آموزش SQL

شامل مطالب

Oracle Database 12c R2: SQL Workshop I

و

Oracle Database 12c R2: SQL Workshop II

به همراه آموزش راه اندازی ابزارهای PL

میلاد خالقی - ۱۳۹۹

فهرست مطالب

مقدمه

زبان SQL یک زبان استاندارد و از نوع رابطه ای است که برای کار با دیتابیس های رابطه ای مورد استفاده قرار می گیرد. منظور از دیتابیس های رابطه ای، دیتابیس هایی هستند که ارتباط بین جدول ها، ساختار و اطلاعات آنها برقرار است مانند دیتابیس SQL SERVER ، اوراکل ،پستگرس و ...

در واقع با استفاده از دستورات SQL می توان عملیات زیر را در دیتابیس مورد نظر انجام داد:

- افزودن ساختار و اطلاعات جدید به دیتابیس
- استخراج و مشاهده داده های دیتابیس به روش های مختلف
- تغییر ویژگی های جدول ها و اطلاعات آنها

در این جزوه، زبان SQL به منظور کار با دیتابیس اوراکل آموزش داده می شود. مطالب فصل های یک الی ده مطابق سرفصل های کتاب آموزشی ORACLE SQL WORKSHOP 1می باشد که در آنها روش های مختلف نمایش اطلاعات دیتابیس و انواع دستورات DML و DDL توضیح داده می شوند. مطالب فصل ۱۲ تا ۱۶ نیز مربوط به کتاب آموزشی ORACLE SQL WORKSHOP 2 است که در آنها روش ساخت و بکارگیری OBJECTهای مهم در دیتابیس اوراکل توضیح داده می شوند.

مثال هایی که در این فصل ها عنوان شده اند بر اساس کاربر یا SCHEMA آموزشی HR است که می توان بعد از نصب نگارش های مختلف نرم افزار اوراکل این SCHEMA را ایجاد نمود که شامل تمام داده ها، جدول ها و CONSTRAINT های استفاده شده در این جزوه می باشد. بنابراین می توان در زمان ایجاد یک دیتابیس با انتخاب گزینه SAMPLE SCHEMA این اطلاعات را به دیتابیس اضافه نمود. لازم به ذکر است بعضی مثال ها و توضیحات این جزوه از اینترنت و منابع معتبر دیگر اضافه شده اند.

در فصل یازدهم، ابزارهای پرکاربرد استفاده از دیتابیس اوراکل و روش اتصال آنها را توضیح می دهیم. این ابزارها به منظور سهولت در کار با دیتابیس اوراکل ارائه شده اند و از زبان های SQL و PL/SQL پشتیبانی می کنند.

زبان برنامه نویسی PL/SQL توسط شرکت اوراکل ارائه شده است که در واقع یک زبان توسعه یافته از دستورات SQL است. در فصل آخر به تفاوت های این دو زبان اشاره می شود.

۱ نمایش اطلاعات دیتابیس با دستور SELECT

دستور SELECT یکی از پرکاربردترین دستورات SQL می باشد که با استفاده از آن اطلاعات دیتابیس اوراکل که در حافظه یا دیسک قرار دارند نمایش داده می شوند. در این فصل انواع روش های استفاده از دستور SELECT برای بازیابی و نمایش اطلاعات از دیتابیس اوراکل را توضیح می دهیم.

همانطور که می دانیم در دیتابیس اوراکل یک جدول از ستون و سطر(فیلد و رکورد) تشکیل شده است. در دستور SELECT ابتدا نام ستون یا ستون هایی از یک جدول که می خواهیم اطلاعات آن را نمایش دهیم را انتخاب می کنیم و سپس بعد از عبارت FROM نام آن جدول را می نویسیم. در خروجی این دستور اطلاعات ستون های انتخاب شده برای سطرهای جدول نمایش می یابند.

مثال: در دستور SELECT زیر ستون های DEPARTMENT_ID و LOCATION_ID از جدول DEPARTMENTS انتخاب شده اند و در خروجی این دستور، اطلاعات مورد نظر نمایش می یابند.

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

	DEPARTMENT_ID	LOCATION_ID
1	10	1700
2	20	1800
3	50	1500
4	60	1400
5	80	2500
6	90	1700
7	110	1700
8	190	1700

نکته: به هر ترتیبی که نام ستون ها را در دستور SELECT انتخاب کنیم، نمایش اطلاعات نیز به همان ترتیب خواهد بود. در مثال قبلی ابتدا اطلاعات ستون DEPARTMENT_ID و سپس اطلاعات ستون LOCATION_ID نمایش داده می شود.

نکته: اگر در دستور SELECT بجای نام ستون ها از علامت * استفاده شود، اطلاعات تمام ستون های جدول در خروجی نمایش داده می شوند.

مثال: تمام اطلاعات (سطرها و ستون ها) جدول DEPARETMENT را در خروجی نمایش دهید.

```
SELECT *
FROM   departments;
```

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700

۱.۱ نکات مهم در نوشتن عبارات SQL

- کلمات، نام جداول و ستون ها در دستورات SQL به بزرگی یا کوچکی حروف حساس نیستند و می توان به دلخواه از حروف بزرگ یا کوچک استفاده نمود.
- می توان عبارات SQL را در یک یا چند خط مجزا نوشت البته به شرطی که یک کلمه کلیدی مانند SELECT ، FROM و... بین خطوط از هم جدا نشود.
- در ابزار SQL DEVELOPER بکارگیری **علامت ;** در انتهای دستورات، اختیاری می باشد مگر اینکه چندین عبارت SQL پشت سر هم نوشته شده باشند. ولی در ابزار SQL*PLUS باید در انتهای هر دستور SQL از **علامت ;** استفاده شود.

۱.۲ استفاده از عبارت های محاسباتی در دستورات SQL

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

می توانیم برای اطلاعاتی که توسط دستور SELECT انتخاب می شوند از عملگرهای محاسباتی استفاده کنیم. بنابراین در خروجی دستور SELECT نتیجه این عملیات ریاضی نمایش می یابد.

نکته: توجه شود که این عملیات فقط مربوط به زمان **نمایش اطلاعات** می باشد و دیتای اصلی جداول را تغییر نمی دهد.

نکته: از این عملگرهای محاسباتی زمانی استفاده می شود که دیتا از نوع عدد یا تاریخ باشد.

نکته: از این عملگرها نمی توان در قسمت FROM عبارت SELECT استفاده نمود.

نکته: اگر نوع داده (DATA TYPE) ستون های جدول از نوع تاریخی (DATE) و یا TIMESTAMP باشد فقط می توان از عملگرهای جمع و تفریق استفاده نمود

مثال: در زمان اجرای دستور SELECT به حقوق تمام کارکنان ۳۰۰ واحد اضافه شود و در کنار حقوق اصلی آنها نمایش یابد. همانطور که مشخص است در نمایش اطلاعات نام ستون جدید نیز برابر با SALARY + 300 است.

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

	LAST_NAME	SALARY	SALARY+300
1	King	24000	24300
2	Kochhar	17000	17300
3	De Haan	17000	17300
4	Hunold	9000	9300
5	Ernst	6000	6300
6	Lorentz	4200	4500
7	Mourgos	5800	6100
8	Rajs	3500	3800
9	Davies	3100	3400
10	Matos	2600	2900

نکته: زمانی که در دستور SELECT از چند عملگر محاسباتی استفاده می شود قواعد ریاضی زیر در تعیین اولویت آنها درنظرگرفته می شوند:

- ضرب و تقسیم اولویت بالاتری نسبت به جمع وتفریق دارند.
- اولویت بیشتر در ترتیب عملگرها از چپ به راست می باشد.
- بکارگیری پرانتز سبب تعریف اولویت بالاتر می شود.

مثال: در دستور اول و در ستون 12*SALARY+100 ابتدا حقوق کارکنان ۱۲ برابر می شود و سپس ۱۰۰ واحد به آن اضافه می گردد. ولی در دستور دوم به دلیل اینکه پرانتز از بالاترین اولویت برخوردار است ابتدا مقدار (SALARY+100) کارکنان محاسبه می شود و سپس ۱۲ برابر می شود.

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

(1)

	LAST_NAME	SALARY	12*SALARY+100
1	King	24000	288100
2	Kochhar	17000	204100
3	De Haan	17000	204100
4	Hunold	9000	108100

...

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

(2)

	LAST_NAME	SALARY	12*(SALARY+100)
1	King	24000	289200
2	Kochhar	17000	205200
3	De Haan	17000	205200
4	Hunold	9000	109200

۱.۳ مفهوم مقدار NULL

NULL عبارت است از مقداری که یکی از شرایط زیر را دارد:

- در دسترس نیست.
- هنوز اختصاص نیافته است.
- غیر مشخص است.
- غیرقابل بکارگیری است.

بنابراین منظور از NULL مقدار ۰ و یا جای خالی نیست. اگر در یک سطر از جدول برای یک ستون خاص مقداری تعریف نشده باشد مقدار آن برابر با NULL می باشد.

مثال: همانطور که در خروجی دستور SELECT زیر مشخص است فقط تعدادی از کارمندان می توانند حق کمیسیون دریافت کنند. بنابراین مقدار ستون حق کمیسیون برای مابقی کارمندان که حق کمیسیون دریافت نکرده اند برابر با NULL می باشد.

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

	LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
1	King	AD_PRES	24000	(null)
2	Kochhar	AD_VP	17000	(null)
3	De Haan	AD_VP	17000	(null)
...				
12	Zlotkey	SA_MAN	10500	0.2
13	Abel	SA_REP	11000	0.3
14	Taylor	SA_REP	8600	0.2
15	Grant	SA_REP	7000	0.15
...				
18	Fay	MK_REP	6000	(null)
19	Higgins	AC_MGR	12008	(null)
20	Gietz	AC_ACCOUNT	8300	(null)

نکته: می توان مقدارهای NULL را با دستور SELECT انتخاب نمود و نمایش داد. همچنین می توان برای آنها از عملگر محاسباتی استفاده کرد. البته هرگونه عملیات ریاضی بر روی مقدار NULL نتیجه ی NULL می دهد.

نکته: اگر در یک ستون از جدول از CONSTRAINT های PRIMARY KEY و NOT NULL استفاده گردد از درج مقدار NULL در آن ستون جلوگیری می شود.

۱.۴ استفاده از نام مستعار یا ALIAS

می توان در زمان نمایش اطلاعات یک جدول توسط دستور SELECT برای ستون ها یک نام مستعار تعریف کرد تا بجای نام اصلی ستون نمایش یابد. تعریف نام مستعار برای ستون ها به روش های زیر انجام می شود:

۱- **استفاده اختیاری از کلمه AS جهت افزایش خوانایی :**

```
SELECT first_name AS NAME FROM contacts;
```

۲- **بدون استفاده از کلمه AS:**

```
SELECT first_name NAME FROM contacts;
```

۳- اگر نام مستعار شامل فاصله باشد یا شامل کاراکترهای خاص(! _ – و ....) باشد باید از علامت " " استفاده شود. البته می توان به دلخواه از این علامت استفاده کرد:

```
SELECT first_name “NAME!= “ FROM contacts;
```

مثال: ترکیبی از روش های قبلی :

```
SELECT first_name AS “NAME” FROM contacts;
```

مثال: در دستور زیر نام ستون در زمان نمایش اطلاعات تغییر یافته است.

```
SELECT last_name AS name, commission_pct comm
FROM   employees;
```

	NAME	COMM
1	King	(null)
2	Kochhar	(null)
3	De Haan	(null)
4	Hunold	(null)

...

```
SELECT last_name "Name" , salary*12 "Annual Salary"
FROM   employees;
```

	Name	Annual Salary
1	King	288000
2	Kochhar	204000
3	De Haan	204000
4	Hunold	108000

نکته: از روش های بالا جهت تعریف نام مستعار برای ستون ها استفاده می شود ولی جهت تعریف نام مستعار برای نام جدول نیز می توان از این روش ها استفاده نمود که این کار در عملیات JOIN و یا SELECTهای خاص پرکاربرد خواهد بود.

مثال: در دستور JOIN زیر برای جدول PRODUCT از نام مستعار P استفاده کنید.

```
SELECT p.product_id, p.product_name,
categories.category_name
FROM products p
INNER JOIN categories
ON p.category_id = categories.category_id

ORDER BY p.product_name ASC, categories.category_name ASC;
```

۱.۵ عملگر الحاق یا CONCATENATION

با استفاده از این عملگر که با **علامت** || تعریف می شود مقدارهای چند ستون که از نوع کاراکتری هستند در زمان نمایش اطلاعات به هم متصل گردند.

مثال: با استفاده از عملگر الحاق، دو ستون LAST_NAME و JOB_ID به هم متصل شوند و با نام مستعار EMPLOYEES نمایش داده شوند.

