.manaraa.com

Abstract

ATTRIBUTE-LEVEL VERSIONING: A RELATIONAL MECHANISM FOR VERSION STORAGE AND RETRIEVAL

By Charles A. Bell, Ph.D.

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Engineering at Virginia Commonwealth University

Virginia Commonwealth University, 2005

Directors: James E. Ames IV, Ph.D., Associate Professor, School of Engineering and Lorraine M. Parker, Ph.D., Associate Professor, School of Engineering

Data analysts today have at their disposal a seemingly endless supply of data and repositories hence, datasets from which to draw. New datasets become available daily thus making the choice of which dataset to use difficult. Furthermore, traditional data analysis has been conducted using structured data repositories such as relational database management systems (RDBMS). These systems, by their nature and design, prohibit duplication for indexed collections forcing analysts to choose one value for each of the available attributes for an item in the collection. Often analysts discover two or more datasets with information about the same entity. When combining this data and transforming it into a form that is usable in an RDBMS, analysts are forced to deconflict the collisions and choose a single value for each duplicated attribute containing differing

المنارة للاستشارات

www.manaraa.com

values. This deconfliction is the source of a considerable amount of guesswork and speculation on the part of the analyst in the absence of professional intuition. One must consider what is lost by discarding those alternative values. Are there relationships between the conflicting datasets that have meaning? Is each dataset presenting a different and valid view of the entity or are the alternate values erroneous? If so, which values are erroneous? Is there a historical significance of the variances? The analysis of modern datasets requires the use of specialized algorithms and storage and retrieval mechanisms to identify, deconflict, and assimilate variances of attributes for each entity encountered. These variances, or versions of attribute values, contribute meaning to the evolution and analysis of the entity and its relationship to other entities. A new, distinct storage and retrieval mechanism will enable analysts to efficiently store, analyze, and retrieve the attribute versions without unnecessary complexity or additional alterations of the original or derived dataset schemas. This paper presents technologies and innovations that assist data analysts in discovering meaning within their data and preserving all of the original data for every entity in the RDBMS.

المنارة للاستشارات
www.manaraa.com

Chapter One – Introduction

Data analysis systems that rely on traditional structured data stores restrict the choice of values for the attributes of an entity or observation to a single value. Assimilating data from multiple data sources requires analysts and scientists to discard all alternative values in the data. Traditional systems, based on relational database stores, require restricting input data sets or portions of data, or even records or groupings from the data to form a single declarative dataset upon which to conduct their analysis. This restriction is especially difficult when data is gathered from more than one source. The combined data is often a tangle of conflicting observations and facts about the entities the data describes.[1] This progression proceeds to the choice of attribute values for a given set of attributes for a data relation. That is, analysts and scientists are forced to choose one value for each attribute for a given entity in a relation when there are collisions in the incoming data.

Analysts and scientists who are faced with this dilemma ponder questions such as, What is lost by discarding all other assimilated values for the sake of one value? Is there meaning in the relationships of the other values, both among them and to other data? What can be gained from studying the variances in the data? How has the data changed

[1] Some would say that the data contains entities. The author contends the data describes entities. This is a subtle and often overlooked discrepancy, which will become profound once the concept of data versioning is introduced.

المنارة للاستشارات
www.manaraa.com

over time? Analysis of the discarded values of the collisions is not possible without the ability to store multiple values of the data attributes.

The analysis of complex datasets requires the use of specialized algorithms and storage and retrieval mechanisms to identify, deconflict, and assimilate variances of attributes for each entity encountered. These variances, or versions of attribute values, contribute meaning to the evolution and analysis of the entity and its relationship to other entities. A new, distinct storage and retrieval mechanism will enable analysts to efficiently store, analyze, and retrieve the attribute versions without unnecessary complexity or changes to the original or derived dataset schemas.

1.1 Background

Not long ago, developers viewed the data that a system consumes and produces as simply input and output – largely an afterthought. Development practices have progressed beyond that limited viewpoint and now consider data as the central element in the development effort. Researchers in the field of software engineering, specifically software quality and usability, have gained many advances in technology and methodology with this philosophy. Unfortunately, when faced with systems that combine many disparate and duplicated data repositories, some developers take this mantra a bit too far.

Developers and major stakeholders at the start of a project, before analysis of the requirements has begun, often proclaim, "What we need is a large database that…" Sadly, these proclamations are often misguided and lead to systems that are cumbersome, with

المنارة للاستشارات
www.manaraa.com

lengthy development schedules and complex databases that few understand and fewer fully exploit. Nowhere is this mindset more prevalent than in the Department of Defense. Many of the projects involving systems that ingest data from multiple sources attempt to solve the data ingestion problem by building the solution around a poorly or inadequately designed database.

This practice invariably leads to stove piping – a process that provides a solution for a single analytical perspective while failing to create a solution that applies to the entire problem domain. It is difficult to build a new analysis area from a set of disparate data repositories. The developer would spend more time fitting the data together, resulting in yet another stovepipe.

The number and diversity of the incoming data in modern systems requires a new approach to dealing with data. Systems can no longer afford to be developed with this "Mother of all Databases" (MOADB)[2] philosophy. Unfortunately, the debate over the MOADB philosophy is often a classic Paul vs. Feyd[3] clash [Herb65] with the user of the system becoming the innocent bystander and the data (database) becomes the ultimate casualty. Fortunately, system architects are now considering data an actor[4] in the system and thus a key component of the design. Thus, the actor can interact with and be acted on by the system.

Analysts and scientists have at their disposal a seemingly endless supply of data and repositories from which to draw. New repositories become available daily making

[2] Pronounced "Maud'Dib." With apologies to Frank Herbert and Maud'Dibs everywhere.
[3] Again, apologies to Frank Herbert.
[4] Borrowing from the Unified Modeling Language entity of the same name.

المنارة للاستشارات
www.manaraa.com

the choice of which repositories to use more difficult. Furthermore, traditional data analysis has been conducted using structured data repositories such as relational database management systems (RDBMS). These systems, by their nature and design, prohibit duplication of indexed collections forcing analysts to choose one value for each of the available attributes for an item in the collection.

Today's global economy and global communication age, hastened by the explosive adoption of the Internet, has made a great deal of data available that would never have been obtainable in the past. Individuals, corporations, analysts, and scientists can now glean more and more data from a growing variety of sources. No longer is there a lengthy and expensive period during an analysis project where data is hunted down and gathered. The abundance of data available for some projects becomes the inverse problem of having too much data to consume in a reasonable timeframe.

Furthermore, as the last two decades have shown, individuals, corporations, analysts, and scientists have invested in highly sophisticated enterprise applications [ESRI00] and customized analysis products [Paul02]. These systems are often built to accommodate a structured data format in the form of a relational database. The real effort then becomes formatting the data into the correct layout in order to be consumed by the analysis application.

Often analysts discover two or more repositories with information about the same entity. When combining this data and transforming it into a form usable in an RDBMS, analysts are forced to deconflict the collisions and choose a single value for each duplicated attribute containing differing values. This can be complicated if the data does

المنارة للاستشارات
www.manaraa.com

not have common attributes, making matching data between data sets difficult. This process has become the source of a considerable amount of guesswork and speculation on the part of the analyst [Raso01].

Deconfliction, the assimilation and removal of conflicts, of data is expensive. It requires large amounts of time and resources to conduct thoroughly. Although some automation applications can be created to lesson the burden, it has become too costly to deconflict and reformat the data [Hern95].

Among the issues concerning deconfliction is the desire to assimilate the data in a manner that preserves the meaning of the data. If meaning is lost, the data no longer accurately describes the real world and thus can no longer be considered reliable.

What is lost by discarding those alternative values? Are there relationships between the conflicting repositories that have meaning? Is each repository presenting a different and valid view of the entity or are the alternate values erroneous? If so, which values are erroneous?

A solution is necessary to overcome these burdens and provide analysts and scientists with the ability to save all variants of the data – at the attribute level – to preserve the meaning of the data, both inferred and implied. This solution would allow the storage and retrieval of data through the intersection or exploitation of all of the variants, or versions, of the data. Thus, it would be possible to produce a dataset that is based on certain views of the data, be that based on the source, some quality (attribute) of the data, a historical or temporal perspective, and so on. The cost of performing data deconfliction could therefore be reduced to the time it takes to form queries of the data,

للاستشارات
www.manaraa.com

rather than a lengthy process of choosing subsets of the data to save and discarding the rest.

1.2 What is Data Versioning?

There are many views of what it means to store versions of data. Views ranging from analogies with 'variant' [Chatt04] to analogies with 'alternative' [Elma93] are presented and argued among the leaders in the fields of temporal databases [Gadi88] and document management [Varz98]. In fact, the word versioning is often misused and overvalued. This section presents the accepted meaning of the phrase 'data versioning' and establishes its scope within the confines of this work.

1.2.1 Definition of Versioning

The *Merriam-Webster Dictionary* defines the word *version* as:

Version

1: a translation from another language; especially: a translation of the Bible or a part of it

2: an account or description from a particular point of view especially as contrasted with another account

3: an adaptation of a literary work <the movie version of the novel>

4: an arrangement of a musical composition

5: a form or variant of a type or original <an experimental version of the plane>

المنارة للاستشارات
www.manaraa.com

This work considers the second and fifth forms of the definition with emphasis on the fifth form. That is, a version is a variant of a type or original data value. This variant is therefore no more or less important or valid than the original. Only the originator or viewer of the data can make the determination of which is lesser or greater in the context of the data and its use.

1.2.2 Versioning Explained

Versioning is the process of storing variants of data. Each variant of the data can have a number of additional properties that can be used to determine the source or even validity of the data. These additional properties form metadata about the variant itself. The combination of the variant value and its metadata become what this work refers to as a version of the attribute.

Attribute	Value	Source_Id	Reliability	Confidence
Height	100	98	Low	Low
Height	105	99	High	Medium

Table 1-1: An Example Set of Versions

Table 1-1 depicts an example of this philosophy at work. In this example, there are two values for the height attribute. It is assumed this data describes the same entity. Thus, the possible values for the height attribute are 100 and 105. If the source identified by the sentinel key of 98 is chosen or required to be the default value, the value of the height attribute for the referenced entity could be set to 100. Notice that the source identified by the sentinel key of 99 is of higher reliability and confidence. Thus, with some simple queries, one could determine that a more accurate height of the entity is 105.

المنارة للاستشارات
www.manaraa.com

Reliability is the qualitative assessment of accuracy. Confidence is the qualitative assessment of validity.

However, there may be additional restrictions that would not permit the use of one or more of the sources – be that for legislative, declarative or other impenetrable forces of nature[5]. One could still conduct analysis first without using the prohibited sources and then again with the prohibited sources and compare the results. The result could then be used to measure the validity or accuracy of the analysis.

This scenario has many real-world applications such as the predictive nature of financial futures. If a brokerage had the ability to substitute all types of data it acquires, be they speculative or historical, the financial analysts could plan their investments much farther into the future and be able to adapt accordingly as the data changes. They could also use the ability to track changes over time and analyze the effects of unobvious relationships among the data.

This concept, albeit simple and without mystery, is fundamental in understanding the power of a system that can store and process versions of data. This concept can give analysts and scientists the ability to determine the reliability, confidence and source of the information of an assimilated dataset in a timely manner. Furthermore, this concept can also provide the ability to manipulate that data to form alternative views of the data by limiting the source of the data. It would then be possible for corporations and organizations to present unique views of the data based on the recipient of the data or

[5] Such as analyst preference, scientific, or philosophical bias.

المنارة للاستشارات
www.manaraa.com

product as well as adapt to a changing world without having to lose the historical relationships among the data.

1.2.3 Ship of Theseus – Version ad Infinitum

Versioning the data will quickly become a question of originality. The concept of originality has been debated by philosophers for centuries. One such example is the philosophical concept of the Ship of Theseus [Cohe04].

Theseus' ship was preserved by the Athenians during the lifetime of Demetrius Phalerius, a student of Aristotle, ca. 350-280 BC. Over the years, as each plank of the ship deteriorated,[6] the Athenian historians replaced the plank with a new one. Over time, it became apparent that all of the planks and bits of wood had been replaced. The question raised was, "Is this the ship of Theseus or just a very good copy?"

Another version of this puzzle is presented where Theseus sails from Athens towing a second ship full of lumber. During a prolonged voyage and many hardships, the crew was forced to replace every part of the ship. The questions this puzzle raises are, "How many ships did Theseus depart with?" and "How many ships did he arrive with?"

Yet another version of the puzzle is presented where Theseus sails from Athens with a cargo of spare parts[7] towing behind a smaller vessel containing a crack crew of shipbuilders. As the parts on the original ship were replaced, the old parts were thrown overboard; the shipbuilders trailing behind salvaged the parts and reassembled them. This

---

[6] Presumably from the growing pollution caused by the chariot expressway and tourism.
[7] Figuratively speaking. The concept of uniform manufacture was not invented until many centuries later.

المنارة للاستشارات
www.manaraa.com

puzzle raises a number of questions. Is the ship that Theseus arrives in the same as that he sailed from Athens? Which ship is the original and which is the copy?

These puzzles apply to the data-versioning concept. How many versions of the data need be replaced before the data is no longer the original data? How do we determine equivalence (in meaning) of a version to the original data? If the version is equivalent to the original is it a version or a copy? If we can not say the version is equivalent, what does that say about the version – is it a new entity or truly an alternative form of the same entity? Quantitatively, the question of how many versions can be answered simply by counting the combination of the available attribute versions and the original attributes. Once all of the attributes have been "replaced," the data is no longer the original data.

Furthermore it can be argued that the combination of versions generate uniquely original entities. That is, no matter how many attributes are replaced, the resulting entity is a unique instance and thus not the original at all. For example, consider a case where the attributes of an entity change over time. With each observation stored, the resulting change is a new state for the original entity and therefore can be considered a new instance of the same entity. Conversely, it can be argued that the data from which the versions were drawn have equal value to that of the original values and that as long as the relationship between the entity and its versions are maintained, the data is still original no matter how selective the versions, which is precisely the premise on which this work is based. It is a matter of evaluating the new attributes in a qualitative manner – the

المنارة للاستشارات

www.manaraa.com

interpretation of the applied attribute versions can mean to some a new entity and to others the same entity.

Some have argued that as long as the keel of the ship remains intact, the two ships are the same and that there has been no transference of originality. The same can be said about data versioning. At what point does the data lose its originality? Can one create so many versions of the data that the data has lost its meaning? Is it possible that storing every alternative value encountered becomes akin to never throwing away anything because it might be useful one day?[8]

Whereas this conundrum can be argued for many sessions and from many views, this work attempts to eliminate what can be called "The Theseus Factor" – the versioning of the data to the point of duplication – by permitting the storage of the alternate value, metadata, and its relationship to the entity with each version. The relationship of the version and its metadata to the original entity identifies the firmament of the data and therefore becomes the "keel" of the data. As long as the keel is never replaced, the data (ship) remains the original no matter how many attribute values (parts) are replaced.

1.3 What is Attribute-Level Versioning?

The motivation for this research is a knowledge engineering project designed to combine multiple datasets about a subject area. This solution must be capable of storing known associations and relationships among the data, resolving collisions, storing

[8] Garages and basements everywhere are piled high with the debris of life by pe ople who never throw anything away. Ponderously, most seem to know where everything is no matter how insignificant or small the item.

www.manaraa.com

collisions among the data attributes as alternative values, analyzing the alternative values, and forming a hypothesis from the analysis. The output of this process will be a dataset that is the result of a query from the combined data and the versioned attributes. This project and the underlying technologies have been named Attribute-Level Versioning (ALV)[9].

ALV provides a mechanism to store and retrieve all values for an assimilated attribute. Having a comprehensive version repository with efficient retrieval mechanisms permits analysts and scientists to perform additional analysis using advanced statistical and knowledge inference models that identify meaning or infer relationships among the data. Adding this new dimension of analysis opens a new chapter in the data-mining paradigm.

1.3.1 Definitions

Some new concepts have been defined in this work. Although most are self explanatory, some can be confusing when seen or read among the mass of experimental results and theoretical explanations. The following paragraphs define the more commonly used terms. The list begins with a clarification of an existing database term.

An ***attribute*** is a property that describes an entity either in part or in whole. An entity can have any number of attributes [Date04, Elma03].

An ***attribute value*** is the value of an attribute given by its domain.

[9] The initial concept of ALV was originated in November 2002 by Mr. M. Facemire. His vision of a better data versioning me chanism survived the test of time and persevered in the face of apathy. This work is the fruition of his vision and the imagination of the author.

المنارة للاستشارات
www.manaraa.com

Attribute value metadata is the additional set of properties that describe, define or form the reference for an attribute value. Each attribute can have its own set of metadata defined.

An ***attribute version*** is unique set of an attribute value, the attribute value metadata and the relationship of the attribute value to the entity.

A ***versioned table***[10] is the relation that is being versioned. This is the logical representation of the data as presented by the database.

An ***attribute version table*** is the data store that contains all of the attribute versions that relate to entities in the versioned table. This becomes the version store for the versioning system (ALV). This is the logical representation of the attribute versions as presented by the ALV system.

A ***version store*** is the physical store for attribute versions organized into a single file and organized using a clustered mechanism. This is the physical storage mechanism for an attribute version table.

1.3.2 ALV Requirements

The ALV project was born from an idea that analysts need to track the source and lifetime of data. Tracking the source of information is a relatively intuitive approach that can be easily incorporated in a relational database management system (RDBMS) as one or more relations with referential integrity between the relation that describes the entity

[10] In many instances, the word "table" is synonymous with 'relation." Thus, a versioned relation can be referred to as the versioned table and vice-versa.

المنارة للاستشارات
www.manaraa.com

and the relation(s) that define the source information. A known system has been created to accomplish this for traditional RDBMS implementations[11].

There is now a solution that can provide the capability to track data over its lifetime (see Chapter 2 for a review of existing technologies) using a variety of attribute value metadata. The ALV system is capable of meeting this need and more. The vision for the ALV system is:

- Combine disparate data sources with observations on that data that contain one or more variances of the values of the attributes within the data.
- Create sound theoretical methodologies and architectures to manipulate the combined data.
- Form patterns from the data based on links among entities found in the data.
- Identify gaps in the data and extrapolate the missing data from patterns of observations.
- Form unique views of the data based on an analytical case study.
- Provide a common process for version analysis
- Leverage advanced modeling to “Know what we don’t know”
- Correlate data based on source
- Corroborate observations
- Track source, classification, and confidence at the attribute-level
- Version the data by attribute

[11] This system is protected by intellectual property laws. Additional details cannot be disclosed.

المنارة للاستشارات
www.manaraa.com

- Permit a single attribute to have many alternative values (versions) from one or more sources
- Automate identification of duplicate objects
- Merge duplicate objects
- Track relationships between suspected duplicate objects
- Automate identification of data gaps and suspect attributes

There are a number of unique technology achievements in the ALV system.

- Advanced storage and retrieval mechanisms
- Advanced query mechanisms through the extension of Structured Query Language (SQL) for version extraction
- Advanced index mechanism for fast version retrieval
- Data mining algorithm applications for version analysis

A complete and thorough description of these technologies and their importance is contained in Chapter 3.

1.3.3 Goals

To be considered a success, this project must be capable of integrating with traditional database management systems (DBMS). The goal is to leverage a versioning technology from within the DBMS. The technology should become an extension of the

المنارة للاستشارات
www.manaraa.com

DBMS rather than an add-on module or procedural access mechanism. An extension to existing technology will allow legacy applications to leverage versioning without having to be redesigned or altered. More importantly, this goal will ensure that the output of this project can be consumed by legacy applications that rely on database storage.

Therefore, this project focuses on implementing the solution as an extension to the MySQL DBMS which requires adapting the MySQL engine, extending SQL, and providing additional tools to manipulate the data prior to and after storage.

1.4 Can Technology Solve the Problem?

The problem presented in this chapter is one that arises in many application domans. The ability to save all pertinent data, whether in the form of archival, historical, temporal or even analytical variance, will give data analysts the ability to formulate answers to questions such as:

- What did the data look like before an event?
- How has data changed since an event?
- Where did the data come from?
- What correlations exist within the data?
- How would changing the values of certain attributes affect the outcome of the experiment?

المنارة للاستشارات
www.manaraa.com

This dissertation presents four contributions to the fields of database theory, knowledge management, and engineering; 1) a physical store for attribute versions, 2) an indexing mechanism for versioned data, 3) extensions to SQL and query optimizer, and 4) application of data mining algorithms to classify the version data. These contributions together with the successful implementation of the ALV system will demonstrate that ALV is a sound methodology for tracking and maintaining data from many sources. This work also presents the applicability of the ALV system to enhance temporal analysis, historical archiving, and mining of data stores. However, it is the summation of this work in the form of the ALV system that presents the greatest contribution. This system will enable analysts and scientist to study data from a new and unique perspective.

1.5 Dissertation Overview

This dissertation examines the project, solutions, and the philosophy of their creation. Chapter 2 presents the result of a literature review. Chapter 3 begins with a detailed discussion of the supporting project. Subsequent chapters examine each of the new technologies. Topic chapters introduce the area or subdiscipline and discuss the findings and arguments encountered during the research, also giving the reader enough background information to understand the points being argued. For example, a discussion of the social issues regarding data-mining will be limited to presenting the body of knowledge for that topic as it applies to the argument at hand rather than introducing the common issues and points of view regarding the social impact of data-mining. The final chapter will review the research findings and present alternative views of the results of

المنارة للاستشارات

www.manaraa.com

experiments along with areas for future research. The chapter layout is as follows:

- Chapter 2 – Survey of Existing Solutions, Technologies, and Theory
- Chapter 3 – An Introduction to Attribute-Level Versioning Technologies
- Chapters 4-7 – Discuss each technology in detail with experiments and results with emphasis on:
    - Thesis statement for creation of the technology
    - Comparison to existing technologies and theories
    - Conclusion: Contribution to ALV
- Chapter 8 – Summarize use of new technologies and theories in the ALV solution applications and present areas for future work.

المنارة للاستشارات
www.manaraa.com

Chapter Two – Existing Solutions, Technologies and Theories

This chapter encompasses an in-depth search within the academic and commercial venues for versioning systems, technologies, theory, and practice related to the ALV technologies. Presented is a survey of all applicable areas including, document management, relational database, temporal database, object relational database, and object-oriented database systems. Sections 2.1 – 2.5 describe each of these areas, presenting the viewpoints of the researchers in the area and an overview of the technology represented. Each section concludes with a comparison of how the area compares to this dissertation, the ALV project, and its technologies. Section 2.6 describes a collection of related topics and considerations discovered during the research. Section 2.7 concludes with a presentation of the application of this research to the dissertation work and related project.

2.1 Document Management Systems

Document management systems (DMS) are systems designed around a persistent store of artifacts (documents and document fragments). These documents are often stored in smaller blocks of data that are then later combined to form a specific view of the document. This technique of storing portions of the document in order to more fully manage change and traceability is called "virtual documents" in light of the fact that the

المنارة للاستشارات

www.manaraa.com

document itself is simply a view (filter) of the data (documents) stored. Technology research has focused on understanding the system-level requirements of versioning. In particular, the questions of how to support alternative versions, configurations, long transactions and control access to versions have received substantial attention [Lu96].

A DMS is an implementation of an information retrieval system. An information retrieval system is one that is designed to represent, store, organize, and access data. The objective is to retrieve the information that best fits the user's need. This is typically accomplished using advanced query mechanisms. Information retrieval systems are therefore designed to store all forms of information – data that has meaning to the user. DMS do not have all of the features normally found in information retrieval systems because DMS focus on the management of the document in its original form and changes that occur to the document. In a DMS, it is the document that is the item operated on. In information retrieval systems, it is the data (information) that is acted on [Rals03].

Documents are data that evolve over time. One possible source of evolution comes from the modifications made to the content of a document by its author(s). These modifications are typically small changes to only a portion of the document content. The result is a historical reference as to the state of the document before the change and the state of the document after the change – an application of versioning. This reality is especially relevant in environments where frequent modifications occur such as collaborative authoring environments (each author can propose a series of modifications to the global work). Among the different possibilities of applying these modifications to a specific document, it is frequent that authors describe these modifications as a new

المنارة للاستشارات
www.manaraa.com

document or residing inside another document. For each modification, the author cites the document fragment he/she wants to change and indicates how the selected fragment could be modified (eliminating it, substituting it, etc.). The new version obtained by the application of these changes is a virtual one, in the sense that the users know it exists, but there is no physical copy of it available. The collective conscience of authors, the original version, the modified version, and the document containing the modification (which is a separate document with its own identity) coexist. This leads to a problem that can be stated as follows: "Given an abstract document A and the collection of versions of ΔA (also abstract entities), is it possible to apply any number of ΔA's[1] to a document version to produce yet another version of the document in this library? [Mart02]"

2.1.1 Technologies

The technology used in DMS is often based on existing storage mechanisms and databases such as Microsoft SQL Server[2], Oracle, and other commercial database systems. Some DMS use proprietary storage mechanisms in order to leverage unique features or to ensure exclusivity. Most of the current research in DMS technologies is concerned with text exploitation, indexing, and document translation and transformation. These technologies include XML, XQuery, XPath, XLink, and other newer XML-based document handling mechanisms [Anto04]. Many consider documents, or text in general, as unstructured and therefore difficult to incorporate in technologies such as semantic

---

[1] ΔA represents a modification that is in the historical chain of the given document. Modifications are linked to their original source. One would not arbitrarily apply a modification of any document to any other.

[2] All product names referenced in this document are trademarks of their respective companies.

www.manaraa.com

web [Fens03]. Fortunately, a recent effort is underway to incorporate document versioning in the semantic web[3].

The work that is most intriguing and most related to this work is the management of the documents over time. For example, it is of interest to know how the documents are altered from one version to the next. Most of the effort related to document versioning is concerned with knowing about the fact that two electronic documents are versions of the same abstract document. There are four techniques of resolving this issue.

The first involves saving the original document and the revisions as separate files while maintaining links, called revision links, among the related versions [Mart02]. Implementation of this technique involves creating two databases that are updated simultaneously: the document database and the link database. The main problems with this approach are keeping the revision links database up-to-date and conflict resolution among multiple authors.

The second involves using different timestamps of the document fragments associated with a record in a database comparing them in order to detect changes that reflect the fact that an object has been versioned [Sche03]. This technique can be used with object databases and therefore can be considered when modeling documents as objects. This solves the problem of two different database stores, but makes the version detection and generation more complex.

The third technique, which has gained considerable attention, is to store the data hierarchically (in much the same way as structured XML documents). This is much easier

[3] This system is protected by intellectual property laws. Ad ditional details cannot be disclosed.

المنارة للاستشارات
www.manaraa.com

to accomplish if the documents are stored in XML format. In this case, the documents are broken down, or shredded [Date04], into the unique individual XML elements (e.g., attributes). These elements are then stored as discrete data items in a tree-based repository. Each level of the tree represents a specific version of the document. Versions are therefore represented as nodes in the tree. Each node represents changes and has associated with it metadata tags that facilitate queries about version history or change logs. The detection of versions and version resolution is accomplished using tree comparisons [Mart02].

The fourth and most common method of document versioning is called a version control system. These systems maintain a database that stores the selected versions. They use query interfaces designed around workflow applications that retrieve the data from the database [Mart02]. Most version control systems store only the main, or initial, document. Changes are stored as deltas of the original document. The applications of deltas to the original document form a version of the document and provide a definitive version history. Some version control systems store each successive revision as a separate file, thereby allowing for faster retrieval at the expense of storage space [Tuck04]. This latter method is the easiest to implement as you simply allow the operating system facilities to manage the document save and retrieval. Additional storage and retrieval mechanisms are created to save only the metadata concerning the revision creation and modification. This additional data is often saved in a relational database or other similar storage repository.

المنارة للاستشارات
www.manaraa.com

Another variant of this method is called snapshot versioning where copies of part or all of the file system are archived [Soul03]. Each snapshot of the file system represents a static copy of the file system, as it existed at the time the snapshot was taken. This allows versioning systems to record versions in a real time environment with little or no user initiation. Recovery of previous versions is problematic, as it requires restoring the snapshot – which contains the entire state of the file system. Restoring a single file to a specific revision becomes impractical. The snapshot version method is best used for real time systems or systems that must track large numbers of changes to many files (sets of files, large files, etc.).

There is currently additional research being conducted in the application of temporal theory to document management systems. One such work incorporates a temporal element for storing XML documents [Gran03]. This work encompasses research concerning the temporal management of normative texts in XML format. In particular, four temporal dimensions (publication, validity, efficacy, and transaction times) are used to correctly represent the evolution of norms in time and their resulting versioning. This technology introduces a multiversion[4] data model based on XML schema and defines basic mechanisms for the management of normative texts.

2.1.2 Applications

The primary implementation of document management systems is in the form of Computer Aided Software Engineering (CASE) tools, document control systems,

[4] Not to be confused with multiversioning, which is a form of version and update conflict resolution for distributed databases.

www.manaraa.com

configuration management systems[5], and document collaboration (sharing) systems. There are many document management systems available both commercially and academically. Examples of these systems include Microsoft SharePoint Portal Server [Micr03], IBM Rational Unified Toolset [IBM05], DOORS [Tele05], CORE [Vite05], and Interchange^{SE} [Trid04].

Perhaps the most interesting of these systems is Trident System's Interchange^{SE} [Trid04]. Interchange^{SE} is an object-oriented information repository designed especially for engineering projects. It contains a central repository that stores all of the artifacts for an engineering project and provides configuration management tools for managing changes to those artifacts. The central repository provides features that provide access control at the property (attribute, relationship, and method) level. The technology employed to manage change (versioning) is called object versioning. Object versioning provides three operations; 1) object copy – permits the copy of any attribute or artifact which may be altered to present a new object or unique view of an existing object, 2) object clone – permits a duplicate object to link to an existing object, thereby creating another instance of the original object, and 3) object perspective – permits the presentation of the objects based on the filters applied to the objects in the data store (generates views) [Trid04].

The object versioning capabilities of Interchange^{SE} are limited to the proprietary data repository that is at the core of the system. Trident Systems has no plans to release the technology or share its capabilities via subcomponent development. Interchange^{SE} is

[5] Although commonly considered as a type of CASE tool, there are configuration management systems that have little or nothing to do with software development or engineering.

www.manaraa.com

therefore an example of an industry isotropic application that has little or no academic merit.

2.1.3 Comparison with ALV

Although DMS have a concept of versioning, it isn't the same as that of ALV. Versioning in a DMS is simply the tracking of changes to a document element. Tracking the changes over time and the application of those changes using views (filters) is what produces the version of the virtual document. ALV is designed to track changes of any sort and is fully customizable in what data is stored along with the version. This additional data, or metadata, permits the storage of pertinent data about the version. This data can include traditional temporal or delta information or it could be used to store information about the origin and application of the version data itself. ALV therefore is more flexible than the technology employed in DMS.

One form of DMS uses a collision resolution technique that tags the data with a timestamp, resulting in a priority queue for either earliest or latest strategy for conflict resolution [Trid04]. ALV is more expressive in that the goal is to store all data regardless of temporal aspects. Thus collisions are simply choices of versioned data rather than duplicate or repeated values. Pure duplicates are supported, but practice is likely to exercise a more normal uniqueness constraint.

ALV is not designed to be used with large blocks of data as are stored in a DMS. However, ALV can be used to track changes of data over time. Date metadata tags can be assigned to each attribute value and thus permit the inclusion or exclusion of the attribute

المنارة للاستشارات
www.manaraa.com

value in a data query based on the date stored. As mentioned previously, this is an example of how the temporal database theory can be applied to ALV.

Finally, despite the claims of Trident Technology Solutions to have true attribute-level versioning (object versioning) [Trid04], their Interchange^{SE} product does not, in fact, permit attribute-level versioning as defined in this work.

Although most DMS are not viable solutions for meeting the needs of this work, the research presented has shown one significant feature of DMS that has bearing. Specifically, the concept of treating the deltas, or changes, of documents as components of the original document is analogous to having complex data types as attributes. In fact, this concept is prevalent in the ALV system – all attribute values are connected to the original record and become meaningless if separated from the collection of versions or disassociated with the original entity.

2.2 Relational Database Systems

A relational database system (RDBS) is a data storage and retrieval service based on the Relational Model of Data as proposed by E. F. Codd in 1970. These systems are the standard storage mechanism for structured data. A great deal of research is devoted to refining the essential model as proposed by Codd as discussed by Date in *The Database Relational Model: A Retrospective Review and Analysis* [Date01]. This evolution of theory and practice is best documented in *The Third Manifesto* [Date00].

المنارة للاستشارات

www.manaraa.com

The relational model is an intuitive concept of a storage repository (database) that can be easily queried, using a structured query language (SQL) that resembles[6] natural language to retrieve, update, and insert data. The relational model has been implemented by many vendors because it has a sound systematic theory, a firm mathematical foundation, and a very simple structure.

The data is represented as related pieces of information (attributes) about a certain entity. The set of values for the attribute is formed as a tuple (sometimes called a record[7]). Tuples are then stored in tables containing tuples that have the same set of attributes. Tables can then be related to other tables (hence the "relational" in relational theory) through constraints on domains, keys, and tuples [Date04, Elma03, Rals03, Rama03, Silb96, Tuck04].

The query language of choice for most implementations is structured query language (SQL). SQL was proposed as a standard in the 1980s and is currently an industry standard. Unfortunately, many seem to believe SQL is based on relational theory and therefore is a sound theoretical concept. This misconception is perhaps fueled by a phenomenon brought on by industry. Almost all relation database management systems (RDBMS) implement some form of SQL. This popularity has mistakenly overlooked the many sins of SQL including the following:

- SQL does not support domains as described by the relational model.

[6] The degree of resemblance is often in the eye of the beholder.
[7] Many mistakenly consider a record as a colloquialism for tuple. The distinction is that a tuple is a set of ordered elements whereas a record is a collection of related items without a sense of order. Interestingly, in SQL a result from a query can be a record whereas in relational theory each result is a tuple. Many texts use these terms interchangeably, cre ating a source of confusion for many.

المنارة للاستشارات
www.manaraa.com

- In SQL, tables can have duplicate rows.
- Results (tables) can contain unnamed columns and duplicate columns.
- The implementation of nulls (missing values) by host database systems has been shown to be inconsistent and incomplete. Thus, many incorrectly associate the mishandling of nulls with SQL when, in fact, SQL merely returns the results as presented by the database system[8].

2.2.1 Technologies

The technologies used in RDBS are many and varied. Some systems are designed to optimize some portion of the relational model or some application of the model to data. Versioning is possible with RDBS, but the mechanisms are manifestations of the logical design of the data model (schema) rather than embedded as a technology. There are two basic mechanisms for representing version data in relational database systems, horizontal and vertical.

Horizontal refers to the traditional mechanism of normalizing the data into tables and relationships. Versioning can be achieved through parent/child relationships among the tables. The concept of differing metadata per attribute can also be achieved in a similar manner, but at the expense of time consuming queries. A fully optimized horizontal application does not perform well enough to meet the high demand of applications that require large, complex data feeds.

[8] Some of the ways database systems handle nulls range from the absurd to the unintuitive.

المنارة للاستشارات

www.manaraa.com

Vertical refers to an implementation by which the data is abstracted and stored as proxies. That is, the database is implemented such that each table contains part of the information necessary to determine the type and value of an attribute. Thus, all entries in the tables for attributes “refer” to enumerated types and even enumerated values. Proxies are therefore similar to how object-relational database systems store data. Versioning is possible due to the ability to store duplicates in the proxy tables. Uniqueness is compromised in this mechanism and mitigated solely through the use of surrogate keys[9]. Due to the fragmented nature of the tables, deeply nested queries are necessary to successfully retrieve an entity from the database. Thus, vertical mechanisms are even more prone to performance issues than horizontal mechanisms.

A similar concept to versioning available in RDBS is database archiving. Owners of large systems often rely on complex relational databases as the foundation of their business data. Due to the nature of the business operations and decision making, the databases are often allowed to grow without bounds. This creates a performance problem where the queries necessary to retrieve data force the system to issue many sub-queries and joins among the proxy tables. As more data is added, the queries take longer and longer to complete. It has been shown that performance degrades rapidly as new data is added. Ironically, most of this data is stored in production databases but rarely accessed. Database archiving allows for removing this rarely accessed data and storing it on a variety of storage media while providing easy access. This poses intriguing problems when data that has been archived is reintroduced into the database – effectively creating

[9] Often exploited to force data models to conform to relational standards.

المنارة للاستشارات

www.manaraa.com

two copies of the data. This duplication is often intentional on the part of the database administrators (at the behest of the analysts) and can be used to maintain versions of data [Lee04].

Another technology employed in RDBS is called multiversioning. Multiversioning is a collision detection and resolution mechanism for distributed relational databases. There are several forms of multiversioning, but the essential implementation uses timestamps as a means to detect a version collision – the arrival of conflicting or duplicate operations on the same data. Multiversioning can be extended as a mechanism for version detection using ALV [Hada96, Lome90]

2.2.2 Applications

The application of RDBS is manifold with a plethora of implementations too numerous to entertain a notion of listing individually. Some of the more popular commercial relational database management systems (RDBMS) include Microsoft SQL Server [Micr00], Oracle [Orac05], and MySQL [MySq05]. These systems are designed with the relational model as the core architecture goal, but as is the case in nearly all implementations, fail to completely meet that goal. Some, such as Oracle, extend the implementation to include technologies that are well beyond the scope of the relational model[10]. Specific detriments to these systems are the adoption of SQL and the treatment of nulls. SQL is not a true representation of the relational model and thus any implementation that limits its query expressiveness to that of SQL is implementing a

[10] Some would consider this innovation.

المنارة للاستشارات
www.manaraa.com

deviant of the relational model. Fortunately, most of the successful systems overcome these limitations with optimized architectures that perform and scale well to a wide variety of data[11].

2.2.3 Comparison with ALV

Relational database systems are the target platform for ALV. Relational theory is sound and well practiced. Although versioning is a relatively new concept brought on by object-oriented techniques, little in the way of back fitting versioning to relational systems has been considered. This work is an effort to fill that gap and provide relational database designers the ability to store all data pertinent to an analysis without loss or compromise.

However, it has been suggested that ALV violates Codd's original data integrity constraint that each tuple in a relation is permitted one and only one value for each of its attributes (the sanctity of first normal form). How can a technology such as ALV dare to violate this premise? ALV is designed to augment RDBS with extensions of the database server capabilities. It permits database designers to maintain Codd's premise in the original table structure. The premise is never violated per se; rather ALV provides a mechanism whereby versioned data can be substituted with that of the original data in the original structure without modification. Furthermore, the version store for ALV is designed to contain unique entries for each attribute version thus preserving Codd's

[11] A positive example of industry isotrophism versus academic rigor.

www.manaraa.com

original premise and extending it to versioning. This is perhaps the greatest advantage of the ALV technology.

2.3 Temporal Database Systems

Temporal database systems (TDBS) are based on relational database theory and are often implemented on relational database systems as extensions or services. TDBS are designed to incorporate time as a storage and retrieval mechanism. Time is considered not only a data element but also an operation that can be used to answer queries. Thus temporal databases store information about the entities (objects) as a parameter of the time dimension. Query mechanisms are therefore optimized for exploitation of the temporal states of the data [Lu96]. “The goal of temporal databases is to uniformly integrate past, current and future information in a unified system,” [Elma93]. Early work in temporal databases demonstrates a marking concept that marks tuples with time reference points rather than creating dedicated (and complicated) internal relationships expressed in time [Gadi88].

Tansel in his work on temporal database theory stated, “Conventional databases were designed to capture the most recent data, that is, current data. As new values become available through updates, the existing data values are removed from the database. Such databases capture a snapshot of reality. Although conventional databases serve some applications well, they are insufficient for those in which past and/or future data are also required. What is needed is a database that fully supports the storage and querying of information that varies over time,” [Tans93].

المنارة للاستشارات
www.manaraa.com

RDBS store information about the real world they attempt to represent. However, any useful representation of the real world needs to address the issue of the temporal nature of information, since the real world is very dynamic. In the relational model, the temporal nature of data has been largely ignored, being reflected only through updates while ignoring the past states. Early TDBS attempted to address this deficiency in an ad hoc fashion, primarily through applications that ran on top of the RDBS. This prohibits a high level of independence between the data and the application programs. TDBS are an attempt to integrate time as an intrinsic part of the model.

The difference between relational databases and temporal databases is that temporal databases incorporate one or more attributes into the structure of all objects (database, table, tuple). Thus, most temporal database systems are relational systems with time as a key element. Conventional or static [Dean, 1989] databases reflect the most current state of the domain of interest. When they are updated; existing data is discarded and the new data inserted. Temporal databases reflect current state and state history in applications where no data is discarded and replacements are characterized by an element of time. A data model represents the semantics necessary to support a specific application's purpose. If that purpose includes time, then the temporal nature of the data must be represented in the data model. For example, an event occurs at a point in time and is recognized by the fact that it changes the state of a thing in the application domain, resulting in a state history [Gora95].

There has been a considerable amount of work in the area of temporal databases. Most of the research efforts have been directed towards extending the relational model to

المنارة للاستشارات

www.manaraa.com

incorporate time. Two approaches have been proposed in the literature for temporally extending the relational algebra: tuple time stamping, and attribute time stamping. Tuple time stamping uses a timestamp as a special attribute of the relation scheme and hence is part of every tuple. An initial implementation of this was first proposed in LEGOL 2.0 [Dey96], which uses two implicit time attributes, start and stop. Attribute time stamping is essentially the same concept only applied to each attribute where appropriate[12] [Dey96].

A very useful paper by Jensen et. al. [Jens92] presents a proposal for consensus of temporal database terminology. Since no rebuttal has been published, much of the content of this article is now considered standard practice. Unfortunately, the same cannot be said about data versioning or versioning in general.

2.3.1 Technologies

Major DBMS tools are incorporating facilities for temporal data management (e.g., Oracle's Spatial Cartridge and Informix's Datablades). There are two basic technologies or theories of time used in these systems. The first can be best described as enabling a “point in time” relationship among the data (timestamps) [Jens92]. The second can be best described as enabling a “change over time” relationship among the data (time sequences) [Jens92]. These two technologies can be combined to form a hybrid that can relate the data to time using timestamps, time sequences, or both (bi-temporal) [Jens92].

[12] One would have to have a very good reason to store temporal changes to primary key values.

المنارة للاستشارات
www.manaraa.com

Because the concept of versioning was not included in Codd's model, some researchers have investigated how versioning could be implemented. The research to date has been in the area of temporal databases. There are three general approaches: table-level versioning, where new snapshots are created when any attribute change occurs; tuple-level versioning, where an entity's history is maintained and monitored through changes to individual records in the attribute table; and attribute-level versioning, where variable length fields hold lists of time-stamped attribute versions. Storing lists of time stamped attribute versions demands alternative and complex algebra for fast retrieval [Blak94].

Likewise, the concept of storage of temporal data involves a complex mechanism that must permit fast retrieval. Each attribute of each entity that has a temporal element has a value that is determined by a sequence of events. Thus, all attribute values are time-dependent. In an application where values change frequently, this could lead to the need to store many time-dependent values. This concept requires modification to the database system to optimize storage and retrieval primarily at the physical level, but will also require small modifications at the logical level.

Numerous modifications and structural extensions to the relational model and relational databases have been posed to accommodate large amounts of temporal data. Research topics include temporal functional dependency [Rodd02], temporal query language extensions [Elma03], changes to the relational model and relational algebra [Dey96], and specialized indexing mechanisms to support fast temporal queries [Elma03].

المنارة للاستشارات
www.manaraa.com

Most of the research on temporal databases has concentrated on extending the relational model to store and retrieve time in an appropriate manner. These extensions can be grouped into two main categories. The first approach uses First Normal Form (lNF) relations in which special time attributes are added to a relation and the history of an object (attribute) is modeled by several lNF tuples. 1NF is typically implemented using timestamping. The current temporal techniques all comply with first normal form (1NF) – each tuple contains only atomic attributes. These approaches are collectively labeled tuple-versioning approaches to temporal design.

The second approach uses Non-First Normal Form (N 1NF) relations in which time is attached to attribute values of a relation and the history of an object (attribute) is modeled by relaxing the 1NF constraint, thereby allowing multi-valued attributes [Gadi88]. These approaches have been referred to by several researchers as attribute-versioning approaches to temporal design [Hada02, Piss94]. Table 2-1 shows the employee table expressed using this multi-valued attribute approach (assuming that only salary and department have temporal concern):

Emp#	SSN	LName FName	Salary	Department
3025	086630763	Lyons James	{<15K (1/8/95,1/15/98]>, <25K (1/8/95, now]>}	{<dept1 (1/8/95,11/5/98]>,<dept2 (11/5/98, now,]>}
3089	579659458	Gordon Walden	{<25K (2/26/95, now]>}	{< dept1 (2/26/95, now]>}
3092	129548660	Charles Davis	{<18K (2/28/95, now]>}	{<dept2 (2/28/95,8/22/98]>, <dept3 (8/22/98, now,]>}
3105	454625914	Eric Wood	{<23K (11/2/95, now]>}	{< dept1 (11/2/95, now]>}

Table 2-1: A multi-valued attribute temporal "employee" relation

Because this approach allows multiple values for each temporal attribute, it allows the database to maintain the desirable quality of having a single tuple represent a single

المنارة للاستشارات
www.manaraa.com

entity in the real world. However, the use of multi-valued attributes is problematic under current relational database management standards [Alle00]. The problem is that each of the multi-valued attributes are presented as having separate parts and not as a whole. This eliminates any control and predictability over the meaning of the attributes. Attribute values are only meaningful when considered as a whole. An individual member of the list of attribute values has no meaning outside the list. By managing the multiple values (versions) as a whole object, it would be possible to retain the tenets of the relational model, thus preserving the meaning of the attribute-versions.

Studies have verified that the major performance trade off between different types of TDBS is between the restructuring (unpack) operation needed in temporal databases using attribute timestamping and the join operation needed in temporal databases using tuple timestamping. Furthermore, the experiments show that keeping all temporal tuples in one single relation does not prove to be an effective alternative for temporal databases which use tuple timestamping [Gora95]. Although logically the concept is sound, in practice the extra space needed to pack the information into the table forces the data to become fragmented. Even if measures are taken to treat the fragmentation problem, the data retrieved for each entity in the table grows with each timestamp value added to the entity. Thus, this solution is not scalable to larger data sets or data sets with a high degree of time variability in the data.

Lastly, some researchers have a view that it is that database designers should determine the most appropriate data structures for an application without taking into account which information items have a temporal element. That is, temporal elements

المنارة للاستشارات
www.manaraa.com

should be permitted to be stored outside the confines of the database design (schema). This approach makes easier the addition of space or time features to legacy databases that usually do not contain explicit temporal specifications [Pare99]. This philosophy is the same for that of ALV – to extend RDBS without the need to alter the schema of the original data. This extension will occur primarily at the physical level – the creation of a physical store for the version data.

2.3.2 Applications

Numerous authors recognize the importance of representing the temporal aspects of data. Analysts must frequently be able to retrieve not only the most current value of an attribute, but also its entire history. Such examples include, a specific customer's account balance on a specific date, the length of time an employee has been at his or her current salary level, the date on which an employee's salary was last changed, or the last date on which an out of stock event was experienced for a specific inventory item [Gora95].

The application of TDBS fall into two broad categories, applications that use time as an element (data item) and applications that use time as a dimension to define the data. Most applications have a very practical implementation of data structures to store temporal data and are not normally as sophisticated as the research in temporal databases that is currently proceeding.

The application of time as data (or as a data element) has many uses. Most notable are those applications that rely on time as a dimension to define or shape the meaning or application of data. Examples include insurance, in which claims and policy processing

المنارة للاستشارات
www.manaraa.com

carry a time element, and health care where patient history is necessary to diagnose systemic or illnesses with long time frames. Reservation systems in general are areas where time has an important meaning. TDBS of this type typically use timestamps to store time at the tuple level.

The application of time as a defining mechanism is a bit more complicated and requires an application where data has a definitive lifetime. Such areas include logistical applications, where entities in the database have certain properties (payloads) at certain times, and historical archives, where data as events are valid only during specified periods. TDBS of this type typically use time sequences (valid/invalid, duration, etc.) and sometimes employ bi-temporal techniques, the use of both timestamps and time sequence attributes, to track changes over time.

2.3.3 Comparison with ALV

Temporal data storage and versioning have become particularly important in several application areas, including temporal databases and version control and management systems. Although these two areas have a similar conception, the systems developed for each area have different concerns. Versions and temporal data are different concepts; temporal data reflects the states of objects in a time-oriented way; time does not normally apply to versions with regard to state. Rather, time is used to qualify when or during which time frame a version is created or valid. Techniques developed in temporal databases cannot solve the problems of versioning. That is, temporal databases are not designed to store versions of attributes. Temporal databases store events marked by time.

المنارة للاستشارات
www.manaraa.com

These events could indeed be a change to a single attribute, but there is no association with that change (event) to any other data nor is there association of additional metadata with the event. Those TBMS that do store versions always store the entire entity as a version (marked by timestamping or time sequences). In contrast, the techniques developed in the area of version control and management lack of the ability to support time-varying data [Lu96].

Temporal database theory can be applied to ALV by including temporal elements in the metadata for the attribute values stored. That is, it is possible to apply temporal elements to the attribute versions by storing temporal metadata that can be evaluated using either or both of the temporal data evaluation techniques. A study of how to add a temporal element to versioning is not a primary focus of this work. It is the use of metadata attributes associated with the attribute version that permits the storage of any number of timestamps or time sequence attributes. The implementation of temporal databases using attribute timestamping is a technique that accomplishes some of the goals of ALV, and represents a successful attempt to associate metadata at the attribute level versus the tuple level. The application of temporal database theory in versioning is an area that can be explored as future work.

2.4 Object-Oriented Database Systems

Object-oriented database systems (OODBS) are storage and retrieval mechanisms that support the object-oriented programming paradigm through direct manipulation of the data as objects. They contain true object-oriented type systems that permit objects to

المنارة للاستشارات
www.manaraa.com

persist between applications and usage. However, most lack a standard query language[13] (access to the data is typically via a programming interface) and therefore are not true database management systems.

OODBS are an attractive alternative to relational database systems, especially in application areas where the modeling power or performance of relational database systems is insufficient. These applications typically maintain large amounts of data, and additionally, often want to manage the whole history of the individual objects is stored and no data is ever deleted. “A key feature of object-oriented databases is to provide support for complex objects by specifying both the structure and the operations that can be applied to these objects [via an object-oriented programming interface],” [Date00, Elma93].

Although ORDBS are similar to OODBS, OODBS are very different in philosophy. OODBS try to add database functionality to object-oriented programming languages via a programming interface and platforms. By contrast, ORDBS try to add a rich data types to relational database systems using traditional query languages and extensions. OODBS attempt to achieve a seamless integration with object-oriented programming languages. ORDBS do not attempt this level of integration and often require an intermediate application layer to translate information from the object-oriented application to the ORDBS or even the host relational database system. Similarly, OODBS are aimed at applications that have as their central engineering perspective an object-oriented viewpoint. ORDBS are optimized for large data stores and object-based systems

[13] There are some notable exceptions [Orac05], but this is generally true.

المنارة للاستشارات
www.manaraa.com

that support large volumes of data (e.g., GIS applications). Lastly, the query mechanisms of OODBS are centered on object manipulation using specialized object-oriented query languages. ORDBS query mechanisms are geared toward fast retrieval of volumes of data using extensions to the SQL standard.

OODBS are particularly suitable for modeling the real world as closely as possible without forcing unnatural relationships between and within entities. The philosophy of object-orientation offers a holistic as well as modeling-oriented view of the real-world. These are necessary for dealing with an elusive subject like modeling temporal change, particularly in adding object-oriented features to the tuple-level database design mentioned previously. Despite the general availability of numerous open source OODBS, most are based in part as a relational system supporting a query language interface and therefore are not truly an OODBS, rather operate more like an ORDBS. All true OODBS require access via a programming interface.

2.4.1 Technologies

There is one very important aspect of OODBS that corresponds directly with that of the ALV technologies. In OODBS, each object can have a particular state and that state can change either over time or as events dictate. Storing the state of the object is a natural primitive for a versioning mechanism. In fact, many OODBS systems include a concept of versioning.

OODBS that track changes for objects use data versioning for tracking historical changes as well as for issues related to transaction management. This concept of

المنارة للاستشارات

www.manaraa.com

versioning permits increased concurrency operations by allowing multiple views of the same entity, each with (perhaps) a different set of historical values of the data, but its primary benefit is that it can be used to execute queries against historical data. In this perspective, objects are versionable in that several versions can be derived from one object. Versions are either active or committed. An active version c of an object begins as a copy of a committed version which can then be manipulated independently of all other such versions. The values of the active version may be modified extensively for some period. Eventually, the modified active version may be promoted to become a new committed version, if its state is consistent with the current state of the other committed versions; otherwise, it is disposed. These multiple versions[14] are placed in a version chain – normally represented as a graph or tree in memory – such that the most recent correct version is stored at the head of the chain and is called the last committed version. A new committed version is added in an appropriate position in the version-chain. The version-chain of an object effectively captures the evolution of the object through time (by preserving its historical information) [Hada02].

However, in object-oriented databases, support for evolution is a critical requirement since evolution is characteristic of complex applications (*e.g.* computer-aided design and manufacturing; office information systems) for which they provide support. Due to the underlying rich data model (in contrast with conventional data intensive record processing applications) these applications require dynamic modifications to both the data residing within the database and the way the data has been

[14] The author [Hada02] refers to them as multiversions and the concept as multiversioning, which is similar to the multiversioning technique used to detect version collisions in distributed database systems.

www.manaraa.com

modeled, *i.e.* both the objects residing within the database and the schema of the database are subject to change. Furthermore, there is a requirement to keep track of changes in case they need to be reverted.

Historically, the database community has employed three fundamental techniques for modifying the conceptual structure of an object-oriented database, namely:

- schema evolution where the database has one logical schema to which class definition and class hierarchy modifications are applied
- class versioning which keeps different versions of each type and binds instances to a specific version of the type
- schema versioning which allows several versions of one logical schema to be created and manipulated independently of each other

In addition, a number of mechanisms have been created for managing the evolution of objects residing in the database. These strategies are called object versioning strategies [Rash00]. These include the following classes of strategies [Katz90]:

- organization of the space of versions – the managing of a version set (a set of all variants)
- dynamic configurations and dereferencing – the realization of versions at run time
- hierarchical compositions across versions (configurations) – the versions are stored and referenced as hierarchical objects using tree and list mechanisms

المنارة للاستشارات
www.manaraa.com

- workspace organization – the versions are managed in a workflow manner, establishing a set of sets of versions that form the workspace that versions can be derived from

One interesting concept for storing temporal version information in object-oriented databases concerns a technology called "temporal versioning mode" (TVM) [Lu96]. This technology employs a combination of schema versioning and object tracking mechanisms where each version (instance of an object) has its own finite lifetime. Access to the individual version history is accomplished via a version identifier (similar to a surrogate key). This technique applies concepts from both the temporal database and document management systems to form a technology that can successfully store and track version history for objects.

2.4.2 Applications

Application areas of object-oriented database systems include GIS systems (geographical information systems), scientific and statistical databases, multimedia systems, PACS (picture archiving and communications systems), and XML warehouses.

2.4.3 Comparison with ALV

ALV is not intended to be used in OODBS systems. Since OODBS systems include a concept of versioning[15] supported by the object-oriented paradigm as state, the

[15] Or could easily given the flexibility of object -oriented technologies.

www.manaraa.com

ALV technology simply isn't needed. However, there is no such correlation for relational database systems. Thus, the integration of a versioning system (ALV) with that of relational databases will permit relational database systems to model state (version history) in a way analogous to that used in object-oriented databases.

Research in the area of exploiting temporal data in OODBS has shown a split in philosophy as to whether to associate time with the objects and their state or within the attributes themselves. Some researchers [Jens92] are exploring object versioning (the storage of time using two time intervals during which time the object version is valid [Elma93]), while others [Gadi88] are exploring attribute versioning (the storage of time intervals with each attribute where the sum of the attribute versions comprise the version of the object in time [Elma93]). Much of the research into the former area will form the foundation for applying temporal data using the ALV technologies in future evolutions of the technology (see Chapter 8).

2.5 Object Relational Database Systems

Object relational database systems (ORDBS) is an application of object-oriented theory to relational database systems. ORDBS provide a mechanism that permits database designers to implement a structured storage and retrieval mechanism for object-oriented data concepts. ORDBS provide the firmament of the relational model – meaning, integrity, relationships, etc. – while extending the model to store and retrieve data in an object-centric manner. Implementation is purely conceptual in many cases as the mapping of object-oriented concepts to relational concepts is tentative at best. The

المنارة للاستشارات
www.manaraa.com

modifications, or extensions, to the relational technologies include modifications to SQL which allows the representation of object types, identity, encapsulation of operations, and inheritance [Elma03]. However, these are loosely mapped to relational theory as complex types. Although expressive, the SQL extensions do not permit the true object manipulation and level of control of OODBS. The most popular ORDBS is ESRI's ArcGIS environment [ESRI00]. Other examples include Oracle and Informix [Elma03].

2.5.1 Technologies

The technology used in ORDBS uses the base relational model. Most ORDBS are implemented using existing commercial RDBMS such as Microsoft SQL Server and Oracle. Since these systems are based on the relational model, they suffer from an acute conversion problem of translating object-oriented concepts to relational mechanisms. There are many problems with using relational databases for object-oriented applications [Risc04]. Such problems include:

- Complex mapping from the OO conceptual model to relations
- Complex mapping implies complex programs and queries
- Complex programs implies maintenance problems
- Complex programs implies reliability problems
- Complex queries implies that the database query optimizer may be very slow

المنارة للاستشارات
www.manaraa.com

- More vulnerable to schema changes than relational systems because of the mappings of object concepts to complex types[16]
- Performance

Although these problems seem significant, they are easily mitigated by the application of an object-oriented application layer that communicates between the underlying relational database and the object-oriented application. These application layers permit the translation of objects into structured (persistent) data stores. Interestingly, this practice violates the concept of a ORDBS in that you are now using an object-oriented access mechanism to access the data, which is not why ORDBS are created. They are created to permit the storage and retrieval of objects in a relational system by providing extensions to the query language[17]

Unlike true OODBS that have optimized query mechanisms, such as ODL/OQL, ORDBS use query mechanisms that are extensions of the SQL query language.

2.5.2 Applications

The ESRI product suite of GIS applications contains a product called the Geodatabase – shorthand for geographic database – which supports the storage and management of geographic data elements. The geodatabase is an object-relation database

[16] This is especially true when the object types are modified in a populated data store. Depending on the changes, the behavior of the objects may have been altered and thus may not have the same meaning. Despite the fact that this may be a deliberate change, the ef fects of the change are potentially more severe than in typical relational systems.
[17] And, presumably, necessary or appropriate changes to the physical layers.

المنارة للاستشارات
www.manaraa.com

that supports spatial data. It is an example of a spatial database [Elma03] that is implemented as an ORDBS[18].

Using the geodatabase, integrity rules and behavior can be defined for spatial data, allowing the modeling of important geographic objects, such as networks, terrains, and image catalogs. For example, the United States Census data can be modeled using the associated integrity rules:

- Census blocks cannot overlap
- Census blocks must fully cover a given geographic region
- Census blocks must be contained within the boundaries of the given region

Integrity rules and behavior are implemented on intermediate representations of the data in an abstract form – an "object" (i.e., software components that instantiate and animate the rules and behavior stored in the database).

Therefore, the geodatabase is an ORDBS implementation of the elements in the GIS environment, or "objects," which are stored in relational tables designed to store the state and operations of the objects in a persistent store [ESRI00]. In general, ORDBS provide a richer environment to implement object concepts in a database while providing the sound firmament of the relational model.

[18] There is no requirement that spatial database systems be implemented in ORDBS, e.g. Oracle [Or ac05]. ESRI has chosen to implement the geodatabase as an ORDBS. The geodatabase is used in this work as a reference/example for a ORDB because it is well understood by the sponsors of this work.

المنارة للاستشارات
www.manaraa.com

2.5.3 Comparison with ALV

ORDBS can persist the state of objects. This is usually implemented in a horizontal fashion where state is persisted in one or more tables in one or more parent/child relationships. Like relational databases, implementations of ORDBS are designed to save only the most current state of the object. The only ORDBS found to have any versioning capabilities is ArcGIS. Fortunately, since the ALV system is an extension of a relational database system, ORDBS can also leverage the ALV mechanisms to provide a versioning mechanism for saving state history of objects. Thus, ALV can be used to enhance the ORDBS extensions to include versioning.

2.6 Other Considerations

The following sections detail additional areas of interest found during this research. Each topic is discussed briefly and concludes with statements of applicability to this work.

2.6.1 Security

Multi-level security (MLS)[19] requires the ability to segregate data based on predefined groupings of access permissions (users) to corresponding partitions of the data. Many organizations require this level of additional security to protect data considered sensitive or otherwise damaging should it be disclosed to unauthorized

[19] The acronym MLS falls into an interesting category of term s that sparks emotional and sometimes aberrant behavior among scientists and engineers – few can define exactly what it is, but many have a long list of personal opinions of what it isn't.

المنارة للاستشارات
www.manaraa.com

personnel. Access to these databases must be restricted and controlled to limit the unauthorized disclosure or malicious modification of data contained in them. However, the conventional models of authorization that have been designed for database systems supporting the hierarchical, network and relational models of data do not provide adequate mechanisms to support controlled access to the data. These systems use a combination of dedicated hardware and customized application access layers that require custom development and integration to use. They are therefore not applicable for use in developing systems of systems solutions[20].

However, there are solutions that do present a viable alternative to these dedicated mechanisms. Pissinou et. al. present a temporal multi-level security (MLS) mechanism for temporal data [Piss94]. The solution involves extending the multilevel secure relational model to capture the functionality required of a temporal database. This is accomplished by assigning class access to temporal attributes and assigning security classifications to the temporal elements. This solution is similar to the goal of tracking security in the ALV system. Security tracking in this sense must associate a classification with each object in the database; all data must be tagged with the appropriate metadata that permits the inclusion or exclusion of data based on classification groupings.

ALV is designed to associate one or more metadata attributes with each attribute version. This meta attribution will allow the data to be tagged with one or more categorical attributes that can in turn be used to partition the data. Thus it is possible to store data in an ALV enabled data store using multiple security levels. This partitioning

[20] Specific reference to systems of this nature is beyond the scope of this work.

المنارة للاستشارات
www.manaraa.com

of the data would then permit the selection of attribute versions (and thus versions of the entities) based on the categorization attributes. An application of this technique would allow retail organizations store cost of goods and inventory data along with sales and customer data – each datum being one or more attribute versions. When the data is presented, say to a customer, that user's access would be set to permit the viewing of data for the customer permission set.

2.6.2 Legacy Application Support

Legacy applications are those applications that are in life cycle maintenance without evolution or technological advancement. Few of the technologies explored[21] addressed the need to continue to support legacy application of the technologies being expanded or created. With the possible exception of ORDBS, supporting legacy applications using modern database systems is impractical at best [Bohl98]. Today's information technology and global economy are forcing businesses and organizations to consider the value of their legacy systems and to plan evolution of those systems through extensions and interoperability rather than total system replacement. It is no longer feasible to retool every 18 months[22].

This is especially true in areas such as finance, marketing, and property and resource management. Many database applications manage data that must be versioned –

[21] This category refers to commercially available and open -source repository systems. There are a few experimental and academic systems that address various aspects of legacy support but none fully satisfy the requirement.
[22] This isn't helping the gap between industry and academia. It only further expands the application of advanced theories that are based on new systems. ALV attempts to close that gap by implementing the technology on existing systems (RDBMS).

المنارة للاستشارات
www.manaraa.com

be that spatial, temporal, or historical. These applications typically use relational systems. If versioning is to supported, the versioned data must either be accessible using the DBMS, or employ the services of a proprietary system that co-exists with the DBMS. Clearly, the goal is to enable versioning for these legacy systems[23]. Versioning of data has been investigated from an enterprise perspective by leveraging enterprise architectures to supply versioning capabilities in a middle tier rather than at the database or repository level [Chatt04]. This technique permits the database to remain unchanged while providing a degree of versioning capabilities. As previously stated, this is also a primary goal of ALV. However, with ALV the versioning mechanisms are being built into the database system itself, thereby ensuring a more tightly coupled and consequently potentially more efficient versioning mechanism.

