ORACLE 10G SQL

الفصل الاول

مميزات نظام إدارة قاعدة البيانات أوراكل

يتميز نظام قاعدة البيانات أوراكل عن غيره من نظم إدارة قواعد البيانات الأخرى بالآتي:

1- القدرة الفائقة على استيعاب كميات كبيرة من البيانات قد يصل عدد السجلات إلى الملايين مع الحفاظ على المستوى العالي في الأداء والسرعة عند استرجاع والتخزين والحذف

2- لاسرية التامة والأمن لا حتوائه على نظام الصلاحيات والحقوق الذي يضمن تطبيق الشروط القياسية والأمنية للحفاظ على قاعدة البيانات

3- فعالية التحكم المركزي بالبيانات لاذي يضمن :
   - تقليل التكررات عير اللازمة في البيانات الدخلة (No Repetition)
   - تجنب التناقض بين البيانات (No Contradiction)
   - إمكانية التشارك في البيانات (Data Sharing)
   - الحفاظ على تكامل البيانات فيما بينها (Data Integrity)

4- السيطرة التامة على عملية النسخ الاحتياطي لقاعدة البيانات وحمايتها من الفقدان أو التلف مع امكانية استرجاعها في أي لحظة

<u>لغة الاستعلام sql:</u>

هى عبارة عن مجموعة من الاوامر التى يحتاجها المبرمجين وكذلك المستخدمين للوصول للبيانات الموجودة ضمن قاعدة اوركل

<u>تاريخها:</u>

تم تطوير هذه اللغة فى البداية من قبل شركة ibm وذلك فى منتصف السبيعينات ثم قامت شركة اوركل فى 1979 بتطويرها وبانتاج اول نسخة تجارية من لغة sql

<u>مميزاتها:</u>

- انها قاعدة بيانات قوية مقارنة بمثيلتها مثل (Access) و(Microsoft SQL Server)
- انها تتمتع بقدر كبير من الامان وهو السبب وراء انتشارها
- سريعة جدا فى عملية البحث من خلالها.

و يمكن التعامل مع قاعدة البيانات من خلالها ومن خلال هذه اللغة يمكن اعطاء صلاحيات وامتيازات ممارسة عمليات معينة مثل:

1- انشاء جداول (create table)
2- التعديل فيها (alter)
3- حذف جداول (drop)
4- ملء جداول البيانات (insert)
5- حذف البيانات المدخلة (delete)
6- التعديل على البيانات المدخلة (update)
7 – البحث عن البيانات (query)

مكوناتها:
تنقسم لغة SQL الى ثلاث اقسام حيث تشكل كل مجموعة اوامر لغة فرعية من هذه اللغة وهى كالاتى:

1- أوامر لغة تعريف البيانات Data Definition Language (DDL)

تستخدم هذه اللغة في تعريف وإنشاء الكائن Object ، ويمكن أن يكون الكائن ملفات وجداول بيانات ، فيمكننا إنشاء وتعديل وحذف الكائن ويمكننا إنشاء امتياز لمستخدم معين ، أو انشاء كائن خيارات لفحص وإضافة تعليقات إلى قاموس البيانات
وتحتوى على ثلاث اوامر وهى :

- امر (Create table) يستخدم لانشاء الجداول
- امر(Alter table) يستحدم للتعديل فى جدول منشا سابقا
- امر (Drop table) يستخدم فى حذف جدول غير مرغوب فيه

حيث يقتصر عمل هذه الاوامر على الجداول وحقولها فقط دون التعرض للبيانات التى بداخل الجداول

2- أوامر لغة معاملة البيانات Data Manipulation Language (DML)

تتيح هذه الأوامر التعامل مع البيانات وتعديلها ضمن الكائن الموجود Object
وتحتوى على اربعة اوامر وهى:

- امر (Insert into) يستخدم فى ادخال البيانات الى الجداول
- امر (Update) يستخدم فى تعديل البيانات فى الجداول
- امر (Delete) يستخدم فى حذف البيانات من الجدول
- امر (Select) يستحدم فى الاستعلام عن شىء معين بيانات الجدول

3-أوامر لغة التحكم في البيانات Data Control Language (DCL)

تتيح هذه الأوامر التحكم في قاعدة البيانات وأدائها كالصلاحيات والمستخدمين والحقوق وغالبا ماتكون هذه الأوامر مخصصة للاستخدام من قبل مدير قاعدة البيانات (DBA) ومن هذه الاوامر : GRANT and REVOKE

أساسيات جملة SELECT

الأهداف :
بعد إكمالِ هذا الدرس، أنت يجِب أن تكون قادر إلى عمل التالي :

1. القدرة على تنفيذجملة SELECT.
2. الفرق بين SQL*PLUS و iSQL* plus

هدف الدرس:
جملة SELECT
تستخدم جملة SELECT لاسترجاع البيانات المخزنة في جدول أو عدة جداول حيث أن عملية الاسترجاع لا تعدل في هذه البيانات ويمكننا من خلال جملة SELECT أن نقوم بالتالي:

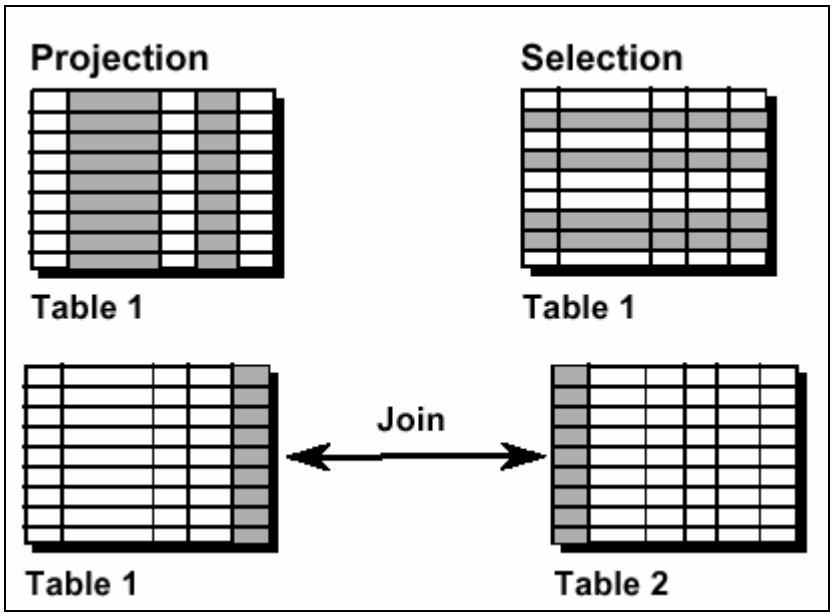
1- اختيار وعرض مجموعة معينة من السجلات المخزنة في الجدول
2- استرجاع بيانات بعض حقول الجدول
3- استرجاع بيانات مخزنة في جداول مختلفة

و يمكن أن تنشئ جمل SELECT أكثر من مرة.
هذا الدرس يغطي أيضا ISQL*Plus حيث تنفذ جمل Sql.
ملاحظ : iSQL* plus يعتبر مكان جديد لكتابة الكود بداخلة وذلك فى الأصدار ORACLE 10G. ذلك بالأضافة الى المكان القديم SQL*PLUS.

استخدامات جملة SELECT :

* **PROJECTION** : يستخدم في اختيار عمود أو أكثر من جدول معين .
* **S ELECTION** : يستخدم في اختيار صف أو أكثر من صف ذلك من جدول معين.
* **JOIN** : يستخدم للحصول على بيانات من جدولين من خلال الربط بينهما باحدى الطرق المختلفة التى سوف نراها فى الفصل الرابع.

الشكل التالى يوضح المفاهيم السابقة و لاحظ التظليل:



جملة **(SELECT Statement)** تسترجع المعلومات من قاعدة البيانات باحدى الطرق التالية :

* **الإسقاط (PROJECTION):** و بهذا يمكن استخلاص عمود أو أكثر من عمود من الجدول ولكن بدون امكانية تحديد صف معين.

* **الاختيار (SELECTION):** وهذا يعنى انة يمكن أن تختار مجموعة من الصفوف ذلك عن طريق آستعمال مجموعة معايير يمكن بها تحديد الصفوف التي تراها (اختيار افقى) .

* **الربط (JOIN):** أنت يمكن أن تستعمل الربط في SQL وهذا يعنى امكانية استخلاص بيانات من جدولين أو أكثر من جدول عن طريق إنشاء ربط بينهم باحدى الطرق المختلفة.

كتابة جمل SQL

نوضح فيما يلي بعض القواعد الإرشادية التي يجب أن توضع بعين الاعتبار غند كتابة جمل SQL

1- يمكن كتابة جمل SQL بالحروف الكبيرة أو الصغيرة
2- يمكن كتابة جمل SQL في عذة أسطر
3- لا يمكن فصل الكلمات المحجوزة عبر السطور مثل FROM
4- اترك مساقات بين مكونات الجملة لتسهيل عملية القراءة
5- في برنامج SQL *PLUS تكتب الأوامر مع مؤشر SQL ويتم تخزين هذا المر مباشرة في الذاكرة

```
SELECT    * | { [DISTINCT]  column | expression  [alias] ,...}
FROM        table ;
```

من الشكل السابق ، جملة SELECT يمكن أن تتضمن التالي :

* فقرة SELECT، تحدد الأعمدة التي سوف تعرض سواء كان عمود أو أكثر.
* FROM ،تحدد الجداول التي تحتوي على الأعمدة التى سجلت في فقرة SELECT السابقة على النحو التالي :

SELECT	تعنى اختيار عمود أو أكثر من عمود.
*	تعنى اختيار كل الأعمدة بدون تحديد اسمائهم.
DISTINCT	اخفاء التكرار فى بيانات العمود.
Column /expression	العمود الذى تم العمل علية عملية حسابية بداخل جملة SELECT.
alias	(الاسم)المستعار يعطي الأعمدة المختارة عناوين غير أسمائها الحقيقية.

كيفية عمل (ALIAS):

عندما تريد تغير اسم العمود من last_name الى name كما فى المثال التالى فهذا هو ما يسمى alias و هذا يكون فى الجدول المعروض فقط و يبقى اسم العمود بالجدول داخل قاعدة البيانات كما هو دون تغير.
FROM table تحدد الجداول التي تحتوي على الأعمدة

طرق استخدام Alias :-

- يمكن كتابة الاسم المستعار(ALIAS) بعد كتابة اسم العمود الحقيقى بشرط وجود مسافة بين الاسمين وان يكون الاسم المستعار لا يحتوى على مسافة.
- يمكن كتابة الاسم المستعار(ALIAS) بة مسافة ولكن لابد من كتابة الاسم المستعار بين double quotation

Example 1

```
Select ename NAME from "T emp" ;
```

تم وضع double quotation لان الاسم المستعار يحتوى على مسافة.

Example 2

```
Select  FIRST_NAME NAMESSS ,JOB_ID
from employees
```

NAMESSS	JOB_ID
Steven	AD_PRES
Neena	AD_VP
Lex	AD_VP
Alexander	IT_PROG
Bruce	IT_PROG
David	IT_PROG
Valli	IT_PROG
Diana	IT_PROG
Nancy	FI_MGR
Daniel	FI_ACCOUNT
John	FI_ACCOUNT
Ismael	FI_ACCOUNT
Jose Manuel	FI_ACCOUNT
Luis	FI_ACCOUNT

نلاحظ ان هنا FIRST_NAME تحول اسمه الى NAMESSS

مثال لاختيار كل الأعمدة بدون ذكر اسمائها :

```
SELECT *
FROM   departments;
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

8 rows selected.

في **المثال** تم عرض كل الصفوف بالجدول وذلك لأن أمر SELECT لم يتبعه شرط تحديد عدد الصفوف.

استخدام أوامر SQL ← SELECT

وهو يستعرض بيانات الجدول وتشير النجمة المستخدمة * الى إظهار جميع أعمدة الجدول ثم نكتب أمر FROM يليه أسم الجدول الذي يراد الاستعلام عن بياناته وبعد انتهاء الأمر يتم وضع ;

لاحظ أن أوامر SQL تطلب كتابة فصلة منقوطة(;) في آخر الأمر وهنا سيتم استعراض جميع بيانات أعمدة الجدول ملحوظة أوامر SQL لا تختصر مثل كتابة SEL . أنت يمكن أن تعرض كل أعمدة الجدول بكتابة الكلمة الرئيسية SELECT ويليها بنجمة (*)

من المثال السابق نجد أن جدول الاقسام يحتوي علي أربعة أعمدة وهم:
DEPARTMENT_ID , DEPARTMENT_NAME, MANAGER_ID , LOCATION_ID .

أنت يمكن أن تعرض أيضا كل الأعمدة في الجداول بتسجيل كل الأعمدة بعد كلمة SELECT
بدلا من استخدام النجمة(*).

Example:

```
SELECT  department_id, department_name, manager_id,location_id
FROM    departments  ;
```

ويمكن ان نختار عدد محدد من الاعمدة فقط ذلك عن طريق كتابة اسماء الاعمدة المراد عرضها بعد كلمة SELECT ووضع coma (,) بين كل عمود كما فى المثال التالى:

اختيار الأعمدة المحددة

```
SELECT department_id, location_id
FROM   departments;
```

DEPARTMENT_ID	LOCATION_ID
10	1700
20	1800
50	1500
60	1400
80	2500
90	1700
110	1700
190	1700

8 rows selected.

وهذا يعنى اننا من الممكن عرض أعمدة محددة وليست كل الأعمدة.

ولاحظ ان ترتيب ظهور الاعمدة يكون على اساس ترتيب تلك الاعمدة فى جملة SELECT.
فالفرق بين المثالين حيث الأول كان SELECT بعدها department_id وعرضت أولا يسارا في الجدول عكس المثال التالى حيث جاءت في الترتيب الثاني .

```
SELECT    location_id,  department_id
FROM      departments  ;
```

LOCATION_ID	DEPARTMENT_ID
1700	10
1800	20
1500	50

اختيار 2 عمود من dept table:

```
SQL> select  DEPTNO,DNAME from dept;

    DEPTNO DNAME
---------- --------------
        10 ACCOUNTING
        20 RESEARCH
        30 SALES
        40 OPERATIONS
```

كتابة جمل SQL

1. جمل SQL يمكن أن تكون فى سطر أو أكثر من سطر أي يمكن كتابتها فى عدة سطور.
2. كلماتها الرئيسية لا يمكن أن تختصر أو تنفصل عبر السطور .
3. فقراتها توضع عادة على سطور منفصلة لتسهيل مراجعتها .
4. ترك مسافات بين مكونات الجملة كي تحسن القراءة.

تنفيذ جمل SQL

في ISQL*Plus، انقر على زر التنفيذ (Execute) لكي يتم التنفيذ .

ملاحظة

يتطلب في ISQL*PLUS او SQL*PLUS أن تضع semicolon (;) في نهاية الفقرة الأخيرة

- أختلاف عنوان العمود بين SQL*PLUS & ISQL*Plus:

Sql*plus:

1. عناوين الحروف و التواريخ لرأس العمود تظهر يسارا .
2. عناوين رأس العمود للأرقام تظهر يمينا.
3. تعرض العناوين بحروف كبيرة

```
SELECT ename, date, sal
FROM   emp     ;
```

ENAM	DATE	SAL
KING	17-JUN-87	1500

```
SQL> select  DEPTNO,DNAME from dept;

    DEPTNO DNAME
---------- --------------
        10 ACCOUNTING
        20 RESEARCH
        30 SALES
        40 OPERATIONS
```

ISQL*Plus:

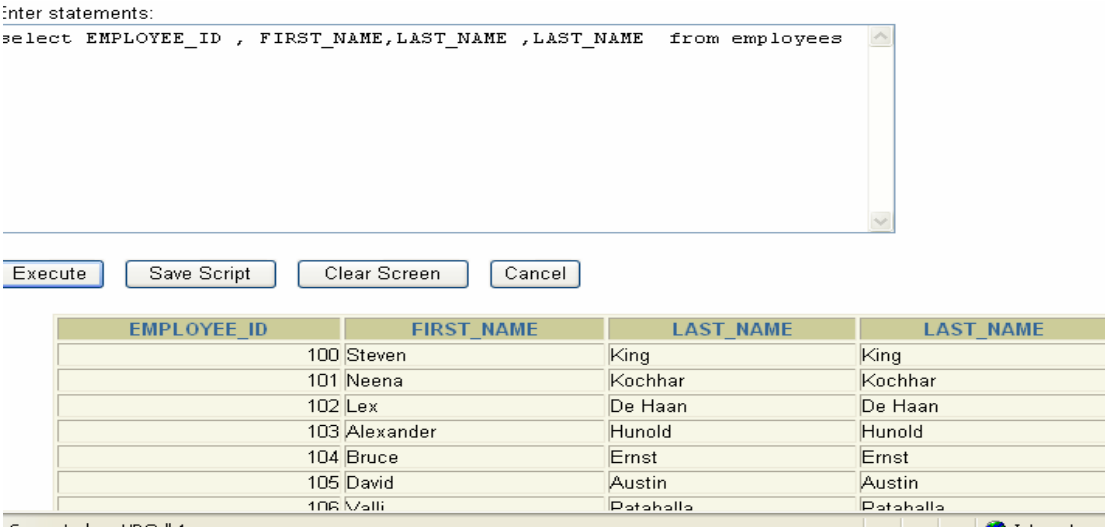
* عنوان العمود يوضع في المنتصف.

- عرض العنوان (في أعلى الصفحة) بحروف كبيرة وفى الوسط.

```
SELECT last_name, hire_date, salary
FROM employees;
```

LAST_NAME	HIRE_DATE	SALARY
King	17-JUN-87	24000
Kochhar	21-SEP-89	17000
De Haan	13-JAN-93	17000
Hunold	03-JAN-90	9000
Ernst	21-MAY-91	6000
...		
Higgins	07-JUN-94	12000
Gietz	07-JUN-94	8300

20 rows selected.



العملية	الوصف
+	الإضافة
-	الطرح
*	الضرب
/	القسمة

العمليات الحسابية

وربما تحتاج أن تعدل طريقة عرض البيانات لتؤدي حسابات معينة ،دون أن تؤثر على البيانات المخزنة في الجدول. هذا يتم بآستعمال العمليات الحسابية. يمكن أن يحتوي التعبير الحسابي علي أسم العمود،او قيمة ثابتة،والعوامل الحسابية مثل(-,*,+,/) .

العوامل الحسابية

يشرح الجدول السابق العوامل الحسابية المتوفرة في SQL. أنت يمكن أن تستعمل العوامل الحسابية في أي فقرة SQL ماعدا فى الفقرة FROM

ملاحظة:

يمكن أن تستعمل مع التاريخ العوامل الجمع والطرح فقط.

استخدام العماليات الحسابية

```
SELECT last_name, salary, salary + 300
FROM   employees;
```

LAST_NAME	SALARY	SALARY+300
King	24000	24300
Kochhar	17000	17300
De Haan	17000	17300
Hunold	9000	9300
Ernst	6000	6300
...		
Hartstein	13000	13300
Fay	6000	6300
Higgins	12000	12300
Gietz	8300	8600

20 rows selected.

المثال : يوضح استعمال عملية الجمع حيث يحسب زيادة للراتب 300$ مع كل الموظفون

و يعرضون المرتب الجديد +300 في العمود .

ملاحظة الناتج من حساب(راتب العمود +300) ليست عمود جديد في جدول الموظفين إنما هو عمود للعرض فقط.

أسم العمود الجديد يجيء من ناتج حساب + المرتب(راتب + 300) وهنا يمكن استعمال
(ALIAS) حتى نتمكن من تسمية تلك العمود.
الملاحظة: ORACLE 10G يهمل المسافات الفارغة قبل وبعد العوامل الحسابية .

اسبقية عمل العوامل الحسابية:

*	/	+	-

1. الأولوية تكون للضرب الأول ثم القسمة ثم الجمع ثم الطرح.
2. الأولوية من اليسار آلي اليمين.
3. تستخدم الأقواس حتى تتمكن من تنفيذ الجمع او الطرح اولا قبل الضرب او القسمة.

أسبقية العوامل

إذا كان هناك اكثر من عملية مختلفة ، ضرب وقسمة وجمع و طرح. أنت يمكن أن تستعمل الأقواس لضمان تنفيذ العمليات التى بداخل الأقواس أولاً . ثم الباقي بالترتيب من اليسار إلى اليمين.
تستطيع أن تستخدم الأقواس كي تنفذ الذى بداخلة اولا. أي انك عندما تضعهم فى الأقواس ينفذ الاقواس اولا سواء كان جمع أو طرح حيث ان بين الأقواس ينفذ اولا.

مثال على أسبقية العوامل

```
SELECT last_name, salary, 12*salary+100
FROM   employees;
```

LAST_NAME	SALARY	12*SALARY+100
King	24000	288100
Kochhar	17000	204100
De Haan	17000	204100
Hunold	9000	108100
Ernst	6000	72100
...		
Hartstein	13000	156100
Fay	6000	72100
Higgins	12000	144100
Gietz	8300	99700

20 rows selected.

أسبقية العامل:
المثال السابق يعرض (الاسم الأخير، المرتب، المرتب*12+100).

ملحوظة عملية الضرب يتم تنفيذها قبل إضافة رقم 100.
ملاحظة: تستخدام الأقواس كي توضح اولوية العملية الحسابية وتسهل تصحيح الاخطاء.
لاحظ يمكن أن يكتب (salary *12) +100 لا يحدث تغيير في النتيجة السابقة لكن التعبير اصبح اكثر وضوحاً .

```
SELECT    EMPLOYEE_ID , SALARY *15/100
FROM      employees;
```

EMPLOYEE_ID	SALARY*15/100
100	3600
101	2550
102	2550
103	1350
104	900
105	720
106	720
107	630
108	1800
109	1350
110	1230
111	1155
112	1170
113	1035

نلاحظ فى هذا المثال ان تم ضرب المرتب salary فى 15 ثم القسمة /100

استخدام الأقواس

```
SELECT last_name, salary, 12*(salary+100)
FROM   employees;
```

LAST_NAME	SALARY	12*(SALARY+100)
King	24000	289200
Kochhar	17000	205200
De Haan	17000	205200
Hunold	9000	109200
Ernst	6000	73200
...		
Hartstein	13000	157200
Fay	6000	73200
Higgins	12000	145200
Gietz	8300	100800

20 rows selected.

استخدام الأقواس:

يمكن أن تتجاوز قواعد الأسبقية فى تنفيذ العمليات الحسابية باستخدام الأقواس لتحدد ترتيب حساب العمليات الحسابية التي تنفذ اولا حسب احتياجاتك.

المثال السابق يقوم بعرض الاسم الأخير، المرتب، (المرتب+100)*12.

هو بذلك يقوم بجمع 100 على المرتب اولا كل موظف ثم بعد ذلك يقوم بضرب الناتج فى 12.

بسبب الأقواس،يمكن لعملية الجمع ان تنفذ قبل الضرب .

فبسبب الأقواس يتم حساب (salary+100) أولا ثم يضرب الناتج × 12.

مثال اخر :-

```
SELECT    EMPLOYEE_ID, 15*(SALARY +100 )
 from          employees;
```

EMPLOYEE_ID	15*(SALARY+100)
100	361500
101	256500
102	256500
103	136500
104	91500
105	73500
106	73500
107	64500
108	181500
109	136500
110	124500
111	117000
112	118500
113	105000

نلاحظ ان عملية الجمع تمت اولا ثم عملية الضرب

التعامل مع القيمة Null

- Null هي قيمة غير متاحة غير محددة غير معروفة وغير قابلة لاجراء اى عملية
  حسابية عليها.
- Null ليست مساوية صفر أو مسافة.

```
SELECT last_name, job_id, salary, commission_pct
FROM   employees;
```

LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
King	AD_PRES	24000	
Kochhar	AD_VP	17000	
...			
Zlotkey	SA_MAN	10500	.2
Abel	SA_REP	11000	.3
Taylor	SA_REP	8600	.2
...			
Gietz	AC_ACCOUNT	8300	

20 rows selected.

قيم Null
اى خلية فى الجدول فارغة تكون NULL.
Null قيمة غير متاحة غير محددة غير معروفة وغير قابلة لاجراء عمليات حسابية عليها.
Null لا تساوى صفر أو مسافة فارغة .حيث ان الصفر عدد، والفراغ يعتبر حرف .
أي نوع من البيانات يمكن أن تحتوي على null.
على أية حال، يمكن وضع بعض القيود على الجدول مثل NOT NULL و PRIMARY KEY
وهذا يعنى عدم امكانية أحتواء العمود NULL .
ذلك كما سوف نرى فيما بعد .

لاحظ فى المثال السابق ان الذي لديه عموله هو مدير مبيعات أو مندوب المبيعات وباقى الموظفون ليس لديهم نسبة عمولات لذلك قيم العمولة عندهم null .

قيمة Null في العمليات الحسابية
- لا يمكن اجراء عمليات حسابية على NULL.

```
SELECT last_name, 12*salary*commission_pct
FROM   employees;
```

LAST_NAME	12*SALARY*COMMISSION_PCT
King	
Kochhar	
...	
Zlotkey	25200
Abel	39600
Taylor	20640
...	
Gietz	

20 rows selected.

ملحوظة:

عندما تحاول أن تقسم أي رقم على صفر، تصبح النتيجة خطأ. وايضا عندما تحاول اجراء اى عملية حسابية على Null، فتكون النتيجة NULL.

من المثال السابق لاحظ ان (KING) لا يحصل على أي عمولة ولكنة يحصل على مرتب.ومع ذلك النتيجة اصبحت Null فى الخلية الخاصة بة و هذا يؤكد ان اجراء اى عملية حسابية على NULL تؤدى الى NULL.

- للمزيد من المعلومات، يرى مرجع ORACLE 10g SQL، " BASIC ELEMENTS OF SQL"

علامة اللصق (الضم) (||)
Concatenation

وهي تستخدم لكي نضم عمودين فى عمود واحد فيتم استخدام علامة || كي تلصق الأعمدة.

Example

```
SELECT   last_name||job_id AS "Employees"
FROM     employees;
```

Employees
KingAD_PRES
KochharAD_VP
De HaanAD_VP
HunoldIT_PROG
ErnstIT_PROG
LorentzIT_PROG
MourgosST_MAN
RajsST_CLERK
...

20 rows selected.

تم دمج عمود اسم الموظف مع وظيفتة فى عمود واحد وسمى Employees.

```
select FIRST_NAME||last_name as "name of employee" from employees
```

name of employee
StevenKing
NeenaKochhar
LexDe Haan
AlexanderHunold
BruceErnst
DavidAustin
ValliPataballa
DianaLorentz
NancyGreenberg
DanielFaviet
JohnChen
IsmaelSciarra
Jose ManuelUrman
LuisPopp
name of employee
DenRaphaely

نلاحظ من المثال السابق ان الاسم الاول والثانى اصبحا معا تحت اسم name of employee

سلسلة الحروف الحرفية

1. LITERAL تعبر عن حرف او عدد أو تاريخ و تتضمن في جملة SELECT.
2. اى قيم حرفية يجِب أن تكون مرفقة ضمن العلامة (' ').
3. LITERAL تكون حرف او عدد أو تاريخ.
4. الحروف و التواريخ يوضعان بين العلامة (' ') .

استخدام مجموعة الحروف الحرفية

```
SELECT last_name ||' is a '||job_id
       AS "Employee Details"
FROM   employees;
```

Employee Details
King is a AD_PRES
Kochhar is a AD_VP
De Haan is a AD_VP
Hunold is a IT_PROG
Ernst is a IT_PROG
Lorentz is a IT_PROG
Mourgos is a ST_MAN
Rajs is a ST_CLERK
. . .

20 rows selected.

<u>مثال اخر</u>

المثال يعرض الأسماء و الوظيفة لكل الموظفون. وعنوان العمود Employee Details .

الفراغ يحسن قراءة الناتج .

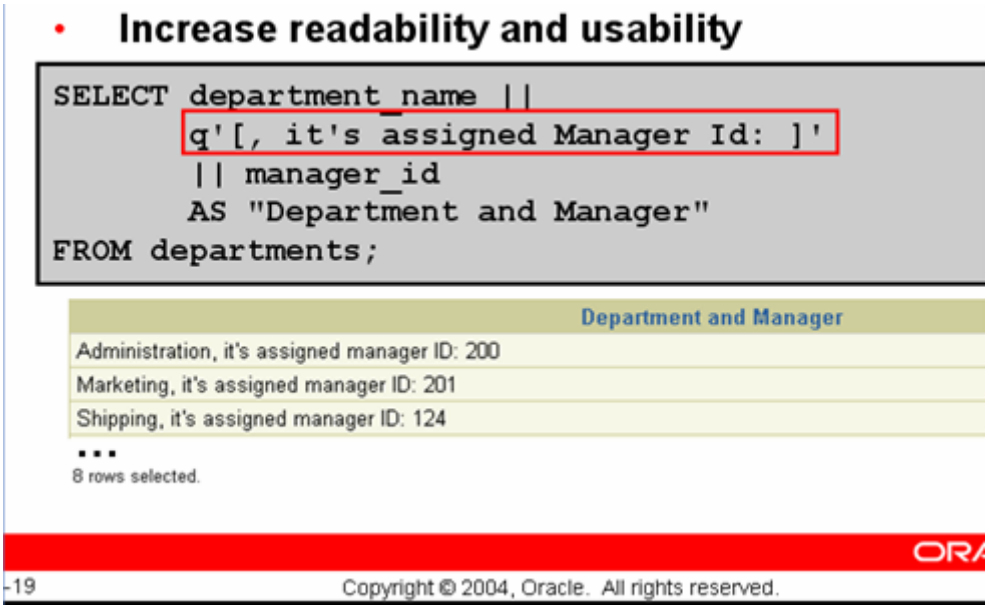
عندما نريد ان نضيف كلمة او اكثر من كلمة قبل او بعد ظهور العمود لابد من:

اولا: يجب وضع الكلمات التى تريد ادراجها بين single quotation.

ثانيا: عمل concatenation مع هذة الكلمات.

ففى **المثال** التالي، تم ضم الاسم الأخير مع جملة (= 1 month salary :)مع الراتب لكل موظف و**لاحظ** انك لابد من وضع تلك الجملة بين single quotation.

* اذا كنت تريد وضع علامة Single Quotation داخل الجملة التى تريد ظهورها قبل او بعد او بين العمود يجب عليك وضع علامة q وضع الجملة المراد أضافتها بين [ ]



SELECT last_name ||': 1 Month salary = '||salary Monthly
FROM employees;

MONTHLY
King: 1 Month salary = 24000
Kochhar: 1 Month salary = 17000
De Haan: 1 Month salary = 17000
Hunold: 1 Month salary = 9000
Ernst: 1 Month salary = 6000
Lorentz: 1 Month salary = 4200
Mourgos: 1 Month salary = 5800
Rajs: 1 Month salary = 3500

20 rows selected.

(أي أنك عند الرغبة في وضع حروف بين عمودين يراد لصقهما فأننا نكتب هذه الحروف بين علامتين تنصيص مفردتين (|| ' .. ' ||) ولو أرقام فلا توضع بين شئ.

كيفية ازالة الصفوف المتكررة

إنّ الاستعلام يكون لكل الصفوف بالجدول و يشمل الصفوف المتكررة .

```
SELECT department_id
FROM   employees;
```

DEPARTMENT_ID
90
90
90
60
60
60
50
50
50

20 rows selected.

ملحوظة أعداد القسم تتكرر .

إزالة الصفوف المتكررة

إزالة الصفوف المتكررة باستخدام DISTINCT

```
SELECT DISTINCT department_id
FROM   employees;
```

DEPARTMENT_ID
10
20
50
60
80
90
110

9 rows selected.

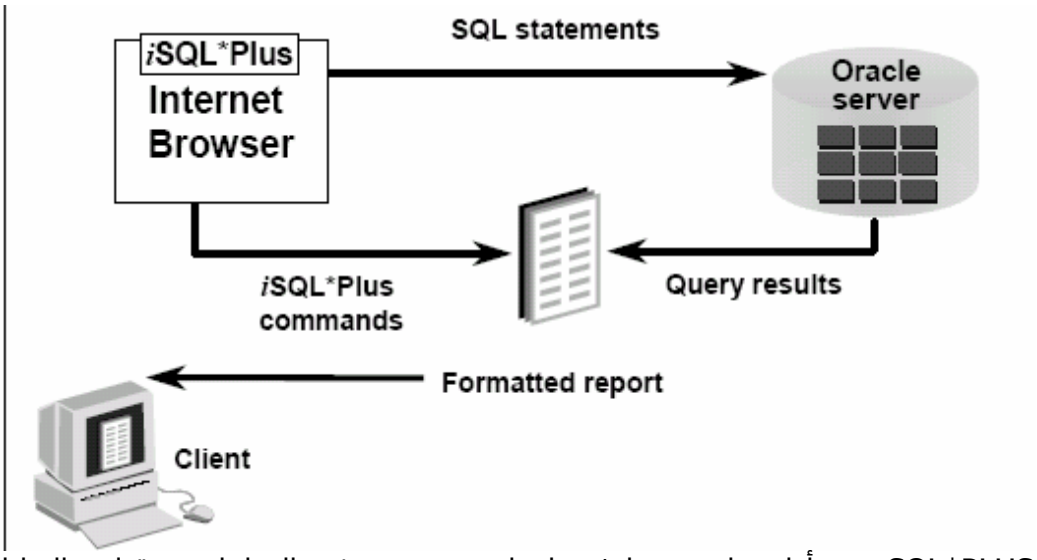
لكي تزيل الصفوف المتكررة يجب وضع كلمة DISTINCT قبل اسم العمود المراد ازالة التكرار منة ففى المثال السابق،جدول الموظفون الفعلي يتكون من 20 صف لكنة ظهر بدون تكرارات بعد استخدام كلمة DISTINCT .

Example:
SELECT DISTINCT department_id
FROM employees;

DEPARTMENT_ID	JOB_ID
10	AD_ASST
20	MK_MAN
20	MK_REP
50	ST_CLERK
50	ST_MAN
60	IT_PROG
...	
	SA_REP

13 rows selected.

الفرق بين ISQL*PLUS و SQL



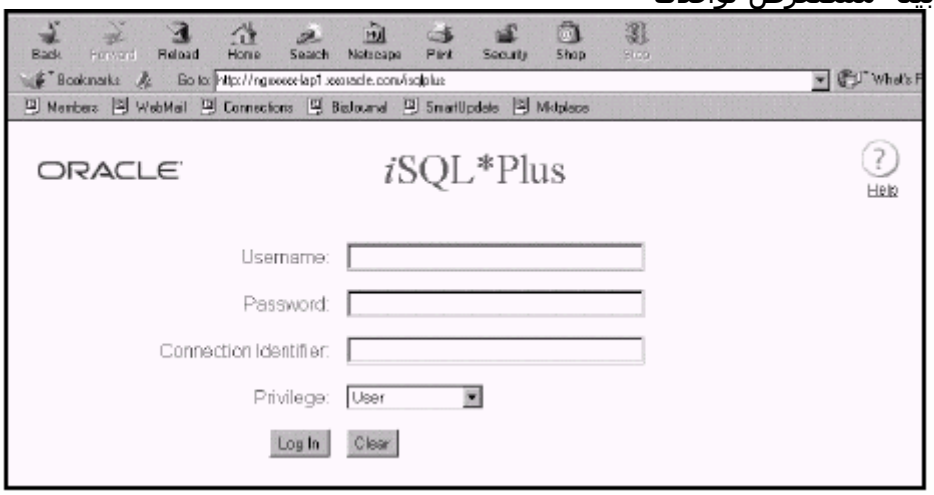
SQL*PLUS هي أوامر ثابتة متعارف عليها و تستخدم في التعامل مع قواعد البيانات .

ISQL*PlUS هي أوامر خاصة للعمل في مكان isql*plus

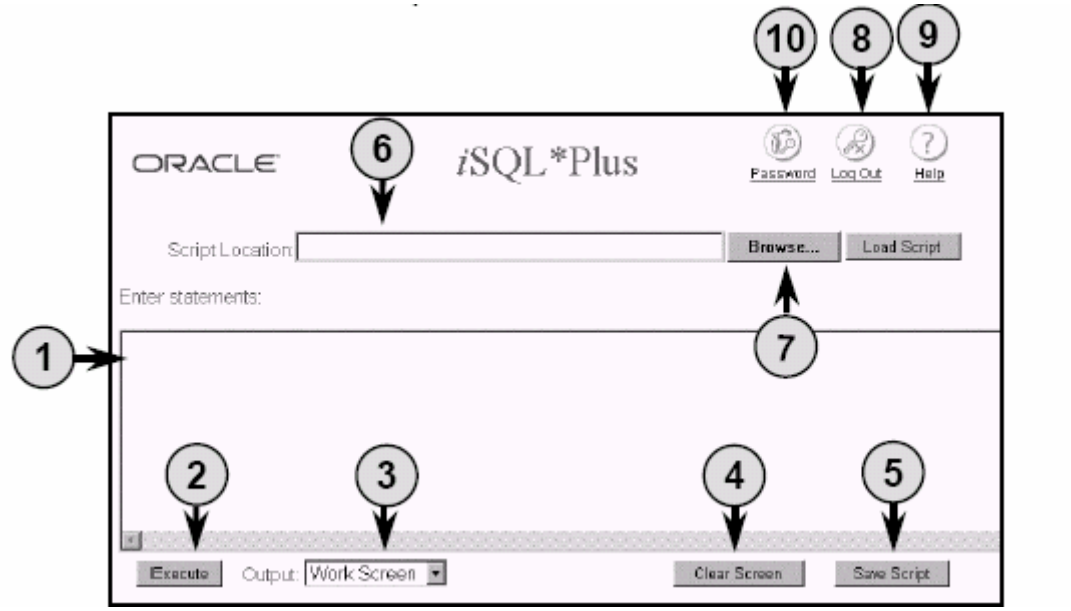
الفرق بين SQL*PLUS و iSQL* PLUS

iSQL* PLUS	SQL
هي أوامر خاصة بالعمل في isql*plus	أوامر sql هي أوامر ثابتة متعارف عليها وتستخدم في التعامل مع قواعد البيانات بكافة اشكالها.
يمكن اختصار أوامر isql*plus	لا يمكن اختصار أوامرها.
لا يذهب آخر أمر في isql*plus إلى buffer	تذهب أوامر sql إلى جزء في الذاكرة يسمي buffer وذلك لحفظ آخر أمر لسهولة تعديله وهو جزء في الذاكرة يحفظ آخر أمر sql لسهولة الاسترجاع والتعديل
تاتى بالبيانات من على SERVER بطريقة غير مباشرة .بمعنى عدم وجود قاعد بيانات على نفس الجهاز.	تاتى بالبيانات من على SERVER مباشرة بمعنى انة لابد من وجود قواعد بيانات على نفس الجهاز .
يمكنك استخدام علامة (--) لكى يمكنك من ايقاف الكود الذى على نفس السطر.	لا يمكنك من من استخدام تلك العلامة.
يمكنك من تعديل الكود اذا حدث بة خطأ بسهولة.	يمكنك من تعديل الكود اذا حدث بة خطأ عن طريق كتابة ED ثم اضغط ENTER فتظهر لة NOT PAD يتم عمل تعديل الكود بداخلها.

تسجيل الدخول إلي Isql*Plus
من بيئة مستعرض نوافذك



تَسجيل الدخول إلي :iSQL*Plus
1- أنقر علي عنوان (URL) في بيئة iSQL*Plus
2- أدخل أسم المستخدم و الرقم السري .



ملخص الفصل

تناولنا في هذا الفصل كتابة جمل SQL البسيطو (جملة SELECT) التي تستخدم في استرجاع البيانات وكذلك بعض القواعد الإرشادية التي يجب اتباعها عند كتابة وتنفيذ جمل الـ SQL كما تناولنا استخدام العمليات الحسابية مع جملة SELECT والعملية الإلحاقية وكذلك عبارة DISTINCT التي تستخدم لمنع تكرار ظهور السجلات

الفصل الثاني

اختيار صفوف محددة:-

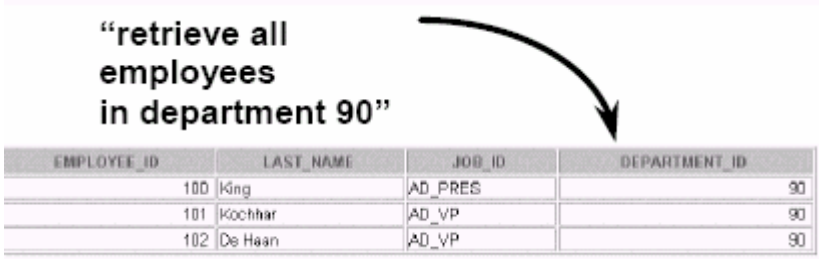
الاهداف:

- امكانية استخلاص مجموعة فقط من البيانات.
- امكانية ترتيب البيانات الناتجة عن جملة SELECT .

مقدمة

لقد قمنا في الوحدة السابقة بالتعرف الى جملة الاستعلام البسيط (SELECT Statement) التي من خلالها تم استرجاع البيانات من الجداول . وفي هذا الفصل سنتابع الحديث عن هذه الجملة بشكل أوسع ، حيث سنسترجع البيانات من الجداول بناء على شروط معينة ، أو مرتبة حسب بيانات حقول معينة ، أي سندرس المور المتعلقة بجملة الشرط في جملة الاسترجاع ، من حيث المعاملات الشرطية وطريقة التعبير عن الشرط والترتيب التصاعدي والترتيب التصاعدي والترتيب التنازلي للبيانات

هنا يتم آسترجاع كل الموظفون في الادارة رقم 90



فى **المثال** السابق ، يريد عرض كل الموظفين في الادارة رقم 90 وسوف تجد أنه الوحيد الذي تم عرضه . هذا يتم عن طريق اضافة العبارة WHERE في جملة Select.

- امكانية استرجاع الصفوف باستخدام فقرة WHERE .
-

```
SELECT *|{[DISTINCT] column|expression [alias],...}
FROM table
[WHERE condition(s)];
```

فقرة WHERE تاتى بعد فقرة FROM .

اختيار صفوف محددة:

يمكن أن تحدد الصفوف التى سوف تستخلصها باستخدام كلمة WHERE التى تحتوي على الشرط لامكانية حصر البيانات التي سيتم استخلاصها من الجدول .

تتضمن WHERE علي ثلاثة عناصر (مع مراعاة الترتيب) :

* أسم العمودِ الذى تريد اجراء المقارنة علية.
* شرط المقارنةِ.
* أسم العمودِ أو القيمة أو مجموعة من القِيمِ.

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees
WHERE  department_id = 90 ;
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90

نجد فى المثال السابق ظهور البيانات الخاصة بالقسم رقم 90 فقط حيث
تم استخدام WHERE لتحديد البيانات الخاصة بهذا القسم فقط .

مثال اخر :

Enter statements:

```
select FIRST_NAME ,DEPARTMENT_ID from employees
where
DEPARTMENT_ID =60
```

FIRST_NAME	DEPARTMENT_ID
Alexander	60
Bruce	60
David	60
Valli	60
Diana	60

تستخدام WHERE
عندما نريد تحديد صف أو مجموعة صفوف معينه نستخدم WHERE و هذا الشرط يقوم بحصر الاختيار لتلك الصفوف فقط .
وتبدأ جملة الشرط بكلمة WHERE يليها الشرط المراد تحققه.
فى المثالِ السابق ، جملة (SELECT statement) تسترجع الاسم و رقم القسم لكل موظف يعمل فى القسم رقم 60 .

ملاحظة:

مجموعة الحروف والتواريخ يوضع حولها علامات الترقيم الفردية ' ' (Single Quotation) و ايضا صيغة التاريخ المختارةِ DD-MON-RR
بمعنى اليوم يكتب في حرفين والشهر يكتب في ثلاثة أحرف و السنة تكتب في حرفين.

```
SELECT last_name, job_id, department_id
FROM employees
WHERE last_name = 'Whalen';
```

هنا الحرف الأول كبير والباقي حروف صغيرة

فى **المثال** السابق يجب ان نعرف شكل كتابة LAST_NAME بداخل الجدول هل مكتوبة Capital او Small حيث ان عند تحديد الشرط لابد من معرفة شكل كتابة الاسم.
قواعد بيانات ORACLE تخزن التواريخ في صيغة أشكال عددية، يمثل القرن، السنة، الشهر، اليوم ،الساعات، الدقائق، والثواني. إن عرض التاريخ المختار DD-MON-RR.
ملاحظة: تغيير صيغة عرض التاريخ تغطي في الفصل التالى .
ملاحظة:بعض الطلاب ربما يسألون كيف يمكن تغير شكل التاريخ ذلك يكون في الفصل
التالى ، سوف نغطي استخدام single-row functions .

شروط المقارنة (معاملات المقارنة)

المعنى	العوامل
يساوي	=
أكبر من	>
أكبر من أو يساوى	> =
أقل من	<
أقل من أو يساوي	< =
لا يساوي	< >

شروط المقارنة (معاملات المقارنة)
شروط المقارنة تستعمل لامكانية عمل المراقنة .

مثال:

. . WHERE hire_date ='01-JAN-95'
... WHERE salary >=6000
... WHERE last_name ='Smith'

الأسم المستعار (Alias) لا يمكن أن يستعمل في فقرة WHERE .

استخدام معاملات المقارنة :-

تستخدم معاملات المقارنة في جملة الشرط لمقارنة تعبير بآخر في جملة WHERE .

كما في المثال:

```
SELECT last_name, salary
FROM   employees
WHERE  salary <= 3000;
```

LAST_NAME	SALARY
Matos	2600
Vargas	2500

ففى المثال السابق تم استخلاص السم الاخير و المرتب لكافة الموظفين الذين يحصلون على مرتب اقل من او يساوى 3000 .

ملاحظة: يمكن استخدام =^ او != بدلا من <> للدلالة على انة لا يساوى
كما فى المثال التالى :

Enter statements:

```
SELECT last_name, job_id,salary department_id
FROM employees
WHERE salary^=5000;
```

LAST_NAME	JOB_ID	DEPARTMENT_ID
King	AD_PRES	24000
Kochhar	AD_VP	17000
De Haan	AD_VP	17000
Hunold	IT_PROG	9000
Ernst	IT_PROG	6000
Austin	IT_PROG	4800
Pataballa	IT_PROG	4800
Lorentz	IT_PROG	4200

فى **المثال** السابق تم استخلاص السم الاخير و الوظيفة و المرتب و تم اعطاء اسم مستعار للمرتب ليصبح DEPARTMENT_ID وذلك لكل الموظفين الذين يحصلون على مرتب لا يساوى 5000 . ملاحظة هنا تم اعطاء قيمةُ واضحةُ بفقرة WHERE.

اى إن 5000 تقارن بقيم عمود المرتبات بجدول الموظفين ويستخلص المرتبات التى لا تساوى 5000.

معاملات أخرى للمقارنة:

المعنى	المعامل	نفي المعامل
هل القيمة داخل النطاق الذى يبدأ بالقيمة small وينتهى بالقيمةBig	BETWEEN small AND Big	NOT BETWEEN
هل القيمة بين مجموعة من القيم التى بين القوسين	IN(set)	NOT IN
هل القيمة شبيهه بقيمة معينة	LIKE	NOT LIKE
هل هي قيمة فارغة	IS NULL	IS NOT NULL

استخدام شرط BETWEEN:

علامة الحصر BETWEEN تستخدم لحصر قيمة بين قيمتين وهي تكتب كالتالي BETWEEN ثم يليها القيمة الصغرى اولا ثم يليها AND ثم يليها القيمة الكبرى .
تستخدم لاسترجاع بيانات الصفوف فى المدي بين القيمتين.

```
SELECT last_name, salary
FROM   employees
WHERE  salary BETWEEN 2500 AND 3500;
```

أصغر قيمة أكبر قيمة

LAST_NAME	SALARY
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500

استخدام شرط BETWEEN

وفي المثال تم عرض بيانات الموظفين الذين تنحصر مرتباتهم بين 2500 و 3500 أي أنه يتم عرض بيانات الموظفين الذين يتقاضون مرتب قدرة 2500 أو يزيد عن ذلك وفي نفس الوقت لا يزيد عن 3500.

في المثال السابق تم استرجاع الصفوف من جدول الموظفين الذي راتبه ما بين 2,500 و 3,500 .

تحديد القيم مع شرط BETWEEN يجب أن تحدد أقل قيمة أولا ثم أكبر قيمة ثانيا .

مثال اخر:

Enter statements:

```
select FIRST_NAME, salary from employees
where
salary between 5000 and 6000;
```

FIRST_NAME	SALARY
Bruce	6000
Kevin	5800
Pat	6000

استخدام معامل الشرط IN

يستخدم هذا المعامل للبحث عن قيمة داخل مجموعة من القيم وتوضع بين قوسين.

مثال :

```
SELECT employee_id, last_name, salary, manager_id
FROM   employees
WHERE  manager_id IN (100, 101, 201);
```

EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
202	Fay	6000	201
200	Whalen	4400	101
205	Higgins	12000	101
101	Kochhar	17000	100
102	De Haan	17000	100
124	Mourgos	5800	100
149	Zlotkey	10500	100
201	Hartstein	13000	100

في **المثال** السابق تم استخلاص رقم الموظف و الاسم الاخير و المرتب و رقم المدير لاى موظف يعمل للمدرين الذين ارقامهم فى (100,101,201) .
ولاحظ عمود MANAGER_ID تجد انة يحتوى على تلك الارقام المحددة بين القوسين.
شرط IN يمكن أن يستعمل مع أي نوع من البيانات(ارقام او حروف او .....).
فالمثال التالي يسترجع رقم الموظف و رقم المدير و رقم القسم للموظفين الذين اسمائهم مثل (Hartstein او Vargas)

```
SELECT employee_id, manager_id, department_id
FROM employees
WHERE last_name IN ('Hartstein', 'Vargas');
```

لاحظ عندما نريد المقارنة (بحروف أو بتواريخ) فيجب أن يكونوا مرفقين في علامات التقدير (' ') SINGLE QUOTATION .
مثال على ذلك:

Enter statements:

```
select FIRST_NAME, salary from employees
where
salary IN(5000 , 6000);
```

FIRST_NAME	SALARY
Bruce	6000
Pat	6000

نلاحظ فى المثال السابق ان نتائج SALARY كلها 6000 وذلك لانة لا يوجد مرتبات تساوى 5000.

استخدام شرط LIKE

كيف يمكن البحث عن اسماء عن طريق البحث بحرف او اكثر من حرف ؟؟؟

- تستخدم (_) UNDER SCORL لتفادى حرف او اكثر من حرف من الكلمة المراد البحث عنها.فيعوض عن الحرف الواحد بعلامة(_) UNDER SCORL .
- تستخدم (%) percentage لتفادى اكثر من حرف بدون استخدام (_) .

- (تستخدم LIKE للبحث العشوائي الغير محدد ويمكن استخدامها مع الحروف والأرقام وبعدها يجب وضع القيمة التي تلي LIKE بين علامة ' ' SINGLE QUOTATIONحيث سيتم مطابقة حروف النص المذكورة)

* استخدام شرط LIKE للبحث العشوائي علي مجموعة القيم الصحيحةِ .
(%) Percentage يدل علي صفر أو العديد من الحروف .
(_) UNDER SCORL يدل علي حرف واحد .

```
SELECT   first_name
FROM     employees
WHERE    first_name LIKE 'S%';
```

فى **المثال** السابق تم استخلاص الاسم الاول من جدول الموظفين، للموظفين الذين
يبدأ اسمائهم بحرف S . و هنا لا يشترط باقى الاسم و لابد هنا من استخدام كلمة LIKE .

```
SELECT SALARY FROM EMPLOYEES
WHERE
SALARY LIKE '%500';
```

SALARY
2500
6500
2500
2500
3500
2500
13500
10500
9500
7500
9500
7500
10500
9500

وهنا في المثال السابق نريد معرفة مرتبات الموظفين الذين لهم ثلاثة ارقام أخيرة 500.