TABLE EMPLOYEES . . .

	LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
1	King	AD_PRES	24000	(null)
2	Kochhar	AD_VP	17000	(null)
3	De Haan	AD_VP	17000	(null)
12	Zlotkey	SA_MAN	10500	0.2
13	Abel	SA_REP	11000	0.3
14	Taylor	SA_REP	8600	0.2
15	Grant	SA_REP	7000	0.15

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

	Employees
1	AbelSA_REP
2	DaviesST_CLERK
3	De HaanAD_VP
4	ErnstIT_PROG
5	FayMK_REP
6	GietzAC_ACCOUNT
7	GrantSA_REP
8	HartsteinMK_MAN

نکته: اگر یک ستون که دارای مقدار NULL می باشد با ستون دیگر الحاق شود نتیجه برابر با مقدار ستون دیگر خواهد بود.

۱.۶ رشته های LITERAL

عبارت LITERAL به صورت مجموعه ای از کاراکتر، عدد یا تاریخ تعریف می شود. با استفاده از LITERAL یک عبارت خاص برای **نمایش در هر سطر** از جدول تعیین می شود.

استفاده از LITERAL همانند این می باشد که یک ستون دلخواه در زمان اجرای دستور SELECT به جدول اضافه شود. اگر عبارت تعریف شده شامل کاراکتر یا تاریخ بود باید از علامت ‘ ’ استفاده گردد ولی برای اعداد نیاز به استفاده از علامت ‘ ’ نمی باشد.

مثال: همزمان از الحاق، نام مستعار و LITERAL استفاده شده است:

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

	Employee Details
1	Abel is a SA_REP
2	Davies is a ST_CLERK
3	De Haan is a AD_VP
4	Ernst is a IT_PROG
5	Fay is a MK_REP
6	Gietz is a AC_ACCOUNT
7	Grant is a SA_REP
8	Hartstein is a MK_MAN
9	Higgins is a AC_MGR
10	Hunold is a IT_PROG
11	King is a AD_PRES

...

نکته: اگر داخل عبارت LITERAL علامت **'** بکار رود به جهت اینکه این علامت با **'** شروع یا پایان آن عبارت اشتباه گرفته نشود بجای LITERAL از یک روش دیگر به نام ALTERNATE QUOTE استفاده می گردد.

عبارت ALTERNATE QUOTE با علامت **q'** تعریف می گردد. همچنین می توان بجای علامت **[ ] از علامت < >، ( ) یا { }** استفاده نمود.

مثال:

```
SELECT department_name || q'[ Department's Manager Id: ]'
       || manager_id
       AS "Department and Manager"
FROM departments;
```

	Department and Manager
1	Administration Department's Manager Id: 200
2	Marketing Department's Manager Id: 201
3	Shipping Department's Manager Id: 124
4	IT Department's Manager Id: 103
5	Sales Department's Manager Id: 149
6	Executive Department's Manager Id: 100
7	Accounting Department's Manager Id: 205
8	Contracting Department's Manager Id:

۱.۷ عدم نمایش سطرهای تکراری با عبارت DISTINCT

در حالت عادی وقتی از دستور SELECT استفاده می شود محتویات تمامی سطرها نمایش داده خواهد شد حتی اگر مقادیر در سطرهای مختلف، تکراری باشند. اگر بخواهیم فقط سطرهایی از جدول که براساس محتویات یک ستون، تکراری نیستند را نمایش دهیم می توانیم از عبارت DISTINCT برای آن ستون استفاده کنیم.

مثال: در جدول زیر (جدول PERSONS) فقط نام تمام شهرها به صورت غیر تکراری را می خواهیم:

City	Address	FirstName	LastName	P_Id
Sandnes	Timoteivn 10	Ola	Hansen	1
Sandnes	Borgvn 23	Tove	Svendson	2
Stavanger	Storgt 20	Kari	Pettersen	3

دستور زیر را اجرا می کنیم تا مقدارهای غیر تکراری ستون CITY را نمایش دهیم:

```
SELECT DISTINCT City FROM Persons;
```

بنابراین مقدارهای زیر نمایش داده می شود:

City
Sandnes
Stavanger

نکته: می توان از عبارت DISTINCT برای مجموعه ای از ستون ها استفاده کرد که در این حالت فقط سطرهایی که به صورت غیر تکراری ترکیبی از آن ستون ها را دارند نمایش داده می شوند.

مثال:

```
SELECT DISTINCT product_id, quantity FROM ORDER_ITEMS;
```

در این دستور تمام سطرهایی که ستون های PRODUCT_ID و QUANTITY آنها مقداری متفاوت و جدید از مابقی سطرها می باشند نمایش می یابد.

PRODUCT_ID	QUANTITY
1	43
1	57
1	95
1	127
1	135
2	65
2	99
3	46
3	101
4	82

۱.۸ مشاهده ساختار یک جدول

از دستور DESCRIBE یا DESC جهت نمایش ساختار جدول ها استفاده می شود. این دستور نام تمام ستون های جدول، نوع داده (DATA TYPE) آنها و امکان قرارگیری مقدار NULL در هر ستون را نمایش می دهد.

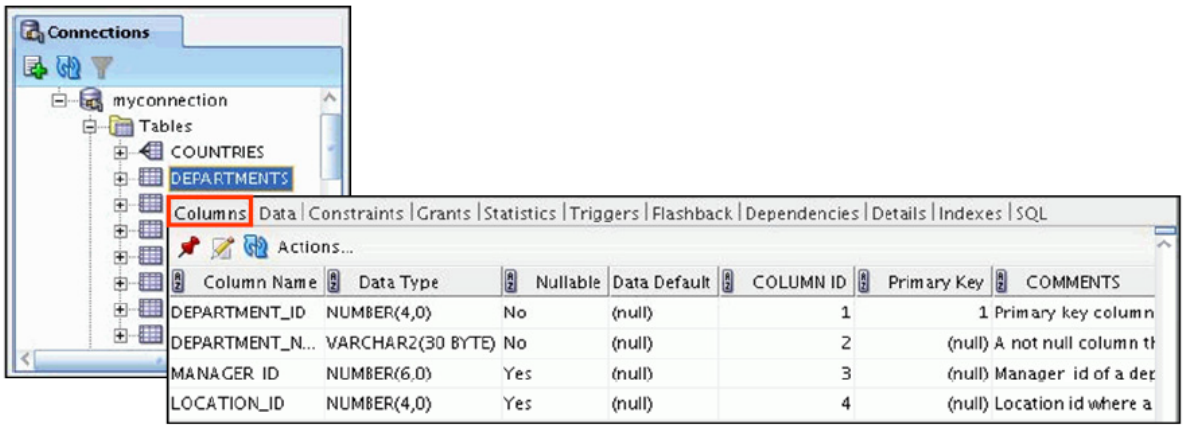
مثال: جدول EMPLOYEES دارای یازده ستون می باشد که پنج ستون NOT NULL هستند. نوع داده هر ستون نیز در قسمت Type مشخص است.

```
DESCRIBE employees
```

```
DESCRIBE Employees
Name           Null     Type
-------------- -------- ------------
EMPLOYEE_ID    NOT NULL NUMBER(6)
FIRST_NAME              VARCHAR2(20)
LAST_NAME      NOT NULL VARCHAR2(25)
EMAIL          NOT NULL VARCHAR2(25)
PHONE_NUMBER            VARCHAR2(20)
HIRE_DATE      NOT NULL DATE
JOB_ID         NOT NULL VARCHAR2(10)
SALARY                  NUMBER(8,2)
COMMISSION_PCT          NUMBER(2,2)
MANAGER_ID              NUMBER(6)
DEPARTMENT_ID           NUMBER(4)
```

در این دستور ستون NULL مشخص می کند که آیا یک ستون خاص از جدول می تواند مقدار NULL داشته باشد یا نه. اگر آن ستون از نوع NOT NULL باشد یعنی در زمان تعریف آن جدول از یک شرط خاص استفاده شده است تا مقدار NULL در آن ستون درج نشود

نکته: دستور DESC فقط در ابزار SQL*PLUS اجرا می شود و در ابزارهای دیگر باید از نمایش گرافیکی درختی ستون ها استفاده نمود.



۲ محدودسازی اطلاعات با کمک عبارت WHERE

زمانی که می خواهیم فقط تعداد مشخصی از سطرهای یک جدول را انتخاب کنیم باید در دستور SQL از عبارت WHERE استفاده کنیم. عبارت WHERE به منظور محدودسازی در زمان نمایش یا ویرایش اطلاعات استفاده می شود بنابراین فقط بخش هایی از اطلاعات که نیاز داریم را انتخاب می کنیم. محل قرارگیری عبارت WHERE در دستور SELECT بعد از عبارت FROM است. این عبارت به ترتیب از سه قسمت زیر تشکیل می شود:

- نام ستون
- شرط مقایسه ای
- نام یک ستون یا مقداری ثابت یا لیستی از مقادیر

مثال: اطلاعات کارمندانی را که در دپارتمان شماره ۹۰ مشغول به کار هستند را نمایش دهید.

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90
4	103	Hunold	IT_PROG	60
5	104	Ernst	IT_PROG	60
6	107	Lorentz	IT_PROG	60

...

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

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90

نکته: در دیتابیس اوراکل نمی توان از نام مستعار تعریف شده برای ستون ها در عبارت WHERE استفاده کرد. زیرا ممکن است عبارت WHERE زودتر از قسمت اول دستور SELECT که ستون ها و نام مستعار آنها تعریف می شود پردازش شود.

نکته: در عبارت WHERE مقدارهایی که از نوع کارکتر یا تاریخ هستند باید داخل **علامت** ‘ ’ تعریف شوند. مقادیر از نوع کاراکتر حساس به بزرگی یا کوچکی حروف می باشند. همچنین مقادیر تاریخ دارای فرمت مشخصی هستند که باید طبق همان قالب تعریف شوند.

مثال: در عبارت WHERE از مقدارهای کاراکتری و تاریخی استفاده شده است.

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

LAST_NAME	JOB_ID	DEPARTMENT_ID
1 Whalen	AD_ASST	10

```
SELECT last_name
FROM   employees
WHERE  hire_date = '17-OCT-03' ;
```

LAST_NAME
1 Rajs

۲.۱ عملگرهای مقایسه ای در عبارت WHERE

در هنگام استفاده از عبارت WHERE، می توان از عملگرهای مقایسه ای زیر استفاده کرد:

عملگر	مفهوم
=	برابر با
>	بزرگتر از
>=	بزرگتر از یا مساوی با
<	کوچکتر از
<=	کوچکتر از یا مساوی با
<>	برابر نیست با
BETWEEN ...AND...	بین دو مقدار
IN(set)	برابر با یکی از مقدارهای داخل لیست
LIKE	شبیه به یک مقدار
IS NULL	برابر با یک مقدار NULL

در شکل زیر، قالب کلی استفاده از این نوع عملگرها را به همراه سه مثال مشاهده می کنید:

Syntax

```
... WHERE expr operator value
```

Example

```
... WHERE hire_date = '01-JAN-05'
... WHERE salary >= 6000
... WHERE last_name = 'Smith'
```

نکته: جهت تعیین حالت نامساوی علاوه بر **علامت <>** می توان از **علامت های =! ، =^** نیز استفاده نمود.

مثال: نام خانوادگی کارمندانی که حقوق آنها کمتر از ۳۰۰۰ می باشد را نمایش دهید.

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

	LAST_NAME	SALARY
1	Matos	2600
2	Vargas	2500

۲.۲ عملگر BETWEEN در عبارت WHERE

با استفاده از عملگر BETWEEN فقط سطرهایی که مقدار آنها در یک محدوده خاص هستند انتخاب می شوند.

مثال: نام و نام خانوادگی کارکنانی که حقوق آنها بین ۲۵۰۰ تا ۳۵۰۰ می باشد:

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

Lower limit Upper limit

	LAST_NAME	SALARY
1	Rajs	3500
2	Davies	3100
3	Matos	2600
4	Vargas	2500

نکته: می توان از عملگر BETWEEN برای مقادیر تاریخ یا کاراکتر نیز استفاده کرد.

مثال: از جدول order_details فقط سطرهایی را نمایش دهید که تاریخ آنها بین 2014/02/01 تا 2014/02/28 باشد.

```
SELECT * FROM order_details WHERE order_date BETWEEN
TO_DATE ('2014/02/01', 'yyyy/mm/dd') AND TO_DATE
('2014/02/28', 'yyyy/mm/dd');
```

مثال : نمایش نام هایی که از لحاظ ترتیب حروف الفبایی بین king و Whalen هستند:

```
SELECT last_name FROM  employees  WHERE  last_name BETWEEN 'King' AND
'Whalen';
```

۲.۳ عملگر IN در عبارت WHERE

در عملگر IN ، لیستی از مقدارها را مشخص می کنیم تا در دستور SQL فقط سطرهایی که دارای مقداری از این لیست هستند انتخاب شوند.