Legacy support is one of the key areas that ALV is designed to address. Since ALV is designed as an extension of the relational database system, the advanced technology of version storage and retrieval is available to all legacy applications. Thus developers can modify existing legacy applications to enable versioning without modifying their original schemas or worse, porting their database repository to a incompatible repository model (e.g., converting a relational database to that of a pure object-oriented database or vice-versa[24]).

---

[23] It should be noted that the resulting changes to the applications can no longer be considered legacy applications.

[24] Arguments can be made that moving a well designed relational model to that of an object -oriented model can be far less problematic. Unfortunately, few relational systems exhibit the goodness of fit that the relational model enforces – see footnote #11 in section 2.2.2.

www.manaraa.com

A wealth of legacy systems and applications will benefit substantially from built-in, integrated versioning support. Providing a foundation for such support is an important and substantial challenge for this project.

2.6.3 Graph Stores

Repositories that store data in the form of edges (directed or undirected) and nodes are gaining popularity especially in the application of Semantic Web technologies. Graph stores[25] store data in a very different way than traditional relational database systems and their derivatives. As mentioned, data is either an edge or a node. Some graph stores allow storage of metadata along with the elements. There are many efforts underway exploring graph stores. Examples of graph store systems include Inkling, JENA, KAON, Parka, RDFSuite, Sesame with SAIL, and TAP [Beck03, Fens03, Magk02].

Versioning is possible with graph stores. One vendor has implemented a crude form of Attribute-Level Versioning[26]. The graph store permits duplicate entries. These duplicate entries are called versions of the data item stored. Metadata tags are used to distinguish currently selected or preferred values. Although a very effective means of storing all possible information about entities (or links), this implementation fails to meet the needs of this work to preserve connectivity to existing legacy applications, is not optimized for version retrieval, and does not permit any analysis of the version history.

[25] Sometimes called RDF stores, object stores, or object databases.
[26] Under contract for the federal government and subject to protection restrictions which forbid disclosure of the vendor and the technology. The author is principally involved in the development of the versioning mechanism.

المنارة للاستشارات
www.manaraa.com

Further refinement could enable such features, but the implementation will never fully meet the legacy compatibility constraint because the concept of a graph store does not meet the relational model – the model upon which the majority of all data stores are based. Lastly, the graph store concept itself is not designed to store the type of bulk data that most databases hold. This is especially true for query processing. Most graph store implementations store the relationships in memory. The more data, the more relationships must be stored and therefore more memory must be used to process the queries[27]. Thus, performance and scalability are the primary concerns for this technology.

2.6.4 Spatial Data and Temporal Databases

There has been considerable research in temporal-spatial versioning mechanisms. Spatial data consist of spatial objects made up of the states (non-spatial attribute values), positions (the positions with geographic space) and shapes (the geometry features of the object; which may contains line, regions, etc.). Spatial databases facilitate the storage and processing of spatial and non-spatial data. However, regardless of the shapes of the objects, the problem is still the processing of the states of objects where the objects are in a 2- or 3-dimensional geographic space. Nevertheless, the storage of states is quite similar to temporal data processing where data is tagged with temporal attributes that can be queried, although the spatial semantics are different from temporal semantics. Therefore, a temporal-spatial data model can be possibly established by defining a

[27] It is common to see requirements of 64-bit processor support for these systems not because of the advanced instruction set, but for the expanded memory model supported by the 64-bit processor.

المنارة للاستشارات
www.manaraa.com

conceptual temporal-spatial space, specifying the temporal spatial semantics and extending the object-oriented data model to capture temporal-spatial versions [Lu96].

The most popular system that implements spatial-temporal data is ESRI's ArcGIS environment. As discussed previously, the versioning mechanism for ArcGIS is a workflow-based mechanism that permits vetting of changes by the community of users. Unlike the description above, ArcGIS is an object relational database implemented on Microsoft SQL Server, Sybase, or Oracle varieties.

2.6.5 Long Transactions

A "long" transaction can be loosely defined as one which is either left open (uncommitted) indefinitely (for example, while a system processes a large data set), or which is left open for longer than is required by an application. Long transactions are commonplace in distributed computing where it may not be possible to synchronize all of the nodes in the distribution at the same time. Thus, the transaction has an extended lifetime (remains active longer than the process that initiated requires). In some cases, the transaction is used as a means of forming a recovery log thereby making the transaction semi-persistent. The technique of using long transactions in this manner is called multiversioning [Hada96, Lome90].

Long transactions are implemented using row-level versioning. A database table is version enabled by augmenting keys of the entities in the table (rows) with a version number, thereby creating a composite key that is guaranteed to be unique. Changes done in a long transaction are tagged with version numbers unique to the long transaction

المنارة للاستشارات
www.manaraa.com

[Chatt04]. Since there is a version tag for each entity and a storage structure for the long transaction, the system is capable of rolling back the state of an entity to any point in the evolution of the entity (changes of its attributes). The application of long transactions therefore permits the extended storage of state throughout a distributed database system.

2.6.6 Versioning Requirements

The database systems and research areas surveyed present a common set of requirements that define what versioning should be. Although some of these requirements do not apply directly to any one particular database technology, these requirements align neatly with those presented that define ALV (see chapter one).

There are four fundamental requirements for versioning of data. This work and the project created to realize and prove this work are designed to meet the requirements as specified below. Although there are many more specific requirements for ALV, the following are the fundamental requirements found in the literature common to all forms of versioning.

2.6.6.1 Transparency

The most important requirement for versioning is that the versioning mechanism should be as transparent as possible to permit continued operation of relational database systems and software. No changes should be required in the database application code to accommodate data versioning. Furthermore, the logical database should be configurable from outside the application code before starting execution.

للاستشارات المنارة
www.manaraa.com

Development and testing of database applications is difficult because the program execution depends on the persistent state stored in the database. That is, when a particular feature or function of the system is tested, the state of the data that the function operates on must be in a predictable state. Often the case is that the test data is not placed in configuration management and therefore is not held in a predicable state. Versioning of the persistent data stored in the database can solve some critical problems in the development and testing of database applications [Chatt04] by associating the versions of the data with the appropriate state of the test. For example, one version of an entity in the database could have a separate version for before, during, and after a function is executed. This permits testing the function with known data in a predictable state.

2.6.6.2 Multiple Stores

The versioning mechanism must support consistent logical (possibly hierarchical) states of the database simultaneously in the same physical database. Changes made in a physical database state should not be visible outside it. The same data should be simultaneously modifiable in multiple databases states.

There is a catch. Versioning will generate potentially huge amounts of data. Consider adding a metadata set with a total length of 1000 to each attribute version of length 25. The ALV store will require 1025 storage bytes for each attribute version. Thus if there are 1000 tuples in a database that each have 10 attributes which generate 10 attribute versions each, the additional version data will be 1000*10*10*1025 or 102,500,000 storage bytes. If the versioning mechanism is associated with the database

المنارة للاستشارات
www.manaraa.com

system and not by database or table, the version store will quickly grow to an unmanageable size. For this reason, the version store for ALV must be associated with tables rather than the database and thus provide the potential to store thousands of attribute versions per tuple.

This requirement ensures that the versioning mechanism can be leveraged against logical partitions of the data (i.e., at the table level) without affecting or imposing the same mechanisms on other partitions. This is especially necessary for systems that require versioning for a specific table or database and not all of the databases supported by the system. This also ensures that the versioning mechanism is used where appropriate and will not impede normal operation of database systems.

2.6.6.3 Implemented as an Extension

The versioning mechanism must continue to support all widely used relational database services such as triggers, constraints, etc. These must be supported in the native form of the hosted database system and apply to the versioning store itself where appropriate.

This requirement will ensure the versioning mechanism exists as an extension of the host system without impeding functionality, performance, or implementation. This will also ensure that the versioning mechanism conforms to known standards to the extent that the host system supports them.

المنارة للاستشارات
www.manaraa.com

2.6.6.4 Support for Recovery

The versioning mechanism must be maintainable within the context of a logical database server state. This includes administrative tasks designed to support tuning and preventive maintenance. Failure to implement this requirement will render the mechanism less transparent. For example, failures in the versioning mechanism may not be identified or recovered in the same manner as those of the host database system. This requirement will ensure the versioning mechanism conforms to known administrative operations, making the system more easily incorporated into an environment in which database administrators have a defined role.

ALV will permit database designers and analysts to leverage advanced versioning mechanisms that support their legacy applications. Since ALV is implemented in MySQL (see Appendix B for complete details of the MySQL implementation), the system has direct access to all of the services of the RDBMS thereby allowing ALV to become a seamless extension of MySQL. Directing control to the ALV system is a transparent operation that can be implemented where needed and will permit the MySQL system to be modified to include the version store in commonly used administrative tasks.

2.6.7 Concept versus Form

Available database systems fail to demonstrate any reasonable attempt at versioning of data that meets all of the requirements as stated above. Furthermore, several works suggest that one approach – database snapshots – is not only limited but also flawed [Blak04].

المنارة للاستشارات

www.manaraa.com

Similarly, the concept of storing all known values – the packrat concept to data storage – this raises the question of 'When is the data no longer unique and when does it transition from historical reference to junk?' A clear (semantic) problem pinpointed by this question is that known as the 'Ship of Theseus' debate: how much change can a predefined 'entity' undergo before it ceases to be a new version and becomes a new entity?

Interestingly, it seems that current implementations, practices, and theories regarding versioning do not match the analysts' perception of and reasoning about the data. In relational database terms, there is a mismatch between the logical, implementation-oriented view of data supported by the tools, and the application-oriented, conceptual view that users follow in their everyday work. This mismatch is similar to that of traditional database management many years ago, when the market favored the relational approach and the conceptual-to-logical gap was filled by database design CASE tools based on the entity-relationship (ER) approach. Since then, the advantages of the conceptual approach to data modeling have been extensively demonstrated, in terms of user involvement and of durability of the design specifications [Pare99].

It is thus foreseeable that a similar evolution will enable the creation and adoption of a usable versioning mechanism. The versioning mechanism must enable analysts to draw conclusions about their data by considering all known – previous and predicted – values of the attributes of the entities in the data. Patterns of change alone could make the difference between a logical estimation and a modeled behavior.

المنارة للاستشارات

www.manaraa.com

However, words of caution are sprinkled throughout the literature concerning reaching too high too soon. "Experience has shown that striving for the highest expressive power leads to unbearable complexity and eventually results in rejection of the [solution]" [Pare99]. Versioning systems must therefore also be adoptable immediately by the analysts or else they too will become a novelty reserved solely for the experimenter or academic.

2.7 Conclusion

A thorough examination of the state of the technologies and theories in the body of knowledge that is Computer Science clearly presents a gap in the research concerning the versioning of data with regard to relational database theory. Although object-oriented databases can inherently support a versioning concept and object relational databases can support a horizontal mechanism for version storage, none of the database paradigms support versioning at the attribute level while maintaining a functional connection to traditional relational databases.

Centering the ALV technology in the relational database application arena protects the sanctity of relational databases systems[28] while permitting the inclusion of a powerful versioning mechanism. This will enable scientists and analysts to prepare data for use in rigorous database applications drawing from the repository of all known or predicted values.

[28] Pertaining to the theoretical (academic) application of theory. Commercial relational database systems often compromise the finer details of relational theory for the sake of mass reuse and generalization of functionality. Hence the persistent and growing gap between academic rigor and industry isotropic applications.

المنارة للاستشارات
www.manaraa.com

Attribute-Level Versioning therefore is a uniquely conceived idea that has merit in the relational database paradigm. The continued exploration of versioning capabilities and implementation of ALV will permit the growth of a new direction in data versioning – the ability to store every permutation of a data's attributes.

المنارة للاستشارات

www.manaraa.com

Chapter Three – Introduction to ALV Technologies

This chapter contains an introduction to the technologies and research conducted in support of this work and the related technology project. The ALV project is a very large, complex project. There are many opportunities to explore several sub-disciplines within the engineering and computer science disciplines. This work will focus on those areas that require new algorithms, structures, technologies, or unique applications of current technologies. Fortunately, several emerging technologies will solve the more mundane areas of the project such as data repository assimilation and deconfliction (i.e., Semantic Web [Anto04, Fens03]). However, there are no existing solutions for the storage and retrieval of versioned data. These areas are the most critical to the success of this project and therefore comprise the bulk of the work performed.

ALV requires a specialized storage structure, a fast retrieval mechanism using a specialized indexing construct, and extensions to the SQL language. Another important area is the application of data-mining algorithms. New data-mining algorithms are necessary to complete the knowledge engineering process to gain additional knowledge about the datasets and their attribute versions. All of these areas will be explored by researching the following four technologies:

www.manaraa.com

- Storage Mechanisms
- Indexing Mechanisms
- SQL Language Extensions
- Data-mining Algorithms

This work will research, design, and implement these technologies within the aspects of the ALV project and will analyze their application.

This chapter begins with a presentation of the constraints of the related project as a way of explaining the implementation environment and practical limitations. The following sections, 3.2-3.5, will describe the research necessary to master the disciplines to create a viable solution for each of the needed technologies. Section 3.6 will address the application of emerging and related technologies. Section 3.7 concludes with a summary of the ALV technology. Section 3.8 forms an introduction to the format and content of the subsequent chapters.

3.1 Project Constraints

The supporting need for this research is a knowledge engineering project designed to combine multiple datasets about a subject area. This solution must be capable of storing known associations and relationships among the data, resolving collisions, storing collisions among the data attributes as alternative values, analyzing the alternative values,

المنارة للاستشارات
www.manaraa.com

and forming a hypothesis from the analysis. The output of this process will be a dataset that is the result of a query from the combined data and the versioned attributes.

There are a number of restrictions imposed on the project by the project sponsors. These restrictions are due largely to the limitations of the environment in which the project must reside. In many ways, the supporting project for this work must bridge the gap between the application of theory and the practical limitations of using the product. Unlike most research projects where implementation details are left to later evolution and refinement of the prototype or proof of concept, the practical application of this work will be used to evaluate its ability to solve a hitherto unsolved problem – versioning relational data. These restrictions create a unique avenue for applying advanced research to real-world problems. The following paragraphs detail some of the more important restrictions that have a direct bearing on the product of the research of this work.

The implementation of the storage mechanism is limited to the Microsoft Windows platform, specifically Microsoft Windows 2000 Server, and is designed to run on Intel PC-based server hardware. As such, the storage mechanism must operate within the Windows operating system programming interfaces. This prohibits the design of specialized disk access algorithms. However, this does not prohibit the exploration of unique applications of known structures and memory access techniques. The upper memory limit for the chosen hardware is 2GB of RAM.

In order to ensure that the system is performing well for the chosen hardware, the system must include mechanisms that allow systems engineers to tune portions of the system for optimization for a given task. These parameters include, but are not limited to,

المنارة للاستشارات
www.manaraa.com

block size and buffer size for the buffer manager. Another constraint is that the system must be capable of capturing and displaying statistics for all data stored as per traditional relation database management practices.

The ALV project was created to integrate a versioning mechanism with a traditional database management system. The goal is to leverage a versioning technology from within the DBMS. The technology should become an extension of the DBMS rather than an add-on module or procedural access mechanism. An extension to existing technology will allow legacy applications to leverage versioning without having to redesign or alter the underlying database schema. Additional facilities can be added to permit the inclusion of the versioned data without changing the fundamental mission of the software. More importantly, this goal will ensure that the output of this project can be consumed by legacy applications that rely on relational database systems for storage.

Among all of the available DBMS, only those that provide direct manipulation of the server code were considered. This restricted the selection to open source systems. Further implementation guidelines set forth by the sponsor of this work eliminates any open source system based on Java[1] (e.g., Apache Derby). Open source systems are generally licensed using a GNU[2]-based license agreement. Most permit free use of the original source code with a restriction that all modifications be made public or returned to the originator as legal ownership. In the case of ALV, the sponsors have the choice to enlist support from the manufacturer, purchase rights to modify the system, or return the ALV technologies to the owner for incorporation into the product. It is the desire of the

[1] Silly, yes, but true nonetheless.
[2] GNU stands for GNU, not Unix.

المنارة للاستشارات
www.manaraa.com

author that the ALV technology be returned to the originator for incorporation into the next public release of the system. If permitted, this would perpetuate the open source mantra and give back to the community in exchange for what was freely offered[3].

Among all of the goals of this system, most paramount was to remain dedicated to traditional relation database implementations. The only possible candidate under these conditions is MySQL[4]. Therefore, this project will focus on implementing the solution as an extension to the MySQL DBMS, which will require adaptation of the MySQL engine, extension of the Structured Query Language (SQL), and additional tools to manipulate the data prior to and after storage.

Businesses today are frequently forced to adopt new solutions in shorter and shorter timeframes in order to compete in today's global economy. The problem with past solutions is that integration efforts have taken too long, and we have created inflexible, hard-to-change architectures. The stigma of planting the seed of technology and waiting until it grows into a mature product, much like the proverbial bamboo tree[5], is a relic of what could be considered the golden age of industry-sponsored computer research. The ALV project is an attempt to show that cutting edge research can lead to immediate solutions for industry.

The choice of MySQL running on the Windows server platform has introduced a number of unexpected limitations that have required some amount of additional work. For example, although MySQL is designed to run on a wide variety of platforms, the

---

[3] A concept that traditional software industry (Microsoft et. al.) has to date failed to see the wisdom of sponsoring.
[4] MySQL is a registered trademark and product licensed to MySQL AB. http://www.mysql.com
[5] The Chinese bamboo tree must be cultivated and nourished for four years with no visible signs of growth; however, in the first three months of the fifth year, the Chinese bamboo tree will grow 90 feet.

المنارة للاستشارات
www.manaraa.com

preferred development platform is Linux[6]. Consequently, some of the development tools are not available on the Windows platform. Furthermore, many of the debugging capabilities are lost as a result. Despite these issues, the project has been successfully developed and tested. Future plans include porting the source code to Linux for further refinement and evolution.

3.2 Advanced Storage and Retrieval Mechanism

Storage engines are the life of a database system. Although query optimization and query execution comprises a significant amount of the processing time that database systems expend, it is the storage facilities that the system relies on to be fast, efficient, and scalable. Without a fast storage and retrieval mechanism, database systems cannot fulfill the needs of their users. When users query a database for information, they expect answers immediately. If the system must navigate a complex or inefficient file system, performance will suffer and users will eventually abandon it.

The storage engine for ALV is designed to be both fast and efficient with respect to query response time. A common theme throughout this project is to sacrifice space and memory whenever doing so will enable faster access or processing. Today the cost of disk space is more economical than memory or processor technology.

Since the storage engine is an extension of the relational model and subsequently an extension of the host database system, the storage engine requires a reference to an original host record. This reference becomes the primary indexing or storage organization

[6] Considered by MySQL AB as the "only" platform worthy of true server-level development – other platforms are supported only as a convenience for developers and integrators.

المنارة للاستشارات
www.manaraa.com

parameter. Thus, each item stored in an ALV file references another entity stored elsewhere in the host system. This concept is applied at the table level, enabling each table to be versioned. This creates a one-to-many relationship between a table and the ALV version store[7] for that specific table. Thus, the storage engine provides access to all of the version data by extending the relational database system to include a version store for each table store.

The storage engine for ALV is implemented using a clustered storage mechanism that ensures that all attributes for a given host record are stored in the same series of storage blocks on disk. This permits fast retrieval for all of the attribute versions for a given host reference (tuple). Attribute versions are sets of attributes containing the attribute value, metadata (user defined), the reference to the host tuple, and pointers for linking attribute versions for the same attribute together. Each set of attribute versions is saved as a structure within a block of data on disk.

These structures are placed in a section of a file called a block[8]. Files on disk are accessed one block at a time. The sizes of blocks are largely implementation dependent, but typical sizes are 512kb and 1024kb. When files are used to store data such as the data stored in database systems, the data must be partitioned so that objects are stored within the block size specified. When more than one object can fit into a single block, additional care must be taken to fill the block with data. Another consideration is whether to permit objects to span across one of more blocks. The ratio of objects to blocks is referred to as

[7] Version store is the logical term applied to a set of attribute versions organized into a single file and organized using a clustered mechanism.
[8] Sometimes referred to in the literature as a page.

المنارة للاستشارات
www.manaraa.com

the blocking factor [Date04]. Many of the concepts of file systems such as paging algorithms, data compacting (packing), the recovery of delete blocks, etc., are used in constructing storage engines for data.

Most file systems provide mechanisms to buffer the blocks read to memory. Some buffering techniques provide read-ahead algorithms that attempt to anticipate the next block needed [Kim88]. The technique used in buffering to manage the blocks in memory is called paging[9]. Some file systems provide mechanisms to read more than one block at a time. Many of these file systems contain buffering mechanisms [Kim88]. Systems designers can take advantage of this feature by grouping objects together in a set of blocks called a cluster. Cluster technology is the main technique employed in the ALV storage engine.

Attribute versions are stored unordered within a block of data on disk. A mapping mechanism consisting of a lookup table is stored at the front of the file and provides the mapping of each attribute version (similar in many ways to a tuple or "record[10]" layout). Another map, used at the front of each data block, contains the address (offset) of each of the attribute version linked lists (specifically, the first node or "head" of each list). Storing all of the attribute versions for a referenced record within the same block on disk will yield a higher retrieval performance than more traditional interlaced record/block access mechanisms. Thus, the version store is a clustered version store optimized for fast retrieval of attribute versions and attribute version chains.

[9] Thus the tendency for researcher to refer to blocks as pages.
[10] The temptation to use the term "record" is powerful. In many respects, the colloquial expression "record" is more recognizable. The author shall refrain from this ambiguity wherever possible.

المنارة للاستشارات
www.manaraa.com

The blocks in the file are maintained using a queue which manages a free block list. The queue enables the fast recovery of unused data blocks without needing to compress or reallocate the data on disk. The buffer management algorithm uses the free block list to reclaim blocks on disk and order them sequentially whenever space allows. The size of the blocks is adjustable and can be set to maximize the host operating system's file system performance.

A page replacement algorithm is used to reclaim space within the data block as attribute versions are deleted. This algorithm walks the file, moving the next block of data into the first available block at the front of the file. This process continues until all empty blocks are filled, thus leaving empty blocks at the end of the file. The algorithm completes by initializing the free block queue and truncating the file at the first empty block. A block extension mechanism is provided to allow the overflow of data across multiple blocks, which form a linked list on disk. Thus the blocks that comprise the set of all attribute versions are also used as a parameter in the buffer manager as a heuristic in predicting access paths for faster retrieval of sequential attribute version reads. Figure 3-1 depicts the layout of the ALV file storage.

Auxiliary mechanisms are used to provide direct access to the data on disk. These mechanisms are called indexes. Indexes allow systems to read data for a particular object by reducing the number of reads to a single read for the block that contains the object. Note: sequential file reads (also called a table scan) do not require the use of an index. The index mechanism returns a block number and offset for the first attribute version referenced in the index criteria (see section 3.4 for more about the index mechanism).

المنارة للاستشارات
www.manaraa.com

The index and cluster mechanism combine to provide a minimal number of block reads from disk. Minimizing block reads from disk, which are the most costly of all operations save writing to disk[11], will ensure that the storage engine is as efficient as possible.

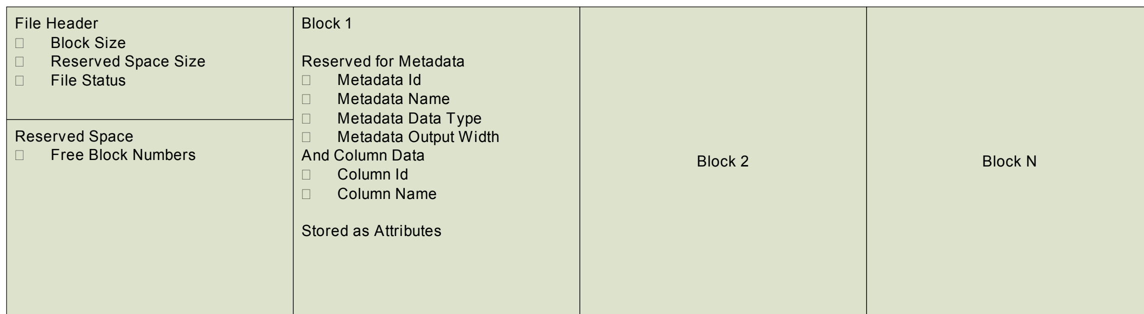


Figure 3-1: The ALV File Storage Layout

Writing data in the storage engine isn't as efficient as reading from it. The clustered mechanism is a bit more complicated whenever data is inserted. If there is space available in the version store for a new attribute version, the performance is fast. However, if an attribute version is to be inserted in a block that does not have space available, an extension for that block is created and the new data added to the new block. Although only slightly slower, this condition is affected by the block size chosen. Similarly, additional processing is required for inserting attribute versions into spaces left vacant from delete operations. Lastly, the buffer manager is designed to flush to disk any blocks in the buffer that have been written. The timing of the flush can be delayed, but studies show that immediate flush operations do not pose a performance problem in low concurrency situations.

[11] Hardware optimization mec hanisms notwithstanding (e.g., RAID).

www.manaraa.com

ALV provides a mechanism to store and retrieve all values for an assimilated attribute *i.e.* an attribute that has been identified as having duplicate values as a result of combining data from two or more data sources. Having a comprehensive version repository with efficient retrieval mechanisms permits scientists to perform additional analysis using advanced statistical and knowledge inference models that identify meaning or predict relationships among the data. Adding this new dimension of analysis opens a new chapter in the data-mining paradigm. Chapter 4 contains details of the implementation of the ALV storage engine.

3.3 Advanced Query Mechanism

A fast storage and retrieval mechanism is only half of the equation for a high performance database system. The system must also be capable of performing complex queries to retrieve answers from the data. The language of choice for queries is SQL [Date04]. The systems must translate query statements from high-level SQL to an executable sequence of sub-processes [Date04a]. Furthermore, this translation must be precise and repeatable, for the power of a database system is its ability to produce the same answer using the same data for the same query statement[12] every time it is executed [Ston76, Ston81].

Implementing a solution for the ALV query processor in MySQL required modifying the MySQL parser and lexical analyzer. The MySQL parser was written using

[12] There are often several permutations of the same query statement that result in the same answer. SQL is often criticized for its freedom of expressiveness. Fortunately, most database system implementers build their query mechanisms to accommodate this expressiveness.

المنارة للاستشارات
www.manaraa.com

Yacc and Lex. Lexical analyzer and parser technology are an area of research in compiler design and language theory. Database systems researchers often refer readers to compiler and language theory texts for implementation methods and explanation. As a result, SQL language extensions require the application of language theory in order to extend the MySQL parser to recognize and validate the new extensions. Modification of these systems required changing the lexical analyzer to recognize new tokens and changing the parser to recognize new patterns in the SQL language. The ALV SQL, named SQL_{ALV}, extensions are designed to mimic those of SQL, representing the typical data manipulation commands such as select, update, insert, delete as well as the typical data definition commands such as create and drop. Samples of these commands are shown in table 3-1 below. A complete explanation of these commands and their implementation is discussed in Chapter 6.

The parser and lexical analyzer in MySQL identifies strings of alphanumeric characters (tokens) that have been identified in the parser. The parser tags all of the tokens with location information (order of appearance in the stream), and identifies literals and numbers using logic that recognizes specific patterns of the non-token strings. Once the parser is done, control returns to the lexical analyzer. The lexical analyzer is designed to recognize specific patterns of the tokens and non-tokens. Once valid commands are identified, control is then passed to the execution code for each command. The MySQL parser and lexical analyzer was modified to include the new tokens (keywords) for the SQL_{ALV} commands and the lexical analyzer was modified to identify and process the new SQL_{ALV} commands.

www.manaraa.com

Sample SQL_{ALV} Commands
`SELECT ALV * FROM myTable WHERE ALV_KEY = 'Tower1';`
`CREATE ALV TABLE abc KEY xyz INTEGER ATTRIBUTE (a VARCHAR(10), b INTEGER INDEXED, c VARCHAR(100)), ATTRIBUTE (d VARCHAR(10), e INTEGER, f VARCHAR(100));`
`DROP ALV MyTable;`
`INSERT ALV INTO abc FOR 123 (a,b,c) VALUES ('a','b','c');`
`DELETE ALV flow FROM myTable where ALV_KEY = 90125;`
`UPDATE ALV abc FOR flow SET b = 1, c = 3 WHERE ALV_KEY = 90125;`
`SHOW ALV DATABASES;`
`SHOW ALV TABLES;`

Table 3-1: Sample SQL_{ALV} Commands

The ALV query processor represents an application of advanced query optimization techniques that translate SQL_{ALV} queries into executable form. A close examination of the MySQL query optimizer shows that its implementation is tied too closely to the internal query representation of MySQL to permit modification without a major redesign [MySQ05]. The MySQL structure is a class-based structure that has numerous lists that hold the parts of an SQL statement; many of these lists are lists of base classes that are overridden by derivative classes. Although iterators are provided to ease the complexity, the query optimizer and the execution engine are written to exploit these lists and the class-based hierarchies, making this portion of the source code in MySQL the most complex[13] and the most difficult to modify. Furthermore, the internal representation does not provide a mechanism that is viable for the query optimizer for version queries.

Rather than modify the mechanisms in place, a new query processor specifically for the ALV data store and SQL_{ALV} commands was created. This permitted not only an

[13] Called the "genius code factor" whereby the code is unintelligible to all save the author.

www.manaraa.com

opportunity to explore advanced implementation of query optimization theory, but it also permitted the core of the ALV technology to execute without modification of the MySQL internal operation. This gives the additional security that the native MySQL core executable code will not be affected by the addition of the ALV technologies. This added benefit helped mitigate some of the risk of modifying an existing system as seen by the project sponsors. A comparison of the ALV query processor and native MySQL query processor is presented in Chapter 6.

The implementation of the parser allowed the code that directs control to specific instances of the execution sub-processes[14] to be written to redirect processing to the ALV query processor. The first step in that process is to convert the SQL_{ALV} commands into an internal representation. The internal representation chosen is called a query tree, where each node contains an atomic relational operation (select, project, join, etc.) and the links represent data flow. For example, a join operation is represented as a node with two children. As data is presented from each child, the join operation is able to process that data and pass the results to the next node up in the tree (its parent). Each node can have zero, one, or two children and has exactly one parent.

The query tree was chosen because it permits the query optimizer to use tree manipulation algorithms. That is, optimization uses the tree structure and tree manipulation algorithms to arrange nodes in the tree in a more efficient execution order. Furthermore, execution of the optimized query is accomplished by traversing the tree to the leaf nodes, performing the operation as specified by the node and passing information

[14] Implemented as a huge SELECT -CASE statement in sql_parser.cc.

المنارة للاستشارات
www.manaraa.com

back up the links. This technique also made possible execution in a pipeline fashion where data is passed from the leaf nodes to the root node one data item at a time.

Traversing the tree down to a leaf for one data item and returning it back up the tree (pulsing) permits each node to process one data item, returning one row at a time in the result set. This pulsing, or polling, of the tree permits the execution of the pipeline. The result is a faster initial return of query results and a perceived faster transmission time of the query results to the client. Witnessing the query results returning more quickly – although not all at once – gives the user the perception of faster queries. Chapter 6 contains a complete explanation of the query tree and optimizer algorithms.

3.4 Advanced Index Mechanism

The storage mechanism previously discussed would be merely practical without the aid of an advanced indexing mechanism. That is, accessing data using the storage mechanism provides faster access to data using sequential access methods, but does not provide a random access mechanism – an index. This indexing mechanism is vital to the success of the versioning technology being created. Without a means of quickly accessing the attribute versions, the solution would be no more efficient than any other versioning strategy. It is this area that will provide the most advancement to the computer science discipline.

The indexing mechanism is designed using the B+ Tree algorithm and structure [Rao00]. A B+ tree is a balanced multi-way tree that stores pairs of index keys and block

المنارة للاستشارات
www.manaraa.com

addresses in the nodes[15][Jann95]. However, unlike most applications that utilize B+ Trees that apply an external buffer management subsystem (if at all) and provide concurrency access using semaphores or mechanisms provided by the operating system or another application, the new algorithm and structures will implement the B+ Tree as a buffered balanced search tree that supports concurrency [Bato81]. That is, the internal nodes of the tree will have constructs designed to support these mechanisms natively. This is possible because each node of the tree is stored in a single block on disk. The application of a buffer provides faster access to the blocks in the tree by maintaining a cache of frequently accessed tree nodes. Chapter 5 provides additional details of the indexing mechanism and its implementation.

Research on query language and query optimization has also revealed a need for another variant of this tree mechanism that supports multiple values for the data reference pointers within the leaves [Lank91, Mcke01, Mond85]. This is necessary because existing B+ trees do not provide a means of having multiple block addresses (attribute versions) for each key. The variant provides the ability to index the versions so that queries can identify all of the host records that contain versions of a given set of attribute versions and their values (one tree per attribute value). In addition, a multivalued B+ Tree is necessary to support the indexing of the file in such a way as to discover all of the records that contain a given set of attribute values. This query will be used to quickly identify the data that is versioned. Likewise, it also necessary to provide an index that

[15] Actually, there are many variants of B+ tree s. This work uses the B+ tree variant that maintains block addresses only on the leaves and links the leaves together, permitting range and sub -range iteration without repeated traversals of the tree.

المنارة للاستشارات
www.manaraa.com

discovers all of the records that have a given set of attribute versions. Without this variant, the queries described above would require a complete traversal of the index followed by (or in conjunction with) a walk of the entire file. The variant would eliminate numerous block accesses and thereby increase the efficiency of the storage mechanism.

The traversal of the tree is implemented using the traditional mechanisms for B+ Trees. Since each node of the tree will have an ordered array pointing to multiple child nodes, the search for a particular value within a node will use a linear search algorithm to locate the value quickly.

3.5 Data Mining Algorithm Applications

Data mining is a term used and overused in industry and to a lesser extent in scientific research. The term data mining has taken on a number of definitions depending on your point of view. Some consider data mining a step in a larger context of knowledge discovery while others generalize data mining to be the extraction of knowledge. This work has chosen the later approach. In this work, data mining is the process of discovering knowledge from data using algorithms designed to identify associations and categorizations among the data [Han01]. Some texts associate data mining with machine learning [Witt05]. However, machine learning is the application of programming methods that permit the improvement of software systems by analyzing examples of desired behavior rather than by direct programming [Rals03]. Thus, the application of data mining algorithms may lead to machine learning, but it is incorrect to conclude that

المنارة للاستشارات
www.manaraa.com

data mining is machine learning. In this work, data mining is viewed as a method by which one can enable machine learning.

An application of machine learning algorithms is finding interesting correlations within the data in databases. Algorithms in this area specialize in statistical analysis of results to form patterns or discover relationships among the data [Dynh03]. The learning part of this process is the knowledge discovery that the patterns provide. This knowledge, if acquired using sound statistical models, represents information that may not normally be ascertained using normal query operations. For example, suppose a large database of financial transactions was queried to discover the highest selling items. This is clearly easily obtained using normal query mechanisms. However, consider a query that can identify buying patterns among goods by certain classification of customers. This information could be obtained using a series of queries, but the time necessary to analyze the data and form the patterns through results of the queries is too great. If a clustering algorithm is run against the data, these sales patterns could be identified much more quickly with a high degree of confidence.

The last research area of this dissertation is the application of new data-mining algorithms for knowledge discovery. The goals of the data-mining algorithms for this project include those algorithms that classify, categorize, and predict values and correlations among the attribute versions. The Achilles heel of most data-mining algorithms is the static nature by which the algorithms operate. Most operate only on data that remains constant. They fail to adapt or scale well (if at all) to changes in the data, producing results with less predictability when the data is changed. An algorithm that can

المنارة للاستشارات
www.manaraa.com

scale and adapt would be the best for the types of analysis that this project will support [Dynh03].

The existing algorithms can be categorized into the following categories [Dynh03]:

- Classification – maps data into predefined groups or classes
- Regression – maps a data item to a real valued prediction
- Time Series Analysis – analyses trends in data over time
- Prediction – a form of classification that maps data into groups based on what may occur in the future
- Clustering – maps data into groups based on values in the data
- Summarization – maps of data into subsets and provides summary information
- Link Analysis – discovers relationships in the data
- Sequence Discovery – identifies patterns in the data

These algorithms are designed to operate on data presented in a fixed structure (cube) that is an implementation of a relational database table. None of the algorithms accept data dimensioned by versions of the attributes. If you consider the cube structure as a three dimensional table with rows, columns, and planes, the ALV data is an additional dimension or extension of each of the planes. The algorithms introduced by this work provide summarization and clustering of the ALV version data.

المنارة للاستشارات

www.manaraa.com

It is hypothesized that the existing realm of algorithms, with some modification, may be sufficient to meet the goals of discovering predictable trends and non-intuitive associations within the data. Another hypothesis suggests the common data structure used in data-mining, the online analytical processing cube, may not adequately represent the versioned data. A new variant or alternative structure may be necessary to store the data for the data-mining algorithm to process. Another hypothesis suggests the structure and semantics of the Web Ontology Language (OWL[16]) [Anto04] may be a viable alternative to the cube structure. If this alternative is viable, it could lead to more discoveries in the effectiveness of data-mining algorithms and their tolerance to changes in the data. The most difficult aspects of this portion of the project will be forming the data into structures that can be consumed by the data-mining algorithms. Research in this area crosses the boundaries of artificial intelligence, machine learning, information retrieval, statistical analysis, and knowledge management.

3.6 Application of Emerging Technologies in Conjunction with ALV

Much of what makes the ALV concept unique is the storage and retrieval of versioned data. However, the research for this project does not address the question of how data is identified as an alternative to existing values. In other words, how do we know there are duplicates or that these duplicates should be considered versions? Furthermore, why should we care?

[16] Many people wonder how OWL got its name. Some are surprised that the Web Ontology Language is nicknamed OWL in honor of Owl in Winnie the Pooh (milne, 1996), who spells his name "WOL."

www.manaraa.com

Recently, manual database coordination was successful in solving the Washington sniper case. Jonathon Alter of *Newsweek* described the success as: “It was by matching a print found on a gun catalog at a crime scene in Montgomery, Ala., to one in an INS database in Washington state that the Feds cracked open the case and paved the way for the arrest of the two suspected snipers. Even more dots were available, but [they] didn’t get connected until it was too late, like the records of the sniper’s traffic violations in the first days of the spree,” [Alte02].

If law enforcement agencies had the ability to assimilate and deconflict data among all of the databases they referenced and to place that data into a versioning system, they might have been able to draw inferences as to how seemingly independent events[17] were in fact related by a common thread – the snipers.

How would one track such data? The real utilitarian power of ALV is in the ability to associate metadata with each attribute version. This allows researchers to record pertinent data with each attribute version such as time of event, duration, source of the information[18], and other important facts. These metadata – facts about facts – give ALV the ability to become a powerful tool in data compilation and exploitation.

The most intriguing use of ALV in emerging technologies is in the inference of semantic data. Metadata is a starting point for semantic representation and processing. The rise of metadata is related to the ability to reuse metadata between organizations and systems. Likewise, the ability to exchange this metadata (data in general) between

[17] Events stored in database systems without context of the actors and related events relegates them to discrete points in time. The ability to link those events and actors through a flow of data using versioning is the goal of ALV.
[18] Also known as the originator.

www.manaraa.com

organizations has enabled communication and data sharing between legacy systems through a common format. The most popular common format is XML [Fens03]. The success of XML can be attributed to the following:

- XML creates application-independent documents and data.
- It has a standard syntax for metadata.
- It has a standard structure for both documents and data.
- XML is not a new technology.

Metadata alone cannot solve problems like the D.C. sniper case described above. The accumulation and processing of metadata alone cannot be used to draw inferences from the data. Additional technologies are needed to apply meaning (semantics) to the data.

One of the many goals of semantic web technology is the mapping of data to relationships and from there making inferences. This is accomplished by forming ontologies of the data. An ontology models the vocabulary and meaning of domains of interest: the objects (things) in domains; the relationships among those things; the properties, functions, and processes involving those things; and constraints on and rules about those things. In broadest terms, an ontology is; **1 :** a branch of metaphysics concerned with the nature and relations of being, **2 :** a particular theory about the nature of being or the kinds of existents.

www.manaraa.com

There have been breakthroughs that have led to a flurry of implementation and product jockeying by industry. Ontology is one of those breakthroughs. Unfortunately, most information technology experts are bombarded with marketing hype that obscures the finer details of the technology. For example, a distinction is made between "big O" Ontology and "little o" ontology. "Big O" Ontology is the philosophical discipline. "Little o" ontology is the information technology engineering discipline that has emerged over the past eight or so years. The application of ontology in computer science today is grounded in the "little o" ontology. More precisely, the application of ontology defines the common words and concepts (the meaning) used to describe and represent an area of knowledge. Similarly, the result of the application (creation) of an ontology is an engineering product consisting of "a specific vocabulary used to describe [a part of] reality, plus a set of explicit assumptions regarding the intended meaning of that vocabulary" – in other words, the specification of a conceptualization. The recent computational discipline that addresses the development and management of ontologies is called ontological engineering [Anto04].

Ontologies form a model of the data that includes not only relationships, but also behavior. This data model requires three levels of representation. First, the knowledge representation level must provide a way to represent the data – this is where the ALV technology will be used. Second, the ontology concept level is where ontologies and ontology hierarchies are formed. A related area of study is known as metaontologies or ontologies of ontologies [Grun02]. Third, the ontology instance level is necessary to store

المنارة للاستشارات

www.manaraa.com

the ontology for processing by inference engines that operate on the data. The repository of choice for this level is a graph store.

Once a model is created for the data, the model can be exploited to infer meaning among the data. Modeling the data allows for quick adaptation to changing data, relationships, and even intent of the model. It is much simpler to change the model (the description) than a thing that, without the model, has no well-defined semantics. Without a model, you are perpetually doomed to try to correlate tuples in multiple databases that have no accompanying semantics. This is why data mining and its parent, knowledge discovery, are such hot technologies now – this is the way we usually do things. No model, no semantics. So we try to infer the semantics, or what the data means. This is a very difficult problem – one that isn't easily solved by simply writing code. Research into the application of semantic methods combined with advanced storage mechanisms and inference engines using data-mining algorithms is a reasonable start to solving the problem.

The Semantic Web technologies have proven to be a viable alternative to creating a custom application for the data assimilation and deconfliction needs of the ALV project [Anto04, Magk02]. Ontologies may prove to be a medium to support the data-mining algorithms. These and other emerging technologies are explored and reported in this work. Supporting research is presented and represents a context in which the ALV system will be used.

المنارة للاستشارات

www.manaraa.com

3.7 Conclusion

The body of knowledge that is computer science has evolved into a distinct discipline characterized by three basic paradigms; theory, abstraction, and design, with fundamental roots in both mathematics and engineering. Functionally, computer science is the systematic study of algorithmic processes concentrating on the theory, analysis, design, efficiency, implementation, and application of systems that describe and transform information [Tuck04].

The central sub-discipline of computer science that forms the foundation for this work is concentrated on is database theory. This work explores topics in database theory such as physical design of structured data stores, query processing, query optimization, indexing mechanisms, concurrency, and disk buffering. The successful implementation of the project demonstrates application of each of these areas. This work examines the implementation with sufficient academic rigor to illustrate the benefits and applicability of the project to support these areas of database theory.

The newest sub-discipline of computer science applied to this work is in the area of classification and categorization algorithms, specifically referred to as data mining. The algorithms designed to enable knowledge discovery are grounded in sound statistical practices[19]. These algorithms require processing of large portions of data as well as the need to process the calculations quickly. Considerable effort was undertaken to ensure

[19] Some texts on data mining go so far as to say data mining is an application of statistical theory. Regardless of one's point of view, it is clear data mining algorithms must exhibit the tenets of sound statistical practices.

المنارة للاستشارات
www.manaraa.com

adherence to the foundations of data mining from the viewpoint of computer science with the goal of maintaining a sound statistics-based application.

The goal of the research in this work is to abstract the salient features of the identified computational tasks in order to permit the development of the versioning capability in a setting unencumbered by insignificant details. The motivation for the solutions examined may have arisen from an application or from purely foundational considerations. In the former case, new abstract models are introduced that will be useful in designing and analyzing algorithms in the project. In the latter case, practical applications of emerging technology are introduced as a by-product of the solutions. Utilization of the body of computational theory enabled the theoretical portions of this work as been guided by the historical precedence of past work, the present-day realities of computing, and helps to ensure the relevance of the work to the future practice of computing.

3.8 To be continued…

This work is intended to contribute to the body of knowledge that is computer science. These chapters comprise the bulk of the knowledge learned, technologies created, and theory advancements of this work. The general format of each chapter is as follows:

- **Abstract** – a short description of the subject matter discussed.
- **Introduction** – an introduction and background information for the technology being developed or explored.

المنارة للاستشارات
www.manaraa.com

- **Technology Description** – one or more sections for each area of the theory and implementation researched.
- **Analysis** – a description of alternatives, comparisons and experiment results are presented.
- **Conclusion** – the overall conclusion of the work as it relates to the advancement of science is presented.
- **Future Work** – a short description of areas of exploration available for expansion of the work presented.

Chapter 8 will conclude this work, its research, and supporting project. A presentation of the avenues for future work and future exploration of implementation of the ALV project and its technologies will conclude this work.

المنارة للاستشارات
www.manaraa.com

Chapter Four - A Clustered Storage Mechanism for Versioning

Abstract

Clustering or co-locating data that is retrieved as a set increases performance of the retrieval system [Tuck04]. This premise motivates the development of a clustered storage mechanism that stores and retrieves versioned information. This chapter shows one implementation of a clustered version store and reports its performance as compared to a commercially available storage mechanism.

4.1 Introduction

A clustered storage mechanism can provide the foundation for a versioning system supported in a relational database system. This chapter presents a clustered version store for storing and retrieving versioned data within a versioning system. This system, called Attribute-Level Versioning (ALV), is an extension of the MySQL database management system.

The following sections present the current research on physical database design and implementation, the technology and design of the clustered version store, analysis of the performance of the mechanism, and a conclusion as to its success in meeting the goals defined above. This chapter concludes with a section outlining future work opportunities to improve the clustered version store.

المنارة للاستشارات
www.manaraa.com

4.2 Background

Physical database design has been important since the very early days of database system development. However, the practice has become less emphasized due to the effectiveness and simplicity of common file systems supported by operating systems. Today, physical database design is merely the application of file storage and indexing best practices[1].

There are set clear goals that must be satisfied to minimize the I/O costs in a database system. These include utilizing disk data structures that permit efficient retrieval of only the relevant data through effective access paths, and organizing data on disk such that the I/O cost for retrieving relevant data is minimized. Both of these goals are addressed in physical database design [Grae93, Marc83]. An overriding performance objective is thus to minimize the number of disk accesses (or disk I/O's) [Date04a].

There are many techniques and opinions on how to approach database design. Fewer exist for actual physical implementation. Furthermore, many researchers agree that the optimal database design (from the physical point of view) is not achievable in general and furthermore should not be pursued. This is mainly due to the much improved efficiency of modern disk subsystems. Rather, it is the knowledge of these techniques and research that permit the database implementer to implement his database system in the best manner possible to satisfy the needs of those that will use the system [Seve77].

In order to create a structure that performs well, many factors must be considered. Early researchers considered segmenting the data into subsets based on content or context

[1] Such as separating the index file from the data file placing each on a separate disk I/O system to increase performance.

المنارة للاستشارات
www.manaraa.com

of the data. These subsets are organized in similar groups as that of the physical database design. For example, all data containing the same department number would be grouped together and stored with references to the related data. This process can be perpetuated in that sets can be grouped together to form supersets, thus forming a hierarchical file organization [Senk73].

Accessing data in this configuration involves scanning the sets at the highest level to access and scan only those sets that are necessary to obtain the desired information. This process significantly reduces the number of elements to be scanned. The researchers also desired to keep the data items to be scanned close together to minimize search time. Researchers call the arrangement of data on disk into files and the structured storage of those files as file organization. The goal was to design an access method that provided a way of immediately processing transactions one by one, thereby allowing us to keep an up-to-the-second stored picture of the real-world situation [Sek73].

File organization techniques were revised as operating systems evolved in order to ensure greater efficiency of storage and retrieval. Modern database systems create new challenges for which currently accepted methods may be inadequate. This is especially true for systems that execute on hardware with increased disk speeds with high data throughput. Additionally, understanding database design approaches, not only as they are described in textbooks, but also in practice will increase the requirements levied against database systems and thus increase the drive for further research [Hain03]. For example, the recent adoption of redundant and distributed systems by industry has given rise to

المنارة للاستشارات
www.manaraa.com

additional research in these areas in order to make use of new hardware and/or the need to increase data availability, security, and recovery.

Since accessing data from disk is very expensive, research has shown that the use of a cache mechanism, sometimes called a buffer, can significantly improve read performance from disk thus reducing the cost of storage and retrieval of data. The concept involves copying parts of the data either in anticipation of the next disk read or based on an algorithm designed to keep the most frequently used data in memory. The handling of the differences between disk and main memory effectively is at the heart of a good quality database system. The tradeoff between the database system using disk or using main memory should be understood. See table 4-1 for a summary of the performance tradeoffs between physical storage (disk) and secondary storage (memory).

Issue	Main Memory VS Disk
Speed	Main memory is at least 1000 times faster than Disk
Storage Space	Disk can hold hundreds of times more information than memory for the same cost
Persistence	When the power is switched off, Disk keeps the data, main memory forgets everything
Access Time	Main memory starts sending data in nanoseconds, while disk takes milliseconds
Block Size	Main memory can be accessed 1 word at a time, Disk 1 block at a time

Table 4-1: Performance Tradeoffs

Advances in database physical storage have seen much of the same improvements with regard to storage strategies and buffering mechanism, but little in the way of exploratory examination of the fundamental elements of physical storage has occurred. Some have explored the topic from a hardware level and others from a more pragmatic level of what exactly it is we need to store. Stonebraker in his paper on multi-level

www.manaraa.com

storage examines the possibility that systems must store data not only on disk and in memory for fast retrieval but also to/from tertiary storage such as magnetic tape and other archival systems [Ston94a]. The subject of persistent storage has been largely forgotten due to the capable and efficient mechanisms available in the host operating system.

4.2.1 File Organization Techniques

Beneath the layers of physical database design lies the file organization layer. This layer has perhaps the most effect on the performance and efficiency of data retrieval. Indeed, this layer is responsible for many of the performance challenges related to database system implementation. There have been many techniques which differ greatly in their sophistication and implementation benefits. Depending on the nature of the data operations (insert, delete, update), retrieval (sequential or random), size of the data retrieved, etc., some techniques perform better than others. Performance issues can be compounded by the wide variety of file organization techniques and their idiosyncrasies and implementations. Designers of database system are often faced with making choices in file organization that deal with tradeoffs and compromises. This can be made all the more problematic when trying to determine or predict the performance of particular structures [Yao76].

There has been research into modeling file organizations in order to gain understanding of the function and performance of file organizations. There are two distinct categories of modeling, simulation and analytical. Simulation deals with repeated runs of experimental scenarios to check the operation and performance of models without

المنارة للاستشارات
www.manaraa.com

live systems. Analytical deals with repeated implement-test-measure-compare scenarios that evaluate different implementations in near live environments. Models have been evaluated using techniques such as multi-list, inverted file, and doubly-chained tree organizations. In addition, simulation models have been used to compare indexed sequential and direct access methods. Unfortunately, these simulations are based on individual file structure models and thus do not provide a common baseline for comparing performance of a set of file organizations [Yao76].

4.2.1.1 Access Methods

One of the areas to receive a great deal of attention is called access methods. This area is concerned with the identification of all necessary resources and execution mechanisms necessary to retrieve, update, and insert data. A single instance of these items to access a single datum is called an access path. The access path therefore contains all of the algorithms, cache mechanisms, and execution sequences necessary to execute the command. A large amount of research concerns the optimization of the access path. *i.e.* the minimization of resources necessary to complete the operation.

An access path is formed by user interaction with the system and the system's availability of methods by which data is retrieved and presented to the user. An access path therefore is the set of algorithms and structures that are used to store, retrieve, and update portions or sets of data [Marc83]. Research has produced a set of guidelines that should be followed to successfully optimize file organization and buffering techniques [Elma03]. These include:

المنارة للاستشارات

www.manaraa.com

- Analyze the queries and transactions to discover frequently used operations
- Analyze the expected frequency of queries and transactions
- Analyze the time constraints of queries and transactions
- Analyze the expected frequency of update operations
- Analyze the uniqueness constraints on attributes

Research has also identified properties that are used to measure the goodness of access methods [Ras03, Tuck04].

- Data should be stored in associated groups according to anticipated queries.
- Disk space should not consume more space than is necessary to ensure efficient storage and retrieval of data.
- Data accessed should not return unnecessary or irrelevant data.
- Inserts and deletes should modify no more than one page at a time in order to avoid blocking reads of other data in the block (inserts and deletes should not prohibit reading of unrelated data).

The application of these techniques will ensure that the access methods employed will be designed with efficiency and optimization.

المنارة للاستشارات
www.manaraa.com

4.2.1.2 Extended Blocks

Associating data that has group membership characteristics can be problematic. When data is added to a database, particularly when data is added to physical storage that already has data with which the new data can be associated, it is not always possible to physically locate the data together. In fact, in a high use environment where many reads and writes as well as updates occur, data can often become fragmented on the physical media.

Although there are offline techniques[2] available for reorganizing the physical store and restoring most of the benefits of collocating data, it is important to build into the system the ability to leave blank spaces for inserting data, thereby allowing reorganization the physical store less often.

One method of ensuring available space for data is using additional pages as overflow space for data that does not fit on a single page. A pointer is used to record the next page in the file that contains the rest of the data. This method is sometimes referred as spanning [Elma03]. In ALV, a similar mechanism is used utilizing two pointers forming a doubly-linked list, so that data can be access both forward and in reverse. This mechanism is referred to as extending the page. The additional pages are called extents of the first page.

---

[2] In this case, offli ne means that the database must be taken out of use and the reorganization performed in isolation.

المنارة للاستشارات
www.manaraa.com

4.2.1.3 Free Blocks

A technique related closely to extended blocks (spanning) is the intentional retention of blocks in the physical store after deletions. That is, when all of the data in a block is deleted, the block remains allocated in the physical store and is marked as empty or available. A list of available blocks is called a free block list [Rama03]. The maintenance of available blocks in the physical store is typically very practical and usually involves the application of a simple data structure such as a list or queue. ALV uses a queue stored in the file header that manages the free block list. A queue permits the free blocks to be used in the order they were released thus permitting the free space to be recovered before appending new blocks to the end of the file.

4.2.1.4 A Comment about Data Independence

C.J. Date defined data independence as "immunity of applications to change in storage structure and access strategy." Modern database systems offer data independence by providing a high-level user interface through which users can interact with the data using concrete logical organizations (e.g., rows, tables, databases), rather than through variables, pointers, arrays, lists, etc. which are used to store the data internally in the database system. Thus, the database system is responsible for choosing an appropriate internal representation for the data which can change without affecting the users and their use of the data [Cham81a]. This internal representation is the subject of this work; indeed, the very focus of the ALV clustered version store.

المنارة للاستشارات
www.manaraa.com

Physical data independence is an abstraction concept in which the portions of the system that process data for query processing do not rely on specific mechanisms of how the data is stored on the physical media. There are three essential capabilities that database systems must provide in order to achieve physical data independence. First, the system must permit the use of alternative representation of data. Second, queries resulting in changes to the data must map correctly to all relevant data on the physical storage media. Third, if the physical design permits multiple access paths, the results of traversing all access paths must result in the same data being accessed, *i.e.* the same query results [Grae93].

Thus, the design of the query processing and optimization engines must clearly take the physical database design options of the underlying database management system into account (e.g., the concept of "interesting orderings" in System R [Cham81a] and many other systems), and a physical database design tool must consider the capabilities of both the database system's file and index level as well as its query optimizer [Grae93].

4.2.1.5 Buffer Manager

Data stored in a database system is subjected to a number of operations including insert, update, delete, and retrieve. These operations are executed in the form of a query against a target database. Every query therefore accesses at least one data item. Often these queries are small and execute quickly. However, due to the relatively high cost associated with moving the data from physical storage to memory for manipulation, most database systems must use mechanisms to minimize the number of I/O operations. In a

المنارة للاستشارات
www.manaraa.com

similar way, care must be taken to maximize the use of memory, making the optimization of the page replacement algorithm very important for the overall performance of the database system [Effe84].

Computing systems that access large amounts of data often employ a cache to store the data in memory prior to use. Given that the overhead for this process is minimal compared to the savings of accessing data from memory rather than disk, the cache can improve performance significantly. For most systems, the cost of disk access is 100 or more times than that of memory access. In systems where the cache is implemented as virtual memory instead of real memory, an increase in the size of the buffer space may cause a decrease in performance due to increased competition for real memory between the program and the buffer [Sher76].

This cache area is often called a buffer. The buffer is organized as discrete pages with each page containing a page from a file. If a page requested is not found in the buffer, a signal is generated called a page fault. The buffer manager then reads the page from disk and stores it in the next available empty page in the buffer (demand paging). When all available pages are used, a page is made available by a replacement policy. If the data has been changed since it was placed in the buffer, it is written back to the disk.

The buffer manager is very similar to the manager of virtual memory in operating systems [Maek87]. In fact, some researchers have suggested that memory mapped files can be used as a specialized form of a buffer. Using the operating system facilities for memory mapping like that of virtual memory could enable a more effective buffer access. Unfortunately, this concept is not a good fit with database systems because it is based on

المنارة للاستشارات

www.manaraa.com

the assumption of a relatively small process space, while it is not uncommon for data access in a database system to be very large [Sacc86].

A common mechanism used in some database systems to reduce the number and frequency of I/O operations is to maintain an internal memory buffer pool of the blocks most frequently used in the database. Access requests to this data are satisfied by searching the buffer for the data rather than accessing the physical media. Changes are written back to the physical media in a variety of ways – either a scheduled dump or a manual operation via an operator-initiated command. Examples of systems that have used this technique include Information Management System/360 (IMS) by IBM, and INGRES. Interestingly, INGRES uses the virtual memory pool of the host operating system (Unix) to satisfy this need.

A buffer manager uses an in-memory cache to store data for faster access. Data in a physical store does not normally fit in the available memory space. Thus, the buffer manager is responsible for ensuring the pages needed are in memory when accessed. When there are multiple files being accessed and all of the pages in the cache are used, the buffer manager must decide which pages in the cache are no longer needed and replace them with the requested pages. This technique is called a buffer replacement policy. One of the most effective replacement policies is based on the time the page is in memory. This mechanism is called least recently used (LRU) [Elma03, Falo95].

The process of loading pages from the physical store into memory has been implemented in a variety of ways and largely becomes a matter of preference or choice for database system implementers. The most frequently used method uses record ids that

المنارة للاستشارات
www.manaraa.com

map the pages in the internal memory buffer to that of the physical reference ids in the physical store. Using record ids for accessing data in files can greatly enhance the implementation and performance of buffer management systems that equate disk blocks with pages in memory [Date04a].

The most important aspect of buffer management implementation is the fact that it must be designed to optimize short-term data requests of higher level subsystems in such a way as to ensure the blocks of data stay in memory until written back to disk and flushed from the buffer [Elha84]. The lifetime of the data in memory is the time spent in the committed buffer pool until no longer needed as defined by the page replacement algorithm.

The most popular replacement policy is least recently used (LRU), which replaces the page that has not been referenced for the longest time. LRU belongs to a family of algorithms called stack algorithms. One premise of the LRU algorithm is that an increase in available buffer space reduces the likelihood of an increase in the page fault rate. More importantly, the LRU strategy is well understood and simple enough to be implemented in any buffer manager. This is especially important as the buffer manager is one of the most heavily used system components [Effe84, Sacc86].

Since physical references are expensive, the optimization of the page replacement algorithm is very important for the overall performance of the system. Optimization in this case is concerned with the minimization of the number of physical disk accesses for a typical transaction load, described by a logical reference string (to include the access path) [Effe84].

المنارة للاستشارات
www.manaraa.com

There has been a great deal of research conducted on buffer management strategies. These include algorithms such as Clock, GClock, LRD, 2Q, and variants of LRU such as LRU-K. Clock and GClock simulate a LRU behavior. Least Recent Density (LRD), a derivative of GClock, calculates elapsed time references in order to keep pages more likely to be needed in the buffer. Frequency-based ensures pages that are used frequently are saved in memory. LRU-K selects a page for replacement based on time. The 2Q strategy improves the performance of LRU-K by using multiple queues [Feng98, Onei93]. LRU and Clock indicate a satisfactory overall behavior under a variety of conditions [Effe84]. The LRU model of replacement page access was very successful in increasing performance in a buffer manager or cache mechanism [Effe84, Maek87].

While many studies on database buffer management focused on various paging problems, more recent efforts have focused on finding buffer management policies that "understand" the database access patterns. Such algorithms include "New", DBMIN, Working Set (WS), and Hot Set [Chou94]. It has been shown that transactions generally have a high degree of locality [Effe84]. That is, transactions often encompass commands that operate on data that is in close proximity. This permits the inclusion of transaction algorithms in buffering mechanisms.

There have been many explorations into building a buffer management strategy that anticipates the behavior of the database system. The Weighing/Waiting Room (W2R) [Jeon98] is an example that prefetches the next block in the pointer chain (also called a one block ahead strategy) and partitions the buffer into two rooms – a weighing room that

المنارة للاستشارات
www.manaraa.com

maintains the currently requested pages and the waiting room that maintains a list of next pages in the prefetch.

There has been a considerable effort to optimize the operation of the buffer pool. Early works concentrated on the selection of blocks to move into the buffer pool. Likewise, the selection of blocks to remove from the buffer pool has also been extensively explored. One such effort created an algorithm for the selection of blocks for the buffer pool called prefetching.

The prefetching of data blocks into database system buffer pool is very similar to the prefetching of program execution statements in a virtual memory system. Simple sequential prefetching of pages has generally been found to be ineffective, but more sophisticated methods which either analyze the program in advance, accept user advice, or maintain relevant statistics during program execution can significantly improve system operation. Sequential prefetching of lines for cache memory has been shown to work very well because the amount of data arranged in sequence is large compared to the cache page (line) size; for most programs the amount of data arranged in sequence is not large compared to the main memory page size [Smit78].

Considered by many to be the most important property of reference strings, metadata that stores the access path, within page replacement algorithms is the locality of the reference behavior [Effe84]. Locality in this case is used for frequency counting of page hits and page faults. The goal is to watch for blocks that generate higher localities

المنارة للاستشارات
www.manaraa.com

(more frequently accessed) and store the frequency in the reference string[3]. This data is then used by the page replacement and retrieval algorithms to optimize the blocks in the buffer and thereby optimize access to frequently used data. If locality is observed in a reference string, most of the virtual memory allocation and replacement algorithms can be applied to buffer management; these algorithms were designed to keep the most recently referenced pages in main memory, since programs executing under virtual memory operating systems show high locality in their reference behavior [Effe84].

Research has shown that the operating system virtual memory algorithm and the database system buffer management algorithm are affected by each other, but it isn't clear how to find a balance between the two and the performance is dependent on the types of each algorithm. Some pairs complement the behavior of the other and others limit the performance of the system as a whole [Kim88].

It has also been pointed out that in multitasking environments information from the query processor/optimizer may not be appropriate for performance enhancement [Jeon98]. Effelsberg and Haerder suggest database systems running in virtual memory operating systems should use a table search technique to reduce page faults [Effe84].

The ALV architecture is modeled after the architecture of PostgreSQL. Although PostgreSQL is an ORDB database system, the structure of the system is largely the reason for its success. Like PostgreSQL, ALV is divided into several modules that each provide an abstraction of the fundamental elements of a database system [Dong04]. The

[3] Also called a working set by Denning and Randell in their work on modeling and controlling program behavior [Denn78, Rodr73].

المنارة للاستشارات

www.manaraa.com

ALV buffer manager is a rudimentary implementation that does not take advantage of the access patterns.

