Object-relational concepts in database management systems

Gerald Anleitner
Seminar on applications and concepts of database systems

12th February 1998

Contents

1 Introduction

During the last decade, the object-oriented approach got more and more accepted as *the* way to model and to program in computer industry and science. Thus, it seems only natural that the wave of OO influences the world of databases as well.
The relational model dominates the world of databases for quite a long time now. But OO creates movement in this scene: Object-oriented databases appear and the big vendors like Informix or Oracle sell their products as *object-relational* database management systems. The following is an introduction to this new "style" of database management systems.

2 What we want

The already mentioned rise of object-oriented methodology and technology, the growing demand and usage of multimedia, compound documents and computer-added design applications creates more and more pressure on database systems to support these new technologies. As (CLC95) state, if relational databases can adapt to this new challenge, they might well stay in business.

- Multimedia and related technologies have formed new datatypes that are currently not represented in relational database management systems. It would be desirable to extend an existing DBMS with new base datatypes. And this means: Not only adding a new type like `external file`, but adding real support for these datatypes (e.g. methods that send video streams to clients, that compare images, operators to reason on the datatype etc.).

- An object-oriented approach requires in particular the possibility of defining new, complex datatypes and, in addition, the possibility of establishing a hierarchical order on the defined datatypes (inheritance etc.).

- The database administrator or programmer must be able to create new procedures within the database to allow comparison or other operations on the datatypes. This enables the processing of more complex transactions, triggers, adds the possibility to create an intelligent database.

So what can/must be done to make RDBMSs ready for the challenge? There are two main approaches:

- Adding object extensions to the standard relational model. This is the approach this paper focuses on.

- Adding an object layer on top of the relational database.
  This object layer (another word might be "wrapper") simulates an object-oriented interface on top of a relational database. It might as well be seen as some kind of middleware.

datatype	comment	max value
varchar2	variable size character string	4000 bytes
nvarchar2	variable size, natioanl character set	4000 bytes
char	fixed length character set	2000 bytes
nchar	fixed length national character set	2000 bytes
number	integer or real	[38],[-84 - 127]
date	type for date and time	up to 31/12/4712
long	character string	$2^{31} - 1$ bytes
raw	binary data	2000 bytes
long raw	alpha-numeric	2 GB
rowid	unique address of a row in its table	
clob	character large object	4 GB
nclob	national ...	4 GB
blob	binary large object	4 GB
bfile	locator to external filesystem	4 GB

Table 1: Oracle8's base datatypes

In the following sections, I will provide a deeper introduction to the above mentioned requirements for object-relational database management systems (base datatype definition, complex datatypes, inheritance, programming) together with examples from Oracle8 and Postgres95.

3 Object-relational database technology: a dummies' guide

The term "object-relational" does not seem to have an exact definition. The notion of an object-relational database management system (subsequently: ORDBMS) in this paper is based on text in (SM96) and (SKS97), enriched with theoretical thoughts and implementation approaches which the new capabilities of ORDBMSs might require.

3.1 Base type extension

3.1.1 The desire

According to (SM96), one new feature of ORDBMSs is the possibility of extending the database system with new base types. First, have a look at the base datatypes of Oracle8, which claims to be an ORDBMS (see table 1). In (SQL92), the corresponding operators for each datatype are precisley defined and there is no possibility to add new base types to the system[1]. Oracle8 extends the standard by providing types like "external". The impossibility of defining new base types is a obvious lack of flexibility.

[1] Actually, (SQL92) defines even less base datatypes, but normaly, DBMS vendors do not care too much about the standard...

To allow the definition of datatypes, we first have to make sure what we mean with this, resp. what the user has to do to define a new datatype.

datatype specification A name has to be assigned to the datatype, together with information about the storage requirements of the datatypes (handling of dynamic size?!). It might be necessary to let the user decide where instances of the type should be stored: Within the database system, or by using some kind of external resource.

operators and methods There is no use in defining new database types without any means to reason on them. What is an integer without a sum operator, without comparison operators, what would SQL be without the greater-equals operator? Thus, there must be a way to define new operations on the defined datatypes.

implementation and handling This requires the possibility of dynamically linking new operators to the database as it would not be adequate to shut down and start up, or relink, the database each time a new base type is added. Furthermore, to assure efficient and fast work with new datatypes, the programmer should be able to create access structures for new datatypes.
This shows already a dilemma of the base datatype story: Lots of possible security wholes, as dynamic linking can add problems when new code is not "clean", external storage mediums which are not controlled by the database can cause problems, not to think about many errors that the code of the programmed operators can comprise.
It is quite interesting that Oracle8 allows the usage of external procedures. A little introduction of this functionality can be found in the addendum.