مثال: فقط اطلاعات افرادی که شماره MANAGER_ID آنها برابر با ۱۰۰، ۱۰۱ و یا ۲۰۱ است را نمایش دهید.

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

	EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
1	101	Kochhar	17000	100
2	102	De Haan	17000	100
3	124	Mourgos	5800	100
4	149	Zlotkey	10500	100
5	201	Hartstein	13000	100
6	200	Whalen	4400	101
7	205	Higgins	12008	101
8	202	Fay	6000	201

نکته: در لیست عملگر IN می توان از کاراکتر یا تاریخ نیز استفاده نمود به شرطی که از **علامت ‘‘** استفاده گردد.

مثال: اطلاعات افرادی که نام خانوادگی آنها Vargas یا Hartstein است را نمایش دهید.

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

۲.۴ عملگر LIKE در عبارت WHERE

به منظور جستجوی مقدارهای مورد نظر از عملگر LIKE استفاده می شود. همچنین می توان جهت افزایش قدرت جستجو از دو **علامت ٪ و _** استفاده نمود. منظور از علامت ٪ هر تعداد و هر نوع از مقادیر می باشد و علامت _ به معنی فقط یک مورد از هر مقدار است.

مثال: نام افرادی که اسم آنها با حرف S شروع می شود را نمایش دهید.

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

	FIRST_NAME
1	Shelley
2	Steven

مثال: اطلاعات افرادی که نام خانوادگی آنها با هر حرفی آغاز شود و سپس دارای یک حرف o باشد را نمایش دهید.

```
SELECT last_name
FROM   employees
WHERE  last_name LIKE '_o%' ;
```

	LAST_NAME
1	Kochhar
2	Lorentz
3	Mourgos

نکته: می توان از عملگر LIKE برای مقادیر عددی نیز استفاده کرد.

مثال: ستون ACCOUNT_NUMBER از نوع عددی است.

```
SELECT * FROM suppliers WHERE account_number LIKE
'92314_';
```

نکته: اگر در متن جستجوی LIKE حروف دستوری خاص یعنی ٪ و _ وجود داشته باشد آن حروف با استفاده از ESCAPE نادیده گرفته می شوند. توجه شود که می توان بجای علامت ! از هر حرف یا علامت دیگر استفاده کرد.

مثال : عبارت water% را جستوجو کنید و نمایش دهید.

```
WHERE supplier_name LIKE 'Water!%' ESCAPE '!';
```

در این مثال حرف ٪ بعد از کلمه water جستجو خواهد شد. بنابراین نام water% نیز نمایش داده می شود. توجه شود که حرف ! جستوجو نمی شود.

نکته: می توان از عبارت NOT LIKE جهت نمایش سایر مقادیر استفاده نمود.

مثال: تمام اسم هایی که با W شروع نمی شوند را نمایش دهید.

```
SELECT supplier_name FROM suppliers WHERE supplier_name
NOT LIKE 'W%';
```

در این مثال تمام اسم هایی که با w شروع نمی شوند نمایش می یابند.

۲.۵ استفاده از شروط NULL

از آنجایی که NULL یک مقدار نامشخص می باشد نمی توان وجود یا عدم وجود آن را با **شرط =** تعیین کرد. بنابراین برای این منظور از عبارت های is NULL و is not NULL استفاده می شود.

مثال: فقط سطرهایی نمایش یابند که ستون manager_id آنها NULL باشد.

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

	LAST_NAME	MANAGER_ID
1	King	(null)

۲.۶ عملگرهای منطقی

عملگرهای منطقی نتایج چندین شرط را ترکیب می کنند تا یک شرط کلی ایجاد گردد. در جدول زیر عملگرهای منطقی را می بینید.

عملگر	توضیحات
AND	اگر هر دو شرط برقرار باشند مقدار صحیح برمی گرداند
OR	اگر یکی از شرطها برقرار باشند مقدار صحیح برمی گرداند
NOT	اگر شرط برقرار نباشد مقدار صحیح برمی گرداند

مثال : نمایش اطلاعات کارکنان به شرطی که حقوق کارکنان بالای ۱۰۰۰۰ و job_id آنها نیز شبیه به کلمه MAN باشد.

```
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
1	149	Zlotkey	SA_MAN	10500
2	201	Hartstein	MK_MAN	13000

نکته: زمانی که در بین چند عبارت شرطی از عملگر AND استفاده نشود نتیجه عملیات به صورت جدول زیر می باشد.

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

نکته: زمانی که از عملگر OR استفاده می شود، فقط کافیست یکی از شرط ها برقرار باشد.

مثال : نمایش اطلاعات افرادی که حقوق آنها بیشتر از ۱۰۰۰۰ است یا job_id آنها شبیه به MAN می باشد.

```
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
1	100	King	AD_PRES	24000
2	101	Kochhar	AD_VP	17000
3	102	De Haan	AD_VP	17000
4	124	Mourgos	ST_MAN	5800
5	149	Zlotkey	SA_MAN	10500
6	174	Abel	SA_REP	11000
7	201	Hartstein	MK_MAN	13000
8	205	Higgins	AC_MGR	12008

نکته: زمانی که در بین چند عبارت شرطی از عملگر OR استفاده گردد نتیجه کلی به صورت زیر می باشد.

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

نکته: از عملگر NOT برای بررسی عدم وجود یک شرط استفاده می شود.

مثال: نمایش اطلاعات افرادی که job_id آنها در لیست IN تعریف نشده است.

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

	LAST_NAME	JOB_ID
1	De Haan	AD_VP
2	Fay	MK_REP
3	Gietz	AC_ACCOUNT
4	Hartstein	MK_MAN
5	Higgins	AC_MGR
6	King	AD_PRES
7	Kochhar	AD_VP
8	Mourgos	ST_MAN
9	Whalen	AD_ASST
10	Zlotkey	SA_MAN

۲.۷ بررسی اولویت

در جدول زیر اولویت هرکدام از عملگرها و شرط ها نسبت به هم مشخص گردیده است. به این ترتیب که شماره ۱ بالاترین اولویت را دارد.

شماره عملگر	توضیحات
1	عملگرهای محاسباتی
2	عملگر CONCATENATION
3	شرط های مقایسه ای
4	IS [NOT] NULL , LIKE , [NOT] IN
5	[NOT] BETWEEN
6	شرط غیر مساوی
7	عملگر منطقی NOT
8	عملگر منطقی AND
9	عملگر منطقی OR

نکته: استفاده از پرانتز سبب ایجاد اولویت بالاتر می شود.

مثال: اطلاعات کارمندانی نمایش داده شوند که department_id آنها ۶۰ است و یا department_idآنها ۸۰ است ولی حقوقی بالاتر از ۱۰۰۰۰ دریافت می کنند.

```
SELECT last_name, department_id, salary
FROM   employees
WHERE  department_id = 60
OR     department_id = 80
AND    salary > 10000;
```

1

	LAST_NAME	DEPARTMENT_ID	SALARY
1	Hunold	60	9000
2	Ernst	60	6000
3	Lorentz	60	4200
4	Zlotkey	80	10500
5	Abel	80	11000

مثال: با استفاده از پرانتز در مثال قبلی، فقط اطلاعات کارمندانی که حقوق آنها بیشتر از ۱۰۰۰۰ است، نمایش می یابد.

```
SELECT last_name, department_id, salary
FROM   employees
WHERE  (department_id = 60
OR     department_id = 80)
AND    salary > 10000;
```

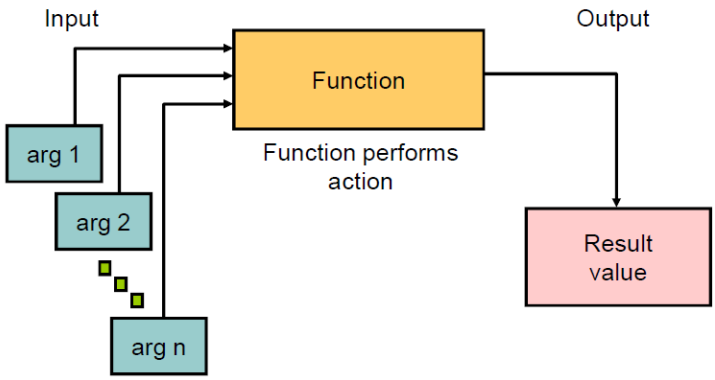
2

	LAST_NAME	DEPARTMENT_ID	SALARY
1	Zlotkey	80	10500
2	Abel	80	11000

۳ استفاده از توابع در SQL

در دستورات SQL هر تابع مجموعه ای از عملیات انجام می دهد و می تواند تغییراتی را بر داده های دیتابیس اعمال کند. در دیتابیس اوراکل می توان یک تابع جدید تعریف نمود یا از توابع از پیش تعریف شده استفاده کرد. توابع از پیش تعریف شده برای اطلاعات با نوع داده های مختلف (کاراکتر، عددی، تاریخی و...) استفاده می شوند. در این فصل ساختار کلی توابع در SQL را معرفی می کنیم و تعدادی از توابع از پیش تعریف شده در دیتابیس اوراکل و روش استفاده از آنها را با ذکر مثال توضیح می دهیم.

توابع SQL می توانند یک یا چند آرگومان ورودی داشته باشند ولی تنها یک مقدار را به عنوان خروجی (output) برمی گردانند.



نکته: مقادیر ورودی یا همان آرگومنت ها می توانند عبارت، ستون، متغیر و یا یک مقدار ثابت باشند.

نکته: توابع SQL دو نوع هستند:

Single-Row: این مدل از توابع برای هر ردیف از جداول اعمال می شوند و یک نتیجه مشخص به ازای هرکدام از ردیف ها برمی گردانند. مانند توابع کاراکتری، اعداد، تاریخ، توابع تبدیل نوع داده و توابع عمومی که در این فصل و فصل بعدی توضیح داده می شوند.

Multiple-Row: این دسته از توابع برای مجموعه ای از ردیف های جداول اعمال می شوند و به ازای هر مجموعه از ردیف ها یک خروجی مشخص بازمی گردانند.

۳.۱ توابع کاراکتری

ورودی توابع کاراکتری از نوع کاراکتر است و خروجی آنها می تواند عدد یا کاراکتر باشد. بنابراین اگر خروجی توابع کاراکتری، عدد باشد می توان آن عدد را در ستونی از نوع عدد ذخیره کرد.

در جدول زیر سه مورد از توابع کاراکتری را می بینید که ورودی آنها عبارت 'SQL Course' است.

Function	Result
LOWER('SQL Course')	sql course
UPPER('SQL Course')	SQL COURSE
INITCAP('SQL Course')	Sql Course

- **تابع LOWER**: این تابع عبارت کاراکتری ورودی را به یک عبارت با حروف کوچک تبدیل می کند.
- **تابع UPPER**: این تابع عبارت کاراکتری ورودی را به یک عبارت با حروف بزرگ تبدیل می کند.
- **تابع NITCAP**: این تابع اولین حرف در هر کلمه از عبارت کاراکتری ورودی را به حرف بزرگ تبدیل می کند و سایر حروف را به صورت حروف کوچک نمایش می دهد.

مثال: اطلاعات شغلی کارمند Higgins را نمایش دهید.

در **اولین دستور SELECT** نام کارمند را به صورت 'higgins' جستجو می کند و هیچ نامی در دیتابیس پیدا نمی کند اما در **قسمت دوم** با تغییر شرط WHERE مشخصات کارمند مورد نظر نمایش می یابد.

بنابراین اگر در دیتابیس نام این کارمند به صورت HIGGINS ، Higgins یا ... ثبت شده باشد همچنان دستور SELECT دوم نتیجه مورد نظر را نمایش می دهد.

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

```
0 rows selected
```

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

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	205	Higgins	110

همانطور که در مثال می بینید نام کارمند به همان شکلی که در دیتابیس درج شده است در خروجی نمایش داده می شود.

گروه دیگری از توابع کاراکتری را همراه با یک مثال در جدول زیر می بینید.

Function	Result
CONCAT('Hello', 'World')	HelloWorld
SUBSTR('HelloWorld',1,5)	Hello
LENGTH('HelloWorld')	10
INSTR('HelloWorld', 'W')	6
LPAD(last_name,12,'-')	*****24000
RPAD(first_name, 12, '-')	24000*****

—**تابع CONCAT** دو مقدار را به هم متصل می کند.

—**تابع SUBSTR** یک رشته کاراکتری با طول مشخص شده در آرگومنت سوم را از آرگومنت اول جدا کرده و بازمی گرداند. در ضمن دومین آرگومنت نشانگر نقطه شروع می باشد(نشانگر اولین حرف عبارت شماره ۱ می باشد).