4.2.1.6 Shadow Paging

One technique shown in the research that is unique enough to warrant special mention is the concept of shadow paging. With shadow paging, transaction logs do not hold the attributes being changed but a copy of the whole disk block holding the data being changed. This sounds expensive, but actually is highly efficient. When a transaction begins, any changes to disk follow the following procedure:

1. If the disk block to be changed has been copied to the log already, jump to 3.
2. Copy the disk block to the transaction log.
3. Write the change to disk.

On a commit the copy of the disk block in the log can be erased. On an abort all the blocks in the log are copied back to their old locations. As disk access is based on disk blocks, this process is fast and simple. Most database system systems use a transaction mechanism based on shadow paging [Elha84].

Shadow paging is relatively uncomplicated to implement. The most sophisticated portion of this technology is the log. Log entries can be implemented and stored in cache memory any time before query operation completes. This has an added benefit in that log pages (a block of log entries that fit into a block on disk) can be written to disk any time

www.manaraa.com

before the transaction commits. Modified data blocks can be written to disk at any time provided a manual operation to dump, or checkpoint[4], has not occurred. If a system fails before a checkpoint, the system can refer to the last known good checkpoint and recover data from that point[5]. This added benefit is perhaps the greatest advantage that shadow paging offers.

However, there are also some negative aspects of shadow paging. Sequential processing such as a table scan can be detrimental and may lead to inefficiencies if the data is not stored in the same order as the file blocks on disk. In this case, the next data item may not be in the shadow cache and thus defeats the read-ahead algorithm. Since data is likely to exist in several places throughout the physical media, shadow paging may not be fetching the needed blocks from disk and may result in unnecessary transfers. On the other hand, if the data blocks are reasonably co-located, shadow paging will yield good performance. Various solutions to this dilemma have been considered [Trai82].

Unfortunately, shadow paging is not suitable for implementation in ALV. This is especially true considering the high degree of locality built into the concept of the versioning system. That is, all versions of all attributes for a given entity should be co-located while permitting the iteration of all a set of attribute values for a given set of entities. However, the blocks that contain the attribute version data are not guaranteed to be stored sequentially. Thus, a paging algorithm that reads blocks ahead of the current

[4] Called a 'checkpoint' becau se it forces the system to check that no pages are left unwritten. It provides a stable state for the application of recovery algorithms.
[5] The responsibility for ensuring good checkpoint logs is the responsibility of the database professional.

المنارة للاستشارات
www.manaraa.com

block is not guaranteed to have the next block in the set of blocks containing the attribute version data in cache. This clearly defeats the advantages of shadow paging.

4.2.1.7 Sparse Files

Windows NT, and derivatives, support a file concept called 'sparse files' [Silb98]. Sparse files permit the allocation of unused blocks so that the file consumes only as much space as it actually has data. The addressing mechanism is retained and for all appearances the file contains all of the space allocated, but what is actually stored are only the blocks that contain data [VanB03]. This technique can be confused with a concept called sparse file allocation. Research of physical database file implementation and buffer management strategies defines sparse file allocation as the intentional storage of empty blocks to increase the efficiency of file I/O. References to sparse files and/or sparse techniques refers to what is supported by research[6] and has nothing to do with sparse files as supported by the host operating system.

4.2.1.8 Transposed Files

An early attempt to minimize physical storage access time using file storage used a method called a transposed file. A transposed file is a collection of subfiles that are not ordered or necessarily co-located. The data being stored is divided among the subfiles based on a partitioning scheme for the attribute data. That is, each file contains only a subset of all of the attributes in the data. Thus, the data for a single entity is distributed

---

[6] Yet another example of how industry strays from academic rigor.

www.manaraa.com

among the files. Transposed files were designed to reduce unnecessary attribute (data) transfer during queries. This is analogous to the use of secondary indices of inverted files where the objective is to reduce unnecessary record transfers [Bato79].

The disadvantage comes when a projection is performed on the data that requires access to all of the subfiles in order to satisfy the query. The concept of transposed files resembles that of the horizontal implementation described in section 4.2.2.1 below. Transposed files, much like scatter storage techniques [Morr68], separate the data into compartments that can be searched much more quickly than a single file.

4.2.2 Version Store Implementations

There are many techniques that are available to create a storage mechanism for versioned data. These techniques range from utilizing common practices for database logical design (schema) to implementing an object-relational database. Two of the more common techniques are examined in the following paragraphs.

The primary users of the ALV system are interested in research. As they conduct their research, more and more data is accumulated [Isaa93]. The application of data deconfliction techniques described earlier in section 1.1 permits the identification and resolution of collisions within the data which generates versions of the data. This constant addition of versioned data means the system must be optimized for storing large amounts of attribute versions in groups that are accessible quickly. The file system therefore must be optimized to retrieve these sets of attribute versions. In this strategy, time complexity

المنارة للاستشارات

www.manaraa.com

is traded for space complexity by maintaining an index whose growth is asymptotically super-polynomial. The argument is that space is cheaper than time [Isaa93, Kuma03].

Versioned data requires a dedicated storage mechanism that optimizes retrieval of sets of versioned data. This is preferable to and more efficient than using traditional storage mechanisms to store versioned data.

One of the early ideas for the ALV physical implementation was to utilize the facilities of the database system and create a binary large object (BLOB) field for storing the version data. Unfortunately, there are several major flaws in that concept. First, BLOBs are notoriously slow and are often the cause of performance issues [Widm99]. Second, BLOBs are generally not indexed nor can they be indexed using any normal mechanism. Lastly and most importantly, using a BLOB as a multi-valued field violates several normal forms of relational theory.

4.2.2.1 Horizontal

The first choice among database professionals when faced with the requirement to store versioned data is to create a logical design that includes tables that stores the versioned data in separate tables (see figure 4-1). The horizontal name comes from the fact that the data is horizontally partitioned.

للاستشارات المنارة

www.manaraa.com

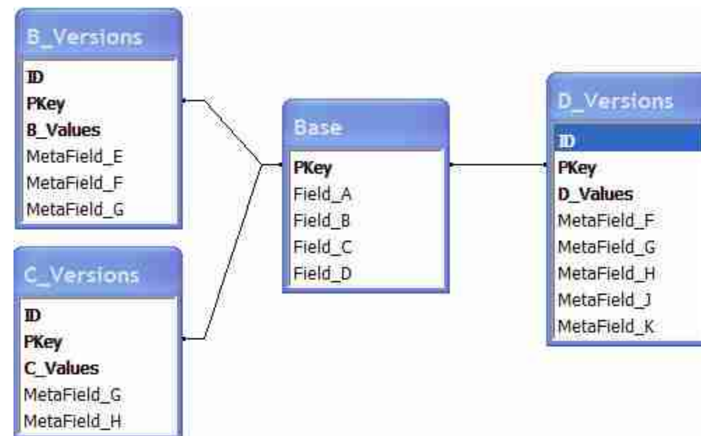


Figure 4-1: Simplified Horizontal Database Example

In this simple example, the base table 'Base' is defined to have a single primary key and four attributes. Three of the four attributes will require the storage of versioned data. Furthermore, the versioned data will have differing sets of metadata. Each version store is implemented as a separate table for each set of versioned data. Notice the need for an identity field (see section 4.1.1.4 below) in the versioned tables in order to overcome the limitation of uniqueness imparted by the relational database. This is necessary because there may exist several values of an attribute for each reference to the base table. Furthermore, there may exist more than one attribute version for a single reference to the base table that differs only in the values of the metadata – same value, but different 'pedigree'. This could have been overcome by adding the metadata fields to the primary key, but would result in a very complex and large index file for each table.

Although simplified, one can easily see that in order to retrieve all of the attribute versions for a given entry in the base table, one would require a number of simple queries to assemble the data. However, note the overlap of the metadata fields. The fields differ and therefore will require additional query commands to form a single result, e.g.,

المنارة للاستشارات
www.manaraa.com

UNION, JOIN, etc. Figure 4-2 contains an example SQL statement that retrieves all of the attribute versions for a specific entry in the base table. The results would be displayed as a single result set.

```
SELECT Base.PKey, Base.Field_A, Base.Field_B,
B_Versions.B_Values, Base.Field_C, C_Versions.C_Values,
Base.Field_D, D_Versions.D_Values, B_Versions.MetaField_E,
B_Versions.MetaField_F, B_Versions.MetaField_G,
D_Versions.MetaField_J, D_Versions.MetaField_K,
C_Versions.MetaField_H
FROM ((Base INNER JOIN B_Versions ON Base.PKey = B_Versions.PKey)
INNER JOIN C_Versions ON Base.PKey = C_Versions.PKey) INNER JOIN
D_Versions ON Base.PKey = D_Versions.PKey
WHERE (((Base.PKey)=12345));
```

Figure 4-2: Horizontal SQL Statement

Clearly, the above SQL statement isn't easy to read and may in fact be difficult to create and modify to retrieve any meaningful relationships among the metadata fields and/or the attribute versions themselves.

The major limitations of the horizontal technique are the proliferation of versioned tables and the complex relationships necessary to complete the relational ties that give a database expressive power. Despite these limitations, the horizontal technique is the most common and the most popular method of storing versioned data. It is most intuitive to database professionals because it follows the common practice for data normalization. Unfortunately, it also complicates the logical design in ways most database professionals don't immediately recognize. What is needed is a storage mechanism that will permit simplified queries that implement the relational pathways between the base table and the attribute versions thereby optimizing the retrieval of all attribute versions for a given entry in the base table.

المنارة للاستشارات
www.manaraa.com

4.2.2.2 Vertical

A less obvious choice is the creation of a logical design that stores information about classes of data. This technique permits the categorization of data and permits a more flexible versioning mechanism for data. This technique is most often used to implement an object-relational database in a purely relational database system. This practice gives the flexibility to include a concept of versioning, which is inherent in object-oriented design, while maintaining the integrity power of a relational database. A simplified example of a vertical implementation is shown in Figure 4-3. The name vertical comes from the fact that the data is partitioned into classes of attributes and then enumerations, etc. continuing to be de-referenced down to the value for that instance of the attribute version.

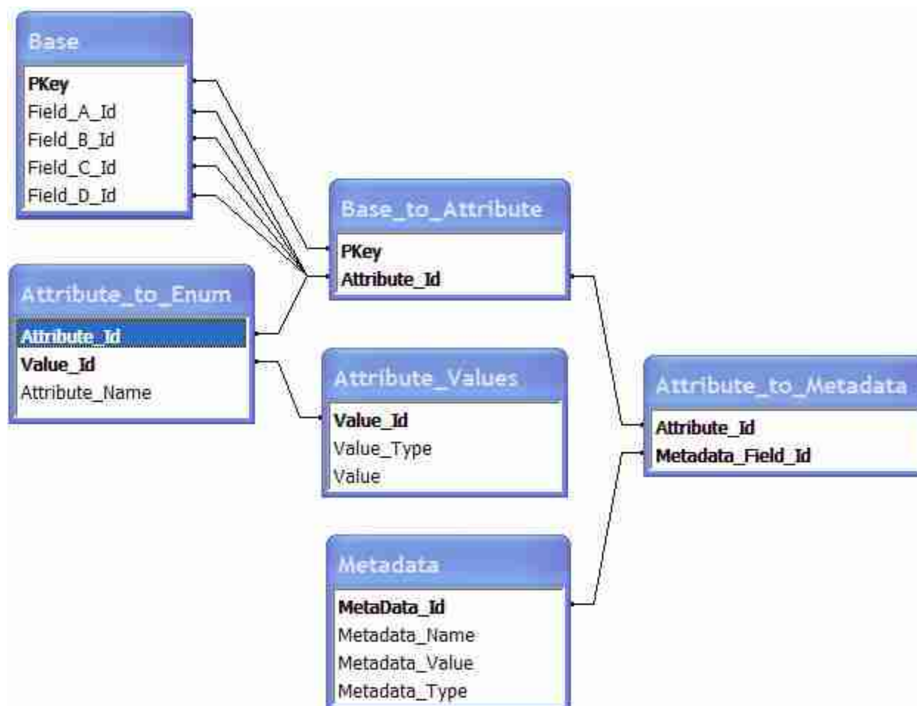


Figure 4-3: Simplified Vertical Database Example

المنارة للاستشارات
www.manaraa.com

```
SELECT Base.PKey, Attribute_to_Enum.Attribute_Name,
Attribute_Values.Value, Metadata.Metadata_Name,
Metadata.Metadata_Value
FROM Metadata INNER JOIN (Base INNER JOIN ((Base_to_Attribute
INNER JOIN (Attribute_Values INNER JOIN Attribute_to_Enum ON
Attribute_Values.Value_Id = Attribute_to_Enum.Value_Id) ON
Base_to_Attribute.Attribute_Id = Attribute_to_Enum.Attribute_Id)
INNER JOIN Attribute_to_Metadata ON
Base_to_Attribute.Attribute_Id =
Attribute_to_Metadata.Attribute_Id) ON (Base.Field_B_Id =
Base_to_Attribute.Attribute_Id) AND (Base.PKey =
Base_to_Attribute.PKey)) ON Metadata.MetaData_Id =
Attribute_to_Metadata.Metadata_Field_Id
WHERE (((Base.PKey)=12345));
```

Figure 4-4: Vertical SQL Statement

Figure 4.4 shows an SQL statement that retrieves all of the metadata and attribute version values for a given entry in the base table for one versioned attribute – field B. The user would have to issue a similar query to retrieve all of the attribute versions for fields C and D as well as a slightly simpler query to retrieve the attribute values for field A. Notice that this implementation does not prohibit any field from being versioned. In fact, the implementation has the advantage that all fields are eligible for versioning.

It is interesting to note the complex inner joins necessary to follow the path of class hierarchies to retrieve the value for the attributes. Although simple in comparison to actual implementations, this clearly shows the complexity introduced in order to achieve a versioning mechanism in an object-relational database.

There are other ways to form this query that would avoid the complexities of the inner joins, but they introduce nested select statements. Nested selects are more difficult for the query processor to evaluate and execute and thus usually results in longer query execution times.

المنارة للاستشارات
www.manaraa.com

Another problem with using the vertical approach to storing versioning information is the expansion of tables and the many-to-many relationships that often develop during schema design. There are many techniques to overcome this problem. One technique is to use a supertype/subtype hierarchy and replace the many-to-many tie tables with class lookup tables. This drastically reduces the query processing time, but results in a less than ideal data relationship among the versioned data [Bane03].

4.2.2.3 Are There Alternative Implementations for a Version Store?

In the realm of possibility, there is always another way. In the case of the clustered version store, there was at least one alternative other than the strategies described above that could have provided a possible solution. The MySQL server provides the database professional with the ability to choose from a set of possible storage engines[7] (table types). These subsystems are responsible for the storage and retrieval of all data stored. Like the various file systems available for operating system, each has its own benefits and drawbacks. Fortunately, many of the differences are transparent at the query layer.

This flexibility allows database professionals to tailor the physical storage of each table to the mechanism that best fits the use of the table. It is even possible to mix and match tables of different types in the same database. MySQL has six distinct table types [Horn03, Vasw04, Zawo04]:

[7] MySQL refers to these as “table types” – a somewhat misused term, but one that we shall use to remain consistent with the MySQL literature and nomenclature.

المنارة للاستشارات
www.manaraa.com

- ISAM
- MyISAM
- InnoDB
- BerkeleyDB (BDB)
- MERGE
- HEAP

ISAM tables are basic implementations of the indexed sequential access method (ISAM). While storing data sequentially, ISAM provides direct access to specific records through an index. This combination results in quick data access regardless of whether records are being accessed sequentially or randomly.

MyISAM tables are an extension of the ISAM table type built with additional optimizations such as advanced caching and indexing mechanisms. These tables are built using compression features and index optimizations for speed. InnoDB is a third-party storage engine licensed from Innobase (www.innodb.com) distributed under the GNU Public License (GPL) agreement. All indexes in InnoDB are B-trees where the index records are stored in the leaf pages of the tree. The default size of an index page is 16KB. BDB is a third-party storage engine licensed from SleepyCat (www.sleepycat.com). Berkeley DB supports hash tables, Btrees, simple record-number-based storage, and persistent queues.

MERGE tables are built using a set of MyISAM tables with the same relvar that can be referenced as a single table. Data is accessed using singular operations or

المنارة للاستشارات
www.manaraa.com

statements such as SELECT, UPDATE, INSERT, DELETE. Fortunately, when a DROP is issued on a MERGE table, only the MERGE specification is removed. The original tables are not altered. The biggest benefit of this table type is speed. It is possible to split a large table into several smaller tables on different disks, combine them using a MERGE table specification and access them simultaneously. Searches and sorts will execute more quickly since there is less data in each table to manipulate. For example, if you divide the data by a category, you can search only those specific portions that contain the category you are searching for. Similarly, repairs on tables are more efficient because it is faster and easier to repair several smaller individual files than a single large table. Presumably, most errors will be localized to an area within one or two of the files and thus will not require rebuilding and repair of the entire data. Unfortunately, this configuration has several disadvantages; 1) One can use identical MyISAM tables to form a single MERGE table, 2) the REPLACE operation is not permitted, and 3) indexed access can be less efficient than for a single table[8].

HEAP tables are in-memory tables that use a hashing mechanism for indexing. Thus, these tables are much faster than those that are stored and referenced from disk. They are accessed in the same manner as the other table types, however the data is stored in-memory and is valid only during the MySQL session. The data is flushed and deleted on shutdown (or a crash). HEAP tables are typically used in situations where static data is accessed frequently and rarely ever altered. Examples of such situations include zip code,

[8] With several tables there exists an ind ex tree for each thus for each file searched a separate index must be tranversed.

المنارة للاستشارات

www.manaraa.com

state, county, category, etc. and other look up tables. HEAP tables can also be used in databases that utilize snapshot techniques for distributed or historical data access.

Some of the table types offered in MySQL support concurrency. The default table type for MySQL is MyISAM. It supports table-level locking for concurrency control. That is, when an update is in progress no other processes can access any data from the same table until the operation is completed. The MyISAM table type is also the fastest of the available types due to optimizations made on the ISAM table principles. The BDB tables support page-level locking for concurrency control. That is, when an update is in progress, no other processes can access any data from the same page as that of the data being modified until the operation is complete. The InnoDB tables support row-level locking for concurrency control. That is, when an update is in progress no other processes can access that row in the table until the operation is complete. Thus, the InnoDB table type provides an advantage for use in situations where many concurrent updates are expected. However, any of these table types will perform well in read-only environments such as web servers or kiosk applications.

Concurrency operations like those discussed above are implemented in database systems using specialized commands that form a transaction subsystem. Currently, only two of the table types listed support transactions – BDB and InnoDB. Transactions provide a mechanism that permits a set of operations to execute as a single atomic operation. For example, if a database was built for a banking institution the macro operations of transferring money from one account to another would preferably be executed completely (money removed from one account and placed in another) without

المنارة للاستشارات
www.manaraa.com

interruption. Transactions permit these operations to be encased in an atomic operation that will back out any changes should an error occur before all operations are complete, thus avoiding data being removed from one table and never making it to the next table. A sample set of operations in the form of SQL statements encased in transactional commands is shown in Figure 4-5.

```
START TRANSACTION;
UPDATE SavingsAccount SET Balance = Balance - 100
WHERE AccountNum = 123;
UPDATE CheckingAccount SET Balance = Balance + 100
WHERE AccountNum = 345;
COMMIT;
```

Figure 4-5: Sample Transaction SQL Statements

In practice, most database professionals specify the MyISAM table type if they require faster access and InnoDB if they need transaction support. Fortunately, MySQL provides facilities to specify a table type for each table in a database. In fact, tables within a database do not have to be the same type. This variety of table types permits the tuning of databases for a wide range of applications [Vasw04].

Interestingly, it is possible to extend this list of table types by writing your own table handler. MySQL provides examples and code stubs to make this feature very accessible to the system developer. By being able to extend this list of table types, it is possible to add support to MySQL for complex, proprietary data formats and access layers.

The main reason this approach was not taken was the need to keep the clustered version store as separate from the MySQL implementation as possible, to reduce risk to

المنارة للاستشارات

www.manaraa.com

the MySQL core functionality and to interface better with the specialized query optimizer and query execution engine that was developed to exploit the clustered version store. From a technical implementation standpoint, this option is not feasible due to the uniqueness of the result sets returned from queries on the clustered version store.

4.2.2.4 The Use of Superkeys

A technique which has been used in both horizontal and vertical implementations is the application of superkeys to data to overcome the uniqueness constraint. A superkey is a special attribute designed to guarantee uniqueness within the table. Superkeys therefore permit an artificial uniqueness characteristic for the rows in a table. Unfortunately, superkeys are seen by many, including the author, as a means to obscure and violate first normal form. Indeed, the use of superkeys disassociates the meaning and significance of uniqueness that is required for relational systems. In many ways, the application of superkeys has become a “cheap” way to implement complex access mechanisms. For these reasons, the application of superkeys has been expressly avoided in this work.

4.2.3 Clustered File Organization

Most database systems store relational data in separate files and utilize the operating system file subsystem for access. Some database systems organize the files by database, others organize the files by table. Data is then added at the end of the file or placed in empty spaces left from deletions. Thus the data can become fragmented

المنارة للاستشارات
www.manaraa.com

resulting in data that is normally accessed together[9] being scattered within the file. Although this mechanism is efficient for smaller data sizes, it becomes much less efficient for larger data sizes. Queries for this data result in many accesses made to the file subsystem, perhaps even one page of data read for every data item. One way to overcome this limitation is to group associated data together in the same place in the file. One could then chain the data together and access the data using a minimal number of file subsystem reads. This type of file organization is called clustering [Rama03, Silb96].

There are several standard methods for storing blocks of a file on disk. Contiguous allocation specifies that the blocks are located in a consecutive block chain. This enables fast retrieval of the file during a file scan using a very fast double buffering mechanism, but it makes expanding the data difficult – new blocks must be inserted in the block chain and may require reorganization of the file for optimal performance. Linked allocation is a physical mapping of a linked list where each block contains a pointer to the next block (in some implementations the previous block as well). This method overcomes the problems of inserting blocks, but does not optimize scanning the entire file since there is no guarantee that the blocks are in close proximity on disk. A combination of these techniques that allocates data in groups of consecutive blocks is called clusters. The clusters are then linked thereby achieving a compromise of the benefits (and detriments) of both techniques. The linking of the clusters is referred to as establishing extents of the previous cluster. Extents therefore permit the expansion of

[9] Some examples include addresses, family names, master/detail associations, etc.

المنارة للاستشارات
www.manaraa.com

files in an orderly and predictable manner that agrees with the basic principles of good file organization [Elma03].

Disk clustering[10] attempts to store all the data which a query would want has been stored close together on the disk. In this way when a query is executed the database system can simply "scoop" up a few tracks from the disk and have all the data it needs to complete the query. Without clustering, it may be necessary to move over the whole disk surface looking for bits of the query data, and this could be hundreds of times slower than being able to get it all at once. Most database system systems perform clustering techniques, either user-directed or automatically.

Clustering is a concept based on the premise of storing data that is logically related (and thus frequently used together) in groups on disk that can be retrieved as a unit. This physical proximity approach is a very important tool in file organization (physical media layout) [Date04a].

Clustering can be implemented in database systems using intra- and inter-file[11] techniques by storing related data items in the same block (page) whenever possible and otherwise on consecutive, adjacent blocks. Thus the database system must be capable of managing the file organization. It also must not interfere with the normal operation of the file system of the host operating system. This database subsystem is often referred to as

[10] Also called aggregation by March [Marc83].
[11] Inter-file clustering techniques include tran sposed files and other hierarchical storage techniques. This chapter concentrates on intra -file techniques.

المنارة للاستشارات
www.manaraa.com

the file manager. When a new data item is created, the file manager must store the data nearby in the same block or in one of the adjacent or consecutive blocks[12] [Date04a].

The most desirable technique is one which allows blocks to be stored physically adjacent or contiguously, allowing an entire collection of data to be read when sequential access is desired. This naturally leads database system implementers to prefer an extent based file system (e.g., VSAM) that allocates blocks of related data. However, such files must grow by adding an extent at a time rather than a block at a time [Ston81]. Clustering is most effective when the greatest amount of related data can be fetched by a single physical media I/O operation [Cham81a].

Although clustering data (tuples) that have a high degree of commonality is beneficial, one should not cluster data (tuples) that have little or no commonality. The reasons are implementation-dependent but generally.

a) Small page sizes typically result in near or adjacent placement on disk thus clustering would have little or no effect. This is especially true in the UNIX environment. Stonebraker stated in his paper concerning the design of INGRES that the concept of adjacent pages in a virtual memory system such as Unix does not imply that the data is physically adjacent [Ston76]

[12] Consecutive in this case indicates the next or previous block in the file access chain. It is often impossible to specify a physical address as mo st operating systems have their own file subsystem that manages disk space allocation.

www.manaraa.com

b) Access methods are more complicated if clustering is supported. The clustering of tuples only makes sense if associated tuples can be linked together as sets of related data, sets of links, or some other scheme for identifying clusters. Incorporating these access paths into the decomposition scheme would greatly increase its complexity.

It should be noted that the designers of System R have reached a different conclusion concerning clustering [Ston76].

4.3 Clustered Version Store

Database systems have very specific physical storage mechanisms that are optimized for relational data storage and retrieval. It is therefore incorrect to assume that the concept of attribute-level versioning could be implemented using conventional techniques. The versioned data does indeed resemble that of a table, but it is more accurate to describe the version data as a collection of sets where each set has its own logical format. Thus, the concept of a relational table store is inadequate to store the versioned data. What is needed is a storage mechanism that can efficiently store and retrieve version information. This information is not very useful without its parent data (the original data item that is versioned). It is therefore important to make a direct association between the parent data and its versions. This concept is exactly what the clustered version store is designed to do [Elma03].

المنارة للاستشارات
www.manaraa.com

The goals of ALV include optimizing the retrieval of version information of an entity and the storage of meaningful metadata. These goals are the driving factor for the research presented here. Why store metadata? Metadata provides the ability to track the origin or pedigree of an attribute version. Some possible metadata to consider include source of information, date, time, qualifier data, etc. Associating this metadata and storing it with the attribute version provides the ability to perform analysis on the versioned data. For example, it would be possible to categorize the data based on a metadata field that stored reliability information.

The clustered version store attempts to solve all of the problems by providing inherent relational operations, that remove the burden of logical design complications, without modification to the base structures and by providing a optimized query mechanism.

The goals of the clustered version store are shown below:

- Store sets of attribute versions in contiguous blocks of data for fast retrieval.
- Permit attribute versions to contain disjoint sets of metadata.
- Utilize the operating system for file-level I/O.
- Retain relationships to the base table without requiring changes to the base table structure.
- Utilize chaining techniques to reduce complex reference mechanisms.

This section presents the inner workings of the clustered version store (CVS). An examination of the implementation design and techniques are presented along with a

www.manaraa.com

detailed explanation of the file layout and operation of the binary file representation of the clustered version store.

4.3.1 Technology Descriptions

The clustered version store is implemented as classes within a C++ program. All aspects of the operation of the clustered version store are abstracted and represented as classes. Starting from the individual data items, an attribute is a class that contains values, a tuple is a class that contains attributes, and finally a relation is a class that contains tuples. In a similar way, the physical representation of data is also represented as a set of classes and helper classes. The lowest level of the system is the data file, which manages data in blocks, followed by the record file which manages data at the attribute-version[13] level. Additional helper classes are a queue class for managing block lists and a hash table class for managing lookup lists for the attribute versions and metadata during read and write operations. For more information about these classes, see Appendix B.

4.3.1.1 Physical Design Goals

The physical design need not be optimal in order to perform and scale well [Bato82]. In a related corollary, it is also considered that underloading, the intentional use of extra storage[14] for use in adding new or modifying data, can have such long-term benefits as reducing the time between maintenance operations and degradation of access

---

[13] Since an attribute version is the attribute value that is versioned along with all of its associated metadata, one can easily associate this as a 'tuple' and thus the record file is operating the same way a traditional record-level manager for a traditional database system implementation [Cham81a].

[14] The over zealous application of these "white spaces" can lead to the opposite effect – too much irrelevant data becomes a burden on the system.

المنارة للاستشارات
www.manaraa.com

path execution. The primary concern of the database designer is to minimize the cost of data access. This is especially important when one considers that accessing data on physical media is several orders of magnitude slower than accessing the same data from memory [Seve77].

Another area where a considerable amount of research has been conducted is the concept of blocking factor – simply the number of tuples stored in a single page (block). The proper balance of blocking factor and underloading can achieve a more stable access path execution for longer periods of time [Yao77].

If a block is full, a new block is allocated and the existing block extended. The block extents are managed using a blocking chain similar to the attribute chains [Date04a]. Figure 4-6 illustrates how this is accomplished using two links – one that points to the block that the current block extends (the backward pointer named extends) and another that points to the next block in the extent (the forward pointer named extended by). Each block has a data element that permits the system to determine if there are more extents by examining the extends data item thus moving "forward" in the chain. Similarly, the system can determine if the current block is an extension of another block thus moving "backward" in the chain.

Extends	Extended by
16	23

Figure 4-6: Extents Addressing

المنارة للاستشارات

www.manaraa.com

4.3.1.2 Attribute Chains

The file blocks store attribute versions using pointer chains [Date04a]. Pointer chains permit fast access to related data without having to search a block. Pointer chains also permit efficient internal storage within the internal representation of the data. That is, pointer chains are simply linked lists and thus all the advantages of linked lists can be had without having to build them from the raw data. In fact, it is matter of address mapping to reference the pointer chains directly from the data buffer (cache). The principal advantage of using pointer chains is it enhances not only retrieval but also insertion and deletion. Since pointer chains chain together the attribute versions, they are called attribute chains.

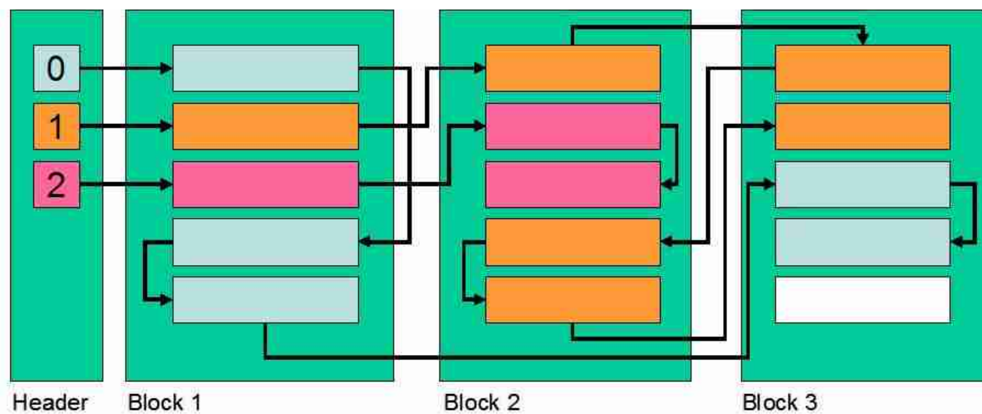


Figure 4-7: Attribute Chains

Figure 4-7 and figure 4-8 depict the logical layout of a hypothetical data file containing three attribute chains that store version data for three attributes and the actual logical mapping of the physical data store. The drawing depicts two blocks of an ALV file. The two blocks shown illustrate how the block extents are implemented. Note in block 3 that the next block is block 4 and in block 4 the previous block is block 3. The

www.manaraa.com

drawing also illustrates how the block offsets are used to continue attribute chains across a set of blocks. For example, the attribute chain that begins in block 3 is extended to block 4. To maintain backwards linking, the attribute chain that is continued in block 4 points to the previous link the attribute chain in block 3.

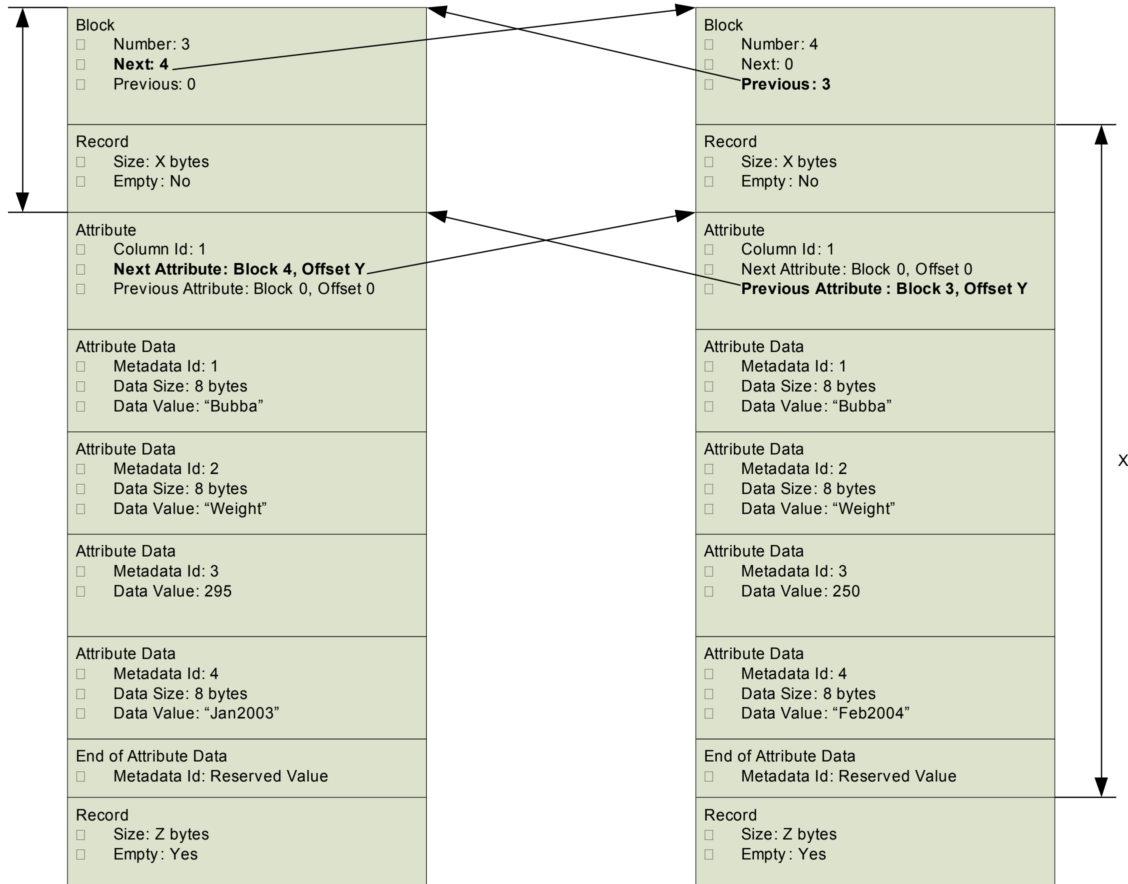


Figure 4-8: Attribute Chain Layout with Block Headers

Since the attribute chains require only a blocknum:offset mechanism to locate the next attribute version in the attribute chain, it is possible to reclaim unused space from delete operations to store attribute versions in order. It should be noted at this point that the ordering of the attribute versions is arbitrary and does not have bearing on the order

المنارة للاستشارات
www.manaraa.com

of the data itself[15]. This technique is similar in many ways to the implementation of INGRES [Ston76].

In this case, the header contains the starting pointer references (roots) of all of the attribute chains. The header is repeated for each starting block of the version data for a given base table entry. Thus, it is possible to determine which entities have version data for a given attribute.

Conventional physical layout of pages have tuples placed entirely on a single page and no tuple may span more than one page [Ston76]. In much the same way, attribute versions are required to be placed on a single page and are not permitted to span multiple pages. However, attribute version chains may span more than a single page using the extension technique described above.

4.3.1.3 Secondary Representation

The primary storage media isn't the only layout that must designed. It is also important to consider the layout and representation of data in memory. The reasoning follows the same argument as that of physical media – one must consider the many ways that the data will be used and thus one must design to maximize the most important requirements levied. This secondary or internal representation must be capable of two important tasks. First, the storage mechanism must permit the logical traversal of the data, *i.e.* the ability to find the next item in the collection. Second, additional control data

[15] Although it could if one wanted to ensure ordering of the attribute versions. This is possible and perhaps even worthy of exploration as it would not require the relocation of any data to achieve.

المنارة للاستشارات
www.manaraa.com

must be minimized and all irrelevant control data eliminated [Marc83]. Data in the ALV system is represented in two ways:

1. Data that is cached in the buffer is stored in the same format as that on the physical store with a few additional pieces of reference data attached and the remapping of the access chains to memory pointers. The greatest advantage of using this technique is that the data need not be transformed when read from disk and placed in the buffer. Additionally, the mapping of access chains to memory pointers simplifies the data traversal implementations.
2. The internal representation of a tuple is encapsulated in a tuple class that provides all of the necessary operations and structures to represent a tuple in memory. The greatest advantage gained by using this technique is that the query processor can access and manipulate the tuple in ways that are intuitive to the implementer – the data "acts" like a tuple should.

4.3.2 Execution Sequence

The execution of the ALV system follows the same model as that of MySQL. That is, it is a multithreaded server application where each command is given its own thread of execution. Once the thread is created, control is passed to the parser where the SQL statements are parsed and directed to the appropriate execution method. A very large case statement is used to contain all of the possible execution methods for all of the available commands.

المنارة للاستشارات
www.manaraa.com

In order to integrate tightly with the MySQL system, the parser was modified to include catches for special ALV keywords. The location and type of keyword identified will cause the MySQL system to redirect commands to the ALV system for processing. For a complete explanation of this technique, including the execution sequence from the parser to the ALV system and its implementation, see Chapter 6. For a more in-depth study of how the MySQL code was modified, see Appendix B.

What is of interest for this discussion is the execution sequence during physical store access and the translation of that data to internal representation and out to the caller. Figure 4-9 gives a UML sequence diagram depicting this process from the point of entry to the ALV execution manager (named ALV_Manager).

Once a fetch is issued to a particular physical store, control is passed to a class called a Relation that encapsulates the concept of a versioned data store, which is very similar to that of a relation in relational theory. The Relation class then calls the ALVRecordFile class (implemented in the source code as ALVRecordFile) that encapsulates the physical store and includes a dedicated buffer manager. Depending on whether the datum is in the buffer, the ALVRecordFile will either retrieve the attributes for the tuple from the buffer using a hash for accessing attribute version metadata or load the page from disk, and then build a queue for storing the attribute information. The use of a hash permits the storage of attribute version chains that have different sets of metadata attributes. The hash table stores the lists of attribute chains indexed (hashed) by the definition of the attribute chain definition. Thus, the clustered version store can store

المنارة للاستشارات

www.manaraa.com

sets of attributes that do not share the same set of metadata[16]. Once the attribute structures are created, the Relation class will use the Tuple class to create a tuple in memory and return that to the relation class for storage in its internal memory cache. Execution would then return to the ALV execution manager and then on to the query execution engine, which evaluates and finally returns the tuple to the caller (provided it meets the criteria of the command).

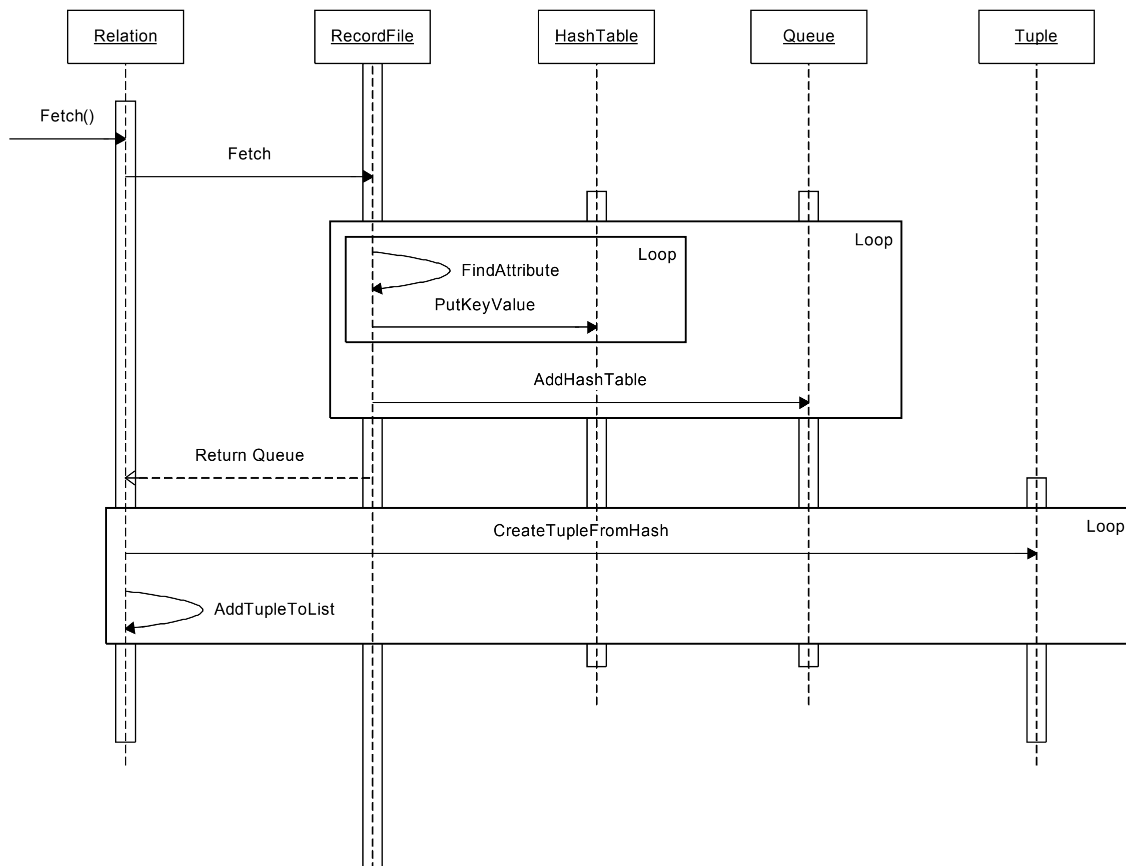


Figure 4-9: ALV Execution Sequence

[16] This is perhaps the most unique aspect of the clustered version store. No other relational database system can accomplish this.

المنارة للاستشارات
www.manaraa.com

4.3.3 Class Descriptions

The ALVDataFile is a set of C++ classes (see Appendix B) designed to manage the myTable.alv file (a disk file that stores the versioned attributes). It stores data in extensible blocks (block size is adjustable) and provides access to header information (header size is adjustable), free space, and statistical information about the data file. It also supports clustered block access for use in buffer management algorithms. The implementation of block extents (described above) is similar to those implemented in POSTGRES system [Ston94].

The class is designed to read whole blocks or clusters of blocks to/from the data file. Functions available to the caller include the ability to retrieve, update, and insert a random block by block id, a range of blocks, a cluster by starting block id (the entire cluster is returned in sequential order regardless of the order of those blocks on disk), and a range of clusters. The implementation of these functions follows the standard common practices for file I/O and data structure manipulation [Ston76].

The ALVDataFile is designed to manage records containing versioned attribute data and metadata within blocks managed by bptBlockMgr. Each block is mapped to a single record in a single standard MySQL table. The records within a block contain only versioned attribute values and metadata for that single record. The records are chained together in a singly linked list, where each record is empty or contains attribute values. Records know their size since they may be slightly larger than needed for the attribute data. Figure 4-10 depicts these classes and how they fit with the other major ALV classes. A complete description of the MySQL integration is shown in Appendix B.

المنارة للاستشارات
www.manaraa.com

Versions of an attribute are chained together in a doubly linked list. Attributes contain one attribute version value plus the primary key from the source row, the name of the column from the source row for this version, and any additional metadata values. Each attribute has a column id that is used in find operations.

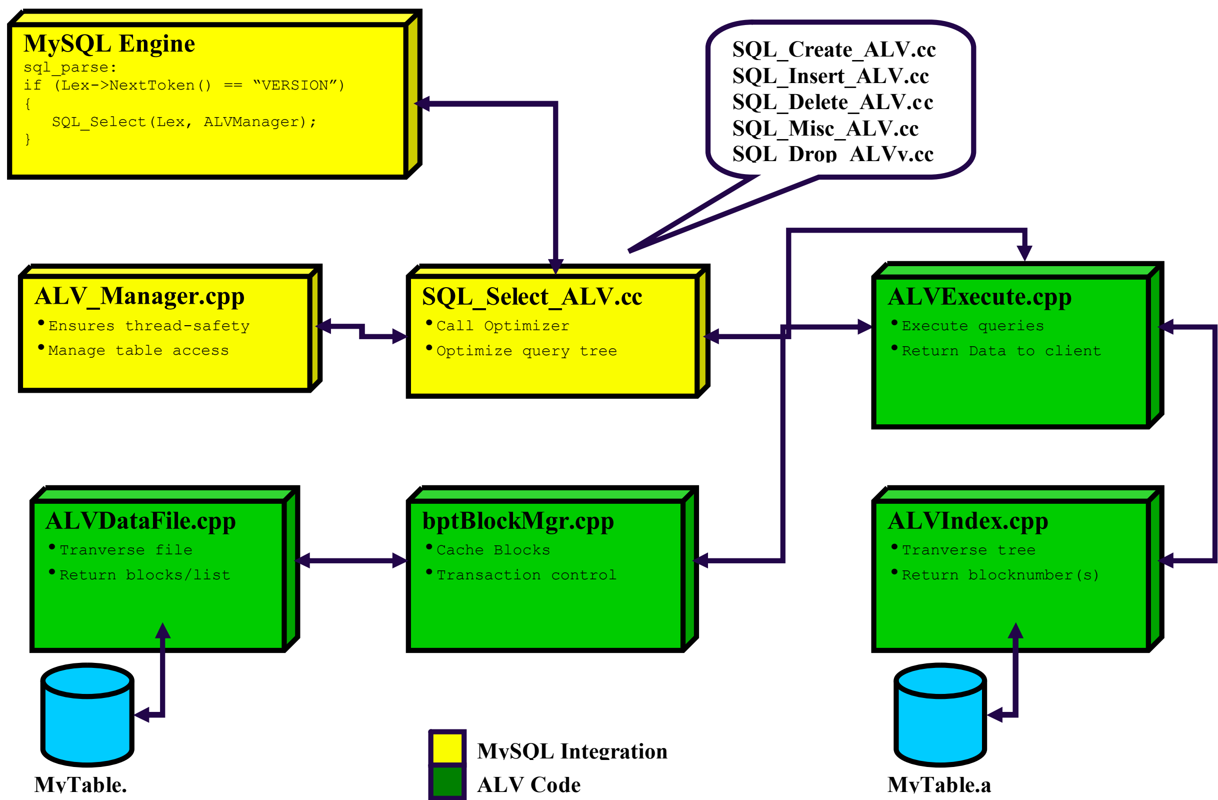


Figure 4-10: Major ALV C++ Classes

Records are allocated to exactly the size needed for the attribute data, except when too few bytes to form a new record would remain. Attribute values are never updated in place. Updates are always performed by deleting the old record and inserting a new record. This is necessary to ensure the attribute chains do not become too convoluted

www.manaraa.com

and remain in a general progression across the blocks. Over time, the allocation of records within a block may cause some fragmenting, but records are compacted every time one is deleted, so fragmentation should be minimized.

Attributes are always inserted into the attribute list in sequential order. This is important because it allows blocks to be locked going forward through the block list, preventing deadlocks from occurring. The only exception to locking blocks going forward is when an attribute is deleted. If the previous attribute is in a prior block, an attempt is made to lock the prior block. If it fails, the currently locked block is released to prevent deadlock. This process repeats until both blocks are locked.

If a new record is needed but cannot be allocated within the block, the block will be extended and the record allocated in the new block.

All attribute addresses are NOT implemented as standard pointers, since that would fix the memory location. Each pointer within the linked list is stored as 2 separate pieces of data: the block number and offset within that block. This makes the record independent of its memory location, which is crucial to allowing the blocks to be written to/read from a file.

4.3.4 The ALV Buffer Manager

The ALV Buffer Manager is a cache mechanism responsible for caching pages (blocks) of data stored on the physical media. It wrappers the physical access layer utilizing the interface to the physical access layer and translates the data from its physical form directly to an internal memory buffer adding only a small portion of control data to

المنارة للاستشارات
www.manaraa.com

each page. The primary benefit of the buffer manager is to provide concurrency control in the form of page-level reader-writer locking with writer priority and batch save/discard changes capability.

One buffer manager is associated with each physical store on disk. Each instance of the buffer manager maintains its own memory pool. This is necessary in order to avoid synchronization problems with sharing data from the internal memory pool with other elements of the system.

The control data associated with each page includes mappings from the physical page extent mechanism to memory pointers. This is accomplished by converting the linked list of extents on disk to a linked list in memory. Also included in the control data is the state of each page. The status contains the lock state used to support concurrency with states of "in-use" and "free" (deleted).

4.3.4.1 Integration with Physical Data Store

The buffer manager utilizes features of the physical data store in order to ensure a more successful integration. The file header (a reserved space at the front of the data store) contains a fixed amount of information from the buffer in the form of the block header data structure shown in Figure 4-11. The physical data store also provides mechanisms for external code (higher levels in the system) to specify that a fixed number of words of the file header be reserved for its own use as well.

Figures 4-12 and 4-13 depict an overview of the layout of the headers in the physical data store. The drawings show an example of the binary file layout (Figure 4-

المنارة للاستشارات
www.manaraa.com

12). In the example, the location of the file header is shown in relation to the blocks of data. Note that block 1 is used as an additional header for storing the metadata list for the attribute versions store in the file. The specifics of the headers for both the data file and the index file are shown in Figure 4-13. Note in the drawing that the data structures for the index are the data structures used in the B-Tree implementation of the index.

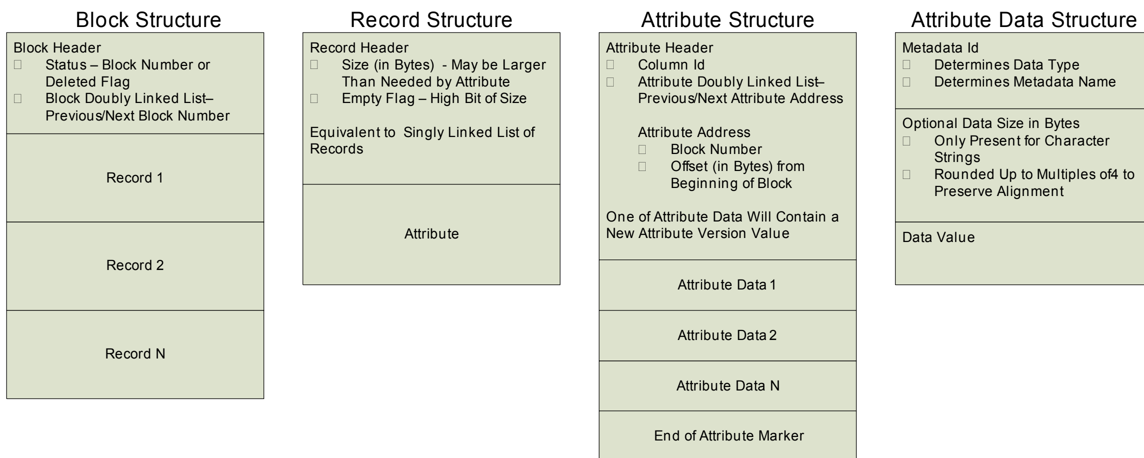


Figure 4-11: ALV Data Structures

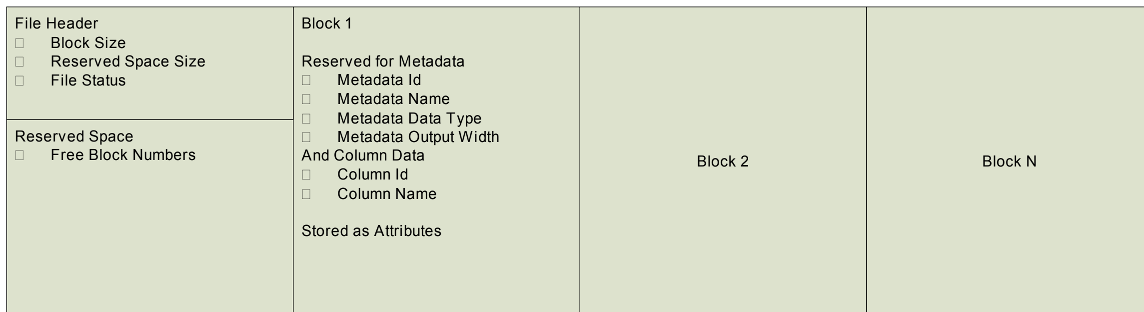


Figure 4-12: ALV Binary Data File Format

www.manaraa.com

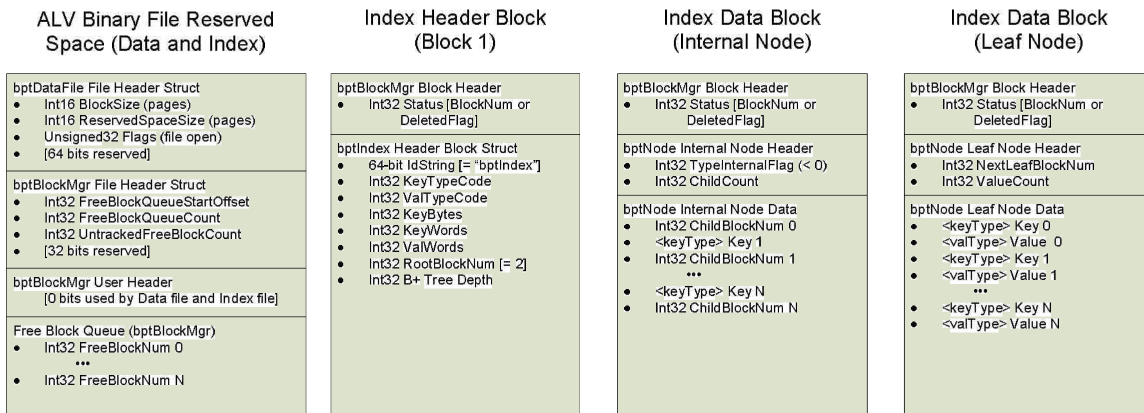


Figure 4-13: The ALV Physical Data Store Header and Block Diagram

The remaining space in the file header is used for recording the queue of free block numbers (one word per block) when the file is closed. The size of the free block queue is defined by the amount of space that will be available. It is important that the queue size be large enough to prevent frequent sequential scans through the file to rebuild it. Because the free block queue is stored in an area with a fixed size, it is possible for the free block queue to overflow. In this case, the free block queue records a counter of all of the insertions into the queue that exceed its capacity. Should the free block queue be exhausted at a later time, an algorithm is run on the first request for a free block. Although this algorithm requires walking the entire file, the cost is minimal compared to a manual (offline) reorganization. In this way, the physical data store can repair its free block queue when needed without requiring down time.

In addition to the file header, each block in the physical data store provides an area at the head of the block for implementation details. The first word (4 bytes) of each block is reserved for Status information.

المنارة للاستشارات
www.manaraa.com

The in-use vs. free [deleted] status of a block is indicated by the status word at the head of the block and is thus information that is persistent. The GetNew method will return a previously deleted block rather than enlarge the file, if possible. The ALV buffer manager also provides some tuning capability that permits the implementer to change certain parameters in order to tailor the performance of the ALV system for specialized needs. Currently, the parameters are implemented as static attributes and thus require compilation to change. It would be very easy to modify this behavior to allow tuning parameters to be passed from the client through the SQL parser down to the ALVRecordFile and ALVDataFile classes.

The constant SAFE_PAGE_SIZE is set at 4096 (bytes). This controls the size of the page and is currently set to the size of a sector in the Windows operating system file subsystem. Buffer blocks are multiples of this value and are created to align with memory page boundaries using the VirtualAlloc Windows API call. This permits the option of writing them to disk without Windows OS buffering.

There are two essential options built into the ALV buffer manager: 1) permit the buffer be as small as possible to support the locking mechanism, and 2) either rely on the Windows OS for performance buffering, or set the initial buffer size large and disable Windows OS caching altogether. The second option for allocating the buffer gave no performance benefit compared to the first. The current implementation is the latter. The buffering mechanism has been tuned to support multiple reads and temporal writes. Thus, the only blocks in memory are those associated with concurrent (shared) access and those that are involved in sustained operations (transactions).

المنارة للاستشارات
www.manaraa.com

There are parameters to set memory size and control locks. The i_bufferMBlockCount parameter of the OpenExisting/OpenNew commands sets the initial number of blocks of memory to reserve for the buffer manager buffer. This will grow if and only if it is required in order to support the number of locked blocks simultaneously held. The i_fileFlags parameter of the OpenExisting/OpenNew commands is passed to the Win32 CreateFile command (via the physical data store layer). Choices for each available flag and their contingencies and consequences are listed below:

- FILE_FLAG_OVERLAPPED – this flag must be set if multithreading is used. By default, the operating system will cache file blocks in memory for maximum performance, delaying disk updates significantly so that multiple updates can be made without a disk write. This can be changed by setting FILE_FLAG_WRITE_THROUGH or FILE_FLAG_NO_BUFFERING. A hint about expected file access patterns may optionally be supplied by setting FILE_FLAG_RANDOM_ACCESS or FILE_FLAG_SEQUENTIAL_SCAN.
- FILE_FLAG_SEQUENTIAL_SCAN causes read-ahead so that several consecutive blocks will be fetched from disk even if only one is requested.

4.3.4.2 Concurrency Support

An additional overseer subprocess (the ALV_Manager) is responsible for ensuring that only a single thread is active in an instance of this class during a call to the

المنارة للاستشارات
www.manaraa.com

constructors and destructors: OpenNew, OpenExisting, or Close. While a file is open, the buffer manager supports concurrency via block-level reader-writer locking with writer priority. Multiple threads can own a reader lock on a single block at the same time, but only one thread can hold a writer lock on a given block. The lock state of a block is a run-time datum that is known to an active buffer manager instance but is not recorded in any way in the data file.

The Get method obtains an MBlock for a given blockNum with the requested reader or writer lock type. If necessary, the calling thread will suspend execution until the desired lock type can be obtained. There is a GetIfAvailable alternative that will not suspend execution. The GetNew method obtains an (MBlock, blockNum) pair with a writer lock for a file block that was either previously deleted (preferred) or that extends the file. The initial contents of MBlock are undefined. The pair is returned already marked as changed. A thread that obtains a locked (MBlock, blockNum) pair via a call to Get or GetNew is expected to regard the locked pair as "thread-private" data not to be shared with other threads. Sharing locked pairs would subvert the locking system, and lead to undefined results. The buffer manager does not presently enforce this requirement by tracking which threads obtained which locks[17].

Calling MarkChanged or MarkDeleted on a WRITER-locked MBlock marks the block as changed or deleted. As presently implemented, you can only alter the mark of a block in the direction unchanged -> changed -> deleted. For example, calling MarkChanged has no effect if MarkDelted has already been called. The block manager

[17] Additional overhead will be necessary to support deadlock recovery.

www.manaraa.com

propagates changes to disk only when [Batch]UnlockSave is called, rather than immediately when the MarkChanged/MarkDeleted methods are called. Disk images are updated only for blocks that have been marked changed or deleted.

If an MBlock is not marked changed or deleted when it is unlocked, the buffer manager assumes that the MBlock can be retained in its buffer as a faithful copy of the file block. This copy may or may not be refreshed from disk the next time a thread requests the block. Thus a thread must ensure that a block is marked changed before unlocking if any changes have in fact been made to the MBlock.

Calling [Batch]UnlockDiscard will discard changes. If (MBlock, blockNum) was obtained with GetNew, discarding changes means effectively deleting the block and making blockNum available for a subsequent GetNew. If (MBlock, blockNum) was obtained with Get, discarding changes means that a subsequent Get will obtain a fresh copy of the disk image if and only if the block was marked changed or deleted. Every (MBlockP, blockNum) pair that is obtained from Get or GetNew must be unlocked via a matching call to [Batch]UnlockSave or [Batch]UnlockDiscard.

4.4 Analysis

This section describes the analysis performed while implementing and experimenting with the clustered version store and its individual components. All of the experiments conducted were run on a 3.0Ghz AMD processor-based system running Windows XP Professional. The disk subsystem used was a hardware raid system incorporating two S-ATA physical devices in a mirrored arrangement. The experiments

المنارة للاستشارات
www.manaraa.com

were repeated using a conventional IDE-133 device with little or no variation in the measurements[18].

4.4.1 Caching and Performance

The cache provided by the buffer manager is inherently write-through. Since the cache mechanism is an additional layer above the file system of the host operating system, performance for writes is optimized by using a write-through mechanism[19]. Depending on parameters passed when the file is opened, the operating system can be instructed to cache file blocks for highest performance, to implement write-through semantics for additional data safety, or to refrain from doing any caching at all on its level. This mechanism is designed for maximum flexibility and to provide support for transaction processing.

The best performance, by a factor of 10 or more for small files, was obtained in tests with the default operating system caching behavior. These tests were performed as a set of reads and writes using the ALVDataFile with a block caching mechanism. The caching mechanism was built to store frequently used blocks into an array of blocks in memory. Performance was the same whether the initial size of the buffer manager buffer was 1 block or 512 blocks. This performance superiority dropped some with file size. With no write-through of file changes, the default operating system caching has a greater risk of data loss if the system suffers from a catastrophic failure. Of the safer alternatives,

---

[18] This is expected because the differences in the performance of the physical devices and their access protocols are not significant. Although throughput on the S -ATA devices theoretically could be faster, the addition of the raid subsystem nullifies any advantage over IDE -133 devices.

[19] That is, as changes are made they are written directly to the physical store.

www.manaraa.com

performance was about the same for FILE_FLAG_WRITE_THROUGH with a buffer manager initial buffer size of 1 or 512 blocks, and for FILE_FLAG_NO_BUFFERING with a buffer manager initial buffer size of 512 blocks. All of these choices reduce disk I/O on reads but not on writes. Since the buffer manager's memory is not shared among instances, we conclude that a small initial buffer size (e.g. 1-32 blocks) with either default operating system caching or FILE_FLAG_WRITE_THROUGH is the best choice.

4.4.2 Blocksize Experiment

Two experiments showed that the ALVDataFile was performing as expected and demonstrated its ability to scale to larger block sizes and larger files. The experiments were conducted using a special executable written to include only the ALV file I/O class ALVDataFile. A series of twelve tests were conducted in total which simulated read and write loads as well as file scans and random access. All experiments were conducted using file sizes of 1Mb, 800Mb, and 2.2Gb.

One experiment was to decide what the optimal blocksize should be and what effect it would have on read and write performance. In this case, write performance was chosen to be implemented as a create (adding a new block/cluster). Create is the most expensive write operation because it requires allocating a new block to the file store and therefore interacts with the file system more than a simple replacement operation. Figure 4-14 depicts the results of a series of tests to show the performance based on blocksizes ranging from 512 bytes to 8194 bytes. The increase in blocksize showed a fairly linear

المنارة للاستشارات
www.manaraa.com

progression of access time (in seconds). Larger blocksizes of 16k and 32k, which exhibited the same linear time increase. Figure 4-14 depicts the results of these experiments.

Another experiment involved the study of cluster sizes based on block size. The test was designed to show what influence, if any, the cluster size had on access time. In theory, reading more data at one time should decrease the access time. Tests were conducted using block sizes ranging from 512 bytes to 8194 bytes and cluster sizes ranging from 1 to 64 (in powers of 2^{20}). Figure 4-15 depicts the results of this experiment. The results showed the expected performance and clearly demonstrated the premise of clustering in action.

File Access Times

Time (sec)
350
300
250
200
150
100
50
0
512
1024
2048
4096
8192
Block Size
Create
Read

المنارة للاستشارات
www.manaraa.com

Walk by Cluster

Time (sec)

Cluster Size

512
1024
2048
4096
8192

المنارة للاستشارات
www.manaraa.com

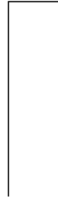
Given the special properties of the ALV physical store (clustering, differing attribute metadata per attribute chain, etc.), data had to be constructed so that the MySQL implementation could approximate that of ALV and vice-versa without the aide of indexing mechanisms. Eliminating indexing mechanisms enabled a more accurate comparison of the two physical store access times. Data for this experiment was taken from instrumentation mechanisms embedded in the live MySQL source code.