3.1.2 Postgres95's implementation of base datatype extension

As Oracle8 does not allow the creation of base datatypes, I show an example using Postgres95 here (see (PG95)).
First of all, C-code has to be written. The actual type definition:

```
typedef struct Complex
{
  double x;
  double y;
} Complex;
```

Then, two access methods are required, the input and output functions which determine how the datatype appears as a string:

```
Complex* complex_in(char *str)
{
  double x,y;
  Complex *result;
```

```
  if(sscanf(str," ( %lf, %lf )", &x, &y) !=2)
  {
    elog(WARN,"complex_in: error in parsing");
    return null
  }
  result = (Complex *)palloc(sizeof(Complex));
  result->x = x;
  result->y = y;
  return result;
}

char* complex_out(Complex *complex)
{
  char *result;
  if(complex==NULL) return NULL;
  result = (char*) palloc(60);
  sprintf(result, "(%g,%g)",complex->x,complex->y);
  return result;
}
```

Now the really exciting part is how to take these definitions and make them usable by a Postgres95 user:

```
create function complex_in(opaque)
  returns complex
  as '/home/postgres/basetypes/complex.so'
  language 'c';

create function complex_out(complex)
  returns opaque
  as  '/home/postgres/basetypes/complex.so'
  language 'c';

create type complex
{
  internallenght = 16,
  input = complex_in,
  output = complex_out
};
```

Finally, if the datatype `complex` is now used in a Postgres95 statement, the library is dynamically loaded into Postgres95.

3.2 A general-purpose programming language

3.2.1 Programming within a database

SQL lacks especially one property: It is not computationally complete. And even though the new SQL standard might once again cover some distance towards computational completeness, it always seems to look as a patch and SQL loses more and more of its elegance (if it ever possessed any elegance...).
Thus, a computationally complete programming language is one of the requirements for an up-to-date database system, which ORDBMSs claim to be. The next section on complex datatypes shows that the introduction of a more complex type system increases the need for a multi-purpose programming language. In general, several things have to be noticed:

Procedures Procedures are independent programming constructs which are not bound to any datatype and can be used to provide general functionality, e.g. as triggers (which requires some additonal construct that observes changes on the database), intelligent interfaces (in cases where simple views don't suffice), for creating reports, etc.

Operators, methods Operators and methods are closely bound to a datatype. An operator can be a method for the increment of some tuple in a special sense, for the comparison, addition, multiplication of datatypes etc. Methods might be used to change the internal state of an instance of a datatype.

Implementation issues It might look easiest to create an open database programming interface that allows the creation of additional modules for the database, e.g. access structures for new datatypes, interfaces to storage mediums etc. This solution brings along severe security problems and the usage of an interface like that must be considered very well. Perhaps it should only be used where the trade-off between security and performance clearly shows the need for such a grave intervention into the database system.
In all other (and probably in most) cases, it is sufficient to provide a built-in programming language like Oracle's PL/SQL. This programming language can perform any tasks like adding operators to the system, create multi-purpose triggers, all that seems to be necessary in the context of server-side programming.

3.2.2 Oracle's PL/SQL

Several reasons led Oracle to ship a oracle-intern programming language already with Version 6: When performing complex database operations, it was desired to do this without going back to the client after each SQL query. It should be possible to perform a whole complex transaction in the database (and, as this reduces traffic between server and client, client/server applications would perform better). Furthermore, it was Oracle's purpose to create a programming

language that is used within the whole framework of Oracle's products[2].
There exist several kinds of PL/SQL programmes:

- 4GL client programming (e.g. Oracle Forms, ReportWriter, Graphics etc.).
- anonymous blocks which are loaded into Oracle via query-like statements.
- programmes which are permanently stored within Oracle.
- triggers, which are activated by a DML operation.

A PL/SQL programme consists of three parts:

declaration That is the place for all kinds of declarations: Variables, constants, cursors, user-defined exceptions.

execution Here is the "real" code, loops, etc.

exceptions At least one surprise: PL/SQL includes exception handling routines to catch errors during the execution of the code.

Examples about PL/SQL will be provided in the next section, in combination with the example on complex datatypes.

3.3 Complex objects

Complex objects, another distinguishing feature of ORDBMSs, provide the following: firstly, complex types which consist of several base types or other complex types (related to C's `struct`) and, secondly, there is the concept of collection types: carry together elements of the same or even different types in a bag, set, list or whatever.
The idea of extending relational databases in this way was developed not late after E. G. Codd (see (C70)) gave birth to relational databases at the beginning of the 1970. Makinouchi (M77) was one of the first to present an idea on relations which was called "non first normal form relation"[3]. As the name already suggests, the *non first normal form* model[4] allows the violation of the first normal form principle:

> All attributes of a relation have an atomic domain.
> A domain D is atomic *iff* all elements of D are indivisible units.