—**تابع LENTGH** طول یک رشته کاراکتری ورودی را محاسبه می کند و بر حسب یک عدد بر می گرداند.

—**تابع INSTR** شماره محل قرارگیری یک حرف خاص که در آرگومنت دوم تعیین می شود را باز می گرداند.

—**تابع LPAD** اگر طول رشته کاراکتری ورودی (آرگومنت اول) کمتر از عدد آرگومنت دوم باشد از حرف آرگومنت سوم در سمت چپ آن رشته کاراکتری استفاده می گردد تا طول عبارت نهایی به عدد آرگومنت دوم برسد.

—**تابع RPAD** همانند LPAD می باشد ولی در سمت راست عبارت کاراکتری اعمال می گردد.

مثال:

```
LPAD('tech', 8, '0');
Result: '0000tech'
LPAD('tech on the net', 16, 'z');
Result: 'ztech on the net'
RPAD('tech', 8, '0')
```

Result: 'tech0000'

```
SELECT SUBSTR('ABCDEFG',3,4) "Substring"
     FROM DUAL;
```

Result: 'CDEF'

مثال:

```
SELECT CONCAT(CONCAT(last_name, '''s job category is '), job_id)
"Job" FROM   employees
WHERE  SUBSTR(job_id, 4) = 'REP';
```

	Job
1	Abel's job category is SA_REP
2	Fay's job category is MK_REP
3	Grant's job category is SA_REP
4	Taylor's job category is SA_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'?
1	102	LexDe Haan	7	5
2	200	JenniferWhalen	6	3
3	201	MichaelHartstein	9	2

نکته: توابع از نوع Single-Row می توانند به صورت تودرتو استفاده شوند. در این حالت ترتیب اعمال توابع از درونی ترین تابع آغاز می شود.

مثال: در این مثال ابتدا تابع SUBSTR اجرا گردیده و نتیجه آن تابع به عنوان آرگومنت تابع CONCAT بازگردانی می شود. همچنین در نهایت تابع UPPER اجرا می شود:

```
SELECT last_name,
  UPPER(CONCAT(SUBSTR (LAST_NAME, 1, 8), '_US'))
FROM   employees
WHERE  department_id = 60;
```

	LAST_NAME	UPPER(CONCAT(SUBSTR(LAST_NAME,1,8),'_US'))
1	Hunold	HUNOLD_US
2	Ernst	ERNST_US
3	Lorentz	LORENTZ_US

۳.۲ توابع عددی

این توابع و مثال آنها در جدول زیر نمایش داده شده اند.

Function	Result
ROUND(45.926, 2)	45.93
TRUNC(45.926, 2)	45.92
CEIL (2.83)	3
FLOOR (2.83)	2
MOD (1600, 300)	100

—**تابع ROUND** دومین آرگومنت ورودی را به عنوان دقت اعشار در عدد آرگومنت اول اعمال می کند و نتیجه را به سمت بالا round کرده و برمی گرداند.

—**تابع TRUNC** همانند ROUND عمل می کند با این تفاوت که نتیجه را به سمت پایین round می کند.

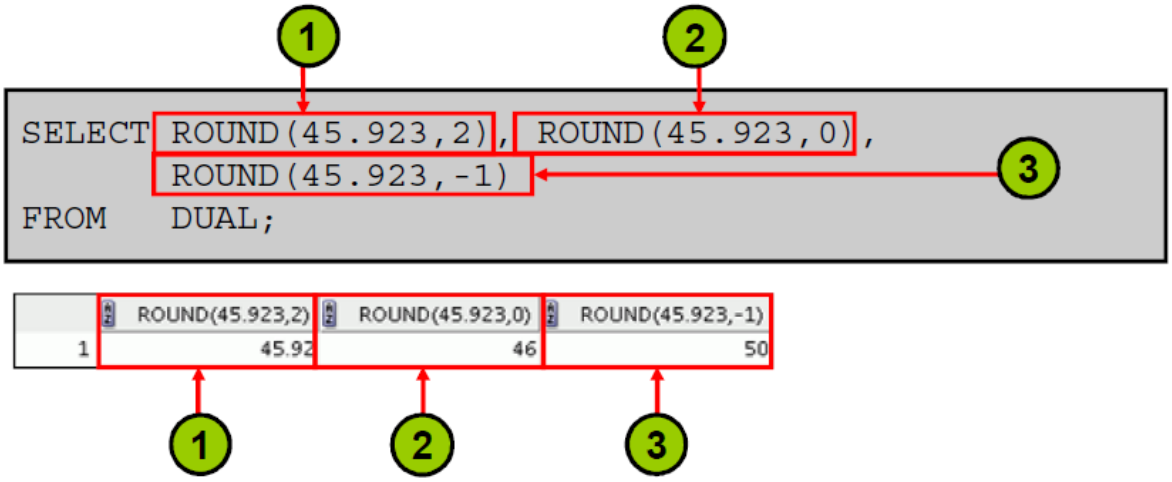
—**تابع CEIL** یک عدد اعشاری دریافت می کند و مقدار صحیح آن که به سمت بالا round شده است را بر می گرداند.

—**تابع FLOOR** همانند تابع CEIL عمل می کند با این تفاوت که نتیجه نهایی به سمت پایین round می شود.

—**تابع MOD** باقی مانده عمل تقسیم دو مقدار ورودی را بر می گرداند.

نکته: اگر در توابع ROUND و TRUNC آرگومنت دوم تعیین نشود این مقدار به صورت پیش فرض برابر با ۰ خواهد بود و اگر مقدار تعیین شده برای این آرگومنت منفی باشد در این صورت عملیات در سطح یکان یا دهگان و ... اعمال می شود.

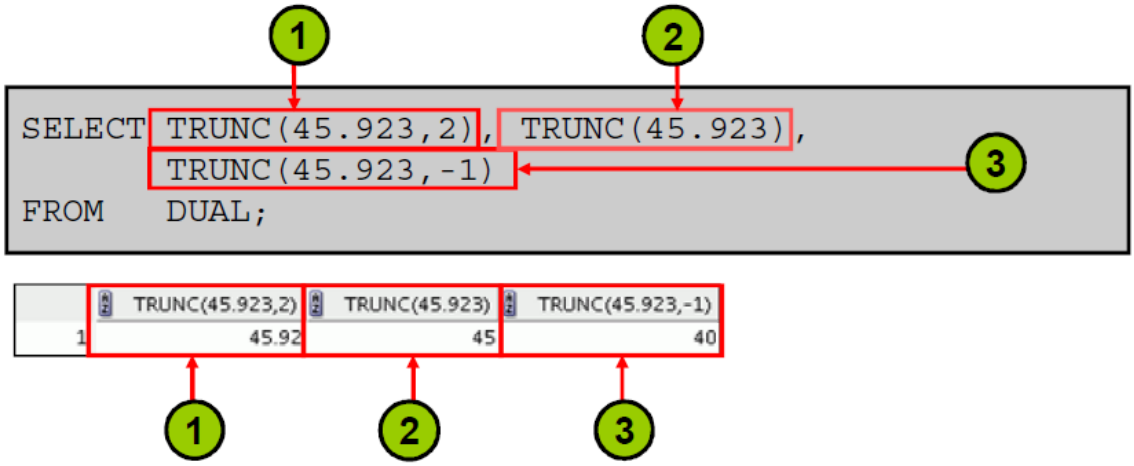
مثال:



نکته: در دیتابیس اوراکل مالک جدول DUAL کاربر SYS است ولی توسط تمام کاربران قابل دسترسی می باشد. این جدول شامل یک ستون به نام DUMMY می باشد.

از این جدول زمانی استفاده می گردد که همانند مثال قبل احتیاجی به دیتای جداول دیگر نیست و فقط نیاز است که یک مقدار ثابت نمایش داده شود. بنابراین به منظور کامل کردن دستور Select نام این جدول در عبارت from ذکر می گردد.

مثال:



مثال: در این مثال به منظور نمایش اطلاعات کارمندانی که شماره EMPLOYEE_ID آنها زوج می باشد از تابع MOD استفاده شده است.

```
SELECT employee_id as "Even Numbers", last_name
FROM employees
WHERE MOD(employee_id,2) = 0;
```

	Even Numbers	LAST_NAME
1	174	Abel
2	142	Davies
3	102	De Haan
4	104	Ernst
5	202	Fay
6	206	Gietz
7	178	Grant
8	100	King
9	124	Mourgos
10	176	Taylor
11	144	Vargas
12	200	Whalen

۳.۳ تاریخ در اوراکل و توابع تاریخی

قالب نمایش تاریخ در دیتابیس اوراکل به صورت **DD-MON-RR**(سال-ماه-روز) می باشد.

مثال:

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date < '01-FEB-2008';
```

	LAST_NAME	HIRE_DATE
1	King	17-JUN-03
2	Kochhar	21-SEP-05

نکته: زمانی که در دیتابیس یک رکورد با ستونی از نوع تاریخ درج می شود اطلاعات مربوط به قرن آن رکورد نیز از طریق تابع SYSDATE تعیین می گردد و همراه آن رکورد درج می شود. هر چند در زمان نمایش اطلاعات تاریخ قرن نمایش داده نمی شود. در واقع اطلاعات تاریخی در دیتابیس به قالب زیر درج می شوند:

ثانیه دقیقه ساعت روز ماه سال قرن

نکته: تابع SYSDATE زمان و تاریخ فعلی سیستم را نمایش می دهد. می توان از این تابع در قسمت مربوط به نام ستون هر عبارت SELECT استفاده نمود ولی معمولا از نام جدول عمومی DUAL استفاده می شود.

```
SELECT sysdate
FROM   dual;
```

SYSDATE
1 24-AUG-12

نکته: تابع SYSDATE تاریخ و زمان مربوط به سروری که دیتابیس اوراکل بر روی آن راه اندازی شده است را نمایش می دهد.

بنابراین اگر در یک SESSION خاص از سرور محلی خود به یک سرور دیتابیس در کشور دیگر متصل شویم و تابع SYSDATE را فراخوانی کنیم، تاریخ آن کشور مقصد نمایش داده می شود.

ولی اگر در این SESSION از تابع CURRENT_DATE استفاده گردد، تاریخ محلی نمایش داده می شود. تابع CURRENT_TIMESTAMP نیز زمان و تاریخ محلی را نمایش می دهد.

مثال:

```
SELECT CURRENT_DATE FROM DUAL;
```

CURRENT_DATE
26-MAY-14

```
SELECT CURRENT_TIMESTAMP FROM DUAL;
```

CURRENT_TIMESTAMP
26-MAY-14 12.25.34.401622000 AM ETC/UNIVERSAL

۳.۴ عملیات ریاضی در داده های از نوع تاریخ

از آنجایی که داده های از نوع تاریخ با فرمت عددی در دیتابیس ذخیره می شوند، می توان از عملیات **جمع** و **تفریق** برای آنها استفاده نمود. اگر در عملیات جمع یا تفریق یک عدد ثابت استفاده شود، روزهای آن تاریخ تغییر می کند. اگر مقدار آن عدد بر ۲۴ تقسیم گردد تعداد ساعت مشخص خواهد شد. همچنین می توان مقدار دو تاریخ مختلف را جمع یا تفریق نمود.

مثال: نام کارمندان به همراه سابقه خدمتی آنها براساس تعداد هفته نمایش داده شود. (تعداد روزهایی که از زمان استخدام یک کارمند سپری شده است از تفاضل تاریخ سیستم و تاریخ استخدام آن کارمند بدست آمده است)

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

	LAST_NAME	WEEKS
1	King	478.871917989417989417989417989417989418
2	Kochhar	360.729060846560846560846560846560846561
3	De Haan	605.300489417989417989417989417989417989

توابع زیر مربوط به داده های از نوع تاریخ می باشند:

—**تابع MONTHS_BETWEEN** دو مقدار از نوع تاریخ به عنوان ورودی دریافت می کند و اختلاف ماه های آنها را برمی گرداند. اگر تاریخ ورودی اول جدیدتر باشد مقدار بازگشتی مثبت خواهد بود. در غیر این صورت یک مقدار منفی برگردانده می شود.

مثال:

Function	Result
MONTHS_BETWEEN ('01-SEP-05','11-JAN-04')	19.6774194

—**تابع ADD_MONTHS** یک مقدار صحیح به عنوان آرگومنت دوم دریافت می کند و این مقدار را به عدد مربوط به ماه در تاریخ ورودی اضافه می کند (این مقدار صحیح می تواند یک عدد منفی باشد).

مثال:

Function	Result
ADD_MONTHS ('31-JAN-04',1)	'29-FEB-04'

—**تابع NEXT_DAY** یک مقدار کاراکتری که بیانگر یک روز خاص از هفته می باشد را توسط آرگومنت دوم دریافت می کند و تاریخی که آن روز از هفته برای بار اول رخ می دهد را بر می گرداند.

مثال:

Function	Result
NEXT_DAY ('01-SEP-05','FRIDAY')	'08-SEP-05'

—تابع LAST_DAY یک تاریخ به عنوان ورودی دریافت کرده و تاریخ آخرین روز در ماه آن تاریخ را بر می گرداند.

مثال:

Function	Result
LAST_DAY ('01-FEB-05')	'28-FEB-05'

نکته: می توان از توابع ROUND و TRUNC برای مقادیر از نوع تاریخ نیز استفاده نمود.

مثال: تاریخ ۲۲ آگوست سال ۰۳ را با تابع ROUND و TRUNC و بر اساس ماه و سال نمایش دهید.