In light of such manipulations, the primary focus of this experiment was to show how the ALV physical store ranked in speed (access time) of small, medium, and large data sets as compared to MyISAM and InnoDB. Figures 4-16 through 4-18 shows the results of those experiments for table sizes ranging from 599 to 201,053 entities.

The results show that the ALV physical store has good performance as compared to MyISAM with small and medium sized tables, but poorer performance when table sizes become large. However, this performance is not typical of how the ALV system will be used. The addition of a clustered version store index (see Chapter 5) greatly enhances the performance of the ALV file retrieval mechanisms.

Furthermore, the times shown are actual times and do not consider the significant size differences between the ALV physical store and MyISAM (or InnoDB). That is, the ALV system is reading 60 times more data for the small and medium table and 20 times more data for the larger table. When one considers this factor, the performance of the ALV physical store is much less than 20 or 60 times slower. In fact, the ALV physical store retrieval times were approximately 7.5, 8.2, and 8.5 times slower respectfully for the three tables read.

المنارة للاستشارات
www.manaraa.com



File Retrieval - Small Table

Time (secs)

ALV
MyISAM
InnoDB

Table Scanned	Customer
ALV	0.0152842700
MyISAM	0.0020339740
InnoDB	0.0011315960

File Retrieval - Medium Table

Time (secs)

1.00
0.90
0.80
0.70
0.60
0.50
0.40
0.30
0.20
0.10
0.00

ALV
MyISAM
InnoDB

Table Scanned	Adults
ALV	0.8611705000
MyISAM	0.1044296000
InnoDB	0.0612593100

المنارة للاستشارات

www.manaraa.com

File Retrieval - Large Table

Time (secs)

ORF

	ORF
ALV	6.6346780000
MyISAM	0.7803103000
InnoDB	0.3553471000

Table Scanned

www.manaraa.com

	Customer	Adults	ORF
ALV	1,484,200	77,588,297	145,520,114
MyISAM	24,576	1,213,440	6,953,984
% Diff	60.39	63.94	20.93
Size (Rows)	599	32,561	201,053
#Blocks	369	19,395	36,403
Avg Att/Blk	1.6233	1.6788	5.523

Table 4-2: File Retrieval Experiment Data

Table 4-2 shows the average attribute values per block of each table. This is equivalent to the concept of blocking factor found in traditional database systems. In this case, we can see that the ALV physical store stores approximately 1.6 attribute versions per block for the small and medium tables and approximately 5.5 attribute versions per block for the larger table.

Figure 4-2 also compares the blocksize of the MyISAM files versus the ALV files. The third row shows the percentage of the ALV file size that the MyISAM files consume. The fourth row lists the number of rows in each dataset and the fifth row lists the number of blocks that the ALV files consume.

Therefore, the table scan performance of the ALV physical store is comparable to that of MySQL for smaller and medium sized tables, but table scans suffer for larger table sizes. Fortunately, table scans are rare in versioning applications. Most queries in versioning systems are targeted for specific attribute version chains that can be accessed by reading only a single block or block chain [Mart02].

المنارة للاستشارات
www.manaraa.com

4.5 Conclusion

The clustered version store is the cornerstone of the ALV system. By demonstrating the ability to store attribute versions in a dedicated, specialized physical storage mechanism that utilizes a buffer management system for caching, the clustered version store is the foundation of a versioning system that can be integrated into a relational database environment. The physical store is sound and performs admirably when compared to the commercially available physical store available in MySQL[21].

4.6 Future Work

Although the clustered version store performs well and outperforms the native storage mechanism of MySQL, there are areas that can be improved. Despite the tendency and practice of database system vendors to rely on the base operating system for file I/O support, much could be gained by developing a native I/O mechanism that communicates directly with the hardware. This would enable a more efficient use of disk space and eliminate the need to coordinate directly with the operating system. The drawback to this approach is that an operating system driver must be created so that the operating system can communicate with the device. It would be enlightening to develop such a storage mechanism and compare its performance with that of native data stores and the data store presented here.

On a more subtle scale, there are improvements that can be made to the clustered version store that may increase performance even further. For example, an active space

[21] Although MySQL supports a number of physical stor es, the comparisons made in this work were down with MyISAM because MyISAM most closely resembles that of the ALV physical store.

www.manaraa.com

reclaim process could eliminate the concern for large gaps in the file structure under heavy insert and delete operations. Furthermore, an active space reclaim process would eliminate the need to perform periodic maintenance on the files.

A vulnerability of the implementation of the clustered version store is that it does not currently have an active deadlock prevention algorithm. Additional overhead mechanisms may be necessary to support active deadlock recovery.

For a more robust application of versioning, one would also consider expanding the buffer management subsystem to include recovery mechanisms that can recover data in the event of an unexpected system termination.

Aside from the above improvements, the most beneficial additional (perhaps even necessary improvement) is to make the implementation code as platform independent as possible. The system currently runs on a Windows-based operating system. Additional work will be necessary to make some of the lower-level I/O code platform independent. Fortunately, the feasibility of platform independence has already been demonstrated in the MySQL source code and could be achieved by applying the same approach.

المنارة للاستشارات
www.manaraa.com

Chapter Five – An Indexing Mechanism for fast Version Retrieval

Abstract

Given that there exists a clustered version store capable of efficiently storing and retrieving version information in the form of attribute versions and attribute chains, the access to this physical store must now be optimized for fast retrieval of data sequentially and randomly. This chapter shows one implementation of an indexing mechanism using a B+ tree for indexing the clustered version store and reports its performance as compared to a commercially available storage mechanism.

5.1 Introduction

A fast indexing mechanism is required to ensure high speed performance of retrieval of versioned data for a versioning system that can be supported in a relational database system. This chapter presents an indexing mechanism for fast sequential and random access of versioned data in a clustered version store within a versioning system. This system, called Attribute-Level Versioning (ALV), is an extension of the MySQL database management system.

The following sections present the current research on database indexing and implementation, the technology and design of an indexing mechanism, an analysis of the performance of the mechanism, and a conclusion as to its success in meeting the goals

المنارة للاستشارات
www.manaraa.com

defined above. The chapter concludes with a section outlining future work opportunities to improve the indexing mechanism presented.

5.2 Background

Storage facilities available in large computing systems allow efficient storage, updating, and retrieval of data from a physical storage device. However, computing systems must retrieve the information from the physical store and place it in memory before applications can use it. The process of accessing data on the physical store must be made efficient for files of great size [Come79]. Although it is many times faster to access data in memory, it is also generally accepted that entire data sets will not fit into memory. Indexing is the process of identifying the location of things within a larger context. That is, an index makes it easier to find a single item among a large set of items[1]. In most cases, the objective is to locate only a subset of the data items stored [Marc83]. This is exactly what database implementers want to do in order to speed up access to a particular datum within a data set. Some alternative names for indexing are external searching [Sedg98], advanced data structures [Corm01], and ordered indexes [Baye72].

Indexing mechanisms should not be confused with ordering mechanisms. Ordering mechanisms such as mergesort, heapsort, and their many variants are not indexes per se [Wegn89]. While they do create an ordering and can be used to access data using that ordering, all mechanisms of ordering eventually change the order of the data

[1] Interestingly, the common definition of indexing does not require that the items be related in any way other than being grou ped together in a (loosely) coupled set. The common application of the definition does require that the items maintain a relative position within the set where order need not apply.

المنارة للاستشارات
www.manaraa.com

on the physical media. Some manage this by using logical pointers that are redirected, others physically relocate the data items. Indexes are designed to be used in addition to the data without requiring the data to be relocated when ordering is desired.

Indexing algorithms and data structures have been around for a long time, even before the proliferation of personal computers and before the days of the behemoths of the golden age of computing[2]. Early pioneer work in the realm of databases systems like that of IBM's System R and INGRES have incorporated indexing mechanisms to enhance the performance of reading data from the physical store [Cham81a, Ston76].

While indexing gives the ability to quickly retrieve data, it also has the disadvantage of slowing down update operations by requiring an insertion into the index for every (unique) datum added [Date04a]. There are many and varied mechanisms; most of which are specialized for a given set of conditions. However, there are a few that have been proven to be robust and adaptable for database applications. The most important characteristic of these mechanisms is the support for persistent storage along with or in addition to the data. These include indexed sequential files, hashing, and trees. Most of the database systems in use today employ one or more of these mechanisms [Date04a].

There are many algorithms and data structures devoted to searching and indexing. These include, balanced trees, binary trees, balanced binary trees, multiway trees (specifically B+ trees), hashing, indexed sequential files, stacks, queues, linked lists, and

[2] The golden age of computing refers to the days when a single computer co nsumed the space of several mini -vans. Some of the greatest innovations employed today in every computing device known to man were invented or conceived during this historical period.

المنارة للاستشارات
www.manaraa.com

many more [Corm01, Knut97, Sedg98]. This work examines those indexing methods directly applicable to database systems.

Trees as indexes are structures that divide the indexed keys into nodes in a data structure where each node (except the leaves) has a specified number of cells that can be used to direct a traversal to a lower node. Typically, these nodes contain the index key values in some predetermined order. Figure 5-1 depicts a typical tree structure designed to contain keys and references to the next node in the tree. The leaves are designed to point to locations outside the tree which are typically used to reference blocks of data in the data store. These trees are called multiway trees [Knut97] because each node can have a number of pointers to other nodes, except for the leaves which are used as described above. Multiway trees are the foundation for more sophisticated derivatives known as B trees[3] and B+ trees.

The keys in the multiway tree are stored such that each node has at most N+1 references where N is the desired number of keys each node can have. The keys must be ordered in a specified order (typically ascending). Furthermore, each key must be greater than or equal to the lowest value key in the node beneath it (child). Thus when the location of a reference to a page on disk is needed, the index is searched starting at the root, searching its list of nodes using either a linear or binary search [Knut97] until a key value that is greater is found, then dereferencing the previous entry's node pointer and following the node pointers to the next node down the tree. The search process repeats until the leaf node is located and the reference is obtained by again searching the list of

[3] It is tempting to conclude that the 'B' in B trees stands for balance d or a host of other b-words. However, it is generally accepted that the 'B' in B tree stands for 'Bayer' for his many contributions on the topic.

المنارة للاستشارات
www.manaraa.com

nodes using either a binary or linear search [Corm01]. The use of N+1 locations for keys permits the use of algorithms that expand and balance the tree for optimal performance.

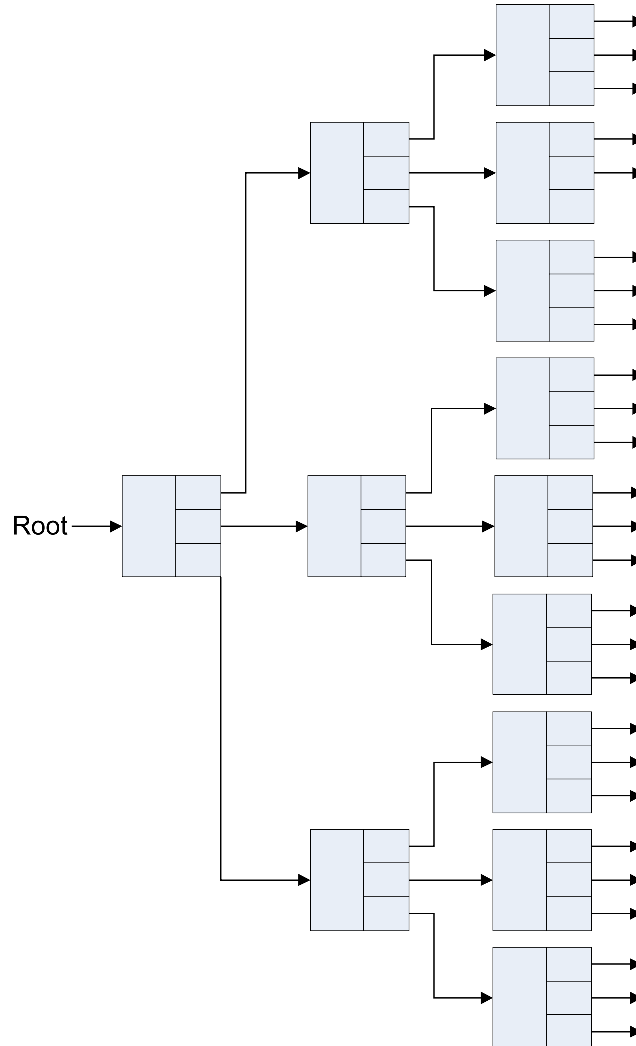


Figure 5-1: Multiway Tree Structure

5.2.1 Indexing Methods

With indexing, we are concerned with finding the data we actually want quickly and efficiently, without having to request and read more disk blocks than absolutely necessary. There are two types of searching that can be performed. An internal search is conducted within the file while it is in memory. When the file will not fit in memory, we

المنارة للاستشارات
www.manaraa.com

conduct an external search using a mechanism that resides outside the file [Gulu02]. The indexing methods presented in this work are all external mechanisms.

The primary benefit of indexing is that it allows us to search through large amounts of data efficiently without having to examine or in many cases read every item until we find the one we are searching for. Indexing therefore is concerned with methods of searching large files containing data that is stored on disk (data store). These methods are designed for fast random access of data as well as sequential access of the data.

Interestingly, block size and the ratio of the cost of accessing a new block to the cost of accessing items in a block affect performance of inserting data, but the implementation of these methods is largely unaffected by the values of these parameters [Sedg98].

In the context of indexing, we shall equate the concept of a page from virtual memory to that of a block of data on a physical store. For the purposes of discussing indexing mechanisms, page and block are synonymous. Similarly, the concept of a search for a page and seek for a block shall both be referred to as a read. For example, if an index contained references to blocks of data on disk and it required an average of 4 searches of the index pages to locate an item and an additional seek to get the block of data from the data store, we would say the operation required 5 reads; 4 reads of the index pages and 1 read of the data store.

There has been research to simplify addresses to data either in memory or on disk. The most sophisticated examples are those that transform addresses into meaningful or unique keys [Seve76]. However, most systems today provide base addressing mechanisms that do not need such sophisticated transformations.

Although there are many indexing mechanisms available to choose from, the most popular implementations described in literature have been indexed sequential files, hashing, B trees and its variant B+ trees. The following sections describe each of these techniques in detail.

المنارة للاستشارات
www.manaraa.com

5.2.1.1 Indexed Sequential Files

An indexed sequential file is a file designed to hold a tree structure of keys and references to data pages on disk. Figure 5-2 depicts an example ISAM structure containing a set of keys. Note that the tree has a root node that contains a reference to leaf pages that, in turn, contain references or addresses of pages on disk. This structure is easily stored on disk by mapping the nodes to pages on disk.

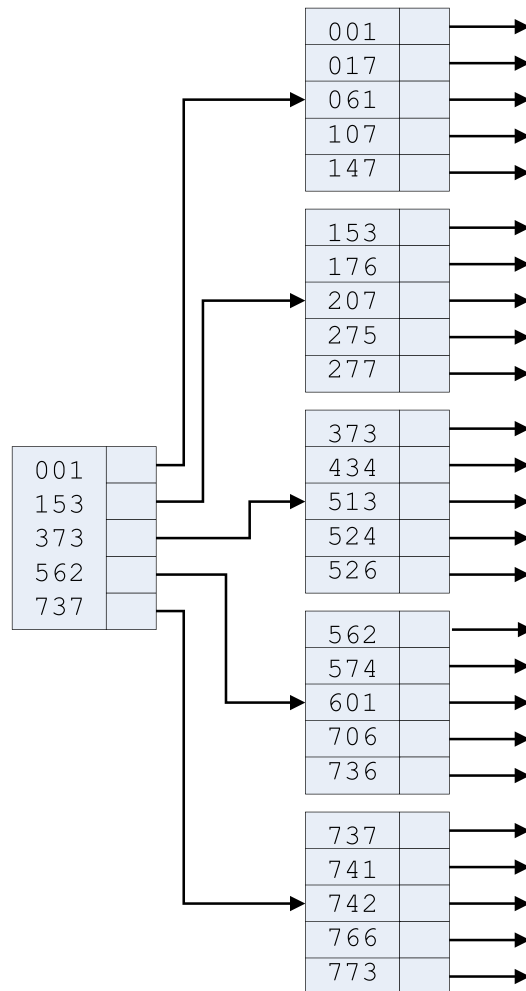


Figure 5-2: Indexed Sequential File Structure

This mechanism is very similar to the disk organization mechanism used in some operating systems. Some early operating systems used a two-level scheme; there the

المنارة للاستشارات
www.manaraa.com

lower (leaf) level corresponded to the pages for a particular disk device, and the level above corresponded to a master index to the individual devices. The top level containing the master index was kept in memory thus limiting access to any device to only 2 reads from disk: once the master index was traversed, one access was necessary to get the lower (leaf) level containing the device reference and another to read the device data from disk. This concept has been expanded to a third and more level hierarchical index in order to store more addresses[4] [Sedg98].

Since this mechanism combines a sequential key organization with that of an indexed access method, the data store on disk that contains the tree described is called an indexed sequential file. The disadvantage of this mechanism is that it is intended for use in environments that do not change frequently consisting mainly of data retrieval. As such, the tree requires reorganization every time data is added to the data store. Mechanisms have been devised to reduce the need for reorganization, such as the use of partially filled nodes and overflow nodes, but none eliminate it [Sedg98].

5.2.1.2 Hashing

Another method of indexing uses hash tables[5]. Hash tables were created for the situation in which the keys that form the index values are not typical range or numerical values, rather they may be values that have either no or limited ordering properties (e.g., strings) [Sedg98, Rama03]. In this case, hashing functions are used to assign pseudo-

---

[4] Sometimes referred to as a directory.
[5] Also called scatter storage [Knut97] and hash addressing [Date04a] .

www.manaraa.com

randomized values to the keys, which are guaranteed to produce a unique value[6] and/or detect collisions on duplicate values, providing a mechanism to store the resultant variant value. Collision resolution techniques include bucket chaining and similar multi-valued lists [Knut97].

The hashing function is an arithmetic function based on the original value of the key. For example[7], a hash function could take an arbitrary character string and convert each character to a numeric value based on an enumeration list, e.g. RED = 0, WHITE = 1, BLUE = 2, etc. Two major types of functions work best. One is based on the use of integer division (e.g., modulo), the other is based on multiplication [Knut97].

Hash function results are not normally associated in an ordered manner. Whereas the result of the hash function is typically an ordinal value and can be ordered, it is usually desired to keep the original values used to calculate the hash value in a different order. Methods to preserve this order by using specialized calculations [Garg86]. Interestingly, these implementations require less storage than other indexing methods (e.g., B+ trees), but are limited in their implementation due to the complexity of the hashing function. A similar mechanism for ordering hash tables is the use of Trie hashing [Litw81]. This mechanism stores the data in the order desired without affecting the performance of the hash itself.

Storage of hashed values is typically done in a multidimensional array, which makes it a prime candidate for in-memory use as well as for persistent storage. When

---

[6] The study of hashing functions is beyond the scope of this work. Research and literature exists that fully explains the use of hashing functions, collision resolution, and uniqueness properties [Corm01].
[7] Trivial perhaps, but valid.

المنارة للاستشارات
www.manaraa.com

storing the hash table in a data store, the array itself is treated as a file where each row of the array is written to a different block or by storing several rows in a block of arbitrary size. The array is then read from disk and loaded into memory as either a multidimensional array or as an array of linked lists.

Hashing illustrates the classical computer science solution to a space versus time tradeoff. On one hand, if there were no memory limitations and indexes could be stored indefinitely in secondary storage, indexing can be made very fast by simple manipulation of in-memory data structures and memory references. On the other hand, if there were no time limitations[8], then one could employ very sophisticated indexing mechanisms that permit the searching and location of any datum in a vast amount of data. Hashing provides a way to use a reasonable amount of memory (the memory needed to store the resulting hash function values and the reference (pointers) to data on the data store) and time to execute and traverse the index, which requires the calculation of the hash value and looking up the reference in the hash table. In fact, it is possible to tune the hash table for hierarchical traversal[9] using cascading hash functions, which permits the storage of portions of the hash table as well as simulating a tree-like structure of smaller hash tables [Sedg98].

The primary advantage of using hash tables for indexing is the fast retrieval of the index value and, if implemented correctly, the substitution of the hashing function for the more traditional traversal mechanism for tree- or graph-based structures.

[8] That is, the tim e to execute a query and return results is not important.
[9] Found in the literature as modular hashing [Sedg98].

www.manaraa.com

The features and performance characteristics detailed above make hash tables a popular choice for database implementers. In fact, hash tables are so utilitarian that they are often employed to solve computer science problems of system structure and execution. For example, the ALV indexing mechanism and the clustered version store use hash tables for fast lookup of frequently used values such as attribute lists and in-memory block (page) lists.

One variant of hashing worthy of note is called extendible hashing [Fagi79]. Extendible hashing guarantees no more than two page faults to locate data associated with a given key. Unlike conventional hashing, extendible hashing has a dynamic structure that permits the hash buckets to expand, or extend, to accommodate growth of the hash table. Fagin's results show that extendible hashing could provide an alternative indexing mechanism.

5.2.1.3 B Trees

The B tree was described in the seminal work by Bayer and McCreight [Baye72] who were the first researchers to consider the use of multiway balanced trees for external file searching. Most use the term B tree to refer to the algorithms suggested by Bayer and McCreight [Sedg98]. B trees have been used in a wide variety of domains including spatial databases, multimedia databases, temporal databases, and object-oriented databases. Each of these domains requires an index structure that is specially designed and tailored for the domain. Interestingly, in each of these domains methods have been

المنارة للاستشارات
www.manaraa.com

used that are based on one distinct structure – the representative B tree-based structures and their search operations [Ooi01].

A great deal of effort has been put into improving fan out[10], which minimizes the height of the tree, leading to faster searches of the tree for keys. Research has also been conducted to study page organization. Results have shown that the various techniques of reducing page size, and therefore the internal node size, can have significant effects on performance [Lome01]. The majority of these advances have been incorporated into implementations of B trees [Corm01, Sedg98].

A B tree of order[11] *m* data structure has the following properties [Baye77a, Gulu02]:

- Every node has at most *m* children.
- Every node, except for the root and the leaves, has at least (*m/2*) children.
- The root has at least two children, unless it is a leaf.
- All leaves appear on the same level and carry no information (data).
- A nonleaf node with *k* children contains *(k-1)* keys.
- A leaf is a terminal node (one with no children).
- All paths from the root to a leaf node have the same length (i.e. Height balanced)
- All non-leaf nodes contain elements (reference keys) which are ordered

[10] Fan out is used to describe the operation that occurs when keys are added to nodes that are already full. The basic strategy is to minimize t he number of nodes at any given level by keeping the node's key list at least half full.
[11] Order refers to the number of keys contained in the node

المنارة للاستشارات
www.manaraa.com

Comer explains B trees best; "In general, each node in a B-tree of order d contains at most 2d keys and 2d + 1 pointers. Actually, the number of keys may vary from node to node, but each must have at least d keys and d + 1 pointers. As a result, each node is at least 1/2 full. In the usual implementation a node forms one record of the index file, has a fixed length capable of accommodating 2d keys and 2d pointers, and contains additional information telling how many keys correctly reside in the node," [Come01]. Figure 5-3 depicts a conceptual B tree with sample keys and physical references.

When used as indexes for database systems, B trees provide many query operations: equality queries which ask the question of what entity has a particular value for a particular indexed attribute, minimum and maximum queries which ask what entity has the most/least value of a certain indexed attribute, and range queries which ask questions such as what entities have a value for an indexed attribute within a given range of values. These operations are easier because B trees contain the value of the attribute that is indexed within the tree structure and may not require retrieval of the entity from the physical store [Tuck04].

Since B trees are typically stored in memory, their performance has been shown to be approximately 5 ms[12] to search the tree [Tuck04]. This value is an average and factors in the likelihood that at least one read from the physical store is necessary on average. The single greatest performance issue with B trees and their variants is the height of the tree and fan-out. The goal is to construct the tree with minimal height and

[12] Tucker et. al. make the assumption that most hardware can support this timeframe [Tuck04].

المنارة للاستشارات
www.manaraa.com

with an optimal fan-out scheme that permits a balance of inserts and deletions in order to minimize the need to balance or reorganize the tree.

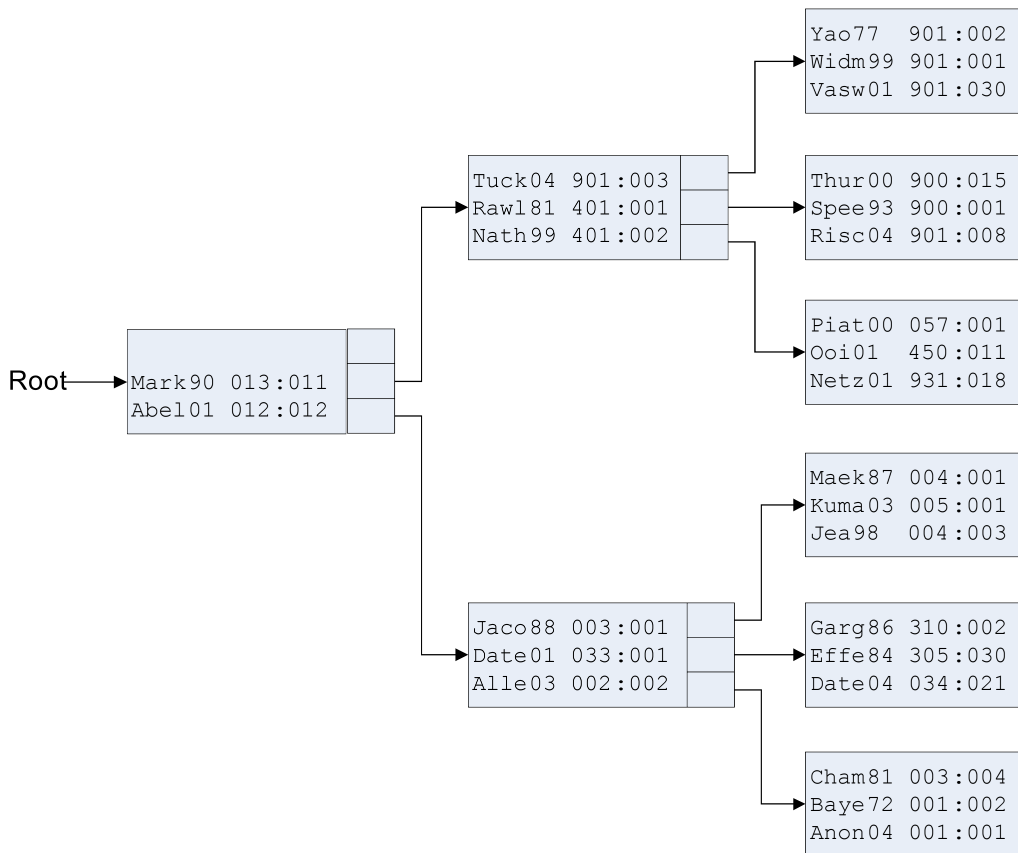


Figure 5-3: Conceptual B Tree with Block Addresses

The best quality of the B tree is that it provides mechanisms for the insertion and deletion operations to automatically balance[13] the tree. With only a small penalty in performance, these mechanisms are guaranteed to minimize worst-case access time [Held78, Silb96]. This ensures that the tree will always perform on average $\log_d n$ where n

[13] Also called "self reorganizing," [Tuck04].

المنارة للاستشارات
www.manaraa.com

is the number of keys and d is the order of the tree [Come79]. Figure 5-3 depicts a typical B tree.

Notice in the example that each node contains keys that have addresses associated with them. This implementation of storing physical storage addresses with keys, albeit conceptual, was revolutionary when employed in early database systems. Since then, the B tree has been explored and modified to fit a host of applications. This research has produced a number of unique variants.

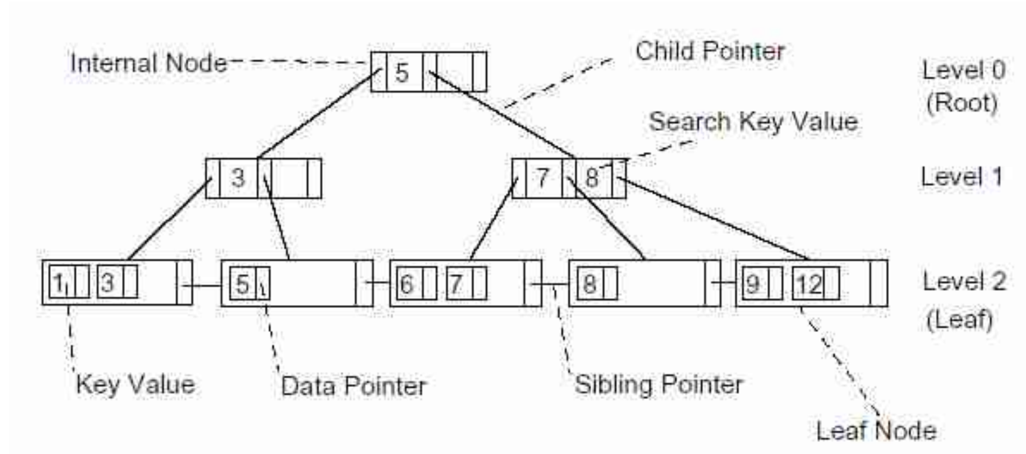


Figure 5-4: B+ Tree Configuration

An early variant of B trees was a B tree designed to store only a prefix of a key value in order to reduce the number of insertions during add operations[14]. This variant was called a prefix B tree and has been shown to increase performance when the range of key values have a large number of values that have the same prefix [Baye77].

Perhaps the most misunderstood variant of the B tree is the B* tree which is used by Oracle [Mcke01]. Knuth defines a B* tree as a B tree with each node at least 2/3 full

[14] In SQL-speak, "INSERT."

www.manaraa.com

versus the convention of 1/2 full [Knut97]. This is accomplished by the balance methods used during insertion and deletion. This variant of the B tree is often confused with another type of B tree, also suggested by Knuth, the B+ tree [Come79].

5.2.1.4 B+ Trees

B+ trees are the latest approach to providing indexes [Elma03]. B+ trees are more sophisticated than hash tables. They attempt to solve the problems of not knowing how many buckets might be needed, and that some collision chains might be much longer than others. They attempt to create indexes such that all rows can be found in a similar number of steps through the storage blocks.

With the B+ tree, the order of the original data is its creation order. This allows multiple B+ tree indices to be kept for the same set of data records. Although academic implementations of B+ trees store the actual data in the leaf[15], it is far more efficient to store references to the data in the leaves and access the data via a dereference rather than storing the information directly in the B+ tree physical store (file). This permits the implementation of a superior physical store (file) for the data and (possibly) a different mechanism optimized for B+ trees. This permits one to persist the B+ tree to the physical media, thus preserving its state and functionality until needed [Lank91, Tuck04].

A B+ tree is a B-tree with certain improvements that permit efficient searches both sequentially and randomly. All references to the data are stored in the B+ tree's

[15] Actually, there are many subtle differences of what a B+ tree really is. Navathe and Elmasri agree on one form while Knuth and others agree on a slightly different form. Most of these differences ce nter on the idea of fullness (1/2 vs. 3/4) of the nodes, others center on whether the tree should include the actual data or just the reference to the data on the physical media.

المنارة للاستشارات
www.manaraa.com

leaves, with only a few of the keys duplicated in the branch nodes [Haer78]. By convention, a B+ tree has leaves that are always at least half full—a new key enters the nonleaf part of the tree whenever a leaf splits [Gulu02].

Some of the characteristics of B+ trees are (see figure 5-4 for nomenclature):

- The lowest level in the index has one entry for each data record.
- The index is created dynamically as data is added to the file.
- As data is added the index is expanded such that each record requires the same number of index levels to reach it (thus the tree stays `balanced'). Likewise, deletion may require rearranging nodes that become empty[16].
- The records can be accessed via an index or sequentially.
- Each index node in a B+ tree can hold a certain number of keys.
- The B+ tree is called a balanced because every path from the root node to a leaf node is the same length. A balanced tree means that all searches for individual values require the same number of nodes to be read from the disk [Mcke01].

Deletion in a B+ tree can be more complicated than for B trees due to the way references are stored in the leaves. Efficient algorithms for deletion are available that solve the initial problems of balancing the tree [Jann95, Mael95]. Although B+ trees

[16] Wirth notes that the deletion balance problem is more sophisticated than in sertion [Wirt76]. Some implementations simply do not permit rebalancing when deletion results in an empty node in a B+ tree. Instead, the nodes are left empty and only the keys are moved if necessary. This permits a tree to increase in size, but does not permit the tree to reduce.

المنارة للاستشارات
www.manaraa.com

perform well for searching, they do require additional space (memory, disk) and resources (CPU, memory) to maintain [Ho04].

Figure 5-5 depicts a conceptual B+ tree with keys and physical references displayed. Although it is possible that the internal nodes of a B+ tree could contain values that are indeed keys, it must be noted that this does not have to occur and, in fact, only occurs when the tree is built from existing data. Once the process of adding and removing data occurs, the internal nodes will no longer maintain copies of the key values [Come79]. Comer writes, "To fully appreciate a B+-tree, one must understand the implications of having an independent index and sequence set. Consider for a moment the find operation. Searching proceeds from the root of a B+ tree through the index to a leaf. Since all keys reside in the leaves, it does not matter what values are encountered as the search progresses as long as the path leads to the correct leaf."

Furthermore, the leaves also linked together giving the ability to easily iterate through a physical store without the need for repeated traversals of the tree. This gives the ability to iterate through a range of values starting at any arbitrary point. There has been an implementation of B+ trees that provides doubly linked leaf nodes, giving the ability to iterate both forward and back through the index keys [Baye72].

B+ trees also provide the advantage of using dynamic allocation and release of storage and a guaranteed utilization optimal of 50%. These features and power of the structure makes B+ trees well suited for database applications that require sequential and random access to physical data stores.

المنارة للاستشارات
www.manaraa.com

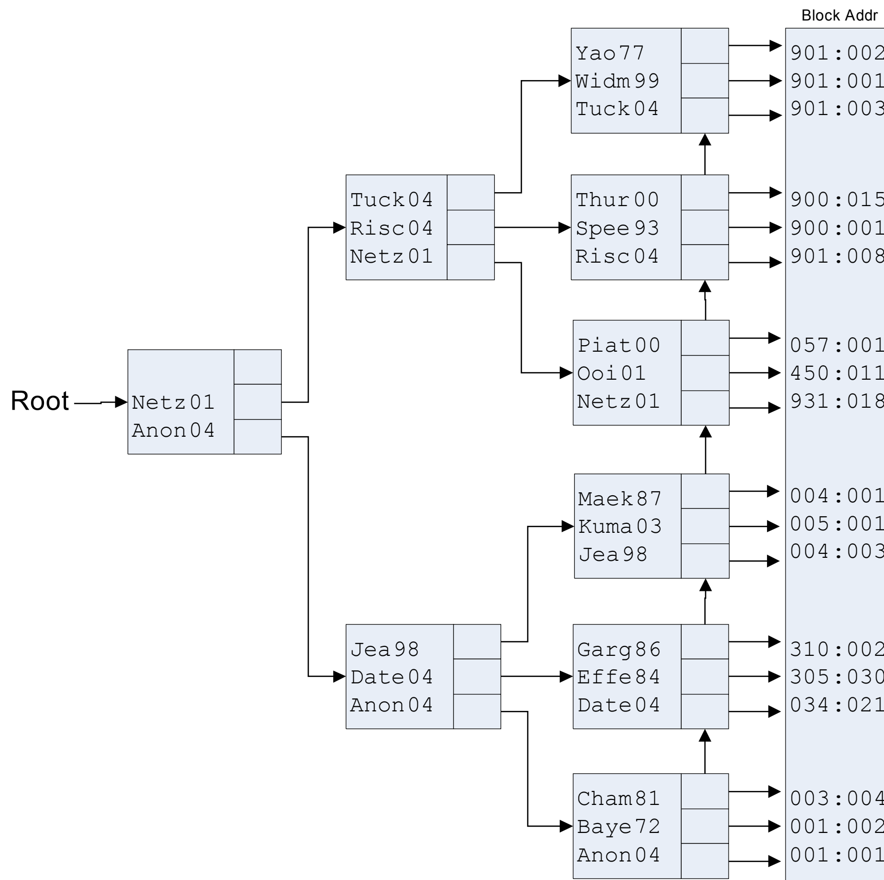


Figure 5-5: Conceptual B+ Tree with Block Addresses

The use of B+ trees varies among database vendors and implementers. However, there are two general approaches with respect to what is stored in the B+ tree and how it is used to access data.

The first approach is to maintain physical addresses in a separate data structure on disk. This approach is used in Sybase and early versions of Microsoft SQL Server[17]. The major advantage of this approach is that it requires only a single read from the physical

[17] Microsoft purchased Sybase and ported it to the Windows platform and renamed it SQL Server. Since then, many changes have been implemented to enhance performance and scalability.

www.manaraa.com

store to retrieve the desired data. The disadvantage is that a single insert can cause a split in the tree and cause up to half of the blocks of data to be moved in order to maintain order within the tree [Chon01].

The second approach is to maintain logical addresses. This approach is used in Compaq NonStop SQL on key-sequenced files and later versions of Microsoft SQL Server[18]. The major advantage is that the logical references to blocks need not change when data is inserted – they can be simply remapped to their new origins. The disadvantage is that at least one additional traversal of the tree is necessary once the datum is located [Chon01].

The approach taken in this work is similar to the second approach where the logical address of the data on the physical store is stored in the leaf nodes of the tree. By storing a logical value in the form of Block:Offset, the system need only traverse the tree once to obtain the logical address, then instruct the physical store to retrieve the desired block. Researchers have studied addressing schemes for divorcing the physical address from index stores for some time [Cook78]. The generally accepted mechanism involves that shown above where a logical block address and an offset or row address is stored in the index then later translated by the physical store access layer.

5.2.2 Concurrency Issues

A considerable amount of time during database access to physical stores is spent searching indexes for references to the data on the physical store. The B tree and its many

[18] In this case, we refer to a clustered index.

www.manaraa.com

variants have become the most popular of all of the indexing mechanisms [Date04a]. However, maximizing their effectiveness in concurrent operations is perhaps one of the most difficult tasks [Bili87, Neub99]. Concurrency is an essential element of any database system. These systems are used by multiple users running multiple processes to access data that may also be being accessed by other users and processes.

Concurrent operations on B trees pose the problem of insuring that each operation can be carried out without interfering with other operations being performed simultaneously by other processes. If data is updated frequently, B trees can become a concurrency bottleneck because all access to the tree begins at the root [Rama03]. If the root node is locked with an exclusive lock, no other processes can use the index. This problem may create unacceptable bottlenecks which become critical if these structures are used to support highly contested access paths[19], like indexes (metadata) to a database system. Thus, there is a need for locking protocols which assure integrity for each access and concurrency for the system. Also, since the cost of resolving deadlocks may be high, the solution should be deadlock-free [Baye77a].

Most of the solutions to this problem implement a technique of locking the affected nodes during write operations and permitting any number of reads to occur. Some of the more popular solutions use strategies such as logical undo logging, rollback, checkpoints, restart recovery, and fuzzy checkpointing [Silb96]. Some implementations lock the entire tree during writes while others attempt to lock only the portion of the tree

[19] An access path is comprised of all of the execution sequences, algorithms, and data acquisition mechanisms which must be taken in order to search the database and retrieve the data requested by the user. These may include such operations as searching index es, tracing linked lists, or sequentially scanning the data for the requested information. A goal of good database system design is to provide efficient access paths to access data [Yao78].

المنارة للاستشارات
www.manaraa.com

that is affected [Bili87, Held78, Lehm81]. However, locking the entire tree has been shown to perform poorly in a multi-user environment with a high degree of concurrent access [Srin93]. Locking only a portion of the tree has its own problems in that it is not always necessary to lock the nodes below the node being locked. Likewise, unless an update causes the balance operation to "push up" key values to form a new node, the nodes above a node affected may not need to be locked. Mechanisms described by researchers have shown that it is possible to write algorithms that can handle these situations. The version index mechanism described in this chapter uses similar mechanisms.

One solution treats the leaves as a separate section so that when a process traverses to a leaf, the locks for the tree are released and a lock is then applied to the leaf node only [Jong90]. In the case of an insert operation, the subtree need be locked if and only if the insertion requires a rebalancing of the tree beneath it. This is complicated when the rebalancing operation requires pushing keys up the tree [Rals03, Tuck04]. Many algorithms used today avoid this complication by implementing a strong reader version of a reader/writer monitor [Ben90, Maek87]. In this situation, a process must request the write lock and wait until all other active readers complete their operations.

It must be noted that the concurrency operations on a B+ tree is susceptible to disk reorganization strategies and could invalidate the physical address pointers. Thus, it is always necessary to reorganize (or rebuild) the index whenever the physical store is reorganized, but not the other way around. Reorganizing the index does not affect the

المنارة للاستشارات
www.manaraa.com

physical store addresses. Thus, it makes it much easier for the database implementer to create and drop indexes on the fly.

Another concurrency access technique that is popular is that of batch updates. This mechanism saves updates to the tree in batches so that a series of updates is done all at once [Poll96]. In his work on postorder B tree construction, West adds, "Typically, no distinction need be made between building a tree versus adding a key to an existing tree. [West92] Some benefits can accrue if building a tree is regarded as a process distinct from adding keys to a previously constructed tree," [West92].

Implementations of this include variations that use a strong reader biased concurrency mechanism that permits a batch of updates to be completed only after all readers are finished. While this does solve many of the problems of concurrent access, it can lead to race conditions where as long as there is a reader, all writers are held. Even if a more sophisticated reader/write were used, the performance in an environment with many users remains to be demonstrated. As a result, this method was not implemented with the version index presented below.

5.2.3 Transaction Processing Issues

A transaction is a set of database operations designed to be executed such that all of the operations succeed or none do. If any of the operations fail, all of the other operations and their affect on the system must be discarded, hence making the transaction an atomic operation. Operations that retrieve data do not affect values in an index, but write operations do. The concurrency control system should control the concurrent

المنارة للاستشارات
www.manaraa.com

execution, so that the computer resources will be used as efficiently as possible while preserving the atomicity [Mond85].

A database system using B+ trees [Knut97] for indexing must provide mechanisms to permit transactions. These operations must be capable of marking nodes that have been changed by operations in a transaction, saving the original values of the data that has changed until such time that the transaction is complete. At this point, all of the changes are committed to the physical store (for both the index and the data). This is most problematic when updates (*i.e.*, inserts or deletes) force the tree to rebalance. In this case, the both the original form and values for the tree must be preserved.

Early research has solved the problem by locking the entire tree and waiting until the final commit is given to proceed. This practice has the side effect of slowing performance down considerably. In an environment that has many transactions executing simultaneously the system defaults to a single process queue [Mond85]. Later research has provided mechanisms to shadow the changes to a tree and has incorporated the locking process into the concurrency access methods. The version index described in this work has adopted a similar concept for transaction processing.

5.2.4 Performance Issues

The use of B+ trees as a file indexing mechanism is one of the primary advantages of B+ trees in that reorganizations are unnecessary, the algorithms are simple and easy to implement, and performance is good even under adverse conditions. It is interesting to note that the performance of B+ trees has not always been considered best.

المنارة للاستشارات
www.manaraa.com

Excepting for the moment the need for reorganizations, indexed sequential files also have simple algorithms and can have good performance. In fact, early comparisons of B+ tree and index sequential file performance showed that B+ trees performed less efficiently. Although the researchers noted that the conclusion could be application and even platform dependent, the results were still favorable for using indexed sequential files [Bato81].

Normal creation of the B+ tree index is done through insertion of keys and data references as data is added to the physical store. The issue is how to create a B+ tree from existing data. There have been attempts to optimize this process, but most database systems assume a certain penalty for creating a B+ tree from existing data. An algorithm, called batch-construction, "…inserts key values into a B+-tree in a random order without considering adjacency between them. This makes each page within the B+-tree accessed frequently, and thus, incurs large overhead for constructing the B+-tree. [This] algorithm gracefully solves this problem by processing all the key values to be placed on each B+-tree page simultaneously when accessing the page," [Kim01]. This algorithm is shown to be best for bulk loading a large database that has an enormous volume of key values. The problem of fast construction of the B+ tree is a concern that has not yet been addressed by the version indexing mechanisms presented.

5.3 Version Indexing

Preserving the relationship of the entity in the host table to the attribute versions in the version store requires an indexing mechanism that is capable of storing and

www.manaraa.com

indexing the key attributes from the host table and relating them to the attribute version chains in the clustered version store. This is necessary in order to preserve the connection from the host table to the version store and to provide a way to retrieve attribute versions for a particular set of entities. When a version store is created for a host table, the host table is a versioned table, or has been versioned.

The version index is a companion mechanism to the clustered version store. The version index is created when the clustered version store is created and does not require any additional operations on the part of the database professional. That is, unlike most database management systems, the ALV extensions to MySQL permit a single CREATE SQL command to not only create the table (clustered version store), but also to create the index (version index) at the same time.

The version index was created using a B+ tree as the base data structure. The decision to use a B+ tree over the many other indexing mechanisms stems from the fact that the B+ tree is the best structure in dynamically changing environments, and B+ trees provide very good performance throughout a variety of uses. Hashing was not used for the same reason presented in [Cham81a], "…because it does not have the convenient ordering property of a B tree index."

One of the areas of concern for implementing an indexing mechanism for the clustered version store was how it would perform in comparison to a storage mechanism that does not use clustering. That is, it was desired that the index be created along with the table using a similar block format that avoided incompatibilities or competition while reading from disk. Furthermore, it was important to ensure that the index would require

المنارة للاستشارات

www.manaraa.com

far less space to store than the original data file. Researchers have concentrated on this area by focusing on the ordering of the keys in the nodes [Dong82]. However, most database systems have addressed this in other ways. For example, Microsoft SQL Server 7.0 was rebuilt to accommodate a different storage engine that uses smaller page sizes in the effort to reduce the amount of space used by tables and indexes on disk [Dela04].

A mechanism developed at IBM predating B+ trees, which is very similar, is called a virtual storage access method (VSAM). In VSAM files, like B+ trees, there is a deliberate attempt to keep a certain amount of space, called distributed free space, in the index for insertions. VSAM, unlike traditional B+ trees, provides a way to do direct addressing rather than chaining to resolve addresses in the physical store [Keeh74]. The version indexing mechanism described in the following sections has been designed to include many of these features.

Related work on adopting a B+ tree for use in a versioning scheme was presented by Lanka and Mays [Lank91]. Their implementation was primarily used to segment the B+ tree into several parts, thereby allowing for faster searching and for version retrieval. Some of the techniques presented in their work were used when designing and implementing the version store. However, in this case, the version store is designed to work with a specialized physical storage layer, the clustered version store, in order to index attribute version chains. An extension of Lanka's and May's concept has taken form in a variant of the version index that permits multiple hits per attribute version key (see section 5.3.1.1).

المنارة للاستشارات

www.manaraa.com

The hB tree presented by Lomet and Salzberg provided inspiration for the creation of the version index mechanism [Lome94]. In their work on the hB tree, Lomet and Salzberg presented many of the arguments for using a B tree in forming an index that stored versions of attributes. However, rather than storing the attribute versions in the tree itself, the version index mechanism leaves the attribute versions in the physical store storing only the reference to the data on the physical media. This makes the application of an advanced structure like the hB tree unnecessary and permits the use of more traditional and less complex mechanisms like the B+ tree that is design to store the physical addresses at the leaf-level.

5.3.1 Technology Description

The version index is implemented as classes within a C++ program. All aspects of the operation of the version index are abstracted and represented as classes. Starting from the lowest level, a class was created to model a B+ tree node which is then contained in class that implements the tree structure. Access to the index is via classes similar to the clustered version store. The structure includes a file access layer as well as a buffer manager and ALV manager tie-in for concurrency locking and transactions. These last two features are paramount to the execution of the version index and add considerable benefit to the effectiveness of the version index.

المنارة للاستشارات
www.manaraa.com

5.3.1.1 B^2+ Tree

Much of the research on B+ trees was focused on making the structure usable in concurrent and transaction processing environments. While there has been much success in these areas, there isn't any implementation that considers concurrency and transactions as design goals for the internal workings of the data structure (the tree) itself. Furthermore, much of the research on caching in databases concerns adding support at a layer above that of the index structure. Figure 5-6 depicts the layout of the index structure.

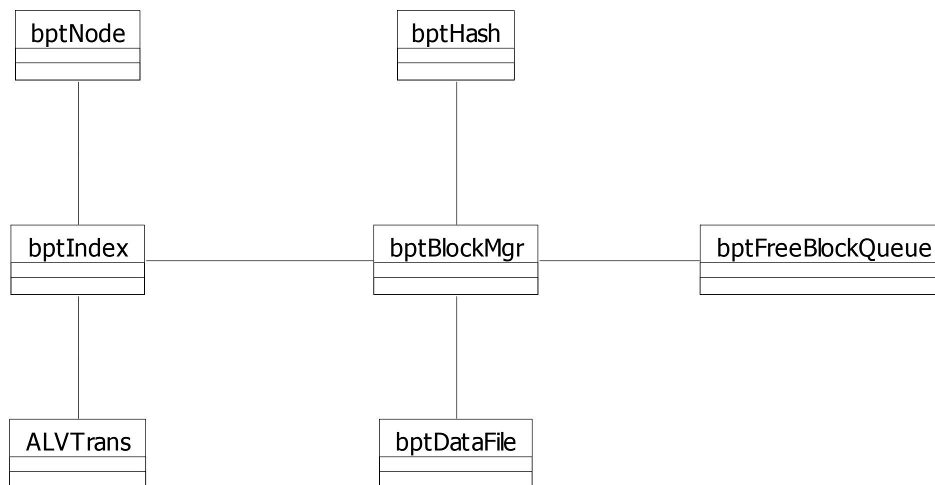


Figure 5-6: B^2+ Class Diagram

The best way to ensure success in any of these areas is to build these features into the tree structure itself. A new variant of the B+ tree was created that has support for concurrency, transactions, and buffering (caching) built into the data structure and its access methods directly. What is needed is a buffered B+ tree that supports concurrency and transaction. This new variant is called a B^2+ tree.

www.manaraa.com

The B^2+ tree is constructed using a similar buffering technique to that of the clustered version store. In fact, the base class that implements the ALV buffer manager is used to implement the buffer manager in the B^2+ tree. Like the clustered version store, the B^2+ tree contains properties such as a status variable that records the state of the node (locked, unlocked, etc.) embedded in the tree node and in the physical store layout to accommodate the buffer manager technology. The buffer management mechanism therefore is designed to buffer the blocks of data that form the nodes for the tree.

Whereas the buffer manager was added to the access methods for the physical store of the tree, concurrency and transactions are handled in a much more intimate manner. A detailed description of these mechanisms is given in the section 5.3.1.1 below. Each node of the tree contains additional data items to identify the state of the node and to permit the storage of seminal values to indicate the last known good operation for use in transactions. Together with the buffer manager, concurrency and transactions can be supported natively with these internal modifications.

The initial use of the B^2+ tree was to store a single reference key as the primary indexed value. The B^2+ tree contained all of these keys and the leaves also contain the physical store address for the first block in the clustered version store that contains the attribute version chains. This is a typical use of a B+ tree in a database environment. What makes it unique, besides the buffering, concurrency and transaction support, is the fact that the index formed by the tree is an index of pointers to attribute chains rather than data itself. Although the attribute chains do contain data in the form of attribute versions, the index retrieves the data for the attribute chains and the traversal of the attribute chains

المنارة للاستشارات
www.manaraa.com

is left to the clustered version store. Thus, the implementation of the B^2+ tree is an index for optimizing the retrieval of attribute versions for a given set of entities in a host table. The greatest benefit of this mechanism is that it performs all of the operations of a B+ tree, allowing for many types of query operations, *e.g.*, range queries, sequential access, etc.

The storing of a single key reference to the host table isn't effective for complex table types. A superkey[20], a single valued key that artificially defines uniqueness, can be used to avoid the use of complex keys. However, it should not be necessary use superkeys when the nature of the database is a model of the real world. That is, if portions of a complex key can be used to define hierarchies or classes, and the desire is to form versions of these classes and hierarchies as well as the entities contained within, the need to version data based on the complex key is essential.

One concept that the B^2 tree supports, that most tree-based indexes do not, is the concept of concatenated keys [Wagn73]. This method was first used in VSAM indexing mechanisms. It permits the formation of an index based on multiple key parts to be treated as a single key. When a version index is created for a multi-part key reference, the B^2 tree concatenates the keys together to form a composite key that is stored in the normal location in the tree. Searching for matches on a composite key where the search criteria contain all parts of the key is trivial. Searches that contain only part of the key are more difficult and require resolution to know when the user is asking for the first part, middle, end, or random portions of the key. The current implementation of the B^2 tree

[20] Also called a surrogate key [Date04].

المنارة للاستشارات
www.manaraa.com

permits partial key matches starting at the left-most portion. For example, suppose a composite key is made up of four parts, a.b.c.d[21]. The B^2 tree will permit searches with partial keys of a, a.b, a.b.c, and a.b.c.d (the trivial partial key). This mechanism, albeit somewhat more primitive than what has been implemented in VSAM [Wagn73], provides the ability to version a table having a complex key without the need to use super keys.

5.3.1.2 *m*B^2+ Tree

What if a database user wanted to know which entities in a database had a certain attribute version (or set of attribute versions)? The version index described above will not help and the search will result in a simple table scan. This search is not as uncommon as originally suspected. In fact, during much of the data mining algorithm presented in Chapter 7, having the ability to search to find data that has certain attributes has been greatly enhanced.

Conflict resolution on indexes is not a new idea. When resolving the indexes of distributed databases which are, in turn, distributed, the attribute values that are associated with an entity in the system may have different values depending on how each of the distributed nodes are used (updated) [Lim96]. In the case of indexing a clustered version store for the purpose of locating all attribute versions regardless of their association to the original (host) data, the implementation presented here permits the

[21] The dot notation here is for emphasis and has no bearing on the actual data that is stored. Composite construction of the key is performed using object-oriented techniques in the source code such as classes and structures.

المنارة للاستشارات

www.manaraa.com

collisions to be stored in the index. This preserves the association of the attribute value to its host data while permitting fast searching for queries like those described in Chapters 1 and 3.

The mB^2+ tree was created to solve this problem. Figure 5-7 depicts a mB^2+ tree with one of the leaf nodes expanded to show how the multiple references are stored and accessed. The subscript *m* in the name indicates the tree can store multiple references for a given entity key value. This tree is a variant of the B^2+ tree and has all of the features that mechanism supports while simultaneously storing multiple values for the values stored in the leaves. Implementation of the mB^2+ tree was simply a modification of the node structures and access methods to add the features necessary to store and iterate through the multiple values. Thus, the mB^2+ tree and the B^2+ tree are the same implementation in the source code and differ only in their use. A mB^2+ tree is used to index all of the attribute versions by storing the reference key to the entity and the B^2+ tree is used to index the attribute version chains for all of the entities in a host table[22].

Performance of the mB^2+ tree is the same as that of the B^2+ tree for retrieving the first value stored in the leaf, but requires a linear retrieval to iterate through the linked list of physical store addresses. However, since many of the uses of this index will be query operations such as returning all of the entities that have the following set of attributes, the index is best used to simply retrieve the keys for all of the entities matching the search criteria which can be used in turn to search the host table for more information or to use

[22] One could say that the MB2+ tree is an inverted B ^{2}+ tree.

المنارة للاستشارات
www.manaraa.com

the resulting list of keys to search the B^2+ tree for all of the values for the particular attribute version in question.

In the implementation of the ALV system, the B^2+ tree is considered the primary index and the mB^2+ tree is considered the secondary index. The mB^2+ tree requires an additional query command to create and is therefore an option for database professionals to optimize the ALV version system.

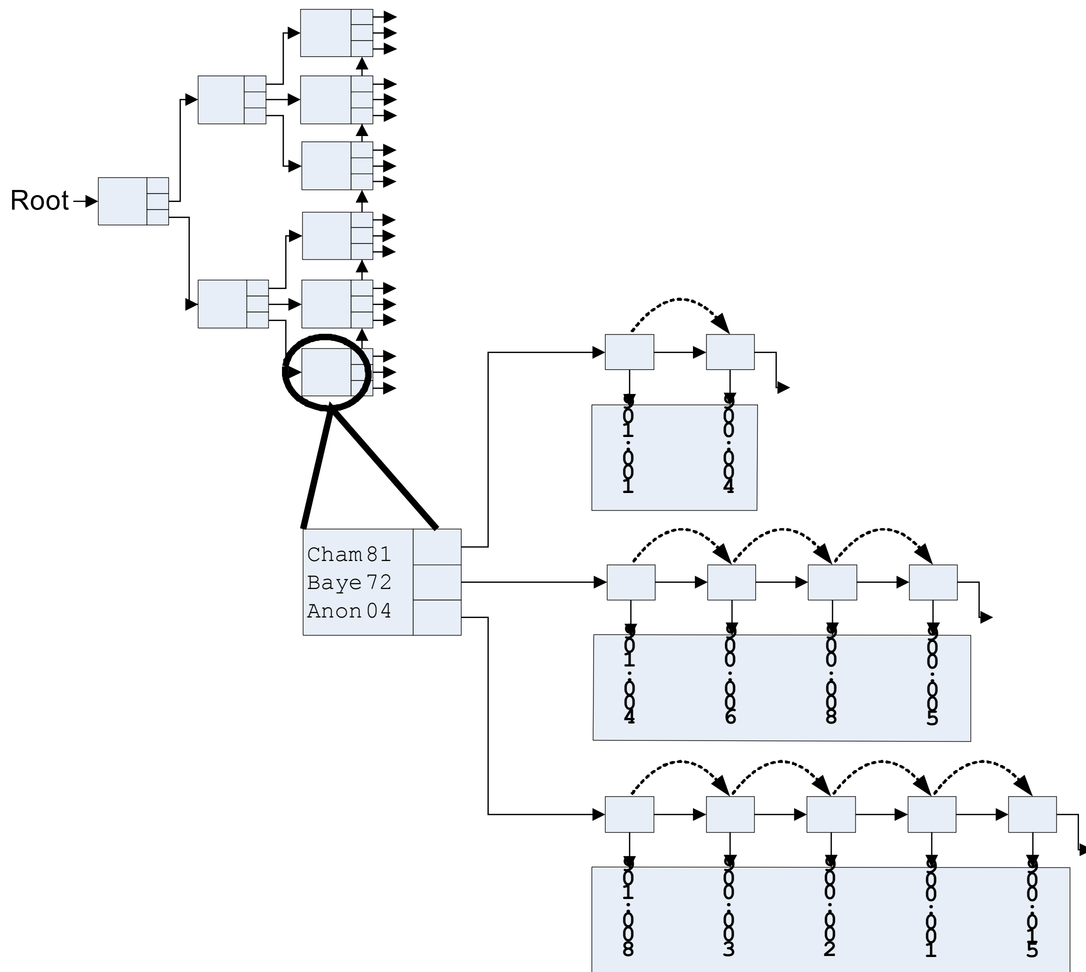


Figure 5-7: mB^2+ Tree Node View

المنارة للاستشارات
www.manaraa.com

5.3.2 Execution Sequence

The execution of the ALV system follows the same model as that of MySQL. That is, it is a multithreaded server application where each command is given its own thread of execution. Once the thread is created, control is passed to the parser where the SQL statements are parsed and directed to the appropriate execution method. A very large case statement is used to contain all of the possible execution methods for all of the available commands.

In order to integrate tightly with the MySQL system, the parser was modified to include catches for special ALV keywords. The location and type of keyword identified will cause the MySQL system to redirect commands to the ALV system for processing. For a complete explanation of this technique, to include the execution sequence from the parser to the ALV system and its implementation, see Chapter 6. For a more in-depth study into how the MySQL code was modified, see the Appendix B.

What is of interest is the execution sequence during physical store access using the version index to retrieve a reference to the attribute version chains for a given set of entities. Figure 5-8 depicts a conceptual flowchart of how the index is used to retrieve data.

First, the system checks to see if an index is available, which it always will be for the primary index. The system returns a pointer to the index class (executable). Control then passes to the ALV_Manager overseer which checks to see if the index has already been loaded into memory and if not loads it into memory (the root node). Next, the index is traversed down to the leaf (requiring $\log_d n$ time, where n is the number of keys and d

المنارة للاستشارات
www.manaraa.com

the order of the tree). The index either returns NULL for no match or the reference requested. It should be noted that the B^2+ tree performs this execution in the same manner used by all B+ trees in database systems. The mB^2+ tree differs only in that it permits the iteration of a list of reference values.

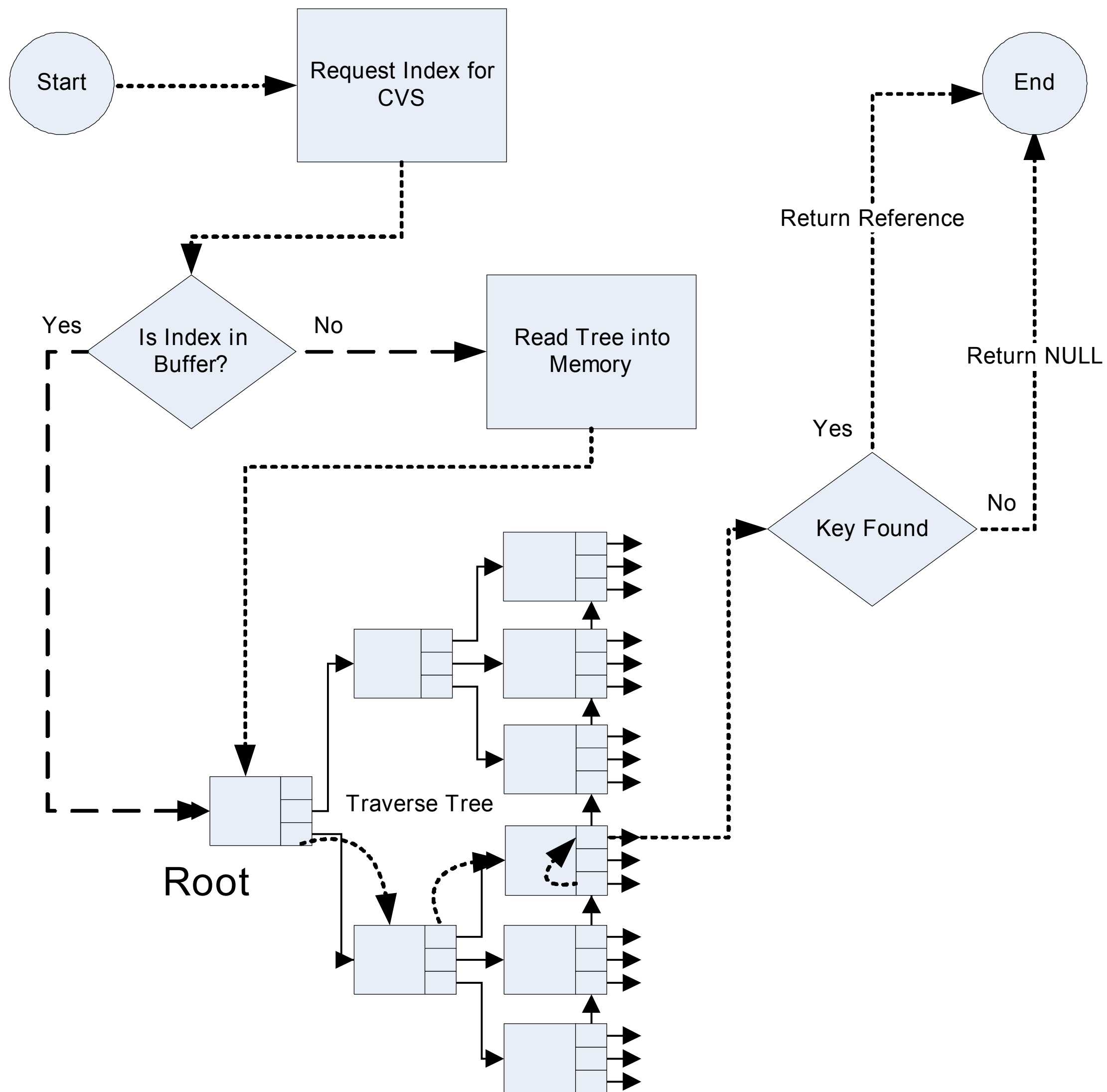


Figure 5-8: Execution Sequence for the B^2+ Tree

المنارة للاستشارات
www.manaraa.com

5.3.3 Class Descriptions

A great deal of information is available for the implementation of B trees and B+ trees [Anon04, Date04, Elma03, Rals03, Rama03, Silb96, Tuck04]. Thus, it was not necessary to reinvent the data structures that make up a typical B+ tree. The only difficulty was creating an implementation written in C++ that supported the concept of storing the reference values in the leaves and leaving deleted key values in the nodes as reference in order to minimize balancing of the tree.