Concerning ORDBMSs, the idea of NF^2 (non first normal form) is interpreted in two ways.

[2] see (S93) for more information on PL/SQL and Oracle7 in general
[3] see (C79), (SS86)
[4] see (KE97) p. 157/158, 370/371, (SKS97) p. 275-291, (KM94) p. 135-150

father	mother	children	pets
Johann	Antoinette	Jean	Ezlla
Jean	Elisabeth	{Tony,Christian}	{Yiek,Oak}

Table 2: Set-based attributes in a relation

father	mother	children	pets
Johann	Antoinette	Jean	Ezlla
Jean	Elisabeth	Tony	Yiek
Jean	Elisabeth	Christian	Yiek
Jean	Elisabeth	Tony	Oak
Jean	Elisabeth	Christian	Oak

Table 3: Set-based attributes converted into a first normal form relation

3.3.1 Set-based datatypes

An attribute of a relation can now be a set of entries (see table 2). This can easily be simplified to fit again into the first normal form relational model (see table 3).
Thus, a first approach to the definition of relations in ORDBMSs might be:

> Attributes of a relation in OR-from are atomic or set valued.

To give a more formal definition:

> A domain D is relation$_{or}$-valued (written: $\mathcal{D}^{or}$) *iff*
> $\mathcal{D}^{or} = \mathcal{D}_i$
> $\mathcal{D}^{or} = 2^{\mathcal{D}^{or}}$
> $\mathcal{D}_i$: basic domains consisting of atoms ("base types")
>
> A relation R is in OR-form (written: $\mathcal{R}^{or}$) *iff*
> $\mathcal{R}^{or} \subseteq \mathcal{D}_1^{or} \times \ldots \times \mathcal{D}_n^{or}$

Be aware: This is just "syntactic sugar", as table 2 and 3 showed. Any relational database is able to model this extended definition of domains. In addition to the way shown above, a second table can be created that holds the set elements. Then, a join is needed to e.g. get all pets that belong to the parents. The "cleaner" way is of course to analyze any multi-valued dependencies and get into normalization etc. But that is not the topic of this paper.

3.3.2 Compound datatypes

In addition to sets, a desired ability is to create new datatypes out of already existing types, the already known `struct` concepts of programming languages like C.
There are two possibilities to realize this concept:

```
create type Workstation
(
name                    varchar2(20),
software                setof(varchar2(20))
);
```

Figure 1: Definition of a complex datatype

```
create table SalesRep
(
PID                     number(8),
name                    varchar2(20),
area                    varchar2(20),
workstation             WorkStation
);
```

Figure 2: Definition of a table with a complex datatype

- Creation of new datatypes and then using this new datatypes as an attribute in a relation. Or it might be possible to create a table whose type is a complex datatype.
- Adding the possibility to create nested tables within a table.

The first approach should be preffered as it provides the ability to reuse already defined datatypes. In addition, this concept is more flexible in a number of other ways (allows inheritance, creation of operators etc.).

Let us consider an example to illustrate the idea of complex types. Firstly, a new datatype is defined (figure 1). This datatype is now used as a type in a relation in OR form (figure 2, table 4). Once again, this relation in OR form can be converted into relations which suffice the requirements of the "normal" relational model (table 5, 6).
So the definition of domains and relations in OR form from above can be extended in the following way:

A domain D is relation$_{or}$-valued (written: $\mathcal{D}^{or}$) *iff*
$\mathcal{D}^{or} = \mathcal{D}_i$
$\mathcal{D}^{or} = 2^{\mathcal{D}^{or}}$
$\mathcal{D}^{or} = \mathcal{D}_1^{or} \times \ldots \times \mathcal{D}_n^{or}$
$\mathcal{D}_i$: basic domains consisting of atoms

PID	name	area	workstation.name	workstation.software
17091974	Radek	Wisconsin	karl	{ Powerpoint, Word }
05071975	Bucharin	Oregon	rosa	{ Doom }
24121997	Joffe	Texas	paul	{ Freelance }

Table 4: Complex attributes used in the relation SalesRep

PID	name	area	CID
17091974	Radek	Wisconsin	12
05071975	Bucharin	Oregon	14
24121997	Joffe	Texas	16

Table 5: NF relation Person

CID	name	software
12	karl	Powerpoint
12	karl	Word
14	rosa	Doom
16	paul	Freelance

Table 6: NF relation WorkStation

A relation R is in OR-form (written: $\mathcal{R}^{or}$) *iff*
$\mathcal{R}^{or} \subseteq \mathcal{D}_1^{or} \times \ldots \times \mathcal{D}_n^{or}$

3.3.3 Implementation issues

Some aspects have to be considered when thinking about extending an existing DBMS to obtain ORDBMS features:

Extenstions to SQL The query, data manipulation and data definition language of a database system has to provide syntactic constructs to allow the above introduced extensions:

- The well-known dot-operator might be used to access data of a complex datatype:

```
select *
from SalesRep s
where s.workstation.name='karl';
```

- As already shown above, an operator *setof(A)* might be implemented to create a set of type A. Further extensions might be conceivable to be able to create list, arrays etc.
- Furthermore, a construct is needed to define new complex datatypes. It might look like the already used constructor of figure 1.