```
ROUND( date [, format] )
ROUND(TO_DATE ('22-AUG-03'),'YEAR')
Result: '01-JAN-04'
ROUND(TO_DATE ('22-AUG-03'),'MONTH')
Result: '01-SEP-03'
TRUNC ( date [, format ] )
TRUNC(TO_DATE('22-AUG-03'), 'YEAR')
Result: '01-JAN-03'
TRUNC(TO_DATE('22-AUG-03'), 'MONTH')
Result: '01-AUG-03'
```

۴ توابع تبدیل نوع داده و توابع عمومی

توابعی که در این فصل توضیح داده می شوند، داده های دریافتی از نوع کاراکتری، عددی یا تاریخ را به نوع دیگر تبدیل می کنند. در ادامه توابعی که مربوط به مقادیر NULL می باشند را توضیح می دهیم. همچنین روش استفاده از عبارات شرطی با منطق IF THEN-ELSE توضیح داده می شود.

گاهی اوقات در یک دستور SQL احتیاجی به نوع خاصی از داده ها می باشد. اگر اطلاعات مورد نظر از نوع داده دیگر باشند باید عمل تبدیل نوع داده انجام گیرد. در دیتابیس اوراکل این عمل تبدیل به دو روش IMPLICIT (تلویحی) وEXPLICIT (صریح) انجام می شود.

- روش IMPLICIT که توسط دیتابیس اوراکل و به صورت اتوماتیک انجام می شود.
- روش EXPLICIT که توسط کاربر یا برنامه نویس انجام می شود.

فرمت دستور تبدیل داده در روش EXPLICIT به شکل زیر است:

نوع داده TO نوع داده

نکته: اوراکل پیشنهاد می کند تا حد امکان در دستورات SQL از روش EXPLICIT استفاده شود.

۴.۱ تبدیل نوع داده به روشIMPLICIT

در این روش، عمل تبدیل نوع داده های مختلف به صورت اتوماتیک توسط دیتابیس اوراکل انجام می شود. تعدادی از تبدیل های نوع داده هایی که به روش IMPLICIT انجام می شوند را در جدول زیر می بینید.

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

نکته: در عمل تبدیل کارکتر به عدد باید رشته کاراکتری شامل یک عدد باشد.

مثال: در دستور SELECT زیر، ستون JOB_ID از نوع داده ای کاراکتر می باشد. بنابراین دیتابیس اوراکل به صورت اتوماتیک و به روش IMPLICIT عدد ۲ را به رشته کاراکتری '۲' تبدیل می نماید.

```
SELECT * FROM EMPLOYEES WHERE JOB_ID=2;
```

مثال: **در دستور SELECT زیر، رشته کاراکتری '01-JAN-90' به روش IMPLICIT به داده از نوع تاریخ تبدیل می شود و در عبارت WHERE استفاده می گردد.**

SQL>SELECT NAME FROM EMPLOYEES WHERE HIRE_DATE > '01-JAN-90';

۴.۲ تبدیل نوع داده به روش EXPLICIT

سه تابع زیر به روش EXPLICIT عمل تبدیل نوع داده را انجام می دهند:

- **TO_CHAR (number|date [, fmt [, nlsparams]])**

این تابع نوع داده ی عدد یا تاریخ را به کاراکتری تبدیل می کند.

- **TO_NUMBER (char[,fmt[, nlsparams]])**

این تابع نوع داده ی کاراکتری را به عددی تبدیل می کند.

- **TO_DATE (char[,fmt[,nlsparams]])**

این تابع نوع داده ی کاراکتری را به نوع داده ی تاریخی تبدیل می کند.

پارامتر fmt مدل فرمت تبدیل داده می باشد. این مدل فرمت باید داخل علامت ' ' قرار گیرد و حساس به بزرگی یا کوچکی حروف می باشد. همچنین باید با یک علامت ویرگول از قسمت قبلی دستور جدا گردد.

پارامتر nlsparams قالب کاراکترهایی که باید برای برخی نشانگرهای عددی مانند علامت دسیمال استفاده گردد را مشخص می کند. این پارامتر معمولا تعریف نمی شود.

نکته: اگر هر کدام از پارامترهای ورودی این توابع تعریف نشوند از مقدار پیش فرض آنها در SESSION استفاده می شود.

مثال: مدل فرمت تاریخ '11-NOV-2000' برابر با 'DD-MON-YYYY' می باشد.

مثال: در این دستور، تاریخ استخدام کارمند Higgins با مدل فرمت دلخواه کاراکتری 'MM/YY' نمایش می یابد.

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

مثال: در این دستور نام افراد همراه با تاریخ استخدام آنها به صورت یک رشته کاراکتری با فرمت دلخواه نمایش می یابد. توجه شود که عبارت fm قبل از DD سبب ایجاد خوانایی بهتر در اعداد خروجی می شود.

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

	LAST_NAME	HIREDATE
1	King	17 June 2003
2	Kochhar	21 September 2005
3	De Haan	13 January 2001
4	Hunold	3 January 2006
5	Ernst	21 May 2007
6	Lorentz	7 February 2007
7	Mourgos	16 November 2007
8	Rajs	17 October 2003

نکته: برای تبدیل مقادیر عددی به مقادیر کاراکتری از تابع TO_CHAR استفاده می شود.

نکته: برخی از فرمت هایی که در تابع TO_CHAR قابل استفاده می باشند را در جدول زیر می بینید.

Element	Description	Example	Result
9	Numeric position (number of 9s determine display width)	999999	1234
0	Display leading zeros	099999	001234
$	Floating dollar sign	$999999	$1234
L	Floating local currency symbol	L999999	FF1234
D	Returns the decimal character in the specified position. The default is a period (.).	9999D99	1234.00
.	Decimal point in position specified	999999.99	1234.00
G	Returns the group separator in the specified position. You can specify multiple group separators in a number format model.	9G999	1,234
,	Comma in position specified	999,999	1,234
MI	Minus signs to right (negative values)	999999MI	1234-
PR	Parenthesize negative numbers	999999PR	<1234>
EEEE	Scientific notation (format must specify four Es)	99.999EEEE	1.234E+03
U	Returns in the specified position the "Euro" (or other) dual currency	U9999	€1234
V	Multiply by 10 *n* times (*n* = number of 9s after V)	9999V99	123400
S	Returns the negative or positive value	S9999	-1234 or +1234
B	Display zero values as blank, not 0	B9999.99	1234.00

مثال: حقوق کارمند ERNEST با فرمت '$۹۹,۹۹۹.۰۰' نمایش داده شود.

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

	SALARY
1	$6,000.00

مثال: نمایش نام خانوادگی کارمندانی که قبل از سال ۱۹۹۰ استخدام شده اند.

```
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(HIRE_DATE,'DD-MON-YYYY')
1	Popp	03-Feb-1989

۴.۳ توابع عمومی

این دسته از توابع با مقادیر NULL سروکار دارند و عملیات این توابع بر اساس NULL بودن یا نبودن مقدارها انجام می شود.

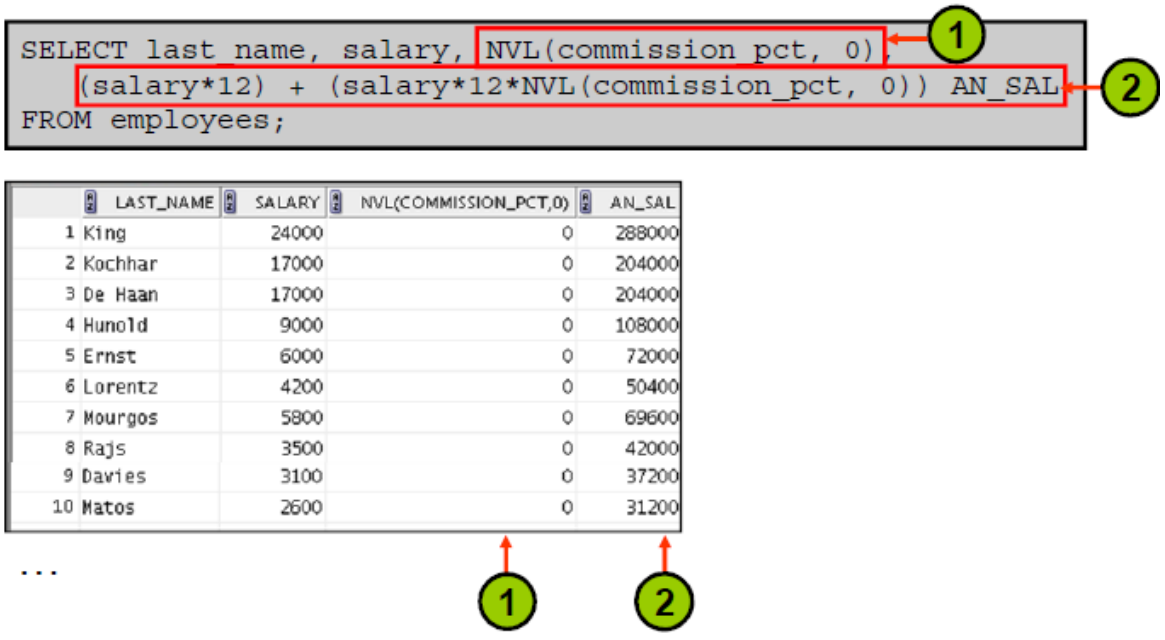
تابع NVL

فرمت کلی این تابع به صورت NVL(EXP1,EXP2) می باشد. این تابع یک مقدار NULL را به یک مقدار حقیقی تبدیل می کند. البته توجه شود که نوع داده های EXP1 و EXP2 باید یکسان باشند. زمانی که از این تابع استفاده می گردد اگر EXP1 شامل یک مقدار NULL باشد مقدار EXP2 جایگزین آن می شود.

مثال:

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

مثال: در فرمول مثال زیر برای محاسبه درآمد سالیانه تمام کارمندان باید مقادیر مربوط به ستون حق کمیسیون آنها مقداری غیر از NULL باشد. لذا مقدار این ستون برای کارمندانی که حق کمیسیون نگرفته بودند برابر با ۰ شده است.



تابع NVL2

این تابع به صورت NVL2(EXP1,EXP2,EXP3) تعریف می گردد. اگر مقدار EXP1 یک مقدار NULL باشد مقدار EXP3 باز می گردد و در غیر این صورت مقدار EXP2 برگردانده خواهد شد.

مثال: اگر کارمندان حق کمیسیون دریافت کرده بودند برای نام ستون INCOME عبارت 'SAL+COMM' چاپ شود و در غیر این صورت عبارت 'SAL' چاپ گردد.

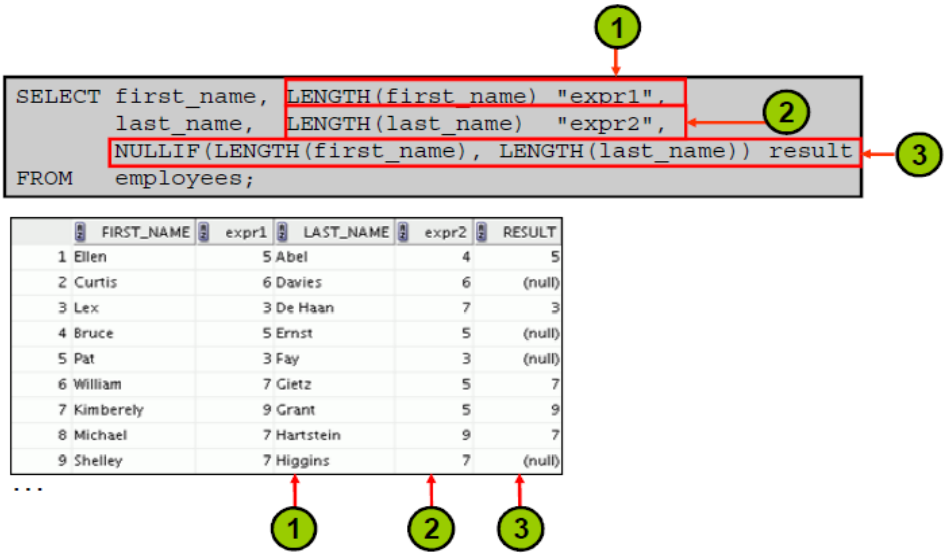
```
SELECT last_name,  salary, commission_pct
       NVL2(commission_pct,
            'SAL+COMM', 'SAL') income
FROM   employees WHERE department_id IN (50, 80);
```

	LAST_NAME	SALARY	COMMISSION_PCT	INCOME
1	Mourgos	5800	(null)	SAL
2	Rajs	3500	(null)	SAL
3	Davies	3100	(null)	SAL
4	Matos	2600	(null)	SAL
5	Vargas	2500	(null)	SAL
6	Zlotkey	10500	0.2	SAL+COMM
7	Abel	11000	0.3	SAL+COMM
8	Taylor	8600	0.2	SAL+COMM

تابع NULLIF

این تابع به صورت NULLIF(EXP1,EXP2) بکار می رود و اگر مقدار EXP1 و EXP2 برابر بود مقدار NULL بازمی گردد و در غیر این صورت مقدار EXP1 بازگردانده می شود.