```
SELECT *
FROM employees
WHERE last_name LIKE '_A% '
```

المثال هنا يطلب عرض بيانات كل الموظفين الذين لهم أسماء بها الحرف الثاني A حرف كبير والحرف الأول مجهول و باقي الحروف ايضا غير معلومة ولذا فإننا نلجأ لاستخدام (_) بدلا من الحرف الأول ثم نضع حرف A ثم % لتحل محل باقي الحروف المجهولة.

مثال اخر:

Enter statements:

```
SELECT FIRST_NAME FROM EMPLOYEES
WHERE
FIRST_NAME  LIKE '_t%'
```

FIRST_NAME
Steven
Steven
Stephen

المثال السابق يطلب عرض الاسم الاول للموظفين الذين لهم أسماء بها الحرف الثاني t حرف صغير والحرف الأول مجهول وباقي الحروف غير معلومة ايضا ولذا فإننا نلجأ لاستخدام (_) بدلا من الحرف الأول ثم نضع حرف t ثم % لتحل محل باقي الحروف المجهولة وهي (%).

```
SELECT last_name, hire_date
FROM employees
WHERE hire_date LIKE '%95';
```

فى **المثال** السابق تم استخلاص الاسم الاول و تاريخ التعين من جدول الموظفين للموظفين الذين تم تعينهم فى اى يوم او اى شهر بشرط ان يكون فى سنة 95.

عندما تريد البحث عن اسماء بها (_) UNDER SCORL
يمكن أن تستعمل الاتى:

```
SELECT employee_id, last_name, job_id
FROM employees
WHERE job_id LIKE '%SA\_%' ESCAPE '\';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID
149	Zlotkey	SA_MAN
174	Abel	SA_REP
176	Taylor	SA_REP
178	Grant	SA_REP

ففى **المثال** السابق يريد رقم الموظف و السم الاول و الوظيفة للموظفين الذين وظيفتهم اول حرفين منها SA و ثالث حرف (_).

استخدام شرط IS NULL
يستخدم هذا العامل لفحص القيمة (لا شئ) أي قيمة لا تحتوي علي بيانات

```
SELECT last_name, manager_id
FROM   employees
WHERE  manager_id IS NULL;
```

LAST_NAME	MANAGER_ID
King	

الشرط IS NULL يختار الخلايا التى لا يوجد بها بيانات اى انها NULL .
قيمة Null هي قيمة غير مخصصة غير متوفرة اى مجهولة، أو غير ملائمة.
إذن لا يمكن استخدام (=) مع Null .
فلا بمكن ان تقول select ename from emp where comm= null

فالمثال السابق يسترجع الأسماء و ارقام المديرين من جدول الموظفين الذين لا يملكون مدير.
(فهنا نجد ان KING هو مدير وليس يرأسه مدير و نجد ان خانة MANAGER_ID فارغة)
المثال التالى يعرض الأسم الأخير و وظيفتة و العمولة من جدول الموظفين الذين ليس لهم عمولة (بمعنى لا يأخذون عمولة) .

```
SELECT last_name, job_id, commission_pct
FROM employees
WHERE commission_pct IS NULL;
```

LAST_NAME	JOB_ID	COMMISSION_PCT
King	AD_PRES	
Kochhar	AD_VP	
...		
Higgins	AC_MGR	
Gietz	AC_ACCOUNT	

مجموعة من الشروط المنطقية

هناك ثلاث شروط منطقية يمكن استخدامها في جملة الشرط مع كلمة WHERE

العوامل	المعني
AND	تستخدم للربط بين شرطين وتعني تحقق الشرطين معا .
OR	تستخدم للربط بين شرطين يجب تحقيق أحدهما فقط .
NOT	تستخدم لنفي الشرط الذي يليها وهي تعني وجوب عدم تحقق هذا الشرط.

استخدام الشرط AND:

معامل AND يتطلب تحقق كلا من الشرطين معا.

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >=10000
AND    job_id LIKE '%MAN%';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
149	Zlotkey	SA_MAN	10500
201	Hartstein	MK_MAN	13000

المثال السابق يستخلص رقم الموظف و الاسم و الوظيفة و المرتب لكل الموظفين الذين يأخذون مرتب أكبر أو يساوى 10000 و فى نفس الوقت اسم الوظيفة يتضمن الحروف التالية MAN .

مثال اخر:

Enter statements:

```
SELECT FIRST_NAME,salary ,COMMISSION_PCT   FROM EMPLOYEES
WHERE
  salary>10000
and COMMISSION_PCT  is null
```

FIRST_NAME	SALARY	COMMISSION_PCT
Steven	24000	
Neena	17000	
Lex	17000	
Nancy	12000	
Den	11000	
Michael	13000	
Shelley	12000	

هنا في المثال نريد استعراض بيانات الموظفين الذين يتقاضون مرتبا اكبر من 10000

و لا يأخذون عمولة وهنا لابد من تحقيق الشرطين معا .

استخدام الشرط OR :

OR يتطلب ان يكون اى من الشرطين صحيح:

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >= 10000
OR     job_id LIKE '%MAN%';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
124	Mourgos	ST_MAN	5800
149	Zlotkey	SA_MAN	10500
174	Abel	SA_REP	11000
201	Hartstein	MK_MAN	13000
205	Higgins	AC_MGR	12000

8 rows selected.

وهنا في **المثال** السابق قمنا بتغير OR بدلا من AND وهذا يعني أننا نريد عرض بيانات الموظفين الذين يتقاضون راتب (أكبر من أو يساوي 10000) أو عرض بيانات الموظفين الذي يعملون فى وظيفة تحتوى على الحروف MAN .

وبالنظر في عمود المرتبات SALARY نجد أن الوظائف غير ST_MAN يقابلها مرتبات أكبر من 10000 .

استخدام الشرط NOT :

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id
       NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP');
```

LAST_NAME	JOB_ID
King	AD_PRES
Kochhar	AD_VP
De Haan	AD_VP
Mourgos	ST_MAN
Zlotkey	SA_MAN
Whalen	AD_ASST
Hartstein	MK_MAN
Fay	MK_REP
Higgins	AC_MGR
Gietz	AC_ACCOUNT

10 rows selected.

هذا **المثال** يوضح استخدام كلمة NOT ويؤدي إلي عكس الشرط بمعني أننا نريد في هذا المثال اختياركل الوظائف ماعدا الذين وظيفتهم 'IT_PROG','ST_CLERK', 'SA_REP' وفي الناتج تجد أن الوظائف لا تحتوي علي أي منهما من الثلاث وظائف السابقة.

مثال اخر

Enter statements:

```
SELECT FIRST_NAME   FROM EMPLOYEES
WHERE
FIRST_NAME not like'_t%';
```

FIRST_NAME
Neena
Lex
Alexander
Bruce
David
Valli
Diana
Nancy

هذا **المثال** يوضح استخدام كلمة NOT ويؤدي إلي عكس الشرط.
بمعني أننا نريد في هذا المثال الموظفين الذين لايحتوى ثانى حرف منهم على الحرف (t).

امثلة على BETWEEN, LIKE, and , null

... WHERE job_id NOT IN ('AC_ACCOUNT', 'AD_VP')
... WHERE salary NOT BETWEEN 10000 AND 15000
... WHERE last_name NOT LIKE '%A%'
... WHERE commission_pct IS NOT NULL

أولويات المعاملات المنطقية ومعاملات المقارنة

الأسبقية	المعامل
1	المعاملات الحسابية (الضرب و القسمة والجمع والطرح)
2	معامل || لصق الأعمدة (ضم الأعمدة)
3	معاملات المقارنة (= , > , >= , < , <= , < >)
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	المعاملات المنطقية 1- (NOT) 2- (AND) 3- (OR)

يمكن أن تتجاوز الاولوية فى ترتيب العوامل باستخدام الأقواس حول التعبيرات التي تريدها أن تتنفذ أولا .

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id = 'SA_REP'
OR     job_id = 'AD_PRES'
AND    salary > 15000;
```

LAST_NAME	JOB_ID	SALARY
King	AD_PRES	24000
Abel	SA_REP	11000
Taylor	SA_REP	8600
Grant	SA_REP	7000

تجاوز قواعد الأسبقية باستخدام الأقواس .
فى هذا المثال يوجد شرطان يتم تنفيذ احداهما بسبب استخدام OR .
* الشرط الأول أن تكون الوظيفة SA_REP .
* الشرط الثاني أن تكون الوظيفة AD_PRES و فى نفس الوقت يحصل على مرتب اكبر من 15000 .

استخدام الأقواس لالغاء الأولوية

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  (job_id = 'SA_REP'
OR     job_id = 'AD_PRES')
AND    salary > 15000;
```

LAST_NAME	JOB_ID	SALARY
King	AD_PRES	24000

استخدام الأقواس للتحكم في تحقيق الشروط والأولوية تكون للأقواس .
فهنا يتم تنفيذ OR قبل AND .
ففى المثالِ السابق ، هناك شرطان :
* الشرط الأول بأن تكون الوظيفة اما SA_REP او AD_PRES .
* الشرط الثاني أن يكون الراتب أكبر من 15,000 .

الفرز والترتيب ORDER BY

لترتيب نتيجة أمر SELECT نقوم باستخدام كلمة ORDER BY و لاحظ انها توضع في آخر أمر SELECT ثم يليها أسم العمود الذي يراد الترتيب به .
*فرز الصفوف بواسطة فقرة ORDER BY
ASC: الترتيب التصاعدي و لا يشترط كتابتة حيث انة يعتبر DEFAULT .
DESC: الترتيب التـــــنازلي (الانحدار) .

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY hire_date ;
```

LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
King	AD_PRES	90	17-JUN-87
Whalen	AD_ASST	10	17-SEP-87
Kochhar	AD_VP	90	21-SEP-89
Hunold	IT_PROG	60	03-JAN-90
Ernst	IT_PROG	60	21-MAY-91

...

20 rows selected.

<u>ملحوظة هامة :-</u>

هنا في المثال لم يحدد DESC & ASC وفي هذه الحالة يتم بالترتيب التصاعدى .
لترتيب نتيجة الأمر SELECT نقوم باستخدام كلمة ORDER BY وهي توضع في آخر جملة SELECT ثم يليها اسم العمود المراد الترتيب به وهنا هو hire_date وعند استخدام الحقل تكون النتيجة كما نري أمامنا.
أنتَ يمكن استخدام الاسم المستعار (ALIAS) فى جملة ORDER BY .

```
FROM table
[WHERE condition(s)]
[ORDER BY {column, expr} [ASC|DESC]];
```

ORDER BY تحدد أى الصفوف المسترجعة التي سوف يتم الترتيب بها.
ASC ترتيب تصاعدِى.
DESC ترتيب تنازلي.

الترتيب بشكل تنازلي

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY hire_date DESC ;
```

LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
Zlotkey	SA_MAN	80	29-JAN-00
Mourgos	ST_MAN	50	16-NOV-99
Grant	SA_REP		24-MAY-99
Lorentz	IT_PROG	60	07-FEB-99
Vargas	ST_CLERK	50	09-JUL-98
Taylor	SA_REP	80	24-MAR-98
Matos	ST_CLERK	50	15-MAR-98
Fay	MK_REP	20	17-AUG-97
Davies	ST_CLERK	50	29-JAN-97

ففي **المثال** السابق يقوم بعرض الاسم و الوظيفة و رقم القسم و تاريخ التعين

و يقوم بترتيب الناتج بعمود تاريخ التعين (تنازلى) .
* قيم الارقام تعرض من الأصغر الى الأكبر كمثالِ 1.....999 .
* قِيَم التاريخ تعرض بالقيمةِ الأقرب أولاً ثم الابعد كمثال :
(01 يناير 92) قبل (01 يناير 95) .
* قيم الحروف تعرضُ بترتيب الأبجدية كمثال : A ثم B وهكذا الى ..... Z.
* قيمة Null تعرض أخيراً في الترتيب التصاعدي وتعرض أولا في الترتيب التنازلي .

■ لكى تعكس الصفوف التي تعرض، يحدد كلمة DESC بعد أسم العمود في فقرة ORDER BY. و بهذة الكلمة يتم عكس الكلام السابق كلة .

ملاحظة: يمكنك أن تفرز برقم العمود في القائمة SELECT. فالمثال التالي يوضح ذلك حيث يتم الترتيب تنازلي للمرتبات ذلك عن طريق تحديد رقم العمود المراد الترتيب بة .

```
SELECT last_name,
                                                    salary
FROM employees
ORDER BY 2 DESC;
```

الترتيب بالاسم المستعار (ALIAS)

```
SELECT employee_id, last_name, salary*12 annsal
FROM   employees
ORDER BY annsal;
```

EMPLOYEE_ID	LAST_NAME	ANNSAL
144	Vargas	30000
143	Matos	31200
142	Davies	37200
141	Rajs	42000
107	Lorentz	50400
200	Whalen	52800
124	Mourgos	69600
104	Ernst	72000
202	Fay	72000
176	Grant	84000

...

20 rows selected.

في هذا المثال قمنا بضرب الراتب الشهري × 12 وبذلك نحصل علي الراتب السنوي واعطيناة اسم annsal .
وكما نري تم وضع annsal بعد Order by .

استخدام أكثر من عمود في الترتيب

```
SELECT last_name, department_id, salary
FROM   employees
ORDER BY department_id, salary DESC;
```

LAST_NAME	DEPARTMENT_ID	SALARY
Whalen	10	4400
Hartstein	20	13000
Fay	20	6000
Mourgos	50	5800
Rajs	50	3500
Davies	50	3100
Matos	50	2600
Vargas	50	2500

...

يمكن استخدام اكثر من عمود في الترتيب بحيث يوضع العمود الأول ثم فصلة (,) ثم العمود الثاني وفي المثال وضعنا رقم الإدارة ثم المرتب بحيث يتم الترتيب تصاعدياً علي رقم الإدارة و فى نفس الوقت يقوم بترتيب المرتب تنازليا وذلك يكون بالنسبة لكل ادارة.

* فاذا نظرنا الى عمود الاقسام فى القسم رقم 20 وفى نفس الوقت نظرنا الى عمود المرتب فنجد ان المرتب بالنسبة للقسم 20 تم ترتيبة تنازليا وهكذا بالنسبة للباقى.

- يمكن أن تستخدم ORDER BY بعمود ليس في قائمة SELECT

```
SELECT last_name, salary
FROM employees
ORDER BY department_id, salary DESC;
```

هنا في **المثال** حيث يمكننا الترتيب باستخدام اسم عمود ليس موجود ضمن القائمة الأعمدة في جملة SELECT فكما نري قد تم الترتيب باستخدام العمود department_id مع إنه غير موجود بالقائمة .

الشكل التالى يوضح كيفية عمل ترتيب (تصنيف) على عدة صفوف

- **The order of ORDER BY list is the order of sort.**

```
SELECT last_name, department_id, salary
FROM   employees
ORDER BY department_id, salary DESC;
```

LAST_NAME	DEPARTMENT_ID	SALARY
Whalen	10	4400
Hartstein	20	13000
Fay	20	6000
Mourgos	50	5800
Rajs	50	3500
Higgins	110	12000
Gietz	110	8300
Grant		7000

20 rows selected.

- **You can sort by a column that is not in the SELECT list.**

فهو يبدأ اولا بالترتيب على العمود الاول فى عبارة ORDER BY اذا حدث تكرار يرتب التكرارت على حسب العمود الثانى وهكذا.

ففى **المثال** يرتب الموظفين على اساس اداراتهم ثم يرتب الموظفين داخل نفس الادارة على اساس مرتباتهم تنازلياً.

الأهداف:

بعد إكمال هذا الدرس يجب أن تكون قادر بإذن الله علي عمل التالي:

1. تقديم الاستعلام الذي يتطلب بديل متغير.
2. حول (تفصيل) بيئة iSQL*Plus .
3. تقديم مخرجات أكثر قراءة.
4. ينشئ وينفذ ملفات نصية.

هدف درس:

من هذا الدرس سوف تتعلم الأوامر iSQL*Plus أن تتضمن إنتاج أكثر قراءة من مخرجات SQL .
ويمكنك أن تنشئ ملف الأوامر الذي يحتوي علي فقرة WHERE كي تحدد الصفوف المعروضة. كي تغير الشرط كل وقت لتشغيل ملف الأوامر،يجب أن تستخدم متغيرات البديل.
متغيرات البديل يمكن أن تبدل قيم في فقرة WHERE،و مجموعة النصوص، وحتى عمود أو أسم جدول .

متغيرات ال بديل(Substitution)

المستخدم 6
يريد الاستعلام للقيم المختلفة.
المرتب = ؟ رقم القسم = ؟ الاسم الأخير = ؟
الأمثلة إلي حد الآن قد كانت منظمة حسب قواعد معينة مع نهاية التطبيق المستخدم سوف يسبب ذلك كتابة التقرير.
والتقرير يجب أن يشغل بدون ادخال متغيرات و طلب اى بيانات من المستخدم (تذكير) آخر.
مجموعة البيانات ستكون محددة مسبقا بفقرة WHERE الثابتة في الملف النصي ل iSQL*Plus
واستخدام iSQL*Plus، أنت يمكن أن تنشئ تقارير التي تتعامل مع المستخدم بحيث يجهز قيمهم الخاصة كي تحدد مجموعة من البيانات التي استرجعت باستخدام متغيرات البديل(Substitution). يمكنك أن تضمن متغيرات البديل في ملف الأوامر أو في تعبير SQL.
متغيرات يمكن أن توضع مع القيم التي تخزن بشكل مؤقت. و عند تشغيل التعبير، تجد أن القيم سوف تستبدل .

متغيرات البديل

فوائد استخدام ISQL*Plus متغيرات البديل :

* تكوين مخزن للبيانات بشكل مؤقت. وذلك باستخدام:
- علامة & المفردة
- علامة && المضاعفة
- أمر DEFINE
* تمرير (ارسال) القيم المتغيرة بين تعبيرات SQL.
* يعدل فاعلية ما يكتب في أعلي الصفحة و ما يكتب في أسفل الصفحة.

فى iSQL*Plus، يمكنك أن تستخدم علامة & المفردة و متغيرات البديل كي تخزن قيم بشكل مؤقت .
يمكنك أن تعرف المتغيرات في iSQL*Plus باستخدام الأمر DEFINEE.
و تخصيص قيم إلي هذه المتغيرات.

أمثلة للبيانات المحصورة

* أشكال التقرير فقط للربع الحالي أو لفترة من التاريخ المحدد.
* يكتب التقرير إلي المستخدم عن البيانات المتعلقة فقط عن الموضعات المطلوب فيها تقرير فقط.

تأثيرات تفاعلية أخري
تأثيرات تفاعلية لا تحدد بالمعلومات المستخدمة المباشرة مع الفقرة WHERE.
نفس المبادئ يمكن أن تنجز أهداف أخري. كمثال :

1. يعدل فاعلية ما يكتب في أعلي الصفحة و ما يكتب في أسفل الصفحة
2. يحصل علي قيم المدخلة من ملفِ بدلاً من شخص .
3. مرور (ارسال) القِيَم المتغيرةِ من التعبيرات SQL الواحدة إلي الآخر.

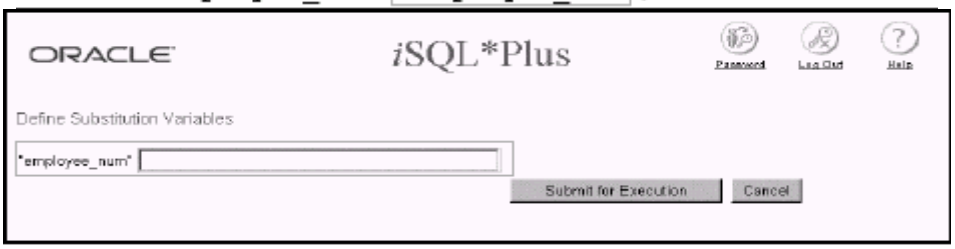
• ISQL*Plus لا يدعم تدقيق المصادقة علي إدخال المستخدم فيما عدا أنواع البيانات) .

ملاحظات

البديل المتغير يمكن أن يستخدم في أي مكان في أوامر SQL , iSQL*Plus ، ماعدا كالكلمة الأولى التي أدخلت في الأمر السريع .

استخدام المتغير البديل علامة ← &
استخدم المتغير في المقدمة بعلامة & لسرعة المستخدم إلي القيمة .

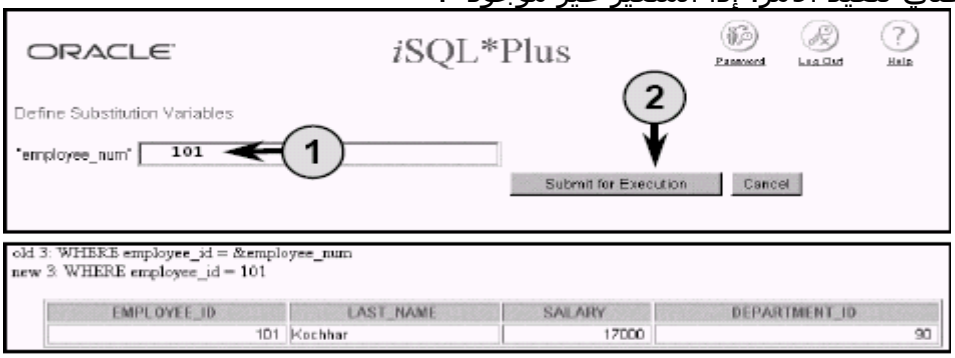
```
SELECT   employee_id, last_name, salary, department_id
FROM     employees
WHERE    employee_id = &employee_num ;
```



عندما يتم تشغيل التقرير، المستخدمون يريدون غالباً أن يحددوا البيانات التي استرجعت بشكل ديناميكيا.
iSQL*Plus تزود هذه المرونةِ بواسطة متغيرات المستخدم.
استخدم علامة & أن تميز كل متغير في تعبير SQL. ولكنك لست بحاجة إلي أن تعرف قيمة كل متغير.

ترقيم	الوصف
&user_variable	المتغير في تعبير SQL يشر إلي ؛ إذا لا يوجد المتغير ، iSQL*Plus تحث المستخدم علي القيمة (iSQL*Plus يرمي المتغير الجديد مرة حيث إنه تم استخدامه)

المثال ينشئ iSQL*Plus متغير بديل لرقم الموظف. عندما الأمر ينفذ iSQL*Plus يحث المستخدم علي رقم الموظف وبعد ذلك يعرض عدد الموظف، الاسم الأخير، الراتب، ورقم القسم لذلك الموظف . بعلامة & الوحيدة، المستخدم يحث كل وقت علي تنفيذ الأمر، إذا المتغير غير موجود .



متغير بديل علامة & المفرد
عند ملاحظة iSQL*Plus أن التعبير SQL يحتوي علي علامة & أنت تحث علي إدخال القيمة لاسم المتغير البديل في الأمر SQL. سوف تدخل القيمة مرة وتنقر علي زر تنفيذ العرض وهو رقم (2) أعلي في الشكل المعروض، النتائج تَعرض في مساحة المخرجات بدورة SQL*Plus .

تحديد الحرف وقيم التاريخ بمتغيرات البديل

استخدم علامات تقدير' ` لتحديد الحرف وقيم التاريخ

```
SELECT last_name, department_id, salary*12
FROM   employees
WHERE  job_id = '&job_title' ;
```

LAST_NAME	DEPARTMENT_ID	SALARY*12
Hunold	60	108000
Ernst	60	72000
Lorentz	60	50400

من فقرة WHERE، قيم التاريخ والحرف يجب أن يكونا مرفقين ضمن علامة التقدير ' ' نفس القاعدة تقدم لمتغيرات البديل .
أرفق المتغير في علامات التقدير ' ' ضمن تعبير SQL نفسه .
يظهر لنا الاستعلام كي يسترجع أسماء الموظف، رقم القسم، ورواتب سنوية لكل الموظفين مستندة علي تحديد العنوان الوظيفي لمتغير البديل في iSQL* Plus

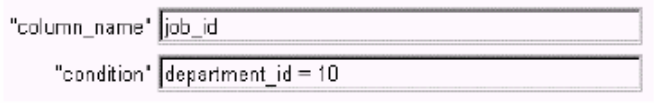
الملاحظة: يمكنك أن تستخدم الدوال مثل UPPER و LOWER مع علامة &.
الاستخدام UPPER('&job_title') وذلك لأن ليس من حق المستخدم أن يدخل العنوان الوظيفي بحروف كبيرة.

تحديد أسماء العمود ،التعبيرات، ونص استخدام متغيرات البديل أن تكمل التالي :

1. شرط WHERE
2. فقرة ORDER BY
3. تعبيرات عمود
4. أسماء جداول
5. ادخل تعبير SELECT

متغيرات البديل لا تستخدم فقط مع فقرة WHERE في تعبير SQL، لكن هذه المتغيرات يمكن أيضا أن تستخدم في آستبدال لأسماء العمود، تعبيرات، أو نص .
المثال:أعرض رقم الموظف وأي عمود آخر وأي أمر للموظفين .

```
SELECT  employee_id, &column_name
FROM    employees
WHERE   &condition;
```



EMPLOYEE_ID	JOB_ID
200	AD_ASST

إذا لم تدخل قيمة المتغيرة للبديل، سوف تحصل علي خطأ عندما تنفذ التعبير السابِق .
الملاحظة: متغير البديل يمكن أن يستخدم في أي مكان في الأمر SELECT، ماعدا كالكلمة الأولي المدخلة في سرعة الأمر .

```
SELECT      employee_id, last_name, job_id,
            &column_name
FROM        employees
WHERE       &condition
ORDER BY    &order_column ;
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
102	De Haan	AD_VP	17000
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000

يعرض **المثال** رقم الموظف، أسم، عنوان وظيفة، وأي عمود آخر حدد من قبل المستخدم في وقت التشغيل، من جدول الموظفين. أنت يُمكن أن تحدد الشرط أيضا لاسترجاع صفوف وأسم العمود مِن قبل البيانات الناتجة التي يجب أن تؤمر .

تعريف متغيرات البديل

* يمكنك تعريف المتغيرات في iSQL*Plus باستخدام أمر DEFINEE.
DEFINEE *variable= Value* إنشاء المتغير المستخدم مع نوع البيانات CHAR.
إذ احتجت أن تعرف متغير يتضمن فراغ NULL ، فيجب أن ترفق القيمة ضمن علامة التقدير ' ' عندما تستخدم أمر DEFINEE.
* متغير معرف ويكون متوفر للدورة.
يمكنك تعريف المتغيرات باستخدام تنفيذ جملة الاستعلام البسيط .
iSQL* Plus مزودة بأمر DEFINEE لتعريف و وضع متغيرات البديل **:**

الأمر	الوصف
DEFINEE *variable = value*	إنشاء المتغير المستخدم مع نوع البيانات CHAR ويخصص قيم إليه
DEFINEE *variable*	عروض المتغير، قيمته، وأنواع بياناته
DEFINEE	عروض كل متغيرات المستخدم بقِيمهم وبياناتهم

أوامر DEFINE , UNDEFINE

* تعريف بقايا متغيرة حتى إنك إما :
- تستخدم أمر UNDEFINE أن يوضحة
- تخرج من (تغلقها) iSQL*Plus
* يمكنك من مع أمر DEFINE.

عندما لا تحدد المتغيرات، حيث أنك يمكن أن تحقيق تغييراتك مع أمر DEFINE.
عندما تغلق نافذة iSQL*Plus،المتغيرات المحددة خلال تلك الدورة تفقد .

استخدام أمر DEFINE مع متغير بديل للعلامة &
*أنشئ متغير معدل مستخدم أمر DEFINE:
DEFINEE employee_num = 200
* استخدام متغير قدم بعلامة & إلي القيمة البديل في تعبير SQL

```
SELECT employee_id, last_name, salary, department_id
FROM employees
WHERE employee_id = &employee_num ;
```

EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
200	Whalen	4400	10

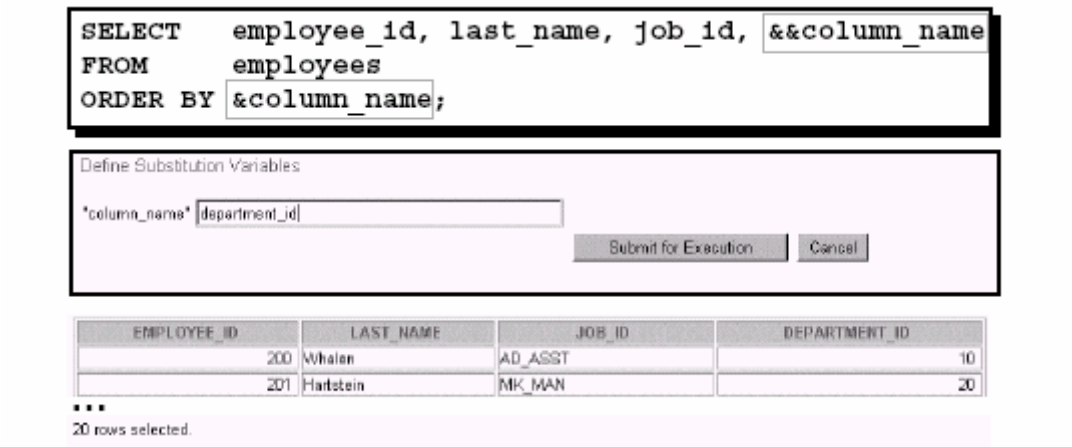
المثال علي iSQL*Plus إنشاء متغير بديل لرقم الموظف بواسطة الأمر DEFINE ، و عروض وقت التشغيل لرقم الموظف، أسم الموظف، راتب الموظف، رقم القسم لذلك الموظف .

لأن المتغير أنشأ مستخدما الأمر DEFINE في iSQL*Plus ، المستخدم لا يحث علي إدخال القيمة لرقم الموظف. بدلاً من، القيمة المتغيرة المعرفة تستبدل آلياً في التعبير SELECT.

رقم الموظف بديل متغير حاضر في الدورةِ حتى إذا كان المستخدم غير معرف في الدورة و خروجه من دورة iSQL*Plus

استخدام المتغير البديل للعلامتين &&

استخدام علامتان من && عندما تريد استخدام ثاني قيمة متغير بدون حث المستخدم علي كل الوقت.(يقوم بسؤال الستخدم عن قيمة المتغير فى المرة الاولى للتنفيذ فقط وبعدها فى تنفيذات التالية يعمل تلقائيا على القيمة المدخلة فى اول مرة بدون سؤال المستخدم مرة اخرى)



يمكنك من استخدام متغير البديل للثنائي العلامة علامتان (&&) عندما تريد أن تستعمل ثاني قيمة من القيمة المتغيرة بدون حث المستخدم كل وقت.

المستخدم سيري القيمة مرة فقط. من المثال ، المستخدم يستعلم كي يعطي قيمة المتغير لاسم العمود مرة فقط.

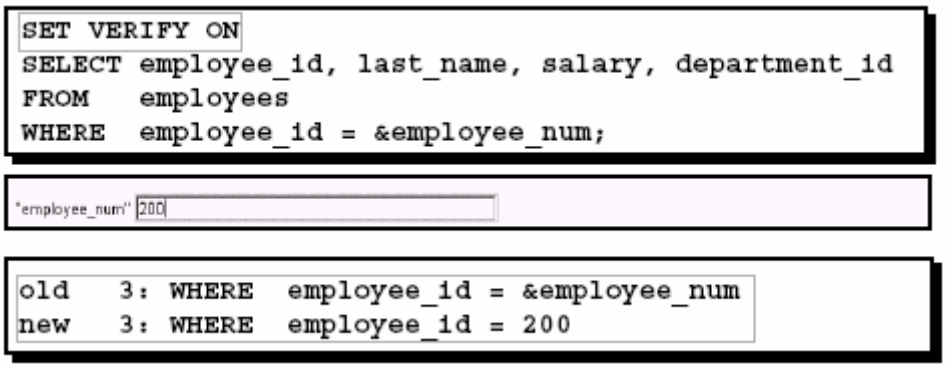
القيمةُ جهزت مِن قِبل المستخدم (رقم القسم) تستخدم كلتا للعرض وأوامر البيانات .

ISQL*Plus يخزن القيمة المجهزة باستخدام أمر DEFIN؛ هو سيستخدمه ثانية حينما تسترجع الاسم المتغير. مرة مع المستخدم المتغير في المكان الصحيح،وهنا سوف تحتاج أن تستخدم أمر UNDEFINE كي تلغيها.

استخدام الأمر VERIFY (حقق)

استخدام الأمر VERIFY كي تثبت العرض عن المتغير البديل قبل وبعد iSQL*Plus بديلات متغيرات البديل مع القيم .

وهو يقوم بعملية متابعة تغيرات قيم المتغيرات داخل التنفيذ.وذلك مفيد فى معرفة سير البرنامج.



أن تؤكد التغيرات في تعبير SQL، استخدام أمر VERIFY في iSQL*Plus مجموعة SET VERIFY تقوي من iSQL*Plus ليعرض أمر نص قبل وبعده كي تبدل متغيرات البديل مع القيم.

المثال علي القيمة القديمة وأيضا الجديدة لعمود رقم الموظف .

حول (تفصيل) بيئة iSQL*Plus
* استخدام أوامر SET كي تتحكم في الدورة الحالية.
SET system_variable VALUE
* يحقق ما وضعته جانبا باستخدام أمر SHO.

```
SET ECHO ON
```

```
SHOW ECHO
echo ON
```

الفصل الثالث

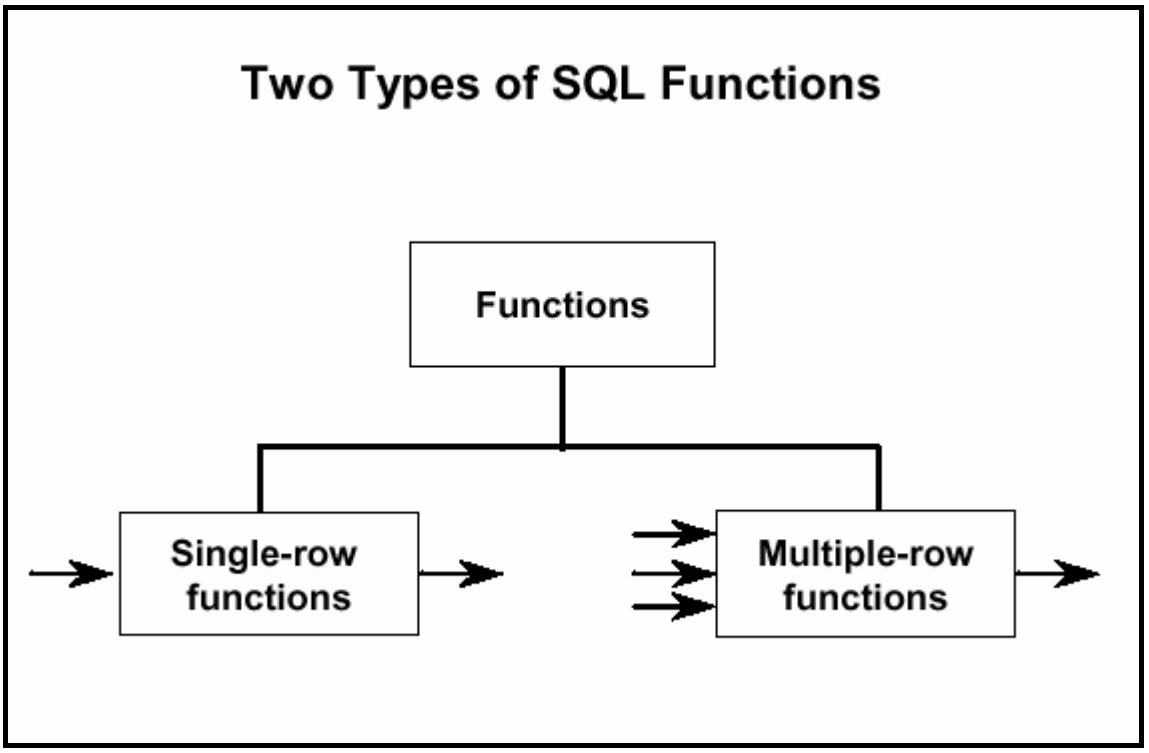
Single row function

سيتم التعرف في هذا الفصل إن شاء الله تعالي علي إمكانيات وكيفية استخدام بعض function .

هدف الدرس :
القدرة على التعامل مع الأنواع المتعددة من function المتوفرةِ في SQL .

هناك نوعان من FUNCTION

1. SINGLE ROW FUNCTION .
2. MULTIPLE ROW FUNCTION.



:SINGLE ROW FUNCTION
وهي تعمل علي بيانات الصف الواحد اى اننا نقوم بادخال صف واحد من البيانات ونقوم بعمل عملية معينة علية ثم تقوم باخراج صف واحد ايضا.
وهذا سوف يَغطّي في هذا الفصل.

:MULTIPLE ROW FUNCTION
هي تعمل علي عدة صفوف معا وتقوم بإخراج نتيجة واحدة لتلك الصفوف.
وهذا سوف يَغطّي في الفصل الخامس .

:SINGLE ROW FUNCTION
1. تتعامل مع اجزاءَ من البيانات.
2. يعمل علي ارجاع قيمة واحدة لكل صف.
3. يُعدّلُ فى شكل عرض البياناتَ .
4. يَقْبلُ (argument) التي يمكن ان تَكُونَ عمودَ أو تعبير.

الشكل العام للصيغة :

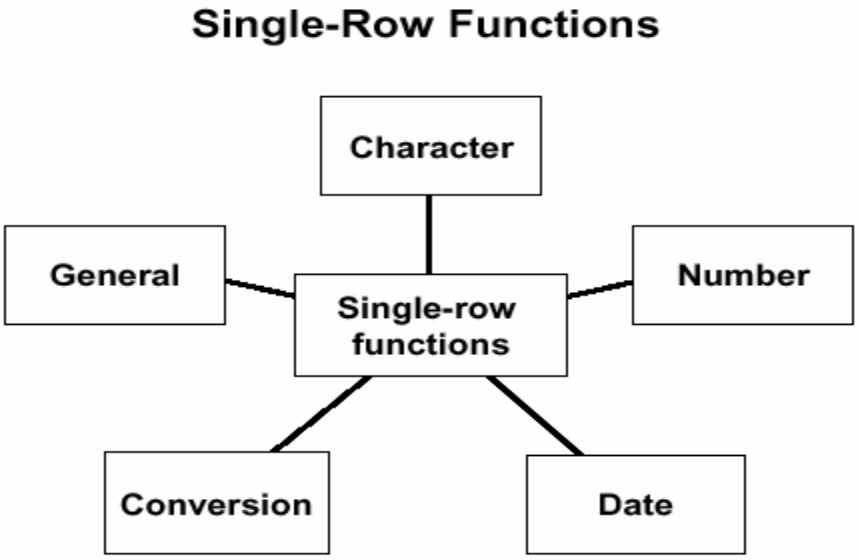
```
function_name [(arg1, arg2,...)]
```

function_name: اسم (function) المستخدمة.
arg 1, arg 2): (argument هذا يُمكنُ أنْ يكون أسم عمودِ أو قيمة .

مميزاتها :

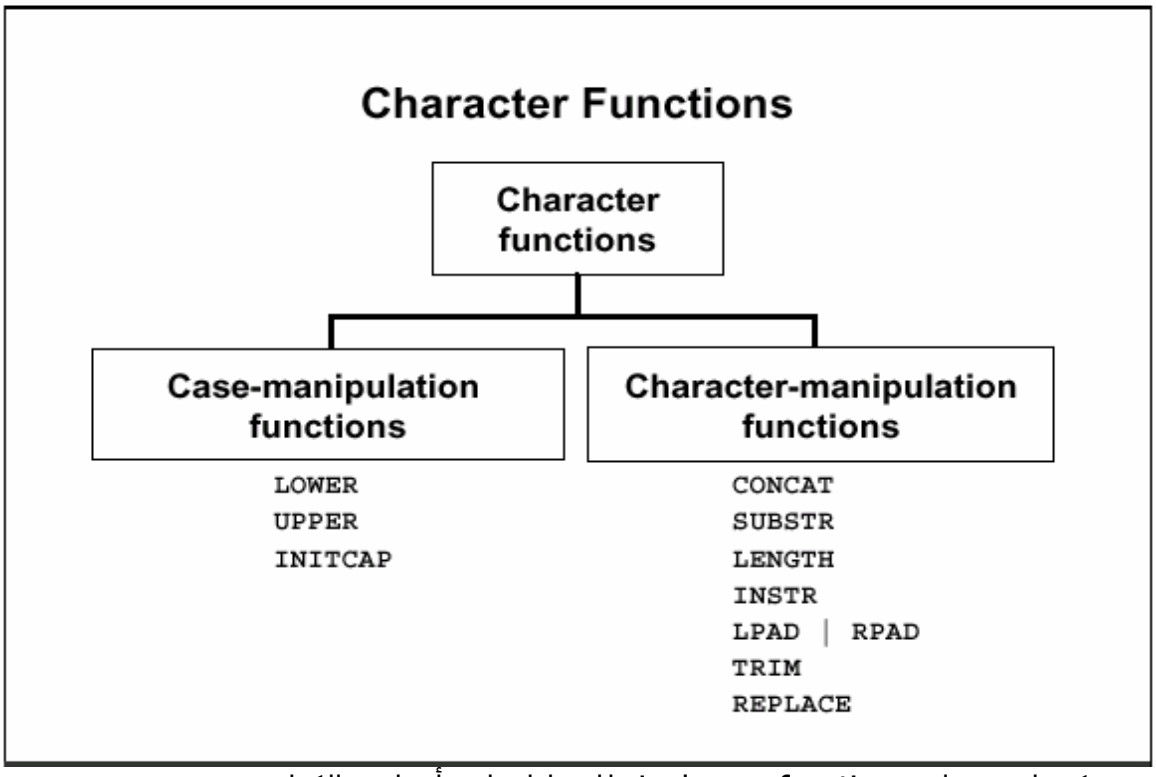
1. تعمل علي استرجاع لكل صف .
2. استرجاع نتيجة واحدة بواسطة الصفوف.

- Single row function يمكن أن تستخدم في جملة SELECT او WHERE او فى فقرة ORDER BY.



الشكل السابق يوضح انواع single row function .

اولاً : CARACTER FUNCTION .



يمكن استخدام single row function للحفاظ علي أسلوب الكتابة.
يمكن استخدام single row function لمعرفة طول أو حجم الكلمات .

الدوال	الغرض

LOWER(*column / expression*)	تقوم بتحويل الحروف إلي حروف صغيرة بغض النظر عن طريقة كتابتها.
UPPER(*column /expression*)	تقوم بتحويل الحروف إلي حروف كبيرة بغض النظر عن طريقة كتابتها.
INITCAP(*column /expression*)	تقوم بتحويل الحرف الأول فقط إلي حرف كبير وبقية الكلمة حروف صغيرة.
CONCAT(*column1/expression1, column2/expression2*)	هي تقوم بضم عمودين أو كلمتين في كلمة واحدة ويلاحظ انها نفس (Concatenation) || التي كانت في البداية مع الفارق أن استخدامها لا يسمح بالتعامل مع أكثر من عمودين .
SUBSTR(*column / expression ,m [,n]*)	تقوم بأخذ جزء من حروف الكلمة أو العمود المراد استخدامه . فالمعامل m يدل علي موقع الحرف الذي سوف نبدأ من عنده. والمعامل n يدل علي عدد الحروف المطلوب ان يصل اليها . لو m بالسالب فالعد سوف يبدأ من نهاية الكلمة او العمود . ولو n محذوفة فهذا يعنى انة سوف يصل الى اخر حرف بالكلمة او بالعمود.
LENGTH(*column /expression*)	تقوم بعرض عدد الحروف للعمود او الكلمة.
INSTR(*column / expression ,'string' , [,m], [n]*)	تقوم بتحديد رقم موقع حرف معين داخل الكلمة او العمود.
LPAD(*column |expression , n, 'string'*) RPAD(*column |expression , n ,'string'*)	LPAD تقوم بضبط عرض العمود بإزاحة الكلمة من اليسار إلي اليمين. (LEFT PAD) وهي RPAD تقوم بضبط عرض العمود بإزاحة الكلمة من اليمين إلي اليسار. (RIGHT PAD)ا وهي
TRIM(*character / expression FROM column*)	تقوم بحذف حرف معين من كلمة او عمود و يظهر الناتج بعد ذلك وقد حذف منه تلك الحرف المحدد .
REPLACE(*text , search_string ,replacement_string*)	يبحث علي حرف معين من مجموعة من الحروف و لو وجدة يعمل له تغيير بالحرف المراد وضعة بدلا منة .

دوال معالجة الحالة

FUNCTION	النتيجة	الوصف
1- LOWER('SQLCourse') 2-PPER('SQLCourse') 3-INITCAP('SQL Course')	1- sqlcourse 2-SQL COURSE 3-Sql Course	1- تحويل الحروف كلها إلي حروف صغيرة 2- تحويل الحروف كلها إلي حروف كبيرة 3- تحويل أول حرف فقط إلي حرف كبير والباقي حروف صغيرة

```
SELECT 'The job id for ' || UPPER(last_name) || ' is '
|| LOWER(job_id) AS "EMPLOYEE DETAILS"
FROM employees;
```

EMPLOYEE DETAILS
The job id for KING is ad_pres
The job id for KOCHHAR is ad_vp
The job id for DE HAAN is ad_vp
...
The job id for HIGGINS is ac_mgr
The job id for GIETZ is ac_account

20 rows selected.

في المثال السابق تم استعراض الاسم بحروف كبيرة و بالتالي ظهر KING و KOCHHAR و HANN و HIGGINS و GITZ بحروف كبيرة عكس الأول فكانت تري King أي الحرف الأول كبير فقط و Kochhar و هكذا ولكن بعد وضع UPPER قبل اسم العمود تم تغير الحروف كلها و أصبحت حروف كبيرة (Capital).
و لاحظنا استخدام LOWER قبل اسم عمود الوظيفة تم تغير الحروف كلها و أصبحت حروف صغيرة (Small).
و لاحظنا ايضا استخدام (Concatenation) فقد تم دمج جملة (The job id for) مع عمود الاسماء ثم بعد ذلك تم دمج كملة (is) مع عمود الوظيفة.

ملحوظة هامة: عندما تريد اضافة كلمة او جملة على عمود معين يجب وضعها بين (Single quotation) كما حدث فى المثال السابق وقد تم تسمية تلك العمود باسم مستعار (Alias) و استخدم (Dabble quotation) لان الاسم المستعار الذى اختارة كان به مسافة.

مثال اخر:

Enter statements:

```
select lower(FIRST_NAME) as asmal,upper(FIRST_NAME) as captial
from employees
```

ASMAL	CAPTIAL
steven	STEVEN
neena	NEENA
lex	LEX
alexander	ALEXANDER
bruce	BRUCE
david	DAVID
valli	VALLI
diana	DIANA
nancy	NANCY
daniel	DANIEL
john	JOHN
ismael	ISMAEL
jose manuel	JOSE MANUEL
luis	LUIS

نرى الناتج من جملة select والفرق بين lower و upper

```
SELECT employee_id, last_name, department_id
FROM   employees
WHERE  last_name = 'higgins';
no rows selected
```

فى المثال السابق يريد عرض رقم الموظف و الاسم و رقم القسم للموظف Higgins. و لاحظنا انة لم يجد تلك الموظف مع انة موجود وهذا بسبب ان ذلك الموظف مسجل باحرف كبيرة (Capital) فهنا لابد من تحويل تلك الاسم الى احرف صغيرة (Small) كما تم تعديل تلك المثال فيما يلى.

```
SELECT employee_id, last_name, department_id
FROM   employees
WHERE  LOWER(last_name) = 'higgins';
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
205	Higgins	110

```
SELECT employee_id, UPPER(last_name), department_id
FROM employees
WHERE INITCAP (last_name) = 'Higgins';
```

ملاحظة:

(INITCAP) تحول اول حرف الى capital والباقى يظل small

هذه Function لمعالجة مجموعة الحروف

الدوال	النتيجة	الوصف
CONCAT('Hello','World')	HelloWorld	تقوم بضم عمودين أو قيمتين معا
SUBSTR('HelloWorld',1,5)	Hello	وهنا تقوم بعرض حروف الكلمة من بداية الحرف H حيث ذكر رقم 1 حتى الحرف رقم 5
LENGTH('HelloWorld')	10	--تقوم بعرض عدد الحروف و عددهم 10 حروف.
--INSTR('HelloWorld', 'W') --INSTR('HelloWorld', 'o',1,2)	6	--تقوم بتحديد موقع حرف W داخل الكلمة وهي موقعها رقم 6. • نلاحظ فى المثال الثانى انة يريد معرفة موقع حرف (O) الثانى ولذلك اضفنا 1 بمعنى انة يتجاهل الاول ثم ياخذ الثانى
LPAD(salary,10,'*')	*****24000	تقوم بمحاذاة البيانات جهة اليسار حيث يتم ملئ حرف معين كالنجمة * مثلا وتكون يسار البيانات.
RPAD(salary, 10, ' #')	24000#####	تقوم بمحاذاة البيانات جهة اليمين حيث يتم ملئ حرف معين مثلا # وتكون يمين البيانات.
TRIM('H' FROM'HelloWorld')	elloWorld	يقوم بحذف حرف H من الكلمة.
LTRIM(salesman ,'sale')	man	يقوم بحذف sale من الكلمة
RTRIM (salesman ,'man')	sales	يقوم بحذف man من الكلمة

امثلة اخرى :

Enter statements:

```
select lower('GOOD BY') FROM DUAL;
seLECT UPPER ('GOOD BY') FROM DUAL;
SELECT INITCAP ('GOOD  BY') FROM DUAL;
SELECT CONCAT ('GOOD','BY') FROM DUAL;
SELECT SUBSTR('GOOD BY',2,3) FROM DUAL;
SELECT LENGTH('GOOD') FROM DUAL;
SELECT LPAD('AHMED',10,'*') FROM DUAL;
SELECT TRIM ('S'FROM 'SAML') FROM DUAL;
```

LOWER('
good by

UPPER('
GOOD BY

INITCAP(
Good By

CONCAT
GOODBY

SUB
OOD

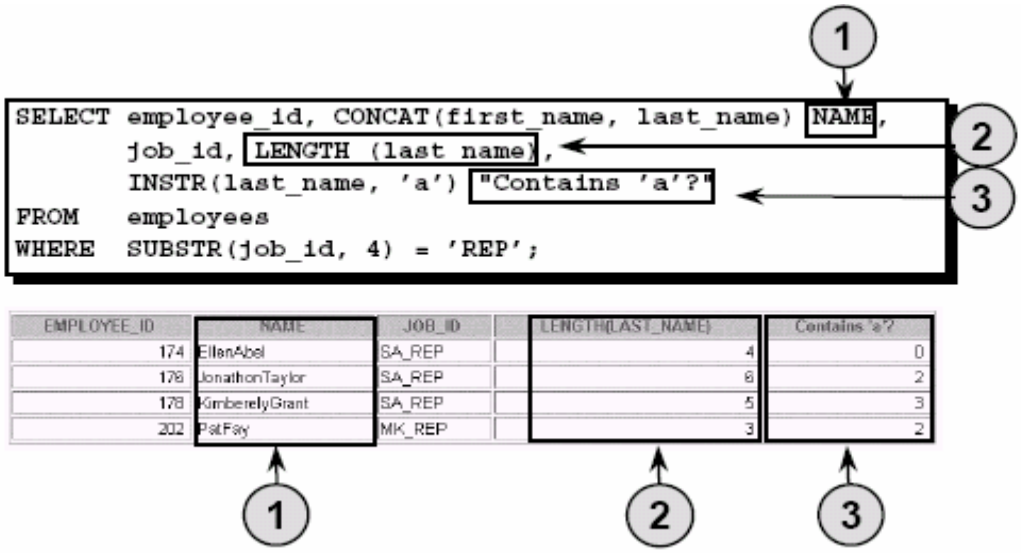
LENGTH('GOOD')
4

LPAD('AHME
*****AHMED

TRI
AML

ملاحظة:

الجدول (DUAL) هو جدول وهمى خاص للوراكل يستخدم لاجراء العمليات التى لا يدخل فيها اى جدول من داخل قاعدة البيانات .



المثال السابق يعرض الأسماء الأولي مع الأسماء الأخيرة عن طريق Concat ربطا معا، ودالة LENGTH للأسماء الأخيرة للموظفين (أي عدد الحروف للاسم الأخير). وتحديد موقع حرف a.و ذلك للموظفين الذين يحتوى اسم وظيفتهم من الحرف الرابع الى الاخر على الحروف التالية (REP).