Operators and methods One problem remains unsolved so far: It is not possible to define operators on the newly created datatypes which is crucial when quering for those datatypes. Thus, some kind of programming language has to be added to the database. This issue was discussed in one of the previous section of this paper. In addition, a syntactic construct is needed to bind a method or an operator to a datatype.

Search structures Especially when dealing with multi-dimensional data (which can of course easily be modelled by using the approach of creating complex objects), already implemented search structures like the B-tree might no longer be optimal. Thus, it is desired to allow the implementation of more efficient multi-dimensional search structures if necessary. This rises again the questions about how to integrate such structures (linking of additional libraries, database programming language, etc.).

3.4 Oracle8's way to complex datatypes

First of all, let me cite the Oracle Server Concepts manual ((OC97A), (OC97B)), chapter 10:

> Relational database management systems (RDBMSs) are the standard tool for managing business data. They provide fast, efficient, and completely reliable access to huge amounts of data for millions of businesses around the world every day.
> The objects option makes Oracle an object-relational database management system (ORDBMS), which means that users can define additional kinds of data-specifying both the structure of the data and the ways of operating on it-and use these types within the relational model. This approach adds value to the data stored in a database. Oracle with the objects option stores structured business data in its natural form and allows applications to retrieve it that way. For that reason it works efficiently with applications developed using object-oriented programming techniques.
> Oracle's support for user-defined datatypes makes it easier for application developers to work with complex data like images, audio, and video.

3.4.1 A little bit of background

Every user of Oracle owns a so called *schema*, which has the name of the user and contains the schema objects (tables, triggers etc.). The so-called Object options now enables the user to create complex and set-valued datatypes, together with operators, views etc. within its schema.
Oracle offers other add-ons for the basic database systems, e.g. the already mentioned video option or a package especially for geographical information systems.

3.4.2 An example database

In this section, I will present examples to demonstrate the features of Oracle8 in the realm of complex datatypes. Linguistics, especially syntax and lexical research provides a nice environment for an example database[5]. Linguistic theories like HPSG and DATR are systems that strongly base on type theory and inheritance. Here, I will concentrate on the modelling of a lexicon component

[5]The example is partially based on ideas of (PS87).

which serves to store words with their different, linguistically relevant features into an ORDBMS.
A type `lexical entry` consists of several subtypes which correspond to different linguistic areas: Information should be stored about pronunciation (phonological/phonetic information), the meaning (semantic information) and about the category (syntactic information) of the word.

phonology/phonetics Every word consists of a ordered set of so-called phonemes representing the pronunciation of the word. Phonemes consist of a set of features describing them, thus they can be nicely modelled by a complex datatype (in general, there are of course more than these two phonemes)[6].

```
create or replace type Phoneme as object
(
  dental number(1),
  labial number(1)
);
/
```

The pronunciation of a word is modelled as a set of phonemes. Oracle8 provides two means to model sets:

- Firstly, there are so-called `VARRAYs`. This is an ordered set of data elements. All elements are of the same type, each element has an index which corresponds to the elements position in the array. Oracle8's arrays are called `VARRAYs` because they're allowed to be of variable size, but even though, a maximum size has to be specified when declaring the `VARRAY`.
- Secondly, nested tables can be used to model a set: It is an unordered set of data elements, all of the same type.

Here, I use a `VARRAY` to model the set[7]:

```
create or replace type PhonemeSet as VARRAY(25) of Phoneme;
/
```

This shows one shortcoming: An intermediate datatype has to be created when using sets, it is not possible to use a syntax like:

```
[..]
Pronunciation setOf(Phoneme),
[..]
```

[6] filename: crtPhoneme.sql
[7] crtPhonemeSet.sql

Together with this datatypes, the type `Phonetics` can be defined[8]:

```
create or replace type Phonetics as object
(
  sound bfile,
  pronunciation PhonemeSet
);
/
```

bfile is a datatype that represents an external file. A directory has to be made available to Oracle8 where files are then stored and handled as LOBs (Large OBjects). The actual bfile column in the database is just assigned a locator to the binary file.