The following sections describe the major code implementations and classes created to implement the version index. Figure 5-6 depicts a high level class diagram for the code implementation of the B^2+ Tree. They are presented in order starting from the internal representation of nodes to the arrangement of the index on disk.

5.3.3.1 bptNode

The bptNode class is used for structuring a tree node. It is used to form a B^2+ tree (or mB^2+ tree[23])-based version index. All of the internal mechanisms and data structures are hidden from the caller (the bptIndex class) and the external code does not need to be concerned with where items are stored in the node when it is stored on the physical media. This permits a flexible storage mechanism that is modularized and therefore can be maintained or altered without requiring the host (bptIndex) to be modified or rebuilt.

The bptNode has two states, a leaf node or an internal node. This is set when the node is created. The differences between the two states are:

---

[23] It should be noted that a B^2+ tree could be considered a mB^2+ tree with the maximum number of keys in each reference set to 1.

المنارة للاستشارات
www.manaraa.com

- Leaf nodes store an array of (key, value) pairs, where value[i] goes with key[i]. Both key and value have a fixed size and type for each tree, according to possibilities in bptKeyAndValueTypes.h.
- Internal nodes, store an array of (key, childId) pairs, where childId[i] is a (1-word) bptNodeId that refers to the child with key[i] <= child_key < key[i+1].

Internal nodes do not actually store key[0], which is implicitly equal to the splitting key found in the common ancestor of the current node and its left neighbor (or to minus infinity). They do, however, store childId[0], whose keys satisfy key[0] <= child_key < key[1]. Similarly, when i is the last index used in an internal node, the keys of childId[i] satisfy key[i] <= child_key < right_neighbor_key[0]. If a copy of key[0] were included in the internal nodes, then the internal node methods could be identical to the leaf node methods (modulo the different value types and sizes). In that case also, each tree level would conceptually constitute a complete linked list of (splitting key, subtree) pairs. The B+ tree operations would be at least somewhat easier. Thus it is helpful to think of B+ trees as "by nature" including key[0] in the internal nodes, and that removing key[0] is an efficiency tweak that adds some complexity.

Each leaf node block is laid out as indicated in Figure 5-9. The layout of an internal node is the same except that there is no space for the 0th key and the value type is different. The keys are presumed kept in order. The array of key pairs in a node can thus be searched using a binary search.

المنارة للاستشارات
www.manaraa.com

5.3.3.2 bptIndex

The bptIndex is an implementation of a traditional B+ tree mechanism. All of the normal operations for traversing a tree and rebalancing the tree are the traditional mechanisms presented in the literature [Date04, elma03, Rals03, Rama03, Silb96, Tuck04]. The only modifications to the source code and data structures are those described above in section 5.3.3.1 and the addition of the buffer manager extensions for caching.

5.3.3.3 ALVDataFile

The ALVDataFile is a set of C++ classes designed to manage the myTable.alvi file (a disk file that stores the version index). It stores the data for the nodes blocks (block size is adjustable) and provides access to header information (header size is adjustable), free space, and statistical information about the data file. It also supports clustered block access for use in buffer management algorithms. Unlike the ALVDataFile implementation in the clustered version store, the version index does not use block extents. Rather, a node is stored as a complete (single) block on disk.

المنارة للاستشارات
www.manaraa.com

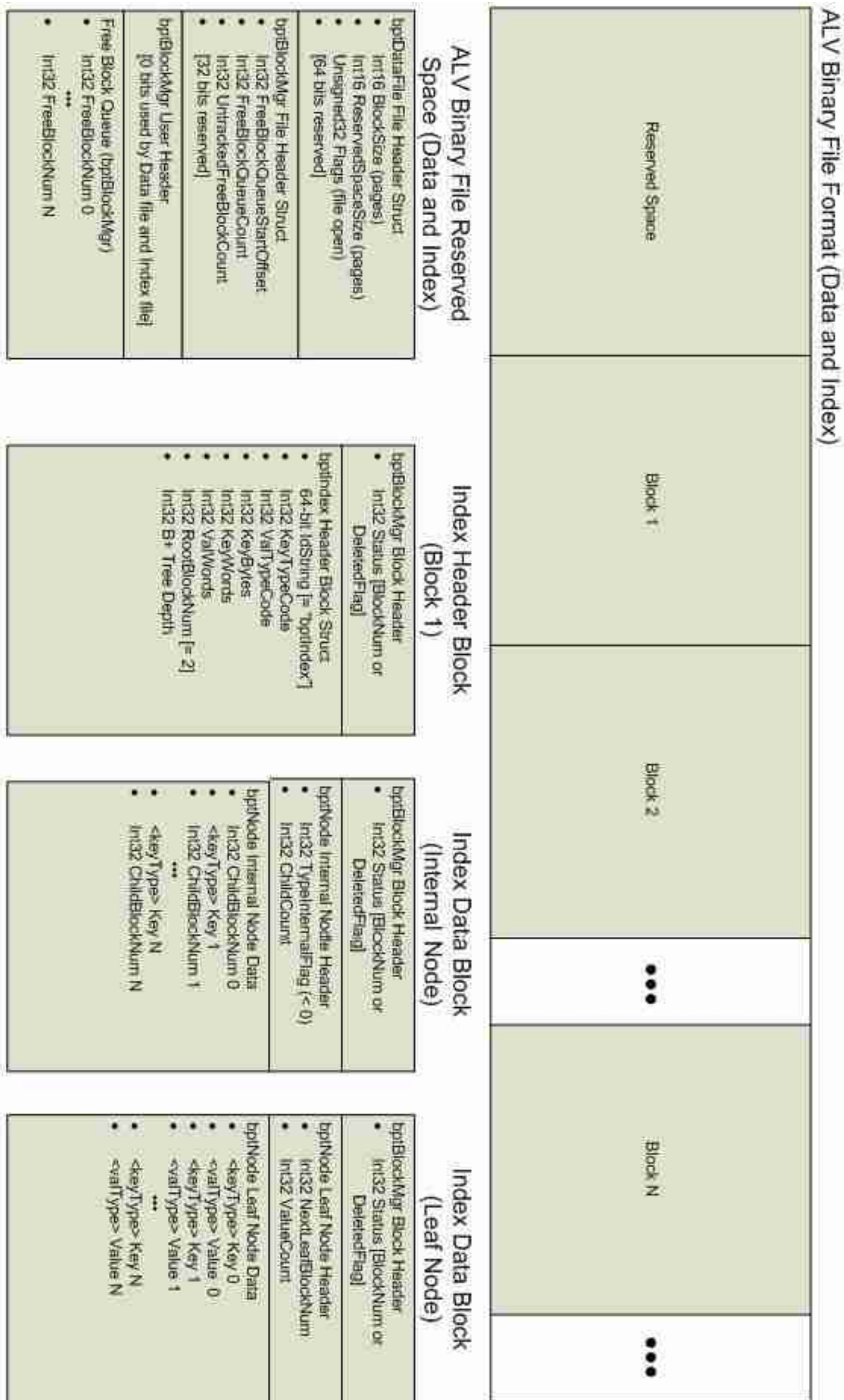


Figure 5-9: Version Indexing File Format

www.manaraa.com

This permits the nodes to be saved on disk in any order, enabling the host operating system to manage space on the physical media. Otherwise, the data file performs in the same manner as that of the clustered version store. Figure 5-9 depicts the layout, data structures, and header information details for a typical version index file (myTable.alvi).

5.3.4 Buffering and Transactions

The version index uses the same buffer management strategy as the clustered version store. To revisit the argument, the buffer is designed to fetch blocks from disk into memory saving them in memory, only when there is either a concurrent request for access (read or write) or a transaction in progress. The argument is that the virtual memory and file systems of the host operating system provide adequate support for buffering at the file level. Thus any sort of prefetching or caching of the blocks on disk is not necessary [Smit78] and the cache mechanism showed little to no gain in performance when implemented.

5.3.4.1 The role of the ALV Buffer Manager

The ALV buffer manager performs the same role for the index store as it does for the physical store. In fact, it performs exactly like that of the physical store implementation except that the concept of block extents is not used and the root node (page) of the tree is always kept in memory [Baye72]. This implementation of the ALV buffer manager is called the bptBlockMgr.

المنارة للاستشارات
www.manaraa.com

5.3.4.2 Transactions in ALV

The implementation of concurrency and transactions is a serious programming challenge [Hugh97]. Concurrency and transaction processing are deeply related. In order to support transactions, one must also support concurrency. By putting support for concurrency into the system and having a suitable buffering strategy, the ground work for supporting transactions is laid. The principal goal of database concurrency is to ensure that the concurrent execution of the transaction does not result in the loss of any consistency or database integrity [Silb96].

Consider this question: If there are two threads executing transactions and they each modify the same table (perhaps even the same records), and if one rolls back but the other commits what is the state of the database tables? What gets rolled back and what gets committed? The answers to these questions are the solution to detecting and preventing deadlock and how to resolve conditions where transactions are interleaved. These are important considerations for designing and implementing a database transaction mechanism.

A transaction log is a common mechanism to store all operations performed on a database file. Transaction systems and log files must match the physical file definition. It would be difficult to create a generic logging and transaction mechanism for any physical layout. Thus, logging systems are typically built to accommodate a particular physical store.

What is stored when the log is updated? If you store only the operations (canonical or verbose), interleaved operations cannot be selectively rolled back.

المنارة للاستشارات
www.manaraa.com

However, if you store the operations and the blocks that are affected (before image), annotating them by a sequence number that corresponds to the transaction, it is possible to rollback interleaved operations. However, the database system must be preempted to accomplish this (all files involved are locked).

Another area of concern is what happens if a transaction is begun but no commit or rollback occurs? Should the system behavior default to always commit or always rollback or commit only if there are no errors?

It is important to discuss transactions in the ALV system because of the effect they have on the version index and its execution. As described above, the version index has been designed to support concurrency and transactions natively with the data structure and its methods rather than in an additional layer or mechanism.

Transactions in ALV follow the theory for database transaction integrity and therefore exhibit the ACID properties: atomicity, consistency, isolation, and durability [Date04, Elma03, Rals03, Rama03, Silb96, Tuck04]. There are three operations for transaction processing: BEGIN_TRANSACTION, COMMIT, and ROLLBACK. Each is explained below:

- **BEGIN_TRANSACTION** – this operation signals the database system to checkpoint the current set of operations and begin recording all of the changes to the database system.
- **COMMIT** – this operation permits all of the operations in a transaction to be permanently written to the database files (physical).

المنارة للاستشارات
www.manaraa.com

- **ROLLBACK** – this operation tells the database system to discard all of the changes made to the database files since the previous BEGIN_TRANSACTION operation.

Transactions are supported using mechanisms designed to use the data structure features of the version index and the clustered version store. The choice to use a two-phase locking strategy was necessary to follow the general practice for implementing a transaction manager[Rals03]. This mechanism was built using two classes, the TransactionManager class and the TransactionalBlockManager class.

The ALVManager contains a single instance (singleton) of the TransactionManager class, which has methods BeginTransaction(key) and EndTransaction(key). Key is an identifier for the transaction, and in this case we are using the thread pointer (THD) from the MySQL executor for the session. BeginTransaction and EndTransaction are called from the MySQL functions begin_trans and end_trans in sql_parse code.

The TransactionalBlockManager is essentially a transaction-conscious proxy for bptBlockMgr. TransactionalBlockManagers are associated with the TransactionManager and report their construction/destruction to that manager. All places in ALVRecordFile, and bptIndex which involve locking, unlocking, changing, or deleting blocks call the equivalent method on TransactionalBlockManager, adding the transaction key to the argument list. Accordingly, the methods of ALVRecordFile and bptIndex have the transaction key as an argument. The TransactionalBlockManager passes through and

المنارة للاستشارات
www.manaraa.com

caches lock requests while buffering unlocks, changes, and deletions until the end of the transaction.

When an end of transaction command is issued for a key, the TransactionManager subsequently calls EndTransaction on each TransactionalBlockManager, which appropriately saves or discards the altered blocks depending on if the transaction is being committed or rolled back.

By using a transaction key and associating operations with it, the ALV transaction mechanism permits multiple transactions to execute in parallel and not affect one another should one be rolled back and the others committed. Caching all of the changes to the data (and index) enhances this ability by providing a reference point for each change. In this manner, queries on the ALV system support concurrency and transactions.

5.3.4.3 Deadlock Prevention

One problem that had to be solved when dealing with concurrency and transactions is how to prevent deadlocks. A deadlock occurs when two or more processes are competing for the resources that are held by the other process waiting on the first (a circular wait occurs). Examples include race conditions and mutual exclusion of locks (process A has an exclusive lock on an object and is requesting an exclusive lock held by process B which, in turn is requesting an exclusive lock on the object held by process A).

The bptTree mechanisms were defined using an ordering on the blocks of the index file as follows: higher-level nodes (closer to the root) are always less than lower-level nodes, and nodes on the same level are ordered left-to-right (in agreement with key

www.manaraa.com

values). The header block is considered to be greater than all other blocks. Deadlock is prevented by requiring that locked blocks must be acquired in increasing order, i.e. if block B is requested and block A is already held with some kind of lock, then A must be less than B.

5.4 Analysis

This section describes the analysis performed while implementing and experimenting with the version index and its individual components. All of the experiments were run on a 3.0Ghz AMD processor-based system running Windows XP Professional. The disk subsystem used was a hardware raid system incorporating two S-ATA physical devices in a mirrored arrangement. The experiments were repeated using a conventional IDE-133 device with little or no variation in the measurements[24].

5.4.1 Index Experiments

Experiments conducted on the version index itself would not be very interesting and would only demonstrate how the index performs in a database system. The indexing experiments would not show the benefits and performance improvements over sequential access. The experiments conducted were designed to be run with the version index fully integrated into the database system. The experiment involved accessing data from the clustered version store both using the index and without using the index. A similar

[24] This is expected because the differences in the physical devices and their access protocols are very similar. Although th roughput on the S-ATA devices theoretically could be faster, the addition of the raid subsystem nullifies any advantage over IDE -133 devices.

المنارة للاستشارات
www.manaraa.com

experiment was conducted showing the performance of two native MySQL physical stores and their indexing mechanisms. Each of these experiments were conducted using a small (599 rows), medium (32561 rows) and large (201053 rows) data set. A complete description of these tables and their composition can be found in Appendix A.

Table 5-1 lists the data for the results of the experiment. The three datasets used (small, medium, large) are grouped in rows depicting access times without and with using the index mechanism. Included is data from the experiment in Chapter 4 used as a comparison of the improvement indexing provides over a table scan. Column 2 contains the access time for the ALV data store, column 3 and 4 contain access times for the MyISAM and InnoDB data stores respectfully. The values presented show that the ALV data store access times using the version index exceed that of both MyISAM and InnoDB. Figures 5-10 through 5-12 depict graphs comparing the performance of the indexing experiments and are graphical representations of the comparison of access times for the ALV data store versus the MyISAM data store.

المنارة للاستشارات
www.manaraa.com

(small)	**Customer**		
Time (seconds)	**ALV**	**MyISAM**	**InnoDB**
(No Index):	0.0152842700	0.0020339740	0.0011315960
(with Index):	0.0000752749	0.0001261892	0.0007118642
% Improvement:	20304.60%	1611.84%	158.96%
(medium)	**Adults**		
	ALV	**MyISAM**	**InnoDB**
(No Index):	0.8611705000	0.1044296000	0.0612593100
(with Index):	0.0000839632	0.0001066895	0.0382407500
% Improvement:	1025652.31%	97881.80%	160.19%
(large)	**ORF**		
	ALV	**MyISAM**	**InnoDB**
(No Index):	6.6346780000	0.7803103000	0.3553471000
(with Index):	0.0000587924	0.0001057397	0.2314253000
% Improvement:	11284926.50%	737953.96%	153.55%

Table 5-1: Results of Indexing Experiments

Indexing Experiment - Small Table

Time (sec)

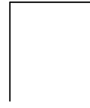
0.00014
0.00012
0.00010
0.00008
0.00006
0.00004
0.00002
0.00000

ALV
MyISAM

	Customer
ALV	0.0000752749
MyISAM	0.0001261892

Table Scanned

المنارة للاستشارات
www.manaraa.com



Indexing Experiment - Medium Table

ALV
MyISAM

Time (sec)

MyISAM

Table Scanned

Indexing Experiment - Large Table

0.000120
0.000100
0.000080
0.000060
Time (sec) 0.000040
0.000020
0.000000

ALV
MyISAM

	ORF
ALV	0.0000587924
MyISAM	0.0001057397

Table Scanned

المنارة للاستشارات

www.manaraa.com

located about 3/4th into the file). The location of the target row in the file is important because without indexes, the system would have to perform a table scan to locate the target rows. In this experiment, the data was placed in various positions and near the end of the file to demonstrate the best performance gain of using indexes to access the data directly.

Table	Type	Sample SQL Statement	Location in file for each result (%)			
Customer	ALV	`select alv * from customer_alv where customer_alv.alv_key = 350;`	0.040067	0.744574	0.93823	.966102
	MyISAM	`select SQL_NO_CACHE * from customer where customer.alv_key = 350;`				
	InnoDB	`select SQL_NO_CACHE * from customer_i where customer_i.alv_key = 350;`				
Adults	ALV	`select alv * from adults_alv where adults_alv.alv_key = 7185;`	0.204969	0.999816		
	MyISAM	`select SQL_NO_CACHE * from adults where adults.alv_key = 7185;`				
	InnoDB	`select SQL_NO_CACHE * from adults_i where adults_i.alv_key = 7185;`				
ORF	ALV	`select alv * from orf_alv where orf_alv.alv_key = 143434;`	0.997916			
	MyISAM	`select SQL_NO_CACHE * from orf where orf.alv_key = 143434;`				

المنارة للاستشارات
www.manaraa.com

Table	Type	Sample SQL Statement	Location in file for each result (%)			
	InnoDB	select SQL_NO_CACHE * from orf_i where orf_i.alv_key = 143434;				

Table 5-2: Description of Data for Indexing Experiments

The MySQL SQL_NO_CACHE option in the SELECT statement was necessary to discount the query cache mechanism inherent in MySQL in order to present a fair comparison. As discussed previously, the ALV mechanisms would benefit from being fitted to use the MySQL query cache mechanism.

Clearly, the addition of the version index increased performance of the query dramatically for all datasets, with the greatest benefit gained for the large data set. Also, the version index and the clustered version store together performed better than the native MySQL mechanisms.

The reason for this increase in performance over the MySQL mechanisms is the design of the clustered version store. Combined with a fast index, accessing a set of attribute versions for a given key is simplified to reading a single block into memory and dereferencing the attribute chain (rows) for the result. The MySQL files required accessing several blocks from the physical store in order to retrieve all of the results. Therefore, the ALV clustered version store and associated index mechanisms provide better performance for retrieval of data that has a high degree of association. Thus, a clustered version store and index are technologies designed to support versioned data while providing high performance access comparable to or better than traditional database access mechanisms.

المنارة للاستشارات

www.manaraa.com

5.4.2 Real World Performance

This section presents observations of the ALV system running in a real environment using live data constructed from actual data sources. Although the disclosure of the actual data is not possible, the results of the experiments are meaningful and demonstrate significant milestones in the integration of the ALV system into a world-class database system.

While using the system without indexing for all but very large data sets, the version queries ran in short enough times as to be unnoticeable. In fact, it wasn't until the data sets grew to over 100 host entities and approximately 10 attribute versions that any noticeable or measurable delay was encountered. By adding the version index mechanism, these delays were eliminated and performance was enhanced beyond the current capabilities of the mechanisms in MySQL.

5.5 Conclusion

The clustered version store is the cornerstone of the ALV system. However, it is incomplete without a mechanism to index and access the data in an efficient (timely) manner. The integration of the version index into the ALV system is therefore imperative in order to provide the speed necessary for a system to be considered for production use. By demonstrating the ability to store attribute versions in a dedicated, specialized physical storage mechanism and accessing the data using rapid index resolution, this work has demonstrated the version index mechanism is reliable and performs well. The experiments and real world experience of using the ALV system with the version

المنارة للاستشارات
www.manaraa.com

indexing mechanisms demonstrate that a fast indexing mechanism is required to ensure high speed performance of retrieval of versioned data for a versioning system and that the version indexing mechanism can be supported in a relational database system.

Additionally, the creation of the B^2+ tree and the mB^2+ tree have shown a unique form of B+ tree that has buffer management, concurrency, and transaction support built into the data structures and algorithms. This tight integration of these features proves that these new variants of the venerable B+ tree are viable mechanisms for increasing the performance of indexing mechanisms.

5.6 Future Work

Construction of the version index from existing data is a concern that should be addressed in the near future. This could be an especially important performance issue if the database systems that implement ALV are used for high-speed data processing. Kim's technology of batch-construction [Kim01] should be investigated for incorporation into the version index mechanisms.

Although the version index performed well with the ALV buffering mechanism, it is possible the buffering mechanism may need to be altered once a sufficiently large data set is used. Currently, none of the large data sets tests have shown any unusual behavior and the index and buffer mechanism work well. Performance under these circumstances has been proportional to the size and complexity of the data. Further research will be necessary to test the cumulative effects of very large data sets on the version index and buffer mechanism.

المنارة للاستشارات
www.manaraa.com

The application of the B^2+ tree and mB^2+ tree in the ALV system is not as flexible as it could be. Additional work will be necessary to provide database professionals the tools necessary to create alternative indexes on any given attribute value or metadata attribute within an attribute version. These extra tools will give database professionals additional opportunities to tune the versioned database for optimal performance based on the need and intended use of the version system.

While the concurrency and transaction mechanisms work well, there is no support for recovery. Database recovery mechanisms are designed to be able to recover the state of the database should the system become unstable or crash. With a recovery system such as a log-based journal where all operations and their outcomes are stored, all but the most severe of system failures could be recoverable and the state of the database rebuilt on restart. The ALV system does not support any form of logging or recovery. Additional work is necessary to implement this feature into the ALV system. By doing so, the ALV system will be more applicable in environments where recovery is a high priority or necessity.

Lastly, the B^2+ tree and mB^2+ tree mechanisms described above should be generalized for use with more traditional physical data stores. This will ensure that the technology is added to the collection of many successful indexing mechanisms bearing the legacy of Bayer and McCreight [Baye72].

المنارة للاستشارات
www.manaraa.com

Chapter Six - A Query Optimizer and Execution Engine for Versioning

Abstract

Now that there is a valid storage and retrieval mechanism that forms the basis of a versioning system, it is now imperative to construct a query optimizer and execution engine that can perform the queries in an expedient manner. This chapter will discuss an implementation of a query optimizer and execution engine and report its performance as compared to a native commercially available query optimizer and execution engine.

6.1 Introduction

To a large extent, the success of a database management system lies in the quality, functionality, and sophistication of its query optimizer, since optimization greatly affects the system's performance. A fast query optimizer and query execution engine is vital to the success of any database system. This chapter presents a query optimizer for fast processing of queries and an execution engine designed to execute queries of versioned data in a clustered version store within a versioning system. This system, called Attribute-Level Versioning (ALV), is an extension of the MySQL database management system.

The following sections present the current research on query optimization and execution, the technology and design of a query optimizer and execution engine, an

المنارة للاستشارات
www.manaraa.com

analysis of the performance of the mechanisms, and a conclusion as to their success in meeting the goals defined above. This chapter concludes with a section outlining future work opportunities to improve the optimizer and execution engine.

6.2 Background

This section explains the necessity and importance of query optimization in relational database systems. Topics examined will include data independence, how a database system processes a query, and, in particular, where query optimization fits into the query process.

Database systems operate in a client-server model. This model is best described in terms of the function of each client and the server. A client is used to interact with and present data from the server. The server performs all of the database processing and transmits results to the client. In this model, there are usually many clients per a single server[1]. In the context of a database system operating in this model, the database server is responsible for processing the queries presented by the client and returning the results accordingly. This has been termed query shipping [Fran96] where the query is shipped to the server and a payload (data) is returned. The benefits of query shipping are a reduction of communication time for queries and the ability to exploit server resources rather than using the more limited resources of the client to conduct the query. This model also permits a separation of how the data is stored and retrieved on the server from the way

[1] However, current trends concerning high availability suggest a client server model where many clients access several servers either as distributed or replicated servers [Daco03].

www.manaraa.com

the data is used on the client. In other words, the client-server model supports data independence.

One of the principal advantages of the relational model introduced by Codd in 1970 is data independence [Date01]. *i.e.* the separation of the physical implementation from the logical model. Codd said, "Users of large data banks must be protected from having to know how the data is organized in the machine … Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed." This separation allowed a powerful set of logical semantics to be developed, independent of a particular physical implementation. The goal of data independence (called physical data independence in Elmasri and Navathe [Elma03]), is that each of the logical elements is independent of all of the physical elements (see Table 6-1). For example, the logical layout of the data into relations (tables) with attributes (columns) arranged by tuples (rows) is completely independent of how the data is stored on the storage medium.

Logical Model	**Physical Model**
• Query Language • Relational Algebra • Relational Calculus • Relvars	• Sorting Algorithms • Storage Mechanisms • Indexing Mechanisms • Data Representation

Table 6-1:The Logical and Physical Models of Database Design

One of the challenges of data independence is that database programming becomes a two-part process. First, there is the writing of the logical query -- describing

www.manaraa.com

what the query is supposed to do. Second, there is the writing of the physical plan -- which shows how to implement the logical query.

The logical query can be written, in general, in many different forms such as a high-level language like structured query language (SQL) [Cham81] or as an algebraic query tree [Tuck04]. For example, in the traditional relational model, a logical query can be described in relational calculus or relational algebra. The relational calculus is better in terms of focusing on what needs to be computed [Date04]. The relational algebra is more concrete, but still leaves out many details involved in the evaluation of a query [Date04].

The physical plan is a query tree [Wern01] in a physical algebra that can be understood by the database system's query execution engine. A query tree is a tree structure in which each node contains a query operator and has a number of children that corresponds to the arity of the operation. The query tree can be transformed via the optimizer into a plan for execution. This plan can be thought of as a program that the query execution engine can execute.

There are several phases that a query statement goes through before it is executed; parsing, validation, optimization, plan generation/compilation, and execution [Grae93a]. Figure 6-1 depicts the query processing steps that a typical database system would employ[2]. Each query statement is parsed for validity and checked for correct syntax and for identification of the query operations. The parser then outputs the query in an intermediate form to allow the optimizer to form an efficient query execution plan. The execution engine then executes the query and the results are returned to the client. This

[2] Some database systems combine the parsing and validation into a single step [MySQ05].

المنارة للاستشارات
www.manaraa.com

progression is depicted in figure 6-1 where once parsing is completed, the query is validated for errors, then optimized, a plan chosen and compiled, and finally the query is executed.

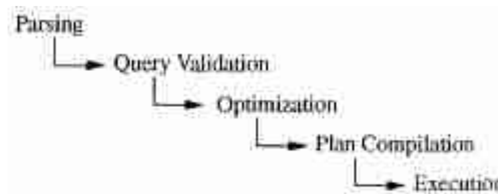


Figure 6-1: Query Processing Steps

The first step in this process is to translate the logical query from SQL into a query tree in logical algebra. This step is done by the parser. The next step is to translate the query tree in logical algebra into a physical plan. There are generally a large number of plans that could implement the query tree. The process of finding the best execution plan is called query optimization. That is, for some query execution performance measure (*e.g.* execution time), we want to find the plan with the best execution performance. The goal is that the plan be optimal or near optimal within the search space of the optimizer. The optimizer starts by copying the relational algebra query tree into its search space. The optimizer then expands the search space and finds the best plan. At this level of generality, the optimizer can be viewed as the code generation part of a query compiler for the SQL language. It produces code to be interpreted by the query execution engine, except that the optimizer's emphasis is on producing "very efficient" code. For example, the optimizer uses the database system's catalog to get information (*e.g.* number of tuples) about the stored relations referenced by the query, something traditional programming language compilers normally do not do [Much97]. Finally, the optimizer

المنارة للاستشارات
www.manaraa.com

copies the optimal physical plan out of its memory structure and sends it to the query execution engine. The query execution engine executes the plan using the relations in the stored database as input, and produces the table of rows that match the query criteria as output.

All of this activity requires additional processing time and places a greater burden on the process by forcing database implementers to consider the performance of the query optimizer and execution engine as a factor in their overall efficiency. Ioannidis states this best, "Relational query optimization is an expensive process, primarily because the number of alternative access plans for a query grows at least exponentially with the number of relations participating in the query," [Ioan97].

One of the primary reasons for the large number of query plans is that optimization will be required for many different values of important run-time parameters whose actual values are unknown at optimization time. Database systems make certain assumptions about the database contents (*e.g.*, value distribution in relation attributes), the physical schema (*e.g.*, index types), the values of the system parameters (*e.g.*, number of available buffers), and the values of the query constants [Ioan97].

6.2.1 Query Language

The details of query languages are beyond the scope of this work and can be found in works from Date [Date04], Ramakrishnan [Rama03] and Silberschatz [Silb96].

المنارة للاستشارات
www.manaraa.com

However, a basic understanding of SQL[3] is necessary to explain how queries are represented and provide the view of the database system to the users.

A query language such as SQL is a language (has a syntax and semantics) that can be used to represent a question posed to a database system. In fact, the use of SQL in database systems is considered one of the major reasons for their success [Elma03]. SQL provides several language groups that form a very comprehensive foundation for using database systems. The data definition language (DDL) is used by database professionals to create and manage databases. Tasks include creating and altering tables, defining indexes, and managing constraints[4] [Rals03]. The data manipulation language (DML) is used by database professionals to query and update the data in databases. Tasks include adding and updating data as well as querying the data [Rals03]. These two language groups form the majority of commands that database systems support which database professionals use to manage data and database systems [Elma03].

SQL commands are formed using a specialized syntax. The following presents the syntax of a SELECT command in SQL[5]. The notation below depicts user-defined variables in italics and optional parameters in square brackets ([]). Note that the expressions are depicted as being in conjunctive normal form (CNF) where all of the

[3] Some researchers have contended that the growth of SQL is an industry phenomenon rather than the expression and evolution of theory [Date90].
[4] Constraints put the relation in the relational model. Concepts include primary -foreign keys, data domain and range control, cascading deletes, etc. [Date04].
[5] Although most database systems implement their own version of the SQL standard, most follow this pattern.

المنارة للاستشارات
www.manaraa.com

predicates are transformed into a set of predicates that form a conjunction of clauses, where a clause is a disjunction of literals[6].

```
SELECT [DISTINCT] listofcolumns
FROM listoftables
[WHERE expression (predicates in CNF)]
[GROUP BY listofcolumns
[HAVING expression]]
[ORDER BY listofcolumns];
```

The semantics of this command are [Ston98]:

1. Form the cartesian product of the tables in FROM clause, forming a projection of only those references that appear in other clauses
2. If a WHERE clause exists, apply all expressions for the given tables referenced
3. If a GROUP BY clause exists, form groups in the results on attributes specified
4. If a HAVING clause exists, apply a filter for the groups
5. If an ORDER BY clause exists, sort the results in the manner specified
6. If a DISTINCT keyword exists, remove the duplicate rows from the results

The example shown above is representative of most SQL commands in that all have required portions of the syntax, and most also have optional sections as well as keyword-based modifiers. The query language extensions presented in section 6.3.4 follow this same pattern.

6.2.2 Query Optimization Strategies

A complete detailed analysis of query optimization is beyond the scope of this work, this section introduces the concepts and techniques of query optimization. A

[6] The CNF form of the predicates is created early in the optimization process.

المنارة للاستشارات
www.manaraa.com

foundation of the challenges and purpose of query optimization are presented along with examples of query optimization strategies in existing database systems. For a more detailed exploration of query optimization, refer to the works of Date [Date04] and Lawrence [Lawr04]. Query optimization is the part of database systems that provides the greatest contribution to the efficiency of the database system [Das95].

Query optimization is the part of the query compilation process that translates a data manipulation statement in a high-level, non-procedural language, such as SQL, into a more detailed, procedural sequence of operators, called a plan. Query optimizers usually select a plan by estimating the cost of many alternative plans and then choosing the least expensive amongst them [Gass93].

Database systems that use a plan-based approach to query optimization assume that there are many plans that can be used to produce any given query. While this is true, not all plans are equivalent in the number of resources (cost) needed to execute the query nor are all plans executed in the same amount of time [Ioan96]. The goal then is to discover the plan that has the least cost and/or runs in the least amount of time. The distinction of either resource usage or cost usage is a tradeoff often encountered when designing systems for embedded integration or running on a small platform (low resource availability) versus the need for higher throughput (time).

Figure 6-2 depicts a plan-based query processing strategy. The query follows the path of the arrows. The SQL command is passed to the query parser where it is parsed and validated then translated into an internal representation, usually based on a relational algebra expression. The query is then passed to the query optimizer which examines all

المنارة للاستشارات

www.manaraa.com

of the algebraic expressions that are equivalent generating a different plan for each combination. The optimizer then chooses the plan with the least cost and passes the query to the code generator, which translates the query into an executable form, either as directly executable or as interpretative code. The query processor then executes the query returning a single row in the result set at a time.

This is a common implementation scheme and typical of most database systems. However, the machines that the database system runs on have improved. It is no longer the case that a set of query plans have diverse execution costs. In fact, most query plans have been shown to execute with approximately the same cost [MySQ05]. This realization has led some database system implementers to adopt a query optimizer that focuses on optimizing the query using some well known good practices (heuristics) for query optimization.

المنارة للاستشارات
www.manaraa.com

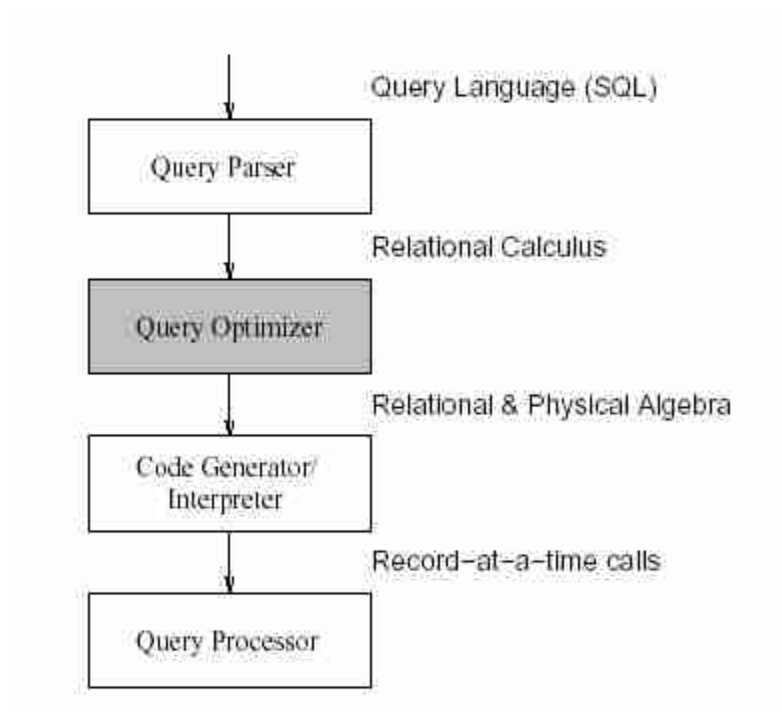


Figure 6-2: Typical Database System Implementation

An example of such a system is MySQL. The query optimizer in MySQL is designed around a select-project-join[7] strategy. The query is broken down into an internal form, then optimized to execute all selects (restrictions) first, then projections, and finally all joins. The internal representation of queries in this model is not a query tree, but rather a collection of collections–based data structure [Badi02]. This strategy ensures an overall "good" execution plan, but does not guarantee to generate the best plan. This strategy has proven to work well for a vast variety of queries running in very different environments. The internal representation performed well enough to rival the execution speeds of the largest of the production database systems [MySQ05].

[7] The use of the select-project-join strategy in MySQL is not new. Researchers have discussed it many times in the literat ure and is frequently referred to as a flat query because the supporting data structures are typically singular structures without branches [Tuck04].

www.manaraa.com

The purpose of query optimization is to form an execution plan for the query issued that minimizes the time necessary to return the correct result set. Complicating this goal is the fact that any given query can be represented, or created and results produced, using many different execution plans. These execution plans, sometimes called execution paths, must be generated and evaluated prior to deciding which one produces the plan with the least cost.

An example of this behavior can be seen in Microsoft's SQL Server [Bere00]. The query optimizer in SQL Server is designed around a classic cost-based optimizer that translates the query statement into a procedure that can execute efficiently and return the desired results. The optimizer uses information (statistics[8]) formed from past queries and the conditions of the data in the database to create alternative procedures that represent the same query command. The application of the statistics to each of the procedures is used to predict which of the procedures can be executed more efficiently. Once the most efficient procedure is chosen, execution begins and results are returned to the client.

The greatest challenge in creating a query optimizer is to choose which optimization decisions to delay and how to engineer a query optimizer that efficiently creates dynamic plans for arbitrarily complex queries at execution [Cole94] while maintaining a sound concurrent execution model [Gray94]. In fact, a great deal of research has been conducted in order to trace and understand the behavior of query

[8] The use of statistics in databases stems from the first cost -based optimizers. In fact, many utilities exist in commercial databases that permit the examination and generation of these statistics by database professionals to tune their databases for more efficient optimization of queries [Bane03].

المنارة للاستشارات
www.manaraa.com

optimizers in database systems [Date92]. Singhal and Smith [Sing97] present advances in this area.

Optimization of queries can be complicated by using parameters that are unbound (a user predicate). In this case, query optimization may not be possible or it may not generate the lowest cost unless some knowledge of the predicate is obtained prior to execution. If very few records satisfy the predicate, even a basic index is far superior to the file scan. The opposite is true if many records qualify. If the selectivity is not known when optimization is performed, the choice among these alternative plans should be delayed until execution [Cole94].

The problem of selectivity can be overcome by building optimizers that can adopt the predicate as an open variable and perform query plan planning by generating all possible query plans that are likely to occur based on historical query execution and by utilizing the statistics from the cost-based optimizer. The statistics include the frequency distribution for the predicate's attribute.

6.2.3 Internal Representation

A query can be represented within a database system using several alternate forms of the original SQL command. These alternate forms exist due to redundancies in SQL, the equivalence of subqueries and joins under certain constraints, and logical inferences that can be drawn from predicates in the where clause. Having alternate forms of the query poses a problem for database implementers because the query optimizer must

المنارة للاستشارات
www.manaraa.com

choose the optimal access plan for a query regardless of how it was originally formed by the user [Gass93].

The PostqreSQL database system [Post05] uses a unique form of internal representation. When a query is processed and the conversion rules are applied to translate the SQL statement, the output is a data structure called a query tree[9]. In PostgreSQL, a query tree is an internal representation of an SQL statement where the single parts that built it are stored separately. That is, the command is stored as a value in the structure to indicate what kind of query it is (SELECT, INSERT, UPDATE, DELETE, etc.). A range table is included that lists all of the relations that are used in the query. Each table entry in the range table identifies a table or view and stores its internal name. Table entries are referenced by index rather than by name. An empty result relation is included to store the query results. A target list is included that is a list of expressions that define the result of the query. Every entry in the target list contains the expressions associated with the command. Lastly, a join tree is used to store the structure of the join clause. Additional portions of the data structure are used to store intermediate and optional operations such as UNION, ORDER BY, etc [Post05].

In many ways, the implementation of the MySQL internal query representation resembles the PostgreSQL data structure. The MySQL data structures mirror many aspects of the PostgreSQL query tree [MySQ05].

---

[9] Not to be confused with a query tree as presented in liter ature. In this case, it isn't really a tree at all. A bush, perhaps, but not a tree as defined in the body of knowledge that is Computer Science.

المنارة للاستشارات
www.manaraa.com

Other systems use a different form of a query tree that is more advanced than that of PostgreSQL and MySQL. Section 6.2.3.2 presents additional details about query trees and their use in forming the internal representation of SQL statements.

6.2.3.1 Relational Calculus and Relational Algebra

Relational algebra forms the basic set of operations for the relational model. These operations are performed against one or more relations producing a relation as a result[10], which can be manipulated with the same set of operations. A set of relational algebra operations is called a relational algebra expression [Elma03]. Relational algebra is very important to database systems because it forms the formal foundation for manipulating the relational model. More importantly, relational algebra is the model for performing query optimization (operations designed to improve the efficiency of the relational algebra expressions) and is the basis (in part) from which query languages such as Structured Query Language (SQL) were created.

As stated earlier, the query optimizer must consider all possible implementations (execution) for a given relational-algebra expression. It is the job of the query optimizer to form query execution plans that compute the same result as the given expression and select the one with the least cost of generating the result [Silb96]. Generation of query execution plans involves two steps; 1) generating expressions that are logically equivalent to the given expression, and 2) transforming the resulting expressions into equivalent statements that can be executed by the database system. Most query

[10] In some cases a relation with no tuples. Not to be confused with a Null relation that is a relation without structure or tuples.

المنارة للاستشارات
www.manaraa.com

optimizers interleave these steps by operating on a part of the expression at a time until all expressions are transformed into all possible variants [Silb96].

The first step is accomplished by means of applying equivalence rules that specify how to transform an expression into a logically equivalent one [Silb96]. The second step can be implemented using one of the following evaluation techniques; 1) cost-based optimization, 2) heuristic optimization, 3) semantic optimization, and 4) parametric optimization. Most query optimizers are implemented as cost-based optimizers, but more commonly they contain elements of all three techniques.

The following query demonstrates how a single query can generate more than one equivalent relational-algebra expression:

```
SELECT balance FROM account WHERE balance > 2500
```

This query can be translated into either of the following relational-algebra expressions:

```
σ balance > 2500 (π balance (account))
π balance (σ balance > 2500 (account))
```

The first query is called a projection biased expression in which the projection is the first operation, whereas the second is called a selection biased expression in which the selection is presented first. Both of these expressions will generate the same results. However, depending on the cost of the projection and selection, one may be of a higher cost to execute than the other.

A considerable amount of research has been conducted in the area of relational algebra. Some researchers have devoted study to extending the relational algebra for use

www.manaraa.com

in set-based query processing (constraints) [Belu98]. The generalized relational algebra presented by Belussi, et. al. provides a mechanism by which multi-dimensional queries can be formed and processed. Interestingly, the concept of versioning fits the descriptions of a constraint query. Indeed, it would be interesting work to explore the use of constraint databases to store and retrieve version information.

Where relational algebra forms the basis for query optimization and query languages, relational calculus provides a higher degree of declarative notation for specifying relational queries [Elma03]. Relational calculus is best known as tuple calculus, where the relational calculus expressions create a new relation in terms of the variables that range over the tuples stored in the relation being operated on.

Relational calculus has also been expressed in terms of columns of relations (domain calculus). Unlike a relational algebra expression in which the order of operations is defined[11], a calculus expression has no order of operations[12] and does not specify how the results are generated; rather it defines what information the result contains. This feature is the most distinguishing difference between relational algebra and relational calculus.

Table 6-2 below presents several queries formed using both relational algebra and tuple relational calculus [Elma03].

[11] The ordering of operations is the key feature that enables optimization. Without order, one cannot predict which pairs of operations can generate the most efficient execution plan.
[12] Relational algebra expressions are called procedur al whereas tuple relational calculus expressions are called non -procedural.

www.manaraa.com

<table>
<tr><th>Query</th><th>Relational Algebra</th><th>Tuple Relational Calculus</th></tr>
<tr><td>Find the names of all employees who work on project number 72.</td><td>P ← $\sigma_{PNum='72'}$(PROJ)
E ← (P $\theta_{PNum=PNum}$ WORKSON)
S ← (E $\theta_{SSN=SSN}$ EMP)
Result ← $\pi_{LName, FName}$ (S)</td><td>Q:[m.LName, m.FName | PROJ(t) AND EMP(m) AND p.PNum = ‘72’ AND ((∃d) (WORKSON(d) AND p.PNum=d.PNum AND d.MSSN=m.SSN))}</td></tr>
<tr><td colspan="3">SELECT FName, LName
FROM EMP JOIN WORKSON ON SSN=SSN JOIN PROJ ON PNum=PNum
WHERE PROJ.PNum = ‘72’;</td></tr>
<tr><td>Retrieve the names and addresses of all employees who work for the ‘Design’ department.</td><td>D ← $\sigma_{DNAME='Design'}$(DEPT)
E ← (D $\theta_{DNum=DnumEmp}$)
Result ← $\pi_{FName, LName, Address}$ (Emp)</td><td>Q:[t.FName, t.LName, t.Address | EMP(t) AND (∃d) (DEPT(d) AND d.DName = ‘Design’ AND d.DNum =t.DNumEmp}</td></tr>
<tr><td colspan="3">SELECT FName, LName, Address
FROM EMP JOIN DEPT ON DNumEmp=DNum
WHERE DEPT.DName = ‘Design’;</td></tr>
<tr><td>For every project located in ‘Warsaw’, list the project number, controlling department number, and the manager’s last name.</td><td>S ← $\sigma_{Loc='Warsaw'}$(PROJ)
C ← (S $\theta_{DNum=DNum}$ DEPT)
P ← (C $\theta_{MSSN=SSN}$ EMP)
Result ← $\pi_{PNum, DNum, LName}$ (P)</td><td>Q:[t.PNum, t.DNum, m.LName | PROJ(t) AND EMP(m) AND p.Loc = ‘Warsaw’ AND ((∃d) (DEPT(d) AND p.Dnum=d.Dnum AND d.DNum =t.DNumEmp AND d.MSSN=m.SSN))}</td></tr>
<tr><td colspan="3">SELECT PNum, DNum, LName
FROM PROJ JOIN DEPT ON DNum=DNum JOIN EMP ON MSSN=SSN
WHERE PROJ.Loc = ‘Warsaw’;</td></tr>
</table>

Table 6-2: Examples of Relational Algebra and Tuple Relational Algebra Expressions

An equivalent SQL statement is included for clarity. In the examples, relational algebra operations are defined as follows; project (π), restrict (σ), join(θ), as well as the production symbol (←). The tuple relational calculus uses fewer mnemonics that include a tuple (shown in italics), existential operators (∀) and (∃), as well as typical Boolean operators.

المنارة للاستشارات
www.manaraa.com

Relational calculus is very important because it permits the use of a mathematics upon which the results of queries can be proven to be correct. Relational calculus can therefore be used to prove the correctness of a database and thus is the formal language used to define relational databases. In fact, elements of tuple relational calculus are present in the SQL query language.

Why do we have two mathematics for relational theory? Simply stated, relational calculus is best for stating what you want and relational algebra is best for stating how to accomplish it. Relational algebra and relational calculus are expressively equivalent [Elma03]. In fact, any expression that can be represented by one language can be represented by the other and vice-versa[13]. This has led to the concept of relationally complete, which means any query represented in relational algebra can also be represented in relational (tuple) calculus.

6.2.3.2 Query Trees

A query tree is a tree structure that corresponds to a query, where leaf nodes of the tree contain nodes that access a relation and internal nodes with zero, one or more children. The nodes contain the relational operators. These operators include project (depicted as π), restrict (depicted as σ), and join (depicted as either θ or $\bowtie$[14]). The edges of a tree represent data flow from bottom to top, *i.e.*, from the leaves, which correspond

[13] Also known as query equivalence [Yann95].

[14] Strangely, few texts give explanations for the choice of symbol. Traditionally, θ represents a theta-join and $\bowtie$ represents a natural j oin, but most texts interchange these concepts resulting in all joins represented using one or the other symbol (and sometimes both).

المنارة للاستشارات
www.manaraa.com

to data in the database, to the root, which is the final operator producing the query results [Ioan96]. Figure 6-3 depicts an example of a query tree.

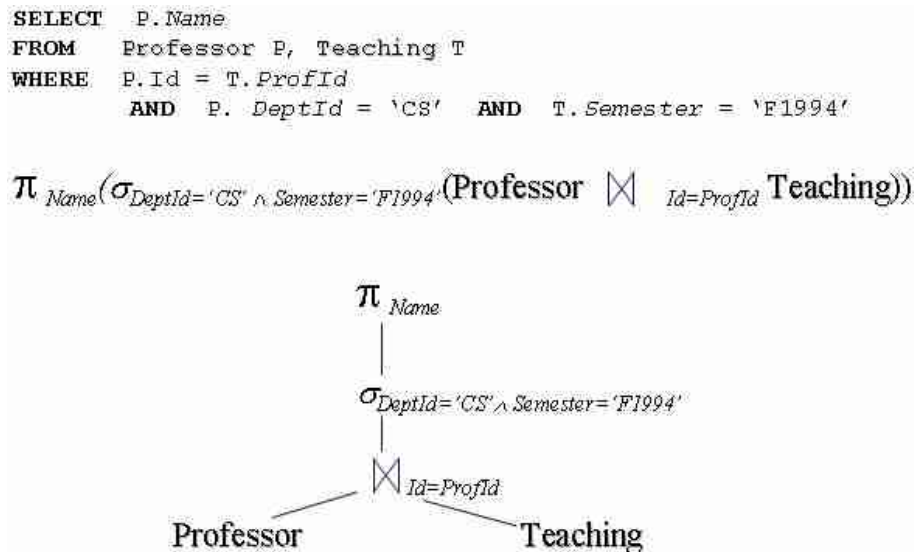


Figure 6-3: Query Tree Example[15]

An evaluation of the query tree consists of evaluating an internal node operation whenever its operands are available and passing the results from evaluating the operation up the tree to the parent node. The evaluation terminates when the root node is evaluated and replaced by the tuples that form the result of the query [Elma03, Bill04].

Data structures have been suggested to improve the way queries can be optimized and executed [Elma03]. This work presents such a method using a variant of the query tree structure. The advantages of using this mechanism versus a relational calculus internal representation are shown in Table 6-3.

---

[15] Although this drawing has appeared in several places in the literature, it contains a subtle nuance of database theory that is often overlooked. Can you spot the often misused trait? Hint: what is the domain of the semester attribute?

المنارة للاستشارات
www.manaraa.com

Operational Requirement	Query Tree	Relational Calculus
Can it be reduced?	Yes. It is possible to prune the query tree prior to evaluating query plans.	Only through application of algebraic operations.
Can it support execution?	Yes. The tree can be used to execute queries by passing data up the tree.	No. Requires translation to another form.
Can it support relational algebra expressions?	Yes. The tree lends itself well to relational algebra.	No. Requires conversion.
Can it be implemented in database systems?	Yes. Tree structures are a common data structure.	Only through designs that model the calculus.
Can it contain data?	Yes. The tree nodes can contain data, operations and expressions.	No. Only the literals and variables that form the expression.

Table 6-3: Internal Representation Requirements

Clearly, the query tree internal representation is superior to the more traditional mechanism employed in modern database systems. For example, the internal representation in MySQL is that of a set of classes and structures designed to contain the query and its elements for easy (fast) traversal. It shows no consideration for any of the above requirements.

6.2.4 Optimizers

This section describes the basic types of query optimizers, their uses, weaknesses, and application. There are four primary means of performing query optimization. These include 1) cost-based optimization, 2) heuristic optimization, 3) semantic optimization, and 4) parametric optimization. While no optimization technique can guarantee the best execution plan, the goal of all of these methods is to generate an efficient execution for the query that guarantees correct results.

المنارة للاستشارات

www.manaraa.com

The first query optimizers were designed for use in early database systems such as System R[16] [Seli79] and INGRES [Ston76]. These optimizers were developed for a particular implementation of the relational model and have stood the test of time as illustrations for how to implement optimizers. Many of the commercially available database systems are based on these works. Since then, optimizers have been created for extensions of the relational model to include object-oriented and distributed database systems.

One example is the Volcano optimizer which uses a dynamic programming algorithm [Seli79] to generate query plans for cost-based optimization in object-oriented database systems [Grae93b]. Another example is concerned with how to perform optimization in heterogeneous database systems (similar to distributed systems, but there is no commonly shared concept of organization). In these environments it is possible to use statistical methods for deriving optimization strategies [Spee93].

Another area in which the requirements for query optimization generate unique needs is that of memory-resident database systems. Memory resident database systems are designed to contain the entire system and all of the data in the computer's secondary memory (disk). While most of these applications are in the area of embedded systems, some larger distributed systems comprised of a collection of systems use memory resident databases to expedite information flow. Optimization in memory resident database systems requires faster algorithms because the need for optimizing retrieval is

[16] Considered by some to be the "Bible of Query Optimization" [Ston98].

www.manaraa.com

insignificant compared to the need for processing the query itself[17]. Researchers have concentrated on forming specialized high-speed algorithms for these systems [Whan90].

All of the research into traditional and non-traditional optimization is based on the firmament of the System R optimizer. The System R optimizer is a cost-based optimizer that uses information gathered about the database and the data in the relations (statistics) to form cost estimates for how the query would perform. Additionally, the concept of arranging the internal representation of the query into different but equivalent (they generate the same answer) internal representations provides a mechanism to store the alternative forms. Each of these alternative forms is called a query plan. The plan with the least cost is chosen as the most efficient way to execute the query [Seli79].

One of the key features identified in the System R work [Seli79] was the concept of selectivity – the prediction of results based on the evaluation of an expression that contained references to attributes and their values. Selectivity is central to determining in what order the simple expressions in a conjunctive selection should be tested. The most selective expression (that is, the one with the smallest selectivity) will retrieve the smallest number of tuples (rows). Thus, that expression should be the basis for the first operation in a query [Silb96]. Conjunctive selections can be thought of as the "intersection" conditions. Conversely, disjunctive selections are the "union" conditions. Order has no affect among the disjunctive conditions.

[17] Query execution in traditional systems includes not only processing the query but also accessing the data from physical media. However, memory resident systems do not have the long access times associated with retrieval from physical media.

www.manaraa.com

Certain query optimizers, such as System R [Cham81a], do not process all possible join orders. Rather, they restrict the search to certain types of join orders that are known to produce more efficient execution. For example, multi-way joins might be ordered so that the conditions that generate the least possible results are performed first. Similarly, the System R optimizer considers only those join orders where the right operand of each join is one of the initial relations. Such join orders are called left-deep join orders. Left-deep join orders are particularly convenient for pipeline execution, since the right operand is normally a relation (versus an intermediate relation), and thus only one input to each join is pipelined [Silb96]. The use of pipelining is a key element of the ALV optimizer and execution engine.

6.2.4.1 Cost-based Optimizers

A cost-based optimizer generates a range of query-evaluation plans from the given query by using the equivalence rules, and chooses the one with the least cost based on the metrics (statistics) gathered about the relations and operations needed to execute the query. For a complex query, many equivalent plans are possible [Silb96].

The goal of cost-based optimization is to arrange the query execution and table access utilizing indexes and statistics gathered from past queries [Date04]. Systems such as Microsoft SQL Server [Micr03] and Oracle [Orac05] use cost-based optimizers.

The portion of the database system responsible for acquiring and processing statistics (and many other utility functions) is called the database catalog. The catalog maintains statistics about the referenced relations, and the access paths available on each

المنارة للاستشارات
www.manaraa.com

of them. These will be used later in access path selection to select the most efficient (least cost) plan [Seli79]. For example, System R maintains statistics on the following [Seli79]:

- For each relation store
    - The cardinality of each relation
    - The number of pages in the segment that hold tuples of each relation
    - The fraction of data pages in the segment that hold tuples of relation (blocking factor or fill)
- For each index I on each relation store
    - The number of distinct keys in each index
    - The number of pages in each index

These statistics come from several sources within the system. The statistics are created when a relation is loaded and when an index is created. They are then updated periodically by a user command[18], which can be run by any user. System R does not update these statistics in real time because of the extra database operations and the locking bottleneck this would create at the system catalogs. Dynamic updating of statistics would tend to serialize accesses that modify the relation contents and thus limit the ability of the system to process simultaneous queries in a multi-user environment [Seli79].

[18] This practice is still in use today by most commercial database systems [Micr03, Orac05].

www.manaraa.com

The use of statistics in cost-based optimization is not very complex [Seli79]. Most database professionals interviewed seem to think the gathering and application of statistics to be a complex and vital element of query optimization. Whereas it is true that cost-based query optimization and even hybrid optimization schemes use statistics for cost and/or ranking, they are neither complex nor critical. Take for instance the concept of evenly distributed values among attributes. This concept alone is proof of the fuzzy nature of the application of statistics. Statistical calculations are largely categorical in nature and not designed to generate a precise value. They merely assist in determining whether one query execution plan is generally more costly than another [Rama03].

Frequency distribution of attribute values is a common method for predicting the size of query results. By forming a distribution of possible (or actual[19]) values of an attribute, the database system can use the distribution to calculate a cost for a given query plan by predicting the number of tuples (rows) that the plan must process. Practical DBMSs, however, deal with frequency distributions of individual attributes only, because considering all possible combinations of attributes is very expensive. This essentially corresponds to what is known as the attribute value independence assumption, and although rarely true, it is adopted by all current DBMSs [Date04].

Gathering the distribution data requires either constant updating of the statistics or predictive analysis of the data. Another tactic is the use of uniform distributions where the distribution of the attribute values is assumed to be equal for all distinct values. For example, given 5000 tuples and a possible 50 values for a given attribute, the uniform

[19] The accumulation of statistics real -time is called piggy back statistic generation [Zhu98].

المنارة للاستشارات
www.manaraa.com

distribution assumes each value is represented 100 times [Date04]. This is rarely the case and is often incorrect [Silb96]. However, given the absence of any statistics, it is still a reasonable approximation of reality in many cases [Silb96].

The memory requirements and running time of dynamic programming grow exponentially with query size (i.e., number of joins) in the worst case since all viable partial plans generated in each step must be stored to be used in the next one. In fact, many modern systems place a limit on the size of queries that can be submitted (usually around fifteen joins), because for larger queries the optimizer crashes due to very high memory requirements. Nevertheless, most queries seen in practice involve less than ten joins, and the algorithm has proved to be very effective in such contexts. It is considered the standard in query optimization search strategies.

The relevant statistics gathered about relations for use in cost-based optimizers include [Silb96]:

- n_r the number of tuples in the relation r
- b_r the number of blocks containing tuples in the relation r
- s_r the size of a tuple of relation r in bytes
- f_r the blocking factor of relation r
- V(A, r) the number of distinct values that appear in relation r for attribute A. This value is the same as the size $\Pi_A(r)$. If A is a key for relation r, V(A, r) is n_r.
- SC(A, r) the selection cardinality of attribute A of relation r. Given a relation r and an attribute A of the relation, SC(A, r) is the average number of records that

المنارة للاستشارات
www.manaraa.com

satisfy an equality condition on attribute A, given that at least one record satisfies the equality condition. For a key attribute, SC(A, *r*) is 1; for non key attribute, we estimate that the V(A, *r*) distinct values are distributed evenly among the tuples, yielding SC(A, *r*) = (n_r / V(A, *r*)).

- f_i is the fan-out of internal nodes of index *i*
- HT_i is the height of the B+ Tree for index *i*
- LB_i is the number of lowest-level index blocs in index *I* – the number of blocks at the leaf level of the index.

The cost of writing the final result of an operation back to disk is ignored. Whatever the query-evaluation plan used, this cost does not change; thus not including it in the calculations does not affect the choice of the plan [Silb96].

Most database systems today use a form of dynamic programming to generate all possible query plans. While dynamic programming offers good performance for cost-optimization, it is a complex algorithm that can require more resources for the more complex queries. While most database systems do not encounter these types of queries, researchers in the areas of distributed database systems and high performance computing have explored alternatives and variants to dynamic programming techniques. The recent research by Kossmann and Stocker [Koss00] shows that we are beginning to see the limits of traditional approaches to query optimization. What are needed are more efficient optimization techniques that generate efficient execution plans that follow good practices rather than exhaustive exploration. In other words, we need optimizers that perform well

المنارة للاستشارات
www.manaraa.com

in a variety of general environments as well as optimizers that perform well in unique database environments.

6.2.4.2 Heuristic Optimizers

The goal of heuristic optimization is to apply rules that ensure "good" practices for query execution [Rama03]. Systems that use heuristic optimizers[20] include INGRES [Ston76] and various academic variants.

Heuristic optimizers use rules concerning how to shape the query into the most optimal form prior to choosing alternative implementations. The application of heuristics, or rules, can eliminate queries that are likely to be inefficient [Ioan97]. Using heuristics as a basis to form the query plan ensures that the query plan is most likely (but not always) optimized prior to evaluation.

Such heuristics are:

- Perform selection operations as early as possible. It is usually better to perform selections earlier than projections because they reduce the number of tuples to be sent up the tree.
- Perform projections early.
- Determine which selection operations and join operations produce the smallest result set and use those first (left-most-deep).
- Replace cartesian products with join operations.

[20] Most systems typically use heuristic optimization as a means of avoiding the really bad plans rather than as a primary means of optimization.

المنارة للاستشارات

www.manaraa.com

- Deconstruct and move as far down as possible lists of projection attributes.
- Identify subtrees whose operations can be pipelined.

Heuristic optimizers are not new technologies. Researchers have created rules-based optimizers for various specialized purposes. One example is the Prairie rule-based query optimizer [Das95]. This rule-based optimizer permits the creation of rules based on a given language notation. Queries are processed using the rules to govern how the optimizer performs. In this case, the Prairie optimizer is primarily a cost-based optimizer that uses rules (heuristics) to tune the optimizer.

Aside from examples like Prairie and early primitives such as INGRES, no commercial database systems implement a purely heuristic optimizer. For those that do have a heuristic or rule-based optimization step it is usually implemented as an addition to or as a pre-processor to a classic cost-based optimizer [Das95] or as a pre- post-processing step in the optimization strategy [MySQ05].

6.2.4.3 Semantic Optimizers

The goal of semantic optimization is to form query execution plans that use the semantics, or topography, of the database and the relationships and indexes within to form queries that ensure the best practice available for executing a query in the given database. Chakravarthy explains this best, saying that the semantic query optimization uses knowledge of the schema (*e.g.*, integrity constraints) for transforming a query into a form that may be answered more efficiently than the original version. Chakravarthy

المنارة للاستشارات
www.manaraa.com

shows how "…semantic query optimization techniques can be extended to databases that support recursion and integrity constraints that contain disjunction, negation, and recursion," [Chak90].

Although not yet implemented in commercial database systems as the primary optimization technique, semantic optimization is currently the focus of considerable research. Semantic optimization operates on the premise that the optimizer has a basic understanding of the actual database schema. When a query is submitted, the optimizer uses its knowledge of system constraints to simplify or to ignore a particular query if it is guaranteed to return an empty result set. This technique holds great promise for providing even more improvements to query processing efficiency in future relational database systems.

6.2.4.4 Parametric Optimizers

Ioannidis, in his work on parametric query optimization, describes a query optimization method that combines the application of heuristic methods with cost-based optimization. The resulting query optimizer provides a means to produce a smaller set of effective query plans from which cost can be estimated, and thus the lowest cost plan of the set can be executed [Ioan97]. Query plan generation is created using a random algorithm, called sipR. This permits systems that utilize parametric query optimization to choose query plans that can include the uncertainty of parameter changes (such as buffer sizes) to choose optimal plans either formed on the fly or from storage [Ioan97].

المنارة للاستشارات
www.manaraa.com

It is interesting to note that in his work, Ioannidis suggests that the use of dynamic programming algorithms may not be needed and thus the overhead in using these techniques avoided. Furthermore, he found that database systems that use heuristics to prune or shape the query prior to applying dynamic programming algorithms for query optimization are usually an enhanced and version of the original algorithm of System R. Ioannidis showed that for small queries (approximately up to ten joins), dynamic programming is superior to randomized algorithms, whereas for large queries the opposite holds [Cole94, Ioan97].