```
SELECT employee_id, CONCAT(first_name, last_name) NAME,
LENGTH (last_name), INSTR(last_name, 'a') "Contains 'a'?"
FROM employees
WHERE SUBSTR(last_name, -1, 1) = 'n';
```

EMPLOYEE_ID	NAME	LENGTH(LAST_NAME)	Contains 'a'?
102	LexDe Haan	7	5
200	JenniferWhalen	6	3
201	MichaelHartstein	9	2

فى المثال السابق يقوم بعرض رقم الموظف و قام بدمج الاسم الاول مع الاسم الاخير واعطى لهذا العمود اسم مستعار وهو (NAME) وقام بتحديد عدد احرف LAST_NAME وتحديد موقع حرف(a) من LAST_NAME.
وذلك فقط للموظفين الذين تنتهى اسمائهم بحرف (n) .

بعض **FUNCTION** الخاصة بالارقام و ايضا بالتواريخ

ROUND

تستخدم لتقريب الأرقام العشرية

EXAMPLE

ROUND (45.926, 2) 45.93

هنا تم التقريب إلي رقمين عشري كما حددنا لة فى المثال.

TRUNC

تقوم باختصار الرقم العشري بدون تقريب

EXAMPLE

TRUNC (45.926, 2) 45.92

يكتب الرقم المراد اختصاره ثم عدد الأرقام العشرية التي تريد ان تظهر بدون تقريب.
(ملحوظة لاحظالفرق بين **TRUNC** وبين **ROUND**)

MOD

تقوم بقسمة رقم علي رقم آخر و تنتج لنا فارق القسمة ان وجد

EXAMPLE

MOD (1600, 300) ⟶ 100

وهنا 1600 / 300 = 5 والباقي 100 والمطلوب هنا هو الباقي فقط 100.

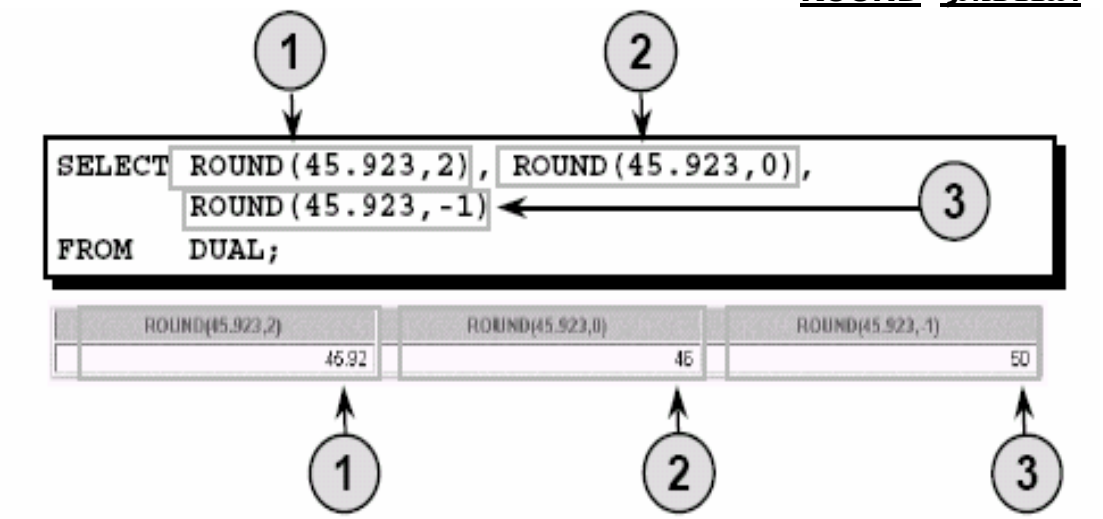
EXAMPLE 2

MOD (1500, 300) ⟶ 0

وهنا 1500 / 300 = 5 ولا يوجد باقي. وبالتالى الناتج يكون 0

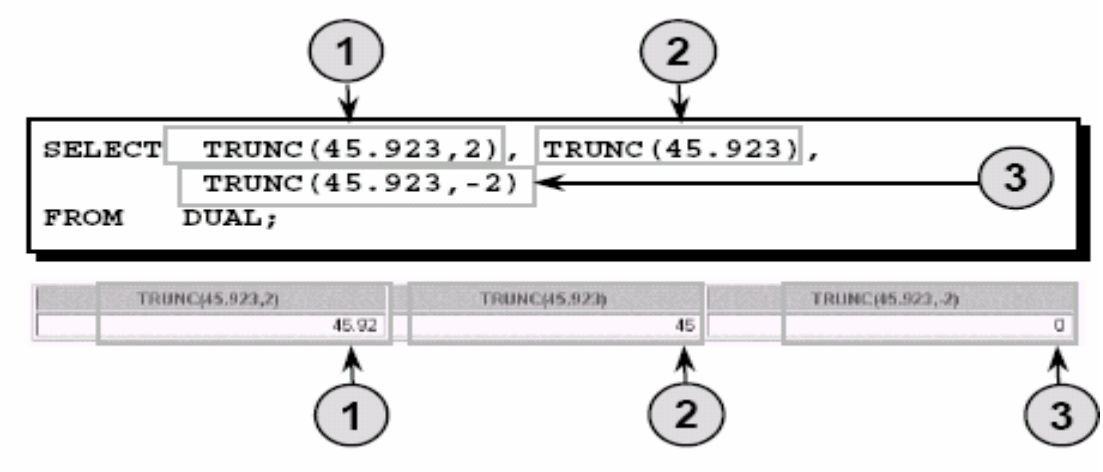
الدالة	الغرض
ROUND (column |expression , n)	تقرب الأرقام العشرية . ولو n حذفت تعنى ان الرقم يظهر من غير كسور. ولو n بالسالب: الرقم الذي علي يسار نقطة العلامة العشرية هو الذى يقرب.
TRUNC (*column* |*expression* ,*n*)	تقوم باختصار الرقم العشري لو n محذوفة يكون العرض ل n بصفر أي يكتب الرقم فقط من غير كسور.
MOD (*m* , *n*)	تحسب قيمة المتبقية من قسمة رقم علي آخر.

استخدام ROUND



لاحظ فى **المثال** السابق ROUND(45.923,-1) كان الناتج 50 حيث انة تم التقريب من بعد العلامة العشرية فاصبحت 5 ⟵ 4 و 0 ⟵ 5 وبالتالى اصبح الرقم 50 .

استخدام TRUNC



نلاحظ هنا فى **TRUNC** انها تقوم باختصار الرقم العشري بدون تقريب لة.

استخدام MOD

احسب الباقي من المرتب بعد قسمته علي 5000 لكل موظف وظيفتة مندوب مبيعات .

```
SELECT last_name, salary, MOD(salary, 5000)
FROM   employees
WHERE  job_id = 'SA_REP';
```

LAST_NAME	SALARY	MOD(SALARY,5000)
Abel	11000	1000
Taylor	8600	3600
Grant	7000	2000

دالة **MOD** تقوم بحساب باقي القسمة.
المثال السابق يعرض الباقي من قسمة المرتب علي 5000 لكل موظف والذي تكون وظيفته مندوب مبيعات .

أنت يمكن أن تغير عرض التاريخ بتنفيذ الأمر :
ALTER SESSION SET NLS_DATE_FORMAT = '*date format model*';

العمل بالتواريخ

- قاعدة بيانات ORACLE تخزن تواريخ في صيغة أعداد داخلية :
  قرن، سنة، شهر، يوم، ساعات، الدّقائق، ثواني .
- صيغة عرض التاريخ المختارة
  DD-MON-RR.

يسمح لك أن تخزن بيانات القرن 21 في القرن 20 بتحديد فقط الرقمان الأخيران من السنة .

```
SELECT last_name, hire_date
FROM   employees
WHERE  last_name like 'G%';
```

LAST_NAME	HIRE_DATE
Gietz	07-JUN-94
Grant	24-MAY-99

-

دالة القيمة المطلقة ABS Function

تستخدم هذه الدالة لإيجاد القيمة المطلقة لرقم معين وغالبا يتم استخدان هذه الدالة مع جمل أخرى

الشكل العام

ABS (COL|VALUE)

حيث أن

COL اسم الحقل أو العمود

VALUE القيمة البديلة للعمود (البيانات)

مثال

SELECT ename, job, and hiredate-sysdate FROM EMP ;

ENAME	JOB	HIREDATE-SYSDATE
SMITH	CLERK	-9234.4759
ALLEN	SALESMAN	-9169.4759
WARD	SALESMAN	-9167.4759
JONES	MANAGER	-9128.4759

SELECT ename, job, ABS (hiredate-sysdate) FROM EMP;

ENAME	JOB	ABS(HIREDATE-SYSDATE)
SMITH	CLERK	9234.48096
ALLEN	SALESMAN	9169.48096
WARD	SALESMAN	9167.48096

ـ الدالة الأسية POWER Function

تستخدم هذه الدالة لإيجاد قيمة رقم مرفوع لأس

الشكل العام

POWER (COL|VALUE,P)

حيث أن

COL اسم الحقل أو العمود

VALUE القيمة البديلة للعمود (البيانات)

P قيمة الأس

مثال

SELECT ENAME, SAL, POWER (SAL, 2) FROM EMP

ENAME	SAL	POWER(SAL,2)
SMITH	800	640000
ALLEN	1600	2560000
WARD	1250	1562500

دالة الجذر التربيعي SQRT Function

تستخدم هذه الدالة لإياد الجذر التربيعي لرقم معين

الشكل العام

SQRT (COL|VALUE)

حيث أن

COL اسم الحقل أو العمود

VALUE القيمة البديلة للعمود (البيانات)

مثال

SELECT ENAME, SAL, SQRT (SAL) FROM EMP

ENAME	SAL	SQRT(SAL)
SMITH	800	28.2842712
ALLEN	1600	40
WARD	1250	35.3553391

دالة الإشارة SIGN Function

تستخدم هذه الدااة بفحص إشارة الرقم فإذا كانت الإشاؤة موجبة تعود بالقيمة (1) أما اذا كانت الإشارة سالبة فتعود بالقيمة (-1)

الشكل العام انظر المثال التالي :

SELECT ENAME, SAL, SIGN (SAL) FROM EMP

ENAME	SAL	SIGN(SAL)
AVRIEM	-5000	-1
SMITH	800	1
ALLEN	1600	1
WARD	1250	1

العمل بالتواريخ

EXAMPLE

```
SELECT SYSDATE
FROM   DUAL;
```

SYSDATE
28-SEP-01

SYSDATE فى المثال السابق تعنى تاريخ اليوم وتأتى بة من على SERVER.

الحساب بالتواريخ

المعامل	نتيجةِ	الوصف
تاريخ + رقم	تاريخ	يضيف عدد من الأيامِ على تاريخ
تاريخ – رقم	تاريخ	يطرح عدد الأيامِ من تاريخ
تاريخ - تاريخ	عدد من الأيام	يطرح تاريخ واحد من الآخر
تاريخ + رقم/24	تاريخ	يضيف عدد من السّاعات على تاريخ

استخدام المعاملات الحسابية بالتواريخ

```
SELECT last_name, (SYSDATE-hire_date)/7 AS WEEKS
FROM   employees
WHERE  department_id = 90;
```

LAST_NAME	WEEKS
King	744.245395
Kochhar	626.102539
De Haan	468.245395

ملحوظة تاريخ التعين HIRE_DATE والتاريخ الحالي SYSDATE

المثال السابق يعرض الاسم وعددِ الأسابيع للموظفين عن طريق طرح تاريخ اليوم الحالى(SYSDATE) من تاريخ تعين الموظف (HIRE_DATE) ويقسم الناتج على 7 حتى يظهر الناتج بالاسابيع .

الملاحظة: SYSDATE تسترجع التاريخ الحالي .
بعض **FUNCTION** المستخدمة فى العمليات الحسابية على التاريخ

FUNCTION	الوصف
MONTHS_BETWEEN	لحساب فرق عدد الشهور بين تاريخيين
ADD_MONTHS	تستخدم لأضافه عدد محدد من الشهور إلي التاريخ المحدد
NEXT_DAY	لتحديد التاريخ التالي لليوم المحدد
LAST_DAY	لمعرفة اليوم الأخير في شهر معين

ROUND	لتقريب التاريخ أي عرض اقرب بداية شهر أو سنه لتاريخ معين تحدده
TRUNC	تقوم بإعادة التاريخ إلي البداية أي تعرض تاريخ أول يوم في شهر أو سنة لتاريخ معين تحدده

1-MONTHS_BETWEEN (*date1, date2*) :

1- يحدد عدد الشهور بين التاريخ الاول والتاريخ الثانى والنتيجة يمكن أن تكون موجبة أو سلبية.

2-ADD_MONTHS (*date, n*)

2- إضافة عدد n من الأشهر للتاريخ المحدد. قيمة n يجِب أن تكون عدد صحيح ويمكن أن يكون سالب .

3-NEXT_DAY (*date,* '*char*')

3- إيجاد التاريخ لليومِ المحدد فى الأسبوعِ التالى.
('*char*'): وتعنى اسم اليوم فى الاسبوع .

4-LAST_DAY (*date*)

4- إيجاد التاريخ لليوم الأخير في شهر معين .

5-ROUND (*date*[,'*fmt*'])

5- يسترجع التاريخ المقرب إلي الوحدةِ المحددة فى fmt اى (شهر او سنة).
إذا fmt حذفت، التاريخ سوف يقرب إلي اليومِ الأقربِ .

6-TRUNC (*date*[, '*fmt*'])

6- يسترجع تاريخ أول يوم في الشهر أو السنة لتاريخ معين .

EXAMPLES:

- `MONTHS_BETWEEN ('01-SEP-95','11-JAN-94')` → `19.6774194`
- `ADD_MONTHS ('11-JAN-94',6)` → `'11-JUL-94'`
- `NEXT_DAY ('01-SEP-95','FRIDAY')` → `'08-SEP-95'`
- `LAST_DAY('01-FEB-95')` → `'28-FEB-95'`

والمثال التالي يعرض رقم الموظف ، تاريخ التعين، عدد شهور التوظيف، ويتم اضافة ستة شهور لتاريخ التعيين وسمى REVIEW، ومعرفة الجمعة الأولي بعد تاريخ التعيين أي تحديد تاريخ أول يوم الجمعة يلي تاريخ التعيين والناتج يوضح لنا التاريخ الموافق لأول يوم الجمعة ويليه تاريخ التعيين لكل موظف ، واليوم الأخير من شهر التعيين (أي المطلوب معرفة اليوم الأخير في الشهر وهنا في المثال يحدد اليوم الأخير في شهر التعيين والناتج يوضح هذا حيث يعرض آخر يوم في الشهر) مع كل الموظفين.وذلك للموظفين الذين لايتعدوا 36 شهر.

```
SELECT employee_id, hire_date,
MONTHS_BETWEEN (SYSDATE, hire_date) TENURE,
```

```
ADD_MONTHS (hire_date, 6) REVIEW,
NEXT_DAY (hire_date, 'FRIDAY'), LAST_DAY(hire_date)
FROM employees
WHERE MONTHS_BETWEEN (SYSDATE, hire_date) < 36;
```

EMPLOYEE_ID	HIRE_DATE	TENURE	REVIEW	NEXT_DAY(	LAST_DAY(
107	07-FEB-99	31.6982407	07-AUG-99	12-FEB-99	28-FEB-99
124	16-NOV-99	22.4079182	16-MAY-00	19-NOV-99	30-NOV-99
149	29-JAN-00	19.9885633	29-JUL-00	04-FEB-00	31-JAN-00
178	24-MAY-99	28.1498536	24-NOV-99	28-MAY-99	31-MAY-99

المثال السابق يقوم بعرض رقم الموظف وتاريخ تعينة والمدة الذى عمل بها فى الشركة عن طريق معرفة الفرق بين اليوم الحالى SYSDATE و تاريخ تعينة HIRE_DATE ذلك باستخدام MONTHS_BETWEEN وايضا يريد معرفة التاريخ بعد 6 شهور من تاريخ تعين الموظفين و يريد معرفة تاريخ يوم الجمعة القادم فى الاسبوع القادم و تاريخ اخر يوم فى سنة تعيين كل موظف.وذلك للموظفين الذين عملوا بالشركة اقل من 36 شهر .

نفترض أن التاريخ (الحالي) هو : '25-JUL-95'

```
ROUND(SYSDATE,'MONTH') ⟶ 01-AUG-95

ROUND(SYSDATE ,'YEAR') ⟶ 01-JAN-96

TRUNC(SYSDATE ,'MONTH') ⟶ 01-JUL-95

TRUNC(SYSDATE ,'YEAR') ⟶ 01-JAN-95
```

EXAMPLE

```
SELECT employee_id, hire_date,
       ROUND(hire_date, 'MONTH'), TRUNC(hire_date, 'MONTH')
FROM   employees
WHERE  hire_date LIKE '%97';
```

EMPLOYEE_ID	HIRE_DATE	ROUND(HIR	TRUNC(HIR
142	29-JAN-97	01-FEB-97	01-JAN-97
202	17-AUG-97	01-SEP-97	01-AUG-97

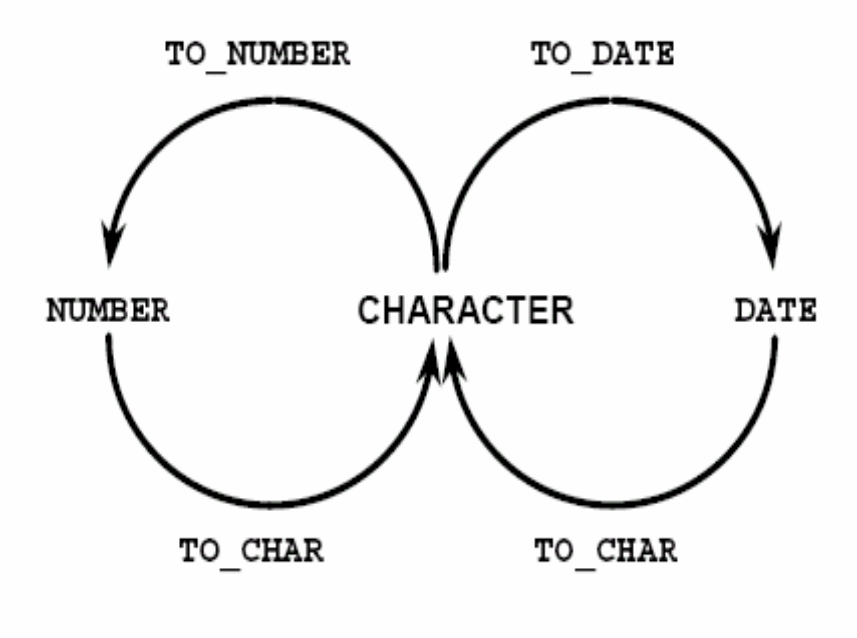
المثال السابق يعرض بيانات الموظفين ويقوم بتقريب تاريخ التعين لمدة شهر ويقوم بقطع تاريخ التعين لمدة شهر بمعنى اظهار اول الشهر بدون تقريب.

الشكل الذي أمامنا يوضح التغيرات التي يقوم بها أوراكل آليا عند الحاجة إليها والمتبع التالي:

From	To
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE
NUMBER	VARCHAR2
DATE	VARCHAR2

التحويلات الخارجية

وهي التحويلات التي يقوم بها المستخدم



SQLيزود بثلاث وظائف أن تحول قيمة من البياناتِ واحدة إلي الأخرى :

الدوال	الغرض
TO_CHAR(*number* |*date*,[*fmt*], *[nlsparams*])	تحويل قيم التاريخ اوالرقم إلي مجموعة من الحروف **VARCHAR2** بالنموذج الشكل **fmt**. تحويل الرقم : إلي نص
TO_CHAR(*number* |*date*,[*fmt*], [*nlsparams*])	تحول التاريخ إلي نص
TO_NUMBER(*char,[fmt], [nlsparams]*)	تحويل الحروف تقوم بتحويل الحروف إلي أرقام
TO_DATE(c*har*,[*fmt*] ,[*nlsparams*])	تحويل الحروف تقوم بتحويل الحروف إلي تاريخ

استخدام **TO_CHAR** مع التواريخ.

TO_CHAR (date, 'format_model')

```
SELECT employee_id, TO_CHAR(hire_date, 'MM/YY') Month_Hired
FROM   employees
WHERE  last_name = 'Higgins';
```

EMPLOYEE_ID	MONTH
205	06/94

ففى **المثال** السابق يقوم بعرض رقم الموظف ويقوم بتحويل شكل تاريخ التعين من (DD-MON-RR) الى (MM/YY). وفيما يلى بعض الاشكال للتاريخ:

YYYY	سنة كاملة من الأرقام مثلا 1999
YEAR	السنة مكتوبة حرفيا
MM	تمثل رقمين للشهر مثل(يناير= 1)
MONTH	أسم الشهر بالكامل
MON	اول ثلاثة حروف من الشهر
DY	ثلاثة حروف مختصرة لليوم والأسبوع
DAY	أسم اليوم بالكامل
DD	(أيام الشهر) 1-31

نموذج صيغةِ التاريخ

عناصر صيغ التاريخ الوقت جزء يقسم من التاريخ .

HH24:MI:SS AM	15:45:32 PM

HH24 : الساعة في اليوم (0-23)
MI : الدقائق (0-59)
SS : الثواني (0-59)
AM : قبل الظهر PM : بعد الظهر

* أضف مجموعة الحروف مشتملة علي علامة " " فمثلا :

DD "of" MONTH	12 of OCTOBER

DD : أيام الشهر (من 1- 31)

العناصر	وصف
AM OR PM	مؤشر قبل الظهر أو بعد الظهر (من غير مدة)
A.M. OR P.M.	مؤشر قبل الظهر أو بعد الظهر(بمدة)
HH OR HH12 OR HH24	الساعات اليومية (1-12) و(1-23)
MI	الدقائق (0-59)
SS	الثواني (0-59)

استخدام دالة **TO_CHAR** مع التواريخ

```
SELECT last_name,
       TO_CHAR(hire_date, 'fmDD Month YYYY')
       AS HIREDATE
FROM   employees;
```

LAST_NAME	HIREDATE
King	17 June 1987
Kochhar	21 September 1989
De Haan	13 January 1993
Hunold	3 January 1990
Ernst	21 May 1991
Lorentz	7 February 1999
Mourgos	16 November 1999

...

20 rows selected.

المثال هنا يعرض الأسماء الأخيرةِ وتاريخ التعيين لكل الموظفين. ولكن سيتم تغيير أسلوب عرض التاريخ حيث يعرض هكذا 17 June 1987.

```
SELECT last_name,
TO_CHAR (hire_date,'fmDdspth "of" Month YYYY fmHH:MI:SS AM')
HIREDATE
FROM employees;
```

LAST_NAME	HIREDATE
King	Seventeenth of June 1987 12:00:00 AM
Kochhar	Twenty-First of September 1989 12:00:00 AM
...	
Higgins	Seventh of June 1994 12:00:00 AM
Gietz	Seventh of June 1994 12:00:00 AM

20 rows selected.

فى **المثال** السابق يريد عرض الاسم وتاريخ التعين ولكن بالشكل المعروض فى المثال. ولمعرفة الفرق بين شكلين عرض التاريخ قم بالاتى بدون تعديل شكل عرض التاريخ ولاحظ الفرق.

```
Select last_name,hire_date from employees;
```

LAST_NAME	HIRE_DATE
SMITH	17-DEC-80
ALLEN	20-FEB-81
WARD	22-FEB-81
JONES	02-APR-81
MARTIN	28-SEP-81
BLAKE	01-MAY-81
CLARK	09-JUN-81
SCOTT	19-APR-87
KING	17-NOV-81

إستخدام دالة **TO_CHAR** مع الأرقام

```
SELECT TO_CHAR(salary, '$99,999.00') SALARY
FROM   employees
WHERE  last_name = 'Ernst';
```

SALARY
$6,000.00

مثال تستخدم TO_CHAR هنا مع الأرقام
الصورة المراد اظهار الرقم بها هى ' $99,999.00'
والناتج يظهر لنا بالشكل الجديد للتنسيق $6,000.00 .
* يعْرضُ ORACLE مجموعة العلامات (#) بدلا من العدد الصحيح عند تجاوزْ عددَ الأرقام المزودة في نموذج الصيغة من (0إلي 9).

EXAMPLE

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-YYYY')
FROM  employees
WHERE hire_date < TO_DATE('01-Jan-90', 'DD-Mon-RR');
```

LAST_NAME	TO_CHAR(HIR
King	17-Jun-1987
Kochhar	21-Sep-1989
Whalen	17-Sep-1987

مثال آخر:-

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-yyyy')
FROM   employees
WHERE  TO_DATE(hire_date, 'DD-Mon-yy') < '01-Jan-1990';
```

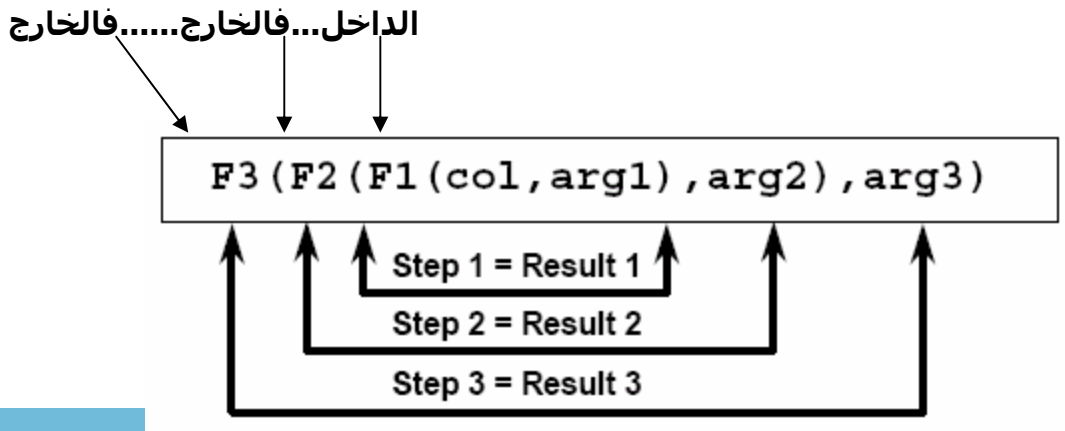
No rows selected

مجموعة من FUNCTION

الدوال	الوصف
NVL (expr1, expr2)	يحول قيمة Null إلي قيمة حقيقية اذا كان exp1 يساوى null يتم تبديلة ب exp2
NVL2 (expr1, expr2, expr3)	لو التعبير الأول expr1 بقيمة ,يتم استرجاع expr2 .ولو التعبير الأول null , يتم استرجاع expr3 . التعبير expr1 يمكن أن يأخذ أي نوع بيانات.
NULLIF (expr1, expr2)	يقارن تعبيرين ويسترجع Null إذا التعبيرين متساويين، و يسترجع التعبير الأول إذا التعبيرين غير متساويين.
COALESCE	يسترجع أول تعبير غير Null في قائمة التعبير.

أولويات تنفيذ **FUNCTION**

يمكن ان تستخدم اكثر من **FUNCTION** فى الكود وتكون اسبقية التنفيذ من الداخل الى الخارج .



مثال على أولويات تنفيذ FUNCTION

```
SELECT last name,
       NVL(TO_CHAR(manager_id), 'No Manager')
FROM   employees
WHERE  manager_id IS NULL;
```

LAST_NAME	NVL(TO_CHAR(MANAGER_ID),'NOMANAGER')
King	No Manager

ففي **المثال** نجد أن أولويات التنفيذ تكون بالترتيب :-

1- TO_CHAR	2- NVL

يهدف المثال السابق الى:

عرض عمود Manager_ID بحيث لو Manager_ID ب NULL يتم ادراج كلمة No Managerبدلا من NULL .وذلك باستخدام NVL .

وحيث ان نوع البيانات للعمود Manager_ID ارقام ونحن نريد كتابة حروف فى هذا العمود فلابد من تحويل الارقام الى حروف وذلك باستخدام TO_CHAR

مثال:

يريد هذا المثال عرض تاريخ يوم الجمعة القادمة بعد ستة شهور من تاريخ التعيين.
و يجِب أن يكون التاريخ المعروض بالشكل التالى
(يوم الجمعة، 13أغسطس 1999) .

```
SELECT TO_CHAR(NEXT_DAY(ADD_MONTHS
(hire_date, 6), 'FRIDAY'),
'fmDay, Month DDth, YYYY')
"Next 6 Month Review"
FROM employees
ORDER BY hire_date;
```

لاحظ ترتيب **FUNCTIONS :**

- يتم اضافة ستة شهور الى تاريخ التعين اولا.
- يتم معرفة يوم الجمعة القادم من التاريخ الذى تم التوصل الية.
- يتم عرض التاريخ بالشكل المطلوب.

شرح لمجموعة FUNCTIONS السابقة :

NVL (Exp1, Exp2)

تقوم NVL بفحص القيمة الاولى واذا وجدت ب NULL يتم تحويلها الى القيمة الثانية واذا وجدت بقيمة تظل تلك القيمة كما هى بدون تغير .

امثلة:

- NVL(commission_pct,0)
- NVL(hire_date,'01-JAN-97')
- NVL(job_id,'No Job Yet')

NVL2 (Exp1, Exp2, Exp3)

تقوم NVL2 بفحص اول قيمة واذا وجدت ب NULL يتم تحويلها الى القيمة الثالثة واذا وجدت بقيمة يتم تحويلها الى القيمة الثانية .

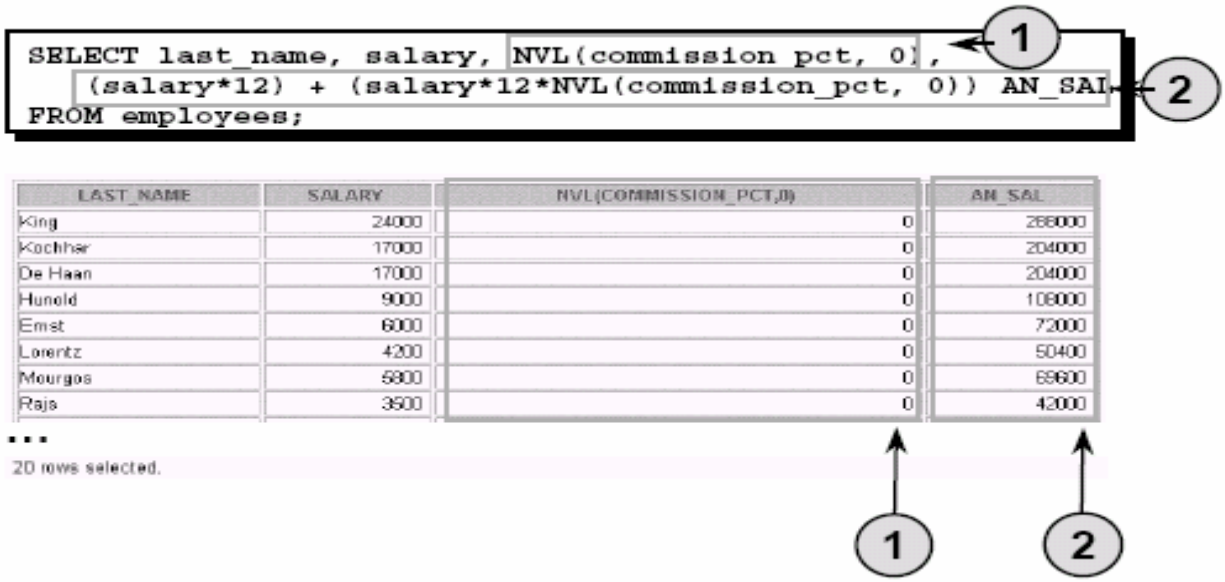
NULLIF (Exp1, Exp2)

تقوم NULLIF بمقارنة القيمة الاولى بالقيمة الثانية واذا تساوى القيمتين يكون الناتج ب NULL واذا لم تتساوى القيمتين يظهر الناتج بالقيمة الاولى.

COALESCE (Exp1, Exp2, Exp3,Exp4,.....)

تقوم COALESCE باسترجاع اول قيمة لاتساوى NULL .

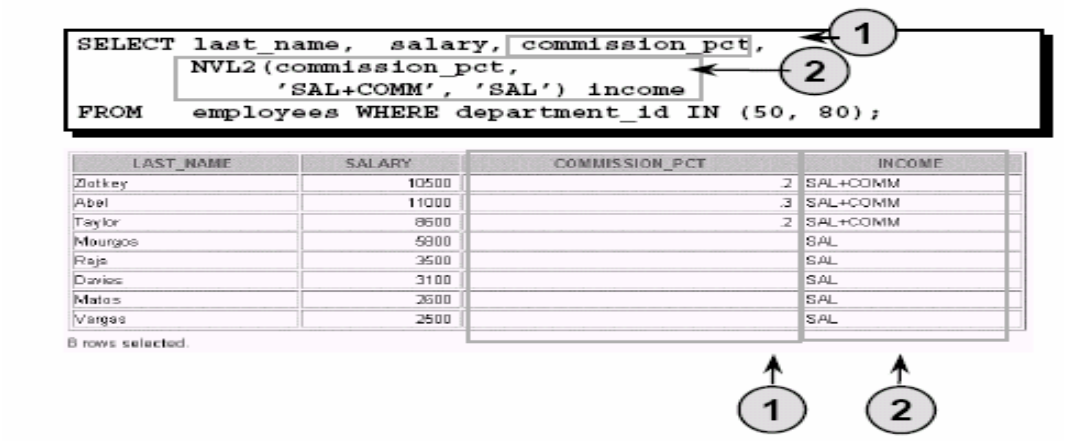
إستخدام NVL



وفى هذا المثال يريد تحويل قيمة COMM الى Zero اذا وجدت ب NULL وذلك لكى يتمكن من جمع المرتب على COMM لكى يعرف مجمل الدخل للموظفين.

و تستخدم NVL لاستبدال القيمة NULL بقيمة أخري للسماح بالقيام بالعمليات الحسابية.وفي المثال الرقم صفر هو الذى سوف يحل محل NULL حيث أن COMM يحتوي علي قيمة NULL وكما نري تم اعتبار قيمة commission العمولة التي تمثل NULL بصفر لذا فقد أمكن إتمام عملية الجمع.

استخدام NVL2



```
SELECT last_name,  salary, commission_pct,
       NVL2(commission_pct,
            'SAL+COMM', 'SAL') income
FROM   employees WHERE department_id IN (50, 80);
```

LAST_NAME	SALARY	COMMISSION_PCT	INCOME
Zlotkey	10500	.2	SAL+COMM
Abel	11000	.3	SAL+COMM
Taylor	8600	.2	SAL+COMM
Mourgos	5800		SAL
Raja	3500		SAL
Davies	3100		SAL
Matos	2600		SAL
Vargas	2500		SAL

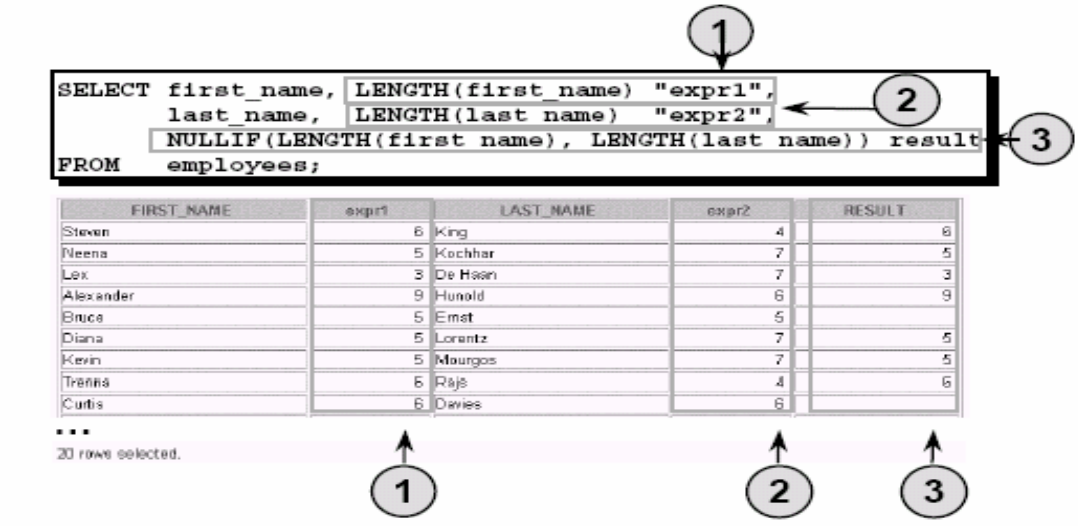
8 rows selected.

يهدف المثال السابق الى معرفة الدخل الشهرى لكل موظف

NVL(*expr1*, *expr2, expr3*)

في هذا **المثال** عندما يجد COMM ب NULL يقوم بتحويلها الى القيمة الثالثة وهى المرتب اما اذا كان الموظف ياخذ COMM فتكون النتيجة جمع العمولة و المرتب لمعرفة الدخل الشهرى .

استخدام NULLIF



```
SELECT first_name, LENGTH(first_name) "expr1",
       last_name,  LENGTH(last name)  "expr2",
       NULLIF(LENGTH(first name), LENGTH(last name)) result
FROM   employees;
```

FIRST_NAME	expr1	LAST_NAME	expr2	RESULT
Steven	6	King	4	6
Neena	5	Kochhar	7	5
Lex	3	De Haan	7	3
Alexander	9	Hunold	6	9
Bruce	5	Ernst	5	
Diana	5	Lorentz	7	5
Kevin	5	Mourgos	7	5
Trenna	6	Rajs	4	6
Curtis	6	Davies	6	

...

20 rows selected.

وهنا يتم مقارنة طول الاسم الاول بطول الاسم الاخير وتكون النتيجة ب NULL اذا كان الاسمين متساويان فى الطول. واذا لم يتساوى يتم عرض طول الاسم الاول.

استخدام COALESCE

```
SELECT   last_name,
         COALESCE(commission_pct, salary, 10) comm
FROM     employees
ORDER BY commission_pct;
```

LAST_NAME	COMM
Grant	.15
Zlotkey	.2
Taylor	.2
Abel	.3
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000
...	

20 rows selected.

هذا **المثال** يوضح ان إذا كانت قيمة العمولة ليست NULL فســـوف تعــرض. COALESCE تبحث عن اول قيمة لاتساوى NULL.

التعبيرات الشرطية IF-THEN-ELSE

1 CASE
2 DECODE

CASE

الشكل العام لل CASE

```
CASE expr WHEN comparison_expr1 THEN return_expr1
          [WHEN comparison_expr2 THEN return_expr2
           WHEN comparison_exprn THEN return_exprn
           ELSE else_expr]
END
```

تستخدم CASE لامكانية استخدام (لو) اى انها تعتبر اداة شرط .

```
SELECT last_name, job_id, salary,
       CASE job_id WHEN 'IT_PROG'  THEN  1.10*salary
                   WHEN 'ST_CLERK' THEN  1.15*salary
                   WHEN 'SA_REP'   THEN  1.20*salary
       ELSE       salary END     "REVISED_SALARY"
FROM   employees;
```

LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
...			
Lorentz	IT_PROG	4200	4620
Mourgos	ST_MAN	5800	5800
Rajs	ST_CLERK	3500	4025
...			
Gietz	AC_ACCOUNT	8300	8300

20 rows selected.

ففى **المثال** السابق يستخدم CASE كأداة شرط على الوظيفة فعندما تكون الوظيفة IT_PROG مثلا يقوم بضرب المرتب فى نسبة معينة وكلما اختلفت الوظيفة اختلفت تلك النسبة المضروبة فى المرتب .

بمعني :-

إذا كانت الوظيفة IT_PROG المرتب يزيد بنسبة 10% ,
إذا كانت الوظيفة ST_CLERK المرتب يزيد بنسبة 15% ,
إذا كانت الوظيفة SA_REP المرتب يزيد بنسبة 20% .ولباقى الوظائف الأخرى لا يوجد زيادة في المرتب . نفس المثال السابق يمكن تنفيذة بال DECODE.

والشكل المستخدم لل DECODE

```
DECODE(col|expression, search1, result1
                      [, search2, result2,...,]
                      [, default])
```

استخدام DECODE

```
SELECT last name, job id, salary,
       DECODE(job_id, 'IT_PROG',  1.10*salary,
                      'ST_CLERK', 1.15*salary,
                      'SA_REP',   1.20*salary,
              salary)
       REVISED_SALARY
FROM   employees;
```

LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
...			
Lorentz	IT_PROG	4200	4620
Mourgos	ST_MAN	5800	5800
Rajs	ST_CLERK	3500	4025
...			
Gietz	AC_ACCOUNT	8300	8300

20 rows selected.

إذا كانت الوظيفة IT_PROG المرتب يزيد بنسبة 10% ,
إذا كانت الوظيفة ST_CLERK المرتب يزيد بنسبة 15% ,
إذا كانت الوظيفة SA_REP المرتب يزيد بنسبة 20% . ولباقى الوظائف الأخرى لا يوجد زيادة لهم في المرتب فيبقى المرتب كما هو. DECODE تستخدم بدلا من IF-THEN-ELSE

IF job_id = 'IT_PROG' THEN salary = salary*1.10
IF job_id = 'ST_CLERK' THEN salary = salary*1.15
IF job_id = 'SA_REP' THEN salary = salary*1.20
ELSE salary = salary

مثال :

أعرض نسبةَ الضّريبةَ القابلة للتطبيقَ لكل موظف في قسم 80 .

```
SELECT last name, salary,
       DECODE (TRUNC(salary/2000, 0),
                         0, 0.00,
                         1, 0.09,
                         2, 0.20,
                         3, 0.30,
                         4, 0.40,
                         5, 0.42,
                         6, 0.44,
                            0.45) TAX_RATE
FROM   employees
WHERE  department_id = 80;
```

سوف نحدد نسبة الضريبة لكل موظف في القسم 80 حسب قيمة الراتب الشهري لكل موظف.

مدي الراتب الشهري	النسبة
$0.00 - 1999.99	00%
$2,000.00 - 3,999.99	09%
$4,000.00 - 5,999.99	20%
$6,000.00 - 7,999.99	30%
$8,000.00 - 9,999.99	40%
$10,000.00 - 11,999.99	42%
$12,200.00 - 13,999.99	44%

$14,000.00 or greater 45%

LAST_NAME	SALARY	TAX_RATE
Zlotkey	10500	.42
Abel	11000	.42
Taylor	8600	.4

ملخص الفصل

تناولنا في هذا الفصل الدوال الرقمية في لغة الاستعلام والاستفهام في نظام قواعد البيانات اوراكل وقد قسمنا هذه الدوال حسب وظائفها فمنها مايتعامل مع سجل أو صف من البيانات ومنها مايتعامل مع مجموعة من السجلات RECORDS او الأعمدة COLUMNS(الحقول)

حيث تطرقنا الى دالة القيمة المطلقة والدالة الاسية والجذر التربيعي وكذلك دوال التقريب ودالة باقس القسمة والاشارة

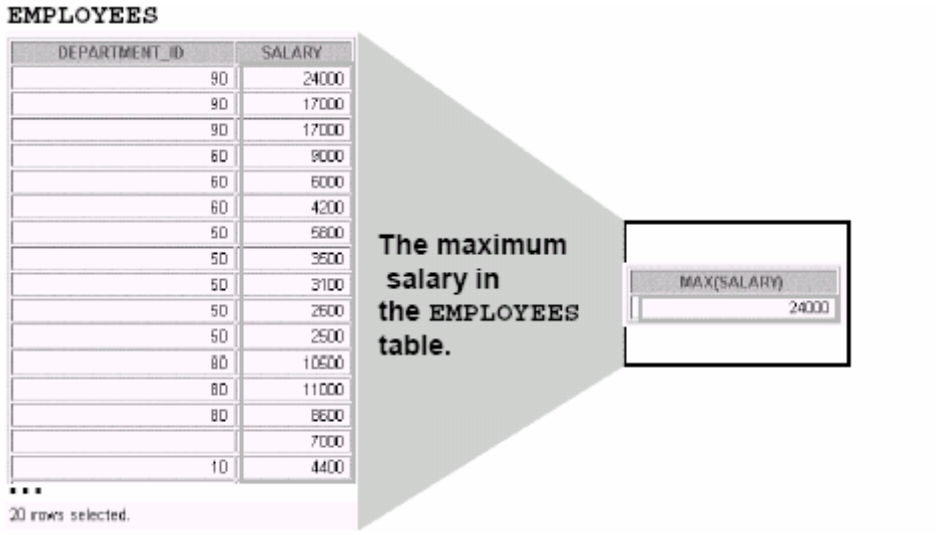
الفصل الرابع

Multiable Row Function

ماذا يعني ب " Grouping Function"؟

هي عبارة عن مجموعة من Functions تعمل علي مجموعة من الصفوف لتعرض نتيجة واحدة.

ويمكن أن تعمل علي كل بيانات الجدول أو علي جزء فقط من بيانات الجدول.



أنواع Group Functions

الدوال	الوصف
AVG([DISTINCT|ALL]*n*)	تقوم بحساب المتوسط الحسابى لمجموعة من الأرقام و تقوم بتجاهل القيمة الفارغة NULL .
COUNT({*|[DISTINCT|ALL]*expr*})	تقوم بحساب عدد الحقول في عمود معين التي لا تحمل قيمة NULL.
MAX([DISTINCT|ALL]*expr*)	وهي اختصار كلمة MAXIMUM وهي تقوم بالبحث عن أكبر قيمة لمجموعة من القيم في عمود معين .
MIN([DISTINCT|ALL]*expr*)	وهي اختصار كلمة MINIMUM وهي تقوم بالبحث عن أقل قيمة موجودة في عمود معين.
STDDEV([DISTINCT|ALL]*x*)	دالة الانحراف المعياري وهي اختصار لكلمة Stander deviation .
SUM([DISTINCT|ALL]*n*)	تقوم بحساب مجموع كل القيم الموجودة في عمود معين او فى مجموعة صفوف فقط.
VARIANCE([DISTINCT|ALL]*x*)	تستخدم لحساب معدل الاختلاف او المعيار لمجموعة مشاهدات أو قراءات .

ملحوظة هامة جدا

كل Group Function تتجاهل NULL فيما عدا دالة COUNT اذا استخدمت مع *.

حيث عند استخدامها مع النجمة * اى (*) COUNT فهنا لا تتجاهل قيمة NULL.

الصيغة العامة للـ Group Function

```
SELECT      [column,]  group function (column), ...
FROM        table
[WHERE      condition]
[GROUP BY   column]
[ORDER BY   column];
```

ملاحظة :
*ت رتب ُ النّتيجةَ بترتيب تصاعديِ عندما تستخدم فقرة GROUP BY. وللترتيب التنازلى نستخدم DESC فى فقرة ORDER BY.

استخدام AVG و SUM

يمكنك استخدام دوال AVG و SUM للبيانات الرقمية.

```
SELECT AVG(salary), MAX(salary),
       MIN(salary), SUM(salary)
FROM   employees
WHERE  job_id LIKE '%REP%';
```

AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
8150	11000	6000	32600

المثال السابق يريد معرفة المتوسط الحسابى لمرتبات الموظفين و اعلى مرتب و اقل مرتبٍ و مجموع المرتبات للموظفين .
* يُمكنُ أتستخدام AVG و SUM و MIN و MAX مع الأعمدةِ التي يمكن أن تخزن بيانات رقمية.

مثال اخر:

استخدام MIN و MAX

يمكنك استخدام MIN و MAX لأي نوع من البيانات.

```
SELECT MIN(hire_date), MAX(hire_date)
FROM   employees;
```

MIN(HIRE_	MAX(HIRE_
17-JUN-87	29-JAN-00

يعرض **المثال** السابق اقدم موظف فى الشركة ذلك عن طريق استخدام MIN مع تاريخ التعين و يعرض ايضا احدث موظف فى الشركة عن طريق استخدام MAX .

```
SELECT MIN(last_name), MAX(last_name)
FROM   employees;
```

MIN(LAST_NAME)	MAX(LAST_NAME)
Abel	Zlotkey

ويعرض المثال السابق اسماء الموظفين وذلك عن طريق استخدام
MAXو MIN فتقوم الـ MAX بأظهار الاسم الذى يبدأ بأخر حرف هجائيا (Z).
و تقوم الـ MIN بأظهار الاسم الذى يبدأ بأول حرف هجائي (A).

لاحظ : كل FUNCTIONS تعمل علي القيم الرقمية فقط ماعدا MAX و MIN فمن الممكن أن يعملان مع التواريخ .

استخدام دالة COUNT

COUNT (*) تقوم بحساب عدد الصفوف في الجدول

```
SELECT COUNT(*)
FROM   employees
WHERE  department_id = 50;
```

COUNT(*)
5

عند استخدام النجمة (*) مع COUNT بدلا من اسم العمود فإنها تقوم بحساب عدد الصفوف الموجودة فى الجدول .
ومن هنا نجد أن **COUNT** لها ثلاث أشكال **:**

1- COUNT(*)	يقوم بحساب عدد الصفوف في الجدول ويتضمن الصفوف التي تحتوي قيمة فارغة NULL.
2- COUNT(expr)	فسوف تقوم بحساب عدد القيم التي لا تحتوي علي قيم فارغة NULL في الأعمدة التي حددت بواسطة expr.
3- COUNT (DISTINCT expr)	يقوم بحساب عدد القيم الفريدة اى الغير مكررة في الأعمدة المحددة.

```
SELECT COUNT(commission_pct)
FROM   employees
WHERE  department_id = 80;
```

COUNT(COMMISSION_PCT)
3

يعرض **المثال** عددِ الموظفين في القسمِ 80 الذين يأخذون عمولة .
عند تحديد اسم عمود COUNT(commission_pct) مع COUNT فإنها تقوم بحساب عدد الصفوف التي لا تحتوي علي قيم فارغة NULL لهذا العمود وعدد الصفوف الموجودة والتي لا تحتوي علي NULL هم ثلاثة .

أعرض عدد الأقسمِ الموجودة فى جدول الموظفين .

```
SELECT COUNT(department_id)
FROM   employees;
```

COUNT(DEPARTMENT_ID)
19

وتعتبر تلك النتيجة السابقة خاطئة لأنة قام بحساب التكرارت فى عمود DEPARTMENT_ID . ولمعالجة ذلك نقوم باستخدام DISTINCT
كما فى **المثال التالى .**

استخدام DISTINCT

COUNT (DISTINCT expr) تستخدم لمنع احتساب الحقول التي تتكرر .اى تقوم بمنع Duplicate .

```
SELECT COUNT(DISTINCT department_id)
FROM   employees;
```

COUNT(DISTINCTDEPARTMENT_ID)
7

ملاحظة:استخدام DISTINCT لكى تمنع حِساب أي قِيَمِ مكررة فى العمودِ .