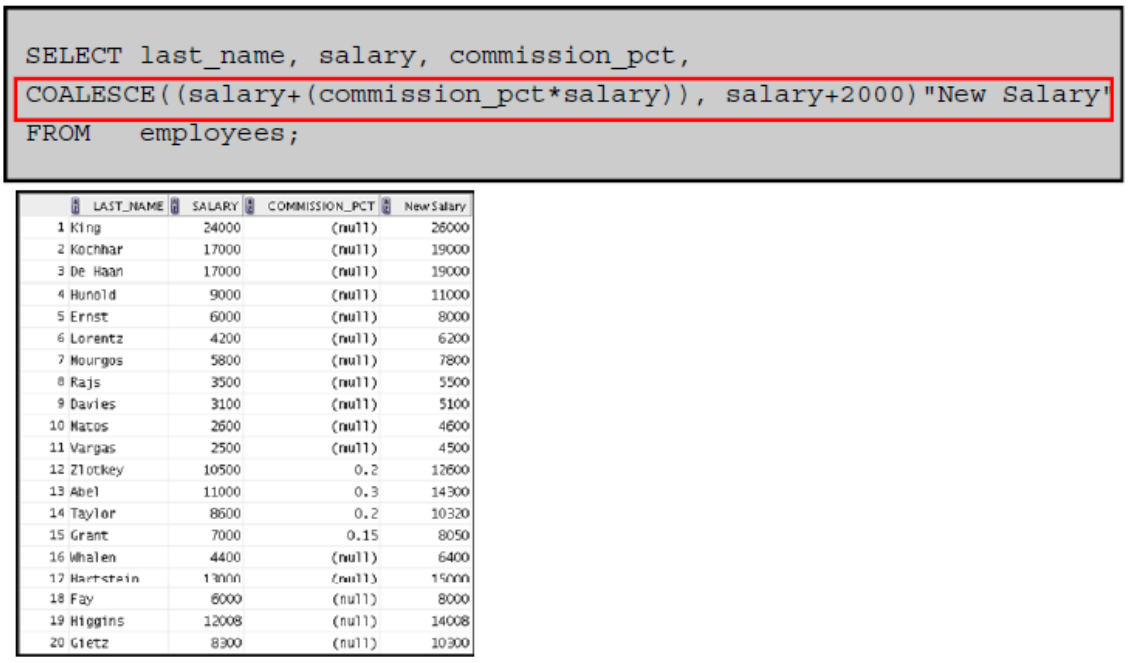
مثال: در این مثال تعداد حروف به کار رفته در نام و نام خانوادگی کارمندان در ستون های EXP1 و EXP2 نمایش داده می شود. اگر این مقدارها برابر بودند در ستون RESULT مقدار NULL چاپ می گردد و در غیر این صورت مقدار EXP1 چاپ می شود.



تابع COALESCE

این تابع به صورت COALESCE(EXP1,EXP2,...,EXPn) تعریف می گردد. طرز کار این تابع همانند تابع NVL می باشد با این تفاوت که چندین مقدار EXP را به ترتیب برای NULL بودن بررسی کند و هر کدام NULL بود به سراغ بعدی برود. به عبارتی دیگر اولین مقدار EXP غیر NULL بازگردانده می شود.

مثال: در این مثال میزان حقوق کارمندان برای سال جدید محاسبه می گردد. اگر کارمندان حق کمسیون دریافت نکرده باشند حقوق آنها ۲۰۰۰ واحد اضافه می گردد و اگر دریافت کرده باشند به نسبت حق کمیسیون دریافتی افزایش حقوق خواهد داشت.



۴.۴ عبارات شرطی با منطق IF-THEN-ELSE

در دستورات SQL به منظور پیاده سازی منطق IF-THEN-ELSE از دو روش CASE و DECODE استفاده می شود.

روش 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
```

در این روش می توان از منطق شرطی، بدون نیاز به فراخوانی هیچ پروسیجری استفاده نمود. اگر هر کدام از COMPARISON_EXPRها برابر با EXP بود مقدار RETURN_EXPR مربوط به آن بازگردانده می شود.

نکته: نوع داده ی EXPR و تمامیCOMPARISION_EXPR ها باید یکسان باشد. همچنین نوع داده ی تمام return_expr ها باید یکسان باشد.

نکته: اگر هیچ کدام از شرط های CASE صدق نکند مقدار NULL بازگردانده می شود.

مثال: افزایش حقوق کارمندان بر اساس حوزه فعالیت آنها. همچنین اگر کارمندی در ۳ حوزه IT_PROG ، ST_CLERK، SA_REP فعالیت نکند، افزایش حقوق نخواهد داشت.

```
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
1	King	AD_PRES	24000	24000
...				
4	Hunold	IT_PROG	9000	9900
5	Ernst	IT_PROG	6000	6600
6	Lorentz	IT_PROG	4200	4620
7	Mourgos	ST_MAN	5800	5800
8	Rajs	ST_CLERK	3500	4025
9	Davies	ST_CLERK	3100	3565
10	Matos	ST_CLERK	2600	2990
11	Vargas	ST_CLERK	2500	2875
...				
13	Abel	SA_REP	11000	13200
14	Taylor	SA_REP	8600	10320
15	Grant	SA_REP	7000	8400

نکته: می توان از روش CASE جهت جستجوی موارد خاص استفاده نمود.

```
CASE
    WHEN condition1 THEN use_expression1
    WHEN condition2 THEN use_expression2
    WHEN condition3 THEN use_expression3
    ELSE default_use_expression
END
```

مثال: کارمندان را بر اساس میزان حقوق دریافتی در ستون خروجی QUALIFIED_SALARY به سه دسته Low ، Medium و Good تقسیم کنید.

```
SELECT last_name,salary,
(CASE WHEN salary<5000 THEN 'Low'
      WHEN salary<10000 THEN 'Medium'
      WHEN salary<20000 THEN 'Good'
      ELSE 'Excellent'
END) qualified_salary
FROM employees;
```

تابع DECODE

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

طرز کار این تابع مانند روش CASE می باشد و همان منطق IF-THEN-ELSE را پیاده سازی می کند.

مثال: افزایش حقوق کارمندان بر اساس حوزه فعالیت آنها را نمایش دهید. در ضمن اگر کارمندان در سه حوزه IT_PROG ، ST_CLERK، SA_REP فعالیت نکنند، افزایش حقوق نخواهند داشت.

```
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
...				
4	Hunold	IT_PROG	9000	9900
5	Ernst	IT_PROG	6000	6600
6	Lorentz	IT_PROG	4200	4620
7	Mourgos	ST_MAN	5800	5800
8	Rajs	ST_CLERK	3500	4025
9	Davies	ST_CLERK	3100	3565
10	Matos	ST_CLERK	2600	2990
11	Vargas	ST_CLERK	2500	2875
12	Zlotkey	SA_MAN	10500	10500
...				
13	Abel	SA_REP	11000	13200
14	Taylor	SA_REP	8600	10320
15	Grant	SA_REP	7000	8400

مثال: محاسبه و نمایش نرخ مالیات کارمندان به نسبت حقوق دریافتی آنها.

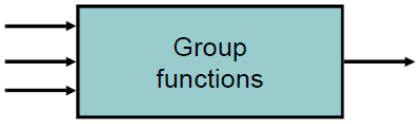
```
SELECT last_name, salary,
       DECODE (TRUNC(salary/2000, 0),
                         0, 0.00,
                         1, 0.09,
                         2, 0.20,
                         3, 0.30,
                         4, 0.40,
                         5, 0.42,
                         6, 0.44,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

۵ توابع گروهی و گروه بندی درSQL

در این فصل توابع گروهی و روش های استفاده از آنها معرفی می شوند. این توابع به ازای هر مجموعه از سطرهای جدول یک نتیجه خاص برمی گردانند. همچنین عبارت های Group By ، Having، Order by و روش استفاده از آنها به همراه مثال توضیح داده می شوند.

۵.۱ توابع گروهی

- AVG
- COUNT
- MAX
- MIN
- SUM
- LISTAGG
- STDDEV
- VARIANCE



در فصل ۳ انواع توابع SQL را معرفی کردیم. توابع گروهی به ازای مجموعه ای از سطرها یک نتیجه خاص بر می گردانند بنابراین از نوع توابع MULTIPLE-ROW هستند. در یک دستور SQL می توان از چندین تابع گروهی استفاده نمود و آنها را با علامت ویرگول از هم جدا کرد.

```
SELECT      group_function(column), ...
FROM        table
[WHERE      condition];
```

نکته: زمانی که در یک تابع گروهی از کلمه کلیدی DISTINCT استفاده شود فقط سطرهای با مقدار غیر تکراری در نظر گرفته می شود. ولی دیتابیس اوراکل به صورت پیش فرض از ALL بجای DISTINCT استفاده می کند که در این حالت تمام سطرهای جدول در نظر گرفته می شوند.

```
group_function([DISTINCT|ALL]expr)
```

نکته: در توابع گروهی نوع داده برای ورودی expr می تواند کاراکتر، عدد یا تاریخ باشد.

نکته: توابع گروهی به ازای مقدارهای NULL در سطرهای جدول هیچ نتیجه ای برنمی گردانند ولی می توان جهت جایگزینی مقادیر NULL از تابع هایNVL ، NVL2 و ... استفاده کرد.

****تابعAVG**

```
AVG([DISTINCT|ALL]n)
```

این تابع از مقدارهای ستون n میانگین می گیرد.

****تابعSUM**

```
SUM([DISTINCT|ALL]n)
```

این تابع مجموع مقادیر ستون n از جدول مورد نظر را محاسبه می کند.

نکته: دو تابع SUM و AVG برای داده های از نوع عدد استفاده می شوند.

****تابعMIN**

```
MIN([DISTINCT|ALL]expr)
```

حداقل مقدار موجود در ستون expr را برمی گرداند.

****تابعMAX**

```
MAX([DISTINCT|ALL]expr)
```

حداکثر مقدار در ستون expr را برمی گرداند.

نکته: از توابع MIN و MAX برای ستون های با نوع داده کاراکتری، عدد یا تاریخ استفاده می شود.

مثال: تعیین میانگین، مجموع، حداکثر و حداقل حقوق دریافتی کارمندانی که JOB_ID آنها مشابه REP می باشد.

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

	AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
1	8150	11000	6000	32600

مثال: تاریخ استخدام اولین و آخرین کارمند سازمان.

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

	MIN(HIRE_DATE)	MAX(HIRE_DATE)
1	13-JAN-01	29-JAN-08

****تابعCOUNT**

```
COUNT([DISTINCT|ALL]expr)
```

از این تابع برای شمارش استفاده می شود. اگر از علامت * بجای expr استفاده شود تعداد کل سطرهای متناظر با آن دستور باز می گردد حتی اگر تعدادی از آنها NULL باشند. ولی می توان بجای expr از نام یکی از ستون ها استفاده نمود که در این حالت تعداد سطرهای غیر NULL آن ستون را بازمی گرداند.

نکته: خروجی تابع COUNT با هر ورودی که نام ستون جدول نیست برابر با خروجی تابع COUNT(*) است. مانند COUNT('sds') ، COUNT(1) و ...

مثال: چند سطر از جدول employees دارای شماره دپارتمان ۵۰ است.

1
```
SELECT COUNT(*)
FROM   employees
WHERE  department_id = 50;
```

	COUNT(*)
1	5

مثال: چند نفر در دپارتمان شماره ۵۰ حق کمیسیون دریافت کرده اند(مقدار غیرNULL دارند).