6.2.5 Query Execution

There are many methods that database systems can use to execute queries. Most database systems use either an iterative or interpretative execution strategy [Rama03].

Iterative methods provide ways of producing a sequence of calls available for processing discrete operations (e.g., join, project, etc.), but are not designed to incorporate the features of the internal representation. Translation of queries into iterative methods uses techniques of functional programming and program transformation. There are several algorithms that generate iterative programs from algebra-based query specifications. One example translates query specifications into recursive programs which are simplified by sets of transformation rules before the algorithm generates an execution plan. Another algorithm uses a two-level translation. The first level uses a smaller set of transformation rules to simplify the internal representation and the second level applies functional transformations prior to generating the execution plan [Frey89].

المنارة للاستشارات
www.manaraa.com

The implementation of this mechanism [Frey86] creates a set of defined compiled functional primitives, formed using a high-level language, that are then linked together via a call stack or procedural call sequence. When a query execution plan is created and selected for execution, a compiler (usually the same one used to create the database system) is used to compile the procedural calls into a binary executable. Due to the high cost of the iterative method, compiled execution plans are typically stored for reuse for similar or identical queries [Date04, Frey86].

Interpretative methods on the other hand form query execution using existing compiled abstractions of basic operations. The query execution plan chosen is reconstructed as a queue of method calls that are each taken off the queue, processed, and the results placed in memory for use with the next or subsequent calls. Implementation of this strategy is often called “lazy evaluation” because the set of available compiled methods is not optimized for best performance, rather they are optimized for generality [Frey86].

Query processing and execution in MySQL is of the interpretive variety. It is implemented within the threaded implementation architecture whereby each query is given its own thread of execution. Figure 6-4 depicts a block diagram that describes the MySQL query processing methodology.

المنارة للاستشارات
www.manaraa.com

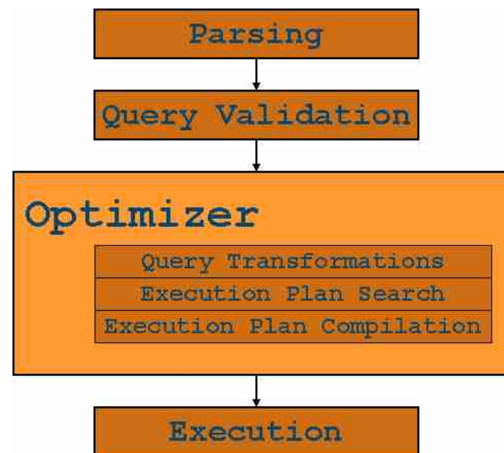


Figure 6-4: The MySQL Query Processing Methodology[21]

When a client issues a query, a new thread is created and the SQL statement is forwarded to the parser for syntactic validation (or rejection due to errors). The MySQL parser is implemented using a large lex-yacc[22] script [John95, Lesk90] that is compiled with Bison [Donn02]. The parser constructs a query structure used to represent the query statement (SQL) in memory as a data structure that can be used to execute the query. Once the query structure is created, control passes to the query processor which performs checks such as checking tables and security access. Once the required access is granted and the tables are opened (and locked if the query is an update), control is passed to individual methods that execute the basic query operations such as select, restrict, and project. Optimization is applied to the data structure by ordering the lists of tables and operations to form a more efficient query based on common practices. This form of

---

[21] Drawing borrowed from the MySQL Query Optimizer presentation by Widenius [Wide03].
[22] Lex stands for "lexical analyzer generator" and is used as a parser to identify tokens and literals as well as syntax of a language. Yacc stands for "yet another compiler compile r" and is used to identify and act on the semantic definitions of the language. The use of these tools together with Bison (a yacc compiler) provides a rich mechanism of creating subsystems that can parser and process language commands. Indeed, that is exa ctly how MySQL uses these technologies.

www.manaraa.com

optimization is called a select-project-join query processor. The results of the query operations are returned to the client using established communication protocols and access methods [Badi02].

One area that is often confusing is the concept of what “compiled” means. In Frey’s work [Frey86], a compiled query is an actual compilation of an iterative query execution plan, but in Date’s work [Date04], a compiled query is simply one that has been optimized and stored for future execution[23]. As a result, one must take care when considering a compiled query. In this work, the use of the word “compiled” is avoided because the query optimizer and execution engine do not store the query execution plan for later reuse nor does the query execution require any compilation or assembly to work [Frey86].

Another area of interest is called extensible query optimization. Extensible query optimization is the application of alternative methods or a repertory of alternative strategies for performing query optimization. In many ways, systems that include extensible query optimization techniques employ multiple query optimization strategies [Lohm88]. However, this term is dated and generally implies that the database system uses a modularized or tunable mechanism to ensure that query optimization is performed using the most appropriate techniques in a given database. In contrast, the ALV query optimizer uses a heuristic optimizer as a primary method followed by the application of available indexed access paths as a secondary optimization step.

[23] The concept of a stored procedure fits this second category – it is compiled (optimized) for execution at a later date and can be run many times on data that meets its input parameters.

المنارة للاستشارات

www.manaraa.com

6.3 ALV Query Optimizer and Execution Engine

The ability to translate queries into an executable form involves the application of query optimization to ensure an efficient execution plan and a fast execution engine. Imperative to the success of these technologies is an effective internal representation. This section describes the technology created to implement these technologies and the problems encountered along the way.

The ALV query optimizer does not resemble the SELECT-PROJECT-JOIN optimizer of MySQL (described above). It is a heuristic query optimizer designed to operate on a specialized internal structure that is a variation of a query tree. The query tree is a tree structure designed to contain the operations of a query at the nodes and the relations at the leaves [Tuck04]. The choice of a tree structure enables fast traversal of the structure using well known tree traversal algorithms [Baas88, Knut97]. Furthermore, it also permits execution of the query also by traversing the tree from the bottom-up, producing results via the root node. The following sections describe the ALV query tree, optimizer, and execution engine in detail.

6.3.1 Technology Descriptions

The ALV query optimizer and execution engine are implemented as classes within the C++ program. Starting from the lowest level, a class was created to model a query tree that implements the tree structure. The execution engine is implemented as a class that consumes the query tree and manipulates it in place without transformation.

المنارة للاستشارات
www.manaraa.com

These last two features are paramount to the efficient execution of the query and add considerable benefit to the effectiveness of the versioning system.

6.3.1.1 ALV Query Tree

The ALV query tree is a tree data structure that uses a node structure that contains all of the parameters necessary to represent the following operations:

- Restriction – provides the ability to include results that match an expression of the attributes
- Projection – provides ability to select attributes to include in result set
- Join – provides the ability to combine two or more relations to form a composite set of attributes in the result set
- Sort (order by) – provides ability to order the result set
- Distinct – provides the ability to reduce the result set to unique tuples

Projection, Restriction, and Join are the basic operations. Sort and distinct are provided as additional utility operations that assist in the formulation of a complete query tree (all possible operations represented as nodes). Join operations can have join conditions (theta joins) or no conditions (equijoins). The join operation is subdivided into the following operations:

- Inner – the join of two relations returning tuples where there is a match

المنارة للاستشارات

www.manaraa.com

- Outer (left, right, full) – return all rows from at least one of the tables or views mentioned in the FROM clause, as long as those rows meet any WHERE search conditions. All rows are retrieved from the left table referenced with a left outer join, and all rows from the right table referenced in a right outer join. All rows from both tables are returned in a full outer join. Values for attributes of non-matching rows are returned as null values.
- Rightouter – the join of two relations returning tuples where there is a match plus all tuples from the relation specified to the right leaving non-matching attributes specified from the other relation empty (null).
- Fullouter – the join of two relations returning all tuples from both relations leaving non-matching attributes specified from the other relation empty (null).
- Crossproduct – the join of two relations mapping each tuple from the first relation to all tuples from the other relation
- Union – the set operation where only matches from two relations with the same schema are returned
- Intersect – the set operation where only the non-matches from two relations with the same schema are returned

While the ALV query tree provides the union and intersect operations, the MySQL parser does not currently support such operations. Further modification of the MySQL parser is necessary to implement these operations.

المنارة للاستشارات
www.manaraa.com

The ALV query tree class also contains the methods used in the ALV query optimizer. That is, there exist HOptimization() and COptimization() methods that implement the heuristic and cost optimization processes respectively.

6.3.1.2 ALV Query Optimizer

When designing the ALV execution engine, it became apparent that little research has been done on the creation of an execution engine that complements an internal structure while remaining an abstraction. Thus, the ALV optimizer is implemented as part of the ALV query tree. The optimizer is implemented as a two-pass operation where the first operation rearranges the tree for execution using a heuristic algorithm. The second pass walks the tree, changing the access method for nodes that have relations with indexes available on the attributes being operated on.

The heuristic optimization process uses a set of rules that have been defined to guarantee "good" execution plans (see section 6.3.1.3 below). The following summarizes the algorithm used to implement the HOptimization() method.

1. SplitRestrictWithJoin() – Find all join nodes that are also restrictions (join nodes that have a where clause) and split into a join with restriction(s) as children.
2. SplitProjectWithJoin() – Find all join nodes that are also projections (join nodes that have an attribute list) and split into a join with projection(s) as children.
3. SplitRestrictWithProject() – Find all restrict nodes that have an attribute list and split into a project with a restrict as the left child.

المنارة للاستشارات
www.manaraa.com

4. FindRestriction() – Find a node with restrictions and push down the tree using a recursive call. Continue until you get the same node twice. This means that the node cannot be pushed down any further.
5. FindProjection() – Find a node with projections and push down the tree using a recursive call. Continue until you get the same node twice. This means that the node cannot be pushed down any further.
6. FindNaturalJoin() – Remove all cross products.
7. PruneTree() – Prune the tree of "blank" nodes. Blank nodes are; 1) projections without attributes that have at least 1 child, 2) restrictions without expressions, but not having two children.
8. RemoveDistinct() – Lastly, check to see if there exists a DISTINCT option. If so, create a new node that is a DISTINCT operation.

The default access method for all relations is a file scan. The cost optimization process walks the query tree looking for leaf nodes and examines the relations specified. If an index is available for any of the attributes in the expressions or projections, the index is used as the access method.

Together, these two methods (the storage of the query as a qquery tree and storing the query operations in the nodes) enable manipulation of the query tree structure to form a query plan that can be executed directly. The most important concept is the omission of generating alternative query plans. By ensuring good practices (rules) are used to form the query, only one pass is necessary and therefore the heuristic optimization process,

المنارة للاستشارات

www.manaraa.com

combined with the index access method application, provides an efficient query optimization process.

6.3.1.3 Rules for Query Tree Optimizations

In the pursuit of generating a heuristic optimizer based on a query tree, it became apparent that the optimization is only as good as its rules. Thus, the following paragraphs describe the rules used to create the ALV query optimizer. Although these rules are very basic, when applied to typical queries, the resulting execution is near-optimal with fast performance and accurate results.

There are some basic strategies that were used to construct the query tree initially. Specifically, all executions take place in the query tree node. Selections and projections are processed on a branch and do not generate intermediate relations. Joins are always processed as an intersection of two paths. A multi-way join would be formed using a series of two-way joins. Lastly, the tree is left deep, or left biased, for generation of new nodes.

Rule 1) Push all restrictions down the tree to leaves. Expressions are grouped according to their respective relations into individual query tree nodes. Although there are some complex expressions that cannot be reduced, most can be easily reduced to a single relation. By placing the restrictions at the leaves, the number of resulting tuples that must be passed up the tree is reduced.

Rule 2) Place all projections at the lowest point in the tree. Projections must be placed in a node above restrictions and can further reduce the amount of data passed

المنارة للاستشارات
www.manaraa.com

through the tree by eliminating unneeded attributes from the resulting tuples. It should be noted that the projections may be modified to include attributes that are needed for operations that reside in the parentage of the projection query tree node.

Rule 3) Place all joins at intersections of projections or restrictions of the relations contained in the join clause[24]. This ensures that the least amount of tuples are evaluated for the most expensive operation of all the join [Date04]. Intermediate query tree nodes may be necessary that order the resulting tuples from the child nodes. These intermediate nodes, called utility operations, may sort or group the tuples depending on the type of join.

Rule 4) Split any nodes remaining after rules 1-3 are applied that contain a project and join or a restrict and join. This step is necessary because some queries specify the join condition in the where clause[25] and thus can "fool" the optimizer into forming join nodes that have portions of the expressions that are not part of the join condition.

An interesting counter argument to the practice of pushing selections and restrictions down the tree is given by Lee, Shih, and Chen [Lee01]. In their work, they suggest that under some conditions selections and projections are executed may be more costly than joins. Their argument presents a query optimizer based on graph theory that can more accurately predict query optimization for situations where complex selects and projections are present. Nevertheless, the general case is that "good" execution plans can be constructed for the majority of queries using the rules listed above for optimization.

[24] May disallow the use of in dexes for the join operation.
[25] A common technique practiced by novice SQL writers and utterly loathed by the author.

www.manaraa.com

In summary, the ALV optimizer is designed to apply the rules above in order to transform the query tree into a form that ensures efficient execution[26].

6.3.2 ALV Query Execution

Query execution in ALV is accomplished using the optimized query tree. The tree structure itself is used as a pipeline for processing the query. When a query is executed, a GetNext() method is issued on each of the children of the root node. Another GetNext() method is called on each of their children. This process continues as the tree is traversed to the lowest level of the tree containing a reference to a single relation. The operation for that node is executed for one row in the relation. Upon completion, the result of that operation passes up the result to the next operation in the tree. If no result is produced, control remains in the current node until a result is produced. As the tree is being climbed back to the root, the results are passed to each parent in turn until the root node is reached. Once the operation in the root node is complete, the resulting tuple is passed to the client. In this way, execution of the query appears to produce results faster because data (results) are shown to the client much earlier than if the query were to be executed for the entire set of operations before any results are given to the client.

6.3.3 Class Descriptions

Tree structures are covered extensively in many areas of research. However, few resources are available that examine the details of internal representations of queries in

[26] In this case, efficient execution may not be the optimal solution.

المنارة للاستشارات
www.manaraa.com

database systems. The following sections describe the major code implementations and classes created to implement the query optimizer and execution engine.

6.3.3.1 Query Transformation

The MySQL parser was modified to identify and parse the ALV SQL commands. Rather than replace the parser code with code that parsed the command into the ALV data structure, it was decided to minimize changes to the parser. Instead the system creates a MySQL internal representation and converts that form into the ALV internal representation. Although this process adds some execution time and requires a small amount of extra computational work, the implementation simplified the modifications to the parser and provided a common mechanism to compare the ALV data structure to that of the MySQL data structure.

The process of transformation[27] begins in the MySQL parser, which identifies commands as being ALV commands. The system then directs control to a class named ALV_SQL_Parse.cpp[28] that manages the transformation of the parsed query from the MySQL form to the ALV internal representation. This is accomplished by a method named BuildALVQueryTree in the ALV_SQL_Parse class. This method is called only for SELECT and EXPLAIN SELECT statements. All other statements are transformed from the MySQL data structure and executed inline.

---

[27] Although many texts on the subject of query processing disagree abo ut how each process is differentiated, they do agree that certain distinct process steps must occur .

[28] Named after the equivalent class in MySQL. The class is misleading because no parsing takes place in the class.

المنارة للاستشارات

www.manaraa.com

6.3.3.2 ALV QueryTree

The heart of the ALV query optimizer is the ALV internal representation data structure. It is used to represent the query once the SQL command has been parsed and transformed.

This structure is implemented as a tree structure, (hence the name query tree), where each node has 0, 1, or 2 children. Nodes with 0 children are the leaves of the tree, those with 1 child represent internal nodes that perform unary operations on data, and those with 2 children are join operations. The actual node structure from the source code is show in Figure 6-5 below.

```
struct QueryNode
[
      QueryNode();
      QueryNode(const QueryNode &o);
      ~QueryNode();
      int                   NodeId;
      int                   ParentNodeId;
      bool                  SubQuery;
      Str                   Child;
      QueryNodeType         NodeType;
      TypeJoin              JoinType;
      JoinConType           JoinCondition;
      Expr::Expr            *where_expr;
      Expr::Expr            *join_expr;
      Relation              *Relations[MAXNODETABLES];
      float                 Cost;
      long                  Size;
      bool                  PreemptPipeline;
      Attribute             *Attributes;
      QueryNode             *Left;
      QueryNode             *Right;
};
```

Figure 6-5: The ALV Query Tree Node Structure

Contained within the query node are variables that contain the elements for query operations. Each variable is described briefly below.

www.manaraa.com

- NodeId -- the internal id number for a node
- ParentNodeId -- the internal id for the parent node (used for insert)
- SubQuery -- is this the start of a subquery?
- Child -- is this a Left or Right child of the parent?
- NodeType -- synonymous with operation type
- JoinType -- if a join, this is the join operation
- JoinConType -- if this is a join, this is the "on" condition
- Expressions -- the expressions from the "where" clause for this node
- Relations[] -- the relations for this operation (at most 2)
- Cost -- what is the calculated cost to execute this node?
- Size -- what is the estimated size of the result set for this node?
- PreemptPipeline -- does the pipeline need to be halted for a sort?
- SelIndex -- the indexes applied to this operation
- Attributes -- the attributes for the result set of this operation
- Left -- a pointer to the left child node
- Right -- a pointer to the right child node

Some of these variables are used to manage node organization and form the tree itself. Two of the most interesting are NodeID and ParentNodeID. These are used to establish parentage of the nodes in the tree. This is necessary as nodes can be moved up

www.manaraa.com

and down the tree. The use of a parent node id variable avoids the need to maintain reverse pointers in the tree[29].

The SubQuery variable is used to indicate the starting node for a subquery[30]. Thus, the data structure can support nested queries (subqueries) without additional modification of the structure. The only caveat is that the algorithms for optimization are designed to use the subquery indicator as a stop condition for tree traversal. That is, when a subquery node is detected, optimization considers the subquery a separate entity. Once detected, the query optimization routines are re-run using the subquery node as the start of the next optimization. Thus any number of subqueries can be supported and represented as subtrees in the tree structure. This is an important feature of the query tree that overcomes the limitation found in many internal representations [MySQ05].

The Expressions variable is a pointer to an Expression class that manages a typical expression tree [Knut97]. While mundane in implementation, the details of the expression class are beyond the scope of this work. This variable is used by both restriction and join operations.

The Relations array is used to contain pointers to relation classes (see Chapter 4) that represent the abstraction of the ALV clustered version store. The array size is currently set at 4. The first two positions (0 and 1) correspond to the left and right child respectfully. The next two positions (2 and 3) represent temporary relations such as

[29] A practice strongly discouraged by Knu th and other algorithm gurus [Corm01, Knut97].
[30] Subqueries were added to MySQL in 2002. Changes to the internal data structures were necessary to support this [Badi02]

www.manaraa.com

reordering (sorting) and the application of indexes. The relation class permits the query tree to process queries against either a normal MySQL query (SQL) or an ALV query.

The Cost and Size variables are used to gather information about the potential efficiency of the query. Although not used in the current implementation, these variables are in place for future expansion of the ALV subsystem to provide a mechanism for query caching.

Attributes is a pointer to a class that abstracts the behavior of an attribute (see Chapter 4). It is useful in projection operations and maintaining attributes necessary for operations on relations (e.g., the propagation of attributes that satisfy expressions but are not part of the result set).

The last variable of interest is the PreemptPipeline variable, which is used by the ALVExecute class to permit the execution of loops within the tree execution. Loops are necessary anytime an operation requires iteration through a child node. For example, a join that joins 2 relations on a common attribute in the absence of indexes that permit ordering may require iteration through one or both child nodes in order to achieve the correct mapping (join) operation.

This class is also responsible for query optimization (described in 6.3.1.2 above). Since the query tree class provides all tree operations for manipulating the tree and since query optimization is also a set of tree operations, optimization was built as methods in the query tree class.

Heuristic optimization is implemented via a method that implements the heuristic algorithm described above. Execution of this algorithm results in the relocation of tree

المنارة للاستشارات
www.manaraa.com

nodes into more efficient tree orders and the separation of some nodes into two or more others that can also be relocated to form a more optimal tree.

Cost optimization is also supported in this class using an algorithm that walks the tree applying available indexes to the access methods for each leaf node (nodes that access the relation stores directly).

This structure can support a wide variety of operations including restrict, project, join, set, and ordering (sorting). The query node structure is designed to represent each of these operations as a single node and can store all pertinent and required information to execute the operation in place. Furthermore, the explain command was implemented as a simple traversal of the tree, printing out the contents of each node starting at the leaves (see section 6.3.4.11 below). The MySQL equivalent of this operation requires much more computational time and is implemented with a complex set of methods.

Thus, the query tree is an internal representation that can not only represent any query, but also provides a mechanism to optimize the query without changing the internal representation. Indeed, the tree structure itself simplifies optimization and enables the implementation of a heuristic optimizer by providing a means to associate query operations as nodes in a tree. This query tree therefore is a viable mechanism for use in any relational database system and can be generalized for use in a production system.

6.3.3.3 ALVExecute

The ALVExecute class is designed to work with the query tree class as the primary data item upon which to operate. Figure 6-6 presents a simplified sequence of the

المنارة للاستشارات
www.manaraa.com

major object classes and messages passed during the execution of an ALV command (query).

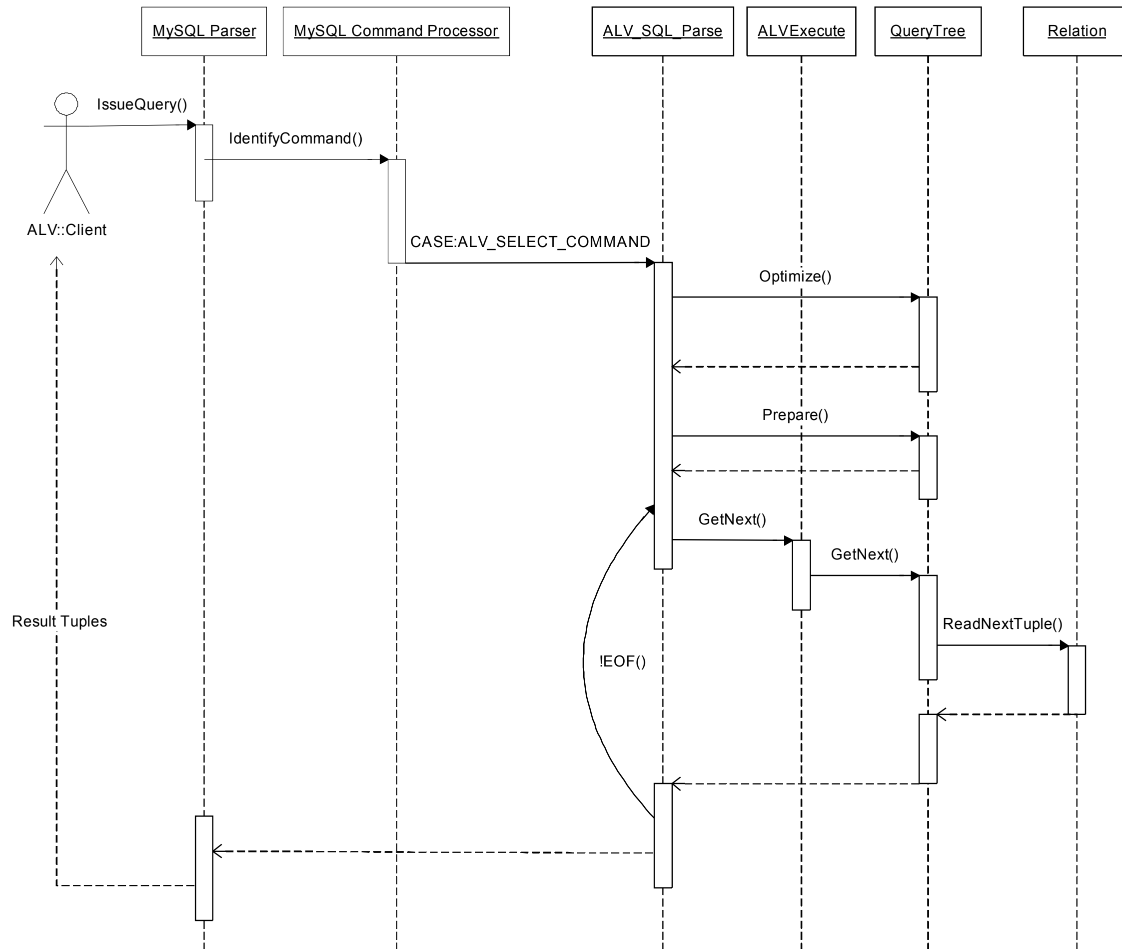


Figure 6-6: Simplified ALV Query Execution Sequence

A Prepare method is called at the start of query execution. The Prepare method walks the query tree, opening all of the relations at the leaves and establishes any required temporary relations. Execution is accomplished by using a while loop that iterates through the result set issuing a pulse to the tree starting at the root node. A pulse is a call to the GetNext() method that is propagated down the tree. Each node that is

المنارة للاستشارات
www.manaraa.com

pulsed issues a pulse to each of its children starting with the left child. In situations where there is no child, the relations in the query node are issued the same GetNext() method call (supported in the relation class).

Once a GetNext() method returns a result (tuple – also a class that abstracts a tuple or row in a table), the ALVExecute class passes the tuple to a method designed to execute each of the query operations. A separate parameterized method is provided for each of the following operations; DoRestrict(), DoProject(), and DoJoin()[31]. These methods operate using one or two tuples as input and return either a null or a tuple. A null return indicates the tuple or tuples do not satisfy the current operation. For example, a DoRestrict() operation accepting a tuple operates using the expression class to evaluate the values in the tuple. If the expression evaluates to false, a null result is returned. If the expression evaluates to true, the same tuple is returned[32].

This process is repeated throughout the tree passing a single tuple up the tree to the root. The resulting tuple from the root is then processed by the external while loop and presented to the client via the existing MySQL client communication protocols. This form of execution is called a pipeline because of the way nodes are traversed, passing a single node through the tree and thus through all of the operations in the query.

6.3.4 SQL_{ALV} Commands

The following are the initial set of SQL commands supported by the ALV system. These commands reflect the minimal operations and parameters necessary to interact

[31] Set operations (intersect, union) and sorting are implemented as specialized forms of join operations.
[32] Actually, all tuples are passed by reference so the item returned is the same pointer.

المنارة للاستشارات

www.manaraa.com

with the version store to create, save, delete, and update data. The purpose of each command is included along with syntax and examples[33].

These commands do not present a departure from the SQL that is supported in MySQL. Furthermore, they do not violate any of the relational algebra or tuple relational calculus rules.

6.3.4.1 Select Command

This command is used to retrieve information from the clustered version store. It can be used to retrieve a range of attribute versions or a single attribute version for one or more records (keys) stored.

```
SELECT [ ATTRIBUTE_LEVEL_VERSION | ALV ]
      * |
      attribname [, attribname] |
      attribname.metafieldname [, attribname.metafieldname] |
      attribname.value [, attribname.value]
FROM tablename
[WHERE [ALV_KEY[.keypartname] = value]  conditions]
```

Notes

- *tablename* = any valid table that is versioned.
- *attribname* = any valid attribute for the attribute version table[34].
- *metafieldname* = any valid metadata field for the attribute version.

[33] These examples use upper case to indicate keywords and italicized to indicate variables.
[34] The name 'attribute version table' refers to the logical representation of the clustered version store. That is, the name of the table structure exposed to the relational database that is stored on disk as the clustered version store. All references in this chapter attribute version store are synonymous with th e phrase clustered version store.

المنارة للاستشارات
www.manaraa.com

- *conditions* = any SQL92 valid combination of *attribname <expression>* | *attribname.metafieldname <expression>*, e.g., Field1.Value = 102, Field1 = 102, Field1.Originated_By = 'cbell', Field1.Source = 90125.
- The default data returned if the metafieldname is omitted is the value of the attribute version.
- The use of ALV_KEY permits the selection of an attribute version for a given record key. Multipart keys are specified by ALV_KEY.*keypartname*.

Examples

```
SELECT ATTRIBUTE_LEVEL_VERSION flow FROM myTable WHERE ALV_KEY = '123';
SELECT ALV flow.class FROM myTable WHERE ALV_KEY = 'Hammond';
SELECT ALV * FROM myTable WHERE ALV_KEY = 'Tower1';
```

Expected response from Server (success)

```
mysql> SELECT ALV * FROM myTable WHERE ALV_KEY = 'Tower1';
+--------+-----------+-----------+--------+----------------+---------------+
| Key    | Attribute | Value     | Source | Confidence     | Reliability   |
+--------+-----------+-----------+--------+----------------+---------------+
| Tower1 | Height    | 100       | 12345  | Low            | Medium        |
| Tower1 | Latitude  | 42.297    | 12345  | Low            | Medium        |
| Tower1 | Longitude | -83.803   | 12345  | Low            | Medium        |
| Tower1 | Height    | 98.7      | 90125  | High           | High          |
| Tower1 | Latitude  | 42.29736  | 90125  | High           | High          |
| Tower1 | Longitude | -83.80310 | 90125  | High           | High          |
| Tower1 | Height    | 99.8      | 90125  | High           | High          |
| Tower1 | Latitude  | 44.567    | 91125  | High           | High          |
| Tower1 | Longitude | -103.21   | 90125  | Low            | High          |
| Tower1 | Height    | 88.7      | 91125  | High           | Low           |
| Tower1 | Latitude  | 47.123    | 90125  | High           | High          |
| Tower1 | Longitude | -85.6012  | 91125  | High           | High          |
+--------+-----------+-----------+--------+----------------+---------------+
12 rows in set (0.02 sec)
```

6.3.4.2 Create Command

This command is used to create the attribute version table (clustered version store) for a given database table. It requires the use of one or more key attributes and at least one attribute to be versioned. Additional metadata can be added as required.

المنارة للاستشارات
www.manaraa.com

```
CREATE [ ATTRIBUTE_LEVEL_VERSION | ALV ] tablename
      KEY attribname datatype,
      [, KEY attribname datatype]
      ATTRIBUTE ( attribname datatype [INDEXED]
      [, metafieldname datatype [INDEXED]]  )
      [, ATTRIBUTE ( attribname datatype
      [, metafieldname datatype [INDEXED]]  )]
```

Notes

- *tablename* = any valid table that is versioned.
- *attribname* = any valid attribute for the attribute version table.
- *metafieldname* = any valid metadata field for the versioned attribute.
- *datatype* = any valid data type supported by host DBMS except large text fields, BLOB, or mapped fields.
- The use of the VERSION_KEY constraint permits the definition of constraints for duplicity of the attribute versions.

Example

```
CREATE ALV TABLE abc KEY xyz INTEGER
ATTRIBUTE (a VARCHAR(10),
           b INTEGER INDEXED,
           c VARCHAR(100)),
ATTRIBUTE (d VARCHAR(10),
           e INTEGER,
           f VARCHAR(100));
```

Expected response from Server (success)

```
Query OK, 0 rows affected (0.00 sec)
Attribute version table created.
```

6.3.4.3 Drop Table Command

This command is used to delete the attribute version table. This operation is irrevocable.

www.manaraa.com

```
DROP [ ATTRIBUTE_LEVEL_VERSION | ALV ] TABLE tablename
```

Notes

- *tablename* = any valid table that is versioned.

Example

```
DROP ALV MyTable;
```

Expected response from Server (success)

```
Query OK, 0 rows affected (0.00 sec)
Attribute version table dropped.
```

6.3.4.4 Drop Database Command

This command is used to delete the attribute version table. This operation is irrevocable.

```
DROP [ ATTRIBUTE_LEVEL_VERSION | ALV ] TABLE tablename
```

Notes

- *tablename* = any valid table that is versioned.

Example

```
DROP ALV MyTable;
```

Expected response from Server (success)

```
Query OK, 0 rows affected (0.00 sec)
Attribute version table dropped.
```

6.3.4.5 Insert Command

This command is used to add data to a attribute version table. It accepts a single attribute version and its metadata and stores the data in the table.

```
INSERT [ ATTRIBUTE_LEVEL_VERSION | ALV ] INTO tablename (
      attribname [, metafieldname])
VALUES ( attribvalue [, metafieldvalue] )
```

Notes

المنارة للاستشارات
www.manaraa.com

- *tablename* = any valid table that is versioned.
- *attribname* = any valid attribute for the attribute version table.
- *metafieldname* = any valid metadata field for the versioned attribute.
- *attribvalue* = any valid value for the attribute named.
- *metafieldvalue* = any valid value for the metafield named.

Example

```
INSERT ALV INTO abc (a,b,c) VALUES ('a','b','c');
```

Expected response from Server (success)

```
Query OK, 0 rows affected (0.00 sec)
Attribute version inserted.
```

6.3.4.6 Delete Command

This command deletes one or more attribute values from a attribute version table.

```
DELETE [ ATTRIBUTE_LEVEL_VERSION | ALV ] FROM tablename
[WHERE [ALV_KEY[.keypartname] = value]  conditions]
```

Notes

- *tablename* = any valid table that is versioned.
- *conditions* = any SQL92 valid combination of *attribname* *<expression>* | *attribname***.***metafieldname* *<expression>*, e.g., Field1.Value = 102, Field1 = 102, Field1.Originated_By = 'cbell', key = 90125.
- *keypartname* = any portion of the key should the key be formed by multiple attributes.
- *value* = literal key value.

Example

```
DELETE ALV flow FROM myTable where ALV_KEY = 90125;
```

المنارة للاستشارات
www.manaraa.com

Expected response from Server (success)

```
Query OK, 0 rows affected (0.00 sec)
Attribute version deleted.
```

6.3.4.7 Update Command

This command is used to alter the values and/or metadata field values for one or more attribute versions.

```
UPDATE [ ATTRIBUTE_LEVEL_VERSION | ALV ] tablename FOR attribname SET
      metafieldname = metafieldvalue
      [, metafieldname = metafieldvalue]
[WHERE [ALV_KEY[.keypartname] = value]  conditions]
```

Notes

- *tablename* = any valid table that is versioned.
- *metafieldname* = any valid metadata field for the versioned attribute.
- *attribname* = any valid attribute for the attribute version table.
- *attribvalue* = any valid value for the attribute named.
- *metafieldvalue* = any valid value for the metafield named.
- *keypartname* = any portion of the key should the key be formed by multiple attributes.
- *conditions* = any SQL92 valid combination of *attribname <expression>* | *attribname***.***metafieldname <expression>*, e.g., Field1.Value = 102, Field1 = 102, Field1.Originated_By = ‘cbell’, key = 90125.

Example

```
UPDATE ALV abc FOR flow SET b = 1, c = 3 WHERE ALV_KEY = 90125;
```

Expected response from Server (success)

```
Query OK, 0 rows affected (0.00 sec)
```

المنارة للاستشارات
www.manaraa.com

```
Attribute version updated.
```

6.3.4.8 Show Tables Command

This command is used to list the tables that have attribute tables created.

```
SHOW [ ATTRIBUTE_LEVEL_VERSION | ALV ] TABLES
```

Example

```
SHOW ALV TABLES;
```

Expected response from Server (success)

```
+---------------------+
| ALV_Tables_in_test |
+---------------------+
| abc                 |
| def                 |
+---------------------+
2 rows in set (0.00 sec)
```

6.3.4.9 Show Databases Command

This command is used to list the databases that have attribute tables created.

```
SHOW [ ATTRIBUTE_LEVEL_VERSION | ALV ] DATABASES
```

Example

```
SHOW ALV DATABASES;
```

Expected response from Server (success)

```
+----------------+
| ALV_Databases |
+----------------+
| test           |
+----------------+
1 row in set (0.00 sec)
```

6.3.4.10 Explain Table Command

This command is used to show the details of an attribute version table. It resembles the explain and describe commands as implemented in MySQL.

المنارة للاستشارات

www.manaraa.com

```
EXPLAIN [ ATTRIBUTE_LEVEL_VERSION | ALV ] tablename
```

Example

```
EXPLAIN ALV MyTable;
```

Expected response from Server (success)

```
mysql> EXPLAIN ALV towers;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| ALV_KEY        | STRING(100) | NO   | PRI |         |       |
| ATTR           | STRING(100) | NO   |     |         |       |
| VALUE          | STRING(100) | YES  |     |         |       |
| CLASSIFICATION | STRING(100) | YES  |     |         |       |
| SOURCE         | STRING(10)  | YES  |     |         |       |
| CONFIDENCE     | STRING(100) | YES  |     |         |       |
| RELIABILITY    | STRING(100) | YES  |     |         |       |
| DATE           | STRING(100) | YES  |     |         |       |
| CUTOFFDATE     | STRING(30)  | YES  |     |         |       |
| REMARKS        | STRING(30)  | YES  |     |         |       |
| USEDIN         | LONG(7)     | YES  |     |         |       |
+----------------+-------------+------+-----+---------+-------+
11 rows in set (0.00 sec)
```

6.3.4.11 Explain Query Command

This command is used to see the query plan that the ALV query optimizer has generated but does not execute the query. It prints out the results of the query tree after heuristic and cost optimization have been performed. It is used to diagnose long running queries and to inform the database professional how the query would be executed.

```
EXPLAIN validselect_alv_command
```

Notes

- validselect_alv_command = any valid select command used to query an attribute version table.

Example

```
EXPLAIN SELECT ALV height.* FROM MyTable WHERE ALV_KEY = ‘tower1’;
```

www.manaraa.com

Expected response from Server (success)

```
mysql> EXPLAIN SELECT ALV height.* FROM towers WHERE alv_key =
'tower1';

+---------------------------+
| Execution Path            |
+---------------------------+
|      telecom.towers       |
|           |               |
|           V               |
|     ------------------- |
|     |    RESTRICT     | |
|     ------------------- |
|     | Access Method:  | |
|     |    iterator     | |
|     ------------------- |
|           |               |
|           V               |
|     ------------------- |
|     |     PROJECT     | |
|     ------------------- |
|     | Access Method:  | |
|     |    iterator     | |
|     ------------------- |
|              |            |
|              V            |
|         Result Set        |
+---------------------------+
25 rows in set (0.03 sec)
```

6.3.4.12 Backup Command

This command is used to make a binary copy of the ALV tables in the database specified. It stores them in the path provided.

```
BACKUP TABLE tbl_name [, tbl_name] TO '/path/to/backup/directory'
```

Notes

- *tbl_name* = any valid table that is versioned.
- The target folder must not contain previous backup files.

Example

www.manaraa.com

```
BACKUP ALV TABLE table1 TO 'c:\\backup';
```

Expected response from Server (success)

```
mysql> BACKUP ALV TABLE table1 TO 'c:\\backup';
+-------+------------+----------+----------+
| Table | Op         | Msg_type | Msg_text |
+-------+------------+----------+----------+
| error | ALV backup | status   | OK       |
+-------+------------+----------+----------+
1 row in set (0.03 sec)
```

6.3.4.13 Restore Command

This command is used to restore a binary copy of the ALV tables in the database or list of tables specified.

```
RESTORE TABLE tbl_name [, tbl_name] FROM '/path/to/backup/directory'
```

Example

```
RESTORE ALV TABLE table1 FROM 'c:\\backup';
```

Expected response from Server (success)

```
mysql> RESTORE ALV TABLE table1 FROM 'c:\\backup';
+----------------+-------------+----------+----------+
| Table          | Op          | Msg_type | Msg_text |
+----------------+-------------+----------+----------+
| testdb1.table1 | ALV restore | status   | OK       |
+----------------+-------------+----------+----------+
1 row in set, 1 warning (0.03 sec)
```

6.3.4.14 Version() Function

This command is used to display the version of the MySQL engine currently running. This command will verify the system is running the ALV enhanced version of the mysqld executable.

```
SELECT VERSION()
```

Notes

المنارة للاستشارات

www.manaraa.com

- Although this command was not altered in the MySQL source code, it is included for completeness.

Example

```
SELECT VERSION();
```

Expected response from Server (success)

```
+-------------------------+
| version()               |
+-------------------------+
| 5.0.10a-beta-nt-ALV 1.0 |
+-------------------------+
1 row in set (0.21 sec)
```

6.4 Analysis

This section describes the analysis performed while implementing and experimenting with the ALV query optimizer and execution components. All of the experiments were run on a 3.0Ghz AMD processor-based system running Windows XP Professional. The disk subsystem used was a hardware raid system incorporating two S-ATA physical devices in a mirrored arrangement. The experiments were repeated using a conventional IDE-133 device with little or no variation in the measurements[35].

The following experiments were conducted using datasets created from publicly available databases. Appendix A describes these data sets, their origin, and translations and reformations for use in this work. Each of the following experiments was conducted using the same set of sample queries.

6.4.1 Transformation

[35] This is expected because the differences in the physical devices and their access protocols are very similar. Although throughput on the S -ATA devices theoretically could be faster, the addition of the raid subsystem nullifies any advantage over IDE -133 devices.

www.manaraa.com

One of the lesser costs in a database system is the transformation of a query from the SQL statement to the internal representation (data structure). MySQL uses a fixed data structure consisting of a class with lists and lists of lists as its members. The ALV query tree represents a decided departure from the MySQL internal representation. In order to minimize the changes to the MySQL server, it was decided to use the MySQL parser and form the MySQL internal representation, then transform that into an ALV query tree. Although this does add some processing time to the total time of a query, it does not add a significant amount of time since all of the transformations are done on structures that are in memory. Table 6-4 below gives examples of the time (in seconds) certain queries required for transformation from the MySQL internal representation to an ALV query tree.

Query	MySQL	ALV
SELECT * FROM authors_alv;	0.0000492241	0.0031957970
SELECT auLName, auFName FROM authors WHERE auState IN ('UT', 'CA');	0.0000820775	0.0001537904
SELECT * FROM authors JOIN bookauthor ON authors.auid = bookauthor.auid;	0.0000737524	0.0008519519

Table 6-4: Query Transformation Data

The first column contains the SQL statement for the sample query. The second column reports the data from a set of executions, reporting the time spent transforming a query into the MySQL internal representation. The third column reports the data from the same set of executions reporting the time spent transforming the MySQL data structure to

المنارة للاستشارات
www.manaraa.com

an ALV query tree[36]. Each value for this experiment is the mean value of twenty trials executed at a variety of times to ensure that the effect of current system load was minimized.

The results of the experiment show that the ALV query transformation from the MySQL internal structure requires two orders of magnitude greater time to complete. However, it must be noted that an execution time of 0.003 is still very efficient and does not add significant delays to overall query processing. However, this time could be greatly reduced if the ALV query tree were to be constructed in the MySQL parser.

6.4.2 Optimization

The greatest cost in a database system is query optimization. MySQL uses a select-project-join optimizer that implements a flat optimization strategy devoid of heuristic rules, but does implement some cost-based analysis for access methods and joins. Table 6-5 below gives examples of the time (in seconds) of the optimization of certain queries.

Query	MySQL	ALV
SELECT * FROM authors_alv;	0.0000324064	0.0000052940
SELECT auLName, auFName FROM authors WHERE auState IN ('UT', 'CA');	0.0000433994	0.0000227403
SELECT * FROM authors JOIN bookauthor ON authors.auid = bookauthor.auid;	0.0000733334	0.0000048610

Table 6-5: Query Optimization Data

[36] This time is in addition to the time in the MySQL column.

المنارة للاستشارات
www.manaraa.com

The first column contains the SQL statement for the sample query, the second column reports the data from a set of executions reporting time required by the MySQL query optimizer, and the third column reports the data from the same set of executions reporting the time required by the ALV query optimizer. Again, each value for this experiment is the mean value of twenty trials executed at a variety of times to ensure that the effect of current system load was minimized.

The results show that the ALV query optimizer performs approximately two orders of magnitude faster than that of the MySQL optimizer. This shows that the ALV query optimizer performs similar to that of a commercially available database system and does not require any significant time delays to optimize queries in a versioning system.

6.4.3 Execution

The next highest cost to that of optimization is query execution. The MySQL execution engine is difficult to describe. In essence, it is implemented as a series of compiled methods that are optimized for executing the individual parts of a select-project-join query. The ALV execution is based on the ALV query tree and provides execution without the need of additional overhead or methods. Table 6-6 below gives examples of the time (in seconds) of the execution of certain queries. The data used in this experiment was implemented in both the native MySQL data store and the ALV clustered version store.

Query	MySQL	ALV
SELECT * FROM authors_alv;	0.0023644350	0.0007561995 (0.000482031)

المنارة للاستشارات
www.manaraa.com

SELECT auLName, auFName FROM authors WHERE auState IN ('UT', 'CA');	0.0003637754	0.0031957970 (0.000482031)
SELECT * FROM authors JOIN bookauthor ON authors.auid = bookauthor.auid;	0.0032613920	0.0016447480

Table 6-6: Query Execution Data

The first column contains the SQL statement for the sample query, the second column reports the data from a set of executions reporting time required by the MySQL query optimizer, and the third column reports the data from the same set of executions reporting the time required by the ALV query optimizer[37]. Each value for this experiment is the mean value of twenty trials executed at a variety of times to ensure that the effect of current system load was minimized.

The results show that the ALV query execution as compared to the MySQL execution varies depending on the type of query. This is expected as the ALV execution engine is very different from the MySQL execution engine. In this experiment, the ALV execution engine outperformed the MySQL execution engine on the trivial query and the simple join. However, the ALV execution engine performed one order of magnitude slower than the MySQL execution engine for the second test case[38]. Note that the timings presented are very small and are of little significance for the overall query execution. Thus, the ALV query execution is also on par with that of a commercially available database system and does not require any significant time delays to optimize queries in a versioning system.

[37] The implementation of the MySQL JOIN operation to/from non ALV data stores requires translating the MySQL data store to an ALV relation object. This time had a mean value of 0.000482031 for each query executed and was not included in the ALV execution times.
[38] The cause for this has since been identified in the expression evaluation. Plans are underway to correct the anomaly.

المنارة للاستشارات
www.manaraa.com

6.5 Conclusion

The ALV query optimizer, query tree, and query execution engine demonstrate the potential of implementing such technologies in a production relational database system. The fact that these technologies are based on academic views of implementation proves that academic rigor can be translated directly to industry without compromising the “science” behind the details. As the experiments show, the technologies presented represent effective and efficient technologies for use in a versioning system.

6.6 Future Work

The cost optimization step of the ALV query optimizer is very effective at identifying the benefits of using indexes for locating data in or iterating through a relation. However, the query optimizer could be designed to optimize or balance joins better than it does. For most joins, the existing strategy works well and will continue to generate near optimal executions. However, for complex joins in an environment that has complex indexes and multi-level indexes, the cost optimization step should be modified to balance joins better. Once this has been accomplished, the ALV query optimizer will be complete and robust enough to handle any environment and use.

The ALVExecute class can be enhanced and query execution reduced by using a multithreaded execution variant of the pipeline. That is, each operation in the tree could be executed as a separate thread providing appropriate synchronization using mechanisms such as queues for preemptive and wait conditions. However, care must be taken to avoid deadlock and race conditions.

المنارة للاستشارات
www.manaraa.com

One of the most important features of MySQL is the query cache [MySQ05]. The query cache is a mechanism that stores the results of executed queries for reuse. This includes storing not only the parsed and optimized query, but also the query results. This gives MySQL the ability to respond quickly to repetitive queries and exceed the performance of database systems that do not cache queries. The ALV query tree can be used to implement a query cache. Work will need to be done to associate a result set (relation class) with a query tree. Fortunately, the implementation can easily be adapted for serialization and organization of a query cache. However, like the query tree and query execution, the ALV query cache will be functionally equivalent but with a distinctly different implementation.

The SQL_{ALV} extensions developed to support versioning in a relational database system are not complete. A complete set of SQL commands would include the alter commands as well as enhancements to the select command for greater flexibility in performing complex queries. Additional development is necessary to complete the SQL_{ALV} commands.

المنارة للاستشارات
www.manaraa.com

Chapter Seven - Data Mining for Version Analysis

Abstract

Given that there now exists a relational database versioning system, called Attribute-Level Versioning (ALV), the attribute version data stored in the system has meaning when combined with the versioned data and inferences can be made by joining the versioned data with the attribute version data, selecting permutations of the multiple values. Also supported in this system is the concept of storing metadata with each attribute version. However, little has been presented as to the benefits of that metadata and how it could be used to gain additional knowledge of the versioned data.

This chapter will show an application of data mining for asking questions of the attribute version metadata. The analysis and results of experiments to demonstrate how version information can be mined will be presented.

7.1 Introduction

Attribute version data as described in this work contains additional information about each attribute version in the form of metadata. The metadata is a means by which the analyst can store any additional information concerning the attribute version. Examples include storing information as to the origin, the confidence, reliability of the data, or sensitivity, or even temporal information for the data. This metadata represents

المنارة للاستشارات

www.manaraa.com

unique descriptions of the data that can be used to gain additional knowledge about the data. That is, it is feasible to use data mining techniques to gain additional information about the attribute versions and apply that to the versioned data.

The following sections present the current research on data mining and machine learning, the application of data mining algorithms to analyze attribute version metadata, an analysis of the results, and a conclusion as to their success in meeting the goals defined above. This chapter concludes with a section outlining future work opportunities to improve the use of data mining algorithms with attribute version data.

7.2 Background

The process of finding useful patterns in datasets has been identified with many different labels, including data mining, knowledge extraction, information discovery, information harvesting, data archaeology, and data pattern processing [Fayy96]. Much of the research to meet these goals represents sub-disciplines such as artificial intelligence, machine learning, database theory and statistics. Data Mining (DM) is a phrase coined by statisticians, database professionals, and information technology experts to describe the quantification and qualification of data in databases.

Data mining has come to mean many things among researchers. Some [Witt05] take the position that data mining is the application of machine learning algorithms to discover knowledge within data. Others [Date04] consider data mining to be “exploratory data analysis” where statistical analysis is used to discover patterns in large datasets. The first perspective can be considered a scientific position and the second a business

www.manaraa.com

strategy-oriented position[1]. That is not to say that there isn't science in the second approach. Rather, the motives for conducting research are largely guided by the needs of industry (business). Some simply consider data mining an artistic application of many disciplines [Thur00a].

While the first two positions are very similar, the most interesting difference lies with how large a dataset must be to produce valid results. While there is no definitive answer to this question, data mining researchers from the machine learning/scientific perspective generally consider the size of the datasets less important than the purity of the data[2] [Witt05]. Conversely, researchers from the business strategy-oriented perspective consider files of greater sizes – the accumulated data for a given period or study area. Some only consider large amounts of data to be essential for evaluating conditions concerning time. Lastly, those that consider data mining an art form do so more from the point of view of the creative application of algorithms to form postulates for suppositions[3] [Thur00a]. These suppositions are considered the problem, pattern expected, speculation to be verified, or hypothesis that is being proven.

A brief mention of the coverage of data mining texts is useful to place this work in context. Some texts about data mining are written from a purely tool-centric view and present tool-centric concepts and implementations of data mining [Seid01]. While informative, these texts often limit exposure to scientific theories and practice and often

---

[1] This distinction permits o ne to draw a parallel among the many texts and tomes written about data mining with fireworks – the data mining texts geared toward business and marketing strategies are the sparklers while texts involving algorithm development and machine learning are the really big bangs. Care must be taken with either, but the later can leave you burned if used improperly.
[2] Purity is the result of anomalies and errors being removed.
[3] Correct formulation of the supposition is not necessary, rather it is usually consider ed a guiding principle or goal of the mining process [Witt05].

www.manaraa.com

do not expose the reader to machine learning or statistical concepts. Other texts are so aligned with business-oriented applications of data mining that the concepts and practices of data mining are diluted and either contradict the current theory and practice or avoid it altogether [Grot98]. Care must be taken to choose a data mining reference text that serves the needs of the reader. Should the reader desire to learn the theories and practices of data mining, she should avoid texts that offer business-oriented or tool-centric views. Rather, the reader should look for texts that include topics such as machine learning, statistics, and in-depth discussions of data mining algorithms[4].

The following sections present historical context and current philosophies on data mining. These sections answer the questions, "Why do we need data mining?" "What is data mining anyway?" and "Isn't it just another fancy marketing strategy designed to sell more donuts and beer?"[5] For example, researchers use the marketing application to demonstrate how data mining can go horribly wrong [Witt05]. The most popular example is where a supermarket uses data mining to (incorrectly) predict the sales of products rather than using data mining to identify patterns of past purchases. Even when there are clear patterns identified, it has been shown that the results can be misinterpreted.

7.2.1 Knowledge Discovery in Databases

The phrase, "knowledge discovery in databases" (KDD), was coined from the artificial intelligence (AI) and machine learning (ML) communities. KDD is concerned

[4] Failure to heed this warning may lead to misapplication of the tools or worse, incorrect interpretation of the results.
[5] A common misconception concerning data mining is that it is only usef ul in predicting market trends.

المنارة للاستشارات
www.manaraa.com

with the study of how knowledge can be gained from data. In fact, the term KDD was originally the term used by researchers in the AI and ML communities to refer to the application of machine learning algorithms for knowledge exploitation. Unfortunately, the term KDD has come to mean something somewhat less than that and the term “data mining” has taken precedence. In fact, the two terms and often interchanged or used together [Piat00]. KDD has also been defined as “the process of identifying valid, novel, potentially useful, and ultimately understandable structure in data,” [Rals03].

Interestingly, some researchers [Fayy96] consider KDD a much broader spectrum of tools, algorithms and techniques that includes data mining as one of its core areas. In fact, data mining is considered a subprocess or step in a much larger process that is KDD. Figure 7-1 below illustrates this distinction.

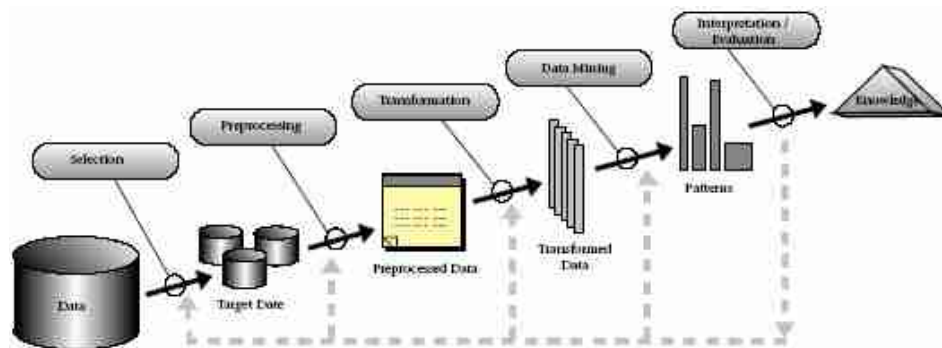


Figure 7-1: The Knowledge Discovery Process

The KDD process begins with developing an understanding of the data and relevant prior knowledge and suppositions formed from examining the data in its native environment (queries). Next, the appropriate data is identified and extracted from the pool of available repositories. The data is then cleaned by removing invalid data and preprocessed for ingestion into the desired model or modeling environment. Then the

www.manaraa.com

data is restricted and projected to form datasets that describe (contain data for the solution of) the problems or knowledge being sought. The data is then processed using a data mining tool to discover patterns and relationships within the data. Lastly, the results are interpreted and the answers formulated [Fayy96].

It should be noted at this point that the general outline of these steps follows that of what many of the texts refer to as the data mining process. In fact, this work follows a similar process. Regardless of what the process described in this chapter is called, the base premise is still valid: knowledge can be gained by applying machine learning algorithms to data to discover new patterns and relationships among the data.

7.2.2 Machine Learning

A brief mention of machine learning is necessary to understand the correlation with data mining. Machine learning is defined as "the study of methods for improving computing systems through observation and analysis of their behavior rather than by direct programming," [Rals03]. That is, machine learning is the identification of significant observations and the application of those observances to form new knowledge – learning from machines.

What is learning? Learning can be defined as any of the following:

- Gaining knowledge through study or experience
- Gaining knowledge through instruction
- Observing and retaining

المنارة للاستشارات
www.manaraa.com

- Remembering and applying
- Adapting to past observations

Most of these do not have any intuitive application to computers and in general seem very human. However, it should be noted that computers are intellectual aides and not self aware beings capable of learning[6]. That is, computers – machines – are in themselves a mechanism by which humans learn. Thus, the application of computer algorithms can indeed aid in the learning process. In essence, this represents learning via the acquisition and application of knowledge [Witt05].

There are many applications of machine learning, most notably in robotics where robots can adapt to their environment by storing observations about their environment and applying those observations to predict new conditions. Another application is prediction of network vulnerabilities based on current usage. Similarly, machine learning algorithms are used to predict credit card fraud. But the most recent application of machine learning is in finding interesting patterns in data, sometimes called data mining [Rals03].

Machine learning tasks can be grouped into three categories: supervised, unsupervised, and reinforced. In supervised learning, the algorithms used are predictive and produce discrete outcomes. The goal of supervised learning is to form a function based on the observations that produces a specific output for a given input. Examples of supervised learning algorithms include decisions trees (CART and C4.5) and naïve Bayes

[6] At least, not yet. With apologies to Asimovians everywhere.

المنارة للاستشارات
www.manaraa.com

learning [Rals03]. In unsupervised learning, the algorithms are used to find structure in the points of data. The goal of unsupervised learning is to discover patterns of data using a probabilistic map of the observances. Examples of unsupervised learning algorithms include EM, SimpleKMeans, and clustering schemes [Rals03]. In reinforced learning, algorithms are created to define a next state based on the observation of a sequence of events. The goal of reinforced learning is to be able to identify patterns in sequences and adapt to form the best response. Examples of reinforced learning include strategy games and robotic discovery (*e.g.*, bomb capture and disposal, disaster victim recovery, and online gaming) [Rals03].

7.2.3 Data Mining

Data collection has become an integral part of almost every organization. As time passes, more and more data is collected, making the interpretation of the data harder to accomplish. It is easy to identify patterns in small or simple data stores, but recognizing patterns becomes increasingly difficult as the complexity and quantity of the data increases [Paul02]. What we need are sophisticated methods of discovering knowledge from the data that can be used to satisfy or contradict suppositions formed from the nature (original meaning) of the data. That is, it is easy to quantify how many of types of entities or their properties exist. It is much harder to identify the trends of the values of the properties of entities.

Data mining is the process of fitting models to, or determining patterns from observing and processing the data [Fayy96]. Some texts define data mining as a

المنارة للاستشارات
www.manaraa.com

companion to or closely related to information retrieval – the study of discovering knowledge in information systems [Tuck04]. Other texts define data mining as an application of machine learning techniques [Witt05]. Others even define data mining as its own genre supported by or incorporating technologies such as statistics, machine learning, database theory, knowledge discover in databases, pattern recognition, artificial intelligence, and information retrieval [Rals03].

Vendors of most major database management systems and data analysis products have included data mining tools that manipulate the data in relational databases. Mining unstructured repositories such as text, imagery, or video has received a lot of attention and remains an area for further research. For instance, applying data mining to the vast quantities of data on the world wide web to extract meaningful information is no small feat [Thur00].

Many of the larger database system providers (Microsoft [Micr03], Oracle [Orac05]) provide tools for performing data mining. Some companies have built protocols and access methods such as Microsoft's extensions to the OLE DB communication protocol [Netz01] that enable database professionals to connect their database system to data mining tools more easily. With the relatively recent rise of popularity of data mining, vendors are finding new markets to explore the application of data mining.

As we shall discover in the next few sections, data mining can be a very powerful tool for extracting useful information from patterns in the data. However, it can also extract erroneous and useless information if it is applied or the results are interpreted

المنارة للاستشارات
www.manaraa.com

incorrectly. A key to avoiding these pitfalls is a basic understanding of what data mining is and what things to consider in planning a data mining project [Thur00].

7.2.3.1 Applications of Data Mining

Data mining systems collect, store, and organize data for use in modeling the data and identifying patterns in the data. Application areas include: medicine, finance, intelligence, law enforcement, defense, logistics, education, and process control. However, the most popular application is in marketing and business strategies. Example uses of data mining include [Thur00]:

- Credit agencies can use data mining to grant loans based on observations of people with similar buying patterns, income, and credit
- Marketers can use data mining to organize merchandise based on buying patterns and information about associations between products
- Pharmaceutical companies can use data mining to analyze prescriptions in order to send promotional material to targeted customers
- Law enforcement agencies can use data mining to review spending patterns and travel data to detect abnormal behavior of suspects
- Physicians can use data mining to analyze X-ray images to detect abnormal patterns

المنارة للاستشارات
www.manaraa.com

- Commercial transportation companies can use data mining to discover travel patterns and trends to maximize passenger/cargo and thus reduce customer costs while maximizing profit
- World wide web search engines can use data mining to make search engines more effective by identifying trends in search parameters through the application of clustering

Recently, work has been done to apply data mining to temporal data. The use of temporal databases has fueled a need to adopt data mining to temporal analysis. One way this is being accomplished is by separating the temporal data from the original data and modifying existing algorithms to identify sequences or patterns of time values in the data. The results of experiments show significant justification for additional work in this area [Rodd02].

7.2.3.2 Data Mining Algorithms

All data mining algorithms exhibit characteristics that can be grouped into three parts:

1. Modeling function – the purpose of the algorithm.
2. Preference – the criteria that corresponds to the selected algorithm[7].
3. Search or iteration routines to process the data.

[7] Not all criteria fit all algorithms.

المنارة للاستشارات
www.manaraa.com

Furthermore, data mining algorithms can be categorized as being either predictive or descriptive in nature. That is, they either make predictions about the data based on the known results from data that contains known descriptive elements, sometimes called training data[8], or the algorithms identify patterns or relationships in the data [Dunh03].

Data mining algorithms can be described by their functions or roles [Dunh03]. These tasks include; classification, regression, time series analysis, prediction, clustering, summarization, association rules, and sequence discovery. Table 7-1 below describes each of these functions and identifies their category.

Function	Description	Category
Classification	Maps data into groups or classes. Sometimes called supervised learning because the classes are usually defined in advance.	Predictive
Regression	Maps a data item (attribute) to a specific value. These algorithms learn the function used to predict the value using a variety of statistical techniques.	Predictive
Time Series Analysis	Examines the values of attributes over time. These algorithms identify similarities over time, classification of behavior over time, and prediction of future values based on historical record.	Predictive
Prediction	A type of classification that determines values or states based on historical analysis.	Predictive
Clustering	Similar to classification where data is grouped into classes, but in this case the classes are discovered from the data. Clustering is sometimes referred to as unsupervised learning because the classes are discovered rather than supplied *a priori*.	Descriptive

[8] The notion of "training" comes from the fact that the training data provides the patterns and/or relationships a priori for the algorithm to use against other data (called test data).

www.manaraa.com

Function	Description	Category
Summarization	Maps data into subsets with associated descriptions. Also called characterization or generalization. These algorithms describe data.	Descriptive
Association Rules	These algorithms are used to discover relationships among the data. Rules are automatically generated from the data to describe the relationships. These algorithms are used to establish link analysis (sometimes called affinity or association analysis).	Descriptive
Sequence Discovery	Used to discover sequential patterns in the data. Usually associated with temporal elements such as time series or time stamps.	Descriptive

Table 7-1: Data Mining Functions

Implementations of each of these functions are what define the list of data mining algorithms. Although there are many different algorithms, most can be grouped by the functions listed above [Dunh03].

One area of data mining algorithm research that expands the realm of prediction and description is the application of uncertain reasoning. Uncertain reasoning is the application of expertise to form conclusions without specifics or complete rules. Uncertain reasoning is a topic of study in artificial intelligence. The two primary algorithms classes studied using uncertain reasoning in data mining are Bayesian networks and artificial intelligence neural networks. Bayesian networks explore unknowns using probability calculations while neural networks explore unknowns using a symbolic graph structure for learning patterns to deal with uncertainty. For more information on this topic, see Chen's text [Chen01].

المنارة للاستشارات

www.manaraa.com

7.2.3.3 The Data Mining Process

While some texts disagree with others [Dunh03, Paul02] on what a typical data mining process is, there are some common practices among the literature. Figure 7-2 below presents a pictorial representation of the data mining process which can be described as follows. The data sources are combined to form a single data set, impurities and errors are removed from the data, the data is mined, the results analyzed, and finally the results are published.