```
SELECT AVG(commission_pct)
FROM   employees;
```

AVG(COMMISSION_PCT)
.2125

كما ذكرنا ان جميع GROUP FUNTIONS تتجاهل قيم NULL فيما عدا COUNT(*) وعند حساب المتوسط لقيم العمود AVG(commission_pct) يتم تجاهل قيم NULL لبقية القيم و يتم حساب المتوسط عن طريق جمع قيم العمود و قسمته علي عدد الحقول التى بها قيمة فقط و بذلك يتم اعطاء نتيجة غير صحيحة للمتوسط الحسابى للعمولات commission وللتغلب علي هذه المشكلة نستخدم NVL.

استخدام NVL مع GROUP FUNCTIONS

```
SELECT AVG(NVL(commission_pct, 0))
FROM   employees;
```

AVG(NVL(COMMISSION_PCT,0))
.0425

وبذلك يتم القسمة على عدد الحقول كلها وليس الحقول التى بها قيمة فقط.

دالة الانراف المعياري STDDEV

تقوم هذه الدالة بإيجاد الانحراف المعياري لمجموعة مشاهدات أو قراءات

الشكل العام

STDDEV (DISTINCT|ALL)

مثال

لإيجاد الانحراف المعياري للأرقام الواردة في حقل الرواتب في جدول الموظفين EMP

SELECT STDDEV (SAL) FROM EMP

STDDEV(SAL)
2321.21995

دالة المعيار VARIANCE

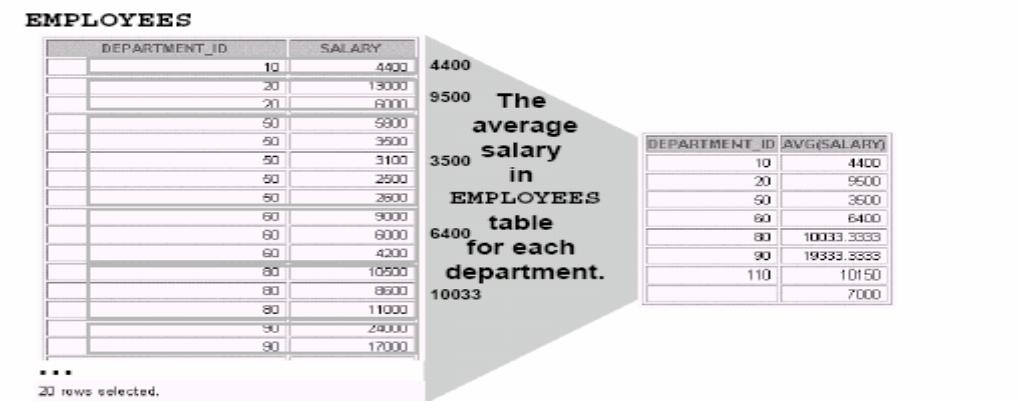
تقوم هذه الدالة بإيجاد المعيار لمجموعة مشاهدات أو قراءات

مثال

SELECT VARIANCE (SAL) FROM EMP;

VARIANCE(SAL)
5388062.07

استخدام GROUP BY



حتى الآن كل GROUP FUNCTIONS تتعامل مع الجدول كمجموعة واحدة.
ولكن باستخدام GROUP BY يمكنك من تقسيم البيانات.فيمكن حساب المتوسط للمرتبات بجدول الموظفين لكل قسم من الاقسام على حدة .

استخدام GROUP BY

```
SELECT        column, group_function(column)
FROM          table
[WHERE        condition]
[GROUP BY     group_by_expression]
[ORDER BY     column];
```

تقسم صفوف الجدول إلي مجموعات صغيرة بواسطة GROUP BY .
يمكنك استخدام **GROUP BY** كي تقسم صفوف الجداول إلي مجموعات.

بعض الشروط لاستخدام **GROUP BY :**

1. اذا تم ادراج عمود فى جملة SELECT و أردت استخدام GROUP BY فلابد من ادراج تلك العمود المدرج فى جملة SELECT فى فقرة GROUP BY وبغير ذلك يحدث Error .
2. يمكن ان تستخدم فقرة WHERE لتحديد الصفوف المراد عرضها ذلك قبل استخدام فقرة GROUP BY .
3. لا يمكن استخدام الاسم المستعار (Alias) فى فقرة Group By .
4. يتم ترتيب الناتج تصاعديا .ويمكن أن تتجاوز هذا باستخدام الفقرة ORDER BY.

استخدام GROUP BY

```
SELECT    department_id, AVG(salary)
FROM      employees
GROUP BY department_id ;
```

DEPARTMENT_ID	AVG(SALARY)
10	4400
20	9500
50	3500
60	6400
80	10033.3333
90	19333.3333
110	10150
	7000

8 rows selected

ففى **المثال** السابق يقوم بحساب المتوسط الحسابى للمرتبات لكل قسم على حدة. وذلك باستخدام Group by .
لاحظ استخدام نفس العمود بجملة SELECT فى جملة GROUP BY .
و ليس من الضروري تواجد اسم العمود المستخدم بجملة GROUP BY بجملة SELECT كما فى **المثال** التالى **:**

```
SELECT    AVG(salary)
FROM      employees
GROUP BY department_id ;
```

AVG(SALARY)
4400
9500
3500
6400
10033.3333
19333.3333
10150
7000

يمكن استخدام **GROUP BY** مع أكثر من عمود.
يجب ذكر جميع أسماء الأعمدة المدرجة في جملة SELECT بجملة GROUP By.
عند إغفال ذكر أحد الأعمدة في عبارة GROUP BY فتظهر رسالة خطأ ولا يتم تنفيذ الآمر.

مثال على ذلك:

Enter statements:

```
SELECT department_id dept_id,job_id,sum(salary) from employees
group by department_id
```

Execute | Save Script | Clear Screen | Cancel

```
SELECT department_id dept_id,job_id,sum(salary) from employees
                             *

ERROR at line 1:
ORA-00979: not a GROUP BY expression
```

نلاحظ ان هناك خطا عند استخدام جملة GROUP BY وذلك لعدم ادراج JOB_ID داخل فقرة GROUP BY .

استخدام GROUP BY مع مجموعة من الأعمدة

```
SELECT    department_id dept_id, job_id, SUM(salary)
FROM      employees
GROUP BY department_id, job_id ;
```

DEPT_ID	JOB_ID	SUM(SALARY)
10	AD_ASST	4400
20	MK_MAN	13000
20	MK_REP	6000
50	ST_CLERK	11700
50	ST_MAN	5800
60	IT_PROG	19200
80	SA_MAN	10500
80	SA_REP	19600
90	AD_PRES	24000
90	AD_VP	34000
110	AC_ACCOUNT	8300
110	AC_MGR	12000
	SA_REP	7000

13 rows selected.

استعلامات غير صحيحة تستخدم Group Function أي عمود أو تعبير في قائمة SELECT يجب أن يكون بفقرة GROUP BY.

```
SELECT department_id, COUNT (last_name)
FROM employees;

SELECT department_id, COUNT (last_name)
*
ERROR at line 1:
ORA-00937: not a single-group group function
```

ولمعالجة الخطأ السابق يجب ادراج فقرة GROUP BY متضمنة العمود الذى فى جملة SELECT كما فى **المثال التالى** .

```
Enter statements:
SELECT department_id, COUNT(last_name)
FROM employees
GROUP BY department_id
```

DEPARTMENT_ID	COUNT(LAST_NAME)
10	1
20	2
30	6
40	1
50	45
60	5
70	1
80	34
90	3
100	6
110	2
	1

استخدام Having لتحديد (شرط) على المجموعات

تستخدم HAVING لامكانية استخدام شرط مع Group Functions.
حيث لايمكن استخدام WHERE كشرط مع Group Functions .
كما سوف نرى فى **المثال التالى:**

```
SELECT    department_id, AVG(salary)
FROM      employees
WHERE     AVG(salary) > 8000
GROUP BY department_id;
WHERE  AVG(salary) > 8000
       *
ERROR at line 3:
ORA-00934: group function is not allowed here
```

لا يمكن استخدام **WHERE** مع **Group Functions** . حيث ان where تعمل على تحديد الصفوف قبل تجميعها فى شكل مجموعات . لذلك نستخدم Having وهى

تعمل داخل صفوف لكل مجموعة على حدى ويمكن استخدام Group Functions داخلها. **كما يلى:**

```
SELECT   department_id, AVG(salary)
FROM     employees
HAVING   AVG(salary) > 8000
GROUP BY department_id;
```

DEPARTMENT_ID	AVG(SALARY)
20	9500
80	10033.3333
90	19333.3333
110	10150

```
SELECT department_id, COUNT(last_name)
FROM employees
where
department_id=10
GROUP BY department_id
having avg(salary)>10
order by department_id
```

DEPARTMENT_ID	COUNT(LAST_NAME)
10	1

مثال أخر:

```
SELECT   department_id, MAX(salary)
FROM     employees
GROUP BY department_id
HAVING   MAX(salary)>10000 ;
```

DEPARTMENT_ID	MAX(SALARY)
20	13000
80	11000
90	24000
110	12000

لا يمكن استخدام WHERE لمنع مجموعة من المجموعات الناتجة حيث أن فقرة WHERE تستخدم لمنع الصفوف الفردية وليس صفوف المجموعة.ويتم استخدام عبارة HAVING بدلا من عبارة WHERE وذلك لإظهار مجموعات من البيانات دون الأخرى كما في المثال حيث لم تظهر المجموعة المستثناة وهي الإدارة 30 حيث انها لم تحقق الشرط و هو أن يكون متوسط مرتبها أكبر من 10000.

ملحوظة:

لايمكن استخدام الاسم المستعار (Alias) مع HAVING . كما فى **المثال التالى:**

```
Select job, max (sal) "MAX_SAL"
 From EMP
 Group by job
 Having MAX_SAL>100;
```

```
  Having MAX_SAL>100
         *

ERROR at line 4:
ORA-00904: "MAX_SAL": invalid identifier
```

ولمعالجة المثال السابق يجب عدم استخدام الاسم المستعار فى فقرة HAVING **كما يلى:**

```
Select job , MAX(SAL)  "MAX_SAL"
 From EMP
 Group by job
 Having MAX(SAL) >100;
```

JOB	MAX_SAL
ANALYST	3000
CLERK	1300
MANAGER	2975
PRESIDENT	5000
SALESMAN	1600

امكانية استخدام اكثر من **FUNCTIONS** كما يلى
فالمثال التالى يريد عرض أكبر متوسط للمرتبات بالنسبة للاقسام :

```
SELECT   MAX(AVG(salary))
FROM     employees
GROUP BY department_id;
```

MAX(AVG(SALARY))
19333.3333

فهنا تم حساب المتوسط لكل إدارة ثم بعد ذلك تم عرض المتوسط الأكبر.

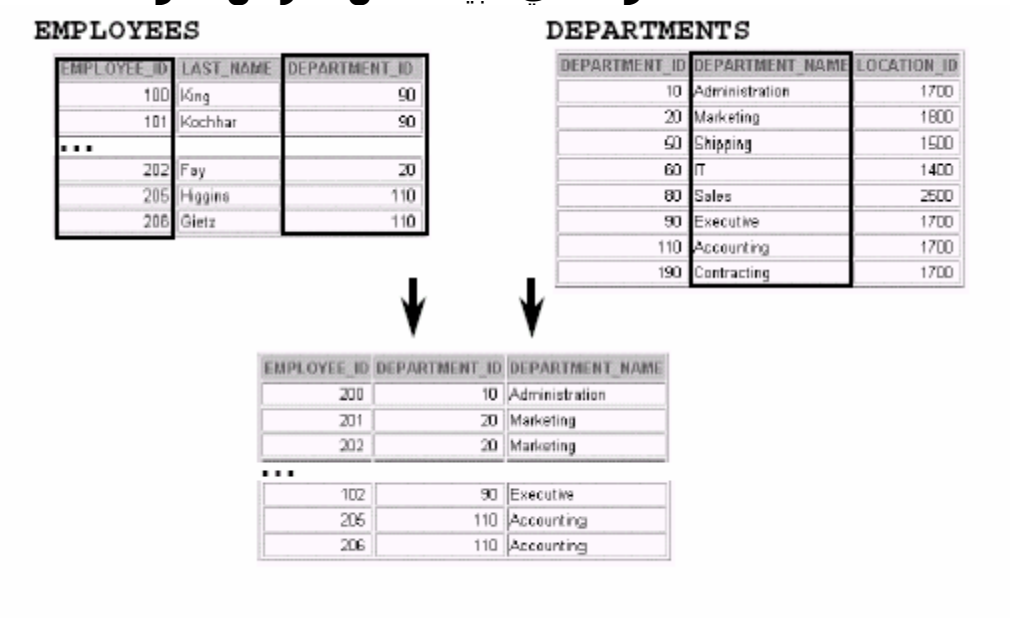
الفصل الخامس

عرض البيانات من اكثر من جدول

الأهداف:

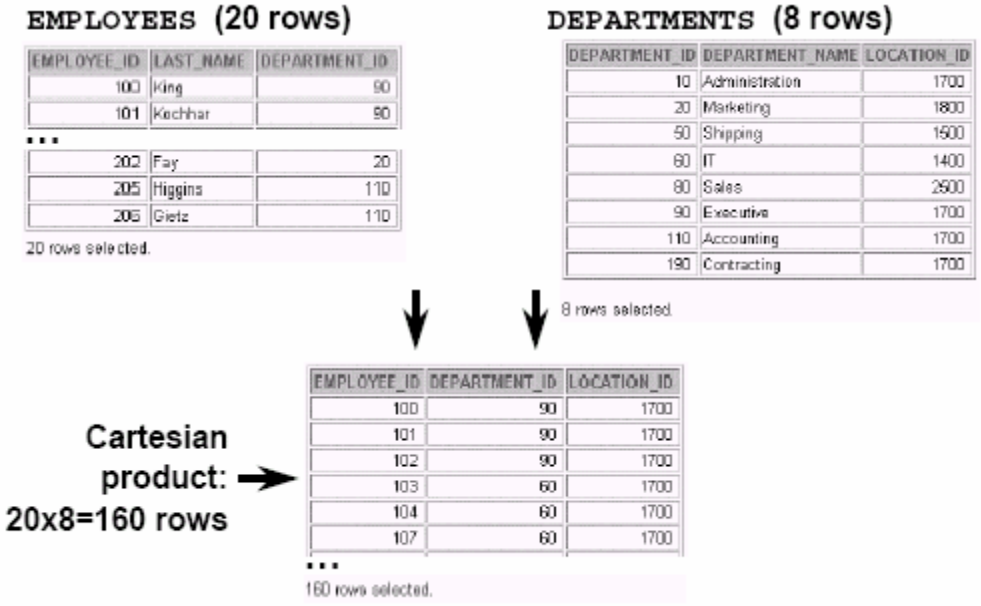
1. استخدام جملة SELECT لاستخلاص بيانات من أكثر من جدول وذلك باستخدام طرق الربط المختلفة .
2. إستخلاص البيانات التي لا تقابل شرط الربط باستخدام Outer Join.
3. ربط عمودين بنفس الجدول ويسمي Self Join.

الحصول علي البيانات من أكثر من جدول



فقد تم اختيار العمود DEPARTMENT_NAME من جدول DEPARTMENTS واختيار العمود EMPLOYE_ID والعمود DEPARTMENT_ID من جدول EMPLOYEES فتم عرض هذه البيانات معا في جدول جديد وهذا ما يسمي بالربط. Cartesian Products عندما لانضع شرط الربط او نضع شرط ربط خاطأ وتكون النتيجة عرض عدد كبير من الصفوف وتكون النتيجة غير ذات معني. كما في المثال التالي:

Cartesian Products



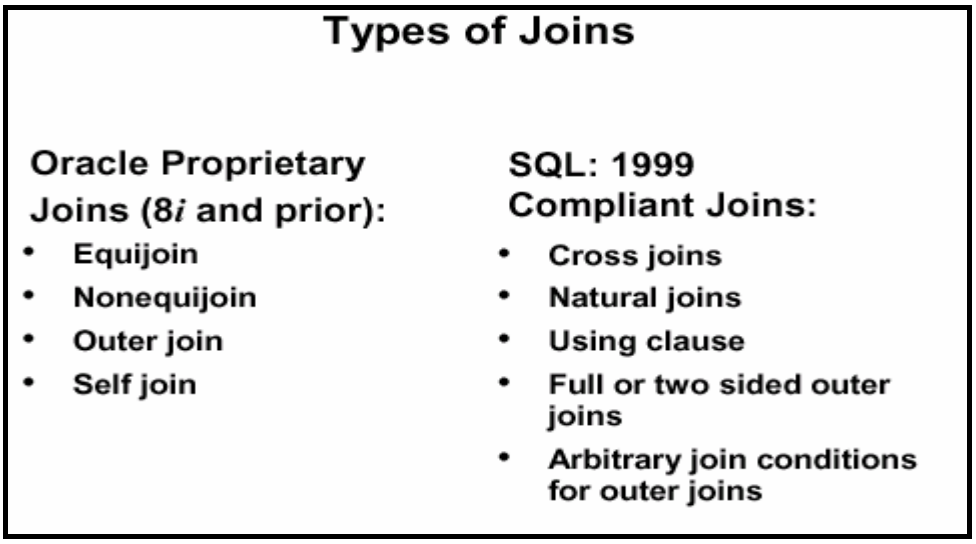
وكما ذكرنا ففي هذا المثال بسبب حذف الشرط WHERE تم ربط كل الصفوف فى جدول الموظفين مع كل الصفوف فى جدول الاقسام و بذلك يكون الناتج 160 صفوف.

```
SELECT last_name, department_name dept_name
FROM   employees, departments;
```

LAST_NAME	DEPT_NAME
King	Administration
Kochhar	Administration
De Haan	Administration

...
160 rows selected.

أنواع الروابط



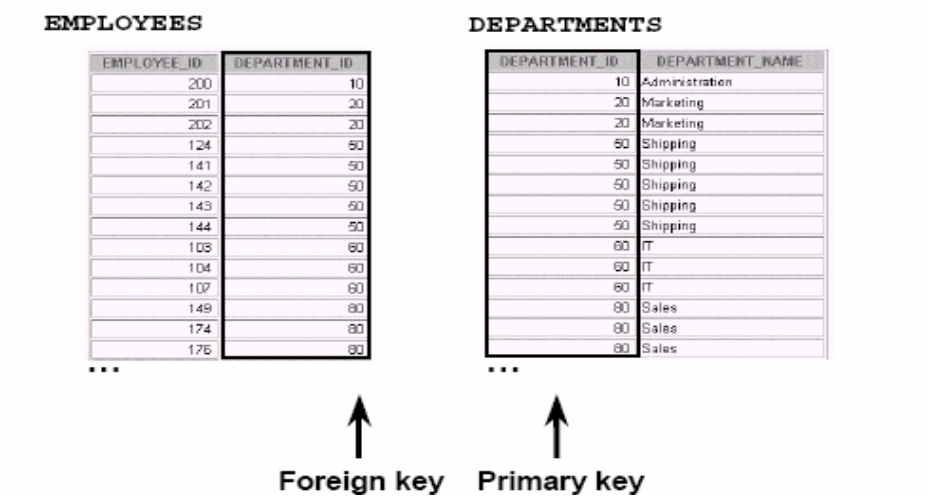
يستخدم الربط لعرض البيانات من أكثر من جدول

```
SELECT   table1.column, table2.column
FROM     table1, table2
WHERE    table1.column1 = table2.column2;
```

يستخدم الأمر SELECT عن طريق كتابة اسم الجدول المراد الاختيار منة table1 واسم العمود المراد عرضة column1 ويفصل بينهم بنقطة .
ثم الجدول الثاني المراد ربطة و أسم العمود الذي يراد عرضه column2 ويفصل بينهم بنقطة .
ثم الشرط WHERE ليتم ربط الجدولين معا حيث يتم مساواة قيم العمودين في الجدولين.
وهنا نجد ان عمود رقم الإدارة في الجدول الأول للموظفين يتساوى مع عمود رقم الإدارة في الجدول الثاني الإدارات.

*لكي تَرْبطَ مجموعة من الجداول معا ، كمثال لَرْبطَ أربعة جداول، يتطلب ذلك ثلاثة روابط كحد أدني.

ما هو الربط المتساوي Equijoin؟



الربط المتساوي

وفية يتم تساوي بين قيم عمود بالجدول الأول مع قيم عمود بالجدول الثاني .
ويتضمن هذا النوع استخدام المفتاح الرئيسي والمفتاح الأجنبي
(Primary Key و Foreign Key) ويسمي هذا النوع من بالربط البسيط أو الربط الداخلي
(Simple Join و Inner Join).

مثال يوضح الربط بطريقة EquiJoin

```
SELECT employees.employee_id, employees.last_name,
       employees.department_id, departments.department_id,
       departments.location_id
FROM   employees, departments
WHERE  employees.department_id = departments.department_id;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	50	1500
142	Davies	50	50	1500
143	Matos	50	50	1500
144	Vargas	50	50	1500

...
19 rows selected

وكما نري فى المثال السابق فى فقرة WHERE يحدد شرط الربط المستخدم هو إشارة التساوي (=) وهو:

EMPLOYEES.DEPARTMENT_ID = DEPARTMENTS.DEPARTMENT_ID

لأن عمود DEPARTMENT_ID موجود فى كلتا الجدولين، ويجِب أن يكون مُسبق بأسم الجدول الخاص بة كي نتجنب الغموض .

إضافة شرط آخر من شروط الربط عن طريق استخدام المعامل and

EMPLOYEES

LAST_NAME	DEPARTMENT_ID
Whalen	10
Hartstein	20
Fay	20
Mourgos	50
Rajs	50
Davies	50
Matos	50
Vargas	50
Hunold	60
Ernst	60

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
20	Marketing
50	Shipping
50	Shipping
50	Shipping
50	Shipping
50	Shipping
60	IT
60	IT

من الممكن إضافة شرط آخر عن طريق استخدام معامل and فمثلا:

AND last_name = 'Matos'

حيث يتم إضافة AND بعد الشرط فى WHERE
ثم يلي AND الشرط الجديد.

```
SELECT last_name, employees.department_id,
```

department_name
FROM employees, departments
WHERE employees.department_id = departments.department_id
AND last_name = 'Matos';

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Matos	50	Shipping

ففى **المثال** السابق تم عرض الاسم و رقم القسم واسم القسم للموظف Matos.

استخدام الأسماء المستعارة للجداول

* استخدام الأسماء المستعارة للجداول يبسط عملية الاستعلام .
* يحسن الأداء باستخدام اول الحروف من اسماء الجداول .

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e , departments d
WHERE  e.department_id = d.department_id;
```

من الممكن استخدام الأسماء المستعارة للجداول المراد ربطها وهي أسماء سهلة مؤقتة بالجداول. تستخدم فقط أثناء تنفيذ أمر SELECT حيث تفيد في اختصار الوقت وكتابة الأمر.
ويتم تعريف Alias في أمر FROM لكل جدول.
كما هو موضح بالمثال

↓

FROM employees e , departments d

بحيث يكتب أسم الجدول كاملا بتبعة الاسم المستعار لة ففي المثال السابق كان أسم الجدول employees و الاسم المستعار لة e
وأسم الجدول الثانى department والاسم المستعار لة d .

1. الأسماء المستعارة للجداول يجب أن تكون فى حدود 30 حرف في الطولِ لكن إذا كانت أقصر في الطول كان ذلك أفضل .
2. الاسم المستعار للجدول يكون في فقرة FROM ، إذن ذلك الاسم المستعار للجدول يجِبُ أن يسبق اسم العمود المراد عرضة.
3. الأسماء المستعارة للجداول يجِب أن تكُون ذات مغزى .

من الممكن الربط بين أكثر من جدول

EMPLOYEES

LAST_NAME	DEPARTMENT_ID
King	90
Kochhar	90
De Haan	90
Hunold	60
Ernst	60
Lorentz	60
Mourgos	50
Rajs	50
Davies	50
Matos	50
Vargas	50
Zlotkey	80
Abel	80
Taylor	80

20 rows selected.

DEPARTMENTS

DEPARTMENT_ID	LOCATION_ID
10	1700
20	1800
50	1500
60	1400
80	2500
90	1700
110	1700
190	1700

8 rows selected.

LOCATIONS

LOCATION_ID	CITY
1400	Southlake
1500	South San Francisco
1700	Seattle
1800	Toronto
2500	Oxford

ولكي تكون هذه العملية صحيحة فإن عدد عمليات الربط دائما تكون أقل من عدد الجداول المستخدمة بواحد.

حيث أن هناك ثلاث جداول تم الربط بينهم بعمليتي ربط فقط وهما:
e.department_id = d.department_id
d.location_id = l.location_id

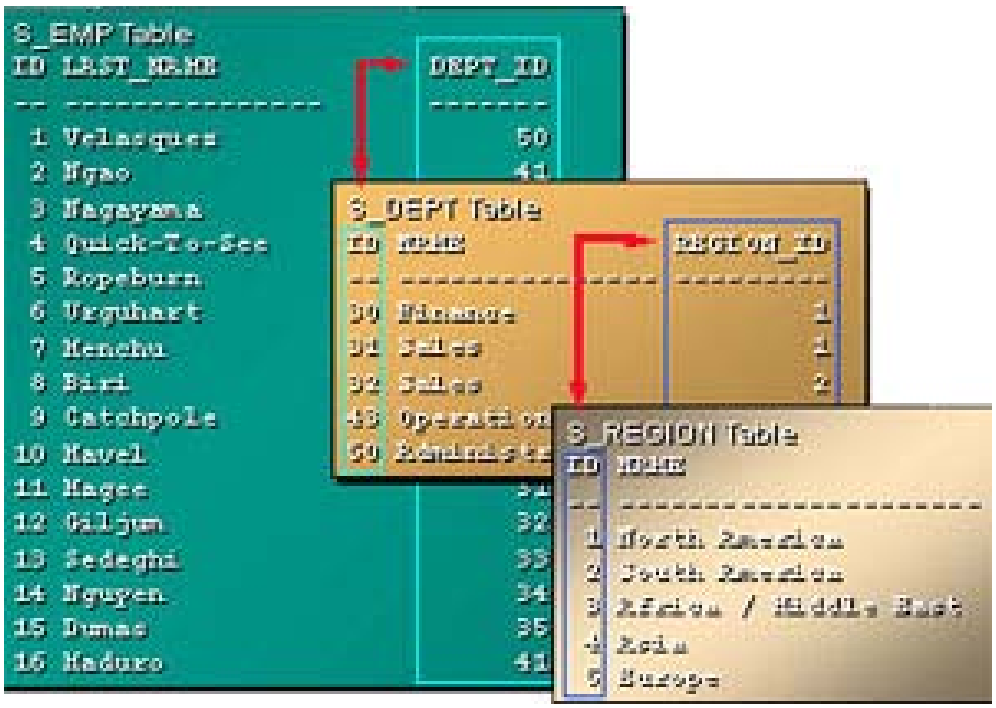
```
SELECT e.last_name, d.department_name, l.city
FROM   employees e, departments d, locations l
WHERE  e.department_id = d.department_id
AND    d.location_id = l.location_id;
```

LAST_NAME	DEPARTMENT_NAME	CITY
Hunold	IT	Southlake
Ernst	IT	Southlake
Lorentz	IT	Southlake
Mourgos	Shipping	South San Francisco
Rajs	Shipping	South San Francisco
Davies	Shipping	South San Francisco

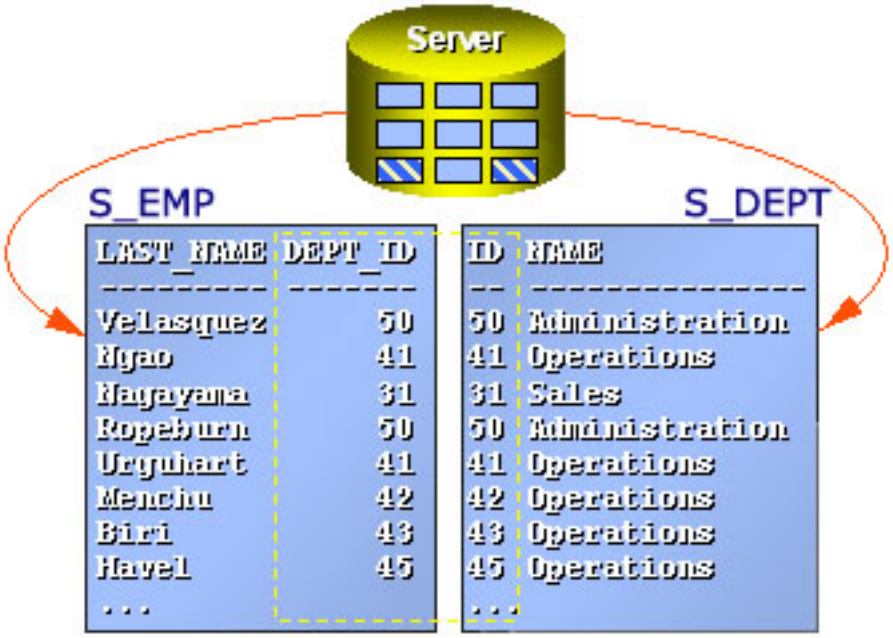
. . .
19 rows selected.

ففى **المثال** السابق تم الربط بين ثلاث جداول Employees ,Department Location وذلك لعرض الاسم و اسم الادارة والمدينة التى بها تلك القسم.

More examples



نستطيع استرجاع بيانات من أكثر من جدول اذا كان بينهم علاقة Relation حيث كل صف من احدهما يرتبط مع صف من الاخر عن طريق حقل مشترك




```
NAME, S_DEPT.NAME _SELECT S_EMP.DEPT_ID, S_EMP.LAST
FROM S_EMP, S_DEPT
WHERE S_EMP.DEPT_ID = S_DEPT.ID;
```

DEPT_ID	*LAST _NAME*	*NAME*
50	VELASQUEA	ADMINISTRATION
41	NGAO	OPERATOPN
31	NAGAYAMA	SALES
50	OPEBURN	ADMINISTRATION
41	RGIHART	OPERATOPNS
42	MENCHU	OPERATOPNS
43	BIRI	OPERATOPNS
45	HAVEL	OPERATOPNS

ويمكن اضافة شرط اخر بعد WHERE وذلك باستخدام AND فنكتب

```
SELECT S_EMP.DEPT_ID, S_EMP.LAST _NAME, S_DEPT.NAME
FROM S_EMP, S_DEPT
WHERE S_EMP.DEPT_ID = S_DEPT.ID  AND S_DEPT = 41;
```

عند التنفيذ ينتج :

DEPT_ID	*LAST _NAME*	*NAME*
41	NGAO	OPERATOPNS
41	URGIHART	OPERATOPNS

الربط الغير متساوي
NonEquiJoin

EMPLOYEES

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000
Ernst	6000
Lorentz	4200
Mourgos	5800
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500
Zlotkey	10500
Abel	11000
Taylor	8600

...
20 rows selected.

JOB_GRADES

GRA	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

المرتبات في جدول الموظفين يجب أن تكون بين أقل مرتب وأعلي مرتب في جدول JOB_GRADES .

الربط الغير متساوي (NonEquiJoin) توجد به علاقة غير مباشرة لربط جدولين ولا تستخدم فيه إشارة التساوي(=).

والعلاقة بين عمود المرتبات في جدول الموظفين يجب أن تكون بين أقل مرتب وأعلي مرتب في جدول JOB_GRADES.

ففى المثال التالى يقوم بمعرفة مركز او موقع مرتب كل موظف وذلك بمعرفة نطاق كل شريحة من المرتبات (بدايتها ونهايتها) ومقارنتها بمرتب كل موظف فى جدول الموظفين. إذا كان ضمن الشريحة فأنه يعرض اسم الشريحة.كما فى المثال التالى...

مثال يوضح الربط الغير المتساوي NonEquiJoin

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e, job_grades j
WHERE  e.salary
       BETWEEN j.lowest_sal AND j.highest_sal;
```

LAST_NAME	SALARY	GRA
Matos	2600	A
Vargas	2500	A
Lorentz	4200	B
Mourgos	5800	B
Rajs	3500	B
Davies	3100	B
Whalen	4400	B
Hunold	9000	C
Ernst	6000	C

...
20 rows selected.

حيث لا توجد علاقة مباشرة بين الجدولين employees e, job_grades المراد الربط بينهم ولذلك تم استخدام BETWEEN بدلا من يساوي في شرط الربط. لاحظ ان قيمة المرتب للموظف تمتد بين أقل قيمة وأعلي قيمة مرتب في جدول درجات المرتب.

كل مرتبات الموظفين ممتدة بواسطة جدول درجات. الوظيفة JOB_GRADE فلا يوجد موظف يكسب أقل من أصغر قيمة في عمود LOWEST_SAL ولا يوجد موظف يكسب أكثر من العمود HIGHEST_SAL .

ملاحظة:

(=>)و (=<) يمكن أن تستخدم و لكن BETWEEN للتبسيط . تذكر عند تحديد القيم يجب ان تكون أول قيمة منخفضة وثاني قيمة مرتفعة وذلك عند استخدام BETWEEN.

الربط الخارجي(outer join)

يستخدم لعرض صفوف موجودة بالجدول ولكن لا ينطبق عليها شرط الربط
ويتم ذلك باستخدام إشارة الجمع (+) مع شرط الربط وتوضع في جهة العمود المراد عرض بياناته .

DEPARTMENTS

DEPARTMENT_NAME	DEPARTMENT_ID
Administration	10
Marketing	20
Shipping	50
IT	60
Sales	80
Executive	90
Accounting	110
Contracting	190

8 rows selected.

EMPLOYEES

DEPARTMENT_ID	LAST_NAME
90	King
90	Kochhar
90	De Haan
60	Hunold
60	Ernst
60	Lorentz
50	Mourgos
50	Rajs
50	Davies
50	Matos
50	Vargas
80	Zlotkey

...
20 rows selected.

لاحظ عدم وجود موظفين في القسم 190

بالنظر الى الجدولين نجد ان هناك ادارة موجودة فى جدول الاقسام وغير موجودة فى جدول الموظفين .فلنفترض ان هذة الادارة جديدة ولم يعين بها موظفين حتى الان. واردنا ان نرى تلك الادارة بطريقة الربط Equijoin كما فى المثال التالى فنجد اننا لا يمكن رؤيتها بتلك الطريقة ولذلك يتم استخدام طريقة الربط Outer join كما سوف نرى.

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e, departments d
WHERE  e.department_id = d.department_id;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Mourgos	50	Shipping

...
19 rows selected.

لاحظ ان تلك الصفوف ليست كاملة حيث انة لم يعرض الادارات التى ليس بها موظفين.ولمعالجة ذلك يتم استخدام **Outer Join .**

الربط الخارجي Outer Join

* يستخدم لكي يمكن رؤية الصفوف التي لا تقابل شرطَ الربط .
* معامل الربط الخارجي هو إشارةُ الزّائد (+).

```
SELECT table1.column, table2.column
FROM   table1, table2
WHERE  table1.column(+) = table2.column;

SELECT table1.column, table2.column
FROM   table1, table2
WHERE  table1.column = table2.column(+);
```

يتم استخدام إشارة الجمع جهة العمود المراد عرض جميع بياناته أثناء عملية الربط بين العمودين وتوضع علامة (+) جهة العمود الذي يوجد فيه النقص .ويجب وضع علامة(+) بين قوسين .
(+) *table2.column* رمز الربط الخارجي (+) يمكن أن يوضع علي جانب واحد فى شرط فقرة WHERE لكن لايمكن ان يوجد على كلا من الجانبين.

استخدام الربط الخارجي Outer Join

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e, departments d
WHERE  e.department_id(+) = d.department_id ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Mourgos	50	Shipping
Rajs	50	Shipping
Davies	50	Shipping
Matos	50	Shipping
...		
Gietz	110	Accounting
		Contracting

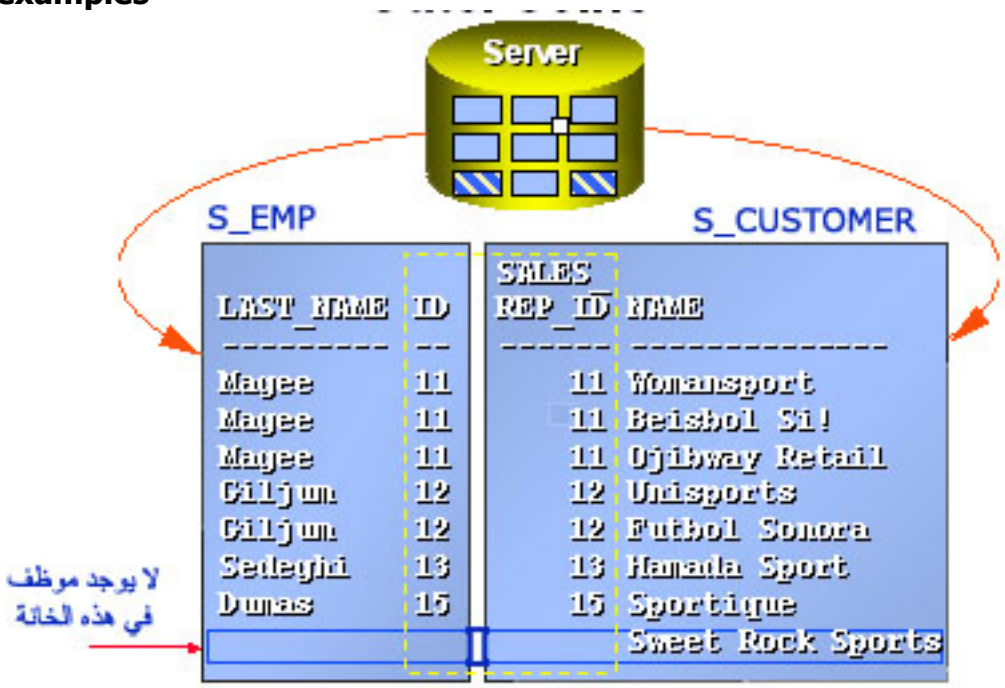
20 rows selected.

لا يوجد موظف في هذه الخانة

في هذا **المثال** حيث أنة تم استخدام إشارة الجمع جهة العمود المراد عرض جميع بياناته أثناء عملية الربط بين العمودين وتوضع علامة (+) جهة العمود الذي يوجد فيه النقص.والمثال يعرض last_name, department_id department_name.

لاحظ ان القسم Contracting لا يشتمل علي أي موظفين.و إن القيمة المعروضة فارغة وظاهرة في الناتج .

More examples



اذا اردنا الاستعلام عن الجزء الذي لا يتواجد به موظف نكتب

```
SELECT   s_emp.last_name, s_emp.id, , s_customer.name
FROM s_emp, s_customer
WHERE    s_emp.id (+) = s_customer.sales_rep_id
ORDER BY s_emp.id;
```

الربط الداخلي Self Join

EMPLOYEES (WORKER)

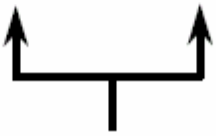
EMPLOYEE_ID	LAST_NAME	MANAGER_ID
100	King	
101	Kochhar	100
102	De Haan	100
103	Hunold	102
104	Ernst	103
107	Lorentz	103
124	Mourgos	100

...

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst
107	Lorentz
124	Mourgos

...



رقم الموظف في جدول الموظفين
يساوي رقم الموظف في جدول المدير

الربط الداخلي Self Join يستخدم لربط عمود بعمود آخر فى نفس الجدول.
وهنا فى هذة الحالة يجب ان تتخيل الجدول الواحد(الموظفين) جدولين. الجدول الاول للموظفين والثانى للمديرين حيث ان العمود Employee_ID يعتبر Primary Key و العمود Manager_ID يعتبر Foreign Key لة.
وبهذا يمكنك التعامل مع تلك الجدولين بالطرق Equijoin او Outer join.

```
SELECT worker.last_name || ' works for '
       || manager.last_name
FROM   employees worker, employees manager
WHERE  worker.manager_id = manager.employee_id;
```

WORKER.LAST_NAME||'WORKSFOR'||MANAGER.LAST_NAME
Kochhar works for King
De Haan works for King
Mourgos works for King
Zlotkey works for King
Hartstein works for King
Whalen works for Kochhar
Higgins works for Kochhar
Hunold works for De Haan
Ernst works for Hunold

...

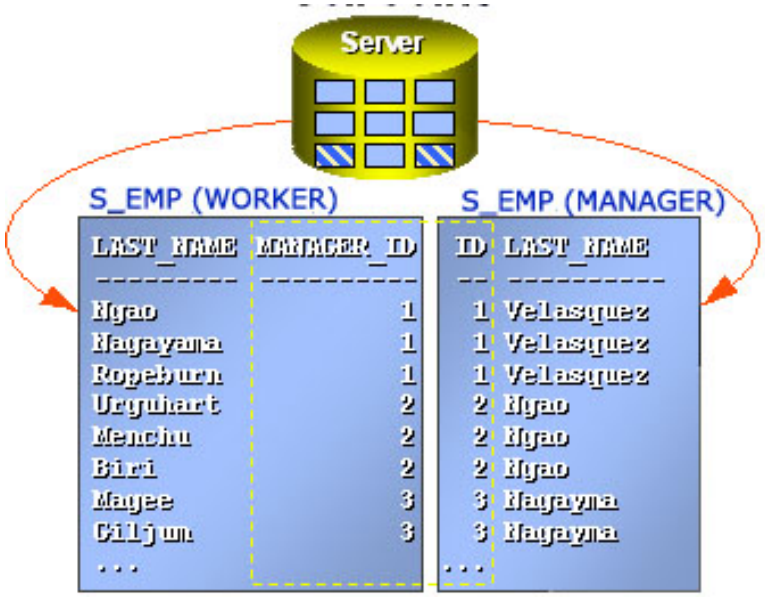
19 rows selected

فى **المثال** السابق يتم إظهار كل موظف بجانب مديرة وبما أن الربط بين عمودين في نفس الجدول فقد تم استخدام الاسم المستعار Employees worker, employees manager وتم استخدامها كي يتم اعتبار الجدول عبارة عن جدولين منفصلين ويمكن ربطهما بين عمودين داخل نفس الجدول وقد سمي جدول الموظفين باسمين الأول وهو Worker ويمثل الموظفين والثاني Manager وهو المدير .

حيث أن رقم المدير هو أصلا رقم موظف لذا فقد تم مساواة رقم الموظف بعمود أرقام الموظفين في جدول المديرين Manager بأرقام المديرين بعمود المديرين في جدول الموظفين Worker
حيث أن الموظف في جدول المديرين هو مدير في جدول الموظفين.

* تم اظهار 19 موظف فقط ،مع ان هناك 20 موظف و هذاحدث لأن الموظف King هو الرّئيس ولا يرئسة احد وبالتالى لا يوجد لة رقم مدير فى العمود Manager_ID ولعرض KING لابد من استخدام Outer Join .

More examples



مثال :

```
SELECT   worker.last_name||' works for '||
manager.last_name
FROM s_emp worker, s_emp manager
WHERE    worker.manager_id = manager.id;
```

عند التنفيذ سيتم طباعة

```
worker.last_name||' works for '||
manager.last_name
-----------------------------------------------
Nago works for Velasquez
Nagayama works for Velasquez
Ropeburn works for Velasquez
Urguhart works for Nago
Biri works for Nago
Magee works for Nagayama
Giljum Nagayama
```

ربطَ الجداول بطرق الربط بعد سنة 1999

الشكل الاساسى لتلك الطرق:

```
SELECT   table1.column, table2.column
FROM     table1
[CROSS JOIN table2] |
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2
  ON(table1.column_name = table2.column_name)] |
[LEFT|RIGHT|FULL OUTER JOIN table2
  ON (table1.column_name = table2.column_name)];
```

* CROSS JOIN يساوى Cartesian product.
* NATURAL JOIN يربطُ جدولين معتمد على نفس أسم العمودِ بالجدولين.
* USING column_name JOIN table يساوى الربط البسيط (Equijoin) معتمد علي أسم عمودِ.
* JOIN table ON يساوى الربط البسيط ايضا معتمد علي الشّرطِ بعد ON .
* LEFT/RIGHT/FULL OUTER Join وهو نفس فكرة Outer Join.

(cross join)

وهي نفس Cartesian Product .

```
SELECT last_name, department_name
FROM   employees
CROSS JOIN departments ;
```

LAST_NAME	DEPARTMENT_NAME
King	Administration
Kochhar	Administration
De Haan	Administration
Hunold	Administration

...

160 rows selected.

المثال يعطي نفس النتائج باستخدام الطريقة الاولى Cartesian Product

```
SELECT last_name, department_name
FROM   employees, departments;
```

LAST_NAME	DEPARTMENT_NAME
King	Administration
Kochhar	Administration
De Haan	Administration
Hunold	Administration
Ernst	Administration

...

160 rows selected.

Natural Join

*يعتمد هنا على أعمدة الربط في الجدولين الذين لها نفس الاسم .
* يختار الصفوف من الجدولين التي تكون قيمتها متساوية في كل الأعمدة المتناظرة.
* إذا كانت الأعمدة لها نفس الأسماء ولكن نوع البيانات مختلف يحدث خطأ عند الاستخلاص.

```
SELECT department_id, department_name,
       location_id, city
FROM   departments
NATURAL JOIN locations ;
```

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
60	IT	1400	Southlake
50	Shipping	1500	South San Francisco
10	Administration	1700	Seattle
90	Executive	1700	Seattle
110	Accounting	1700	Seattle
190	Contracting	1700	Seattle
20	Marketing	1800	Toronto
80	Sales	2500	Oxford

8 rows selected.

وسوف نعرض نفس النتيجة باستخدام Equijoin

```
SELECT department_id, department_name,
departments.location_id, city
FROM departments, locations
WHERE departments.location_id = locations.location_id;
```

شرط إضافي علي Natural Join في فقرة WHERE.
فيتم تحديد صفوف النّاتج للاقسامِ 20 أو 50 .

```
SELECT department_id, department_name,
location_id, city
FROM departments
NATURAL JOIN locations
WHERE department_id IN (20, 50);
```

إنشاء الربط باستخدام USING

* إذا كانت الأعمدة لها نفس الأسماء لكن أنواع البيانات غير مطابقة فيمكنك أنْ تستخدم USING في تحديد الأعمدة التي يجِبُ ان يتم الربط بها .
* تستخدم فقرة USING كي تطابق عمود واحدفقط عندما يكون أكثر من عمود متماثل.
* لا يستخدم أسم الجدول أو الأسم المستعار في الأعمدة التي إسترجعت .

Natural Join تستخدم الأعمدةِ بمطابقة الأسماء وأنواع البياناتِ لربط الجداول.
USING يمكن أن تستخدم كي تحدد فقط تلك الأعمدةِ التي يجِب الربط بها.
كمثال

```
SELECT l.citya, d.department_name
FROM locations l JOIN departments d USING (location_id)
WHERE location_id = 1400;
```

لكن هذا **المثال** خطأ لأنة لا يمكن استخدام Alias في الفقرة WHERE

```
SELECT l.city, d.department_name
FROM locations l JOIN departments d USING (location_id)
WHERE d.location_id = 1400;
```

لا يمكن أن تشتمل
علي Alias

ORA-25154: column part of USING clause cannot have qualifier

مثال على USING

```
SELECT e.employee_id, e.last_name, d.location_id
FROM   employees e JOIN departments d
USING (department_id) ;
```

EMPLOYEE_ID	LAST_NAME	LOCATION_ID
200	Whalen	1700
201	Hartstein	1800
202	Fay	1800
124	Mourgos	1500
141	Rajs	1500
142	Davies	1500
143	Matos	1500
144	Vargas	1500
103	Hunold	1400

...
19 rows selected.

المثال يظهر ربط عمود رقم القسم بجدول الموظفين بجدول الأقسام وتعرض لنا موقع مكان عمل الموظف.

وهذا **المثال** يمكن تنفيذة بـ **Equijoin** كما يلى:

```
SELECT employee_id, last_name,
employees.department_id, location_id
FROM employees, departments
WHERE employees.department_id = departments.department_id;
```

مثال على ON

تستخدم فقرة ON كي تحدد شرط الربط بين الجدولين.

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id);
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	50	1500
142	Davies	50	50	1500
143	Matos	50	50	1500

...
19 rows selected.

فقرة ON يمكن أن تربط أعمده بها أسماء مختلفة:

```
SELECT e.last_name emp, m.last_name mgr
FROM   employees e JOIN employees m
ON     (e.manager_id = m.employee_id);
```

EMP	MGR
Kochhar	King
De Haan	King
Mourgos	King
Zlotkey	King
Hartstein	King
Whalen	Kochhar

...
19 rows selected.

إن **المثال** السابق يعد Self Join اى (ربط داخلي) لجدول الموظفين بنفسه معتمد علي أعمدة رقم الموظف ورقم المدير بنفس الجدول .

امكانية استخدام اكثر من طريقة ربط

```
SELECT employee_id, city, department_name
FROM   employees e
JOIN   departments d
ON     d.department_id = e.department_id
JOIN   locations l
ON     d.location_id = l.location_id;
```

EMPLOYEE_ID	CITY	DEPARTMENT_NAME
103	Southlake	IT
104	Southlake	IT
107	Southlake	IT
124	South San Francisco	Shipping
141	South San Francisco	Shipping
142	South San Francisco	Shipping
143	South San Francisco	Shipping
144	South San Francisco	Shipping

...

19 rows selected.

ويمكن تنفيذ نفس **المثال** السابق بطريقة Equijoin :

```
SELECT employee_id, city, department_name
FROM employees, departments, locations
WHERE employees.department_id = departments.department_id
AND departments.location_id = locations.location_id;
```

ويمكن تنفيذ نفس **المثال** بطريقة الربط USING :

```
SELECT e.employee_id, l.city, d.department_name
FROM employees e
JOIN departments d
USING (department_id)
JOIN locations l
USING (location_id);
```

قبل سنة 1999	بعد سنة 1999
Oracle	**SQL: 1999**
Equi-Join	Natural/Inner Join
Outer-Join	Left Outer Join
Self-Join	Join ON
Non-Equi-Join	Join USING
Cartesian Product	Cross Join

Left Outer Join

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e
LEFT OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
...		
De Haan	90	Executive
Kochhar	90	Executive
King	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
Grant		

20 rows selected.

وهذا يعنى ان النقص فى d.department_id وبدلا من وضع (+) عند تلك العمود كما فى الطريقة Outer Join يتم استخدام LEFT OUTER JOIN.

ملحوظة: هنا توضع علامة(+) يسار الشاشة وليس يسار رؤيتك.
والمثال التالى يوضح ذلك:

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE d.department_id (+) = e.department_id;
```

Right Outer Join

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e
RIGHT OUTER JOIN departments d
ON     (e.department_id = d.department_id) ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
King	90	Executive
Kochhar	90	Executive
...		
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Higgins	110	Accounting
Gietz	110	Accounting
		Contracting

20 rows selected.

وهذا يعنى ان النقص فى e.department_id وبدلا من وضع (+) عند تلك العمود كما فى الطريقة Outer Join يتم استخدام Right OUTER JOIN.

ملحوظة:
هنا توضع علامة(+) يمين الشاشة وليس يمين رؤيتك.
والمثال التالى يوضح ذلك:

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE d.department_id = e.department_id (+);
```

Full outer join

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e
FULL OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
...		
De Haan	90	Executive
Kochhar	90	Executive
King	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
Grant		
		Contracting

21 rows selected.

هذا الاستعلام يسترجع كل الصفوف فى الجدولين حتى إذا كانت غير متماثلة

شروط إضافية

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id)
AND    e.manager_id = 149 ;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
174	Abel	80	80	2500
176	Taylor	80	80	2500

يمكنك ان تستخدم بعض الشروط الاضافية بعد الربط بين الجدولين لتحديد عدد معين من الصفوف.

أنواع الروابط	
Equijoins	الروابط المتساوية (البسيطة)
Non-equijoins	الروابط الغير متساوية
Outer joins	الروابط الخارجية
Self joins	الروابط الداخلية
Cross joins	الروابط التقاطعية
Natural joins	الروابط الطبيعية
Full or outer joins	الروابط المتكاملة أو الروابط الخارجية

ملخص الفصل.