semantics The type `semantics` contains information about the meaning of a word. For several purposes, it is important to know about the relation of one word to another in the context of meaning. E.g. *binocular* has definitley a closer relation to *eyeglasses* than to *fire department*. Thus, there exist so-called semantic nets which model this relation by building a net above all entries of a lexicon. This is modelled by a datatype `Reference`[9]:

```
create or replace type Reference as object
(
  weight number(3),
  wref REF Word
);
/
```

REFs provide a means to access elements of another table directly, which was done by using foreign keys in the relational model. Oracle assigns a unique, immutable identifier to every row object and lets the user use this reference as the *REF* built-in datatype. Several states of a reference have to be distinguished:

scoped REFs References that are constrained to only point to a specified table are called *scoped* REFs.

dangling REFs This is just the same as already known from programming languages: The row a *REF* is pointing to can be deleted and thus become unavailable. Then, the reference is *dangling*. Oracle8 provides a method `is_dangling` to allow testing for dangling references.

[8] crtPhonetics.sql
[9] crtReference.sql

dereferencing REFs Accessing the object referred to by the REF is called *dereferencing* the reference. Oracle8 provides a `deref` operator, that returns a null object if the reference is dangling. In addition, Oracle8 offers implicit dereferencing as well:

```
x.manager.name
y.name, where y=deref(x.manager)
```

There is one important remark on REFs: They can only be used and applied if working with so-called object tables. This is just to remember for now. Another part of this text will be concerned with object tables.

Once again, a set is needed to represent the relations to other words. This time, it does not have to be ordered, thus we can use Oracle8's second approach to set-valued attributes which is called *nested table*[10].

```
create or replace type ReferenceSet as table of Reference;
/
```

Then, the final subtype for semantic content can be defined. This type contains a function that serves for comparing instances of the type semantics. A second function was created which returns the word that is strongest related to the actual word[11].

```
create or replace type Semantics as object
(
  content varchar2(30),
  references ReferenceSet,

  member function refs return varchar2,
  pragma restrict_references
 (refs,wnds,wnps),
  map member function ret_value return varchar2,
  pragma restrict_references
 (ret_value, wnds, wnps, rnps, rnds)
);
/
```

syntax Finally, syntactic information about the word has to be stored. As Oracle8 does not provide a construct for inheritance, I only concentrate on nouns here.
The definition of the syntax subtype[12]:

```
create or replace type Syntax as object
```

[10] crtReferenceSet.sql
[11] crtSemantics.sql
[12] crtSyntax.sql

```
    (
      plural_end varchar2(5),
      genitiv_end varchar2(5)
    );
    /
```

So finally, the datatype word can be created[13]:

```
create or replace type Word as object
(
  syn Syntax,
  sem Semantics,
  phon Phonetics,
  letters varchar2(30)
);
/
```

Oracle8 offers the possibility to define comparison operators, resp. *map methods* and *order methods*. Map methods are quite simple: They use Oracle8's ability to compare built-in types by making the user define a method that provides datatypes that Oracle8 can compare. In this case, I just return the actual word, that is represented by an instance of the datatype. The other method (order methods) requires more work: This methods use their own, programmed logic to compare two instances of the datatype and then might return -1, 0, +1 to indicate that one instance is smaller, equal or bigger than the other. If none of the two possible methods are define, Oracle cannot determine enequality. At least, there is a way Oracle8 attempts to determine equality: This is done just by comparing the attributes of the datatype. In addition, I implemented a second PL/SQL method, just to show a little bit of the functionality of the language[14].

```
create or replace type body Semantics as
  member function refs return varchar2
  is
    ref_table ReferenceSet;
    counter number;
    best number:=0;
    best_word varchar2(20) := 'no connections';
  begin
    if self.references.count>0 then
      for counter in 1..self.references.count loop
        if best<self.references(counter).weight then
          best:=self.references(counter).weight;
  select l.letters into best_word
  from lexicon l
```

[13] crtWord.sql
[14] crtSemanticsBody.sql

```
  where ref(l)=self.references(counter).wref;
        end if;
      end loop;
    end if;
    return best_word;
  end refs;

  map member function ret_value return varchar2 is
  begin
    return content;
  end ret_value;
end;
/
```

Below is the script used for creating the whole set of types. First of all, it deletes any old definitions of types, then it calls the above defined scripts to create the types and their bodies. Finally, it creates a log table, inserts a first entry and finally prints out Oracle8's error table to show the user any errors which occured during the type definition steps[15].

```
@dropAll

@phonetics/crtPhoneme
@phonetics/crtPhonemeSet
@phonetics/crtPhonetics

create or replace type Word
/
@semantics/crtReference
@semantics/crtReferenceSet
@semantics/crtSemantics

@syntax/crtSyntax

@crtWord

@crtLexicon

@semantics/crtSemanticsBody

create table Lexicon_log
(
  when date,
  text varchar2(30)
);
insert into Lexicon_log values
```