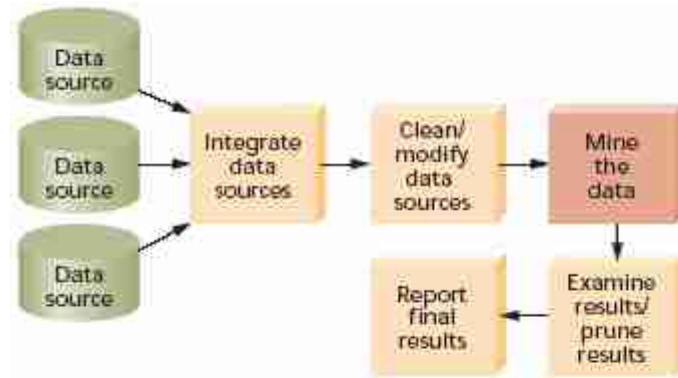


Figure 7-2: The Data Mining Process

These steps can be grouped to form a typical data mining process which includes the following steps:

1. **Problem Definition** – what problem are you trying to solve? What dataset contains the information you want to use to formulate an answer? What are your suppositions about the data?
2. **Data Preparation** – transforming the data into a form usable by the data mining workbench of choice. This include performing queries to define the dataset you

المنارة للاستشارات
www.manaraa.com

want to mine along with removing attributes or rows for a more concise dataset definition.

3. **Model Experimentation** – once the data is prepared and ready to be mined, experiments must be undertaken to determine the correct algorithm necessary to achieve the goals specified in step 1.
4. **Model Validation** – due to the interpretive nature of data mining results, it is necessary to conduct a validation of the model results. That is, examine the results and test the conclusions against known cases to show that the answers produced are the correct ones.

Each application of data mining and even the data mining tools themselves may introduce a specific set of steps. However all contain those listed above. The work presented below follows these four steps to answer suppositions of the version metadata.

Problem definition requires one to understand the data that will be mined. This includes analyzing the needs of the organization, the use of the data, and forming suppositions (questions) about the data. This could take the form of finding a specific pattern within the data, supporting or rejecting a hypothesis formed about the data, predicting missing values based on patterns of data that contain values, and discovering relationships within the data.

Once the supposition is formed, the next decision lies in choosing the appropriate data mining environment or tools. The choice of tool or environment may require specific

المنارة للاستشارات

www.manaraa.com

modification or reshaping of the data for use in the tools. These decisions form the goals of the data mining process and shape the decisions made later in the process.

Data preparation is a process by which the data is transformed into a format that the chosen data mining tools can process. Some tools based in database management systems or those designed to operate with database systems require the data to be formatted into a logical representation known as a cube [Micr00]. An example of a cube is shown in figure 7-3 below.

A cube is a structure that contains a hierarchy of levels. A level is an element of a dimension hierarchy which describes data from the most summarized to the most detailed units the dimension. Levels are arranged in hierarchies which define the relative positions of members. Dimension levels are powerful tools as they can be used to "drill down" to granular levels or "drill up" to summarized levels of data in a cube. For more information on cubes, see Microsoft's SQL Server 2000 documentation and associated help files [Micr00] or Han's text in [Han01].

المنارة للاستشارات
www.manaraa.com

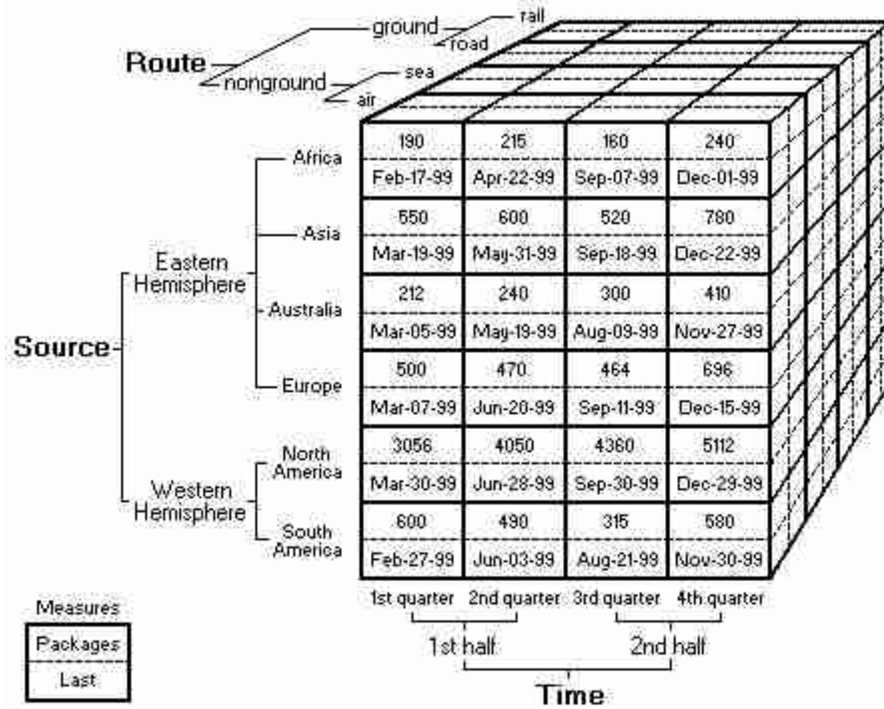


Figure 7-3: A Data Mining Cube

While specialized data structures like the cube described above assist in the data mining process by providing aggregate mechanisms, most data mining solutions use a more basic data structure such as a flat rows and columns format (not unlike a table in a database). For example, the Waikato Environment for Knowledge Analysis (Weka)[9] data mining tools require the data be formatted in either attribute-relation file format (ARFF), comma separated values (CSV), or via a JDBC database connection [Witt05].

Another storage technique that is often associated with data mining is called a data warehouse. A data warehouse is an implementation of a transactional database that is used to store vast amounts of information over a distributed system, sometimes refered to as online analytical processing (OLAP) versus an online transaction processing (OLTP).

[9] Weka is pronounced to rhyme with “Mecca” and is named after a flightless bird with an inquisitive nature found only on the islands of New Zealand [Witt05].

www.manaraa.com

Unlike OLTP systems, OLAP systems are not typically used interactively to modify data. Rather, OLAP systems are used to archive data over time – hence the "warehouse" mantle. OLAP systems are often used as fertile ground for constructing data for data mining [Rals03].

Once the data is manipulated into the proper format, a data cleaning process is performed to remove errors or ambiguities from the data. Unfortunately, this process is often the most time consuming and the most error prone. Experience with the available algorithms and the data modeling environment are often required in order to get this step correct [Witt05]. This is one of the reasons data mining is considered an art form.

Now that the data is formed in the proper format and all errors are removed, the next step is the selection of a data mining algorithm. This step is considered to be the most difficult of all [Witt05] because it requires knowledge of all of the algorithms available in the environment or tool of choice. One must understand the application of each of the algorithms in order to choose the most appropriate. Often times it is necessary and even beneficial to choose several algorithms, record the results of each, and compare them to choose the most correct. However, it should be noted that the nuances of the algorithms are such that variants of each algorithm can generate a slightly different model. That is, the results may be affected with each change of a variable or parameter.

The analysis of the results of the data mining process concludes the effort and it is here that once again that experience and art are visible. Interpreting the results of data mining algorithms is second in difficulty only to selecting an algorithm. Each algorithm presents results in a different way and requires careful analysis to interpret them. This

المنارة للاستشارات
www.manaraa.com

area is perhaps the most prone to scrutiny and the most likely to cause false conclusions or misapplication of the results [Witt05].

7.2.3.4 Analyzing the Results of Data Mining

Now that the data has been culled, shaped, and acted on by the data mining algorithms, we are faced with interpreting the results of the algorithms. Indeed, most data mining practitioners do not consider the data mining process complete until the results and even the performance of the algorithms used are validated [Witt05].

There are two main strategies used in analyzing the results of data mining. The first involves using appropriate viewers to display the results, and the second is to examine the results of the algorithm – the metadata generated during algorithm execution – to determine if the algorithm was the correct algorithm to use and if it produced the desired results [Witt05].

In order to achieve a high degree of confidence in the results, one must visualize the results using the appropriate view of the algorithm's output. Algorithms can be grouped by the types of outcomes they produce. The following describes the types of outcomes [Thur00]:

- **Classification Algorithms** – group or classify data based on a predefined attribute. For example, "people who live in Richmond and own apartments costing more than $100, 000" is a classification of data on residents.

المنارة للاستشارات
www.manaraa.com

- **Regression Algorithms** – make predictions of missing values using examples of existing data. Examples include "records that include value X for attribute A and value Y for attribute C have Z as the value for Attribute B, thus the missing value for a record with no value for Attribute B given {A=X, C=Y}, the value of B is Z."
- **Time Series Analysis** – make predictions of trends over time. Examples include "given the trend of winning seasons in which 6 or more games were won on the road, the Washington Redskins will play in the 2006 superbowl[10]."
- **Prediction Algorithms** – are used to forecast trends such as "in 2010, homes in Northern Virginia will cost an average of $3,000,000[11]."
- **Clustering Algorithms** – classify or group data based on a previously undefined attribute such as, "all employees in cluster A make less than $24,000, those in cluster B make between $24,000 and $50,000, and those in cluster C make more than $50,000."
- **Summarization Algorithms** – examine trends for clues to deduce another characteristic. For example, you might analyze spending patterns and income classes to predict how many children a married couple is likely to have.
- **Association Algorithms** – make correlations among the data, deducing rules that define relationships. Examples include "disposable diapers and beer are purchased together" or "John and Charles have similar travel habits."

[10] It happened in the past, why no t this year?
[11] Don't belittle this example. Have you checked out the cost of housing in America's largest population areas lately?

www.manaraa.com

- **Sequence Discovery** – compares current data to a pre-established norm to detect anomalies. Network management tools use this technique to alert system administrators to unusual user behavior.

Analyzing the performance of a data mining algorithm is a controversial process [Witt05]. There are many techniques one can use to evaluate the performance of an algorithm. Some involve examining the effects different parameters have on the algorithm's outcomes. For example, categorization algorithms are sensitive to the statistical regression technique used (cross-validation versus single pass). Others involve evaluating how well the algorithm performs against the test dataset. For example, is it able to process the data using only data that can fit into main memory or is it able to process data iteratively? These performance factors and the applicability of the results of the algorithm as well as how the outcomes can be viewed all build a ruler by which one can access and interpret the results of a data mining process [Witt05].

7.2.3.5 What about privacy?

One area of deep debate is the protection of information that may be used against persons. While not considered a nefarious activity, the results of data mining network information could lead to the discovery of usage patterns about individuals that could be used against them. For example, it is possible to use data mining to learn patterns of websites visitations. Part of that data could include private information such as logon credentials and financial information. While public disclosure of this information is

المنارة للاستشارات
www.manaraa.com

protected by law, data mining may require the inclusion of this information in order to successfully satisfy the suppositions made. Unfortunately, data mining tools make it easier for even novice users to obtain sensitive information which could compromise an individual's privacy. Researchers tend to agree that technology alone can not solve this dilemma. Amendments to appropriate privacy laws are needed [Thur00].

Wahlstrom and Roddick presented the following argument, "We exist in an environment of rapid change in which technology has an ever-increasing social relevance. The challenge now is to adapt our approaches to the application of new technologies, enabling us to use the tools technology provides wisely and with consideration for our society, its members, and its future," [Wahl01]. Clearly, the application of data mining concerning privacy and the ethical use of the data is a topic that transcends research and philosophy to the level of law and cultural practices[12].

7.2.3.6 Data Mining Tools and Environments

There are many data mining tools available for purchase or via open source or trial license. These products can be grouped into three categories; 1) integrated data mining tools that work with relational database management systems (RDMBS) or statistical systems, 2) workbench data mining environments that offer integrated data mining tools to support data ingestion, algorithm execution, analysis of results, and visualization, and 3) standalone data mining tools that implement one or more of the types of data mining algorithms. Examples of each of these types of tools are shown in

[12] Which also imparts a great deal of responsibility on data mining researchers and practitioners?

المنارة للاستشارات
www.manaraa.com

table 7-2 below. A comprehensive listing of data mining tools can be found in Dunham's text on data mining [Dunh03].

Category	Product	Vendor	Supported Functions
Integrated	Darwin	Oracle Corporation	Clustering, prediction, classification, association rules
Integrated	SQL Server	Microsoft Corporation	Clustering, prediction
Integrated	AnswerTree	SPSS Inc.	Classification
Workbench	AC^2	ISoft	Clustering, classification, prediction, segmentation
Workbench	Knowledge STUDIO	ANGOSS Software Corporation	Classification, clustering, prediction, rules
Workbench	XpertRule Miner	Attar Software Ltd.	Association rules, classification, clustering
Standalone	CART	Salford Systems	Classification
Standalone	Cubist	RuleQuest Research Ltd.	Numerical modeling
Standalone	See5	RuleQuest Research Ltd.	Classification

Table 7-2: Data Mining Products

The data mining tool used in this work is the Weka data mining environment[13]. Weka is available via open source and thus can be downloaded and used royalty-free [Witt05]. Weka is unique in that it is written in Java and can run on many different platforms. Weka provides a rich command-line tool as well as three GUIs for performing data mining. The workbench includes many of the data mining algorithms and contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization [Witt05].

[13] Oddly, Weka was not included in Dunham's list of data mining products.

المنارة للاستشارات
www.manaraa.com

7.2.3.7 Machine Learning versus Statistics: What's the difference?

Cynics may equate machine learning with artificial intelligence and view data mining as simply statistics plus marketing concepts. However, that cannot be farther from the truth. In truth, there is a continuum among machine learning and statistics. Both are used in the process of data analysis and cannot be removed from data mining. That is, one cannot execute a data mining model without statistics and the analysis of the results of those models against live data results is a by product which is machine learning. However, machine learning and statistics have had different traditions. Statistics has been focused on testing hypotheses whereas machine learning has been focused on forming a process as a search of possible hypothesis. But this is a vast oversimplification: statistics is far more than hypothesis testing and some machine learning techniques don't involve searching. Statistics is a foundation for data mining algorithms and evaluation techniques. Machine learning provides methods to learn from the results of data mining algorithms and models [Witt05].

7.3 Data Mining in a Versioning Environment

The goals for mining version data are similar to the goals of mining temporal data. In temporal databases, the goal to mining data is to discover frequent sequences where correlations are discovered among events in time. In a similar way, the goal of mining version data is to discover frequent occurrences of patterns within the version metadata. Here, the version metadata stores the time sequences [Rodd02]. Metadata is used to locate patterns that infer meaning among the metadata and thus add additional

المنارة للاستشارات
www.manaraa.com

knowledge about the versioned data. There have been studies of mining metadata in the text mining research area [Thur00]. Many of those techniques are applicable to mining version metadata.

Research in text mining has identified the need to track and predict certainty or confidence in the results of the analysis [Han01], whether the original data items (the content of the texts that are being mined) contain relevant data with a high degree of confidence [Thur00a]. In a similar way, the identification and tracking of reliability and confidence in the version data is a primary area of concern. The goals of the data mining algorithms for this project are to classify and categorize the data based on the version metadata. Algorithms are included that classify, predict and learn, with learning being the key to a continuum of use. Example suppositions are shown in table 7.3.

Supposition	Possible Outcomes
What is the confidence of an attribute version given its source?	Predicting missing values.
What is the confidence and reliability of a subset of the attribute version data?	Assessing patterns of the data.
What sources tend to provide data of low reliability or induce lower confidence levels?	Categorizing (clustering) the data.

Table 7-3: Example Suppositions for Mining Version Metadata

7.3.1 How are Attribute Versions Created?

The generation of attribute versions is the product of a merger of one or more datasets. There are several ways to accomplish this. Datasets could be merged using a program that reads two or more datasets and maps the data such that the attributes of a table in one dataset is mapped to the attributes of a table in the other dataset. Although

المنارة للاستشارات

www.manaraa.com

this scenario is a possible implementation, it is very dependent on a static mapping between the tables and is difficult to automate.

However, if one considers the advantages of a semantic web environment where all datasets are mapped using ontologies to domains – a process known as knowledge representation or knowledge modeling [Pere03], it is then possible to build relationships among the ontological domains so that any two datasets can be mapped to a common theme [Alle00]. More specifically, the ontologies define how the data in each table is mapped to the ontology of the domain in question. In a semantic web environment it becomes easier to identify collisions in the data. This is possible because the data is mapped to an ontology that defines its properties. When two objects from two different datasets are loaded, the ontologies can be used to detect the collisions [Anto04].

Furthermore, the collisions can be resolved either by collapsing the data to form a composite view of the object using aspects from each dataset or by storing the object using one set of attributes from one dataset (usually the more trusted one) and storing collisions among the attributes as attribute versions. It is this mechanism that is used by the sponsor of this work to generate attribute versions for their data.

7.3.2 ALV Metadata Preparation

The first step in the data mining process is problem definition. In this experiment, the problem or supposition proposed is based on a commonly occurring analytical assessment of the data – how reliable is this data? Applying that question to version metadata, the question can become how reliable is the version data? One possible

www.manaraa.com

supposition of the analysis could be that some sources are less reliable than others. Which sources are less reliable? Furthermore, one might also wish to ask, given sources that are less reliable, how are they grouped by sensitivity or confidence?

In the example data used in this experiment, the version metadata contained attributes that assess or categorize the data. These metadata fields include reliability (High, Medium, Low, Unknown), Confidence (Fact, Validated, Confirmed, Unconfirmed), Source (TV, SGC, PFJ, JPF, Reference, Radio, NID, Article, Newswire), and Sensitivity (UNCLASSIFIED, CONFIDENTIAL, SECRET)[14]. Reliability is a measure of how much the data (attribute version) can be trusted. Confidence is a measure that the analyst has with how factual the data is. Source is the originator or supplier of the information. Sensitivity is a measure of how data should be handled with respect to visibility (who can and can't see it). It was decided to use a generated dataset rather than an actual dataset due to several factors but primarily because the data itself is protected by intellectual rights and law.

All of these metadata attributes support the suppositions discussed above. Thus, an extract of this data along with the keys to link the attribute versions to the versioned data are necessary to begin the data mining process.

The next step in the data mining process is to prepare or transform the data. Since the Weka workbench was chosen as an environment to conduct the data mining experiment, the attribute version data had to be converted to the attribute-relation file format (ARFF). Fortunately, the attribute version data can be represented in this format

[14] This work does not contain any classified material. All markings are used purely for demonstration purposes and do not indicate any sensitivity whatsoever.

المنارة للاستشارات
www.manaraa.com

easily by extracting the data from the cluster version store and writing it to a file using the ARFF syntax. A standalone application (ALVDMPrep) was created to perform this conversion. ALVDMPrep requires the clustered version store to be offline to be processed. This was necessary to permit reading the file twice – once to produce the metadata that the ARFF format requires and once to read an attribute version and write it out in ARFF format. For more information on the ARFF format, see Witten and Franks' text on data mining [Witt05].

The creation of the ALVDMPrep application made it possible to produce a summary of the data concerning the attribute versions included in the version store. Attributes that contain nominal values (all of the attributes in the example do) are grouped and reported by frequency where all of the numeric data is reported with the mean, standard deviation, and variance. This data is helpful in permitting the analyst to view how the data is composed and can provide the analyst with evidence for choosing an algorithm. For instance, some algorithms work with nominal values while others work with numeric or date/time values. An example of the statistics generated for a simple test file are shown below.

```
% Statistics
% Attribute Versions: attribute_name
%  int, float values:  (mean, stddev, var)
%  nominal values      (value, count)
%                      frequency
% Metadata: attribute_name
%  int, float values:  (mean, stddev, var)
%  nominal values      (value, count)
%-------------------------------------------------------
% Attribute Version: x
%     (15.0000, 2.2804, 5.2000)
%     Frequency: 100.00%
%
% Metadata: status
```

www.manaraa.com

```
%     (ok3,4), (ok2,3), (ok1,3)
% Metadata: fvalue
%     (1.5760, 0.3077, 0.0947)
% Metadata: ivalue
%     (1328.6000, 26.0546, 678.8400)
% Metadata: alv_key
%     (5.5000, 2.8723, 8.2500)
%
```

The data used in the experiment consists of approximately 124,000 attribute versions of the airmen dataset (see Appendix A for a complete description of this dataset). A training set was generated for the data that included all permutations of all of the metadata attributes (excluding keys). This training set was used to train the classifier algorithms. An example of the statistics generated for this file when the ALVDMPrep application was run is shown below.

```
% Statistics
%
% Attribute Versions: attribute_name
%  int, float values:  (mean, stddev, var)
%  nominal values      (value, count)
%                      frequency
% Metadata: attribute_name
%  int, float values:  (mean, stddev, var)
%  nominal values      (value, count)
%-------------------------------------------------------
%
% Metadata: Sensitivity
%     (UNCLASSIFIED,322), (SECRET,322), (CONFIDENTIAL,340)
% Metadata: Source
%     (TV,106), (SGC,44), (PFJ,46), (Reference,100), (JPF,45),
(Radio,84), (NID,209), (Article,186), (Newswire,133)
% Metadata: Confidence
%     (Unconfirmed,332), (Confirmed,142), (Fact,202), (Validated,252)
% Metadata: Reliability
%     (High,117), (Medium,101), (Unknown,244), (Low,416)
```

7.3.3 Algorithm Choice

Now that the data is prepared, the next step in the data mining process is experimenting with algorithms. This was accomplished by first examining all of the

المنارة للاستشارات
www.manaraa.com

applicable algorithms and choosing the types of algorithms that would best answer the suppositions. In this case, a clustering algorithm was best because we wanted to find patterns in the data and group those patterns to visual analysis.

The Weka workbench provides many different clustering algorithms. These include, Cobweb, EM, FarthestFirst, MakeDensityBasedClusterer, and SimpleKMeans. Cobweb implements both the Classit and Cobweb algorithms for both numeric and nominal values. EM is an algorithm that uses expectation maximization and permits the analyst to choose the number of clusters to generate. FarthestFirst implements the farthest first algorithm using k-means. MakeDensityBasedClusterer is a wrapper for a cluster algorithm used to return distribution and density information. SimpleKMeans implements a k-means algorithm for locating clusters for nominal values.

The available algorithms were run several times on the training data producing few or no comprehensible results. This is because the training data contains all possible groupings thus no clusters beyond the examples. For instance, if one were to cluster the training data on reliability, the algorithms generally reported exactly four clusters. However, when applied against the test data, the SimpleKMeans algorithm produced the expected results showing two clusters for the reliability attribute. A check through the test data and a comparison of the output of the ALVDMPrep application confirm this. None of the other algorithms tested provided the expected results. Most showed widely varying cluster groupings and were very sensitive to changes in input parameters. Thus, the SimpleKMeans algorithm was chosen as the best clustering algorithm for addressing the

المنارة للاستشارات
www.manaraa.com

suppositions and fitting the data. The results of running the sample data using the SimpleKMeans algorithm are shown below.

```
=== Run information ===
Scheme:       weka.clusterers.SimpleKMeans -N 2 -S 10
Relation:     airmen_alv
Instances:    112799
Attributes:   4
              Reliability
              Confidence
              Source
              Sensitivity
Test mode:    user supplied test set: 1018 instances

=== Model and evaluation on test set ===
kMeans
======
Number of iterations: 3
Within cluster sum of squared errors: 232191.0

Cluster centroids:

Cluster 0
      Mean/Mode:  Low Unconfirmed NID SECRET
      Std Devs:   N/A   N/A   N/A   N/A
Cluster 1
      Mean/Mode:  Medium Validated Article SECRET
      Std Devs:   N/A   N/A   N/A   N/A

Clustered Instances
0       810 ( 80%)
1       208 ( 20%)
```

The last step in the process is model validation. The next section explains the results of the experiments and presents a validation of the model generated.

7.4 Analysis

Validation of data mining model results should take into consideration not only what the results are, but also how they answer the suppositions, and how well the algorithm performed. For this experiment, the algorithm ran in a reasonable amount of

المنارة للاستشارات
www.manaraa.com

time, requiring only approximately 40 minutes for a complete run, and did not require any additional system resources[15].

In this case, validation of the data mining model generated in the experiments required examining the results using visualization methods (graphs) of the data with respect to the clusters found and the distribution of the data itself. Since distribution counts were available, the results of the data mining algorithm should have depicted the same distribution with respect to the location of the data points on the graph. The most important aspect to consider is how well the results answered the suppositions. This can once again be best addressed by examining the results in a graphical form.

What was not known, however, was how the data was going to be grouped. Furthermore, it was not possible to anticipate how many groups to expect. The following figures are the visualization of the results of the SimpleKMeans algorithm.

Figure 7-4 depicts the locations of the reliability data (y-axis) and source (x-axis). The clusters are represented as different colors. Clearly, cluster 0 (indicated in blue) contains most of the data that has a reliability value of low or unknown and is primarily from three sources; Radio, NID, and Newswire[16]. The knowledge that can be learned from this graphic is that most of the data is of low reliability and originates from three of the sources more frequently than the others.

[15] It should be noted, however, that the system that hosted the Weka software suffered degraded performance while the algorithm was running. In essence, it became a denial of service incident. Clearly, this algorithm should either be run on a dedicated workstation or left alone to ponder itself for a time.
[16] This fictional data shows that you can't always believe what you hear on the radio, read in the paper, or get from three letter agencies.

المنارة للاستشارات
www.manaraa.com

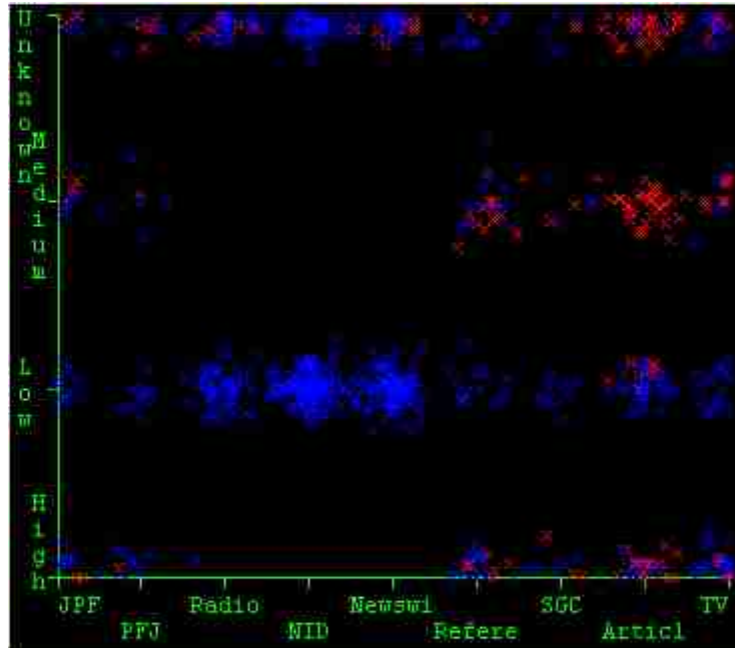


Figure 7-4: Results of Clustering - Source versus Reliability, Clusters Highlighted

Figure 7-5 depicts the same results presented using color coding on the sources themselves. As expected, the sources are grouped by color vertically shown in relation to the associated reliability value. This form of the graph made it possible to identify which sources had the highest concentration of hits for each reliability value. The knowledge that can be learned from this graphic is that the NID source can be a candidate for removal from the pool of sources of information due to its demonstrated poor reliability.

المنارة للاستشارات
www.manaraa.com

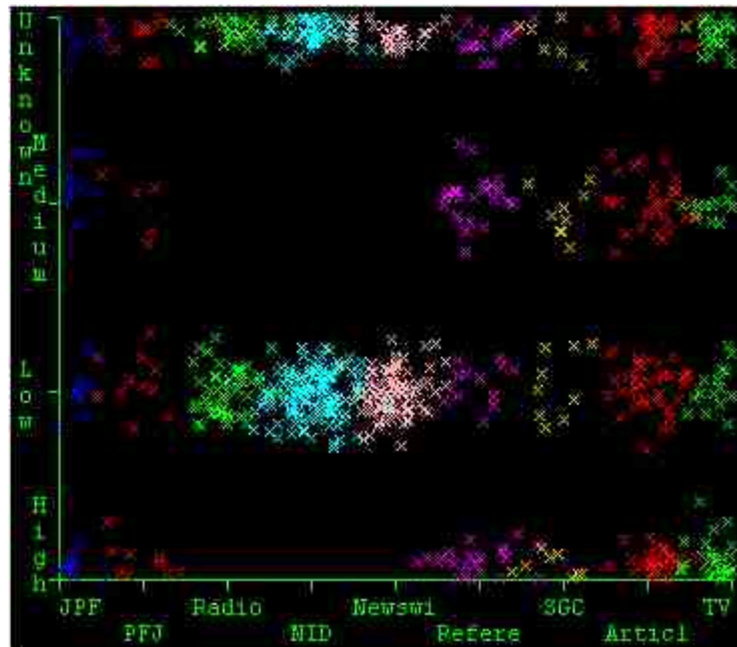


Figure 7-5: Results of Clustering - Source versus Reliability, Sources Highlighted

Figure 7-6 depicts the results presented using sources (x-axis) and confidence (y-axis). This form of the graph made it possible to identify that the confidence factor for the data seems to have an even distribution among the range of confidence values but shows that the data is once again centered in the three sources previously identified. The knowledge that can be learned from this graphic is that although the NID source seems to have more data points that the other sources, the confidence attribute was not a factor in determining how the data was clustered. Also, the data appears to have an even distribution of confidence assignments across the sources. What this could permit is the analysis of the data from the viewpoint that confidence level is not affected clustering of the data by reliability.

www.manaraa.com

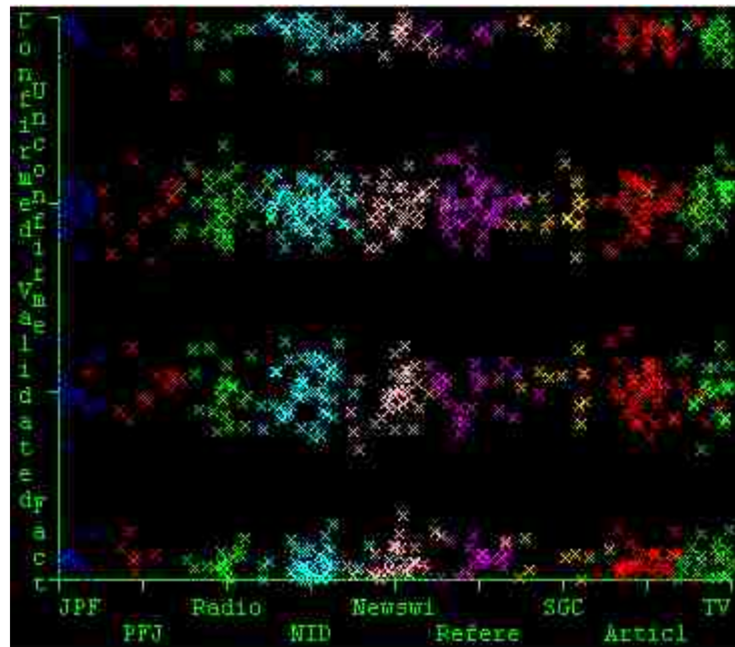


Figure 7-6: Results of Clustering - Source versus Confidence, Sources Highlighted

Figure 7-7 depicts the results presented using sources (x-axis) and confidence (y-axis). This form of the graph made it possible to identify that the data also has an even distribution of sensitivity and is again centered around the three sources. The knowledge that can be learned from this graphic is that the sensitivity attribute also does not affect the clustering of the data by reliability.

We have seen in figure 7-4 a description of the data as to its originators, figure 7-5 identifies patterns of reliability factors with respect to the sources, figure 7-6 confirms that the confidence factor for the data seems to have an even distribution among the range of confidence values but shows again that the majority of the data is centered in the three sources previously identified, and figure 7-7 confirms that the data also has an even distribution of sensitivity and confirms again that the data is centered around the three sources.

المنارة للاستشارات
www.manaraa.com

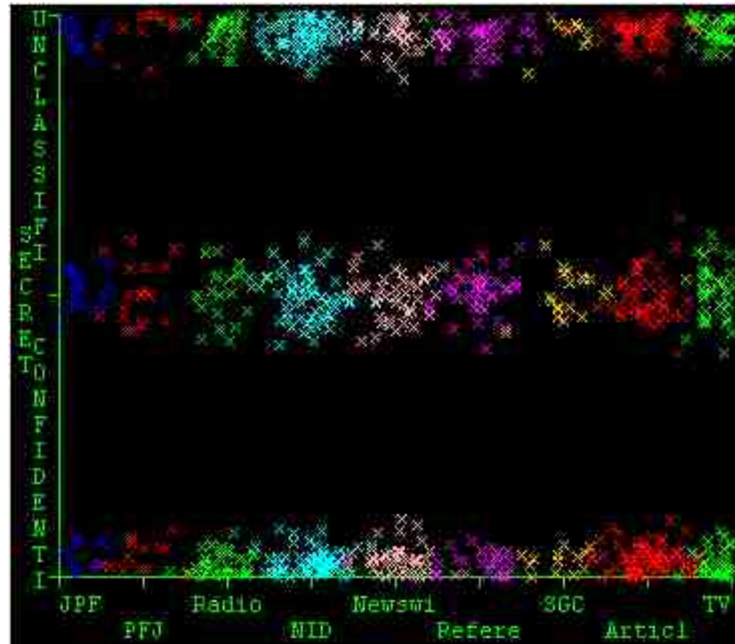


Figure 7-7: Results of Clustering - Source versus Sensitivity, Sources Highlighted

Therefore, the results of the data mining algorithm answer the suppositions in the following manner:

How reliable is the version data?

The data can be grouped into two groups; 1) a group (cluster) where reliability is low, confidence is unconfirmed, the majority source provider is NID, and the sensitivity of the data is SECRET, and 2) reliability is medium, confidence is validated, the source is primarily articles, and the sensitivity of the data is SECRET. From closer examination of the results, one can determine that the NID source is generally a bad choice for reliable data.

www.manaraa.com

Are some sources less reliable than others? Which sources are less reliable?

Yes. The NID, Radio, and Newswire sources are less reliable than the others.

Given sources that are less reliable, how are they grouped by sensitivity or confidence?

The graphs indicate an even distribution among the confidence and sensitivity values.

7.5 Conclusion

The experiments show one example of how data mining can be used to find patterns in the attribute version data. In this case, the experiments demonstrate the benefits of mining the attribute version metadata to gain knowledge about the version data. In this case, it confirmed the supposition that there are sources that are less reliable than others. This information can be used by analysts to generate extracts of their versioned data joined with the attribute version data that can be more reliable and instill a greater confidence with the analyst. Therefore, data mining is a viable method for gaining additional knowledge about the attribute versions which can be applied to the versioned data for further analysis.

7.6 Future Work

The Achilles heel of most data mining algorithms is the static nature by which the algorithms operate. Most operate only on data that remains constant. They fail to adapt well (if at all) to changes in the data. An algorithm that can scale and adapt (learn) would be the best for the types of analysis that this project will support [Dunh03].

المنارة للاستشارات

www.manaraa.com

Additionally, no data mining algorithms are designed to work with version data. An interesting research project would be to develop or modify one of the clustering algorithms to consider version data by generating all possible permutations of the versioned data joined with the attribute versions. Similarly, modifying a cluster algorithm to display those permutations would permit analysts to locate patterns of version data that adds specific value (through suppositions). This area alone would comprise a great deal of work and could prove very beneficial to analyzing data in versioning systems.

المنارة للاستشارات

www.manaraa.com

Chapter Eight - Conclusion

This chapter examines the implementation of the research and supporting project, presents an analysis of the experiments conducted, states the conclusion and supporting evidence for the theoretical material presented, and provides descriptions of key areas for future study.

8.1 Analysis of the ALV Experiment

The supporting project for this work was an experiment to test the feasibility and access the implementation of the Attribute-Level Versioning (ALV) concept. Test the ability of the sponsor to support an endeavor that has been considered by many to be too risky to accomplish in a reasonable timeframe[1]. Furthermore, ALV was considered a technology that could not be integrated into a relational database system.

The project as an experiment was a success. Not only was the ALV technology feasible, the experiments included in this work demonstrate that the ALV technologies have a sound foundation in theory and perform well in use. This work also shows that the ALV concept itself is sound and that it can be successfully integrated into a commercial

[1] As a result, the project and the author's professional reputation has survived several assassination attempts.

المنارة للاستشارات
www.manaraa.com

relational database system. The following section summarizes the results of all of the experiments conducted on the technologies included in this work.

8.2 Conclusion

The literature search has shown that the state of the technologies and theories in Computer Science did not include any in-depth work in the area of versioning of data in a relational database. Although object oriented databases can inherently support a versioning concept and object relational databases can support a horizontal mechanism for version storage, none of the database paradigms support versioning at the attribute level while maintaining a functional connection to traditional relational databases.

This work demonstrates that the ALV technology can be integrated with a relational database system while not affecting its performance, functionality, and stability[2]. The benefit of adding a versioning system like ALV to a relational database system is that it enables scientists and analysts to prepare data for use in rigorous database applications, drawing from the repository of all known or predicted values. Therefore, ALV is a uniquely conceived idea that has merit in the relational database paradigm. The continued exploration of versioning capabilities and the implementation of ALV permits growth of a new direction in data versioning, the ability to store every permutation of a data's attributes.

[2] Pertaining to the theoretical (academic) application of theory. Commercial relational database systems often compromise the finer details of relational theory for the sake of mass reuse and generalization of functionality. Hence the persistent and growing ga p between academic rigor and industry isotropic applications.

المنارة للاستشارات

www.manaraa.com

However, simply adding a versioning mechanism to a relational database system is not the only challenge. Traditional relational database systems store data to maximize the retrieval of relational data. None of the relational database systems have storage mechanisms to store and retrieve version information. The integration of ALV with a relational database system using traditional data storage mechanisms would have required force-fitting versioned data into generalized and thus less efficient storage structures that often result in larger tables, the use of surrogate or superkeys, duplication problems, and complexity issues.

The clustered version store (CVS) is the cornerstone of the ALV system. By demonstrating the ability to store attribute versions in a dedicated, specialized physical storage mechanism that utilizes a buffer management system for caching, the clustered version store is the foundation of a versioning system that can be integrated in a relational database environment. The physical store is sound and performs admirably when compared to the commercially available physical store available in MySQL[3].

The CVS would be incomplete without a mechanism to index and thus access the data in a time-efficient (timely) manner. The integration of the version index into the ALV system is therefore imperative in order to provide the speed necessary for a system to be considered for production use.

This work has shown that the version index mechanism is reliable and performs well as demonstrated by the ability to store attribute versions in a dedicated, specialized physical storage mechanism and accessing the data using rapid index resolution. The

[3] Although MySQL supports a number of physical stores, the comparisons made in this work were down with MyISAM because MyISAM most closely resembles that of the ALV physical store.

www.manaraa.com

experiments and real world experience using the ALV system with the version indexing mechanisms demonstrate that a fast indexing mechanism is required to ensure high speed performance of retrieval of versioned data for a versioning system and that the version indexing mechanism can be supported in a relational database system. Furthermore, the version indexing mechanism was shown to have superior performance for retrieving an attribute version chain from the CVS as compared to the inherent data storage and indexing mechanisms in MySQL.

Additionally, the B^2+ and mB^2+ trees are a unique form of B+ tree that has buffer management, concurrency, and transaction support built into the data structures and algorithms. This tight integration of these features makes the new variants of the venerable B+ tree viable mechanisms for increasing the performance of indexing mechanisms.

The data mining experiments conducted against version metadata show how data mining can be used to find patterns in the attribute version data. In this case, the experiments demonstrate the benefits of mining the attribute version metadata to gain knowledge about the version data. In this case, it confirmed the supposition that some sources are less reliable than others. This information can be used by analysts to generate extracts of their versioned data joined with the attribute version data that are deemed more reliable and instill a greater confidence with the analyst. Therefore, data mining is a viable method for gaining additional knowledge about the attribute versions which can be applied to the versioned data for further analysis.

المنارة للاستشارات
www.manaraa.com

The ALV query optimizer, query tree, and query execution engine represent technology that demonstrates the potential of implementing such technologies in a production relational database system. The fact that these technologies are based on academic views of implementation proves once again that academic rigor can be translated directly to industry without compromising the "science" behind the details. As the experiments show, the technologies presented represent effective and efficient technologies for use in a versioning system.

The analysis of modern datasets requires the use of specialized algorithms and storage and retrieval mechanisms to identify, deconflict and assimilate variances of attributes for each entity encountered. These variances, or versions of attribute values, contribute meaning to the evolution and analysis of the entity and its relationship to other entities. A new, distinct storage and retrieval mechanism will enable analysts to efficiently store, analyze and retrieve the attribute versions without unnecessary complexity or additional alternations of the original or derived dataset schemas. This mechanism is the ALV system. The ALV system enables the storage and retrieval of version information and can be used to add considerable knowledge to a versioned data store. All of this can be accomplished by integrating the versioning system with a commercial relational database system.

8.3 Future Work

The following sections summarize the areas of future work that have been identified in this work. While the ALV project is a success and performs well in practice,

المنارة للاستشارات
www.manaraa.com

there are still many small enhancements to the experimental source code needed to achieve a fully reliable and robust server. For example, while concurrency was a major design point and the technologies tested to achieve good concurrent behavior, extensive tests in a highly concurrent multi-user environment were not conducted. Additional tests are necessary to ensure that the concurrent mechanisms in the ALV technologies continue to operate in a dynamic environment under heavy load.

8.3.1 Clustered Version Store

Although the clustered version store performs well and outperforms the native storage mechanism of MySQL, there are areas that can be improved. Despite the tendency and practice of database system vendors to rely on the base operating system for file I/O support, much could be gained by developing a native I/O mechanism that communicates directly with the hardware. This would enable a more efficient use of disk space and eliminate the need to coordinate directly with the operating system. The drawback to this approach is that an operating system driver must be created so that the base operating system can communicate with the device. It would be enlightening to develop such a storage mechanism and compare its performance with that of native data stores and the data store presented here.

On a more subtle scale, there are improvements that can be made to the clustered version store that may increase performance even further. For example, an active space reclaim process could eliminate the concern for large gaps in the file structure under

المنارة للاستشارات
www.manaraa.com

heavy insert and delete operations. Furthermore, an active space reclaim process would eliminate the need to perform periodic maintenance on the files.

A vulnerability of the implementation of the clustered version store is that it does not currently have an active deadlock prevention algorithm. Additional overhead mechanisms may be necessary to support active deadlock recovery.

For a more robust application of versioning, one would also consider expanding the buffer management subsystem to include recovery mechanisms that can recover data in the event of an unexpected system termination.

Aside from the above improvements that are largely expansive in nature[4], the most beneficial additional (perhaps even necessary) modification is to convert the implementation code to be as platform independent as possible. The system currently runs on a Windows-based operating system. Additional work will be necessary to convert some of the lower-level I/O code to a platform independent basis. Fortunately, that capability has already been demonstrated in the MySQL source code.

8.3.2 Version Indexing

Construction of the version index from existing data is a concern that should be addressed in the near future. This could be an especially important performance issue if the database systems that implement ALV are used for high-speed data processing. The technology of batch-construction of Kim [Kim01] should be investigated for incorporation into the version index mechanisms.

[4] That is, they tend to make the system larger and more prone to performance and complexity issues.

المنارة للاستشارات

www.manaraa.com

Although the version index performed well with the ALV buffering mechanism, it is possible that the buffering mechanism may need to be altered once a sufficiently large data set is discovered. Currently, none of the large data sets tests have shown any unusual behavior and the index and buffer mechanism work well. Performance under these circumstances has been proportional to the size and complexity of the data. Further research will be necessary to test the cumulative effects of very large data sets on the version index and buffer mechanism.

The application of the B^2+ tree and mB^2+ tree in the ALV system is not as flexible as it could be. Additional work will be necessary to provide the tools necessary to create alternative indexes on any given attribute value or metadata attribute within an attribute version. These extra tools will provide additional opportunities to tune the versioned database for optimal performance based on the need and intended use of the version system.

While the concurrency and transaction mechanisms work well, there is no support for recovery. Database recovery mechanisms are designed to be able to recover the state of the database system should the system become unstable or crash. With a recovery system, such as a log-based journal, where all operations and their outcomes are stored, all but the most severe of system failures could be recoverable and the state of the database rebuilt on restart. The ALV system does not support any form of logging or recovery. Additional work is necessary to implement this feature into the ALV system, making it applicable in environments where recovery is a high priority or necessity.

المنارة للاستشارات
www.manaraa.com

Lastly, the B^2+ tree and mB^2+ tree mechanisms described above should be generalized for use with more traditional physical data stores. This will ensure that the technology is added to the collection of many successful indexing mechanisms bearing the legacy of Bayer and McCreight [Baye72].

8.3.3 ALV Query Optimizer and Execution Engine

Although the cost optimization step of the ALV query optimizer is very effective at identifying the benefits of using indexes for locating (iterating) through a relation, the query optimizer could be designed to optimize or balance joins better than it does. For most joins, the existing strategy works well and will continue to generate near optimal executions. However, for those complex joins that are formed in an environment that has complex indexes and multi-level indexes, the cost optimization step should be modified to balance joins better. Once this has been accomplished, the ALV query optimizer will be complete and robust enough to handle any environment and use.

The ALVExecute class can be enhanced and query execution reduced by using a multithreaded execution variant of the pipeline. That is, each operation in the tree could be executed as a separate thread, providing appropriate synchronization using mechanisms such as queues for preemptive and wait conditions. However, care must be taken to avoid deadlock and race conditions.

One of the most important features of MySQL is the query cache [MySQ05]. The query cache is a mechanism that stores the results of executed queries for reuse. This includes storing the parsed and optimized query, but also the query results. This gives

المنارة للاستشارات

www.manaraa.com

MySQL the ability to respond quickly to repetitive queries and exceed the performance of database systems that do not cache queries. The ALV query tree can be used to implement a query cache. Work will need to be done to associate a result set (relation class) with a query tree. Fortunately, the implementation can easily be adapted for serialization and organization of a query cache. However, like the query tree and query execution, the ALV query cache will be functionally equivalent but with a distinctly different implementation.

The SQL_{ALV} extensions developed to support versioning in a relational database system are not complete. A complete set of SQL commands would include the alter commands as well as enhancements to the select command for greater flexibility in performing complex queries. Additional development is necessary to complete the SQL_{ALV} commands.

8.3.4 Data Mining in a Versioning Environment

The Achilles heel of most data mining algorithms is the static nature by which the algorithms operate. Most operate only on data that remains constant. They fail to adapt well (if at all) to changes in the data. An algorithm that can scale and adapt (learn) would be the best for the types of analysis that this project will support [Dunh03].

Additionally, no data mining algorithms are designed to work with version data. An interesting research project on the development or modification of one of the clustering algorithms would be to consider version data by generating all possible permutations of the versioned data joined with the attribute versions and clustering those

المنارة للاستشارات
www.manaraa.com

permutations would permit analysts to locate patterns of version data that adds specific value (through suppositions). This area alone would comprise a great deal of work and could prove very beneficial to analyzing data in versioning systems.

8.3.5 Other Areas

One of the guiding goals for the ALV system was to include the ability to store attribute versions and assign temporal properties to them. While the version metadata can be constructed to store temporal data both using time series and time stamping (hence ALV support bi-temporal data), the ALV system was not designed to support temporal analysis. Temporal queries are more than just simple roll-up or sequence chains. Temporal queries involve logic that can be used to discover how values change over time, the values of an entity's attributes at a specific time reference, and can even be used to store data that is not yet relevant[5]. Additional work would be necessary to expand the ALV system to process temporal queries. This can be accomplished by redesigning the query processor and query execution to include the syntax for and a mechanism to evaluate temporal query statements. Adding temporal query capabilities to ALV will ensure that further analysis of version data is possible with respect to time.

Another area of interest would be to replace the MySQL query data structure with that of the ALV query tree and the MySQL query engine with the ALV query optimizer. The results should be a more efficient, robust, and maintainable system than what is currently available as the MySQL system. This is possible because the query tree is a

[5] This list is by no means exhaustive. A host of many interesting temporal queries can be constructed using temporal logic.

المنارة للاستشارات
www.manaraa.com

well designed data structure that, when combined with the ALV query optimizer and execution engine, would permit the MySQL database system to execute queries faster. Furthermore, it has been shown that the ALV query tree and query optimizer can outperform the equivalent technologies in MySQL. However, it should be noted that there is a performance issue with the MySQL parser with respect to building query trees. This limitation will have to be overcome in order to fully replace and enhance the MySQL database system with ALV technologies. When complete, the ALV technologies could take the MySQL database system to new heights of performance.

المنارة للاستشارات
www.manaraa.com

Bibliography

[Alle00] Allen, G. N., and S. T. March. "Temporal Database Management and the Representation of Temporal Dynamics." Univeristy of Minnesota Working Papers 2000. April 2000 <http://misrc.umn.edu/workingpapers/abstracts/0004.aspx>

[Alle02] Allen, R., and K. Kennedy. Optimizing Compilers for Modern Architectures. San Francisco: Morgan Kaufmann, 2002.

[Alte02] Alter, J. "Actually, the Database is God." Newsweek Nov 04 2002

[Anto04] Antoniou, G. and F. van Harmelen. A Semantic Web Primer. Cambridge, Massachussetts: MIT Press, 2004.

[Anon04] Anonymous. "Implementation of a B-Tree Database Class." Online posting. 09 Jun 2004. The Code Project. 29 Jun 2004. <http://www.codeproject.com/database/btreedbclass.asp>.

[Baas88] Baase, S. J. Computer Algorithms : Introduction to Design and Analysis. 2nd Edition Reading: Addison-Wesley, 1988.

[Badi02] Badia, A., M. Chanda, and B. Cao. "Adding Subqueries to MySQL, or What Does it Take to Have a Decision-Support Engine?" Data Warehousing and OLAP McLean, VA 2002. 29-46.

[Bane03] Bane, J. "How Many's Too Many?" SQL Server Magazine (June 2003): 31-35.

[Bato79] Batory, D. S. "On Searching Transposed Files." ACM Transactions on Database Systems 4.4 (1979): 531-544.

[Bato81] Batory. D. S. "B+ Trees and Indexed Sequential Files: A Performance Comparison." Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data Ann Arbor, Michigan 1981. 30-39.

[Bato82] Batory, D. S. "Optimal File Designs and Reorganization Points." ACM Transactions on Database Systems 7.1 (1982): 60-81.

[Baye72] Bayer, R., and C. McCreight. "Organization and maintenance of large ordered indexes." Acta Informatica 1.3 (1972): 173-189.

[Baye77] Bayer, R., and K. Unterauer. "Prefix B-trees." ACM Transactions on Database Systems 2.1 (1977): 11--26.

المنارة للاستشارات

www.manaraa.com

[Baye77a] Bayer, R., and M. Schkolnick, "Concurrency of Operations on B-Trees." Acta Informatica 9.1 (1977): 1-21.

[Beck03] Beck, D. and J. Grant. "Semantic Web Scalability and Storage: Mapping Semantic Web Data with RDBMSes." W3C Semantic Web Advanced Development for Europe Chapter 10 (2003).

[Belu98] Belussi, A., E. Bertino, and B. Catania. "An Extended Algebra for Constraint Databases." IEEE Transactions on Knowledge and Data Engineering 10.5 (1998): 686-705.

[Ben-90] Ben-Ari, M. Principles of Concurrent and Distributed Programming. New York: Prentice-Hall, 1990.

[Bere00] Berenson, H. and K. Delaney. "Microsoft SQL Server Query Processor Internals and Architecture." 2000 <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsql7/html/sqlquerproc.asp>.

[Bili87] Biliris, A. "Operation Specific Locking on B-Trees." Proceedings of the Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems San Diego, CA 1987. 159-169.

[Bili92] Biliris, A. "An Efficient Database Storage Structure for Large Dynamic Objects." Proceedings. Eight International Conference on Data Engineering Tempe, AZ 1992. 301-308.

[Bill04] Billings, K. "A TPC-D Model for Database Query Optimization Cascades." MS Thesis. Portland State University. 2004.

[Blak04] Blake, M. "Spatio-Temporal Databases." 05 May 2004 <http://www.geog.leeds.ac.uk/people/m.blake/magis94/gisjg/exproc/papp.htm>.

[Bohl98] Bohlen, M., C. S. Jensen and B. Skjellaug. "Spatio-Temporal Database Support for Legacy Applications." Proceedings of the 1998 ACM Symposium on Applied Computing Atlanta, Georgia, United States 1998. 226 - 234.

[Bona03] Bonatti, P., Y. Deng, and V. S. Subrahmanian. "An Ontology-Extended Relational Algebra." IEEE International Conference on Information Reuse and Integration 2003. 192-199.

[Brei04] Breitman, K. K., J. C. Sampaio do Prado Leite. "Lexicon Based Ontology Construction." Lecture Notes in Computer Science Editors: Carlos Lucena, Alessandro Garcia, Alexander Romanovsky, et al. Springer-Verlag Heidelberg, February 2004, pp.19-34

[Chak90] Chakravarthy, U. S. "Logic-Based Approach to Semantic Query Optimization." ACM Transactions on Database Systems 15.2 (1990): 162-207.

المنارة للاستشارات

www.manaraa.com

[Cham81] Chamberlain, D. D., et. al. "Support for Repetitive Transactions and Ad Hoc Queries in System R." ACM Transactions on Database Systems 6.1 (1981): 70-84.

[Cham81a] Chamberlain, D., et. al. "A History and Evaluation of System R." Communications of the ACM 24.10 (1981): 632-646.

[Chatt04] Chatterjee, R., G. Arun, S. Agarwal, B. Speckhard and R. Vasudevan. "Using Applications of Data Versioning in Database Application Development." Proceedings of the 26th International Conference on Software Engineering (ICSE'04) Washington, DC. 2004. 315-325.

[Chen01] Chen, Z. Data Mining and Uncertain Reasoning: An Integrated Approach. New York: John Wiley & Sons, 2001.

[Chon01] Chong, E. I., S. Das, A. Yalamanchi, M. Jagannath, C. Freiwald, J. Srinivasan, A. Tran; R. Krishnan. "B+-Tree Indexes with Hybrid Row Identifiers in Oracle8i." Proceedings. 17th International Conference on Data Engineering (2001):341-348.

[Chou94] Chou, H. T., and D. J. DeWitt. "An Evaluation of Buffer Management Strategies for Relational Database Systems." Readings in Database Systems. Ed. M. Stonebraker. San Mateo: Morgan Kaufmann Publishers, 1994. 153-167.

[Cohe04] Cohen, S. M. "Identity, Persistence, and the Ship of Theseus." 14 Jan 2004. University of Washington. 18 May 2004 <http://faculty.washington.edu/smcohen/320/theseus.html>.

[Cole94] Cole, R. L. "Optimization of Dynamic Query Evaluation Plans." Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data Minneapolis, MN 1994. 150-160.

[Come79] Comer, D. "The Ubiquitous B-Tree" ACM Computing Surveys 11.2 (1979):121-137.

[Cook78] Cook, T. "A Dynamic Address Computation Mechanism for use in Database Management." Proceedings of the 1978 ACM SIGMOD Conference on Management of Data 1978. 78-87.

[Corm01] Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein An Introduction to Algorithms. 2nd ed. Boston: McGraw-Hill, 2001.

[Daco03] Daconta. M. "Designing the Smart-Data Enterprise." Enterprise Architect (Winter 2003): 18-23.

[Das95] Das, D., and Batory, D. "Prairie: A Rule Specification Framework for Query Optimizers." Proceedings of the Eleventh International Conference on Data Engineering Austin, TX 1995. 201-210.

المنارة للاستشارات
www.manaraa.com

[Date90] Date, C. J. Relational Database Writings 1985-1989. Reading: Addison-Wesley, 1990.

[Date92] Date, C. J., and H. Darwen. Relational Database Writings 1989-1991. Reading: Addison-Wesley, 1992.

[Date00] Date, C. J., and H. Darwen Foundation for Future Database Systems: The Third Manifesto. Reading: Addison-Wesley, 2000.

[Date01] Date, C. J. The Database Relational Model: A Retrospective Review and Analysis. Reading: Addison-Wesley, 2001.

[Date04] Date, C. J. An Introduction to Database Systems. 8th ed. Boston: Addison Wesley, 2004.

[Date04a] Date, C. J. "Appendix D Storage Structures and Access Methods." An Introduction to Database Systems. Boston: Addison-Wesley, 2004. 985-1030.

[Dela04] Delaney, K. "Mixed Extent Usage." SQL Server Magazine (September 2004): 49-50.

[Denn78] Denning, P. "The Working Set Model for Program Behavior." Communications of the ACM 11.5 (1968): 323-333.

[Devo04] Devore, J. L. Probability and Statistics for Engineering and the Sciences. 6th ed. Belmont, CA: Thompson Brooks/Cole, 2004.

[Dey96] Dey, D., T. Barron, and V. Storey. "A complete temporal relational algebra." VLDB Journal 5.3 (1996):167-180.

[Ding01] Ding, Y."IR and AI: The role of ontology." Proceedings of the 4th International Conference of Asian Digital Libraries Bangalore, India: 2001.

[Dong82] Dong, J., and R. Hull. "Applying Approximate Order Dependency to Reduce Indexing Space." Proceedings of the 1982 ACM SIGMOD International Conference on Management of Data Orlando, Florida 1982. 119-127.

[Dong04] Dong, X., Y. Lin, and L. Zou. "Conceptual Architecture of PostgreSQL." 28 2002. University of Waterloo. 02 Jun 2004 <http://swag.uwaterloo.ca/~lzou/cs798arch/as1/a1.htm>.

[Donn02] Donnelly, C. and R. Stallman. "Bison: The Yacc-compatible Parser Generatore." Free Software Foundation, Inc. 2002 <http://www.gnu.org/software/bison/bison.html>.

[Dunh03] Dynham, M. H. Data Mining Introduction and Advanced Topics. Upper Saddle River, New Jersey: Prentice Hall, 2003.

www.manaraa.com

[Effe84] Effelsberg. W., and T. Haerder. "Principles of Database Buffer Management." ACM Transactions on Database Systems 9.4 (1984): 560-595.

[Elha84] Elhardt, K., and R. Bayer. "A Database Cache for High Performance and Fast Restart in Database Systems." ACM Transactions on Database Systems 9.4 (1984): 503-525.

[Elma90] Elmasri, R. and G. T. J. Wuu. "A Temporal Model and Query Language for ER Databases." Proceedings. Sixth International Conference on Data Engineering Los Angeles, CA 1990. 76-83.

[Elma93] Elmasri, R., V. Kouramajian and S. Fernando. "Temporal Database Modeling: An Object-Oriented Approach." Proceedings of the second international conference on Information and Knowledge Management Washington, D.C. 1993. 574-585.

[Elma03] Elmasri, R., and S. B. Navathe Fundamentals of Database Systems. 4th ed. Boston: Addison-Wesley, 2003.

[ESRI00] Modeling our World -- An ESRI Guide to Geodatabase Design. ESRI, 2000.

[Fagi79] Fagin, R., J. Nievergelt, N. Pippenger, and H. R. Strong. "Extendible Hashing - A Fast Access Method for Dynamic Files." ACM Transactions on Database Systems 4.3 (1979): 315-344.

[Falo95] Faloutsos, C. "Flexible and Adaptable Buffer Management Techniques for Database Management Systems." IEEE Transactions on Computers 44.4 (1995): 546-560.

[Fayy96] Fayyad, U., G. Piatetsky-Shapiro, and P. Smyth. "From Data Mining to Knowledge Discovery in Databases." Artificial Intelligence Magazine Fall 1996: 37-54.

[Feng98] Feng, L., H. Lu, and A. Wong. "A Study of Database Buffer Management Approaches: Towards the Development of a Data Mining Based Strategy." 1998 IEEE International Conference on Systems, Man, and Cybernetics San Diego, CA 1998. 2715-2719.

[Fens03] Fensel, D., J. Hendler, and H. Lieberman Spinning the Semantic Web. Cambridge: MIT Press, 2003.

[Fran96] Franklin, M. J., B. T. Jonsson, and D. Kossmann. "Performance Tradeoffs for Client-Server Query Processing." Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data Montreal, Canada 1996. 149-160.

[Frey86] Freytag, J. C., and N. Goodman. "Rule-Based Translation of Relational Queries into Iterative Programs." Proceedings of the 1986 ACM SIGMOD

المنارة للاستشارات
www.manaraa.com

International Conference on Management of Data Washington, D.C. 1986. 206-214.

[Frey89] Freytag, J. C., and N. Goodman. "On the Translation of Relational Queries into Iterative Programs." ACM Transactions of Database Systems 14.1 (1989): 1-27.

[Gadi88] Gadia, S. K. and C. Yeung. "A Generalized Model for a Relational Temporal Database." Proceedings of the 1988 ACM SIGMOD international conference on Management of Data Chicago, Illinois 1988. 251-259.

[Gass93] Gassner, P., G. M. Lohman, K. B. Schiefer, and Y. Wang. "Query Optimization in the IBM DB2 Family." Bulletin of the Technical Committee on Data Engineering 16.4 (1993): 4-17.

[Garg86] Garg, A. K., and C. C. Gotlieb. "Order-Preserving Key Transformations." ACM Transactions on Database Systems 11.2 (1986): 213-234.

[Gora95] Goralwalla, I. A., A. U. Tansel and M. T. Ozsu. "Experimenting with Temporal Relational Databases." Proceedings of the fourth international conference on Information and knowledge management Baltimore, Maryland 1995. 296 - 303.

[Grae93] Graefe, G. "Options in Physical Database Design." ACM SIGMOD Record 22.3 (1993): 76-83.

[Grae93a] Graefe, G. "Query Evaluation Techniques for Large Databases." ACM Computing Surveys 25.2 (1993): 73-170.

[Grae93b] Graefe, G., and W. J. McKenna. "The Volcano Optimizer Generator: Extensibility and Efficient Search." Proceedings. Ninth International Conference on Data Engineering Portland, Oregon 1993. 209-218.

[Gran03] Grandi, F., F. Mandreoli, P. Tiberio and M. Bergonzini. "A Temporal Data Model and Management System for Normative Texts in XML Format." Proceedings of the 5th ACM international workshop on Web Information and Data Management New Orleans, Louisiana 2003. 29-36.

[Gray94] Gray, J. N., R. A. Lorie, G. R. Putzolu, and I. L. Traiger. "Granularity of Locks and Degree of Consistency in a Shared Data Base." Readings in Database Systems. Ed. M. Stonebraker. San Mateo: Morgan Kaufmann Publishers, 1994. 181-208.

[Grot98] Groth, R. Data Mining: A Hands-On Approach for Business Professionals. Upper Saddle River: Prentice Hall, 1998.

[Grun02] Gruninger, M., and J. Lee. "Ontology Applications and Design." Communications of the ACM 45.2 (2002): 39-41.

المنارة للاستشارات
www.manaraa.com

[Gulu02] Gulutzan, P., and T. Pelzer. SQL Performance Tuning. Reading: Addison-Wesley, 2002.

[Hada96] Hadaegh, A. R. and K. Barker. "Value-serializability and an Architecture for Managing Transactions in Multiversion Objectbase Systems" Third International Workshop on Advances in Databases and Information Systems Moscow, USSR 1996. 1-12.

[Hada02] Hadaegh, A. R. and S. Ehikioya. "The Role of Reconciliation in Retrieving Historical Objects in a Multi-Version Object-Based System" Proceedings of the 2002 Canadian Conference on Electrical & Computer Engineering Winnipeg, Man., Canada 2002. 1482-1486.

[Haer78] Haerder, T. "Implementing a Generalized Access Path Structure for a Relational Database System." ACM Transactions on Database Systems 3.3 (1978): 285-298.

[Haer94] Haerder, T., and A. Reuter. "Priciples of Transaction-Oriented Database Recovery." Readings in Database Systems. Ed. M. Stonebraker. San Mateo: Morgan Kaufmann Publishers, 1994. 227-242.

[Hain03] Hainaut, J. L. "Research in Database Engineering at the University of Namur." ACM SIGMOD Record 32.4 (2003): 124-128.

[Han01] Han, J., and M. Kamber Data Mining: Concepts and Techniques. San Francisco: Morgan Kaufmann, 2001.

[Held78] Held, G., and H. Morgan ed. "B-Trees Re-examined." Communications of the ACM 21.2 (1978): 139-143.