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

	COUNT(COMMISSION_PCT)
1	0

نکته: اگر در تابع COUNT از کلمه DISTINCT استفاده شود فقط تعداد مقادیر غیر تکراری شمرده می شوند.

مثال: نمایش تعداد دپارتمان های سازمان(به صورت غیر تکراری)

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

	COUNT(DISTINCTDEPARTMENT_ID)
1	7

نکته: تمام توابع گروهی سطرهایی که مقادیر NULL دارند را رد می کنند و درنظر نمی گیرند.

مثال: در قسمت اول میانگین مقادیر ستون حق کمیسون محاسبه می گردد در حالی که سطرهای NULLدر عملیات محاسبه میانگین قرار ندارند. ولی در قسمت دوم با استفاده از تابع NVL تمام سطرهای NULL به مقدار 0تبدیل شده و سپس میانگین تمام سطرها محاسبه شده است.

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

	AVG(COMMISSION_PCT)
1	0.2125

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

	AVG(NVL(COMMISSION_PCT,0))
1	0.0425

۵.۲ عبارت GROUP BY

در دستورات SQL با استفاده از GROUP BY اطلاعات یک جدول را به چند دسته مختلف تقسیم می کنیم. این دسته بندی بر اساس ستون یا ستون هایی که نام آنها بعد از عبارت GROUP BY بیان می شود انجام می گیرد.

نکته: در عبارت GROUP BY نمی توان از نام مستعار برای ستون ها استفاده نمود.

نکته: در دستورات SQL نام تمام ستون هایی که بعد از کلمه SELECTاستفاده می شوند باید در عبارت GROUP BY نیز بکار رود مگر آنکه نام یک ستون در یک تابع گروهی استفاده شده باشد.

مثال: نام ستون DEPARTMENT_ID که بعد از کلمه کلیدی SELECTاستفاده شده در عبارت GROUP BY نیز بکار رفته است. همچنین در این دستور هر کدام از دپارتمان ها به عنوان یک گروه مجزا در نظر گرفته می شوند و سپس میانگین حقوق در هر دپارتمان محاسبه می گردد.

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

	DEPARTMENT_ID	AVG(SALARY)
1	(null)	7000
2	90	19333.333333333333333333333333333333333
3	20	9500
4	110	10154
5	50	3500
6	80	10033.333333333333333333333333333333333
7	60	6400
8	10	4400

۵.۳ عبارت ORDER BY

با استفاده از عبارت ORDER BY می توان سطرهای انتخاب شده را به صورت صعودی(ASC) و یا نزولی (DESC) نمایش داد. عبارت ORDER BY بعد از عبارت WHERE استفاده می شود.

مثال: اطلاعات کارمندان بر اساس ترتیب تاریخ استخدام مرتب شده و نمایش می یابد.

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

	LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
1	De Haan	AD_VP	90	13-JAN-01
2	Gietz	AC_ACCOUNT	110	07-JUN-02
3	Higgins	AC_MGR	110	07-JUN-02
4	King	AD_PRES	90	17-JUN-03
5	Whalen	AD_ASST	10	17-SEP-03
6	Rajs	ST_CLERK	50	17-OCT-03

نکته: اگر در عبارت ORDER BY ترتیب صعودی یا نزولی مشخص نگردد به صورت پیش فرض نمایش اطلاعات به صورت صعودی یا ASC خواهد بود.

نکته: اگر بعد از عبارت ORDER BY چندین ستون ذکر شود، ابتدا مرتب سازی بر اساس ستون اول انجام می گیرد و سپس اگر مقادیر ستون اول برای سطرها یکسان بود از ستون بعدی جهت مرتب سازی استفاده می گردد.

مثال:

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

4

نکته: برخلاف عبارت WHERE در عبارت ORDER BY می توان از نام مستعار (ALIAS) یک ستون نیز استفاده نمود. **مثال:**

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

2

نکته: هر کدام از مقادیر عددی، تاریخ یا کاراکتری می توانند توسط عبارت ORDER BY مرتب شوند.

نکته: اگر ORDER BY به صورت صعودی باشد مقادیر NULL در انتهای سطرها نمایش می یابند و اگر نزولی باشد در ابتدای سطرها.

نکته: می توان عبارت ORDER BY را براساس یک ستون که در کل دستور SELECT به کار نرفته است نیز استفاده کرد.

نکته: می توان بعد از عبارت ORDER BY یک عدد عنوان کرد که منظور از آن شماره ستون انتخاب شده در دستور SELECT خواهد بود.

مثال: ستون DEPARTMENT_ID به عنوان سومین ستون در دستور SELECT انتخاب شده است بنابراین منظور از شماره ۳ بعد از عبارت ORDER BY ستون DEPARTMENT_IDاست.

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY 3;
```

نکته: در اوراکل نگارش ۱۲ به بعد می توان در دستور SELECT و بعد از عبارت ORDER BY از عباراتی جهت تعیین حداکثر تعداد سطرهایی که می خواهیم نمایش یابند استفاده نمود. همچنین می توان شماره سطری که می خواهیم نمایش اطلاعات از آن محل شروع شود نیز مشخص کرد (OFFSET,FETCH).

مثال:

```
SELECT employee_id, first_name
FROM employees
ORDER BY employee_id
FETCH FIRST 5 ROWS ONLY;
```

Script Output | Query Result | SQL | All Rows Fetched: 5

	EMPLOYEE_ID	FIRST_NAME
1	100	Steven
2	101	Neena
3	102	Lex
4	103	Alexander
5	104	Bruce

```
SELECT employee_id, first_name
FROM employees
ORDER BY employee_id
OFFSET 5 ROWS FETCH NEXT 5 ROWS ONLY;
```

	EMPLOYEE_ID	FIRST_NAME
1	107	Diana
2	124	Kevin
3	141	Trenna
4	142	Curtis
5	143	Randall

نکته: اگر بعد ار عبارت GROUP BY از عبارت ORDER BY استفاده شود نتایج به صورت صعودی و یا نزولی نمایش می یابند.

مثال:

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

مثال:

```
SELECT department, MAX(salary) AS "Highest salary" FROM employees
GROUP BY department order by salary desc;
```

نکته: اگر در یک دستور SQL نام یک ستون در عبارت GROUP BY استفاده شود لزومی ندارد از آن ستون بعد از کلمه کلیدی SELECT نیز استفاده گردد.

نکته: می توان از چندین ستون در عبارت GROUP BY استفاده نمود و جدول را بر اساس آن ستون ها دسته بندی نمود.

مثال: مطابق این دستور ابتدا جدول EMPLOYEE بر اساس شماره دپارتمان دسته بندی می شود و سپس در هر دپارتمان افرادی که JOB_ID یکسان دارند در یک گروه قرار می گیرند.

```
SELECT   department_id, job_id, sum(salary) FROM     employees
GROUP BY department_id, job_id ORDER BY job_id;
```

EMPLOYEES

	DEPARTMENT_ID	JOB_ID	SALARY
1	10	AD_ASST	4400
2	20	MK_MAN	13000
3	20	MK_REP	6000
4	50	ST_CLERK	2500
5	50	ST_CLERK	2600
6	50	ST_CLERK	3100
7	50	ST_CLERK	3500
8	50	ST_MAN	5800
9	60	IT_PROG	9000
10	60	IT_PROG	6000
11	60	IT_PROG	4200
12	80	SA_REP	11000
13	80	SA_REP	8600
14	80	SA_MAN	10500
...			
19	110	AC_MGR	12000
20	(null)	SA_REP	7000

Add the salaries in the EMPLOYEES table for each job, grouped by department.

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	110	AC_ACCOUNT	8300
2	110	AC_MGR	12008
3	10	AD_ASST	4400
4	90	AD_PRES	24000
5	90	AD_VP	34000
6	60	IT_PROG	19200
7	20	MK_MAN	13000
8	20	MK_REP	6000
9	80	SA_MAN	10500
10	80	SA_REP	19600
11	(null)	SA_REP	7000
12	50	ST_CLERK	11700
13	50	ST_MAN	5800

مثال: استفاده از چند ستون در GROUP BY و استفاده از عبارت.ORDER BY

```
SELECT    department_id, job_id, SUM(salary)
FROM      employees
WHERE     department_id > 40
GROUP BY department_id, job_id
ORDER BY department_id;
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	50	ST_CLERK	11700
2	50	ST_MAN	5800
3	60	IT_PROG	19200
4	80	SA_MAN	10500
5	80	SA_REP	19600
6	90	AD_PRES	24000
7	90	AD_VP	34000
8	110	AC_ACCOUNT	8300
9	110	AC_MGR	12008

نکته: اگر در یک دستور SQL بعد از کلمه کلیدی SELECT یک تابع گروهی و نام یک ستون مثلا exprبکار رفته باشد باید نام ستون expr در یک عبارت GROUP BY استفاده شود وگرنه آن دستور اجرا نمی شود و از طرف اوراکل خطا دریافت می کنیم.

مثال: در دستور اول نام ستون DEPARTMENT_ID و در دستور دوم ستون JOB_ID در عبارت GROUP BY استفاده نشده است.

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

```
ORA-00937: not a single-group group function
00937. 00000 - "not a single-group group function"
```

A GROUP BY clause must be added to count the last names for each department_id.

```
SELECT department_id, job_id, COUNT(last_name)
FROM   employees
GROUP BY department_id;
```

```
ORA-00979: not a GROUP BY expression
00979. 00000 - "not a GROUP BY expression"
```

Either add job_id in the GROUP BY or remove the job_id column from the SELECT list.

نکته: نمی توان از توابع گروهی در قسمت WHERE یک دستور SELECT استفاده نمود.

مثال: نمایش میانگین حقوق دپارتمان هایی که از ۸۰۰۰ بیشتر هستند. بدلیل استفاده از تابع گروهی در قسمت WHERE خطای اوراکل دریافت می کنیم.

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

```
ORA-00934: group function is not allowed here
00934. 00000 - "group function is not allowed here"
*Cause:
*Action:
Error at Line: 3 Column: 9
```

۵.۴ عبارت HAVING

زمانی که از عبارت GROUP BY جهت گروه بندی یک جدول استفاده می شود برای محدودسازی گروه ها از HAVING استفاده می گردد. بنابراین مفهوم عبارت HAVING وWHERE یکسان است ولی کاربرد متفاوت دارند. عبارت WHERE سطرهای جدول را محدود می کند ولی عبارت HAVING به منظور محدودسازی گروه های جدول استفاده می شود.

نکته: فقط در دستوری که عبارت GROUP BY بکار رفته است می توان از HAVING استفاده نمود.

نکته: دیتابیس اوراکل در زمان مشاهده یک دستور با عبارت HAVING به ترتیب این ۳ مرحله را اجرا می کند:

- سطرهای جدول با توجه به عبارت GROUP BY گروه بندی می شوند.
- اگر یک تابع گروهی استفاده شده باشد در گروه ها اعمال می شود.
- گروه هایی که مطابق عبارت HAVING محدود شده اند نمایش می یابند.

مثال: گروه هایی که حداکثر حقوق دریافتی در آنها بیشتر از ۱۰۰۰۰ واحد می باشد نمایش می یابند

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

	DEPARTMENT_ID	MAX(SALARY)
1	90	24000
2	20	13000
3	110	12008
4	80	11000

مثال: همزمان از عبارت WHERE جهت محدودسازی سطرها و از عبارت HAVING جهت محدودسازی گروه ها استفاده شده است.

```
SELECT   job_id, SUM(salary) PAYROLL
FROM     employees
WHERE    job_id NOT LIKE '%REP%'
GROUP BY job_id
HAVING   SUM(salary) > 13000
ORDER BY SUM(salary);
```

	JOB_ID	PAYROLL
1	IT_PROG	19200
2	AD_PRES	24000
3	AD_VP	34000

۵.۵ توابع گروهی تودرتو

می توان در دستورات SQL از توابع گروهی به صورت تودرتو (حداکثر ۲ مورد تابع) استفاده نمود. در این حالت استفاده از عبارت GROUP BY اجباری می باشد.

مثال: نمایش حداکثر میانگین حقوق در بین تمام دپارتمان ها.

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

	MAX(AVG(SALARY))
1	19333.333333333333333333333333333333333

۶ روش های مختلف JOIN در SQL

گاهی اوقات در گزارش گیری از دیتابیس باید اطلاعات مورد نیاز از جدول های مختلف در کنار هم نمایش یابند، بنابراین در این موارد عملیات JOIN بین جداول انجام می شود. در این فصل روش های مختلف JOIN که با استاندارد SQL:1999 سازگار می باشند توضیح داده می شوند.