[15] crtAll.sql

```
       (sysdate,'creation of object table');

select * from Lexicon_log;

select * from user_errors;

quit;
```

Firstly, an incomplete type `Word` was created, as it would cause a compilation error to create `Reference` without the type `Word` already existing.
Just the last script is missing: Normaly, a table would be created consisting of the desired types. But as REFs can only be used when working with object table, another new feature of Oracle8 has to be explained and used.
Object tables are not too surprising. With them, the user is able to create a table that just comprises one datatype, and they offer the possibility to use references. An object table is created this way[16]:

```
create table Lexicon of Word
nested table sem.references store as references_table;
```

The second line is necessary to create the nested table of the semantic subtype which contains the references to other words.
Now the newly created table has to be populated. This is one way to do it:

```
delete from lexicon;

insert into Lexicon values
(
Word(
     Syntax('-e',''),
     Semantics('FahrzeugLand',ReferenceSet()),
     Phonetics(NULL,PhonemeSet()),
     'Mercedes'
    )
);

insert into Lexicon values
(
Word(
     Syntax('-s','-s'),
     Semantics('FahrzeugLand',ReferenceSet()),
     Phonetics(NULL,PhonemeSet(
       Phoneme(1,2),Phoneme(1,2),Phoneme(2,1))),
     'BMW'
    )
);
```

[16] crtLexicon.sql

```
insert into the
(select l.sem.references
 from Lexicon l
 where l.letters='BMW'
)
select 2,ref(l)
from Lexicon l
where l.letters='Mercedes';

insert into the
(select l.sem.references
 from Lexicon l
 where l.letters='BMW'
)
select 20,ref(l)
from Lexicon l
where l.letters='BMW';

quit;
```

Especially the last SQL statement is interesting: First, the attribute of the table Lexicon is selected, in which an item should be inserted. Then, the second select decides which entry of the Lexicon qualifies. A reference to the qualifying entry is then created and is inserted together with a value denoting the strength of the semantic reference.

These are examples of some queries on the lexicon:

- This query just prints the words in the lexicon.

  ```
  select l.letters
  from lexicon l;

  quit;
  ```

- This query is a little bit more complicated. In the nested query, it selects the references of the nested table of the lexicon entry for the word 'BMW'. Then, the *where* clause selects the entries of the lexicon which are referenced by the reference entry in the nested query.

  ```
  select rt.weight,l.letters, l.sem.content
  from the (select l.sem.references
            from lexicon l
            where l.letters='BMW'
           ) rt,
       lexicon l
  where ref(l)=rt.wref;
  quit;
  ```

This is the result of the query:

```
    WEIGHT LETTERS              SEM.CONTENT
---------- -------------------- --------------------
         2 Mercedes             FahrzeugLand
        20 BMW                  FahrzeugLand
```

This would be the result, if the *where* clause of the outer query is deleted:

```
    WEIGHT LETTERS               SEM.CONTENT
---------- -------------------- --------------------
         2 Mercedes             FahrzeugLand
        20 Mercedes             FahrzeugLand
         2 BMW                  FahrzeugLand
        20 BMW                  FahrzeugLand
```

- And here is one last query. In the inner nested query it selects all semantic references of the entry of word 'BMW'. Then, from those entries, it selects all entries that have a value bigger than 10. Then it takes the references of those values and queries for the word entry (the *letters* attribute) of those references.

```
select l.letters
from lexicon l
where ref(l)=
( select r.wref
  from the (select l.sem.references
    from lexicon l
    where l.letters='BMW') r
  where r.weight>10
);
quit;
```

Now follow some little PL/SQL procedures:

- First of all: anonymous blocks. They can be seen as a special kind of transaction. The source code is sent to the Oracle server and then executed there. This is especially interesting when transactions are required which cannot be expressed in "normal" SQL. With the help of PL/SQL, those transactions can be performed completely at the server, whereas without PL/SQL, data would have to be sent between server and client to perform the parts which are not possible in SQL in a client-side application. This PL/SQL code just inserts some data into the lexicon. It is quite similar to the SQL code above that inserts values into lexicon. Just note that there is a short exception handling routine at the end of the anonymous block.

```
begin
  insert into Lexicon values
  (
    Word(
     Syntax('-s','-s'),
     Semantics('FahrzeugLand',ReferenceSet()),
     Phonetics(NULL,PhonemeSet(
       Phoneme(1,1),Phoneme(1,2),
       Phoneme(1,1),Phoneme(1,2))),
     'Auto'
    )
);

insert into the
(select l.sem.references
 from Lexicon l
 where l.letters='Auto'
)
select 14,ref(l)
from Lexicon l
where l.letters='BMW';

insert into the
(select l.sem.references
 from Lexicon l
 where l.letters='Auto'
)
select 10,ref(l)
from Lexicon l
where l.letters='Mercedes';

exception
  when others then
    rollback;
    insert into lexicon_log values
   (sysdate,'could not insert word Auto');

end;
/

commit;

select l.letters from lexicon l;
select * from lexicon_log;

quit;
```