تناولنا في هذا الفصل جمل SQL الخاصة باسترجاع بيانات من اكثر من جدول كغرض رقم اقسم الموظفين من جدول الموطفين وتطرقنا كذلك إلى عملية استرجاع البيانات من خلال اسم مستعار

الفصل السادس

Subqueries

هدف الدرس:

معرفة انواع و مميزات استخدام Subqueries حيث انها تعالج الكثير من المشاكل.

أنت يمكن أن تكتب استعلام فرعي في فقرة WHERE .

لنفترض أنك تريد معرفة الموظفين الذين تزيد رواتبهم عن الموظف SMITH . لكى تحل هذه المشكلة تحتاج الى أستعلامين:

استعلام لإيجاد راتب الموظف SMITH .

استعلام لإيجاد الموظفين الذين تزيد رواتبهم عن المبلغ المحسوب في الاستعلام الأول.

الاستعلام الفرعي

مقدمة

لنفترض أنك ترغب في كتابة استعلام لإيجاد بيانات الموظفين الذين تزيد رواتبهم عن راتب أحد الموظفين

في هذه الحالة تحتاج الى استعلامين

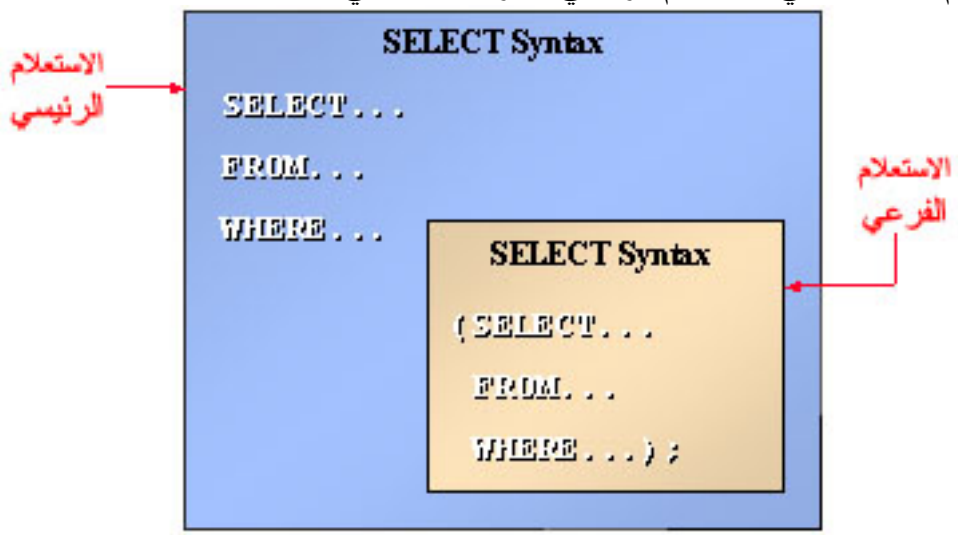
1- استعلام لايجاد راتب الموظف المعني
2- استعلام لايجاد الموظفين الذين تزيد رواتبهم عن المبلغ المحسوب في الاستعلام الأول

ويمكنك دمج الاستعلامين وذلك برتكيب احد الاستعلامين في الاخر ، الاستعلام الداخلي سوف يعود بقيمة (قيم) والتي يستخدمها الاستعلام الخارجي (الرئيسي)

إن استخدام الاستعلامات الفرعية يشبه تماما تنفيذ الاستعلامين بشكل متتال واستلاعم نتيجة الاستعلام الاول كنتيجة بحث في الاستعلام الثاني

تعريف الاستعلام الفرعي

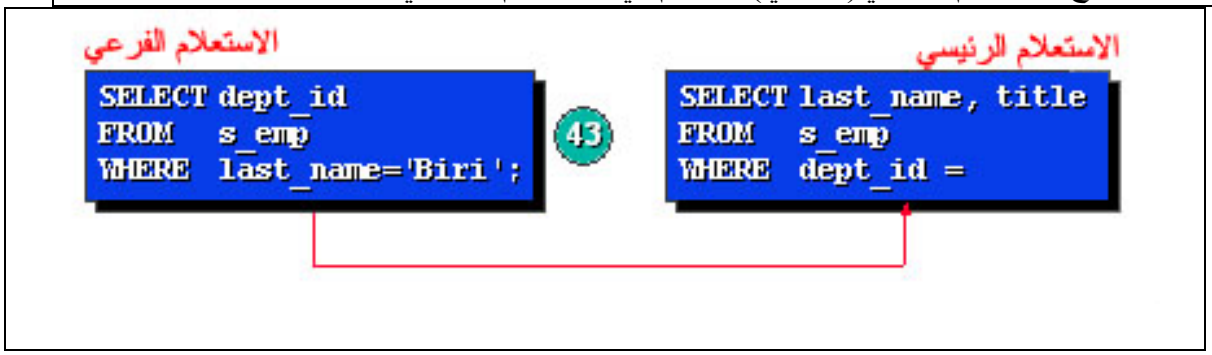
هو جملة استفسار SELECT مضمنة داخل جملة استفسار رئيسية لاسترجاع قيمة أو مجموعة من القيم ليتم استخدامها في الاستعلام الرئيسي انظر الشكل التالي



الشكل العام

```
SELECT  select_listFROM  tableWHERE  expr operator
        (SELECT  select_list
         FROM        table);
```

ملاحظة
1- الاستعلام الداخلي(الفرعي) ينفذ اولا 2- ناتج الاستعلام الداخلي(الفرعي) يستخدم في الاستعلام الرئيسي



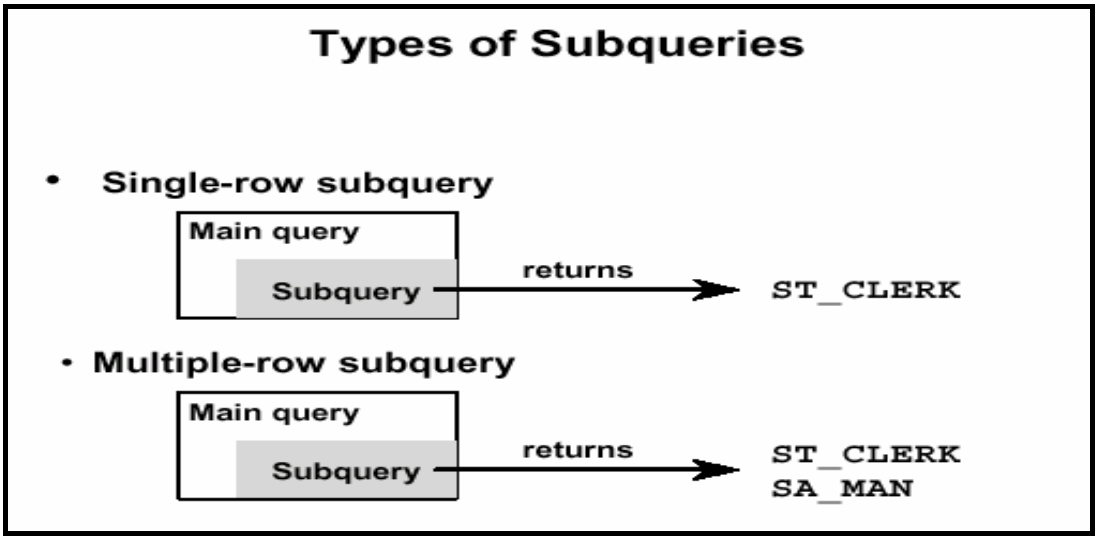
متطلبات الاستعلام الفرعي

1- وضع الاستعلام الفرعي بين قوسين (.....)
2- وضع الاستعلام الفرعي يمين عملية المقارنة.
3- لا يمكن استخدام ORDER BY داخل الاستعلام الفرعي
4- استخدام المعاملات أحادية الصف مع الاستعلامات الفرعية الأحادية الصف
5- استخدام المعاملات متعددة الصف مع الاستعلامات الفرعية المتعددة الصف

أنواع الاستعلامات الفرعية

يمكن تصنيف الاستعلام الفرعي حسب

1- استعلام فرعي أحادي الصف يسترجع صفا واحدا
2- استعلام فرعي متعدد الصفوف يسترجع أكثر من صف واحد



SINGLE ROW SUBQUERY

* استرجاع صف واحد فقط , استخدام عوامل المقارنة مع الصفوف الأحادية .

المعاملات
=
>
>=
<
<=
<>

مـثـال : عرض الموظفين الذين وظيفتهم نفس وظيفة الموظف 141 .

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id =
                (SELECT job_id
                 FROM   employees
                 WHERE  employee_id = 141);
```

LAST_NAME	JOB_ID
Rajs	ST_CLERK
Davies	ST_CLERK
Matos	ST_CLERK
Vargas	ST_CLERK

استخدام الاستعلام الفرعي أحادية الصف

ويمكن استخدام Sub Query اكثر من مرة كما فى **المثال** التالى:

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id =
                (SELECT job_id
                 FROM   employees
                 WHERE  employee_id = 141)
AND    salary >
                (SELECT salary
                 FROM   employees
                 WHERE  employee_id = 143);
```

ST_CLERK

2600

LAST_NAME	JOB_ID	SALARY
Rajs	ST_CLERK	3500
Davies	ST_CLERK	3100

يتضمن المثال علي ثلاث استعلامات: الاستعلام الخارجي واستعلامين داخلين.

الاستعلام الداخلى: الاول ينفذ اولا ويخرج لنا الوظيفة ST_CLERK

والاستعلام الداخلى: الثانى ينفذ بعدة و يخرج لنا المرتب 2600 ثم بعد ذلك يتم **الاستعلام الخارجى(OUTER QUERY)**: بناء على نتيجة الاستعلامين السابقين

كلتا الاستعلامات الداخلية تسترجع قيم واحدة (ST_CLEARK و 2600) علي التوالي وهذا يسمي استعلامات فرعية أحادية الصف.ولذلك تم استخدام (= و >).

ملاحظة: الاستعلامات الخارجية OUTER QUERY والداخلية INEER QUERY يمكن أن تحصل علي بيانات من جداول مختلفة **مثل :**

```
Select ename, job from EMP
Where deptno=any(select deptno from dept where dname='ACCOUNTING');
```

ENAME	JOB
CLARK	MANAGER
KING	PRESIDENT
MILLER	CLERK

ففى **المثال** السابق كانت نتيجة SubQuery من جدول الاقسام وتم مقارنتها مع بيانات من جدول الموظفين .

يمكن استخدام Group Functions مع Subquery

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  salary =
                (SELECT MIN(salary)
                 FROM   employees);
```

2500

LAST_NAME	JOB_ID	SALARY
Vargas	ST_CLERK	2500

استخدام HAVING مع Sub query

```
SELECT   department_id, MIN(salary)
FROM     employees
GROUP BY department_id
HAVING   MIN(salary) >
                (SELECT MIN(salary)
                 FROM   employees
                 WHERE  department_id = 50);
```

2500

- فقرة HAVING يمكن أن تستخدم مع الاستعلامات الفرعية (SubQuery) وليس فقط في الفقرة WHERE .

م ـ ث ـ ال ـ :

أوجد رقم الموظف واسمة للموظفين الذين ياخذون اقل مرتب فى كل قسم.

What Is Wrong with This Statement?

```
SELECT employee_id, last_name
FROM   employees
WHERE  salary =
                (SELECT MIN(salary)
                 FROM   employees
                 GROUP BY department_id);
```

Single-row operator with multiple-row subquery

```
ERROR at line 4:
ORA-01427: single-row subquery returns more than
one row
```

ما الخطأ في هذا الاستعلام؟

الخطأ هنا ان SubQuery ينتج عنها اكثر من نتيجة وتم مقارنتها بالمعامل يساوى (=)

ولمعالجة ذلك يجب استخدام الفقرة (ANY =) كما يلى:

Select employee_id, last_name from Employees

Where salary =any (select min (salary) from Employees group by department);

-هل هذا الاستعلام سوف يسترجع صفوف؟

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id =
                (SELECT job_id
                 FROM   employees
                 WHERE  last_name = 'Haas');
```

```
no rows selected
```

ملاحظة:

سبب النتيجة السابقة ان Sub Query كانت نتيجتها NULL .

الاستعلام الفرعي للصفوف المتعدد

* استرجاع أكثر من صف واحد
* استخدام معاملات المقارنة للصفوف المتعددة.

المعنى	المعامل
البحث عن قيمة من بين مجموعة من القيم	IN
يقارن قيمة باى قيمة مسترجعة بواسطة الاستعلامات الفرعية	ANY
مقارنة قيمة لكل القيم المسترجعة	ALL

استخدام معامل ALL

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ALL
                (SELECT salary
                 FROM   employees
                 WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

5000, 6000, 4200

141	Rajs	ST_CLERK	3500
142	Davies	ST_CLERK	3100
143	Matos	ST_CLERK	2600
144	Vargas	ST_CLERK	2500

المثال السابق يريد معرفة الموظفين الذين يقل مرتبهم عن مرتب الموظفين بوظيفة IT_PROG .

معامل ALL يقارن القيم بكل القيم التي استرجعت مِن SubQuery .

قيمة Null في الاستعلام الفرعي

```
SELECT emp.last_name
FROM   employees emp
WHERE  emp.employee_id NOT IN
                        (SELECT mgr.manager_id
                         FROM   employees mgr);

no rows selected
```

عندما تكون نتيجة SUb Query = NULL فهذا يعنى ان هذا الاستعلام تكون نتيجة NO ROWS SELECTED .

المعامل IN يساوى (=ANY)

Select ename, job from EMP where

Sal in (select max (sal) from EMP group by deptno);

ENAME	JOB
BLAKE	MANAGER
SCOTT	ANALYST
FORD	ANALYST
KING	PRESIDENT

المثال السابق يريد الاستعلام عن الموظفين الذين ياخذون اعلى مرتب بكل قسم.

يمكن استخدام (=ANY) بدلا من (IN) فى المثال السابق وتكون بنفس النتيجة :

select ename,job from EMP where

sal =any(select max(sal) from EMP group by deptno);

ENAME	JOB
BLAKE	MANAGER
SCOTT	ANALYST
FORD	ANALYST
KING	PRESIDENT

ملخص الفصل

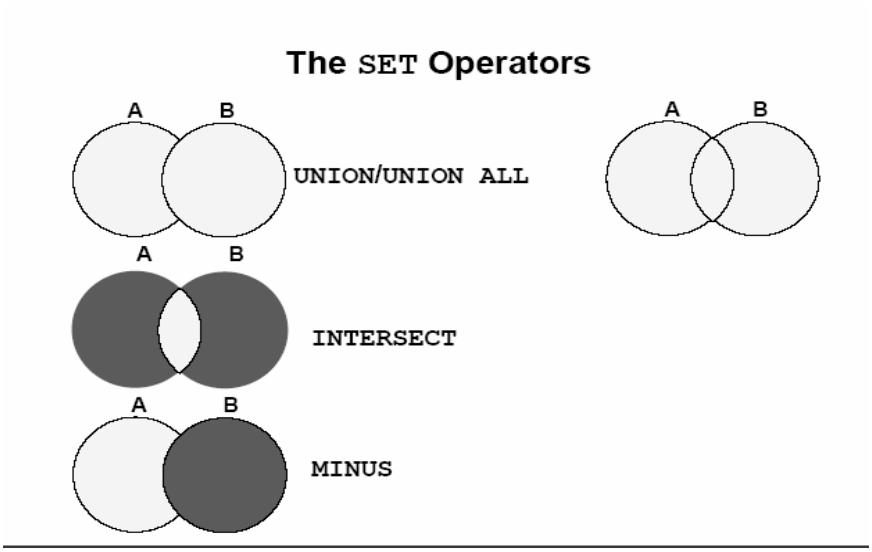
في هذا الفصل تعرفنا على الاستعلامات الفرعية واهميتها وشروطها وانواعها الأحادية والمتعددة

الفصل السابع

Using set operators

بعد الانتهاء من هذا الفصل يجب أن تكون على معرفة التالى :-

- **أنواع الـ set operator**
- **دمج أكثر من query داخل query واحد**

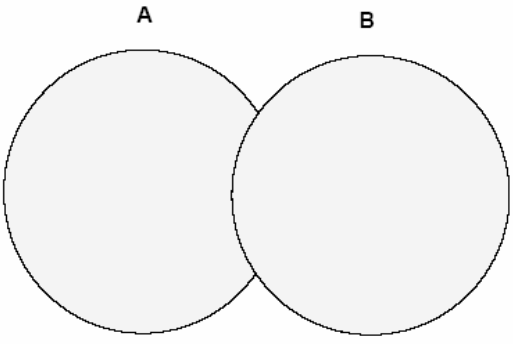


Union-: تعرض لك كل الصفوف الغير مكررة فى أى من الأستعلامات الموجودة.

Unionall-: تعرض كل الصفوف حتى المكرر منها.

Inteisect-: تعرض كل الصفوف الغير مكررة و المتشابهة فكل من الأستعلامات الموجودة.

Minus-: تعرض لك كل الصفوف غير المتشابهة المختارة من الـ query الاولى و لم تختار من خلال الـ query الثانى.



The UNION operator returns results from both queries after eliminating duplications.

Using the UNION Operator

Display the current and previous job details of all employees. Display each employee only once.

```
SELECT employee_id, job_id
FROM   employees
UNION
SELECT employee_id, job_id
FROM   job_history;
```

EMPLOYEE_ID	JOB_ID
100	AD_PRES
101	AC_ACCOUNT
101	AD_VP
178	SA_REP
200	AC_ACCOUNT
200	AD_ASST

28 rows selected.

Union:- ستعرض جميع الصفوف الناتجة عن الـ query الاولى و الثانية ونلاحظ أنها لم تعرض الصفوف المتكررة الناتجة من الـ query الاولى و الثانية.

* ولكن سنجد رقم الموظف 200 متكرر وهذا لأن الوظيفة تتغير كل مرة للموظف رقم 200 فهو مسجل فىجدول الـ employee بوظيفة Ad-AssT ويوجد أيضا فى جدول الـ Jop history مرة وظيفتة Ad-AssT و الأخرى AC-Account و لأن Union تمنع التكرار فآن الوظيفة AD-Asst لم تظهر مرتين.

ملحوظة الـ Null لا تحذف أثناء تكرارها.

عندما نستخدم Union أيضآ فى حالة أضافة Column الـ Department نلاحظ أن الموظف رقم 200 ظهر ثلاث مرات لانه فى الصف الثانى الوظيفة Ac-Asst و رقم القسم 10.

وفى الصف الثالث الوظيفة Ad-Asst رقم القسم 90.

وكل صف توجد به قيمة مختليفة عن الصف الأخر.

وهذا أيضآ يدل على أن Union تمنع التكرار.

ملحوظة:-

* الناتج يكون مرتب تصاعدى بالـ Column الأول.

appears twice as the JOB_ID is different in each row.

Consider the following example:

```
SELECT  employee_id, job_id, department_id
FROM    employees
UNION
SELECT  employee_id, job_id, department_id
FROM    job_history;
```

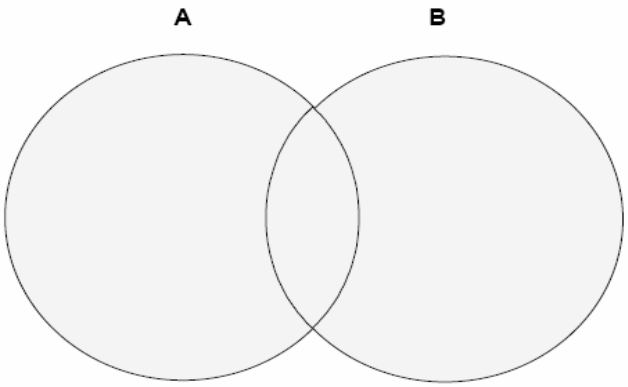
EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
100	AD_PRES	90
101	AC_ACCOUNT	110
101	AC_MGR	110
101	AD_VP	90
200	AC_ACCOUNT	90
200	AD_ASST	10
200	AD_ASST	90
201	MK_MAN	20

29 rows selected.

Union All

Display the current and
Previous departments of all employees.

The UNION ALL Operator



The UNION ALL operator returns results from both queries including all duplications.

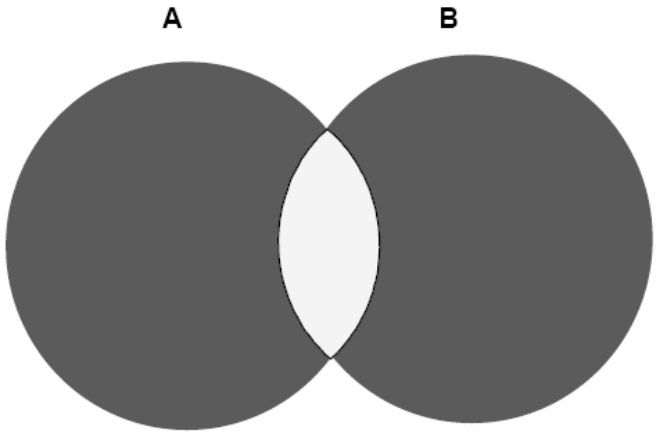
EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
100	AD_PRES	90
174	SA_REP	80
176	SA_REP	80
176	SA_MAN	80
176	SA_REP	80
205	AC_MGR	110
206	AC_ACCOUNT	110

Union all :- سنعرض جميع الصفوف حتى المكرر أو المتشابة منها.

*هنا يظهر الموظف رقم 176 ثلاث مرات و يظهر فى الصف الاول الوظيفة Sa-Rep و القسم 80.وظهر ايضا فى الصف الثالث الوظيفة Sa-Rep ولقسم 80.

The Intersect Operator

The INTERSECT Operator



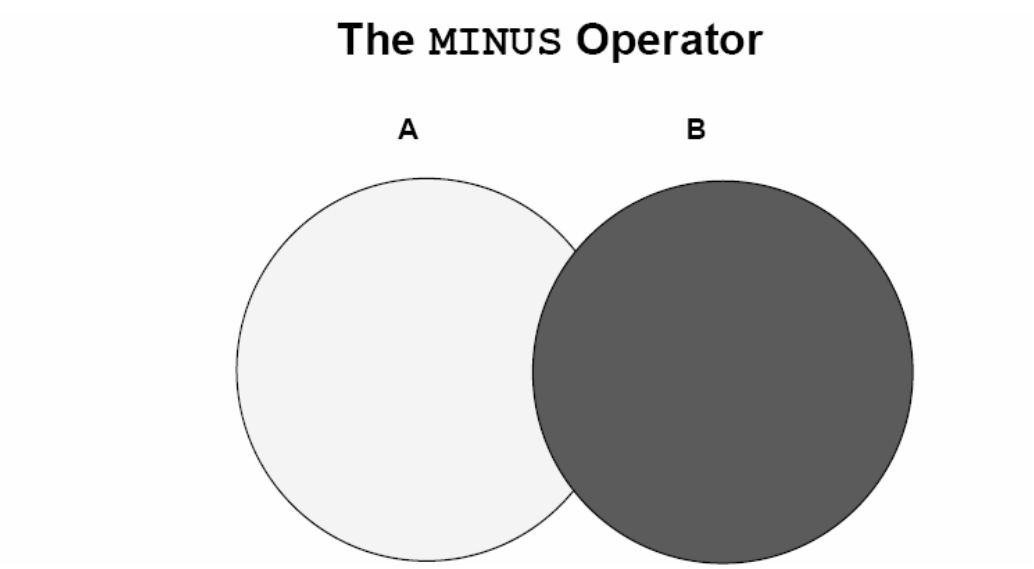
Intersect:- سنعرض لك الصفوف الموجود فى الـ query الاولى وتوجد أيضا فى الـ query الثانية فى وجود فى هذا الاستعلامات يعرض مرة واحدة وغير مكرر.

```
SELECT employee_id, job_id
FROM   employees
INTERSECT
SELECT employee_id, job_id
FROM   job_history;
```

EMPLOYEE_ID	JOB_ID
176	SA_REP
200	AD_ASST

*الناتج سيكون هو الموظف رقم 176 و وظيفتة Sa-rep و وظيفتة رقم 200 AD-Asstلانهم موجودين فى الـ query الاولى و الثانية.

The Minus Operator



Minus:- تعرض الصفوف الموجودة فى الـ query الاولى و غير موجودة فى الـ query الثانية.

```
SELECT employee_id
FROM   employees
MINUS
SELECT employee_id
FROM   job_history;
```

EMPLOYEE_ID	JOB_ID
100	AD_PRES
101	AD_VP
202	MK_REP
205	AC_MGR
206	AC_ACCOUNT

18 rows selected.

نرى أن الناتج هى مجموعة الصفوف الموجودة فى الـ Query الاولى و غير موجود فى الـ Query الثانى.
يمكن أستخدام الـ set operation من خلال الـ sub query .

ıtor Guidelines

expressions in the select lists of the queries must match in number and
ON, UNION ALL, INTERSECT, and MINUS SET operators in their W
ame number and type of columns in their SELECT list. For example:

```
SELECT employee_id, department_id
FROM   employees
WHERE  (employee_id, department_id)
       IN (SELECT employee_id, department_id
           FROM   employees
           UNION
           SELECT  employee_id, department_id
           FROM   job_history);
```

كيفية عمل دمج لاكثر من query من خلال الـ sub query.

```
SELECT employee_id, job_id,salary
FROM   employees
UNION
SELECT employee_id, job_id,0
FROM   job_history;
```

EMPLOYEE_ID	JOB_ID	SALARY
100	AD_PRES	24000
101	AC_ACCOUNT	0
101	AC_MGR	0
205	AC_MGR	12008
206	AC_ACCOUNT	8300

30 rows selected.

وهنا تم وضع صفر فى ال query الثانية لانة لايوجد salary فى جدول الـ job history ويجب اعادة نفس عدد الـ column فى الـ query الاولى و الثانية .

الفصل الثامن

Manipulating Data

الأهداف

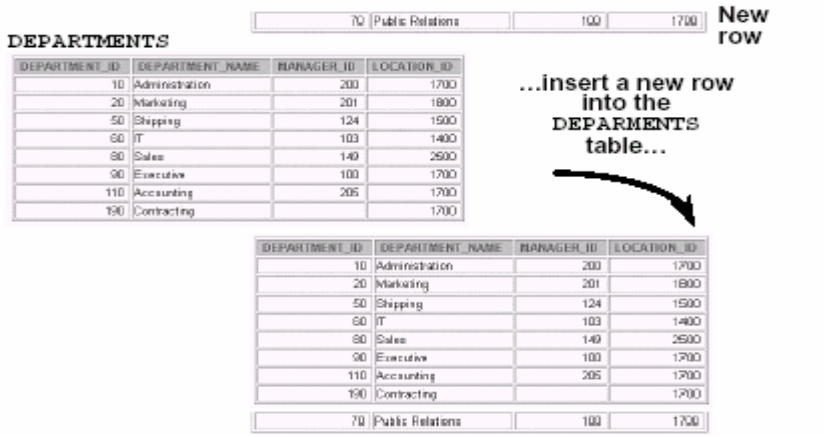
- بعد إكمال هذا الدرس أنت يجب إن شاء الله أن تكون قادر علي عمل التالي:
  - وصف كل تعبيرات DML (Data Manipulating Language) .
  - إدخال صفوف جديدة للجدول.
  - عمل تحديث وتغير لبيانات الجدول.
  - حذف صفوف معينة من الجدول .
  - دمج صفوف من جدولين في جدول واحد.
  - امكانية التعامل مع كلا من (Rollback, Commit and Savepoint).

تعبير DML ينفذ عند :

- اضافة صفوف جديدة للجدول.
- تعديل صفوف موجودة في الجدول.
- حذف صفوف موجودة فى الجدول.

Adding a New Row to a Table

إضافة صف جديد للجدول



الصف الجديد المراد أضافته رقم قسمة 70 و أسم القسم هو علاقات عامة ورقم المدير هو 100 ورقم الموقع 170

و يوضح لنا الرسم الفرق بين الجدول قبل إدخال الصف وبعد إدخال الصف. يمكنك إضافة صف جديد للجدول باستخدام امر(INSERT) والصيغة العامة لتلك الامر كما يلى:

```
INSERT INTO      table [(column1 [, column2...])]
VALUES           (value1 [, value2...]);
```

حيث أن

`Table`	اسم الجدول المراد إلحاق السجل به
`Column1,Column2`	اسماء الأعمدة(الحقول) المطلوب إدخال البيانات إليها

Value1, value2	القيم المطلوب إضافتها في حقول السجل الجديد وكل قيمة يتم إدراجها في الحقل المناسب في القائمة وبالتالي سيتم إضلفة القيمة1 في الحقل 1 وهكذا

الشرح

بعد تنفيذ جملة الإدخال بهذه الطريقة سوف يتم إضافة بيانات سجل واحد فقط في الجدول المذكور يتوي على قيم تم إ دراجها في جملة الإدخال وللحقول المذكورة فقط

القواعد التي يجب التقيد بها في هذه الطريقة

1) يجب أن يكون عدد القيم التي سيتم إدخالها هو نفس عدد الحقول المذكور في جملة **INSERT**

2) يجب أن يكون نوع بيانات القيم التي سيتم إدخالها من نفس نوع بيانات الحقول وأن تكون هذه القيم مرتبة حسب ترتيب الحقول في جملة **INSERT**

مثال

```
INSERT INTO departments(department_id, department_name,
                        manager_id, location_id)
VALUES      (70, 'Public Relations', 100, 1700);
1 row created.
```

3) عند إدخال حقول التاريخ والنصوص يجب وضع القيم المخلة بين علامتي تنصيص مفردتين

4) يجب مراعاة وجوب إدخال قيم في الحقول الإجبارية التي تم تعريفها على أنها لا تحتوي فراغ **NOT NULL** ويتم ادخال NULL في الحقول التي بها فراغ

مثال

```
INSERT INTO       dept
VALUES      (13, 'Administration', NULL);
```

5) يجوز عدم ذكر أسماء الحقول في جملة **INSERT** في حالة إدخال بيانات جميع الحقول لهذا السجل على أن تكون القيم المخلة مرتبة حسب الترتيب الافتراضي للحقول في الجدول عند بنائه

ملاحظة
لمعرفة الترتيب الافتراضي للحقول في الجدول و لمعرفة نوع البيانات لكل عمودعندما تريداضافة بيانات جديدة ذلك بستخدم الأمر **DESC**

نستخدام الامر DESCRIBE او DESC كما فى المثال التالى:

```
DESCRIBE  departments
```

Name	Null?	Type
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

إضافة صفوف تحتوي علي قيم فارغة او NULL .

من الممكن إضافة صفوف تحتوي علي قيم فارغة ونقوم بعمل ذلك **بطريقتين:-**

- الطريقة الاولى : حذف العمود من قائمة الأعمدة.

```
INSERT INTO departments    (department_id,
                           department_name)
VALUES          (30, 'Purchasing');
1 row created.
```

الطريقة الأولى وهي إغفال ذكر أسم العمود المراد جعل قيمته فارغة (NULL). وتسمي هذه الطريقة بالطريقة الضمنية.

- **الطريقة الثانية:** ادراج العمود و تحديد كلمة NULL في فقرة VALUES

```
Insert into dept (deptno,dname,loc)
Values (50,'EDUCATION',NULL);
```

وهذه هي الطريقة الثانية وهى وضع كلمة NULL و تسمي بالطريقة الواضحة أو الصريحة .

الطرق لإدخال قيم فارغة

Implicit	حذف العمود من قائمة الاعمدة المراد ادخال بيانات بها
Explicit	تحديد NULL للعمود او تحديد مجموعة فارغة (' ') في قائمة VALUES

* أخطاء شائعة تحدث عند استخدام INSERT :-

- عدم ادخال قيمة فى عمود لابد من ادخال قيمة بة مثل عمود (Primary Key).
- ادخال قيمة سبق ادخالها فى عمود (Primary Key) .
- ادخال قيم عدد حروفها اكبر من المتاح لتلك العمود.

```
Insert into EMP (ename, job)
Values ('ZAKI','MANAGER')    ;
```

```
ERROR at line 1:
ORA-01400: cannot insert NULL into ("SCOTT"."EMP"."EMPNO")
```

وسبب هذا Error :

ان عمود EMPNO يعتبر Primary Key ولا يمكن ان يترك بدون قيمة.

استخدام SYSDATE و USER فى جملة Insert

عند إضافة صفوف جديدة يمكن استخدام SYSDATE للتعبير عن التاريخ الحالى بدون كتابتة يدويا. ويمكن استخدام كلمة User كقيمة لاضافة اسم المستخدم الحالى كما فى **المثال** التالى:

```
Insert into login values (user, sysdate);
```

فلنفترض وجود جدول يحتوى على عمودين الاول لاسم المستخدم والاخر لتسجيل الوقت الحالى.

مثال:

```
Insert into employees
 (employee_id, last_name, job_id, hire_date, commission_pct)
Values (113,user,'ac_account',sysdate,null);
```

```
SELECT employee_id, last_name, job_id, hire_date, commission_pct
FROM   employees
WHERE  employee_id = 113;
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	HIRE_DATE	COMMISSION_PCT
113	Popp	AC_ACCOUNT	27-SEP-01	

هنا نجد أن SYSDATE أدت إلي إدخال تاريخ النظام وقت تنفيذ الأمر وأدت كلمة user إلي إدخال أسم المستخدم الحالي وهو Popp. وادى استخدام كلمة NULL الى ادراج NULL بداخل العمود. إدخال قيم التاريخ

```
INSERT INTO employees
VALUES      (114,
             'Den', 'Raphealy',
             'DRAPHEAL', '515.127.4561',
             TO_DATE('FEB 3, 1999', 'MON DD, YYYY'),
             'AC_ACCOUNT', 11000, NULL, 100, 30);
1 row created.
```

- **للتحقق من الإضافة**

```
Select * from employees
Where employee_id=114;
```

عند إدخال تاريخ يجب مراعاة كتابة بطريقة مبدئية وهي اليوم مكون من رقمين والشهر مكون من ثلاثة حروف ثم السنة مكونة من أربعة أرقام.
(DD-MON-YY)وعند إدخال التاريخ بطرقة مختلفة نستخدم الدالة TO_DATE
وذلك لتحويل التاريخ إلي الطريقة التي يقبلها الأوراكل والناتج يوضح شكل التاريخ بعد إدخالها كما هو موضح أعلي.
حيث يكون الناتج كالتالي 03-FEB-99

اضافة عدة سجلات إلى جدول البيانات

من خلال هذه الطريقة يمكن ادخال أكثر من سجل واحد إلى جدول البيانات عن طريق استخدام متغيرات الادخال وهي عبارة عن متغيرات توضع في جملة الادخال بدلا من القيم نفسها ةيمكن أن نطلق على هذه الطريقة جملة الادخال متعددة السجلات

```
INSERT INTO table [(column1 [, column2...])]
VALUES (&variable1 [,&variable1 ...]);
```

حيث ان

Table	اسم الجدول المراد إلحاق السجل به
Column1,Column2	اسماء الأعمدة(الحقول) المطلوب إدخال البيانات إليها
Variable1, Variable 2	متغيرات الإدخال التي سوف يتم استبدالها بقيم حقيقية بعد تنقيذ جملة الادخال

الشرح

بعد تنفيذ جملة يهذه الطريقة سوف يطلب منك إدخال القيم للمتغيرات المذكورة في جملة الإدخال وبعد الانتهاء من ادخال القيمة تضغط مفتاح ENTER من لوحة المفاتيح وهكذا حتى تنتهي من إدخال حقول لاسجل الأول ولإدحال سجل آخر يمكنك الضغط على حرف (R) ثم مفتاح ENTER من لوحة المفاتيح وهو يعني تكرار الإدخال لسجلات اخرى

مثال

```
INSERT INTO  s_dept (id, name, region_id)
vALUES(&department_id,'&department_name,&region_id);
Enter value for department_id: 61
Enter value for department_name: Accounting
Enter value for region_id: 2
```

القواعد التي يجب التقيد بها في هذه الطريقة

تنطبق على هذه الطريقة جميع القواعد المذكورة في الطريقة الأولى ويضاف إليها مايلي :

1- تستبدل القيم في جملة الإدخال بمتغيرات
2- يعود اختيار أسماء المتغيرات إلى المستخدم مع مراعاة شروط تسمية المتغيرات
3- يجب أن توضع علامة & قبل متغير الادخال
4- في جملة الإدخال يمكن وضع علامتى تنصيص مفردتين حول متغير الادخال الخاص بالحقول النصية

Coping a Rows from Another Table
امكانية استعمال SubQuery مع جملة INSERT

من خلال هذه الطريقة يمكن إدخال أكثرمن يجل واحد إلى جدول البيانات عن طريق نسخ يث نستخدم SELECTهذا السجل / السجلات من جدول آخر بوتسطة جملة الاستفسار مع استعلام فرعي انظر الى المثال التاليINSERTجملة

```
INSERT INTO history(id, last_name, salary, title,
start_date)
SELECT id, last_name, salary,title, start_date
FROM s_emp
WHERE start_date < '01-JAN-94';
```

الشرح

بعد تنفيذ جملة الإدخال بهذه الطريقة ينم نسخ السجلات التي تحقق الشرط من الجدولا لمصدر إلى الجدول الهدف وللحقول االمذكورة في جملة الإدخال

القواعد التي يجب التقيد بها في هذه الطريقة

- كتابة جملة الإدخال INSERT محتوية على جملة استفسار SELECT
- تستبدل القيم في جملة الإدخال بأسماء حقول الجدول المصدر
- عدم استخدم العبارة VALUES
- يجب مراعاة وجوب إدخال قيم في القول الاجبارية التي تم نعريفها على أنها لا تحتوي فراغ NOT NULL
- مطابقة الحقول بين الجدولين من حيث ترتيب الحقول ونوع البيانات وعدد الحقول

- البيانات.
- كي ننشئ نسخة من الصفوف في الجدول استخدم *SELECT في الاستعلام الفرعي.

Changing Data in a Table

تغيير البيانات في الجدول

الموظفين

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMMISSION_P
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	90	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	90	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	90	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	60	
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	60	
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	4200	60	
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50	

تعديل الصفوف في جدول الموظفين.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMMISSIO
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	90	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	90	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	90	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	30	
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	30	
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	4200	30	
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50	

المثال المعروض أمامنا يقوم بتغيير رقم قسم الموظفين بقسم 60 إلي القسم رقم 30.

الصيغة الرئيسية لتعبير UPDATE تحديث أكثر من صف واحد في نفس الوقت ،إذا تطلب ذلك يمكن أن تعدل صفوف موجودة باستخدام تعبير UPDATE.على النحو التالى:

Table أسم الجدول

Column أسم العمود المراد تغيرة.

Value القيمة الجديدة.

Condition تحديد الصفوف التى ينطبق عليها الشرط المطلوب.

وباغفال الشرط يحدث تغيير بكل الاعمدة.

ملحوظة :

بشكل عام يمكن استخدام المفتاح الرئيسي (Primary Key) لتحديد صف وحيد. حيث استخدام الأعمدة الأخرى يمكن أن يؤدى الى نتائج غير متوقعة والسبب أن صفوف عديدة ينطبق عليها نفس الشرط.

كمثال: تحديد صف واحد من جدول الموظفين عن طريق استخدام عمود الاسماء كشرط لاجراء التغير المطلوب وهذا قد يؤدى الى حدوث خطأ لأنة من الممكن ان يكون أكثر من موظف بنفس الاسم. وفى تلك الحالة من الافضل استخدام عمود Primary Key لكى نتمكن من اجراء التغير بدقة حيث ان البيانات بتلك العمود تكون غير متكررة.

تعديل بيانات في الجدول

تحديد صف أو مجموعة صفوف فى فقرة WHERE .

```
UPDATE employees
SET    department_id = 70
WHERE  employee_id = 113;
1 row updated.
```

يمكن لصفوف الجدول كلها ان تعدل عن طريق حذف فقرة WHERE.

```
UPDATE   copy_emp
SET      department_id = 110;
22 rows updated.
```

عندما نريد تعديل بيانات نستخدم الأمر UPDATE ويكتب علي النحو التالي:

الأمر UPDATE يليه أسم الجدول مثلا Employees المراد تعديل بياناته ثم كلمة SET يليها أسم العمود المراد تغيير قيمته ثم علامة يساوي ثم القيمة الجديدة. و يلي ذلك جملة الشرط (Condition) التي سوف تحدد الصفوف المراد تعديلها في الجدول.

ملاحظة: اذا لم تقم بتحديد الصفوف المراد تغيرها في جملة الشرط (Condition) فإن التعديل سيكون علي جميع صفوف الجدول.

كمثال نريد نقل الموظف رقم 7788 (SCOTT) للقسم رقم 30 .

Update EMP set deptno=30

Where empno=7788;

قبل اجراء التعديل كان رقم القسم 20 للموظف SCOTT .

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20

بعد اجراء التعديل اصبح رقم القسم 30 للموظف SCOTT .

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7788	SCOTT	ANALYST	7566	19-APR-87	3000		30

*إذا حذفت فقرة WHERE فأن كل صفوف الجدول سوف تعدل ويصبح الموظفين كلهم فى الادارة رقم 30.

Update Tow Column With SubQuery تعديل البيانات باستخدام Subquery

المثال التالى يريد تعديل الموظف رقم 114 بوظيفة ومرتب الموظف رقم 205.

```
UPDATE    employees
SET       job_id   = (SELECT   job_id
                      FROM     employees
                      WHERE    employee_id = 205),
          salary   = (SELECT   salary
                      FROM     employees
                      WHERE    employee_id = 205)
WHERE     employee_id     =   114;
1 row updated.
```

هنا نجد أمكانية تعديل البيانات باستخدام اكثر من استعلام فرعي كما فى المثال السابق.

ملحوظة :

إذ لم ينطبق الشرط بمعنى عدم وجود اى موظف بالرقم 114 تظهر لنا تلك الرسالة 0 Rows Updated بمعنى ان الشرط لم يتحقق وبالتالى لم يقم باجراء اى تغير بالجدول.

*تعديل صفوف معتمدا علي جدول آخر

```
UPDATE    copy_emp
SET       department_id    =    (SELECT department_id
                                 FROM employees
                                 WHERE employee_id = 100)
WHERE     job_id             =    (SELECT job_id
                                 FROM employees
                                 WHERE employee_id = 200);
1 row updated.
```

يمكن أن تستخدم الاستعلام الفرعي كي تعدل صفوف بجدول عن طريق استخدام بيانات من جدول اخر كما فى المثال السابق .

المثال المعروض لتعديلات بجدول COPY_EMP معتمد علي قيم من جدول الموظفين.

Updating Rows:
Integrity Constraint Error

```
UPDATE employees
SET    department_id = 55
WHERE  department_id = 110;
```

Department number 55 does not exist

```
UPDATE employees
       *
ERROR at line 1:
ORA-02291: integrity constraint (HR.EMP_DEPT_FK)
violated - parent key not found
```

عند محاولة تعديل المفتاح الأجنبي (Foreign Key) برقم غير موجود فى عمود (Primary Key) المرتبط بة بالجدول الاخر فينتج عن ذلك Error.

ففى **المثال** السابق حاول تغير رقم القسم 110 بالرقم 55 مع العلم ان الرقم 55 غير موجود بعمود الاقسام (Primary Key) بجدول الاقسام ولهذا حدث Error .

وهذا يعني أن رقم الإدارة الجديد لابد أن يكون موجود فى عمود (Primary Key) بالجدول الرئيسى اولا.

Removing a Row from Table
حذف الصف من الجدول

جدول الأقسام

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing		
100	Finance		
50	Shipping	124	1500
60	IT	103	1400

تم حذف صف من جدول الأقسام

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing		
50	Shipping	124	1500
60	IT	103	1400

لحذف صف أو مجموعة من الصفوف لجدول ما نستخدم الأمر Delete
ويتم كتابتة كما يلي:

1-Delete ثم 2- From 3- Table

الجدول الذي يتم الحذف منه

ثم جملة الشرط WHERE التي ستحدد الصف المراد حذفه. ويجب ملاحظة أنه في حالة إغفال جملة الشرط فإنه يقوم بحذف كل صفوف الجدول.
والمثال السابق يحذف قسم المالية من جدول الأقسام.

ملحوظة :
إذا لم يكن هناك صف ينطبق علية الشرط فتظهر تلك الرسالة 0 Rows Deleted وتعنى عدم القيام بحذف اى صف.
حذف صفوف معتمدة علي جدول آخر ذلك عن طريق استخدم الاستعلام الفرعي (SubQuery) في فقرة Where .

```
DELETE FROM employees
WHERE  department_id =
                       (SELECT department_id
                        FROM   departments
                        WHERE  department_name LIKE '%Public%');

1 row deleted.
```

يمكنك استخدام الاستعلام الفرعي كي تحذف صفوف من جدول معتمدا علي قيم من جدول أخر فى جملة الشرط.
والمثال السابق يقوم بحذف كل الموظفين بالقسم الذي يحتوي اسمة علي مجموعة الحروف " Public" ، والاستعلام الفرعي يبحث فى جدول الأقسام عن رقم القسم الذي يحتوي اسمة علي مجموعة الحروف " Public" .

Deleting Rows:
Integrity Constraint Error

```
DELETE FROM departments
WHERE       department_id = 60;
```

You cannot delete a row that contains a primary key that is used as a foreign key in another table.

```
DELETE FROM departments
            *
ERROR at line 1:
ORA-02292: integrity constraint (HR.EMP_DEPT_FK)
violated - child record found
```

***ملاحظة :**

عند اجراء عملية حذف فى جدول فلا يمكن حذف عمود يتم الاعتماد علية فى جدول اخر اى ان تلك العمود Primary Key و لعلاج ذلك لابد من حذف كل الصفوف الموجودة فى الجدول الاخر بعمود (Foreign Key) اولا .

*يمكنك الاستغناء عن كلمة From كما فى المثال التالى:

```
Delete employees
Where employees_id=7788;
```

استخدام الاستعلام الفرعي في تعبير INSERT

```
INSERT INTO
        (SELECT employee_id, last_name,
                email, hire_date, job_id, salary,
                department_id
         FROM   employees
         WHERE  department_id = 50)
VALUES (99999, 'Taylor', 'DTAYLOR',
        TO_DATE('07-JUN-99', 'DD-MON-RR'),
        'ST_CLERK', 5000, 50);

1 row created.
```

يمكن أن تستخدم الاستعلام الفرعي (Subquery) بدلا من اسم الجدول في فقرة INTO فى فقرة INSERT .

لاحظ ان قائمة الاعمدة فى الاستعلام الفرعي (SubQuery) يجب أن يكون بنفس الترتيب للقيم فى فقرة VALUES.

Using With Check Option on DML Statement

تستخدم فقرة With Check Option عندما تريد تحديد بعض الشروط عند اضافة صفوف جديدة .

```
INSERT INTO  (SELECT employee_id, last_name, email,
                     hire_date, job_id, salary
              FROM   employees
              WHERE  department_id = 50 WITH CHECK OPTION)
VALUES (99998, 'Smith', 'JSMITH',
        TO_DATE('07-JUN-99', 'DD-MON-RR'),
        'ST_CLERK', 5000);
INSERT INTO
          *
ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause violation
```

فقرة WITH CHECK OPTION تمنعك من اضافة صفوف ليست في الاستعلام الفرعي.

فالمثال السابق كان سبب Error انة لم يقم باضافة رقم القسم المحدد فى جملة الشرط وهو (50) فى عمود رقم الاقسام بالنسبة للصف الجديد.

ولمعالجة ذلك نقم باضافة القسم 50 للموظف الجديد حيث انة تم تحديدة فى جملة Where .اى اننا لايمكنا من اضافة موظفين الا فى القسم رقم 50.

```
Insert into (select employee_id, last_name, email,
hire_date,job_id,salary,
department_id from employees
where department_id=50 with check option)
values (9999, 'SMITH', 'JSMITH',
 to_date ('12-oct-81','dd-mon-rr'), 'ST_CLERK' ,50000 ,50);
1 row created
```

اما اذا قمنا بتغير رقم القسم بحيث يكون مختلف عن رقم القسم المحدد فى جملة WHERE ففى تلك الحالة لايسمح بذلك كما فى المثال التالى ولاحظ رقم القسم الجديد:

```
Insert into (select employee_id, last_name, email,
hire_date,job_id,salary,
department_id from employees
where department_id=50 with check option)
values (9999, 'SMITH', 'JSMITH',
 to_date ('12-oct-81','dd-mon-rr'), 'ST_CLERK' , 50000 ,30);
```

```
Insert into (select employee_id, last_name, email,
*

ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause violation
```

استخدام القيم المحددة سابقا (DEFAULT)

- DEFAULT with INSERT:

```
INSERT INTO departments
  (department_id, department_name, manager_id)
VALUES (300, 'Engineering', DEFAULT);
```

- DEFAULT with UPDATE:

```
UPDATE departments
SET manager_id = DEFAULT WHERE department_id = 10;
```

Default هو قيمة افتراضية تم وضعها لعمود معين وذلك عند انشاء الجدول. بحيث اذا لم يتم ادخال بيانات محددة فى تلك العمود يمكنك استخدام
Default المحدد مسبقا كما فى المثال السابق.
وهذا سوف يناقش إن شاء الله تعالي في الفصل التالى.

The MERGE Statement
(الدمج) MERGE

1- امكانية التعديل فى بيانات بشكل شرطي أوإدخال بيانات في جداول قواعد البيانات.
2- زيادة الأداء والسهولة في الاستخدام و مفيد في تخزين البيانات.
3- أمر MERGE يجمع أوامر INSERT, UPDATE حيث أنك تحتاج كلاهما.
4- أمر MERGE يقوم بدمج جدولين معا ولكى يتم ذلك فلابد من التاكد ان كلا من الجدولين يتضمن نفس عدد الاعمدة وبنفس نوع البيانات.

الصيغة الرئيسية لجملة MERGE

```
MERGE INTO table_name table_alias
  USING (table|view|sub_query) alias
  ON (join condition)
  WHEN MATCHED THEN
    UPDATE SET
    col1 = col_val1,
    col2 = col2_val
  WHEN NOT MATCHED THEN
    INSERT (column_list)
    VALUES (column_values);
```

يمكنك تعديل الصفوف الموجودة و إدخال صفوف شرطية جديدة باستخدام تعبير MERGE ويعنى ادمج.

فقرة INTO تحديد الجدول المستهدف الذي يتم تعديلة او أدخال بيانات عليه.
فقرة USING يعنى مصدر البيانات كي تعدل أو تدخل.
فقرة ON الشرط الذى يعتمد علية فى MERGE .
WHEN MATCHED و يعنى ماذا يفعل عند تحقق الشرط .
WHEN NOT MATCHED و يعنى ماذا يفعل عند عدم تحقق الشرط .

مثال علي دمج صفوف كلا من جدول Employees وجدول Copy_emp

```
MERGE INTO copy_emp c
  USING employees e
  ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
  UPDATE SET
    c.first_name      = e.first_name,
    c.last_name       = e.last_name,
    c.email           = e.email,
    c.phone_number    = e.phone_number,
    c.hire_date       = e.hire_date,
    c.job_id          = e.job_id,
    c.salary          = e.salary,
    c.commission_pct  = e.commission_pct,
    c.manager_id      = e.manager_id,
    c.department_id   = e.department_id
WHEN NOT MATCHED THEN
  INSERT VALUES(e.employee_id, e.first_name, e.last_name,
       e.email, e.phone_number, e.hire_date, e.job_id,
       e.salary, e.commission_pct, e.manager_id,
       e.department_id);
```

الشرط الموجود فى **المثال** السابق هو تساوى رقم الموظف فى كلا من الجدولين وعند تحقق تلك الشرط يقوم بتعديل البيانات فى جدول Copy_emp بالبيانات الموجودة فى جدول Employees .

وعند عدم تحقق تلك الشرط فهذا يعنى ان الموظف غير موجود فى جدول Copy_emp ولهذا سوف نقوم باضافة تلك الموظف الى جدول Copy_emp .

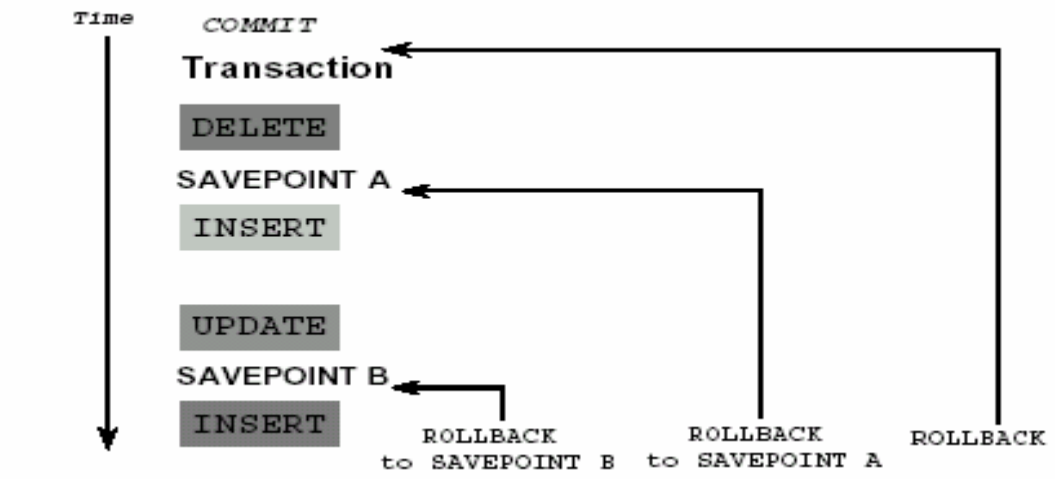
والهدف من MERGE ان يكون بيانات الجدولين متماثلة.

(Database Transactions)

تحتوي علي واحد من التالي:-

- تعبير DML (Data Manipulating Language)وتحتوى على:
- (INSERT,UPDATE,DELETE,MERGE)
- تعبير DDL (Data Definition Language) وتحتوى على:
- (CREATE,MODIFY,DROP,RNAME,TRUNCATE)
- تعبير DCL (Data Control Language)و تحتوى على: (GRANT,REVOKE)

معاملات التحكم



يمكنك التحكم في المعاملات باستخدام أوامر COMMIT,SAVEPOINT,ROLLBACK.

الأوامر	الوصف
COMMIT	يقوم بحفظ اى تعديلات على قاعدة البيانات من Insert,Update,Delete,MERGE.,
SAVEPOINT name	مؤشرات لحفظ البيانات الى حد او مكان معين
ROLLBACK	يقوم بالغاء كل التغيرات التى تمت
ROLLBACK TO SAVEPOINT name	يقوم بالرجوع حتى نقطة معينة محددة فى فقرة (savepoint). حتى يتراجع عن جزئية معينة من العمليات .

*يستخدم الامر Commit لحفظ اى تعديلات فى قاعدة البيانات فأذا قمت بعمل DML واردت حفظ تلك التغيرات فلابد من استخدام الامر Commit.

مثال:

```
Update emp set ename ='ASAS'
Where empno=7782;
```

ثم بعد ذلك

```
Commit;
```

فهذا يعنى انك تريد حفظ تلك التغير.

*يستخدم الامر Savepoint لامكانية انشاء علامة تذكرك بأخر التعديلات التى قمت بها و يمكنك الرجوع اليها .

مثال:

```
Insert into emp (empno,ename)
Values(10,'MOMO');
```

ثم اردت حفظ هذا التعديل بعلامة لكى يمكنك بعد ذلك الرجوع لها.