[Herb65] Herbert, F. Dune Philadelphia, PA: Chilton Book Company, 1965.

[Hern95] Hernandez, M. A. and S. J. Stolfo. "The Merge/Purge Problem for Large Databases." Proceedings of the 1995 ACM SIGMOD international conference on Management of Data San Jose, CA 1995. 127-138.

[Ho04] Ho, P. T. "B+ - Tree & B - Tree." 2004 <http://www.cs.sjsu.edu/faculty/lee/cs157/class_presentation_Btree.ppt#256,1,B+ - Tree & B - Tree>.

[Hoch98] Hochin, T., and T. Tsuji. "A Storage Structure for Graph-Oriented Databases Using an Array of Element Types." Proceedings of the 8th Great Lakes Symposium on VLSI Lafayette, LA 1998. 452-457.

[Horn03] Horn, J. "Pros and cons of MySQL Table Types." Database Developer (2003). 15 Jun 2003 <http://www.developer.com/db/article.php/2235521>.

[Horr03] Horrocks, I., and P. F. Patel-Schneider. "Three Theses of Representation in the Semantic Web." Proceedings of the Twelfth International World Wide Web Conference Budapest, Hungary 2003. 39-47.

المنارة للاستشارات
www.manaraa.com

[Hugh97] Hughes, C., and T. Hughes. Object-Oriented Multithreading Using C++. New York: John Wiley & Sons Inc., 1997.

[Hunt04] Hunter, J., J. Drennan and S. Little. "Realizing the Hydrogen Economy through Semantic Web Technologies." Intelligent Systems, IEEE 19.1 (2004):40-47.

[Ioan96] Ioannidis, Y. E. "Query Optimization." ACM Computing Surveys (CSUR) 28.1 (1996):121-123.

[Ioan97] Ioannidis, Y. E., R. T. Ng, K. Shim, and T. Sellis. "Parametric query optimization." VLDB Journal 6 (1997):132-151.

[Isaa93] Isaac, D. "Hierarchical Storage Management for Relational Databases." Proceedings Twelfth IEEE Symposium on Mass Storage Systems 121 (1993): 139-144.

[IBM05] "Rational Software." 2005 <http://www-306.ibm.com/software/rational/>.

[Jann95] Jannink, J. "Implementing Deletion in B+-Trees." ACM SIGMOD Record 24.1 (1995): 33-38.

[Jea98] Jea, K. F., H. B. Feng, Y. R. Yau, S. K. Chen, and J. C. Dai. "A Difference-based Model for OODBMS." Proceedings. Asia Pacific Software Engineering Conference Taipei, Taiwan 1998. 369-376.

[Jens92] Jensen, C. S., J. Clifford, S. K. Gadia, A.Segev and R. T. Snodgrass. "A Glossary of Temporal Database Concepts." ACM SIGMOD Record 21.3 (1992): 35-43.

[Jeon98] Jeon, S. H., and S. H. Noh. "A Database Disk Buffer Management Algorithm Based on Prefetching." Proceedings of the seventh international conference on Information and knowledge management Bethesda, MD 1998. 167-174.

[John95] Johnson, S. C. "Yacc: Yet Another Compiler-Compiler." Technical Symposium on Computer Science Education Nashville, TN 1995.

[Jone98] Jones, D.M., T. J. Bench-Capon and P. Visser. "Methodologies for Ontology Development." Proc. IT&KNOWS Conference, XV IFIP World Computer Congress Budapest, August 1998.

[Jong90] Jonge, W., and A. Schiif. "Concurrent Access to B-trees." PARBASE-90 (1990) 312-320.

[Karl99] Karlsson, J. S., and M. L. Kersten. "Omega-Storage: A Self Organizing Multi-Attribute Storage Technique for Very Large Main Memories." The National Research Institute for Mathematics and Computer Science in the Netherlands (1999)

[Katz90] Katz, R. H. "Toward a Unified Framework for Verion Modeling in Engineering Databases." ACM Computing Surveys 22.4 (1990): 375-409.

المنارة للاستشارات
www.manaraa.com

[Kear89] Kearns, J. P., and S. DeFazio. "Diversity in Database Reference Behavior." Performance Evaluation Review 17.1 (1989): 11-19.

[Keeh74] Keehn, D. G., and J. O. Lacy. "VSAM data set design parameters." IBM System Journal 13.3 (1974): 186-212.

[Khan04] Kahn, L., D. Mcleod, and E, Hovy. "Retrieval Effectiveness of an Onology-based Model for Information Selection." VLDB Journal 13 (2004): 71-85

[Kim88] Kim, Kyongsok, and Geneva G. Belford. "The Interactions Between operating System Paging Algorithms and Database Buffering Algorithms." Proceedings of the 1988 ACM Annual Conference on Computer Science Atlanta, Georgia 1988: 602-607

[Kim01] Kim, S. W., and H. S. Won. "Batch Construction of B+-Trees." Proceedings of the 2001 ACM Symposium on Applied Computing Las Vegas, NV 2001: 231-235

[Knut97] Knuth, D. E. The Art of Computer Programming. 2nd ed. Reading: Addison-Wesley, 1997.

[Koss00] Kossman, D. and K. Stocker. "Iterative Dynamic Programming: A New Class of Query Optimization Algorithms." ACM Transactions on Database Systems 25.1 (2000): 43-82.

[Krus89] Kruse, R. Programming with Data Structures: Pascal Version. Englewood Cliffs: Prentice Hall, 1989.

[Kuma03] Kumar, S., and S. Srinivasa. "A database for storage and fast retrieval of structure data: a demonstration." IEEE Proceedings. 19th International Conference on Data Engineering. Bangalore, India: March 2003. 789-791.

[Kung94] Kung, H. T., and J. T. Robinson. "On Optimistic Methods for Concurrency Control." Readings in Database Systems. Ed. M. Stonebraker. San Mateo: Morgan Kaufmann Publishers, 1994. 209-215.

[Lank91] Lanka, S., and E. Mays. "Fully Persistent B+-Trees." Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data 20.2 (1991): 426-435.

[Lawr04] Lawrence, R. "Query Optimization." Lecture Notes University of Iowa 2004 <http://www.cs.uiowa.edu/~rlawrenc/teaching/244/Notes/244_All_Notes_by6.pdf>.

[Lee01] Lee, C., C. Shih, and Y. Chen. "A graph-theoretic model for optimizing queries involving methods." VLDB Journal 9 (2001):327-343.

المنارة للاستشارات

www.manaraa.com

[Lee04] Lee, J. "Database Archiving: A Critical Component of Information Lifecycle Management." Database Journal (2004). 04 Jun 2004 <http://www.databasejournal.com/sqletc/print.php/3340301>.

[Lehm81] Lehman, P. L., and S. B. Yao. "Efficient Locking for Concurrent Operations on B-Trees." ACM Transactions on Database Systems 6.4 (1981): 650-670.

[Lent04] Lentz, A. "MySQL Storage Engine Architecture." Online posting. 02 Apr 2004. MySQL Tech Resources. 16 Apr 2004. <hoot://dev.mysql.com/tech-resources/articles/storage-engine>.

[Lesk90] Lesk, M. E., and E. Schmidt, A. "Lex - A Lexical Analyzer Generator." Unix Vol II: Research System Philadelphia, PA W. B. Saunders Company 1990. 375-387.

[Lian03] Liang, V. C., and C. J. J. Paredis. "A Port Ontology for automated Model Composition." Proceedings of the Winter simulation Conference 2003. 613-622.

[Lim96] Lim, E. P., J. Srivastrava, and S. Shekhar. "An Evidential Reasoning Approach to Attribute Value Conflict Resolution in Database Integration." IEEE Transactions on Knowledge and Data Engineering 8.5 (1996): 707-723.

[Litw81] Litwin, W. "Trie Hashing." Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data Ann Arbor, Michigan 1981. 19-29.

[Lohm88] Lohman, G. M. "Grammar-like Functional Rules for Representing Query Optimization Alternatives." ACM International Conference on Management of Data Chicago, IL 1988. 18-27.

[Lome90] Lomet, D. and B. Salzberg. "The Performance of a Multiversion Access Method." Proceedings of the 1990 ACM SIGMOD international conference on Management of data Atlantic City, New Jersey. 1990. 353 - 363.

[Lome94] Lomet, D. B., and B. Salzberg. "The hB-Tree: A Multiattribute Indexing Method with Good Guaranteed Performance." Readings in Database Systems. Ed. M. Stonebraker. San Mateo: Morgan Kaufmann Publishers, 1994. 136-152.

[Lome01] Lomet, D. "The Evolution of Effective B-tree Page Organization and Techniques: A Personal Account." ACM SIGMOD Record 30.3 (2001): 28-32

[Lu96] Lu J, P. Barclay, J. Kennedy. "On Temporal Versioning in Object Oriented Databases" MoBIS'96 Modelling Business Information Systems Cottbus, Germany, 1996.

المنارة للاستشارات

www.manaraa.com

[Maek87] Maekawa, M., A. E. Oldehoeft, and R. R. Oldehoeft. Operating Systems : Advanced Concepts. Menlo Park: Benjamin Cummings, 1987.

[Mael95] Maelbrancke, R., and H. Olivie. "Optimizing Jan Jannink's Implementation of B+-Tree Deletion." ACM SIGMOD Record 24.3 (1995): 5-7.

[Magk02] Magkanaraki, A., G. Karvounarakis, T. T. Anh, V. Christophides, and D. Plexousakis. "Ontology Storage and Querying." Foundation for Reasearch and Technology, Hellas Institute of Computer Science No. 308 (2002).

[Mani02] Mani, M., and D. Lee. "XML to Relational Conversion using Theory of Regular Tree Grammars." Proceedings of VLDB Hong Kong: 2002. 81-103.

[Marc83] March, S. T. "Techniques for Structuring Database Records." ACM Computing Surveys 15.1 (1983): 45-79.

[Mart02] Martinez, M., J. Dernaime and P. dela Fuente. "A Method for the Dynamic Generation of Virtual Versions of Evolving Documents." Proceedings of the 2002 ACM Symposium on Applied Computing (SAC) Madrid, Spain. 2002. 476-482.

[Mcke01] McKearney, S. "B+-Trees." Advanced Databases (2001).

[Meye96] Meyers, S. More Effective C++: 35 New Ways to Improve You Programs and Designs. Reading: Addison-Wesley, 1996

[Micr00] "Microsoft SQL Server 2000." 2000 <http://www.microsoft.com/sql/default.mspx>.

[Micr03] "Microsoft SharePoint Server 2003." 2003 <http://www.microsoft.com/office/sharepoint/prodinfo/default.mspx>.

[Mond85] Mond, Y., and Y. Raz. "Concurrency Control in B+ Tree Databases Using Preparatory Operations." Proceedings of VLDB Stockholm: 1985. 331-334.

[Morr68] Morris, R. "Scatter Storage Technologies." Communications of the ACM 11.1 (1968): 38-44.

[Much97] Muchnick, S. Advanced Compiler Design Implementation. San Francisco: Morgan Kaufmann, 1997.

[Muss96] Musser, D. R., and A. Saini. STL Tutorial and Reference Guide. Reading: Addison-Wesley, 1996.

[MySq05] "MySQL 5.0." 2005 <http://dev.mysql.com/>.

[Netz01] Netz, A., S. Chaudhuri, U. Fayyad, and J. Bernhardt. "Integrating Data Mining with SQL Databases: OLE DB for Data Mining." Proceedings.

المنارة للاستشارات

www.manaraa.com

17th International Conference on Data Engineering Heidelberg, Germany 2001. 379-387.

[Neub99] Neubauer, P. "B-Trees: Balanced Tree Data Structures." 1999. 12 Jul 2004 <http://www.bluerwhite.org/btree/>.

[Omel04] Omelayenko, B. "Machine Learning for Ontology Learning." 01 2000. Vrije University. 10 Feb 2004 <www.cs.vu.nl/~borys/papers/JSS10.pdf>.

[ONei93] O'Neil, E. J., P. E. O'Neil, and G. Weikum. "The LRU-K Page Replacement Algorithm for Database Disk Buffering." Proceedings of the 1993 ACM SIGMOD international conference on Management of data Washington, D.C. 1993. 297-306.

[Ooi01] Ooi, B. C., and K. L. Tan. "B-trees: Bearing Fruits of All Kinds." Thirteenth Australian Database Conference Melborne, Australia: 2001. 13-20.

[Orac05] "Oracle Database 10g." 2005 <http://www.oracle.com/database/index.html>.

[Oual95] Oualline, S. Practical C++ Programming. Sebastopol: O'Reilly & Associates, 1995.

[Pare99] Parents, C., S. Spaccapietra and E. Zimanyi. "Spatio-Temporal Conceptual Models: Data Structures + Space + Time." Proceedings of the 7th ACM international symposium on Advances in geographic information systems Kansas City, Missouri: 1999. 26-33.

[Paul02] Paul, S., N. Gautam, R. Balint. Preparing and Mining Data with Microsoft SQL Server 2000. Redmond: Microsoft Press, 2002.

[Pere03] Perez, A. G., M. Fernandez-Lopez, and O. Corcho. Ontological Engineering. London: Springer-Verlag, 2003.

[Phil01] Phillips, J., and B. G. Buchanan. "Ontology-Guided Knowledge Discovery in Databases." K-CAP'01. Victoria, British Columbia, Canada: 2001. 123-130.

[Piat00] Piatetsky-Shapiro, G. "Knowledge Discovery in Databases: 10 Years After." SIGKDD Explorations 1.2 (2000): 59-61.

[Piss93] Pissinou, N., K. Makki and Y. Yesha. "On Temporal Modeling in the Context of Object Databases." ACM SIGMOD Record 22.3 (1993): 8-15.

[Piss94] Pissinou, N., K. Makki and E. K. Park. "Towards a Framework for Integrating Multilevel Secure Models and Temporal Data Models." Proceedings of the third international conference on Information and Knowledge Management Gaithersburg, MD 1994: 280-287.

[Plai99] Plaisant, C. M. Venkatraman, K. Ngamkajornwiwat, R. Barth, B. Harberts, and W. Feng. "Refining Query Previews Techniques for Data

www.manaraa.com

with Multivalued Attributes: The Case of NASA EOSDIS." Proceedings of the IEEE Forum on Research and Technology Advances in Digital Libraries 1999. 50-59.

[Poll96] Pollari-Malmi, K., E. Soisalon-Soininen, and T. Ylonen. "Concurrency Control in B-Trees with Batch Updates" IEEE Transactions on Knowledge and Data Engineering 8.6 (1996): 975-984.

[Post05] PostgreSQL. "PostgreSQL." 2005. <http://www.postgresql.org/>.

[Rals03] Ralston, A., E. D. Reilly and D. Hemmendinger. Encyclopedia of Computer Science. 4th ed. Chichester, West Sussex, England: Wiley, 2003.

[Rao00] Rao, J., and K. A. Ross. "Making B+-Trees Cache Conscious in Main Memory." Proceedings of the 2000 ACM SIGMOD international conference on Management of Data Dallas, TX 2000: 475-486.

[Rama03] Ramakrishnan, R. and J. Gehrke. Database Management Systems. 3rd ed. New York: McGraw-Hill, 2003.

[Rash00] Rashid, A. "A Database Evolution Approach for Object-Oriented Databases" Proceedings of the DEXA 2000. Volume 1873 of Lecture Notes in Computer Science, SpringerVerlag 2000: 125-134.

[Raso01] Rasolofo, Y., F. Abbai and J. Savoy. "Approaches to Collection Selection and Results Merging for Distributed Information Retrieval." Proceedings of the tenth international conference on Information and knowledge management Atlanta, Georgia 2001. 191-198.

[Risc04] Risch, T. "Introduction to Object-Oriented and Object-Relational Databases." 2004. <http://www.ida.liu.se/~dbt/OH/oointro.pdf>.

[Rodd02] Roddick, J. F., and M. Spiliopoulou. "A Survey of Temporal Knowledge Discovery Paradigms and Methods." IEEE Transactions on Knowledge and Data Engineering 14.4 (2002) 750-767.

[Rodr73] Rodriguez-Rosell, J., and J. P. Dupuy. "The Design, Implementation, and Evaluation of a working Set dispatcher." Communications of the ACM B. Randell ed. 16.4 (1973): 247-253.

[Sacc86] Sacco, G. M., and M. Schkolnick "Buffer Management in Relational Database Systems." ACM Transactions on Database Systems 11.4 (1986): 473-498.

[Sche03] Scheffczyk, J., P. Rodig, U. M. Borgoff and L. Schmitz "Consistent Document Engineering." Proceedings of the 2003 ACM Symposium on Document Engineering Grenoble, France 140-149.

[Sedg98] Sedgewick, R. Algorithms in C++. 3rd ed. Boston: Addison-Wesley, 1998.

المنارة للاستشارات
www.manaraa.com

[Seid01] Seidman, C. Data Mining with Microsoft SQL Server. Redmond: Microsoft Press, 2001.

[Seli79] Selinger, P. G., M. M. Astraham, D. D. Chamberlin, R. A. Lories, and T. G. Price. "Access Path Selection in a Relational Database Management System." Proceedings of the ACM SIGMOD International Conference on the Management of Data. Aberdeen, Scotland: 1979. 23-34.

[Senk73] Senko, M. E., E. B. Altman, M. M. Astrahan, and P. L. Fehder. "Data Structures and Accessing in Data-base Systems: Evolution of Information Systems." IBM Systems Journal 1 (1973): 30-93.

[Seve76] Severance, D. G., and R. Duhne. "A Practitioner's Guide to Addressing Algorithms." Communications of the ACM H. Morgan ed. 19.6 (1976): 314-326

[Seve77] Severance, D. G., and J. V. Carlis. "A Practical Approach to Selecting Record Access Paths." Computing Surveys 9.4 (1977): 259-272.

[Shas88] Shasha, D., and N. Goodman. "Concurrent Search Structure Algorithms." ACM Transactions on Database Systems 13.1 (1988): 53-90.

[Sher76] Sherman, S. W., and R. S. Brice. "Performance of a Database Manager in a Virtual Memory System." ACM Transactions on Database Systems 1.4 (1976): 317-343.

[Silb96] Silberschatz, A., H. F. Korth, S. Sudarshan Database System Concepts. 3rd ed. New York: McGraw-Hill, 1996.

[Silb98] Silberschatz, A., P. B. Galvin Operating System Concepts. 5th ed. Reading, Massachusetts: Addison-Wesley, 1998.

[Sing97] Singhal, S., and A. J. Smith. "Analysis of Locking Behavior in Three Real Database Systems." The VLDB Journal 6. (1997): 40-52.

[Sliv01] Slivinskas, G., C. S. Jensen, and R. T. Snodgrass. "Adaptable Query Optimization and Evaluation in Temporal Middleware." Santa Barbara, California: 2001. 127-138.

[Smit78] Smith, A. J. "Sequentiality and Prefetching in Database Systems." ACM Transactions on Database Systems 3.3 (1978): 223-247.

[Snod85] Snodgrass, R., and I. Ahn. "A taxonomy of Time Databases." Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data Austin, TX: 1985. 236-246.

[Soul03] Soules, C. A. N., G. R. Goodson, J. D. Strunk and G. R. Ganger. "Metadata Efficiency in Versioning File Systems." 2nd USENIX Conference on File and Storage Technologies San Francisco, CA: 2003.

[Spee93] Speegle, D., and M. J. Donahoo. "Using Statistical Sampling for Query optimization in Heterogeneous Library Information Systems." Proceedings

المنارة للاستشارات
www.manaraa.com

of the 1993 ACM conference on Computer Science Indianapolis, Indiana 1993. 475-482.

[Spyn02] Spyns, P., R. Meersman, and M. Jarrar. "Data Modelling versus Ontology Engineering." ACM SIGMOD Record 31.4 (2002): 12-17.

[Srin93] Srinivasan, V., and M. J. Carey. "Performance of B+ Tree Concurrency Control Algorithms." VLDB Journal 2 (1993): 361-406.

[Ston76] Stonebraker, M., E. Wong, P. Kreps. "The Design and Implementation of INGRES." ACM Transactions on Database Systems 1.3 (1976): 189-222.

[Ston81] Stonebraker, M. "Operating System Support for Database Management." Communications of the ACM 24.7 (1981): 412-418.

[Ston87] Stonebraker, M. J. Anton, and E. Hanson. "Extending a Database System with Procedures." ACM Transactions on Database Systems 12.3 (1987): 350-376.

[Ston94] Stonebraker, M. "The Design of the POSTGRES Storage System." Readings in Database Systems. Ed. M. Stonebraker. San Mateo: Morgan Kaufmann Publishers, 1994. 278-289.

[Ston94a] Stonebraker, M. "Managing Persistent Objects in a Multi-Level Store." Readings in Database Systems. Ed. M. Stonebraker. San Mateo: Morgan Kaufmann Publishers, 1994. 408-417.

[Ston94b] Stonebraker, M. "Relational Implementation Techniques." Readings in Database Systems. Ed. M. Stonebraker. San Mateo: Morgan Kaufmann Publishers, 1994. 77-83.

[Ston98] Stonebraker, M. and J. L. Hellerstein. Readings in Database Systems 3rd edition, Michael Stonebraker ed., Morgan Kaufmann Publishers, 1998

[Sury01] Suryanto, H., and P. Compton. "Discovery of Ontologies from Knowledge Bases." K-CAP'01. Victoria, British Columbia, Canada: 2001. 171-178.

[Tans93] Tansel, A; J. Clifford, S. Gadia, S. Jajodia, A. Segev; and R. Snodgrass. Temporal Databases : Theory,Design, and Implementation Redwood City, CA : Benjamin/Cummings, 1993

[Tele05] "TeleLogic DOORS." 2005 <http://www.telelogic.com/products/doorsers/doors/>.

[Thur00] Thuraisingham, B. "A Primer for Understanding and Applying Data Mining." IEEE IT Pro January, February 2000: 28-31.

[Thur00a] Thuraisingham, B., and M. G. Ceruti. "Understanding Data Mining and Applying it to Command, Control, Communications and Intelligence Environments." Computer Software and Applications Conference Taipei, Taiwan 2000.171-175.

المنارة للاستشارات

www.manaraa.com

[Torp98] Torp, K., C. S. Jensen, and R. T. Snodgrass. "Effective Timestamping in Databases." VLDB Journal 8 (1998): 267-288.

[Trai82] Traiger, I. L. "Virtual Memory Management for Database Systems." Proceedings of the 1982 ACM SIGOPS Operating System Review New York, N.Y. 1982. 26-48.

[Trid04] "InterchangeSE." 2004 <http://www.interchangese.com>.

[Tuck04] Tucker, A. B. Computer Science Handbook. 2nd ed. Boca Raton, Florida: CRC Press LLCC, 2004.

[VanB03] Van Brakel, M. "NTFS Compression and Sparse Files." Delphi Magaznine 98 (2003).

[Varz98] Varzi, A. C. "Basic Problems of Mereotopology." Formal Ontology in Information Systems. Ed. N. Guarino. Amsterdam: IOS Press, 1998. 29-38.

[Vasw04] Vaswani, V. MySQL: The Compete Reference. New York: McGraw-Hill/Osborne, 2004.

[Vite05] "Vitech CORE 5." 2005 <http://www.vtcorp.com/overview.html>.

[Wagn73] Wagner, R. E. "Indexing Design Considerations." IBM Systems Journal No. 4 (1973): 351-367.

[Wahl01] Wahlstrom, K., and J. F. Roddick. "On the Impact of Knowledge Discovery and Data Mining." 2nd Australian Institute of Computer Ethics Conference. Canberra, Australia: 2001. 22-27.

[Wegn89] Wegner, L. M., and J. I. Teuhola. "The External Heapsort." IEEE Transactions on Software Engineering 15.7 (1989): 917-925.

[Wern01] "Inside the SQL Query Optimizer." Progress Worldwide Exchange 2001, Washington D.C. 2001

[West92] West. L. W. "Postorder B-tree Construction." ACM Southeast Regional Conference Raleigh, NC 1992. 449-452.

[Whan90] Whang, K., and R. Krishnamurthy. "Query Optimization in a Memory-Resident Domain Relational Calculus Database System." ACM Transactions on Database Systems 15.1 (1990): 67-95.

[Wide03] Widenius, M. "The Anatomy of The MySQL Query Optimizer." MySQL AB 2003 <http://www.mysql.com>

[Widm99] Widmann, N., and P. Baumann. "Performance Evaluation of Multidimensional Array Storage Techniques in Databases." Database Engineering and Applications, 1999. IDEAS '99. International Symposium Proceedings Montreal, Quebec Canada 1999. 385-389.

المنارة للاستشارات
www.manaraa.com

[Wirt76] Wirth, N. Algorithms + Data Structures = Programs Englewood Cliffs, NJ: Prentice-Hall, 1976.

[Wirt96] Wirth, N. Compiler Construction. Reading: Addison-Wesley, 1996.

[Witt05] Witten, I. H. and E. Frank. Data Mining - Practical Machine Learning Tools and Techniques. San Francisco: Morgan-Kaufmann, 2005.

[Yann95] Yannakakis, M. "Perspectives on Database Theory." Proceedings. 36th Annual Symposium on Foundations of Computer Science Milwaukee, WI 1995. 224-246.

[Yao76] Yao, S. B. "Modeling and Performance Evaluation of Physical Data Base Structures." Proceedings of the Annual ACM/CSC-ER Conference 1976. 303-309.

[Yao77] Yao, S. B. "Approximating Block Accesses in Database Organizations." Communications of the ACM 20.4 (1977): 260-261.

[Yao78] Yao, S. B., and D. DeJong. "Evaluation of Database Access Paths." Proceedings of the 1978 ACM SIGMOD International Conference on Management of Data Austin, Texas 1978. 66-77.

[Yao79] Yao, S. B. "Optimization of Query Evaluation Algorithms." ACM Transactions on Database Systems 4.2 (1979): 133-155.

[Zawo04] Zawodny, J. and D. J. Balling. High Performance MySQL. Sebasopol, CA: O'Reilly, 2004.

[Zhou03] Zhou, J., and K. A. Ross. "A Multi-resolution Block Storage Model for Database Design." Proceedings of the Seventh International Database Engineering and Applications Symposium 2003. 22-31.

[Zhu98] Zhu, Q., B. Dunkel, N. Soparkar, S. Chen, B. Sciefer, and T. Lai. "A Piggyback Method to Collect Statistics for Query Optimization in Database Management Systems." Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research Toronto, Ontario, Canada 1998. 25-31.

المنارة للاستشارات

www.manaraa.com

Appendix A – Dataset Descriptions

A.1 Introduction

The technological challenges of a work of this size and complexity can be overcome with ingenuity and patience[1]. However, there is one problem that is seemingly very difficult to solve – locating datasets of sufficient size and complexity to demonstrate the benefits of one's work. Unfortunately, the data used in this work is protected by legal and legislative protections that prohibit disclosure. As a result, much of the data used in the experiments were either borrowed from publicly available sources or contrived from said sources. This appendix describes the datasets used and describes the methods, implementation details, and the challenges and solutions concerning the data used for the experiments in this work.

A.2 Datasets Used

This section presents the details of each of the datasets used. This includes their origin, any customizations made to the data for use in the experiments, and a sample of each. A number of datasets were used in the development and implementation of the

[1] Also known as the bull headed s tubbornness required of successful graduate students to carry out the task to its logical conclusion.

المنارة للاستشارات
www.manaraa.com

technologies in this work. The following table lists a description, origin, and size of each dataset used.

Dataset	Description	Source	Size (approx)
Dialysis Facility Compare	Information for research on medicare facilities.	http://www.medicare.gov/Download/DownloadDB.asp	100,000 entities
Federal Aviation Admini stration Airmen Certification	Law requires the FAA to release names, addresses, and ratings information for all airmen after the 120th day following the date of enactment. This file contains the names, addresses, and certificate information of those airmen who did not respond to indicate that they wished to withhold their address information.	http://www.faa.gov/licenses_certificates/airmen_certification/releasable_airmen_download/	128,000 entities
Home Health Compare	Information for research on medicare facilities.	http://www.medicare.gov/Download/DownloadDB.asp	4,500 entities
Mammalian Reproductive Genetics (MRG)	This database consists of genes and literature related to mammalian reproduction.	http://mrg.genetics.washington.edu/	200,000 entities
Prescription Drug Assistance Program	Information for research on medicare facilities.	http://www.medicare.gov/Download/DownloadDB.asp	35,000 entities
Sakila	This sample DB is based on a DVD rental shop model. There are two stores and two employees. The stores are open 24 hours and customers can shop at either store. The 2 employees work at both stores but each is manager of one store. Payment is due at time of rental, with a $1 per day late fee.	http://www.mysql.com	5,000 entities
UCI Adults	This data set contains weighted census data extracted from the 1994 and 1995 current population surveys conducted by the U.S. Census Bureau.	http://www.census.gov/	32,000 entities

Table A-1: Datasets Used

www.manaraa.com

A.3 Experimental Data

This section presents the datasets used for each experiment. Subsections below describe each experiment arranged by chapter.

A.3.1 Chapter 4 and 5

The dataset used in these chapters was generated from one table from each of three of the above datasets. These datasets were; Sakila, UCI Adults, and the MRG databases. Each dataset was implemented as a MyISAM and InnoDB table in MySQL and the ALV system. To simulate the utility of the clustered version store, the data was modified to include a grouping attribute that represented the key association supported in the ALV system. To simulate random insertion, the datasets were sorted using a random value generated per row prior to insertion. All datasets were inserted in the same order. The following table depicts the statistics for the tables used.

	Customer	Adults	ORF
ALV	1,484,200	77,588,297	145,520,114
MyISAM	24,576	1,213,440	6,953,984
% Diff	60.39	63.94	20.93
Size (Rows)	599	32,561	201,053
#Blocks	369	19,395	36,403
Avg Att/Blk	1.6233	1.6788	5.523

Table A-2: Statistics of Datasets Used

A.3.1.1 Customer Table from Sakila Database

The customer table has 599 rows with a record size of approximately 100 bytes. The following depicts the CREATE statements used to create the dataset and a sample of some of the INSERT statements used to populate the dataset.

المنارة للاستشارات

www.manaraa.com

A.3.1.1.1 ALV Statements

```
CREATE ALV TABLE customer_alv KEY ALV_KEY integer ATTRIBUTE (
  ATTR varchar(15),
  Value integer,
  phone varchar(50),
  active integer
);

INSERT ALV INTO customer_alv FOR Customer (ALV_KEY, ATTR, Value, phone,
active) VALUES ( 99, 'Customer', 293, '96604821070', 1);
INSERT ALV INTO customer_alv FOR Customer (ALV_KEY, ATTR, Value, phone,
active) VALUES ( 412, 'Customer', 179, '866092335135', 1);
INSERT ALV INTO customer_alv FOR Customer (ALV_KEY, ATTR, Value, phone,
active) VALUES ( 206, 'Customer', 349, '53912826864', 1);
INSERT ALV INTO customer_alv FOR Customer (ALV_KEY, ATTR, Value, phone,
active) VALUES ( 343, 'Customer', 22, '161968374323', 1);
INSERT ALV INTO customer_alv FOR Customer (ALV_KEY, ATTR, Value, phone,
active) VALUES ( 222, 'Customer', 186, '760171523969', 1);
```

A.3.1.1.2 MyISAM Statements

```
CREATE TABLE customer (
  ALV_KEY integer,
  ATTR varchar(15),
  Value integer,
  phone varchar(50),
  active integer
) TYPE=MyISAM ROW_FORMAT = FIXED;

INSERT INTO customer VALUES (99, 'Customer', 293, '96604821070', 1);
INSERT INTO customer VALUES (412, 'Customer', 179, '866092335135', 1);
INSERT INTO customer VALUES (206, 'Customer', 349, '53912826864', 1);
INSERT INTO customer VALUES (343, 'Customer', 22, '161968374323', 1);
INSERT INTO customer VALUES (222, 'Customer', 186, '760171523969', 1);
```

The row format variable was used to prevent packing. Packing is a form of data compression that is effective in reducing the amount of disk space used to store data. This feature was not relevant for use in the experiments.

المنارة للاستشارات
www.manaraa.com

A3.1.1.3 InnoDB Statements

```
CREATE TABLE customer_i (
  ALV_KEY integer,
  ATTR varchar(15),
  Value integer,
  phone varchar(50),
  active integer
) TYPE=INNODB;

INSERT INTO customer_i VALUES (99, 'Customer', 293, '96604821070', 1);
INSERT INTO customer_i VALUES (412,'Customer', 179, '866092335135', 1);
INSERT INTO customer_i VALUES (206, 'Customer', 349, '53912826864', 1);
INSERT INTO customer_i VALUES (343, 'Customer', 22, '161968374323', 1);
INSERT INTO customer_i VALUES (222,'Customer', 186, '760171523969', 1);
```

A.3.1.2 Adults Table from UCI Adults Database

The adults table has 32,561 rows with a record size of approximately 85 bytes. The following depicts the CREATE statements used to create the dataset and a sample of some of the INSERT statements used to populate the dataset.

A.3.1.2.1 ALV Statements

```
CREATE ALV TABLE adults_alv KEY ALV_KEY integer ATTRIBUTE (
  `ALV_KEY` INTEGER,
  `ATTR` varchar(30),
  `VALUE` integer,
  `Education` varchar(30),
  `Class` varchar(10)
);

INSERT ALV INTO adults_alv FOR ADULT (ALV_KEY, ATTR, VALUE, EDUCATION,
CLASS) VALUES (20175,'Adult',77516,' Bachelors',' <=50K');
INSERT ALV INTO adults_alv FOR ADULT (ALV_KEY, ATTR, VALUE, EDUCATION,
CLASS) VALUES (4761,'Adult',83311,' Bachelors',' <=50K');
INSERT ALV INTO adults_alv FOR ADULT (ALV_KEY, ATTR, VALUE, EDUCATION,
CLASS) VALUES (21569,'Adult',215646,' HS-grad',' <=50K');
INSERT ALV INTO adults_alv FOR ADULT (ALV_KEY, ATTR, VALUE, EDUCATION,
CLASS) VALUES (14917,'Adult',234721,' 11th',' <=50K');
INSERT ALV INTO adults_alv FOR ADULT (ALV_KEY, ATTR, VALUE, EDUCATION,
CLASS) VALUES (8652,'Adult',338409,' Bachelors',' <=50K');
```

المنارة للاستشارات
www.manaraa.com

A.3.1.2.2 MyISAM Statements

```
CREATE TABLE adults (
  `ALV_KEY` INTEGER,
  `ATTR` varchar(30),
  `VALUE` integer,
  `Education` varchar(30),
  `Class` varchar(10)
) TYPE=MyISAM ROW_FORMAT = FIXED;

INSERT INTO `adults` VALUES (20175,'Adult',77516,' Bachelors','
<=50K');
INSERT INTO `ADULTS` VALUES (4761,'Adult',83311,' Bachelors',' <=50K');
INSERT INTO `ADULTS` VALUES (21569,'Adult',215646,' HS-grad',' <=50K');
INSERT INTO `ADULTS` VALUES (14917,'Adult',234721,' 11th',' <=50K');
INSERT INTO `ADULTS` VALUES (8652,'Adult',338409,' Bachelors','
<=50K');
```

The row format variable was used to prevent packing. Packing is a form of data compression that is effective in reducing the amount of disk space used to store data. This feature was not relevant for use in the experiments.

A3.1.2.3 InnoDB Statements

```
CREATE TABLE adults_i (
  `ALV_KEY` INTEGER,
  `ATTR` varchar(30),
  `VALUE` integer,
  `Education` varchar(30),
  `Class` varchar(10)
) TYPE=INNODB;

INSERT INTO `adults_i` VALUES (20175,'Adult',77516,' Bachelors','
<=50K');
INSERT INTO `ADULTS_I` VALUES (4761,'Adult',83311,' Bachelors','
<=50K');
INSERT INTO `ADULTS_I` VALUES (21569,'Adult',215646,' HS-grad','
<=50K');
INSERT INTO `ADULTS_I` VALUES (14917,'Adult',234721,' 11th',' <=50K');
INSERT INTO `ADULTS_I` VALUES (8652,'Adult',338409,' Bachelors','
<=50K');
```

المنارة للاستشارات
www.manaraa.com

A.3.1.3 ORF Table from MRG Database

The ORF table has 201,053 rows with a record size of approximately 100 bytes. The following depicts the CREATE statements used to create the dataset and a sample of some of the INSERT statements used to populate the dataset.

A.3.1.3.1 ALV Statements

```
CREATE ALV TABLE `orf_alv` KEY ALV_KEY INTEGER ATTRIBUTE (
  `ATTR` varchar(12),
  `Value` varchar(100),
  `GeneCount` int(11)
);

INSERT ALV INTO orf_alv FOR Zygote (`ALV_KEY`, `ATTR`, `Value`,
`GeneCount`) VALUES (1,'Zygote','MGI:2180337',25);
INSERT ALV INTO orf_alv FOR Zygote (`ALV_KEY`, `ATTR`, `Value`,
`GeneCount`) VALUES (1,'Zygote','Zar1',26);
INSERT ALV INTO orf_alv FOR Zygote (`ALV_KEY`, `ATTR`, `Value`,
`GeneCount`) VALUES (1,'Zygote','zygote arrest 1',20);
INSERT ALV INTO orf_alv FOR Zygote (`ALV_KEY`, `ATTR`, `Value`,
`GeneCount`) VALUES (18,'Zygote','03B03R',26);
INSERT ALV INTO orf_alv FOR Zygote (`ALV_KEY`, `ATTR`, `Value`,
`GeneCount`) VALUES (18,'Zygote','G48145',27);
```

A.3.1.3.2 MyISAM Statements

```
CREATE TABLE `orf` (
  `ALV_KEY` int(11) default NULL,
  `ATTR` varchar(12) default 'Zygote',
  `Value` varchar(100) default NULL,
  `GeneCount` int(11) default NULL
) ENGINE=MyISAM ROW_FORMAT = FIXED;

INSERT INTO `orf` (`ALV_KEY`, `ATTR`, `Value`, `GeneCount`) VALUES
(1,'Zygote','MGI:2180337',25);
INSERT INTO `orf` (`ALV_KEY`, `ATTR`, `Value`, `GeneCount`) VALUES
(1,'Zygote','Zar1',26);
INSERT INTO `orf` (`ALV_KEY`, `ATTR`, `Value`, `GeneCount`) VALUES
(1,'Zygote','zygote arrest 1',20);
INSERT INTO `orf` (`ALV_KEY`, `ATTR`, `Value`, `GeneCount`) VALUES
(18,'Zygote','03B03R',26);
INSERT INTO `orf` (`ALV_KEY`, `ATTR`, `Value`, `GeneCount`) VALUES
(18,'Zygote','G48145',27);
```

المنارة للاستشارات
www.manaraa.com

The row format variable was used to prevent packing. Packing is a form of data compression that is effective in reducing the amount of disk space used to store data. This feature was not relevant for use in the experiments.

A3.1.3.3 InnoDB Statements

```
CREATE TABLE `orf_i` (
  `ALV_KEY` int(11) default NULL,
  `ATTR` varchar(12) default 'Zygote',
  `Value` varchar(100) default NULL,
  `GeneCount` int(11) default NULL
) TYPE=INNODB;

INSERT INTO `orf_i` (`ALV_KEY`, `ATTR`, `Value`, `GeneCount`) VALUES
(1,'Zygote','MGI:2180337',25);
INSERT INTO `orf_i` (`ALV_KEY`, `ATTR`, `Value`, `GeneCount`) VALUES
(1,'Zygote','Zar1',26);
INSERT INTO `orf_i` (`ALV_KEY`, `ATTR`, `Value`, `GeneCount`) VALUES
(1,'Zygote','zygote arrest 1',20);
INSERT INTO `orf_i` (`ALV_KEY`, `ATTR`, `Value`, `GeneCount`) VALUES
(18,'Zygote','03B03R',26);
INSERT INTO `orf_i` (`ALV_KEY`, `ATTR`, `Value`, `GeneCount`) VALUES
(18,'Zygote','G48145',27);
```

A.3.2 Chapter 6

The dataset used in this chapter was taken from the Airmen database. The foreign pilot table was reconstructed as a versioned table representing clusters of the original data. These clusters were formed by assigning an arbitrary grouping value (the ALV_KEY) to each row. A single attribute was used to simulate a versioned table (Pilot). There were a total of 85 attribute versions (rows) created. Each row is approximately 180 bytes. The following depicts the CREATE statement used to create the dataset and a sample of some of the INSERT statements used to populate the dataset.

المنارة للاستشارات
www.manaraa.com

This dataset was used solely for the testing of the query processing technologies. A copy of this table was constructed in a MySQL table for use in comparing the performance of the ALV query processor to that of MySQL. The following depicts the CREATE statement used to create the dataset and a sample of some of the INSERT statements used to populate the dataset.

A.3.2.1 ALV Statements

```
CREATE ALV TABLE `forcert_alv` KEY ALV_KEY varchar(8) ATTRIBUTE(
  `ATTR` varchar(12),
  `FIRST_NAME` varchar(30),
  `LAST_NAME` varchar(30),
  `CERTIFICATE_TYPE` varchar(1),
  `CERTIFICATE_LEVEL` varchar(1),
  `CERTIFICATE_EXP` varchar(8),
  `RATINGS` varchar(99)
);

INSERT INTO `forcert_alv` FOR Pilot (ALV_KEY,  FIRST_NAME, LAST_NAME,
CERTIFICATE_TYPE, CERTIFICATE_LEVEL, CERTIFICATE_EXP, RATINGS) VALUES
('A1817736','Pilot', 'DAVID GEORGE', 'STAVELEY', 'F', '', '06302005',
'F/ASE');
INSERT INTO `forcert_alv` FOR Pilot (ALV_KEY,  FIRST_NAME, LAST_NAME,
CERTIFICATE_TYPE, CERTIFICATE_LEVEL, CERTIFICATE_EXP, RATINGS) VALUES
('A1817736','DAVID GEORGE', 'STAVELEY', 'G', '' ,'' ,'G/INST');
INSERT INTO `forcert_alv` FOR Pilot (ALV_KEY,  FIRST_NAME, LAST_NAME,
CERTIFICATE_TYPE, CERTIFICATE_LEVEL, CERTIFICATE_EXP, RATINGS) VALUES
('A1817736','DAVID GEORGE', 'STAVELEY', 'M', '', '', 'M/AIRFR');
INSERT INTO `forcert_alv` FOR Pilot (ALV_KEY,  FIRST_NAME, LAST_NAME,
CERTIFICATE_TYPE, CERTIFICATE_LEVEL, CERTIFICATE_EXP, RATINGS) VALUES
('A1817802','ATHANASIOS CONST', 'STAVROPOULOS', 'Y', 'Y', '',
'Y/ASEL');
INSERT INTO `forcert_alv` FOR Pilot (ALV_KEY,  FIRST_NAME, LAST_NAME,
CERTIFICATE_TYPE, CERTIFICATE_LEVEL, CERTIFICATE_EXP, RATINGS) VALUES
('A1817809','PETER STELIOU', 'STAVRINIDES', 'Y', 'Y', '', 'Y/ASEL');
```

A.3.2.2 MyISAM Statements

```
CREATE TABLE forcert (
  UNIQUE_NO char(8) default NULL,
  FIRST_NAME char(30) default NULL,
  LAST_NAME char(30) default NULL,
  CERTIFICATE_TYPE char(1) default NULL,
```

www.manaraa.com

```
  CERTIFICATE_LEVEL char(1) default NULL,
  CERTIFICATE_EXP char(8) default NULL,
  RATINGS char(99) default NULL
) TYPE=MyISAM ROW_FORMAT = FIXED;

INSERT INTO forcert VALUES ('UNIQUE N', 'FIRST NAME', 'LAST NAME', 'T',
'L', 'EXPIRE D', 'RATING1');
INSERT INTO forcert VALUES ('A0000053', 'JESUS ALBERTO', 'DIAZ', 'P',
'A', '', 'A/AMEL');
INSERT INTO forcert VALUES ('A0000053', 'JESUS ALBERTO', 'DIAZ', 'Y',
'Z', '', 'Z/ASEL');
INSERT INTO forcert VALUES ('A0000130', 'JUERGEN F P', 'HOPPE', 'Y',
'Y', '', 'Y/ASEL');
INSERT INTO forcert VALUES ('A0000133', 'JAMES HARRY', 'SEVERNS', 'G',
'', '', 'G/ADV');
```

The row format variable was again used to prevent packing.

A.3.3 Chapter 7

The dataset used in this chapter was more difficult to create than the ones used in previous chapters. Since the subject of Chapter 7 is data mining, the dataset used must be of sufficient complexity and uniformity to test the usefulness of a data mining algorithm. As a result, a program was created to generate the dataset.

This program was written to use the FAA Airmen database as a basis for creating a dataset with versioning information. The names of the pilots from the original dataset were changed using a name generator drawn from common surnames and male first names. The pool of surnames totaled 88,800 names and the pool of first names totaled 1,128 names. The version information generated were attribute versions for the alias and phone number attributes from the original dataset. The attribute versions used are shown in the following SQL statement.

This dataset was used solely for the testing of the application of data mining technologies to evaluate and quantify the version information. The following depicts the

المنارة للاستشارات
www.manaraa.com

CREATE statement used to create the dataset and a sample of some of the INSERT statements used to populate the dataset.

```
CREATE ALV TABLE `airmen_alv` KEY ALV_KEY varchar(8) ATTRIBUTE(
  `ATTR` varchar(12),
  `VALUE` varchar(40),
  `RELIABILITY` varchar(20),
  `CONFIDENCE` varchar(20),
  `SOURCE` varchar(10),
  `SENSITIVITY` varchar(20)
);
```

The metadata lists used to generate this dataset is shown in the table below.

Metadata	Values
Source	{NID, SGC, PFJ, JPF, Newswire, TV, Radio, Article, Reference}
Reliability	{Unknown, Low, Medium, High}
Confidence	{Unconfirmed, Confirmed, Validated, Fact}
Sensitivity[2]	{Unclassified, Confidential, Secret}

Table A-2: Metadata Values Used

The following describes the algorithm used to generate the dataset:

```
for each entity

      select attributes for versioning

      for each attribute selected

            generate 0-10 attribute versions

                  for each attribute version generated

                        generate a random value from lists

                        generate meta data values from list

                  next attribute version

      next attribute
```

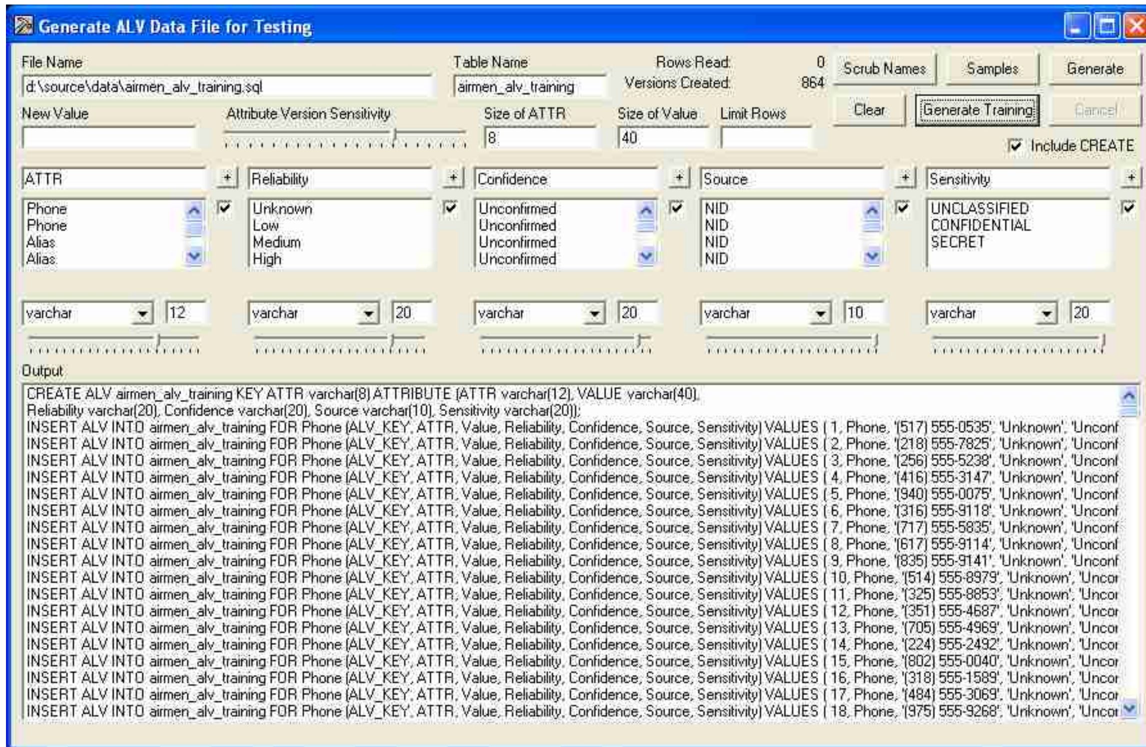
[2] Despite the inclusion of a sensitivity or classification attribute, none of the data used in this work is classified. All data is either public accessib le or manufactured to represent similar data.

المنارة للاستشارات

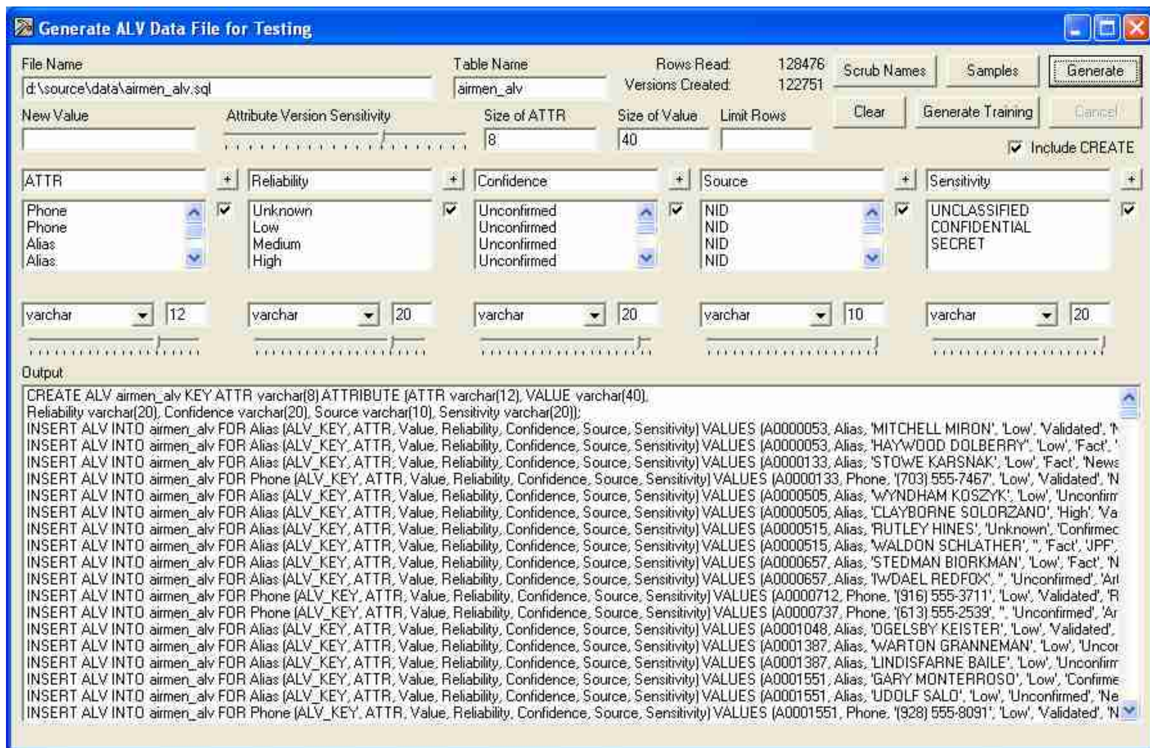
www.manaraa.com

```
next entity
```

The resulting dataset contained approximately 122,000 attribute versions.



المنارة للاستشارات
www.manaraa.com



A.4 Challenges and Solutions

The Attribute-Level Versioning (ALV) system provides a versioning mechanism for use in a traditional relational database system. The data that ALV stores is in the form of an attribute, its value, and associated metadata (called an attribute version). Since ALV is an entirely new concept and its implementation unique, no datasets are available with which to test or conduct experiments. Thus, the datasets had to be contrived.

These contrivances were designed to provide realistic properties to the data. A small program was used to create the datasets using as input the original dataset as described above. This program used sets of terms and possible values generating a random attribute version for a random number of iterations for a random set of entities. While this method may generate data that is bound by the sets from which the values are

المنارة للاستشارات
www.manaraa.com

drawn, it does provide a reasonable manufactured dataset with which to demonstrate the ALV technologies.

In fact, observations of existing real-world datasets show that the distribution of duplicate values and collisions during dataset merging is an approximately random event. In the live data, it is not an uncommon event to have a few entities with a range of 2 to 12 attribute versions for 4 or fewer attributes. Those datasets that are truly duplicates containing alternative views of all attributes for all entities are rare. Thus, the mechanisms by which the contrived data was formed depicts the types and seeding factor that are expected in a production use of the ALV system.

Another area of concern was the creation of a mechanism by which the ALV data can be exported to SQL statements. An application called ALVDump was created to generate a text file containing INSERT statements for all of the data in the ALV data store. The application will also generate the CREATE statement as an option. This application is useful in that it permits database professionals to export all of the data to a script, delete the version store, and then recreate it using the script. This has the side benefit that the version store is created with all of the attribute chains in order with unused blocks are removed.

A.5 Conclusion

While finding sufficient data was a very challenging problem, the data presented in this appendix describes existing and created datasets that meet the minimal requirements for demonstrating the benefits of this work.

المنارة للاستشارات

www.manaraa.com

Appendix B – Modifying MySQL for use with ALV

B.1 Introduction

This section describes the methods, implementation details, and the challenges and solutions in integrating ALV technologies into the MySQL server.

B.2 Development Approach

The purpose of the supporting project for this work was to create a fully functional versioning system implemented in an operational database management system.

Goals

- Minimize changes to the MySQL code base so that the operation of the original MySQL functionality is not impeded.
- Integrate the ALV functionality in such a way that it becomes a seamless execution when mixing normal and ALV commands.
- Exploit the best theories and invent advanced techniques for solving the versioning problem.

المنارة للاستشارات
www.manaraa.com

- Provide a robust set of operations that permit administrators to administer the database server using traditional database maintenance operations, e.g., backup, restore, dump, etc.

B.3 Modifications to MySQL Source Files

The primary goal of the integration of the ALV technologies with the MySQL database core was to minimize the number of changes necessary to the MySQL code and to prohibit modification of any of the MySQL functions. Fortunately, only seven source files were changed, and many of the changes were simply minor additions to the code. The most significant changes were those to the sql_yacc.yy file. A complete listing of all of the files changed and a brief description of the changes are shown in the table below.

Source File	Line*	Description
lex.h	0052	This section identifies the internal code symbols and values for the ALV tokens.
Mysql_priv.h	0026	A reference to the ALVExecute class.
mysqld.cpp	0017	`#include "alv_manager.h"`
	3377	This section calls the ALV_Manager and sets the path to the ALV data stores.
	6856	This section adds the ALV version number to the MySQL version number.
Sql_class.h	1208	This section defines the ALV_Manager pointer for each thread.
Sql_lex.h	0054	This section captures the enumerations for the ALV command tokens.
Sql_parse.cpp	0024	`#include "ALV_sql_parse.h"`
	0191	Begin ALV transaction hook.
	1172	End ALV transaction hook for the case where the session times out or disconnects unexpectedly.
	1401	End ALV transaction hook.
	1412	End ALV transaction hook.
	1420	Begin ALV transaction hook.
	1431	End ALV transaction hook.

المنارة للاستشارات
www.manaraa.com

Source File	Line*	Description
	3387	This section is the main modification or plugin for the ALV code. It establishes the connection for activating the parsed ALV commands.
Sql_yacc.yy	0175	This section defines the tokens for the ALV commands.
	1126	This section captures (parses) the `CREATE ALV TABLE` statement.
	2549	This section captures (parses) the subparts of the CREATE ALV TABLE statement.
	3762	This section captures (parses) the `RESTORE ALV` statement.
	3786	This section captures (parses) the `BACKUP ALV` statement.
	4050	This section captures (parses) the `SELECT ALV` statement.
	4140	This section captures (parses) the sub parts of the `SELECT ALV` statement.
	5619	This section captures (parses) the where clause for the `SELECT ALV` statement.
	5952	This section captures (parses) the `DROP ALV` statement.
	6089	This section captures (parses) the `INSERT ALV` statement.
	6260	This section captures (parses) the `UPDATE ALV` statement.
	6340	This section captures (parses) the `DELETE ALV` statement.
	6372	This section captures (parses) the subparts of the `DELETE ALV` statement.
	6458	This section captures (parses) the `SHOW ALV` statements.
	6805	This section captures (parses) the `EXPLAIN (DESCRIBE) ALV` statements.

Table B-1: Changes to MySQL Source Files
*Line numbers are approximate

B.3.1 ALV Technologies Source Files

The source files that make up the ALV technologies can be categorized as follows:

- Data – these files contain classes, structures, and methods that manipulate the data in the internal data representation. They provide an abstract layer over the lower-level file I/O classes.

المنارة للاستشارات
www.manaraa.com

- Execution – these files contain classes and methods that perform query operations.
- File I/O – these files contain classes and methods used to perform low-level file input and output operations.
- Utility – these files contain general methods and structures for common or widely reused features.

The table below lists all of the source files including the category and brief description of each.

Source File†	Category	Description
ALV.cpp	Utility	Includes general utility functions, error message handling.
ALV_Manager.cpp	Execution	Manages threading controlled access to the main ALV data storage files and the table-level locks for the ALV system.
ALV_sql_parse.cpp	Execution	Receives control from the parser providing the execution code for the SQL_{ALV} commands.
ALVDataFile.cpp	File I/O	Manages the low-level access for the ALV database tables.
ALVExecute.cpp	Execution	Executes the query tree.
ALVRecordFile.cpp	File I/O	Responsible for managing the low-level access for the ALV records within blocks.
ALVString.h	Utility	Standard C-style string encapsulates a char pointer and deletes it upon destruction.
Attribute.cpp	Data	Abstracts an attribute for use in internal representation of ALV data.
bptBase.h	Indexing	Serves as the base class for the B+Tree indexes and their ALV derivatives.
bptBlockMgr.cpp	Indexing	Implements an in-memory buffer management system.
bptDataFile.cpp	File I/O	Manages the low-level access for the ALV indexes within blocks.
bptFreeBlockQueue.cpp	Indexing	Implements an in-memory free block queue for

المنارة للاستشارات
www.manaraa.com

Source File†	Category	Description
		deleted blocks of data. No .h file.
bptHash.cpp	Indexing	Implements an in-memory hash table for storing keys in the multiple key indexes.
bptKeyAndValueTypes.h	Indexing	Defines various simple data types to be key and value template parameters for bptIndex. No .cpp file.
bptNode.cpp	Indexing	Structures an MBlock as a B+ tree node.
Expression.cpp	Data	Contains the expressions for a query.
Hash.cpp	Indexing	Implements a HashTable, using Quadratic Probing when a hash clash occurs.
MetaData.cpp	Data	Manages the meta data name, id, and data type used in ALVRecordFile.
QueryTree.cpp	Execution	Contains the internal representation of the query to be executed. Provides methods for optimizing and forming and inspecting the query tree.
Queue.h	Utility	Contains a template for a FIFO queue. Uses the standard template declaration and supports operations for Put, Get, Empty, QueueSize, and Print.
Relation.cpp	Data	Encapsulates the relational notion of a table.
Tuple.cpp	Data	Encapsulates the relational notion of a tuple. Supports operations on the tuple for manipulating the data and order of the attributes.
ValueBucket.cpp	Data	Provides a hashed storage area for storing and comparing tuple values based on type.

Table B-2: List of ALV Source Files
†All cpp files have corresponding h files except w here noted.

B.3.2 Challenges and Solutions

Aside from the technological challenges presented in previous chapters, there were many challenges in integrating the ALV technologies into an operational platform. Perhaps the most challenging was modifying the MySQL parser to recognize the new SQL_{ALV} commands. Although not precisely a complex or new implementation language, modification of the yacc files required careful attention to the original developers' intent.

www.manaraa.com

The biggest hurdle was where to change the parser in order to capture the SQL_{ALV} commands. The solution involved placing the new commands at the top of each of the parser command definitions. This permitted intercepting the flow of the parser in order to redirect the query processor to execute the ALV commands directly.

The most frequent and least trivial challenge of all was keeping up with the constant changes in the MySQL code base. Since MySQL is an open source product and the ALV technologies were developed without participating in the evolution of the MySQL code base, the frequency of upgrades was unpredictable. In order to keep up with the feature changes, the integration of the ALV technologies required reinserting the modifications to the seven source files with each upgrade of the MySQL code base. To avoid repeating this process, it was decided to upgrade the MySQL code base on major and minor updates only, *e.g.*, 4.0 to 4.1 or 4.X to 5.0. However, should the ALV technology become mainstream and a permanent feature of MySQL, the ALV source code and all technologies will require merging with the community-wide source repository. There are three possibilities available for the sponsor of this work to consider in order to achieve this goal; 1) continue to maintain the ALV technologies apart from the MySQL code base and perform integration with each major or minor release, 2) honor the entirety of the GNU license and turn over the ALV technologies to MySQL AB for incorporation into the main code base, or 3) purchase extended support from MySQL AB

المنارة للاستشارات
www.manaraa.com

and provide the ALV technologies as a sole propriety act[1]. Naturally, the author will strive to encourage the second option with due enthusiasm.

The challenge that presented the least amount of effort but required a considerable amount of time was examining the MySQL code base and discovering the meaning, layout, and use of the various internal data representations. As stated in several MySQL documents, the code suffers from "Genius Intuition[2]" which renders the code indecipherable to all but the most informed and advanced C++ programmers. This challenge was eventually overcome by sheer determination and many, many visits to MySQL blogs and message forums.

The challenge that presented the most limiting constraints was the choice of the operating system a basis for the solution. Although mandated by the project sponsors, implementing the ALV technologies on the Microsoft Windows operating system platform presented a number of significant problems that resulted in extra work and some limited functionality. MySQL supports the Windows operating system by providing both binary installation and execution files as well as the source code for compilation. Most developer forums dedicated to MySQL warn of these problems and attempt to steer experimenters away from using the Windows platform for development. This due to the unfortunate fact that many of the utilities created to manage the MySQL source code are simply unavailable or do not have an analogous operation. The implementation and

---

[1] In other words, pay someone to maintain the code but not share it with the community. This is an option that the sponsor is very likely to take.

[2] A self imposed description by the code authors themselves. Th is phenomena isn't new. It is an all -to-common occurrence among the C and C++ community. It seems that there are two classes of developers who produce this type of code. Those that do so out of ignorance, arrogance, or intolerance and those that do so in the pursuit of refined code. Fortunately, the developers at MySQL AB are of the latter variety.

www.manaraa.com

integration of the ALV technologies was achieved with great success despite the limitations of the Windows platform. However, one must consider how much more productive[3] this project would have been if developed on the MySQL native/preferred platform, Linux.

B.4 Conclusion

The MySQL system has proven to be difficult to learn and troublesome to diagnose when things go awry. However, it is clear that once one has mastered the intricacies of the MySQL genius-inspired code, the system is very accommodating and has the promise of being perhaps the first and best platform for experimental database work. The ALV system works well within the confines of the MySQL server and has shown no signs of violating any of the native MySQL features or performance. Overall, the integration of the experimental versioning code has been very successful.

[3] The reduction in gnashing of teeth and follicle depletion would have been reward enough to warrant developing this project on Linux.

المنارة للاستشارات
www.manaraa.com

Vita

Charles Andrew Bell was born on October 12, 1962 in Fredericksburg, Virginia, and is an American citizen. He graduated from Stafford High School, Stafford, Virginia in 1981. He received his Bachelor of Science in computer Science from Virginia Commonwealth University in 1995. He received a Master of Science in Computer Science from Virginia Commonwealth University in 1997. He currently works for the federal government conducting research in emerging technologies.

المنارة للاستشارات
www.manaraa.com