- It is of course possible to call any methods of datatypes, this little query prints the word and its strongest reference:

```
select l.letters,l.sem.refs()
from lexicon l;
quit;
```

- And finally, a stored procedure is created with this script:

```
create or replace procedure pronounce(word varchar2)
is
  phon_set PhonemeSet;
  act_phon Phoneme;
  max_phon number;
begin
  select l.phon.pronunciation into phon_set
  from lexicon l
  where l.letters=word;
  max_phon := phon_set.count;
  act_phon := phon_set(1);
  -- send phoneme-set to speakers...
  exception
    when others then
    insert into lexicon_log values
   (sysdate,'procedure pronounce failed');
end;
/
quit;
```

  Stored procedures are executed this way:

```
execute pronounce('BMW');
```

3.5 Inheritance

Inheritance is a powerfull means for modelling. Inheritance is mainly regarded in the context of datatype inheritance, but can also be viewed as inheritance of tables with consequences on the data manipulation language.

3.5.1 Domain inheritance

Data inheritance is another extension to ORDBMSs' type system. So far, types in ORDBMSs comprise:

- built-in base types
- user-defined base types
- complex types (compositional, set-valued)

Datatypes and datatype inheritance can be seen from different view points: As already discussed in the section about complex datatypes, there is the possibility of creating a table with nested relations instead of creating datatypes that are then used in the table. This concept can be used here as well: Instead of creating datatypes and then applying inheritance, a table might be created and then another table, which inherits the properties of the first table.
The other case is what we are used to: Having datatypes, creating new ones as shown in the sections above and then adding inheritance as another way of creating new datatypes.
In both cases, there are some interesting consequences: Assuming a table hierarchy, where a table `manager` is a subtype of table `employee`. Then, when doing an update of table employee, we might want to do the update on all subtables of table employee as well. Thus, a key word for an *inheritance update* might be desirable. Something similar might be the case when thinking about datatypes and their operators. If applicable, the operators of a type should be inherited by the subtype as well, thus allowing code reuse.
As both of the above mentioned approaches to inheritance might be desired in a database context, the question on how to generalize this idea arises. The approach that (D95) uses is not to define inheritance on the basis of datatypes or tables but on the basis of the domains of datatypes resp. tables. The question wether to use tables or datatypes to get a physical implementation can then be decided by the programmer.

sub/superdomains Let A,B be OR-valued domains.
A inherits from B if the set of attributes of A is a superset of the attributes of B.
B is then said to be a superdomain of A.
A is then said to be a subdomain of B.

operators of a domain Let Op be an operator of some domain A. Every subdomain of A inherits the operator.
Considering the paramters of the operator, the following has to be remarked:
If instead of a parameter of domain B a parameter of a subdomain of B is handed over to the operator, the operator can perform its job without any problems.

The inheritance relation within a database can be visualized by using a graph. In general, an inheritance graph is always a DAG, sometimes, it might even be desired to force only one inheritance graph for the whole system (see Java), which assures a common structure for all objects of the inheritance hierarchy.

3.5.2 Inheritance in Postgres95

As Oracle does not provide inheritance, here is once again an example using Postgres95:

Inheritance is implemented in a rather straight forward way here[17]:

```
create table cities
(
  name       text,
  population float,
  altitude   int
);

create table captials
(
  state text
) inherits (cities);
```

In Postgres, a type can inherit from one or more other types, and a query can reference either all instances of a class or all instances of a class plus all of its descendants. A symbol * is used to indicate that the actual type of the query and its subtypes should be queried:

```
select c.name, c.altitude
from cities c
where c.altitude > 500;

select c.name, c.altitude
from cities* c
where c.altitude > 500;
```

4 The future of ORDBMSs

In this seminar paper, I provided an overview concerning the four new features that ORDBMSs are said to provide: base datatype extension, programming language, complex datatypes, inheritance. These extensions can offer a framework for easier and better modelling of database applications. Even though, it has yet to be proved that those new features are usable without heavy loss of performance. In addition, the question has to be addressed if a change to a real object-oriented database system would be possible and even more reasonable. The future will show if the new name "object-relational" and the implied new level in data processing is justified or if it was just an idea of marketing, if the object-relational features just remain as a nice add-on to relational systems which does not prove to be very helpfull in real life or if object-relational features are a real added-value to database systems that pays off.