```
Savepoint a;
```

ثم قمت بعد ذلك بتعديل مرتب موظف معين مثلا

```
Update emp set sal=2500
Where empno=4444;
```

واردت بعد ذلك الغاء تلك التغير الاخير فى المرتب .بدون الغاء جملة Insert الاولى. وذلك عن طريق استخدام Rollback.

*يستخدم الامر Rollback لامكانية الرجوع عن كل التعديلات التى تمت او لتحديد المكان الذى ترغب فى الرجوع الية.

مثال:

اذا اردنا الرجوع الى اخر تعديل حدث فنقوم بالاتى:

```
Rollback to a;
```

ملخص الفصل

تناولنا من خلال هذا الفصل موضوع اضافة البيانات إلى الجداول بواسطة جملة INSERT وهنالك ثلاثة طرق لإجراء عملية الإضافة

- **إضافة سجل واحد إلى جدو ل البيانات**

بهذه الضريفة سوف يتم إضافة بيانات سجل واحد فقط إلى جدول يحتوي على القيم التي ستذكر في جملة الإدخال

- **إضافة عدة سجلات إلى جدول البيانات**

بهذه الطريقة يمكن إدخال أكثر من سجل واحد إلى جدول البيانات عن طريق استخدام متغيرات الادخال وهي عبارة عن متغيرات توضع في جملة الإدخال بدلا من القيم نفسها ويمكن أن نطلق على هذه الطريقة جملة الادخال متعددة السجلات

- **إضافة سجلات في جدول عن طريق نسخها من جدول آخر**

بهذه الطريقة يمكن إدخال أكثر من سجل واحد إلى جدول البيانات عن طريق نسخ هذا السجل / السجلات من جدول آخر بواسطة جملة SELECT الموجودة بداخل جملة INSERT

- **تعديل بيانات حقل/ حقول لسج واحد أو اكثر**

حيث يتم الغاء القيم الموجودة بها لتحتوي على القيم الجديدة المذكورة في جملة التعديل علما بأن السجلات التي سيتتم التعديل بها هي السجلان التي تحقق الشرط.

- **نعديل بيانات حفل / حقول لجميع السجلات في الجدول**

حيث يتم الغاء القيم الموجودة بها لتحتوي على القيم الجديدة المذكورة في جملة التعديل علما انه سيتم تعديل بيانات جميع الحقول لعدم وجود شرط

ثم تناولنا موضوع حذف السجلات عن DELETE وهنالك طريقتان لإجراء عمليةا الحذف

- **حذف سجل أو أكثر من الجدول**

بهذه الطريقة سوف يتم حذف السجل / السجلات التي تحقق الشرط في جملة الشرط

- **ثانيا : حذف جميع سجلات الجدول**

بهذه الطريقة سوف يتم حذف جميع السجلات في الجدول

نظرة عامة على الكورس

مقدمه :

- **SQL / PL هو أمتداد للغة SQL بأضافة مميزات لغات البرمجة الاجرائية .**
- **وجمل التعامل والأستعلام الخاصة بلغة SQL ويتم أدراجها داخل الكود الخاص بلغة SQL / PL .**
- **بيئة لغة SQL / PL فصل جمل SQL وتنفيذ جمل SQL / PL كلا على حدى لعمل مخرجات البرنامج .**

- **مزايا لغة SQL / PL :**

أ- التكامل : وهذه اللغة تلعب دور أساسى بين اجزاء وأدوات أوراكل حيث يكتب بها اكواد (FORMS) ويتم بها برمجة أجزاء وأدوات الاوراكل .

ب- تحسين الأداء : حيث يمكن SQL / PL تحسين اداء التطبيقات وذلك من خلال :

- **تجميع SQL معا فى بلوك واحد (كتلة واحدة) وأرسلهما الى خادم (Data base) لتنفيذها دفعة واحدة وذلك يؤدى الىا أرتفاع مستوى الاداء عامة .**
- **يمكن لـ SQL / PL العمل داخل اى جزء من أجزاء وأدوات أوراكل وذلك يضيف قوة المعالجة الاجرائية الى هذه الادوات مثل oracle reports ، Oracle forms ، ..... مما يؤدى الى تحسن مستوى الأداء .**

جـ- تطوير البرنامج Modularized :

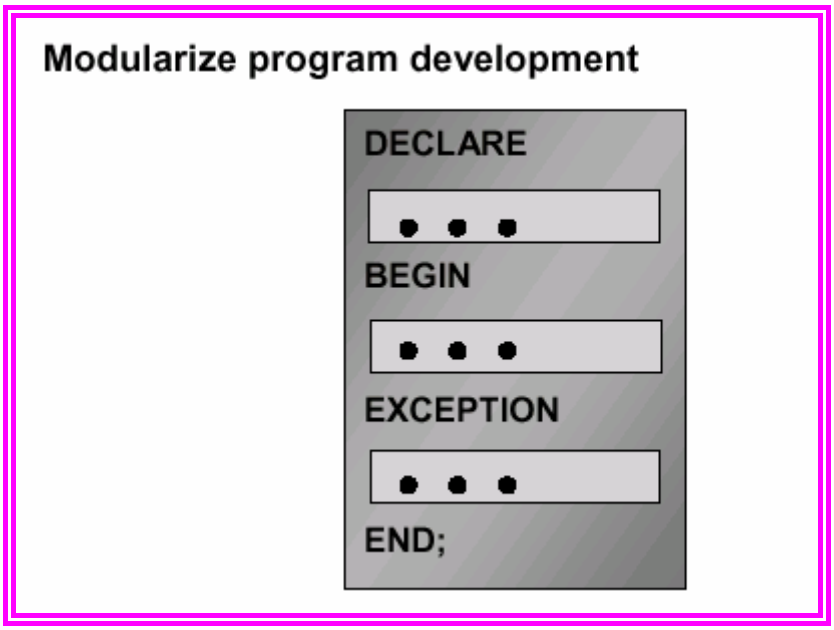
- **وذلك بتجميع منطقى للبيانات داخل كتل (Blocks) البرنامج .**
- **الكتل المتداخلة (Nested blocks) تتيح العديد من المزايا .**
- **اتاحة تقسيم المشاكل المعقدة الى مجموعة أبسط من المشاكل يمكن حلها ببساطة .**
- **الأستفادة من خبرات وأكواد سابقة بجمعها فى شكل مكتبات (Libraries) يمكن أن الاستفادة منها بين أدوات أوراكل المختلفة .**

د - يمكن تنفيذ الكود SQL / PL من اى أدوات من أدوات اوراكل

هـ- يمكن تعريف المتغيرات التى تستقبل العديد من أنواع البيانات المختلفة مثل النصوص والأرقام والصور والفيديو والبيانات المركبة ..... الخ .

و - وتحتوى أيضا على المميزات لأى لغة أجرائية من حيث تواجد أوامر loop والتحكم فى سير البرنامج ومعالجة الاخطاء والأستثناءات .

- **شكل كتلة SQL / PL :**



يتكون بلوك او كتلة PL / SQL من:

- جزء تعريفى للمتغيرات للتعامل بها وهو جزء أختيارى ليس اساسى يبدأ بـ Declare

- جزء أخر أساسى ويحتوى على جمل البرمجة يبدأ من (Begin) وينتهى بـ (End;)

- جزء أخر أختيارى هو (Exception) يحتوى على جمل المعالجة والتعامل مع الأخطاء والأستثناءات المختلفة Exception .

- **المثال التالى يوضح شكل برنامج اولى للـ PL/SQL:**

وهو عند تنفيذه يقوم باظهار رسالة على الشاشة:

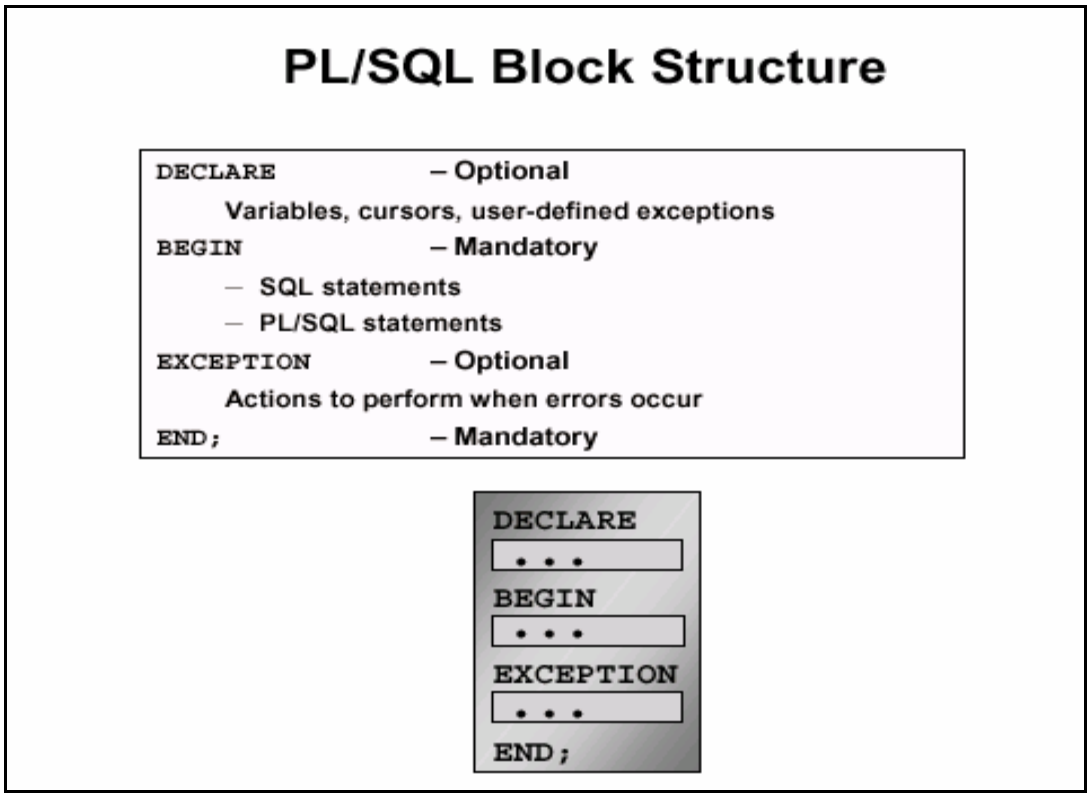
```
/* the second part : PLsql
rem first program: */

Declare

Begin
dbms_output.put_line('Hello world');
end;
/
```

- **المثال التالى يقوم بعرض رسالة ايضا ولكن مع بعض التغييرات:**

```
/* the second part : PLsql
 first program: */
set serveroutput on;

Declare

Begin
dbms_output.put_line('Hello world'||'  hi');
end;
```

الفصل الأول: تعريف المتغيرات

- **الهيكل البنائى لكتلة PL / SQL :**



PL / SQL : هى لغة هيكلية وتعنى أن البرنامج يمكن تقسيمه الى اجزاء وكتل منطقية ويتكون برنامج PL / SQL من وحدة أو كتلة (Block) وتحتوى البلوك على ثلاثة أجزاء:

أ- جزء تعريف : (أختيارى) وهو يضم تعريفات المتغيرات ، المؤشرات، الاستثناءات المستخدمة داخل البلوك أو الكتلة .

ب- جزء تنفيذى : (أساسى) يحتوى على جمل وعبارات اللغة لمعالجة وتناول البيانات .

جـ- جزء للأستثناءات : (أختيارى) تحدد الأعمال وماذا سيتم أذا حدث خطأ ما أو استثناء أثناء تنفيذ البلوك .

- **تنفيذ جمل وعبارات PL / SQL:**

- يتم وضع علامة (;) فاصلة منقوطة فى نهاية كل جملة أو عبارة من جمل لغة PL / SQL

- عند تنفيذ البلوك بدون اخطاء يظهر الجملة التالية لتوضيح تمام وصحة التنفيذ PL /SQL procedure successfully completed .

- لاحظ عدم وجود علامة فاصلة منقوطة فى نهاية الجمل Declare ،او Exception،او Begin.

- لاحظ وجود علاقة فاصلة منقوطة فى نهاية جملة (END ;)

- يمكن كتابة أكثر من جملة (عبارة) على نفس السطر والفصل بينهم بعلامة الفاصلة المنقوطة (;) ولكن لا يحبذ ذلك لجعل قراءة البرنامج أسهل وكذلك التعديل فيه .

- **انواع بلوكات وكتل بغة PL / SQL :**

لاحظ أن انواع كتل لغة PL / SQL يمكن أن تكون منفصلة كليا أو متداخلة ضمن بعضها وتنقسم هذه الكتل الى قسمين ويطلق عليها أسم عام وهو (Sub programs)

1- كتلة مجهولة " Anonymous block " :

وهو برنامج وليس له أسم ويتم تعريفها عند نقطة التطبيق وتنفذ ساعتها ويتم أرسالها الى معالج أوراكل لترجمتها وتنفيذها .

2- " Subprograms" وحدات برمجية :

وهى وحدات وكتل من البرامج تقبل معاملات (عوامل) يمكن أن تستند عليها فى تنفيذ البرنامج .

وتنقسم هذه الوحدات الى انواع أخرى مثل الأجراء (Procedure) دالة (Function) ، زناد (Trigger) .....

وهذه الوحدات يتم حفظها داخل الأوراكل ويتم أستدعاءها عند الحاجة اليها .

- **أستخدام المتغيرات :-**

يمكن أستخدام المتغيرات فى :

- تخزين مؤقت للبيانات .
- التعامل مع قيم مخزنة .
- أعادة أستخدام البيانات نتيجة تغيرات داخل وأثناء سير البرنامج .
- سهولة التعديل والصيانة وذلك من خلال أستخدام Rowtype % ، type % التى سوف تشرح لاحقا ، ومن خلالهما يمكن تعريف متغير حسب نوع عمود أو صف فى قاعدة البيانات مما يتيح قدر كبير من المرونة دون التقيد بنوع معين من البيانات .

- **التعامل مع المتغيرات :**

- اولا يتم تعريف المتغيرات وذلك فى الجزء التعريفى " Declare" وكذلك يمكن أن يتم قيم أبتدائية لهذه المتغيرات .
- تعيين وأدخال قيم جديدة للمتغيرات فى الجزء التنفيذى .
- يمكن أن تمرر القيم بين كتل PL / SQL من خلال المعاملات (parameters)
- عرض النتائج من خلال المتغيرات .

أنواع المتغيرات

Types of variables

تنقسم المتغيرات الى نوعين أساسيين :-

1- متغيرات PL / SQL : وتحتوى على عدة أنواع منها

- Scalar المفردة
- Composite المركبة (المعقدة)
- Reference المشار بها (عناوين)
- lob "large objects" ذات الأحجام الكبيرة

2- متغيرات ليست PL / SQL : مثل متغيرات Bind ، host

أولا المتغيرات الخاصة PL / SQL :

- المفردة Scalar : وتحتوى على قيم مفردة ولا يمكن أن تجزأ الى قيم مفردة أصغر مثل Number ، Varchar2 ، Data ، Boolean فهى انوع لا يمكن ان يحتوى المتغير سوى على قيمو واحدة كما لا يمكن ان تجزأ هذه القيمة .
- المركبة (المعقدة التركيب) Composite :

وتحتوى على مجموعة من الأجزاء كل جزء ذات تركيب محدد ويمكن أن يختلف عن جزء أخر ويتم التعامل مع كل جزء على حدى مثل Record فهو يمكن أن يحتوى على جزء من النوع Number وجزء أخر من النوع Data وهكذا ، وسيتم شرحها لاحقا .

- المشار بها (عناوين)" Reference " :

وهذه المتغيرات تحتوى قيم تشير الى برامج وتطبيقات أخرى وليست قيم بذاتها يمكن أستخدامها وهذا النوع لن يتم تغطية فى هذا المنهج .

- متغيرات ذات الأحجام الكبيرة " Lob" :

وهى تحتوى على أنواع من البيانات التى تحتاج الى مساحة كبيرة مثل الصور ، الفيديو ، الكتب ، رسومات وسوف يتم شرحها لاحقا .

- أدخال قيم وأخراجها من المتغيرات :

- لغة PL / SQL ليس لديها قدرات وأدوات للأدخال أو الأخراج بداخلها .
- لذلك يتم أستخدام متغيرات من النوع host لأدخال القيم داخل المتغيرات فى زمن التنفيذ (run time) تقوم المتغيرات host (Bind) بمهمة أدخال وأخراج القيم .

تعريف المتغيرات فى PL / SQL :

الصيغة

Identifier (Constant) data type (not null) {:=/ default expr}

أمثلة

```
V_ Hiredate        Date;
V_deptno           number (2) not null := 10 ;
V_location  Varchar2 (15) :='DALLAS' ;
C_ comm.   Contrast number: = 1600 ;
```

شرح الصيغة :

Identifier: هو أسم المتغير يجب أن يلتزم بقواعد التسمية.

Constant: أن المتغير المراد تعريفه هو ثابت لا تتغير قيمته التى سوف يتم بدئه بها.

Data type: نوع بيانات المتغير سواء كان مفرد أو معقد او ذات حجم كبير .

Not null: لا يسمح له بأخذ قيمة Null أثناء والتنفيذ ويجب أن يبدأ بقيمة معينة على الأقل .

Expr: وهو قيمة أبتدائية للمتغير سواء كانت قيمة ثابتة مثل '22-may-oo' او'c' او 5 أو عملية حسابية ما .

<u>خطوط عامة لتسمية المتغيرات</u>

- **يجب أتباع قواعد التسمية الدالة على محتوى المتغير**
- **يجب وضع قيم أبتدائية للمتغيرات خاصة ذات طبيعة not null أو constant ثابت .**
- **تعريف متغير واحد فى كل سطر**
- **أستخدام (:=) أو تعبير (Default) لوضع قيم أبتدائية داخل المتغيرات .**

أمثلة لتعريف وأستخدام المتغيرات :

مثال 1:

```
Declare
        V_hiredate   Date;
Begin
        V_hiredate:= '15-may – 1999';
End;
```

مثال 2

```
Declare
V_mgr number (4) Default 100;
Begin
V_mgr := 120;
End;
```

مثال 3

```
Declare
V_city varchar2 (30)        not null :='oxford';
Begin
V_City := 'Dallas';
```

End;

مثال 1- يتم فيه تعريف متغير أسمه (V_hiredate) من النوع تاريخ (Date) وذلك الجزء التعريفى ، ويتم وضع قيمة داخله فى الجزء التنفيذى وهذه القيمة هى تاريخ (15-مايو 1999) .

مثال 2: يتم تعريف متغير من النوع الرقمى (العددى) مساحته أربع خانات وذات قيمة أتوماتيكية هى(100) وذلك فى الجزء التعريفى .

وفى الجزء التنفيذى يتم وضع قيمة داخله (120)

مثال 3 : فى الجزء التعريفى :

تعريف متغير (V-city) من النوع الحرفى سعته 30 حرف من النوع (not null) لا يأخذ قيم فارغة ذات قيمة أبتدائية (oxford) فى الجزء التنفيذى :

يتم وضع قيمة ('Dallas') فى هذا المتغير .

الأنواع المفردة للمتغيرات :

- تحجز قيمة مفردة .
- لا تحتوى على أجزاء داخلية أو تراكيب .

أمثلة على الانواع المفردة :

Char (length) : متغير حرفى ثابت السعة سواء تم ملاءها أو تركت فارغة وهذا النوع مضر فى المساحة لكنه أسرع فى التعامل .

Varchar2 (length): متغير حرفى ذات سعة معينة لكن هذه السعة متغيرة بحد أقصى ويتم ملاء المتغير بسعة النص فقط بحد أقصى سعة هذا المتغير فمثلا متغير حرفى ذات سعة 30 حرف ولم يوضع سوى 6 حروف يملأ بالحروف الستة ويتم توفير الباقى وهذا النوع مفيد فى المساحة لكن ابطئ من النوع السابق (Char) .

Long: هو النوع الاساسى للبيانات النصية ذات سعة بحد أقصى 32760 بايت أو 312 حرف .

Long row : وهو مثل long لكنه لا يتم التعامل به ولا يفهمه PL / SQL .

Number (p,s) : وهو متغير رقمى يأخذ نطاق من خانات واحدة الى 38 خانة وكذلك من كسر عشرى من -84 الى 127

P : عدد خانات الرقم عامة بما فيها من كسر عشرى .

S : عدد خانات الكسر العشرى فقط.

تابع أنواع المفردة للمتغيرات :

- binary_integer : وهو النوع الرقمى الصحيح (لايأخذ كسور) ويأخذ قيم خلال ±2147483 .
- Pls_integer : وهو مثل النوع السابق لكنه أسرع ويأخذ مساحة أقل .

- Boolean : وهو نوع يمكن أن يأخذ ثلاث قيم فقط هى (null ، false ، true)
ويستخدم فى حالات الشروط والمقارنات المنطقية فقط .
- Data : نوع المتغيرات التاريخية (الوقت) يحتوى على بيانات تاريخ أو وقت أو زمن
وهو يبدأ من 4712 قبل الميلاد الى 9999 ميلادية .

أمثلة أخرى على تعريف المتغيرات :

```
Declare
v_job  varchar2 (15);                    متغير حرفى سعة 15 حرف
v_count    binary-integer:=o;            متغير رقمى صحيح يأخذ قيمة أبتدائية صفر
v_total_sal    number متغير رقمى من سبع خانات منهم أثنان كسر عشرى ويبدأ بصفر
v_order_date  date:=                     متغير زمنى يبدأ فى الأسبوع القادم (7.2):= o;
                                         sysdate +7;
متغير ثابت رقمى مكون من أربع خانات منهم خانتان وقيمته 17.25
C-TAX-RATIO  CONSTANT  NUMBER (4.2):=17.25;
متغير منطقى    " boolean" لا يأخذ فارغ "null" ويبدأ بقيمة  true
V_Flag        Boolean   Not Null :=True;
```

الخاصية %type :

يمكن تعريف متغير على أساس تعريف عمود فى جدول بقاعدة البيانات او نفس تعريف متغير سبق تعريفه . فالمتغير الجديد هنا نفس نوع البيانات "data type" للمتغير القديم دون أخذ ما به من قيمة وهذه الخاصية تتيح قدر كبير من المرونة فى تعريف المتغيرات لأن تعريف متغير رقمى مثلا ويأخذ الرقم التعريفى "praimary key" لبيان ما (موظف مثلا) واذا ما دعت الحاجة لتغيير نوع البيانات فى الجدول من رقمى الى حرفى فأنه سوف يؤدى الى تعطل البرنامج المبنى على اساس انه هو متغير (عمود) حرفى لذلك على اساس انه متغير (عمود) حرفى لذلك نستخرج خاصية type % لتلاقى هذه المشكلة .

مثال

```
Dedare
V_name      emp. Ename % type ;
V_id   emp. Emp no % type ;
V_sal       emp, sal % type := 1200 ;
```

الصيغة العامة : Ideutifier table,cdumn_name% type

أستخدام المتغيرات host , Bind

تعريف هذا النوع يكون خارج كتل PL / SQL كما يأتى :

```
Variable g_sal number
Begin
        :g_sal := 1200 ;
end ;
/
print g_sal ;
```

لاحظ عدم وجود علامة (;) فاصلة منقوطة فى أخر جملة تعريف المتغير (gsal) لأنها ليست جملة pl/sql .

عند أستخدام هذا النوع نضع أمام المتغير (:) وذلك داخل كود PL / SQL .

يتم طبع واظهار قيم هذه المتغيرات من خارج PL / SQL وذلك من خلال جملة print .

أستخدام : DBMS_output . put_line

- وهى دالة داخلية فى الأوراكل الغرض منهما أظهار قيم ونصوص وعرض البيانات داخل بلوكات وكتل PL / SQL .

- يجب أن يكون أختيار (set server out put on) لعرض وما ستعرضه هذه الدالة .

مثال:

```
SET SERVEROUTPUT ON
DEFINE p_annual_sal = 60000
DECLARE
  v_sal NUMBER(9,2) := &p_annual_sal;
BEGIN
  v_sal := v_sal/12;
  DBMS_OUTPUT.PUT_LINE ('The monthly salary is ' ||
                        TO_CHAR(v_sal));
END;
/
```

الفصل الثانى

كتابة جمل PL / SQL

- فى نهاية كل بلوك فى لغة PL / SQL يجب وضع علامة (/) "salash" لتنفيذ ما سبقها من كتل (البلوكات) PL / SQL .

صيغ الملاحظات وجعل الكود لا يأخذ به من قبل المترجم :

1- لجعل سطر ما تعليق (ملاحظة) نضع فى أول السطر علامتين (- -) "dash" أو ناقص أو كتابة فى أول السطر كلمة (rem) .

2- لجعل أكثر من سطر تعليق أو ملاحظة نستخدم علامات فى البداية (*/) وعلامة (/*) فى نهاية التعليق او الملاحظة .

مثال :

```
DECLARE
...
  v_sal NUMBER (9,2);
BEGIN
  /* Compute the annual salary based on the
     monthly salary input from the user */
  v_sal := :g_monthly_sal * 12;
END;      -- This is the end of the block
```

وتستخدم هذه الملاحظات لعمل ملاحظات تعيين المبرمج على فهم البرنامج وكيفية عمله وذلك من الأمور المعروفة .

- الدول العاملة داخل لغة PL / SQL

SQL Functions in PL/SQL

- Available in procedural statements:
  - Single-row number
  - Single-row character
  - Data type conversion
  - Date
  - Timestamp
  - GREATEST and LEAST
  - Miscellaneous functions

} Same as in SQL

- Not available in procedural statements:
  - DECODE
  - Group functions

- كل الدوال " single row function" فى لغة SQL أيضا هنا ما عدا دالة " decode" وبعض الدوال الأخرى
- مثال:

SQL Functions in PL/SQL: Examples

- Build the mailing list for a company.

```
v_mailing_address := v_name||CHR(10)||
                     v_address||CHR(10)||v_state||
                     CHR(10)||v_zip;
```

- Convert the employee name to lowercase.

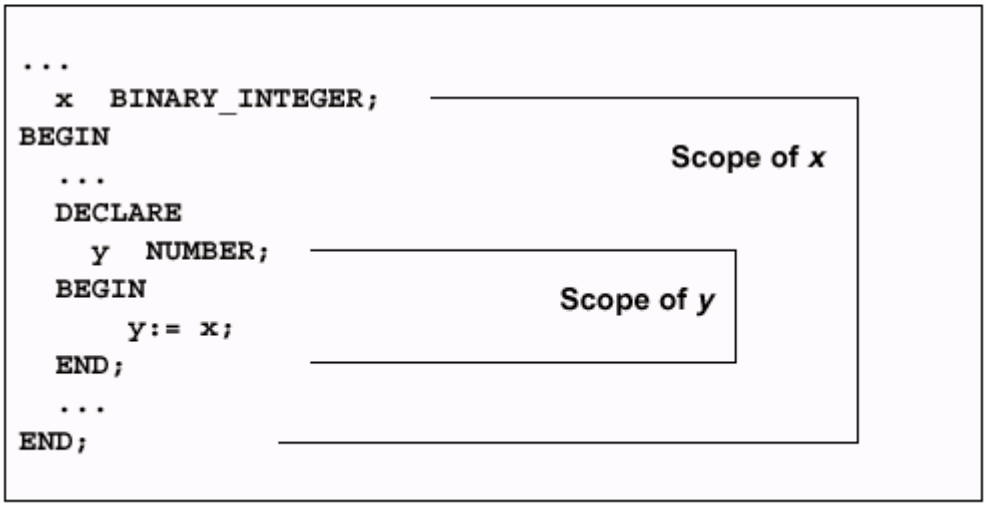
```
v_ename       := LOWER(v_ename);
```

الكتل والبلوكات المتداخلة نطاق عمل المتغيرات :

- يمكن داخل PL / SQL تداخل أكثر من بلوك (كتلة) مع بعضهم البعض .
- نطاق عمل المتغيرات داخل البلوكات كالأتى :

Nested Blocks and Variable Scope

Example:



فى الشكل المتغير (x) يتم التعامل معه فى البلوك الخارجى والداخلى أما المتغير (y) نطاق عمله فقط فى البلوك الداخلى .

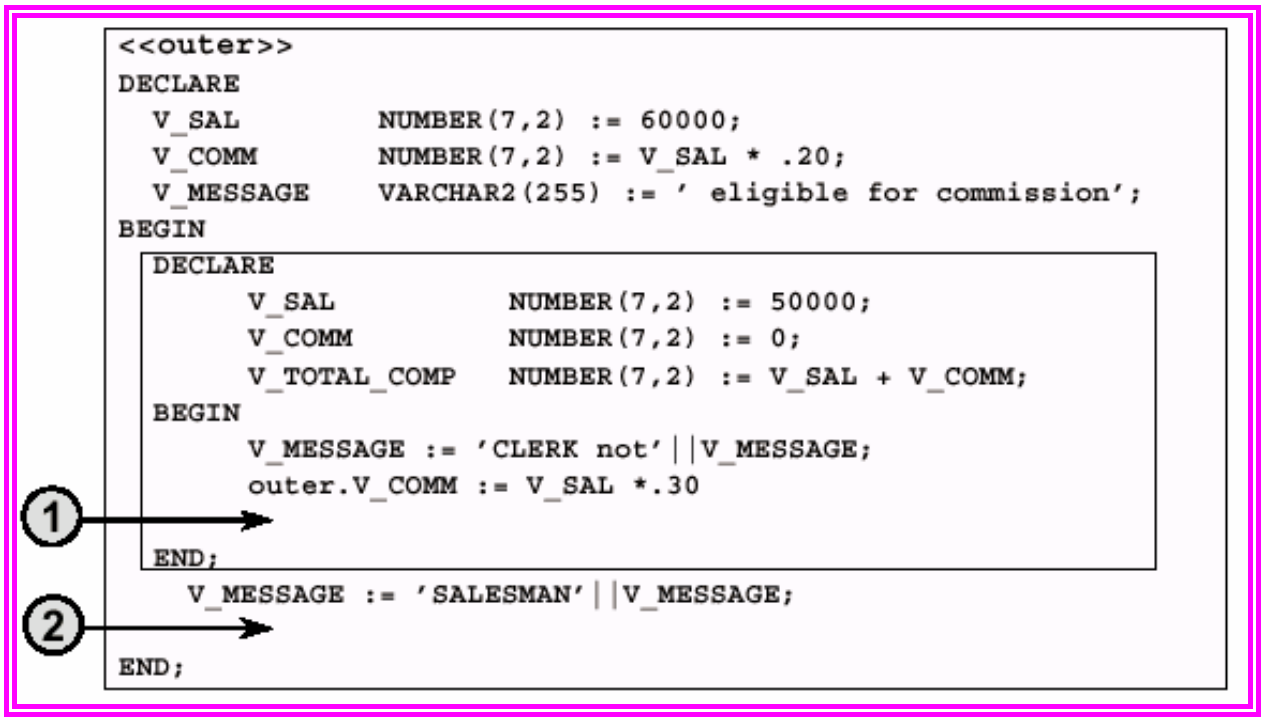
- ومن ذلك يتضح أن – البلوك يرى البلوكات التى تحتويه فقط .
- البلوكات التى داخل هذا البلوك الخارجى لها لا يستطيع التعامل معها هذا البلوك الخارجى .

وضع تسمية البلوكات

```
<<outer>>
   DECLARE
      birthdate DATE;
   BEGIN
         DECLARE
             birthdate DATE;
         BEGIN
            ...
            outer.birthdate :=
                          TO_DATE('03-AUG-1976',
                                  'DD-MON-YYYY');
         END;
    ....
    END;
```

يمكن تسمية البلوك وأستخدام هذه التسمية اذا حدث أن المتغيرات كانت بنفس الأسماء داخل بلوكات متعددة – وكذلك لأعطاء مزيد من التوضيح للكود لمعرفة كل بلوك بأسم معين يساعد على الفهم والأستدلال .

مثال :



فى الموضوع 1- يتم أدخال قيمة المعادلة فى v_com الخارجى

فى الموضع 2- يتم وضع قيمة جديدة v_message وهذا المتغير فى البلوك (outer) العمليات .

- **وضع العمليات الحساسية والمنطقية فى PL / SQL :**

مثل SQL تماما بنفس الفكر والترتيب .

أمثلة :

1- لعمل عداد رقمى ; v:=v+1

2 - لعمل مؤشر منطقى ; v_flag := (v1 = v2)

فى المثال 2 : معناه اذا كانت V1 يساوى V2 فأن المتغير v_flag قيمة true لأنه من نوع Boolean واذا لم يتساوى v1 ، يساوى v2 فان المتغير v_flag يأخذ False لأنه من نوع v_flag ويأخذ v_flag واذا كان أحدهما null يأخذ القيمة null .

الفصل الثالث

تفاعل أوركال كقاعدة بيانات مع لغة PL / SQL

جمل SQL فى لغة PL / SQL :

أحضار بيانات من قاعدة البيانات بأستخدام أمر (select)

- **عمل أضافات وتغيرات لصفوف فى قاعدة البيانات بأستخدام أوامر DML**
- **أستخدام أوامر التحكم فى التفاعل مع قاعدة البيانات (control transaction commands) مثل Commit ، rollback ، save point .**
- **أدراج مخرجات عمليات DML بأستخدام صفات (implicit cursor) المؤشرات الضمنية .**

- **جملة select فى لغة PL / SQL :**

الصيغة العامة :- **Select name,.....**

Into (variable_name / record_name)

From table

[Where condition]

من الملاحظ من الصيغة العامة لا يوجد أى تغيير على جملة select فى لغة PL/SQL سوى أضافة جملة (into) وبعدها أسماء المتغيرات أو (record) الذى سوف يستقبل قيم جملة select

وتوضع جملة (into) بعد سرد أعمدة select وقبل كلمة from وفى أخر جملة select توضع الفاصلة المنقوطة (;) دلالة على نهاية الجملة وليس داخلها .

- **لاحظ ان جملة (where) أختيارية ليست أساسية .**
- **لاحظ ان عدد ونوع بيانات المتغيرات فى into متوافق مع عدد ونوع البيانات المختارة بعد كلمة select**

<u>ملحوظة أساسية</u> : يجب ان ترجع select بقيمة واحدة أو صف واحد فقط وذلك لوضعها داخل المتغيرات .

تابع : جملة select فى لغة PL / SQL

```
DECLARE
  v_deptno           NUMBER(4);
  v_location_id      NUMBER(4);
BEGIN
  SELECT      department_id, location_id
  INTO        v_deptno, v_location_id
  FROM        departments
  WHERE       department_name = 'Sales';
  ...
END;
/
```

فى المثال السابق يتم تعريف متغيرا اسمه v_deptno ونوعه رقمى سعته أربعة حروف ومتغير أسمه v_location_id ونوعه رقمى سعته أربعة حروف .

ثم الجزء التنفيذى :

به جملة Select وتختار رقم الأدارة " department_id" ورقم المكان "location_id " ونضعهم بأستخدام (into) فى المتغيرين v_deptno ، location_id على الترتيب وذلك من جدول (departments) حيث (where) تفعل على ذلك على الأدارة ألتى اسمها ' sales' (department_name).

لا يعمل الكود فى حالتين :

1- اذا رجعت جملة select بأكثر من قيمة ستظهر أستثناء أسمه "too_many_ows" ويظهر خطأ ولا يستكمل البرنامج .

2- أذا رجعت جملة select ولا قيمة (لا توجد قيم ناتجة عن جملة select) وهذا ينشأ أستثناء أسمه " no_date_found " وكذلك يظهر خطأ ولا يستمكل البرنامج .

- أسترجاع البيانات فى لغة PL / SQL :

```
DECLARE
  v_hire_date    employees.hire_date%TYPE;
  v_salary       employees.salary%TYPE;
BEGIN
  SELECT    hire_date, salary
  INTO      v_hire_date, v_salary
  FROM      employees
  WHERE     employee_id = 100;
  ...
END;
/
```

من هذا الكود نستنتج نفس الأفكار مثل الكود السابق مع بعض الأضافات البسيطة مثل – أستخدام خاصية type % فى تعريف المتغيرات .

```
SET SERVEROUTPUT ON
DECLARE
  v_sum_sal    NUMBER(10,2);
  v_deptno     NUMBER NOT NULL := 60;
BEGIN
  SELECT       SUM(salary)   -- group function
  INTO         v_sum_sal
  FROM         employees
  WHERE        department_id = v_deptno;
  DBMS_OUTPUT.PUT_LINE ('The sum salary is ' ||
                        TO_CHAR(v_sum_sal));
END;
/
```

وهذا المثال يوضح كيفية أستخدام المتغيرات وكذلك استرجاع البيانات بأستخدام select ، وعمل group function تتيح لنا معرفة مجموع مرتبات موظفى ادارة معينة كما فى المثال وعرض هذا المجموع فى نهاية المثال .

- **امثلة(على المستخدم Scott):**

```
set serveroutput on;
declare
v_name varchar2(30);
v_sal number;
v_date date;
begin
select ename,sal,hiredate into v_name,v_sal,v_date
from emp
where empno=7788;
dbms_output.put_line('Employee : '||v_name);
dbms_output.put_line('Salary : '||v_name);
dbms_output.put_line('Hire Date : '||v_date);

end;
```

مثال 2:

```
set serveroutput on;
declare
v_deptno emp.deptno%type:=&deptno;
v_sal emp.sal%type;
begin
select deptno,sum(sal) into v_deptno,v_sal
from emp
where deptno=v_deptno
group by deptno;
dbms_output.put_line('deptno : '||v_deptno);
dbms_output.put_line('total Salary : '||v_sal);

end;
```

مثال 3:

```
set serveroutput on;
declare
v_deptno emp.deptno%type:=&deptno;
v_sal emp.sal%type;
v_deptname dept.dname%type;
begin
select d.deptno,dname,sum(sal) into v_deptno,v_deptname,v_sal
from emp e,dept d
where d.deptno=v_deptno
      and e.deptno=d.deptno
group by d.deptno,dname;
dbms_output.put_line('************************');
dbms_output.put_line('deptno : '||v_deptno);
dbms_output.put_line('dept name : '||v_deptname);
dbms_output.put_line('total Salary : '||v_sal);
dbms_output.put_line('************************');
end;
```

قواعد التسمية :

Naming Conventions

```
DECLARE
  hire_date       employees.hire_date%TYPE;
  sysdate         hire_date%TYPE;
  employee_id     employees.employee_id%TYPE := 176;
BEGIN
  SELECT      hire_date, sysdate
  INTO        hire_date, sysdate
  FROM        employees
  WHERE       employee_id = employee_id;
END;
/
```

```
DECLARE
*
ERROR at line 1:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 6
```

يوضح الشكل السابق حدوث تعارض نتج عنه تنفيذ البرنامج وذلك لتسمية المتغيرات بأسماء اجزاء فى قاعدة البيانات (أعمدة فى جداول) وهذا غير مرغوب مما ينتج عنه من مشاكل أثناء التنفيذ والفهم وكذلك بعدها فى التطوير .

التعامل مع البيانات بأستخدام PL / SQL :

ونعنى هنا جمل DML : كما هو معروف أن PL / SQL يتعامل مع جملة DML بطريقة مباشرة كما يلى اما جمل DDL فأنها لا تتعامل معها لغة PL / SQL الا بطريقة غير مباشرة نراها لاحقا وفى الجزء التالى سنتعامل مع جمل

(merge, delete, update, insert)

أولا insert

```
BEGIN
  INSERT INTO employees
  (employee_id, first_name, last_name, email,
   hire_date, job_id, salary)
  VALUES
   (employees_seq.NEXTVAL, 'Ruth', 'Cores', 'RCORES',
    sysdate, 'AD_ASST', 4000);
END;
/
```

فى PL / SQL جملة insert مثل SQL تماما بالأضافة الى :

- أستخدام الدوال المختلفة فيها .
- أستخدام المتغيرات (sequence) لأدخال قيم المفتاح (primary key) .
- أضافة قيم أتوماتيكية لعمود ما فى الجدول .

مثال على جملة Insert:

```
set serveroutput on;
declare
v_deptno emp.deptno%type:=&deptno;
v_dept_name dept.dname%type:='&dept_name';
begin
v_dept_name:=nvl(v_dept_name,'No Name');
insert into dept(deptno,dname)
values(v_deptno,v_dept_name);

dbms_output.put_line('******** data inserted ***************');
dbms_output.put_line('deptno : '||v_deptno);
dbms_output.put_line('dept name : '||v_dept_name);
dbms_output.put_line('************************');
end;
```

ثانيا update

```
DECLARE
  v_sal_increase   employees.salary%TYPE := 800;
BEGIN
  UPDATE      employees
  SET         salary = salary + v_sal_increase
  WHERE       job_id = 'ST_CLERK';

END;
/
```

يمكن أضافة جملة (update) داخل كود PL / SQL ولكن الجديد أنها أصبحت لدى المبرمج الكثير من الحرية نتيجة وجود أمكانيات PL / SQL مثل المتغيرات وكذلك (cursors) فيما بعد سيتم شرحه وباقى أمكانيات اللغة .

مثال على جملة Update :

```
set serveroutput on;
declare
v_deptno emp.deptno%type:=&deptno;
v_dept_name dept.dname%type:='&dept_name';
begin
v_dept_name:=nvl(v_dept_name,'OPerations');

update dept
set  dname=v_dept_name
where deptno=v_deptno;

dbms_output.put_line('******** data updated***************');
dbms_output.put_line('deptno : '||v_deptno);
dbms_output.put_line('dept name : '||v_dept_name);
dbms_output.put_line('************************');
end;
```

ثالثا delete

```
DECLARE
  v_deptno    employees.department_id%TYPE := 10;
BEGIN
  DELETE FROM    employees
  WHERE          department_id = v_deptno;
END;
/
```

يمكن كتابة جمل الحذف delete بكل سهولة ويسر كما نرى فى الكود السابق مع مراعاة شروط

(integrity constraint) التى تحافظ على الربط بين الجداول وبعضها بالأضافة الى أن المتغيرات السابقة ذكرها مع جملة update مطبقة هنا أيضا وسوف نرى مدى جدوى هذه الأمكانات .

المثال التالى يوضح عملية الحذف باستخدام الجملة السابقة:

```
set serveroutput on;
declare
v_deptno emp.deptno%type:=&deptno;
begin

delete from dept
where deptno=v_deptno;

dbms_output.put_line('******** data deleted ***************');
dbms_output.put_line('deptno : '||v_deptno);
dbms_output.put_line('************************');
end;
```

رابعا Merge

```
DECLARE
      v_empno EMPLOYEES.EMPLOYEE_ID%TYPE := 100;
BEGIN
MERGE INTO copy_emp c
     USING employees e
     ON (c.employee_id = v_empno)
   WHEN MATCHED THEN
     UPDATE SET
       c.first_name      = e.first_name,
       c.last_name       = e.last_name,
       c.email           = e.email,
       . . .
   WHEN NOT MATCHED THEN
     INSERT VALUES(e.employee_id, e.first_name, e.last_name,
          . . .,e.department_id);
END;
```

Merging Data (continued)

```
DECLARE
       v_empno EMPLOYEES.EMPLOYEE_ID%TYPE := 100;
BEGIN
MERGE INTO copy_emp c
     USING employees e
     ON (c.employee_id = v_empno)
   WHEN MATCHED THEN
     UPDATE SET
       c.first_name      = e.first_name,
       c.last_name       = e.last_name,
       c.email           = e.email,
       c.phone_number    = e.phone_number,
       c.hire_date       = e.hire_date,
       c.job_id          = e.job_id,
       c.salary          = e.salary,
       c.commission_pct  = e.commission_pct,
       c.manager_id      = e.manager_id,
       c.department_id   = e.department_id
   WHEN NOT MATCHED THEN
     INSERT VALUES(e.employee_id,  e.first_name, e.last_name,
          e.email, e.phone_number, e.hire_date, e.job_id,
          e.salary, e.commission_pct, e.manager_id,
          e.department_id);
END;
/
```

جملة Merge نرى أنها تطبق داخل PL / SQL بكل سهولة ويسر كما فى المثالين السابقين .

المؤشر SQL أو " SQL cursor "

س : ما هو المؤشر SQL أو " SQL cursor " ؟

جـ - هو جزء خاص من الذاكرة يفتحه الأوراكل لتنفيذ أى جملة DML داخله .

خصائص المؤشر SQL (SQL cursor) : -

1- SQL % rowcount يوضح عدد الصفوف التى تمت عليها أخر جملة DML نفذها الأوراكل .

2- SQL % found وهو يرجع قيم Boolean ويرجع بقيمة true اذا كانت أخر جملة DML نفذها الأوراكل نفذت على صف واحد على الأقل والعكس صحيح .

3- SQL % notfound وهى عكس الخاصية السابقة .

4- SQL % ISOPEN توضح ما اذا كان المؤشر مفتوحا ام لا وهى دائما false.

مثال على استخدام SQL CURSOR :

```
VARIABLE rows_deleted VARCHAR2(30)
DECLARE
  v_employee_id employees.employee_id%TYPE := 176;
BEGIN
  DELETE FROM  employees
  WHERE        employee_id = v_employee_id;
  :rows_deleted := (SQL%ROWCOUNT ||
                        ' row deleted.');
END;
/
PRINT rows_deleted
```

يقوم المثال بالتالى :

- تعريف منغير من النوع (Bind) Row_deleted ونوع بياناته varchar2(30) وذلك خارج بلوك Pl/SQL .
- الجزء التعريفى :
  - يعرف متغير اسمه V_EMPLOYEE_ID من نفس نوع العمود EMPLOYEE_ID فى جدول EMPLOYEE ويأخذ قيمة ابتدائية 176 .
- الجزء التنفيذى :
  - يستخدم الأمر DELETE لحذف الصف الذى EMPLOYE_ID يساوى 176 .
  - ثم يستخدم المتغير الخارجى (BIND) "لاحظ وجود (:) قبل استخدامه" ويضع فيه عدد الصفوف المحزوفه بإستخدام الخاصية SQL%ROWCOUAT ليعرف عدد الصفوف التى تم حذفها .

جملة التحكم TRANSACTION CONTEROL STATEMANTS :

- يمكن داخل بلوكات وكود لغة PL/SQL استخدام أوامر ROLLBACK أو CAMMIT وكذلك الأمر SAVEPOINT .

الفصل الخامس

التعامل مع انواع البيانات المعتمدة
Working with Composite datatypes

فى هذا الفصل سوف نغطى نوعين من أنواع البيانات المعتمدة التركيب

1- index by table(array)
2- السجل record

وتتميز هذه الانواع بصفات عامة هى :

- يمكن تقسيمها داخليا الى مكونات مختلفة
- يمكن أعادة استخدامها

أولا: السجل record

وهو مثل المتغيرات المفردة فى كثير من الصفات

- السجل عبارة عن متغيرات مرتبطة ببعضها ويتم التعامل كوحدة واحدة.
- وهو مشابة فى تركيبة لل (structure) فى لغات البرمجة من الجيل الثالث مثل Pascal , c.
- وهى ليست مثل صف من صفوف الجداول تماما.
- يمكن استقبال قيم صف من قاعدة البيانات أو جدول منها مباشرة.

فمثلا ممكن للسجل record أن يحتوى على جزء number للمرتب salary وجزء رقمى اخر لرقم الادارة وأخرى حرفىchar لاسم الموظف وهكذا.....
ولكن كل واحد من هذه الحقول يكون لديه قيمة واحدة فقط
الصيغة العامة :

Creating a PL/SQL Record

Syntax:

```
TYPE type_name IS RECORD
      (field_declaration[, field_declaration]…);
identifier    type_name;
```

Where *field_declaration* is:

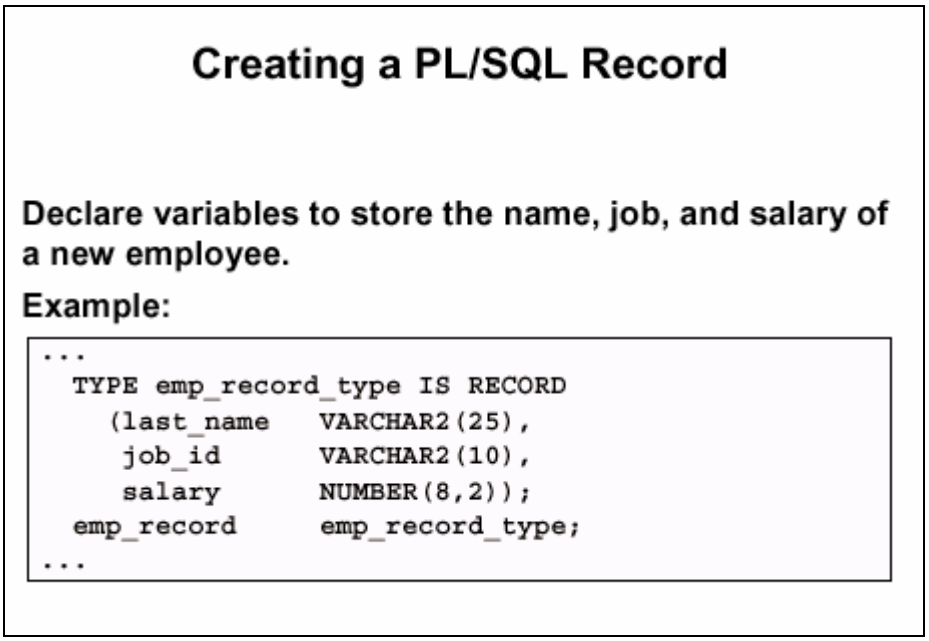
```
field_name {field_type | variable%TYPE
            | table.column%TYPE | table%ROWTYPE}
            [[NOT NULL] {:= | DEFAULT} expr]
```

Type_name : هو نوع السجل
Filed-name : اسم الحقل داخل السجل
Field-type : نوع البيانات داخل الحقل الذى داخل السجل

أولا : يتم تعريف نوع الذى يحتوى على بينة السجل لكى يتم عمل منها سجلات بنفس البينة فمثلا هناك بنية datatype (هيكل) رقمى (number) داخل قاعدة البيانات ونعمل منه أمثلة وأشكال وهى المتغيرات الرقمية.
مثلا : v_id number

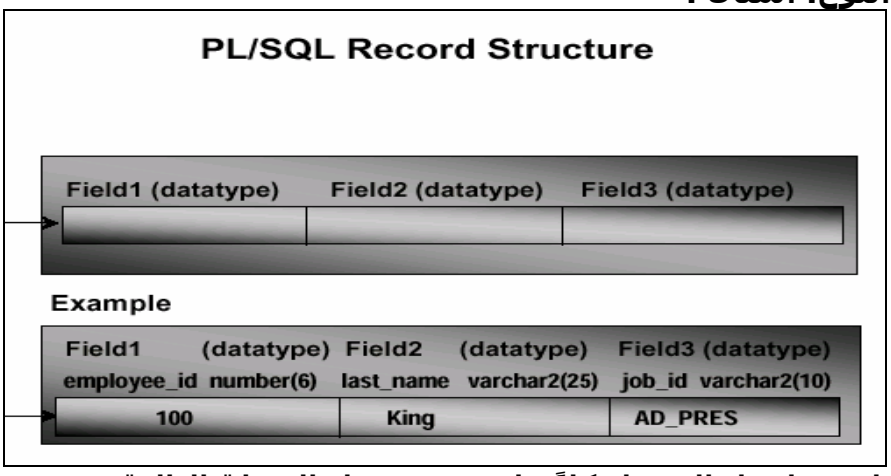
لأن النوع (number) موجود اصلا داخل قاعة البيانات لذلك نضع اولاً لنا نوع جديد وهو السجل ونحدد تركيبه داخليا وبعدها نحدد مثال منه وهو الذى سوف نستخدمه فى داخل البرنامج

مثال :



فى هذا المثال يتم تحديد نوع emp_record_type ونكون هذا النوع كما فى المثال ثم نكون من هذا النوع مثال او حالة وهذا المثال هو الذى سوف يستخدم فى البرنامج بعد ذلك وهذا المثال فى الكود هو emp_record و طبعا له نفس تركيب النوع الذى هو مكون منه.

الفرق بين النوع، المثال :



ويتم التعامل مع اجزاء السجل كلاً على حدى مثل الجملة التالية

Emp_record. Job_id : = 'manger';

الجزء الاول هى اسم السجل ثم نقطة ثم اسم الحقل ويتم التعامل معه بهذه الطريقة .

الخاصية %rowtype

بهذه الخاصية يمكن عمل سجل يأخذ نفس تركيب السجل فى جدول موجود بقاعدة البيانات لذلك لا نحتاج أن (نضع) نعرف اولا نوع موجود بالفعل وهو بنيه هذا الجدول وهنا يتم أخذ بنيه الجدول بنفس أنواع البيانات وأسماء الاعمدة وقيم not null والقيم الاتوماتيكية (الافترضية) default

مثال : -

```
Declare
Emp_record      employees % rowtype;
Begin
Emp_record. last_name: ='smith'
End;
```

المثال التالى يوضح كيفية استخدام السجل "record" فى استقبال نتائج جملة "select"

```
Declare
 Emp_record employees%RowType;
Begin
 Select  *      into  emp_record
From   employees
Where   employee_id = 100;
```

مميزات الخاصية % rowtype

- فى بعض الاحيان يكون مطلوب عدم معرفة انواع البيانات وعدد الاعمدة فى قاعدة البيانات.
- من الممكن تعديل عدد وانواع بيانات اعمدة قاعدة البيانات اثناء استخدام التطبيقات
- هذه الخاصية مفيدة جدا عن استخدام جملة (* select)

مثال على استخدام الخاصية % rowtype

```
DEFINE employee_number = 124
  DECLARE
   emp_rec          employees%ROWTYPE;
  BEGIN
   SELECT * INTO emp_rec FROM   employees
   WHERE  employee_id = &employee_number;
   INSERT INTO retired_emps(empno, ename, job, mgr, hiredate,
                            leavedate, sal, comm, deptno)
   VALUES (emp_rec.employee_id, emp_rec.last_name, emp_rec.job_id,
   emp_rec.manager_id, emp_rec.hire_date, SYSDATE, emp_rec.salary,
   emp_rec.commission_pct, emp_rec.department_id);
   COMMIT;
  END;
  /
```

المثال يوضح استقبال نتيجة (* select) فى سجل ثم استخدام هذا السجل لادخالة فى جدول آخر له نفس الهيكل البنائى للسجل

Index by

وهو نوع أخر من انواع البيانات المعقدة ويتكون من جزئين اساسيين : -

1 - مفتاح اساسى من النوع binary_integer او النوع integer

2 - عمود من النوع المفرد او النوع سجل

طول هذا النوع غير محدد ولا قيود عليه

وهو نفس الشكل (array) او المصفوفه فى اى لغة برمجه مثل c لكن هنا هو (array) من بعد واحد فقط. (one dimension).

الصيغة العامة :

Creating an INDEX BY Table

Syntax:

```
TYPE type_name IS TABLE OF
     {column_type | variable%TYPE
     | table.column%TYPE} [NOT NULL]
     | table.%ROWTYPE
     [INDEX BY BINARY_INTEGER];
identifier    type_name;
```

Declare a INDEX BY table to store names.

Example:

```
...
TYPE ename_table_type IS TABLE OF
                              employees.last_name%TYPE
  INDEX BY BINARY_INTEGER;
ename_table ename_table_type;
...
```

وهو يسير على نفس طريقة تعريف وعمل سجل حيث انه يتم :

أولا - تعريف النوع وضع شكل البنيه

ثانيا - عمل امثلة وحالات من هذا النوع الذى تم تعريفه مسبقا.

من خلال الصيغة العامة :

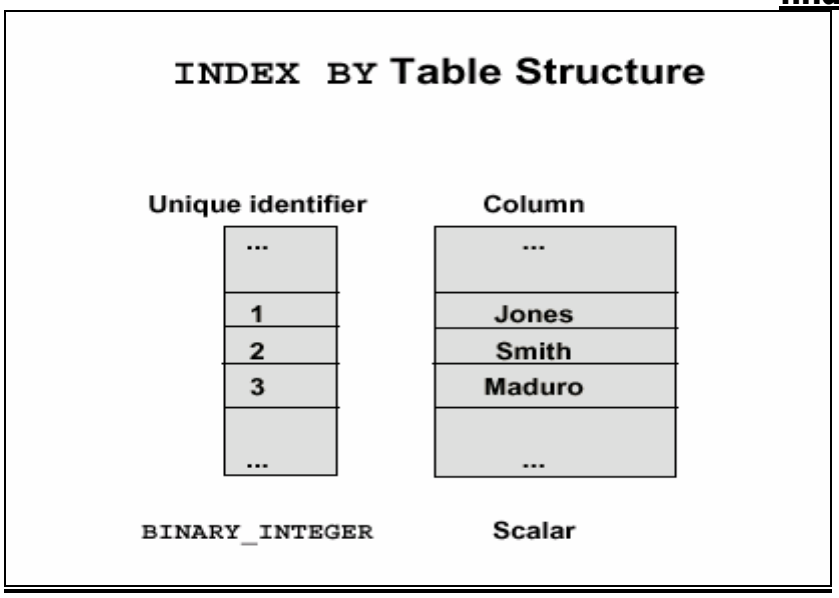
TYBE_NAME : اسم النوع المراد تصميمه

Column_type : اى نوع مفرد أو مركب مثل (varchar2 ,% type) date,

Identifier : هو المثال او الحالة التى تستخدم فى كود PL \ SQL

هيكل index by:

INDEX BY Table Structure



فى الشكل كما نرى يشبه حد كبير جداول قاعدة البيانات
لاحظ ان index by لا يحدد له طول أو بدايه و نهايه ولكن يتم اضافة كل صف داخله عند استخدامه.
من الشكل نرى ان هناك عمود يحتوى على "primary key" والعمود الاخر الذى تحدد نوعه ويوضع به القيم
لاحظ : لايمكن عمل قيم ابتدائية فى تعريف index by
@ مثال على index by :

Creating an INDEX BY Table

```
DECLARE
  TYPE ename_table_type IS TABLE OF
       employees.last_name%TYPE
       INDEX BY BINARY_INTEGER;
  TYPE hiredate_table_type IS TABLE OF DATE
       INDEX BY BINARY_INTEGER;
  ename_table       ename_table_type;
  hiredate_table    hiredate_table_type;
BEGIN
  ename_table(1)     := 'CAMERON';
  hiredate_table(8)  := SYSDATE + 7;
    IF ename_table.EXISTS(1) THEN
      INSERT INTO ...
    ...
END;
/
```

فى المثال عمل اثنين من index by واحد يحتوى على اسماء الموظفين ى الاخر يحتوى على تاريخ التعيين 0
من المثال يتم ادخال اسم "Cammron" فى الصف رقم (1) فى index by الاول و يتم ادخال تاريخ اسبوع من الان فى الصف رقم (8) فى (index by) الاخر
- من خلال المثال يتم استخدام :- index by
كتابة اسم index by ثم كتابة رقم الصف داخل قوسين ( ) بعد اسم index by
وهكذا يتم التعامل مع كل خلية (صف) داخل index by
اسم index by (رقم الصف) : = قيمة ;

<u>خصائص index by</u> : -

- exist : - وهى لمعرفة هل هذه الخلية (صف) موجود فى index by
- count : - ياتى بعدد الخلايا داخل index by التى تحتوى على قيم
- first : - تاتى برقم (primary key) الخاص باول خلية (صف) فى index by
- last : - ياتى برقم اخر خلية فى index by
- prior (n) : - يرجع (index number) بعدد خلايا n.
- next (n): تاتى (index number) بعدد الخليه الحاليه بعدد خلايا (n)
- term (n): يمسح عدد خلايا (n) من نهاية index by
- delete (m,n): يمسح الخلايا من النطاق m الى n فى index by

<u>مثال على index by مع السجل " record "</u>

يتم فى المثال عمل الخلايا من النوع سجل يحتوى على صف كامل للبيانات من الجدول employees ويتم ادخال الصفوف كما نرى فى شكل Loop.

```
DECLARE
  TYPE dept_table_type IS TABLE OF
       departments%ROWTYPE
       INDEX BY BINARY_INTEGER;
  dept_table dept_table_type;
  -- Each element of dept_table is a record
```

مثال اخر على استخدام الخواص :

```
BEGIN
e:=&n1;
for i in 0..e-1 loop
k(i):=ln(i+1)*tan(i);
end loop;
for z in 0..e-1 loop
dbms_output.put_line(k(z));
end loop;
dbms_output.put_line('----------------------');
dbms_output.put_line('last item '||k.last);
dbms_output.put_line('count of array '||k.count);
k.delete(5);
dbms_output.put_line(k.count);
k.delete(3,7);
dbms_output.put_line(k.count);
dbms_output.put_line('----------------------');
for i1 in 0..k.last loop
if k.exists(i1) then
dbms_output.put_line(k(i1));
end if;
end loop;
dbms_output.put_line('----------------------');
k.delete;
dbms_output.put_line(k.count);

END;
```

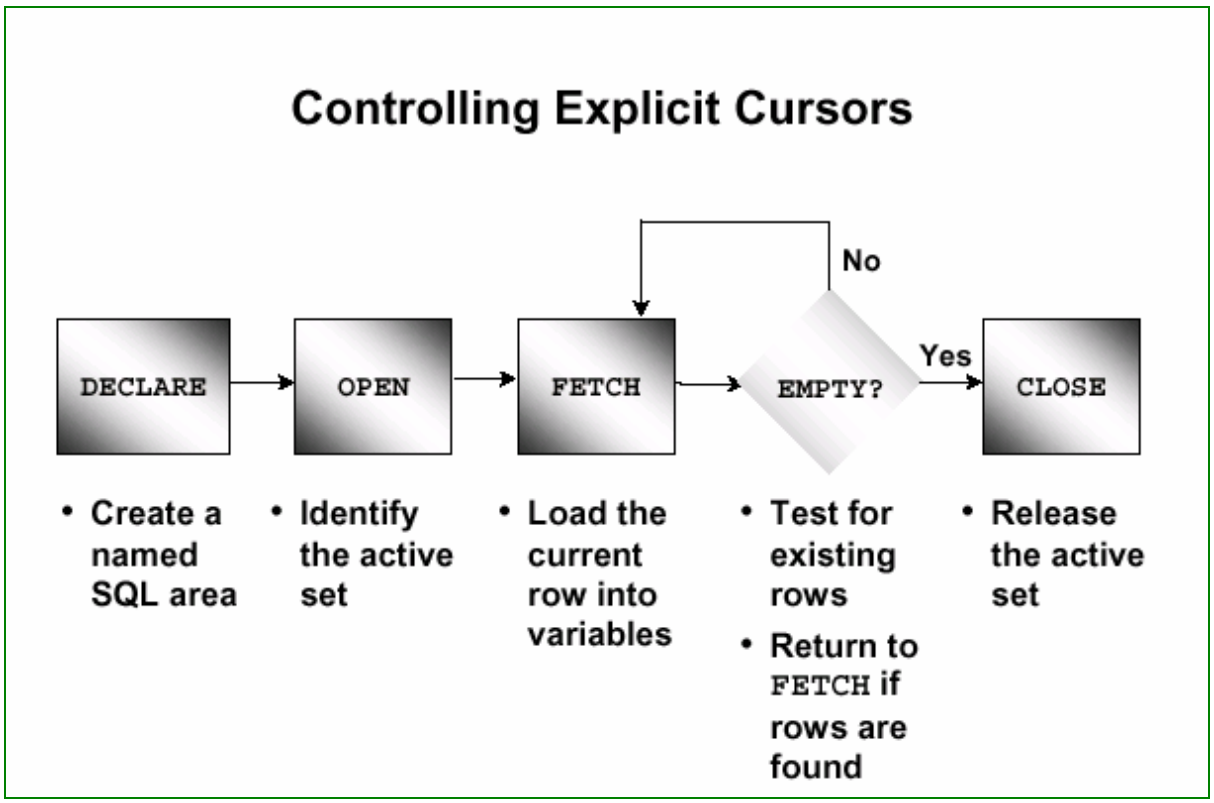
الفصل السادس

التعامل مع cursors

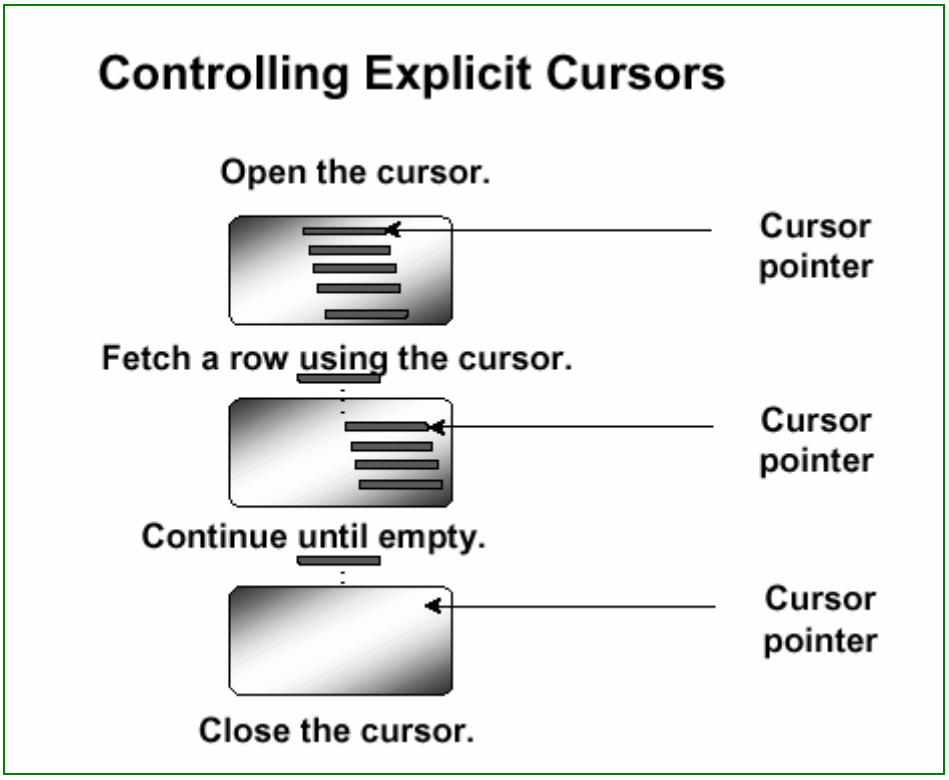
ما هو "cursors"
هو عباره عن مكان فى RAM او الذاكرة يتم حجزه لعمل جمله او امر فى لغة SQL.
انواع cursors :-

1 - صريح explicit
2 - ضمنى implicit

مراحل عمل cursor من النوع الصريح :-



من الشكل يتضح الآتى :-
يمر cursor بعدة مراحل
1 - مرحلة التعريف :
يتم خلالها تعريف cursor وماذا سيحتوى واسمه
2 - مرحلة الفتح :
ويتم فيها فتح cursor وتنفيذ جملة الاستعلام"query" والمتغيرات " bind"
3 - مرحلة احضار البيانات : ويتم فيها احضار البيانات
4 - مراحل غلق cursor
ويتم فيها غلق cursor ومسحه من الذاكره و لا يمكن التعامل بهذا "cursor " قبل فتحه او بعد غلقه .
الشكل التالى يوضح كيفية عمل "cursor" مع البيانات :



لاحظ ان "cursor" يتعامل مع البيانات صف صف فاذا انتهى من الصف يتم احضار غيره ولايرجع ثانيه لصف تم التعامل معه نفس طريقة عمل "stack".
الصيغة العامة لتعريف "cursor" الصريح :

```
CURSOR cursor_name IS
     select_statement;
```

فى هذا الشكل يتم تعريف "cursor" وذلك فى الجزء التعريفى داخل بلوك pl \ spl
لاحظ :لا تستخدم كلمة into فى جملة "select" الخاصة بالـ "cursor"
يمكن عمل أى شكل لجملة select سواء join \ subquery \----

مثال:

```
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, last_name
    FROM   employees;

  CURSOR dept_cursor IS
    SELECT *
    FROM   departments
    WHERE  location_id = 170;
BEGIN
  ...
```

فى الشكل يتم تعريف أثنين من "cursor" الاول لجملة تستدعى رقم الموظف واسمه \ والاخر يستدعى بيانات الادارات التى رقم مكانها يساوى 170.
لاحظ أن
حتى الان لايمكن التعامل بهذا "cursor" لانه لم يتم فتحه بعد

- **فتح cursor "open cursor"**

الصيغه هى

```
OPEN  cursor_name;
```

هنا يتم فتح "cursor" بعدها يمكن التعامل معه

عملية الفتح تنفذ ما يلى : -

1 - يتم تحديد جزء من الذاكره لهذا " cursor"
2 - تنفيذ جملة select
3 - وضع قيم "bind variables"
4 - يتم تنشيط هذه الصفوف التى فى cursor لتجهيزها للعمل
5 - وضع مؤشر على أول صف لكى يتم التعامل معه مباشرة.

لاحظ ان :

جملة select اذا لم تاتى بصفوف فانها لا تنتج اخطاء مثل جملة select داخل بلوك pl/sql

***عملية احضار البياتات من "fetching data from "cursor**

صيغة احضار البيانات :

```
FETCH cursor_name INTO  [variable1, variable2, ...]
                        | record_name];
```

- **ويتم فيها استرجاع البيانات الحليه من "cursor " الى المتغيرات**
- **يجب مراعاة عدد المتغيرات و طبيعة بيانتهم داخل "cursor"**

والمتغيرات التى سوف تستقبلهم باستخدام جملة ---- into ---- fetch

وتقوم جملة fetch بالاتى :

1 - احضار البيانات من داخل "cursor" الى متغيرات الاخراج
2 - تعديل مؤشر "cursor" ليشير على الصف الجديد (التالى) داخل "cursor"

ونستعمل جمل "Loop" ليتم استرجاع البيانات التى داخل "cursor" والتعامل معها كما فى المثال التالى

```
SET SERVEROUTPUT ON
    DECLARE
     v_empno   employees.employee_id%TYPE;
     v_ename   employees.last_name%TYPE;
     CURSOR    emp_cursor IS
       SELECT employee_id, last_name
       FROM    employees;
   BEGIN
     OPEN emp_cursor;
     FOR i IN 1..10 LOOP
       FETCH emp_cursor INTO v_empno, v_ename;
       DBMS_OUTPUT.PUT_LINE (TO_CHAR(v_empno)
       ||'      '|| v_ename);
     END LOOP;
   END ;
```

فى المثال يتم عمل عشرة لفات كل لفه يتم خلالها احضار البيانات التى فى 'cursor" و عرضها

لاحظ ان :

اذا كان "cursor" يحتوى على اكثر من عشره صفوف فانه سيتم عرض 10 صفوف فقط هم العشر صفوف الأول فقط

اذا كان "cursor" يحتوى على اقل من عشرهة صفوف فانه يعرض الصفوف كلها التى فى cursor ويستمر فى عرض اخر صف حتى يكمل "Loop" الى عشرة لفات

****عملية الغلق closing cursor***

الصيغة :

```
CLOSE    cursor_name;
```

ةهذه العمليه تجعل "cursor" غير نشط ويصبح غير ممكن التعامل معه ىالا اذا تم فتحه مره اخرى واذا تم طلب استرجاع بيانات ملف "cursor" وهو مغلق يظهر خطأ invalid-cursor-exception وطبعا يتم تحرير الذاكره التى تم حجزها لهذا "cursor" وعملية غلق "cursor" تتم فى نهاية البلوك او البرنامج الذى يحتوى هذا "cursor" لكن عملية غلقه يدويا تفيد فى استعمال مصادر الجهاز و النظام بطريقه افضل وهناك حد للمستخدم لفتح عدد من "cursor" ويتم تحديد هذا العدد داخل قاعدة البيانات للمستخدمين وهو يساوى 50 لذلك يجب غلق "cursor" عند عدم الحاجه له ثانية

والمعامل المسئول عن "cursor" التىيتم فتخها فى وقت واحد هو Open cursor parameter

*خصائص (cursor) الصريح :

1 - الخاصيه %is open : لتحدد ما اذا كان (cursor) مفتوح ام لا وهى من النوع Boolean

2 - الخاصيه %not found : من النوع Boolean لتحدد ما اذا كان cursor له صفوف ام لا

3 - الخاصيه %found: من النوع Boolean لتحدد ما اذا كان cursor له صفوف ام لا وهى عكس الخاصيه السابقه

4 -% ROW COUNT :لترجع بعدد الصفوف داخل CURSOR

مثال على استخدام الخاصية % ISOPEN

```
IF NOT emp_cursor%ISOPEN THEN
   OPEN emp_cursor;
END IF;
LOOP
  FETCH emp_cursor...
```

وهذا الكود اذا كان (CURSOR)غير مفتوح يفتحه و اذا كان مفتوح يبدأ على الفور فى استخدامه بدون فتح.

- مثال على استخدام الخاصيه % NOT FOUND

```
LOOP
   FETCH c1 INTO my_ename, my_sal, my_hiredate;
   EXIT WHEN c1%NOTFOUND;
   ...
END LOOP;
```

والمثال يوضح كيفية الخروج من (LOOP) اذا لم يوجد صفوف داخل "CURSOR"

- مثال على استخدام الخاصيه %ROWCOUNT

```
LOOP
   FETCH c1 INTO my_ename, my_deptno;
   IF c1%ROWCOUNT > 10 THEN
      ...
   END IF;
   ...
END LOOP;
```

المثال يوضح استخدام الخاصية فى جملة IF

- **مثال على استخدام CURSOR مع السجل "RECORD"**

```
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, last_name
    FROM   employees;
  emp_record emp_cursor%ROWTYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO emp_record;
    EXIT WHEN emp_cursor%NOTFOUND;
    INSERT INTO temp_list (empid, empname)
    VALUES (emp_record.employee_id, emp_record.last_name);
  END LOOP;
  COMMIT;
  CLOSE emp_cursor;
END;
/
```

استخدام (FOR LOOP) مع CURSOR

- **يمكن استخدام FOR LOOP مع CURSOR**
- **بحيث ان الجملة تفعل الاتى :-**

1- تفتح CURSOR فى بدايتها ويحصل عملية FETCH داخلها ويغلق اوتوماتيكيا مع نهاية جملة FOR LOOP

- **لا يسمح داخلها بتعريف عداد او فتح وغلق CURSOR التى تعمل FOR فيه**
- **يمكن ارسال معاملات للـ CURSOR (يتم شرحه لاحقا)**

مثال على استخدام FOR LOOP مع CURSOR :-

```
DECLARE
  CURSOR emp_cursor IS
    SELECT last_name, department_id
    FROM   employees;
BEGIN
  FOR emp_record IN emp_cursor LOOP
         -- implicit open and implicit fetch occur
    IF emp_record.department_id = 80 THEN
      ...
  END LOOP; -- implicit close occurs
END;
/
```

لاحظ

عدم وجود جمل FETCH \ CLOSE \ OPEN

يستخدم عداد FOR فى استحضار البيانات من داخل CURSOR

امثلة على الفصل

مثال 1 : على استخدام cursor الضمنى:

```
/*applying cursor in begin not in declaration*/
set serveroutput on;
set verify off;
declare

Begin
for i in (select empno,ename,sal+comm total_salary from emp) loop
dbms_output.put_line(i.total_salary);
end loop;
end;
/
```

مثال 2 : استخدام cursor لعرض بيانات الادارات وعدد اللموظفين بكل ادارة.

```
set serveroutput on;
set verify off;
/* using cursor to disply data of given dept and
 no of its employess */
DECLARE
x1 int;
v_count int;
v2 int;
BEGIN

for i in (select * from dept) loop
dbms_output.put_line('Departement id       :'||i.deptno);
dbms_output.put_line('Departement name     :'||i.dname);
dbms_output.put_line('Departement location :'||i.loc);
select count(*) into v_count
from emp
where deptno=i.deptno;
dbms_output.put_line('Departement employess:'||v_count);
dbms_output.put_line('******************************');

end loop;

END;
```

الفصل السابع
التعامل المتقدم مع cursor

- **ادخال المعاملات مع cursor**
  - **الصيغه العامة :**

```
CURSOR cursor_name
  [(parameter_name datatype, ...)]
IS
  select_statement;
```

```
OPEN  cursor_name(parameter_value,.....) ;
```

توضح الصيغة كيفية تعريف معامل لل cursor وكيفية فتح (cursor) وتستخدم المعاملات

- **تجعل (cursor) اكثر مرونه اسهل فى البرمجه بحيث البرنامج يستقبل هذه المعاملات وعلى اساسها يبدأ فى عملية الاختيار وقصر الصفوف داخل (cursor) على الصفوف المرادة فقط**

مثال :

```
DECLARE
  CURSOR emp_cursor
  (p_deptno NUMBER, p_job VARCHAR2) IS
     SELECT employee_id, last_name
     FROM   employees
     WHERE  department_id = p_deptno
     AND    job_id = p_job;
BEGIN
  OPEN emp_cursor (80, 'SA_REP');
  . . .
  CLOSE emp_cursor;
  OPEN emp_cursor (60, 'IT_PROG');
  . . .
END;
```

يوضح المثال كيفية تعريف (cursor) بمعاملات وكيفية مناداته من داخل كود PL \ SQL

- **مثال على استخدام المعاملات فى cursor:**

يقوم المثال بعرض بيانات كل موظفي ادارة معينة من جدول الموظفيين.

```
/*Cursor in varible in defining*/
set serveroutput on;
set verify off;
DECLARE
cursor c1 (x number) is select * from emp
where deptno=x;
BEGIN
for i in c1(10) loop
dbms_output.put_line(i.empno||':'||i.sal);
end  loop;
dbms_output.put_line('***************');
for j in c1(20) loop
dbms_output.put_line(j.empno||':'||j.sal);
end loop;
END;
```

مثال 2: يقوم بادخال رقم الادارة ويقوم بعرض بيانات الادارة وكذلك عدد موظفها:

```
set serveroutput on;
/*using cursor to disply data of given dept and no
of  its employess*/
DECLARE
x1 int;
v_count int;
v2 int;
cursor c1(x dept.deptno%type)
is select * from dept
where deptno=x1;
BEGIN
x1:=&id_of_dept;
select count(*) into v_count
from emp
where deptno=x1;
select count(*)into v2
from dept;
for i in c1(x1) loop
dbms_output.put_line('Departement id       :'||i.deptno);
dbms_output.put_line('Departement name     :'||i.dname);
dbms_output.put_line('Departement location :'||i.loc);
dbms_output.put_line('Departement employess:'||v_count);
dbms_output.put_line('******************************');

end loop;
END;
```

استخدام جملة for update

وتضاف هذة الجمله فى نهايه تعريف cursor وذلك لعمل حجز لصفوف الجدول الذى سيتم فيه التعديل أو الحزف
فاذا قام المستخدم باستخدام هذا cursor فأنه يحدث عملية حجز(Luck)للجدول لحمايتهمن المستخدمين الاخريين.

- الصيغة :

```
SELECT ...
FROM          ...
FOR UPDATE [OF column_reference][NOWAIT];
```

وتضاف كلمة (nowait) للرجوع بخطأ اذا كانت هذه الصفوف lock محجوزة من قبل مستخدم آخر
مثال :

```
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, last_name, department_name
    FROM   employees,departments
    WHERE  employees.department_id =
           departments.department_id
    AND employees.department_id = 80
    FOR UPDATE OF salary NOWAIT;
```

****

where current of استخدام جمله

- الصيغة :

```
WHERE CURRENT OF cursor ;
```

مثال :

The WHERE CURRENT OF Clause

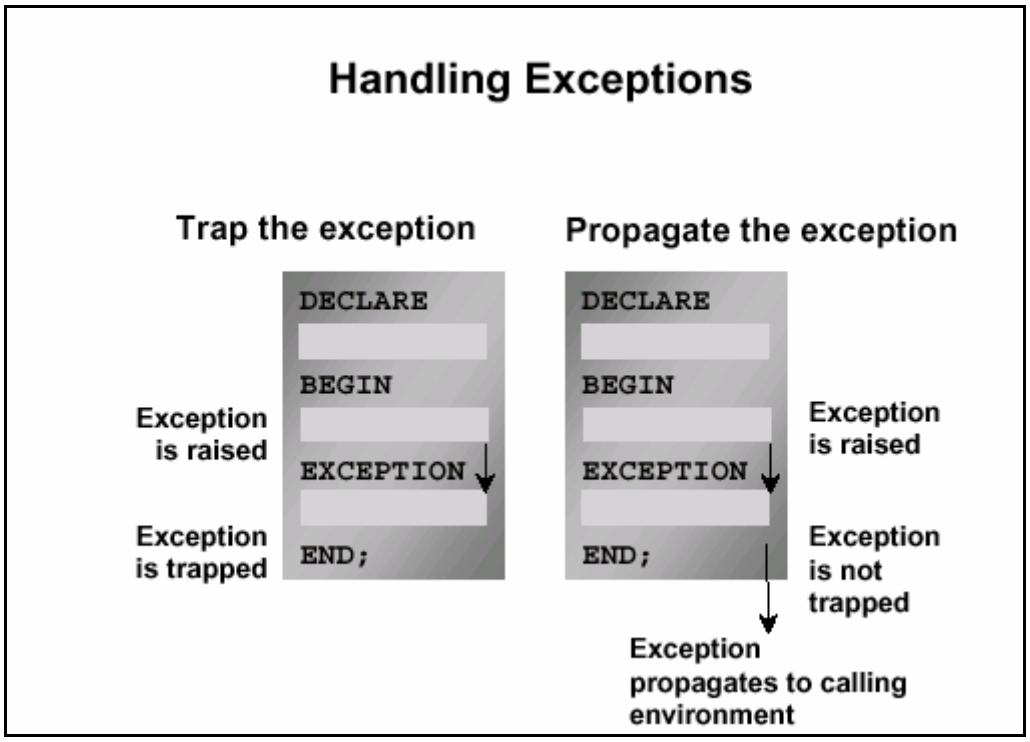
```
DECLARE
CURSOR sal_cursor IS
 SELECT e.department_id, employee_id, last_name, salary
 FROM   employees e, departments d
 WHERE  d.department_id = e.department_id
  and   d.department_id = 60
 FOR UPDATE OF salary NOWAIT;
 BEGIN
 FOR emp_record IN sal_cursor
LOOP
  IF emp_record.salary < 5000 THEN
       UPDATE employees
       SET    salary = emp_record.salary * 1.10
       WHERE CURRENT OF sal_cursor;
  END IF;
END LOOP;
END;
/
```

وتستخدم هذه الجملة مكان جملة where العادية فى update او select او delete ومعناها اذا كان الصف الذى فى cursor هو نفسه الذى يتم العمل عليه فى جمله DML عليه وهى تبسيط بدلا من استخدام شرط مساواة primary key فى cursor و table .
ففى المثال السابق يمكن استبدالها بـ :
where emp_record.employee_id = employee_id

الفصل الثامن

االتعامل مع الاستثناءات والاخطاء
HANDLING EXCEPTION

س : ما هو " EXCEPTION " ؟
ج : هو خطأ داخل SPL \ PL يظهر عن حدوث عملية خطأ او بناء على طلب المستخدم وينتج عنه عدم استكمال البرنامج وخروجه الى بيئه التنفيذ ISQLPLUS
س : كيفية نشوء الاستثناء EXCEPTION ؟
ج : أ - داخلى : بواسطة الاوراكل تقاعده بيانات : مثل عندما جملة SELECT لا ترجع بيانات ينتج خطأ(استثناء)
ب – خارجى : بواسطة المستخدم : مثل ادخال المستخدم لبيانات غير مقبوله وقتها ينتج هذا الخطأ
س : كيفية معالجة الاستثناء EXCEPYION ؟
ج : - استقبال الخطأ (اصطياده) والتعامل معه
- تركه للخروج بالبرنامج الى بيئة التنفيذ SQLPLUS او ISQLPLUS
س : معالجة الاستثناءات " HANDLING EXCEPTION " ؟
ج:



فى الشكل :

- **الجزء الايمن يوضح كيفية نشوء استثناء و لا يوجد له معالجه لذلك يتوقف بلوك (PL / SPL) عند نقطة نشوء الاستثناء يخرج الى بيئة ISQLPLUS**
- **الجزء الايسر : يوضح كيفية نشوء الاستثناء ولكنه يخرج خارج البلوك ويستكمل البرنامج داخل جزء (EXCEPTION) وينفذ معالجة الاستثناء.**

انواع الاستثناءات EXCEPTION TYPES

آ - داخلى بواسطة الاوراكل هو نوعان : -
1 - أخطاء معرفة مسبقا
2 - أخطاء غير معرفة مسبقا
ب - خارجى ويعتمد على تعريف المستخدم (المبرمج) له.

م	نـــــــــوع الخـــــطـأ	شـــــــــرحـــه	طـــــــريقـــة العـــــلاج
-1	اخطأ معرفة مسبقا	هم 20 خطأ معروف حدوثهم داخل PLSQ	لا يتم تعريفهم ويتم نشوْهم تلقائيا
-2	اخطأ من غير معرفة مسبقا	هم باقى الاخطاء القياسية داخل الاوراكل	يتم تعريفهم اولا ونشوْهم تلقائيا
-3	اخطاء معتمدة على تعريف المستخدم	ظرف او حالة يعتبرها المبرمج حالة غير عادية تستوجب توقف البرنامج	يتم تعريفهم اولا ويتم حدوثهم خارجيا بواسطة الكود

- **اصطياد الاخطاء : -**

الصيغة العامة :

Trapping Exceptions

Syntax:

```
EXCEPTION
  WHEN exception1 [OR exception2 . . .] THEN
    statement1;
    statement2;
    . . .
  [WHEN exception3 [OR exception4 . . .] THEN
    statement1;
    statement2;
    . . .]
  [WHEN OTHERS THEN
    statement1;
    statement2;
    . . .]
```

يمكن اصطياد اى اخطاء وذلك عن طريق استقباله يكون مناسب له لمعالجته ويتم ذلك فى جزء الاستثناءات كما فى الشكل السابق .

كل معالجة تبدأ بكلمة (when) والتى تحدد بعدها ما هو الاستثناء الذى يتم تنفيذه عن حدوث هذا الاستثناء.

فى الشكل السابق :

- **Exception : هو الاسم القياسى للاخطاء المعرفه مسبقا أو الاسم الذى تم تعريفه فى الجزء التعريفى .**
- **Statement : وهى الجمل التى ستنفذ بعد حدوث الاستثناء 0**
- **Others : وهو جزء اختيارى وهو يعالج كل الاستثناءات ويتم وضعه فى آخر الاستثناءات لمعالجة اى استثناء لم يتم معالجته .**

المعالجه باستخدام " when others ":

كما عرفنا من قبل عند حدوث اى خطأ او استثناء فان الاوراكل يذهب الى جزء الاستثناءات فاذا وجد له معالجة يخرج الى بيئه تنفيذ البلوك
لذلك تم عمل المعالجه " when others" لتقوم بمعالجة جميع الاخطاء وتوضع بعد كل المعالجات بحيث لم يجد الاوراكل المعالجة المناسبة للاستثناء يذهب اليها كحل اخير لكى لا يتوقف البرنامج ويخرج الى بيئة التنفيذ
لاحظ ان جملة (when others) توضع اخ معالجة فى جزء الاستثناءات فاذا وضعنا بعد اى معالجة ينتج خطأ فى syntax

*** خطوط عامة فى اصطياد الاخطاء :-**
1 - يبدأ قسم الاستثناءات بكلمة EXCEPTION
2 - يمكن معالجة كل الاخطاء
3 - معالجة واحدة فقط تتم قبل الخروج من البلوك
4 - المعالجة WHEN OTHERS هى اخر معالجة فى قسم الاستثناءات .

التعامل مع الاخطاءالمعرفه مسبقا

وهى لها اسماء قياسيه تنادى بها وهم حوالى 20 خطاء فقط مثل

- **No data found :عند عدم رجوع جملة select بأى بيانات**
- **Too-many-rows : عند رجوع جملة select باكثر من صف واستقباله فى متغير منفرد**
- **Invalid-cursor : خطأ فى استخدام " cursor "سواء فتحه وهو مفتوح أو غلقه وهو مغلق**
- **Zero-divide : القسمه على صفر**

@ مثال على استخدام الاخطاء المعرفه مسبقا :-

Predefined Exceptions

Syntax:

```
BEGIN
. . .
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    statement1;
    statement2;

  WHEN TOO_MANY_ROWS THEN
    statement1;
  WHEN OTHERS THEN
    statement1;
    statement2;
    statement3;
END;
```

فى الشكل عند حدوث الخطأ too-many-rows ينفذ الكود الذى بعده واذا حدث الخطأ data-not-found ينفذ الكود الذى بعده ايضا.
لاحظ ان الخطأين لا يمكن ان يحدثوا معا . لماذا ؟

لاحظ ان هذه الاخطأء لا يتم تعريفها فهى معرفه مسبقا.

الأخطاء غير معروفة مسبقاً nonpredefined_error

كل خطأ أو استثناء فى أوراكل له رقم (لاحظ أنه سالب) يعرف بهذا الرقم لذلك نستخد م هذه الارقام لمعرفة الأخطاء الغير معروفه مسبقاً .

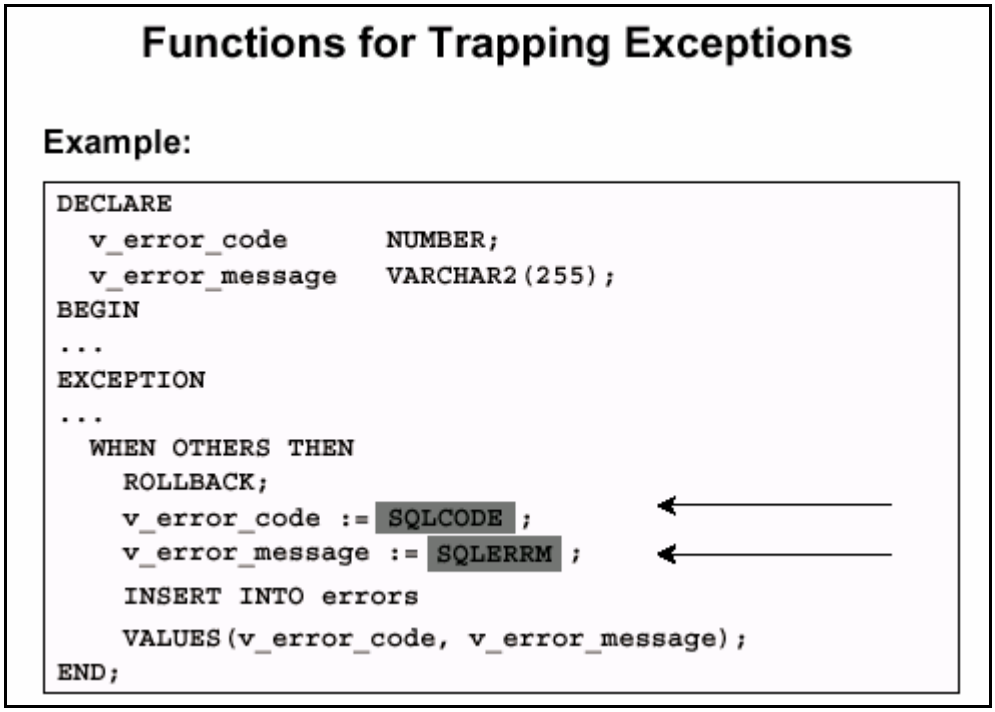
```
DEFINE p_deptno = 10
DECLARE
  e_emps_remaining EXCEPTION;                          (1)
  PRAGMA EXCEPTION_INIT                                (2)
  (e_emps_remaining, -2292);
BEGIN
  DELETE FROM departments
  WHERE  department_id = &p_deptno;
  COMMIT;
EXCEPTION
  WHEN e_emps_remaining  THEN                          (3)
   DBMS_OUTPUT.PUT_LINE ('Cannot remove dept ' ||
   TO_CHAR(&p_deptno) || '.  Employees exist. ');
END;
```

لكى يتم معالجة هذه الاخطاء نقوم بالآتى :-
أولاً : تعريف متغير من النوع EXCEPTION فى الجزء التعريفى .
ثانيا : يتم استخدام جملة PRAGMA EXCEPTION_INIT لوضع رقم الإستثناء للمتغير الذى تم تعريفة فى أولاً كما فى المثال .
ثالثا : يتم معالجة الإستثناء فى جزء الإستثناءات بإستخدام اسم المتغير فى جملة WNEN-TNEN كال فى المثال عند حدوث هنا الخطأ يتم استدراجه ومعالجة بإستخدام المتغير الذى أخذ رقم الخطأ كما فى المثال .
لاحظ أن : الخطأ هذا غير معرف مسبقاً لكنه ينشأ تلقائيا عند حدوثه ويعرفه الأوراكل .
المثال التالى يوضح كيفية استخدام جملة PRAGMA EXCEPTION_INIT :

```
/*PL to discrip the exception , using <pragma> */
 set serveroutput on;
 set verify off;
DECLARE
 x exception;/*create a variable of type exception*/
 pragma exception_init(x,-2292);/*to retreive name of error in x*/
BEGIN
 delete from dept where deptno =10;

EXCEPTION

 when x then /*using x as name for error*/
 dbms_output.put_line('this record is master has detail records');
  /*to get number of error*/

END;
```

الدوال لإصطياد ومعرفة الأستثناءات :

1- الدالة (SQLCODE) :
وهى دالة ترجع برقم الخطأ أو الأستثناء .
2- الدالة (SQLERRM) :
وهى دالة ترجع بالرسالة النصية التى تشرح متى يحدث الخطأ وبعض المعلومات عنه .
المثال الثانى يوضح كيفية استخدام هذه الدوال لمعرفة رق مالخطأ والرسالة الخاصة بها .



المثال التالى يوضح استخدام هذه الدوال:

```
/*PL to discrip the exception , using <when others> */
set serveroutput on;
set verify off;
DECLARE
 x int ;
 y int;
 v_sal int;
BEGIN
 delete from dept where deptno =10;

EXCEPTION

 when others then
 dbms_output.put_line(sqlcode);/*to get number of error*/

END;
```

USER DEPINED EXCEPTIONS **الأخطاء المعرف من المستخدم**

وهى الأخطاء التى يتم تعريفها ونشؤها من خلال المستخدم (المبرمج) وذلك من خلال :

أولا : تعريف متغير من النوع EXCPTION فى الجزء التعريفى .

```
DECLARE
EXCEPTION_NAME      EXCEPTION;
BEGIN
```

ثانياً : نستخدم جملة RAISE لإنتاج (نشوء) خطأ أو استثناء يدوياً وذلك فى الجزء التنفيذى (لاحظ أن هذا الخطأ تم تعريفة مسبقاً فى الجزء التعريفى) .

```
BEGIN
RAISE       EXCEPTION_NAME;
END;
```

ثالثاً : عمل معالجة واستدراج له فى الجزء الأستثناءات .

```
EXCEPTION
WHEN        EXCEPTION_NAME      THEN
            CODES
END;
```

مثال :

```
DEFINE p_department_desc = 'Information Technology '
DEFINE P_department_number = 300

DECLARE
   e_invalid_department EXCEPTION;                          (1)
BEGIN
  UPDATE      departments
  SET         department_name = '&p_department_desc'
  WHERE       department_id = &p_department_number;
  IF SQL%NOTFOUND THEN
    RAISE e_invalid_department;                             (2)
  END IF;
  COMMIT;
EXCEPTION
  WHEN e_invalid_department  THEN                           (3)
    DBMS_OUTPUT.PUT_LINE('No such department id.');
END;
```

المثال يوضح كيفية تعريف وإنشاء ومعالجة خطأ معرف من المستخدم .

الشرح : هذا البلوك يقوم بتعديل وصف الأدارة وذلك من خلال جملة UPDATE ويطلب من المستخدم الوصف الجديد ورقم الإدارة المراد تغير وصفها فإذا أدخل المستخدم رقم لا توجد له إدارة ينشأ استثناء اسمه (W_INVALID_DEPARTEMENT) ويم استدراجه ومعالجته فى الجزء الإستثناءات كما فى المثال .

س : ماذا يحدث إذا لم يتم معالجة الإستثناء أو الخطأ ؟

ج : يخرج البلوك إلى الجهة أو البلوك أو البيئة التى تحتويه فإذا كان موجود فى بيئة ISGLPLUS يخرج لها مباشرة .

فإذا كان موجود داخل بلوك آخر يخرج إلى هذا البلوك الذى يحتويه فإذا كان فى بيئة SGLPLUS يخرج لها مباشرة

استخدام RAISE_APPLICATIONERROR

الصيغة العامة :

```
raise_application_error (error_number,
            message[, {TRUE | FALSE}]);
```

يمكن استخدامها فى الآتى :
- تعريف خطأ أو استثناء معرف من المستخدم وكذلك وضع رسالة الخطأ الخاصة به .
فى صيغة الجملة :
ERROR_NUMBER : المستخدم يضع رقم معين للخكأ الذى يريده ويجب أن يكون من 20000 إلى 20999 .
MESSAGE : وهى رسالة الخطأ المراد إظهاره عند حدوث الخطأ وهى نص أقصى طول له 2048 بايت .
TRUE/FALSE : وهو جزء إختيارى ويحدد أما إذا كان الخطأ يستبدل ما سبقة من خطأ أم لا والقيمة DEFAELT له FALSE أى يستبدل ما سبقة من أخطاء ,
ملاحظات :
- **يمكن استخدامها فى كلا من الجزء التنفيذى وجزء الأستثناءات .**
- **تظهر الخطأ مثل أخطاء أوراكل تماماً .**

مثال :

Executable section:

```
BEGIN
...
  DELETE FROM employees
     WHERE  manager_id = v_mgr;
  IF SQL%NOTFOUND THEN
     RAISE_APPLICATION_ERROR(-20202,
       'This is not a valid manager');
  END IF;
   ...
```

Exception section:

```
...
   EXCEPTION
     WHEN NO_DATA_FOUND THEN
       RAISE_APPLICATION_ERROR (-20201,
         'Manager is not a valid employee.');
   END;
```

فى المثال إذا لم يجد ما يحذفة بجملة DELETE يقوم بإظهار خطأ رقم 20202 ورسالتة هذا مدير خطأ وكذلك فى الجزء الأستثناءات يقوم بوضع رسالة ورقم الخطأ كما فى جملة

RAISE_APPLICATION_ERROR

امثلة على الفصل:
المثال التالى تطبيق على جملة Pragma ودوال اصطياد الاخطاء:

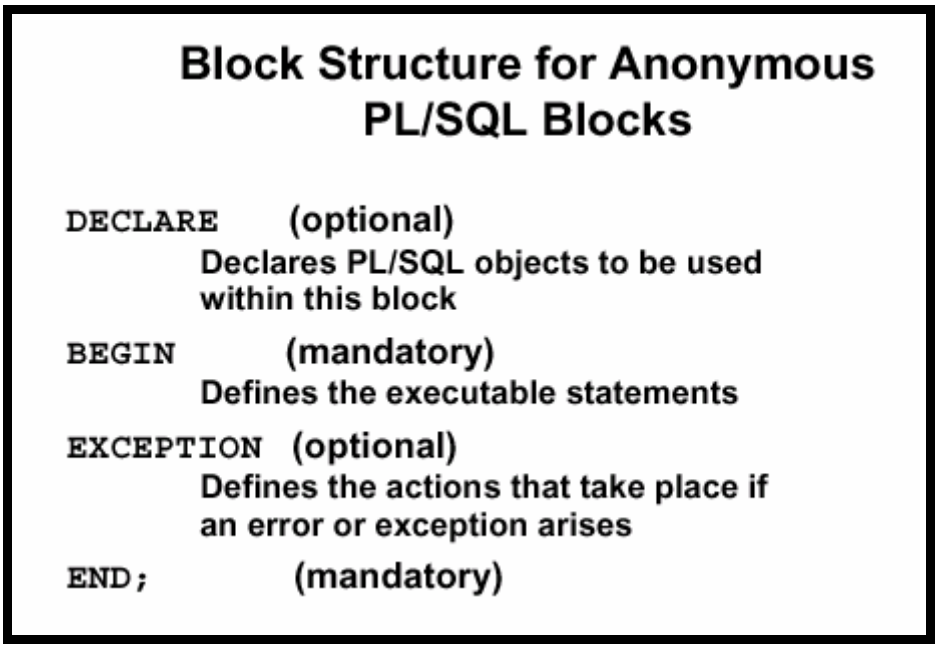
```
/*PL to discrip the exception , using <when others> */
set serveroutput on;
set verify off;
DECLARE
x int ;
y int;
ex exception;
pragma exception_init(ex,-1400);
v_sal int;
BEGIN
x:=&n;
y:=5/x;
dbms_output.put_line(y);
insert into dept
values(null,'law','alex');
select sal into v_sal
from emp;
EXCEPTION

when ex then
dbms_output.put_line('it can not insert null in priamry key ');
/*to get number of error*/
dbms_output.put_line(sqlerrm);/*message of error displyed*/

END;
```

الفصل التاسع

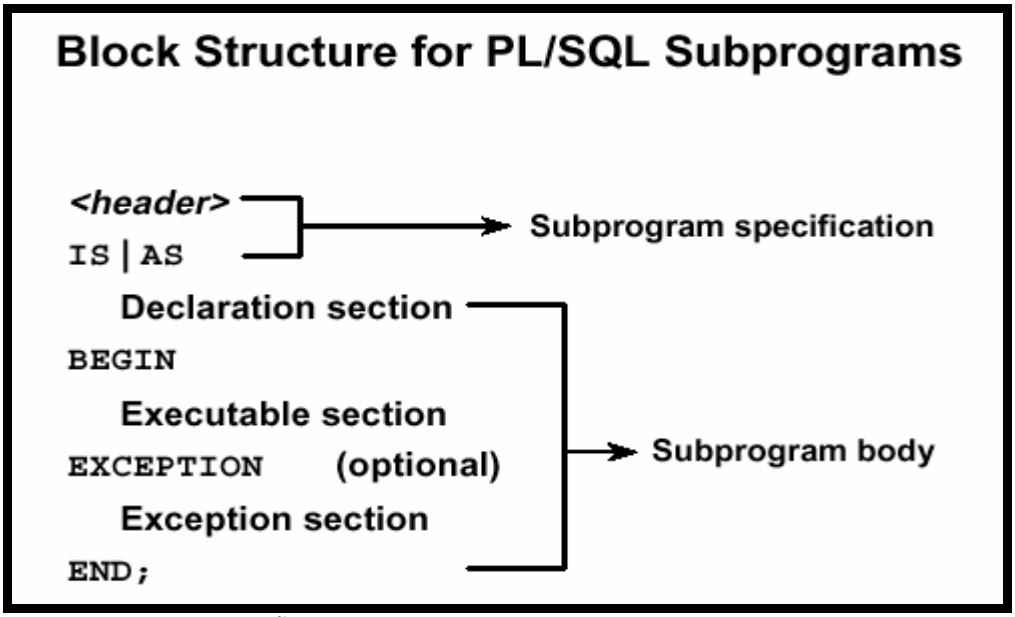
الإجراءات PROCEDVRES

الوحدات البرمجية SUP PROGRAMS

وهى تشمل وحدات أو برامج محفوظة داخل الأوراكل تم إنشأها قبل ذلك مثل الإجراءات PROCEDURES والدوال FUNCTIONS والإجراء هى لعمل مجموعة من العمليات والإجراءات .
والدالة هى لحساب قيمة أو عملية حسابية .
كما رأينا مسبقاً ان البلوك يتكون من عدة أجزاء كما فى الشكل التالى



- **مكونات الاجراء :**



فى SUBPROGRAM يقوم بإستبدال الجزء DECLARE بجزء آخر مكانه هو (HEADER) أو (SPECIFICATION) وهى الرأس او تعريف البرنامج وبقية البلوك تسمى BODY .
يتم فى الجزء Header الأتى :

- **تحديد نوع البرنامج دالة أم اجراء .**

- اسم البرنامج .
- العوامل والمعاملات للربرنامج لووجدت .
- قيمة الرجوع (RETURN) وذلك فى الدوال فقط .
- كلمة AS / IS اساسية وهى تفصل بين تعريف البرنامج وكلمة BEGIN

اما BODY فهوالجزء التنفيذى هو مابين BEGIN-END ويحتوى على الأكواد المراد تنفيذها اما جزء الأستثناءات ويتم فيه استدراج ومعالجة الأخطاء .

س: ما هو الإجراء What Is PROCEDURE :

هو نوع من Subprogram يقوم بتنفيذ عملية معينة ويمكن تخزينه فى قاعدة البيانات كجزء من القاعدة SCHEMA objects لكى يتم استدعائه عند التنفيذ المتكرر.

الصيغة العامة للإجراء :

```
CREATE [OR REPLACE] PROCEDURE procedure_name
 [(parameter1 [mode1] datatype1,
  parameter2 [mode2] datatype2,
  . . .)]
IS|AS
PL/SQL Block;
```

مستخدم كلمة OR REPLACE لإستبدال القديم إذا كان موجوداً قبل ذلك .

- كود PL/SQL يبدأ من كلمة BEGIN أو يتم تعريف متغيرات داخل الإجراء وذلك فى الجزء بين IS/AS وكلمة BEGIN ويتم استخدامها فقط داخل الإجراء قبل كلمة END الخاصة بالإجراء .
- يمكن للإجراء أخذ معامل(Parameter) أو عامل من خارج الأجراء وأنواع المعاملات ثلاثة (IN , OUT , IN OUT)
- عدم تحديد مساحة أو سعة نوع البيانات فى المعاملات يذكر فقط نوع البيانات سواء Number او Varchar2 فقط بدون مساحة .

أنواع المعاملات :

1- النوع IN : وهو النوع الافتراضىDEFAULT للمعامل ويتم من خلال تمرير قيمة من خارج الإجراء إلى الإجراء لأستخدام هذه القيمة . لاحظ لا يمكن تغيير المعامل من النوع IN أى لا يمكن استخدامه كهدف مثل عدا د جملة (FOR-LOOP) تماماً .

2- النوع OUT : ويتم من خلاله تمرير القيم للمتغير من داخل الإجراء إلى خارج الإجراء ولا يأخذ قيمة قبل أستخدام الإجراء ولا يأخذ قيمة قبل استخدام الإجراء وإلا ينتج خطأ ويستخدم فى أرجاع قيمة يحسبها الإجراء داخلياً .

3- النوع IN OUT وهو يمرر القيم من داخل وخارج الإجراء والعكس يمكن أن يأخذ قيمة قبل استخدام الإجراء .

- مقارنه بين الأنواع الثلاثة :

Creating Procedures with Parameters

IN	OUT	IN OUT
Default mode	Must be specified	Must be specified
Value is passed into subprogram	Returned to calling environment	Passed into subprogram; returned to calling environment
Formal parameter acts as a constant	Uninitialized variable	Initialized variable
Actual parameter can be a literal, expression, constant, or initialized variable	Must be a variable	Must be a variable
Can be assigned a default value	Cannot be assigned a default value	Cannot be assigned a default value

مثال على انشاء الاجراء بدون معاملات:

```
rem PL: making first procedure

 create or replace procedure p1
 is
 begin

  dbms_output.put_line('inside procedure p1 ');

 end;
```

والكود التالى يوضح كيفية نداء واستخدام الاجراء داخل كود PL/SQL

```
rem PL: making first procedure
set serveroutput on;
set verify off;

declare
begin
p1;
end;
```

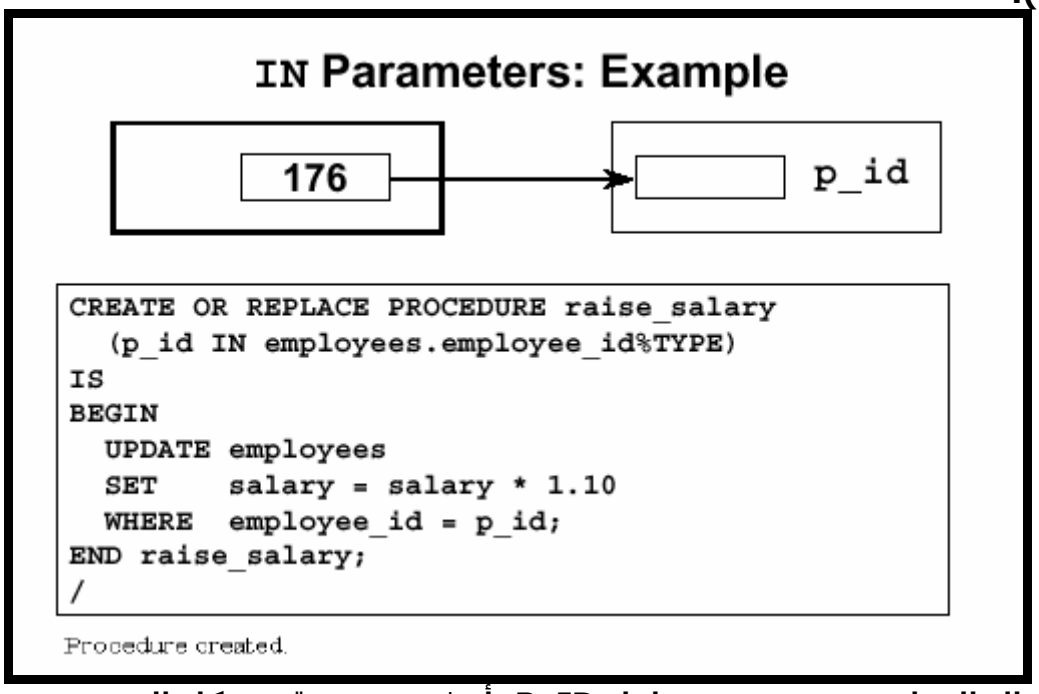
أمثلة : <u>على المعامل من النوع IN</u>

مثال(1):

يقوم بأخذ قيمة من المستخدم فى متغير من النوعIN وعرضها على الشاشة :

```
rem PL: making first procedure
create or replace procedure p1(X in varchar2)
is
begin
dbms_output.put_line(X);

end;
```

مثال(2):



فى المثال السابق يتم تعريف معامل P_ID يأخذ نفس بنية وهيكل العمود Employee_id فى جدول EMPLOYEES من النوع IN .
فى الجزء التنفيذى يقوم بعملية تعديل (UPDATE) فى جدول EMPLOYEES ويضع المرتب (SALAY) بزيادة 10% على المرتب القديم وذلك للموظف رقمه يساوى المعامل P_ID الذى سوف يتم إدخاله عند استخدام هذا الإجراء .

<u>* مثال لإستدعاء الإجراء السابق .</u>

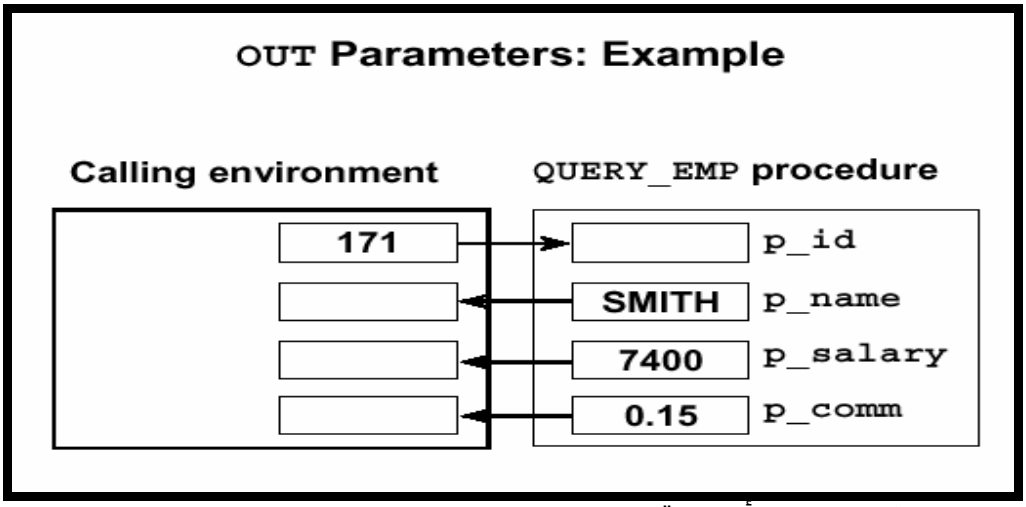
سيقوم بإستدعاء هذ RAISE_SALARY (176) ;

الإجراء ويدخل قيمة P_ID تساوى 176 ليعمل بها هذا الإجراء .

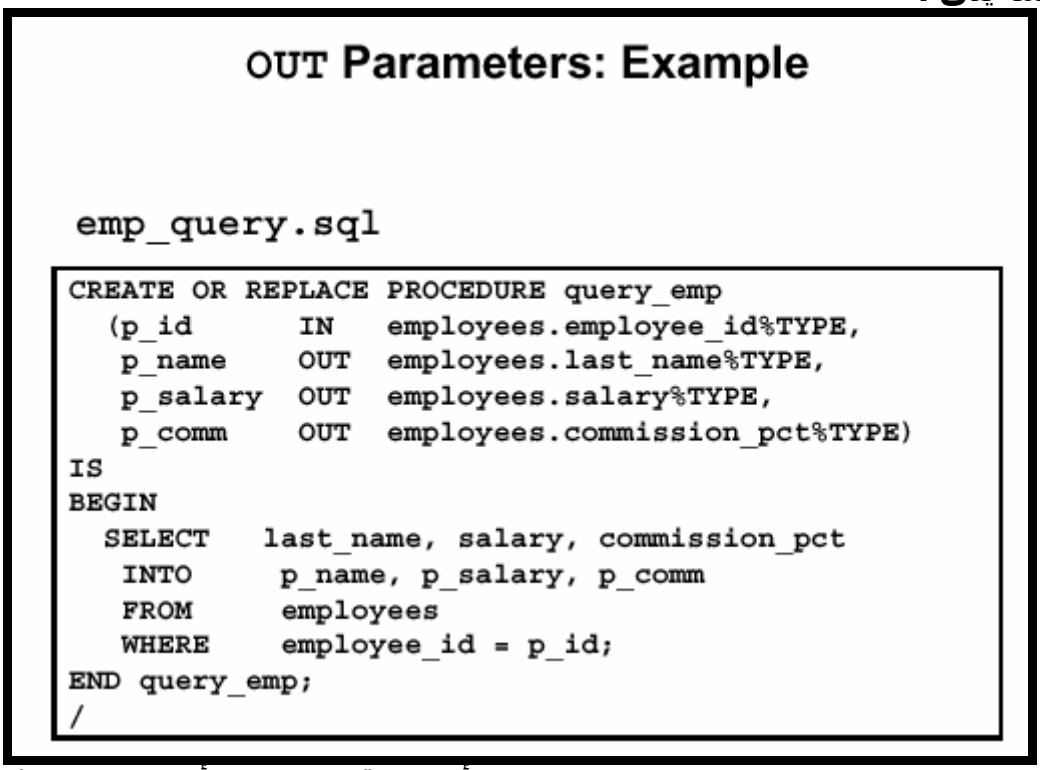
*** يمكن أن نستدعى الإجراء من خارج كود PL/SQL بجملة EXECUTE .**

EXECUTE RAISE_SALARY (176) ;

مثال على المعامل من النوع OUT :



فى المثال والشكل يقوم بأخذ رقم الموظف (معامل من النوع IN) ويرجع باسم الموظف (معامل من النوع OUT) ومرتب الموظف (معامل من النوع OUT) وعمولة الموظف فى (معامل من النوع OUT) .
الكود كما يلى :



لاحظ هنا الكود يستخد المعامل من النوع IN كأنه له قيمة يمكن أن تستخدم فى المقارنة مثلا ولاحظ لا يمكن ان يضع فى المعامل من النوع IN أى قيم .
وذلك عكس المعامل من النوع OUT يستخدم فى استقبال القيم وعرضها وكما فى المثال.

كيفية إستدعاء إجراء يحتوى على معاملات OUT :

داخل كود PL/SQL

- **يتم تعريف متغيرات لها نفس نوع البيانات للمعاملات OUT فى الجزء التعريفى .**
- **ثم يتم استدعاء الإجراء متضمناً هذه المعاملات .**

```
Declare
        V_NAME     VARCHAR2(30) ;
        V_SAL              NUMBER ;
        V_CEMN     NUMBER ;
BEGIN
        QUERY_EMP(171,V_name,V_sal,V_comm);
                DBMS_OUTPUT.PUT_LINE(V_NAME);
                DBMS_OUTPUT.PUT_LINE(V_SAL);
                DBMS_OUTPUT.PUT_LINE(V_COMM);
END;
```

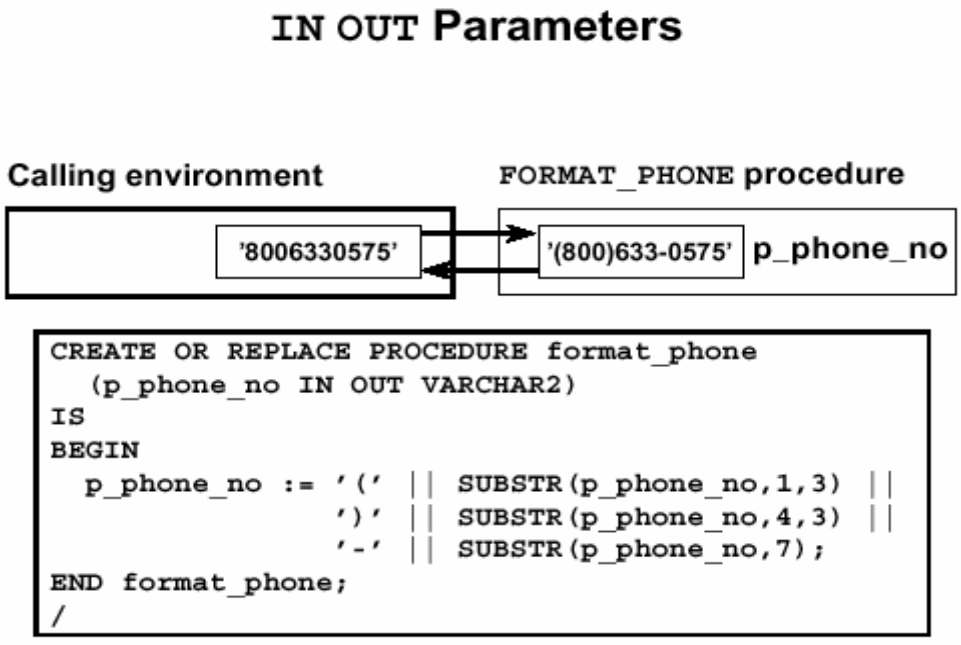
خارج الكود PL/SQL :

- **يتم تعريف متغير من النوع BIND لكل معامل على حدى .**
- **يتم استدعاء الإجراء كما يلى :**

```
VARIALE        G_NAME          VARCHAR2 (30)
VARIALEG_SAL               NUMBER
VARIALEG_COMM    NUMBER
EXECUTE        QUERY_EMP(171,:G_NAME,:G_SAL, :G_COMM);
                PRINT G_NAME;
                PRINT G_SAL
                PRINT G_COMM
```

لاحظ هنا عدم وجود قيم إبتدائية(افتراضية) فى تعريف هذه المتغيرات التى استخدم فى المعاملات OUT .

مثال على استخدام المعامل IN OUT :



ويتم فيه إرسال وإستقبال القيم من داخل وخارج الإجراء فى المثال السابق يتم استقبال القيم للمعامل كأنه معامل IN وبعد العمليات يرجع القيم الجديدة من داخل الإجراء فى نفس المعامل كأنه OUT .

استدعاء إجراء به معامل IN OUT .

Viewing IN OUT Parameters

```
VARIABLE g_phone_no VARCHAR2(15)
BEGIN
  :g_phone_no := '8006330575';
END;
/
PRINT g_phone_no
EXECUTE format_phone (:g_phone_no)
PRINT g_phone_no
```

PL/SQL procedure successfully completed.

G_PHONE_NO
8006330575

PL/SQL procedure successfully completed.

G_PHONE_NO
(800)633-0575

المثال يوضح متغير يتم وضع قيمة لترسل إلى داخل الإجراء كأنه معامل من النوع IN ويتم بعد إستدعاء الإجراء عرض المتغير ثانية ليعرض هذه القيمة الجديدة كأنه معامل من النوع OUT .

<u>طرق تمرير المعاملات</u>

- **حسب الموقع : ويتم عرض وترتيب قيم المعاملات حسب ترتيبها فى جملة إنشاء الإجراء (CREATE).**
- **حسب الأسم : ويتم ذكر اسم المعامل وبعده القيمة التى سوف يأخذها .**
- **الإثنين معاً : عرض بعض القيم حسب الموقع والبعض الآخر حسب الأسم .**

مثال : على إستخدام خاصية DELAUT .

DEFAULT Option for Parameters

```
CREATE OR REPLACE PROCEDURE add_dept
  (p_name  IN departments.department_name%TYPE
                          DEFAULT 'unknown',
   p_loc   IN departments.location_id%TYPE
                          DEFAULT 1700)
IS
BEGIN
  INSERT INTO departments(department_id,
              department_name, location_id)
  VALUES (departments_seq.NEXTVAL, p_name, p_loc);
END add_dept;
/
```

Procedure created.

من الممكن وضع قيمة إبتدائية للمتغيرات من النوع IN بإستخدام DEFAULT أو وضع قيم للمعاملات بالرقم من أنها لها قيم إبتدائية لذلك يمكن إستدعاء الإجراء نفسه بعدة أشكال للمعاملات .

فى المثال السابق يمكن أن ينادى مثل .

```
BEGIN
  add_dept;
  add_dept ('TRAINING', 2500);
  add_dept ( p_loc => 2400, p_name =>'EDUCATION');
  add_dept ( p_loc => 1200) ;
END;
/
SELECT department_id, department_name, location_id
FROM departments;
```

- **فى الطريقة الأولى للإستدعاء يتم نداء الإجراء بدون معاملات لوجود قيم إبتدائية DEFAULT .**
- **فى الطريقة الثانية يتم نداء الإجراء حسب الموقع للمعاملات .**
- **الطريقة الثالثة يستدعى الإجراء حسب الاسم .**
- **الطريقة الرابعة يستدعى الإجراء معاً تجميعة للاسم والموقع معاً .**

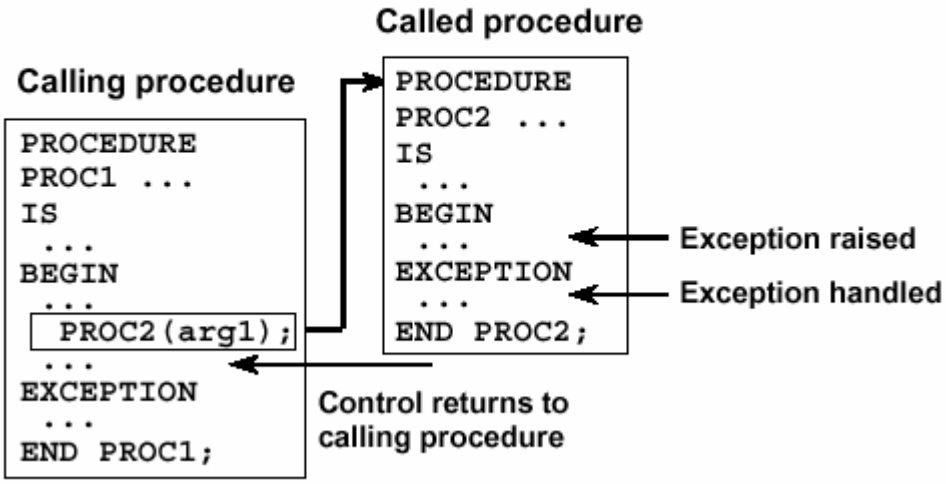
تعريف برنامج داخل إجراء **DECLARE SUSBPROGRAM**

```
CREATE OR REPLACE PROCEDURE leave_emp2
  (p_id  IN  employees.employee_id%TYPE)
IS
  PROCEDURE log_exec
  IS
  BEGIN
    INSERT INTO log_table (user_id, log_date)
    VALUES (USER, SYSDATE);
  END log_exec;
BEGIN
  DELETE FROM employees
  WHERE employee_id = p_id;
  log_exec;
END leave_emp2;
/
```

من المثال يمكن أن يتم عمل إجراء داخلى داخل إجراء ويتم إستدعائه داخل الإجراء الخارجى فقط ففى المثال تم عمل إجراء خارجى leave_emp2 وتم عمل داخل جزء التعريف الخاص به (بين is ، كلمة BEGIN) إجراء داخلى يسمى LOG_EXEC والذى تم إستدعائه داخل الإجراء الخارجى LEAVE_EMP2 كما فى المثال .
لاحظ : لا يمكن إستدعاء الإجراء الداخلى سوى داخل الإجراء الخارجى الذى تم فيه التعريف وكتابة الإجراء الداخلى داخلة لذلك لا يمكن أن تستدعى الإجراء LOG_EXEC فى إجراء آخر أو أى بلوك آخر غير الإجراء LEAVE_EMP2 .
إستدعاء إجراء من إجراء آخر :

```
CREATE OR REPLACE PROCEDURE process_emps
IS
   CURSOR emp_cursor IS
    SELECT employee_id
    FROM   employees;
BEGIN
   FOR emp_rec IN emp_cursor
   LOOP
    raise_salary(emp_rec.employee_id);
   END LOOP;
   COMMIT;
END process_emps;
/
```

فى المثال يتم نداء الإجراء RAISA_SALARY وذلك داخل الإجراء PROCESS_EMPS .
س : ماذا يحدث إذا حدث خطأ أو EXCEPTION فى الإجراء الداخلى ؟
ج : إذا كان له معالجة فى الإجراء الداخلى يتم معالجته والخروج إلى الإجراء الخارجى وإذا كان ليس له معالجة يتوقف الكود عند هذه النقطة فى الإجراء الداخلى ويخرج إلى الإجراء الخارجى وذلك كما فى الشكل التالى .



حذف إجراء :

الصيغة العامة : **DROP PROCEDURE PROCEDUR_NAME;**

مثال : **DROP PROCEDURE RAIS_SALARY;**

امثلة على الاجراءات:

مثال: يقوم الاجراء التالى بعرض اسماء الموظفين فى ادارة من عينة تدخل من المستخدم:

```
create or replace procedure p_dept(X number)
is

cursor c1 is
select ename from emp where deptno=x;

begin
for i in c1 loop
dbms_output.put_line(i.ename);
end loop;
end;
```

استخدام الاجراء السابق:

```
set serveroutput on;
set verify off;
declare
a number;
x exception;
pragma exception_init(x,-6502);
b int;
c int;
begin
a:='&Deptno';
b:='&deptno2';
c:='&deptno3';
p_dept(a);
dbms_output.put_line('************',333333333);
p_dept(nvl(b,0));
dbms_output.put_line('************',333333333);
p_dept(nvl(c,0));
exception
when x then
p1('enter number not charaters');
end;
```

مثال : اجراء يوضح استخدام معاملات من النوع IN والنوع OUT:

```
create or replace procedure
p_s(x number,y out number,z out number,w out number,p out number)
is
begin
select max(sal),min(sal),sum(sal),avg(sal) into y,z,w,p
from emp
where deptno=y
group by deptno;
end;
```

كيفية استدعائه :

```
set serveroutput on;
set verify off;
declare
y number;
z number;
w number;
p number;
q number;

begin
y:=&n;
p_s(y,z,w,p,q);
dbms_output.put_line('Deptno: '||y);
dbms_output.put_line('dept max: '||z);
dbms_output.put_line('dept min: '||w);
dbms_output.put_line('dept sum'||p);
dbms_output.put_line('dept average'||q);
end;
```

الفصل العاشر

إنشاء الدوال CREATING FUNCETION

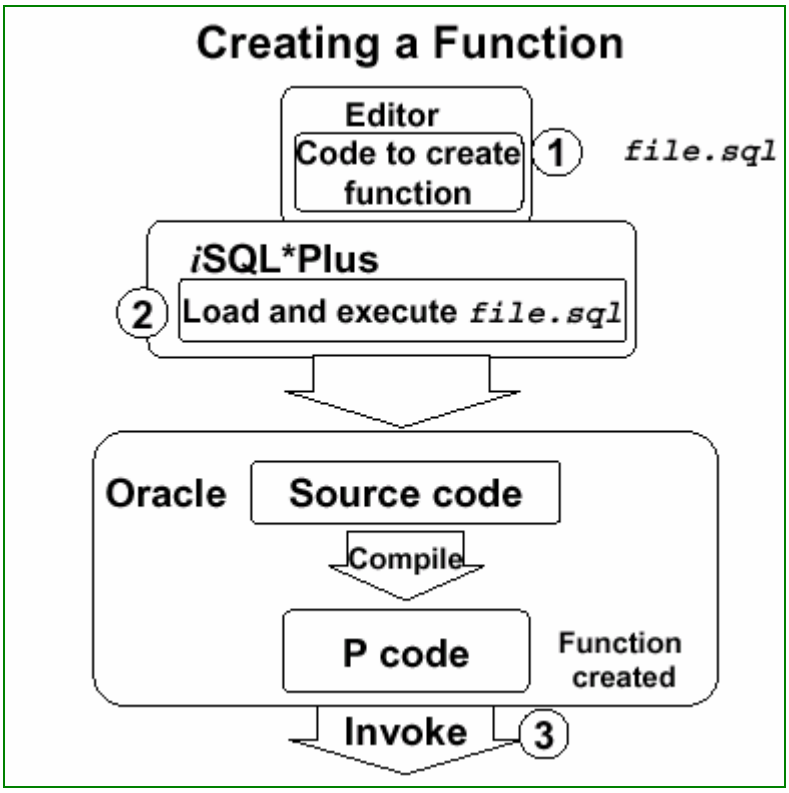
س : ما هى الدوال فى PL/SQL ؟
ج : هى برنامج SUPPROGRAM له اسم ويرجع بقيمة .
- **والدالة يتم حفظها فى قاعدة البيانات مثل (SCHEMA OBJECT) لاستدعائه عند الطلب .**
- **وتطلب كجزء من تعبير أو عملية "expression" .**
- **ويمكن لها أن تأخذ معاملات "parameters" .**

عامة يتم إستخدام الدوال لحساب قيمة وهى قريبة فى الهيكل وطريقة التحرير من كود شكل الإجراء (procedure) ويجب أن ترجع الدالة بقيمة واحدة وبعكس الإجراء ممكن أن يرجع أكثر من قيمة أولا. للدالة جزء رأس "header" وجزء تعريفى وجزء تنفيذى وجزء تنفيذ وهو إختيارى ويتم وضع كلمة return فى جزء الرأس header وكذلك فى الجزء التنفيذى مرة واحدة على الأقل .
والدوال المحفوظة فى قاعدة البيانات تسمى stored function .
يمكن نداء الدوال من جملة SQL وكذلك من PL/SQL بالطبع بعكس الإجراءات التى يتم النداء لها من داخل PL/SQL فقط .
الصيغة العامة لبناء الدوال :

```
CREATE [OR REPLACE] FUNCTION function_name
 [(parameter1 [mode1] datatype1,
  parameter2 [mode2] datatype2,
  . . .)]
RETURN datatype
IS|AS
PL/SQL Block;
```

لاحظ أن كود الدالة يجب أن يحتوى فى الجزء التنفيذى على جملة RETURN واحدة على الأقل من خلال الصيغة نلاحظ أن :
- **جملة CREATE FUNCTION وهى التى تنشأ الدالة .**
- **FUNCTION_NAME اسم الدالة التى سوف يتم به النداء عليها (لاحظ أن هذا الاسم يجب أن يكون UNIQUE) .**
- **PARAMETERS: المعاملات التى سوف ترسل أو تمرر إلى الدالة وهذه الدالة تأخذ [MODE] فقط معاملات من النوع IN فقط ولا يسمح بغير هذا النوع لذا لا تكتب كلمة IN فى تعريف المعامل .**
- **RETURN DATATYPE وهى تحرر ماذا سترجع الدالة من حيث نوع البيانات لاحظ عدم وجود سعة أو مساحة نوع البيانات .**
- **PL/SQL BLOCK : كود PL/SQL الذى سوف يتم تنفيذه .**

*** مراحل إنشاء الدالة :**



الشكل يوضح مراحل إنشاء وعمل الدالة وهى :

1- كتابة الكود أو (SYNTAX)
2- ترجمة COMPILE الكود
3- استخدام الدالة والعمل بها .
4- عملية إرجاع قيمة فى الدالة "RETURN" .

يجب إدخال كلمة RETURN فى الرأس (HEADER) للدالة أثناء كتابتها وكذلك فى الكود فى الجزء التنفيذى .
ويسمح بعدة كلمات من RETURN ولكن عادة مع جملة IF ولكن كلمة RETURN واحدة هى التى تنفذ .

*** إنشاء دالة فى بيئة ISQLPLUS :**

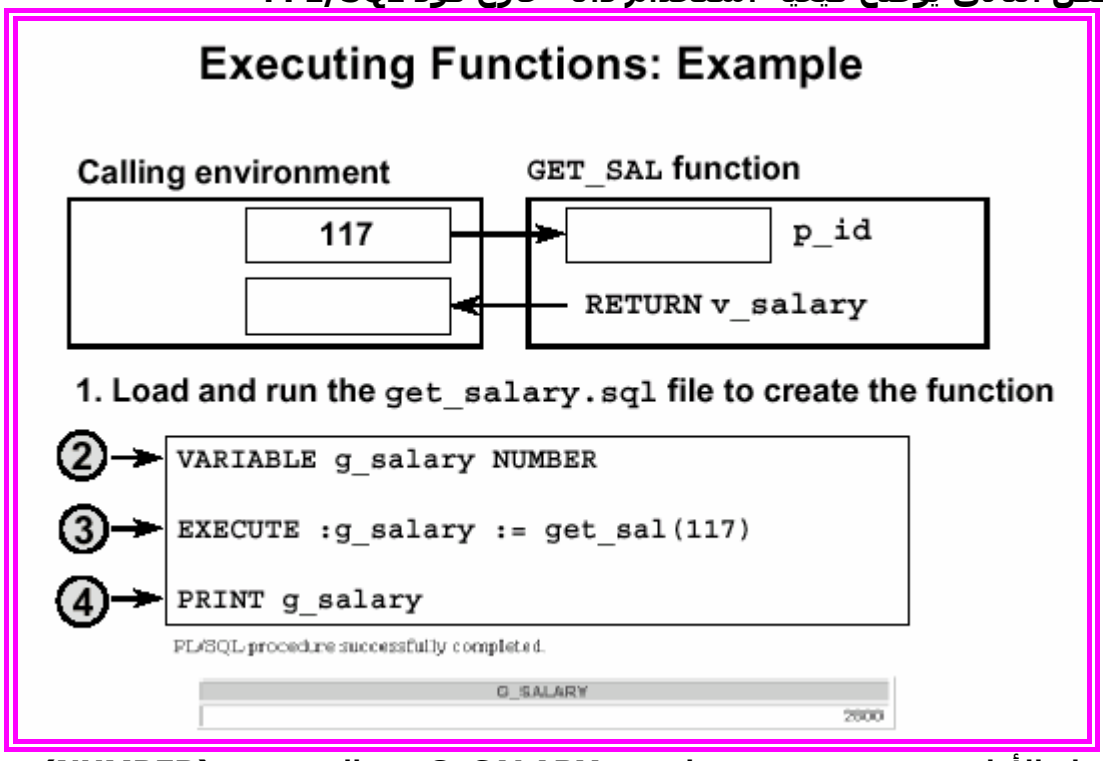
1- كتابة الكود الخاص بالدالة وحفظه كملف SCRIPT .
2- تنفيذ الملف SCRIPT وعملية COMPILE للدالة .
3- إستخدام SHOW ERRER لمعرفة أخطاء Compilation .
4- عند نجاح عملية COMPILATION للدالة يتم إستخدام الدالة .

*** مثال على إنشاء الدالة فى بيئة ISQLPLUS**

```
CREATE OR REPLACE FUNCTION get_sal
       (p_id  IN employees.employee_id%TYPE)
       RETURN NUMBER
    IS
      v_salary employees.salary%TYPE :=0;
    BEGIN
      SELECT salary
      INTO   v_salary
      FROM   employees
      WHERE  employee_id = p_id;
     RETURN v_salary;
END get_sal;
/
```

فى الشكل يقوم بإنشاء دالة اسمها get_sal وتأخذ معامل (parameter) اسمه p_id هو من النوع IN ويأخذ نفس هيكل العمود (EMPLOYEE_id) فى الجدول (EMPBYEE) يلى ذلك كلمة RETURN التى تحدد نوع البيانات التى سوف ترجعها هذه الدالة بين IS ، كلمة BEGIN تعرف بالمتغيرات المحلية للدالة ..
- فى داخل الجزء التنفيذى الذى يبدأ بكلمة BEGIN وينتهى بكلمة END; نرى الآتى ، يختار مرتب الموظف الذى رقمه يساوى P_ID ويضعه فى المتغير V_SALARY ثم يعمل RETURN يرجع هذا المتغير وتنتهى الدالة .
الغرض من الدالة إدخال رقم الموظف تأتى بمرتب هذا الموظف .

* تنفيذ واستدعاء الدالة :
- يجب أن توضع الدالة وترمى فى متغير سواء فى بلوك PL/SQL أو متغير من النوع BIND إذا كانت خارج PL/SQL .
الشكل التالى يوضح كيفية استخدام دالة خارج كود PL/SQL .



السطر الأول : يتم تعريف متغير اسمه G_SALARY من النوع عدد (NUMBER)
السطر الثانى : ينفذ الدالة ويرمى القيمة التى ترجع بها فى المتغير المعرف فى السطر الأول .
السطر الثالث : يطبع قيمة المتغير بإستخدام كلمة PRINT .

* مزايا استخدام الدوال :
1- قلنا سابقاً ان PL/SQL يعتمد على تفتيت المشكلة ثم تحل كل جزئية من المشكلة على حدى وبعدها يتم تجميع هذه الحلول لإنتاج الحل النهائى للمشكلة الأم . ويتم ذلك من خلال الدوال كما رأينا .
2- زيادة معدل فعالية وكفاءة الأكواد بإستخدام الدوال .
3- إمكانية تناول العديد من أنواع البيانات .
4- تبسيط العمليات الحسابية المعقدة .

* كيفية استخدام دالة فى SQL :

Invoking Functions in SQL Expressions: Example

```
CREATE OR REPLACE FUNCTION tax(p_value IN NUMBER)
 RETURN NUMBER IS
BEGIN
   RETURN (p_value * 0.08);
END tax;
/
SELECT employee_id, last_name, salary, tax(salary)
FROM   employees
WHERE  department_id = 100;
```

Function created.

EMPLOYEE_ID	LAST_NAME	SALARY	TAX(SALARY)
108	Greenberg	12000	960
109	Faviet	9000	720
110	Chen	8200	656
111	Sciarra	7700	616
112	Urman	7800	624
113	Popp	6900	552

6 rows selected.

الشكل السابق يوضح كيفية استخدام دالة تم إنشاءها فى PLISQL فى جمل بيئة SQL ويمكن أن تنادى فى SQL فى الأماكن التالية :

1- **فى الأعمدة المرسدة فى جملة SELECT .**
2- **الشرط فى جمل HAVING , WHERE .**
3- **جمل GROUP BY , STARE WITH , ORDER BY .**
4- **جملة VALUES فى جملة INSERT .**
5- **جملة SET فى جملة UPDATE .**

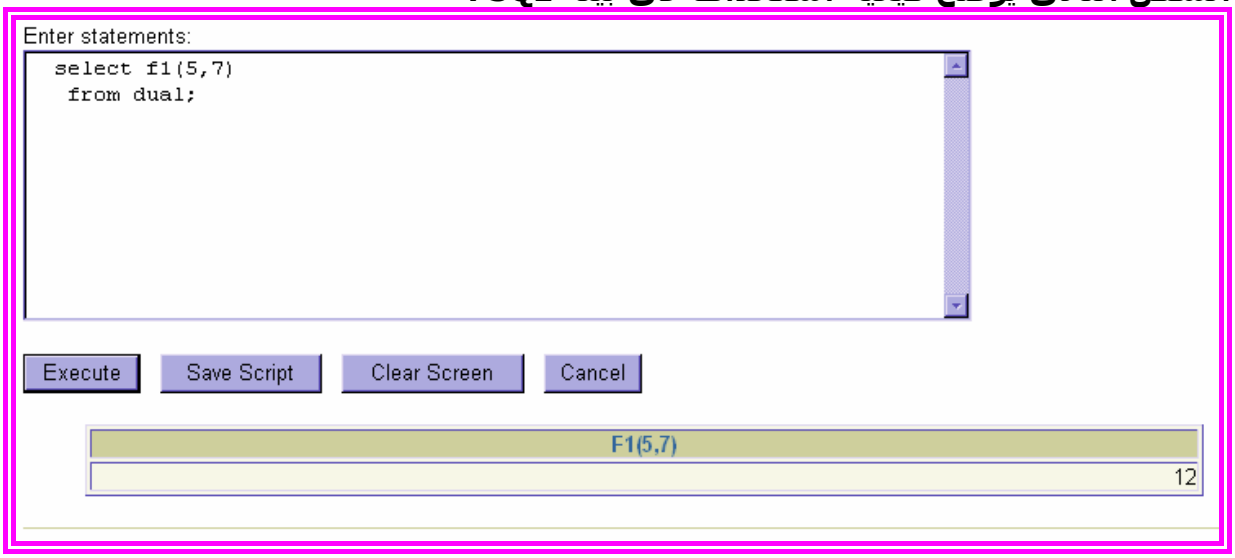
شروط إستخدام الدوال فى تعبيرات SQL :

1- أن تكون دالة تم حفظها فى الأوراكل .
2- تقبل معاملات من النوع IN فقط .
3- تقبل بيانات من أنواع SQL وليست من أنواع بيانات PL/SQL كمعاملات .
4- ترجع بأنواع بيانات الصالحة (الخاصة) لأنواع بيانات SQL وليست أنواع بيانات PL/SQL فهى ترجع بيانات مثل NUMBER , VARCHAR2 , DATE ... وليست مثل RECORD , BOOLEAN , TABEL .
5- لا يجب ألا تحتوى على جمل DML .
6- لا يسمح أن تحتوى الدالة على جمل تحمل إستعلام على نفس الجدول .
7- الدوال التى تنادى من SQL لا يسمح أن تحتوى على جمل تنهى عمليات TRANSACTIONS مثل (COMNIT) or (ROLPBACK) .

مثال1: الشكل التالى يوضح انشاء دالة تقوم باخذ رقمين وترجع بمجموعهما:

```
create or replace function f1( x1 number,x2 number)
return number is
begin
return(x1+x2);
end;
```

الشكل التالى يوضح كيفية استخداها فى بيئة SQL :



مثال2: يوضح دالة من النوع Boolean تقوم بادخال رقم وتختبر هذا الرقم اذاكان اكبر من الصفر ترجع بـ True اذا كان اكبر من الصفر وبـ False اذا كان الرقم غير ذلك.

```
create or replace function f12(x number)
return boolean
is
begin
if x >0 then return true;
else return false;
end if;
end;
/
```

لاحظ ان : لا يمكن استخدامها فى بيئة SQL لأنها ترجع قيمة Boolean وهى نوع موجود فقط فى بيئة PL/SQL.

Restrictions on Calling from SQL

```
CREATE OR REPLACE FUNCTION dml_call_sql (p_sal NUMBER)
   RETURN NUMBER IS
BEGIN
   INSERT INTO employees(employee_id, last_name, email,
                         hire_date, job_id, salary)
        VALUES(1, 'employee 1', 'emp1@company.com',
                  SYSDATE, 'SA_MAN', 1000);
   RETURN (p_sal + 100);
END;
/
```

Function created.

```
UPDATE employees SET salary = dml_call_sql(2000)
 WHERE employee_id = 170;
```

```
UPDATE employees SET salary = dml_call_sql(2000)
                              *
ERROR at line 1:
ORA-04091: table HR.EMPLOYEES is mutating, trigger/function may not see it
ORA-06512: at "HR.DML_CALL_SQL", line 4
ORA-06512: at line 1
```

يشرح دالة تحتوى على DML تستدعى من داخل SQL وكذلك شكل الخطأ المصاحب لها .

*** <u>كيفية حذف دالة</u> :**

الصيغية العامة : DROP FUNCTION FANVTION_NAME;

مثال : DROP FUNCTION GET_SAL;

عند حذف دالة كل المنح والصلاحيات الممنوحة عليها يتم اسقاطها .

الفصل الحادى عشر
إدارة البرامج

منح النظام ومنح الكائنات "SYSTEM & OBJECTPRIVLLEGES"

هناك اكثر من 80 منحة من منح النظام مثل فتحة إنشاء أى جدولCREATE ANY TABLE على سبيل المثال

```
GRANT CREATE ANY TABLE TO GREEN;
```

وهذه المنح والصلاحيات يتم منحها من قبل مستخدم SYSTEM أو SYS ' .

منح الكائنات " OBJECTPRINLEGE " :

وهى الحقوق على إستخدام لكائن معين داخل (SCHEMA) ودائماً يحتوى جملة منح الصلاحية على إسم الكائن .

على سبيل المثال : يمنح المستخدم SCOTT صلاحية تعديل جدول موظفيه EMPLOYEES كلآتى :

```
GRANT ALTER ON EMPLOYEES TO GREEN;
```

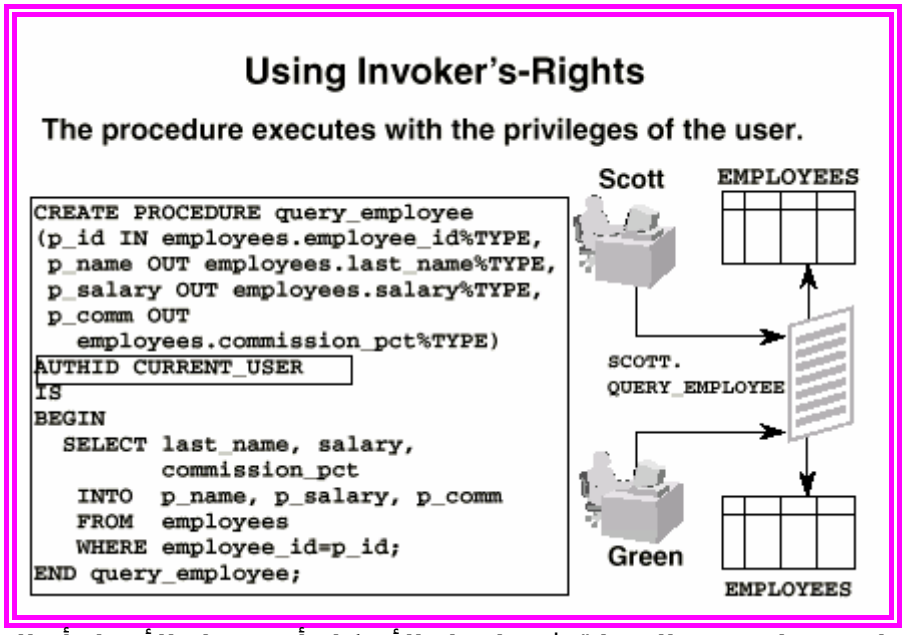
لكى تنشأ دالة أو إجراء أو بلوك PL/SQL يلزم أن يكون لديك صلاحية CREATE PROCEDURE وبراسطة هذه الصلاحية يمكنك ايضاً تعديل (ALTER) أو حذف (DROP) أو تنفيذ هذا الكود (بلوك) .

وإذا كان هذا البلوك أو الكود يعتمد أو يستخدم كائن خارج SCHEMA الخاصة به أو خارج نفس SCHEMA يجب أن يتم منحه الوصول إلى هذا الكائن بصراحة وبدون أن يكون عن طريق ROLL .

- استخدم كلمة ANY تستخدم فى لإنشاء أو تعديل أو حذف أو تنفيذ أى كود ملكك حتى لو خارجSCHEMA الخاصة بك . لاحظ أن كلمة ANY هى إختيارية .
- يجب أن يكون لديك صلاحية EXECUTE ANY أو تكون صاحب هذا الكائن إذا أردت تنفيذ هذا الكائن .
- لاحظ أن : كلمة PROCEDURE للدلاله على PROCEDURE أو FUNCTION أو PACKEGE .

الطريقة المباشرة لمنح الصلاحيات والطريقة غير المباشرة :

- الطريقة المباشرة : وذلك بمنح مستخدم آخر صلاحية استخدام الكائن مثل الجدول مثلاً .
- الطريقة غير المباشر : وذلك بمنح مستخدم آخر صلاحية تنفيذ الكائن فقط مثل الدالة(وهذه الدالة تقوم بمنادة هذا الجدول بذلك هذا المستخدم لديه صلاحية تنفيذ الدالة لكن لا يقدر ان يتعامل مع الجدول مباشرةً) .
- إستخدام جملة AUTHIDCURRENT_USER .



الشكل يوضح استخدام هذه الجملة فى إجبار الأوركل أن يعمل الأجزاء أو الدالة فى حدود الصلاحية الخاصة بالمستخدم وإلا يعمل نهائيا .

*** إستخدام USER_OBJECTS :**

وتستخدم هذه الـVIEW لمعرفة جميع أنواع وبيانات الكائنات فى SCHEMA . مثل

```
SELECT
FROM        USER_OBJECTS;
```

وتحتوى على : OBJECT_NAME; اسم الكائن

- **OBJECT_ID : وهو الرقم التعريفى للكائن داخليا .**
- **OBEJECT_TYPE: نوع الكائن (إجراء ، دالة ، ...) .**
- **CREATED : تاريخ الإنشاء .**
- **LAST_DLL_TIME : آخر تاريخ تم تعديل فى الكائن .**
- **TIMESTAMP : تاريخ ووقت إعادة ترجمة COMPILE للكائن .**
- **STATUS : حالة الكائن صالح للعمل أم معطل .**

يمكن استخدام VIEWS لمعرفة الكائنات مثل DBA_OBJECTS أو ALL_OBJECTS .

*** استخدام USER_SOURCE :**

للحصول علىنص أو كود بناء الكائن فى قاعدة البيانات وذلك للكائنات المحفوظة

نستخدم (VIEW) USER_SOURCE وكذلك ALL_SOURCE وأيضاً DBA_SOURCE .

تحتوى على الآتى :

- **NAME : اسم الكائن**
- **TYPE : نوع الكائن سواء كان إجراء ، دالة ، حزمة ...**
- **LINE : رقم السطر فىبيئة أو نص كود الكائن .**
- **TEXT : النص الخاص بالكود كما كتبه المبرمج .**

*** استخدام : USER_ERRORS**

وتستخدم للحصول على نصوص الأخطاء وتحتوى على :

- **NAME : اسم الكائن .**
- **TYPE : نوع الكائن .**
- **SEQUENCE : رقم تسلسلى أو ترتيب للخطأ .**
- **LINE : رقم السطر الذى به الخطأ .**
- **POSITION : مكان أو موقع الخطأ فى السطر .**
- **TEXT : نص رسالة الخطأ .**

مثال: فى حالة حدوث خطأ فى الكود كالاتى:

```
CREATE OR REPLACE PROCEDURE log_execution
IS
BEGIN
INPUT INTO log_table (user_id, log_date)
                                        -- wrong
VALUES (USER, SYSDATE);
END;
/
```

Warning: Procedure created with compilation errors.

يمكن معرفة الخطأ كالاتى:

```
SELECT line || '/' || position POS, text
FROM   user_errors
WHERE  name = 'LOG_EXECUTION'
ORDER BY line;
```

POS	TEXT
4/7	PLS-00103: Encountered the symbol "INTO" when expecting one of the following: := . (@ % ;
5/1	PLS-00103: Encountered the symbol "VALUES" when expecting one of the following: . (, % ; limit The symbol "VALUES" was ignored.
6/1	PLS-00103: Encountered the symbol "END"

استخدام الأمر SHOW ERRORE

الصيغة :

SHOW ERRORS OBJECT_TYPE OBJECT_NAME

مثال :

SHOW ERROR PROCEDURE LOG-EXECATION