[17] text is a variable length ASCII code string in Postgres95

5 Addendum

5.1 External procedures

PL/SQL does not perform good when used as a programming language for numerical computation. Oracle8 now offers the possibility of calling external C procedures to use the execution speed of this language.
Those external procedures have to be compiled into a library that can be linked dynamically (DLLs,SOs). The procedures are then executed in an own address space to safeguard the database system. This is an example of how to register an external library and procedure for the use within Oracle8:

```
create library c_utils as '/oracle/dll/utils.so';

create function really_heavy_computation
       (x binary_integer, y binary_integer)
return binary_integer
as external
external library c_utils
name calculate
language c;
```

And this is how to call an external function from PL/SQL:

```
create function fourier (x binary_integer)
return binary_integer
as
  z binary_integer :=10;
  result binary_integer;
begin
  result :=  really_heavy_computation(x,z);
  result := result + x;
  return result;
end;
```

PL/SQL provides new datatypes for the data exchange to external procedures (like binary_integer) and an extended parameter system to provide more security for parameter exchange.
This new feature of Oracle8 offers interesting new possibilities in database programming. It will be interesting to see how this feature is extended in the future (multi-threaded programming, IPC etc.).

5.2 Object views

In addition to the new possiblities of the Object Option, Oracle provides a means to add an object layer to an already existing relational database: object views. Those views allow to define a view that behaves like an object table on top of an already existing relational table:

```
drop view OVlex;
drop table Lexicon2;
create table Lexicon2
(
  id number,
  syn Syntax,
  sem Semantics,
  phon Phonetics,
  letters varchar2(30)
)
nested table sem.references store as references_table2;
insert into Lexicon2 values(1,NULL,NULL,NULL,'Auto');
insert into Lexicon2 values(2,NULL,NULL,NULL,'Helikopter');
insert into Lexicon2 values(3,NULL,NULL,NULL,'Pfanne');

create view OVlex of Word
with object oid (letters)
as
select l.syn,l.sem,l.phon,l.letters
from Lexicon2 l;
quit;
```

References

[C70] Codd, E. G. (1970). *A relational model for large shared data banks.* Communications of the ACM, 13(6), p. 377-387

[C79] Codd, E. G. (1979). *Extending the database relational model to capture more meaning.* ACM TODS 4

[CLC95] Chung, J.-Y., Lin, Y.-J., Chang, D. T. (1995). *Objects and relational databases.* OOPSLA '95 Workshop, http://www.eng.uci.edu/ cpeng/jychung/oopsla95-workshop.html

[D95] Date, C. J. (1995). *Relational database writings, 1991-1994.* Addison-Wesley, Reading/Mass.

[KE97] Kemper, A., Eickler, A. (1997). *Datenbanksysteme.* Oldenbourg, München, ST 271 K32 D2

[KM94] Kemper, A., Moerkotte, G. (1994). *Object-oriented database management.* Prentice Hall, Englewood Cliffs New Jersey, ST 271 K32

[M77] Makinouchi, A. (1977). *A consideration of normal form on not-necessarily normalized relations in the relational data model.* Proceedings of the international converence on very large data bases, p. 447-453

[OC97a] Oracle Co. (1997).*Oracle8 Server Concepts Vol. 1.* 811/ST 271 063(8)-4,1

[OC97B] Oracle Co. (1997).*Oracle8 Server Concepts Vol. 2.* 811/ST 271 063(8)-4,2

[PS87] Pollard, C. J., Sag, I. A. (1987). *Information-based syntax and semantics.* CSLI lecture notes, no. 13

[PG95] Yu, A., Chen, J. et al. (1995). *The Postgres95 User Manual.* (comes with Postgres95 distribution)

[S93] Stüerner, Günther (1993).*Oracle7 - die verteilte semantische Datenbank.* dbms publishing, Weissach

[SKS97] Silberschatz, A., Korth, H. F., Sudarshan, S. (1997). *Database system concepts.* McGraw-Hill, New York, NY, ST 271 K85

[SM96] Stonebraker, M., Moore, D. (1996). *Object-relational DBMSs.* Morgan Kaufmann, San Francisco, Calif., ST 271 S881 01

[SQL92] (1992). *Database language SQL, ANSI X3,135-1992.* American National Standards Instituts, New York

[SS86] Scheck, H. J., Scholl, M. H. (1986). *The relational model with relation-valued attributes.* Information System 11(2), p. 137-147