این روش ها را در سینتکس زیر مشاهده می کنید.

```
SELECT   table1.column, table2.column
FROM     table1
[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 table2];
```

۶.۱ NATURAL JOIN

از روش NATURAL JOIN فقط برای JOIN دو جدول استفاده می شود. وقتی از این روش استفاده می کنیم می بایست یک ستون مشترک در هر دو جدول وجود داشته باشد. در این روش فقط سطرهایی نمایش می یابند که **ستون های مشترک** آنها دارای مقدار یکسان باشند.

منظور از ستون های مشترک بین دو جدول ستون هایی می باشند که **نام** و **نوع داده** یکسان دارند. اگر نام این ستون ها یکسان باشد ولی نوع داده آنها متفاوت باشد در زمان اجرای دستور خطایی را از اوراکل دریافت می کنیم.

مثال: دو جدول EMPLOYEES و JOBS با هم NATURAL JOIN شده اند. در هر دو جدول ستون JOB_ID دارای نام و نوع داده یکسان می باشد. بنابراین سطرهایی از دو جدول که مقدار این ستون برای آنها یکسان است در کنار هم نمایش می یابند.

```
SELECT employee_id, first_name, job_id, job_title
from employees NATURAL JOIN jobs;
```

	EMPLOYEE_ID	FIRST_NAME	JOB_ID	JOB_TITLE
1	100	Steven	AD_PRES	President
2	101	Neena	AD_VP	Administration Vice President
3	102	Lex	AD_VP	Administration Vice President
4	103	Alexander	IT_PROG	Programmer
5	104	Bruce	IT_PROG	Programmer
6	105	David	IT_PROG	Programmer
7	106	Valli	IT_PROG	Programmer
8	107	Diana	IT_PROG	Programmer
9	108	Nancy	FI_MGR	Finance Manager
10	109	Daniel	FI_ACCOUNT	Accountant
11	110	John	FI_ACCOUNT	Accountant

...

نکته : در روش NATURAL JOIN می توان از عبارت WHERE به منظور محدودسازی نمایش اطلاعات استفاده نمود.

مثال:

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

۶.۲ روش JOIN با استفاده از عبارت USING

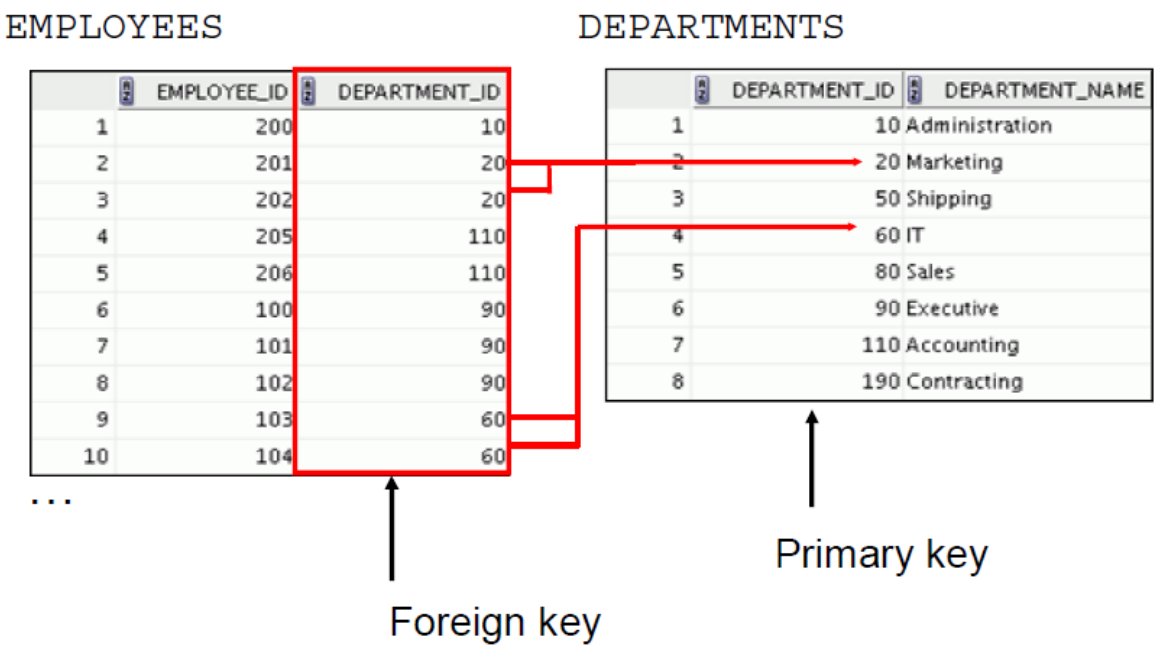
برای اجرای NATURAL JOIN باید تمام ستون های مشترک دارای نام و نوع داده یکسان باشند. اما در روش JOIN با استفاده از عبارت USING **یک یا چند ستون** از دو جدول به عنوان شرط عملیات JOIN انتخاب می شوند که **نام یکسان** دارند ولی می توانند **نوع داده متفاوت** داشته باشند.

نکته: از این روش معمولا برای دو جدول که رابطه از نوع PRIMARY KEY و FOREIGN KEY دارند استفاده می شود.

مثال: نام دپارتمان مربوط به تمام کارمندان شرکت را نمایش دهید.

در این مثال نام دپارتمان کارمندان در جدول DEPARTMENTS ذخیره شده است و سایر اطلاعات کارمندان در جدول EMPLOYEES می باشد. بنابراین با انجام عملیات JOIN این دو جدول بر اساس ستون DEPARTMENT_ID اطلاعات مورد نیاز نمایش می یابد.

```
SELECT EMPLOYEE_ID, DEPARTMENT_NAME FROM employees JOIN
Departments USING (DEPARTMENT_ID);
```



مثال: در این مثال LOCATION_ID هر کارمند در جدول DEPARTMENTS قرار دارد.

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

	EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_ID
1	200	Whalen	1700	10
2	201	Hartstein	1800	20
3	202	Fay	1800	20
4	144	Vargas	1500	50
5	143	Matos	1500	50
6	142	Davies	1500	50
7	141	Rajs	1500	50
8	124	Mourgos	1500	50

...

18	206	Gietz	1700	110
19	205	Higgins	1700	110

نکته: اگر نام جدول را به همراه یک نقطه قبل از نام ستون قرار دهیم سرعت عملیات PARSING در دیتابیس اوراکل بیشتر می شود.

نکته: زمانی که در یک دستور دارای JOIN نام یک ستون در چند جدول یکسان می باشد (بجز نام ستون بکار رفته در عبارت USING یا در روش NATURAL JOIN) ذکر نام جدول یا ALIAS قبل از نام ستون اجباری می باشد تا اوراکل متوجه شود این ستون مربوط به کدام جدول است.

مثال: در این دستور هر دو جدول دارای ستون MANAGER_ID می باشند بنابراین باید قبل از نام این ستون از پیشوند نام جدول مورد نظر استفاده شود.

```
SELECT first_name, e.department_name, d.manager_id
FROM   employees e JOIN departments d USING (department_id)
WHERE  department_id = 50;
```

نکته: به منظور افزایش خوانایی و افزایش سرعت نوشتن کد می توان از نام مستعار برای نام جداول استفاده نمود. این نام مستعار فقط در داخل همان دستور SQL معتبر خواهد بود .در ضمن اگر در قسمت FROM دستور SELECT از نام مستعار برای یک جدول استفاده شود این نام باید در تمامی بخش های دیگر آن دستور استفاده شود. البته برای نام ستونی که در عبارت USING یا در روش NATURAL JOIN مشخص شده است نمی توان از از نام مستعار یا پیشوند نام جدول استفاده نمود.

مثال :

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

مثال: ستون location_id در عبارت USING استفاده شده است. بنابراین زمانی که برای این ستون از پیشوند نام جدول استفاده می شود اوراکل خطای ۲۵۱۵۴ برمی گرداند.

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

```
ORA-25154: column part of USING clause cannot have qualifier
25154. 00000 - "column part of USING clause cannot have qualifier"
*Cause:    Columns that are used for a named-join (either a NATURAL join
           or a join with a USING clause) cannot have an explicit qualifier.
*Action:   Remove the qualifier.
Error at Line: 4 Column: 6
```

۶.۳ روش JOIN با استفاده از عبارت ON

روش های NATURAL و USING به صورت ضمنی شرط مساوی بودن مقدار ستون های دو جدول که نام یکسان دارند را در نظر می گیرند. ولی در این روش می توان شرط عمل JOIN را به صورت **صریح** بیان نمود و از **ستون های با نام** و **نوع داده متفاوت** استفاده کرد.

مثال: فقط سطرهایی نمایش یابند که مقدار DEPARTMENT_ID آنها در دو جدول برابر است.

```
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_1	LOCATION_ID
1	200	Whalen	10	10	1700
2	201	Hartstein	20	20	1800
3	202	Fay	20	20	1800
4	124	Mourgos	50	50	1500
5	144	Vargas	50	50	1500
6	143	Matos	50	50	1500
7	142	Davies	50	50	1500
8	141	Rajs	50	50	1500
9	107	Lorentz	60	60	1400
10	104	Ernst	60	60	1400
11	103	Hunold	60	60	1400

نکته: از آنجایی که در شرط ON نام ستون ها می توانند متفاوت باشند از نام مستعار و پیشوند نام جدول قبل از نام ستون استفاده می شود .البته استفاده از علامت پرانتز برای عبارت ON اختیاری می باشد.

مثال: JOIN سه جدول با استفاده از روش ON.

```
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
1	100	Seattle	Executive
2	101	Seattle	Executive
3	102	Seattle	Executive
4	103	Southlake	IT
5	104	Southlake	IT
6	107	Southlake	IT
7	124	South San Francisco	Shipping
8	141	South San Francisco	Shipping
9	142	South San Francisco	Shipping

...

کد مثال بالا با استفاده از روش USING به صورت زیر می باشد.

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

نکته: در **روش JOIN با عبارت ON** می توان از عبارت WHERE و یا AND به منظور تعیین شرطهای دیگر استفاده نمود.

مثال: این کدها نتیجه یکسان برمی گردانند.

```
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 ;
```

Or

```
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)
WHERE   e.manager_id = 149 ;
```

۶.۴ روش SELF JOIN

گاهی اوقات به منظور بدست آوردن برخی نتایج باید یک جدول با خودش JOIN شود. به عنوان مثال می خواهیم نام مسئول مستقیم یکی از کارمندان شرکت را پیدا کنیم.

نام و اطلاعات فردی تمامی پرسنل در جدول EMPLOYEES قرار دارد. بنابراین با استفاده از عمل **JOIN SELF** در جدول EMPLYEES در مرحله اول از ستون های LAST_NAME و MANAGER_ID نام کارمند و شماره پرسنلی مسئول مستقیم آن شخص را بدست می آوریم و در مرحله دوم با استفاده از مقدار MANAGER_ID، نام آن مسئول که یکی از کارمندان می باشد را از ستون LAST_NAME نمایش می دهیم.

مثال: نام خانوادگی مسئول مستقیم تمام کارمندان شرکت نمایش یابد. بنابراین جدول EMPLOYEES با خودش JOIN می شود.

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

	EMP	MGR
1	Hunold	De Haan
2	Fay	Hartstein
3	Gietz	Higgins
4	Lorentz	Hunold
5	Ernst	Hunold
6	Zlotkey	King
7	Mourgos	King
8	Kochhar	King

۶.۵ NONEQUIJOIN یا JOIN نامساوی

در این روش برخلاف روش های قبلی از شرط مساوی در JOIN استفاده نمی شود و می توان از عملگرهای BETWEEN، منطقی، ریاضی و ... استفاده نمود. بنابراین روش های قبلی از نوع EQUIJOIN بودند و این روش از نوع NONEQUIJOIN است.

مثال: در جدول JOB_GRADE به ازای میزان حقوق دریافتی، GRADE_LEVELهای مختلف تعیین شده است. می خواهیم GRADE_LEVEL مورد نظر برای تمام کارمندان بر اساس میزان حقوق دریافتی آنها را نمایش دهیم. بنابراین در شرط JOIN از عملگر BETWEEN استفاده می شود.

EMPLOYEES

	LAST_NAME	SALARY
1	Whalen	4400
2	Hartstein	13000
3	Fay	6000
4	Higgins	12000
5	Gietz	8300
6	King	24000
7	Kochhar	17000
8	De Haan	17000
9	Hunold	9000
10	Ernst	6000
...		
19	Taylor	8600
20	Grant	7000

JOB_GRADES

	GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
1	A	1000	2999
2	B	3000	5999
3	C	6000	9999
4	D	10000	14999
5	E	15000	24999
6	F	25000	40000

The JOB_GRADES table defines the LOWEST_SAL and HIGHEST_SAL range of values for each GRADE_LEVEL. Therefore, the GRADE_LEVEL column can be used to assign grades to each employee.

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

	LAST_NAME	SALARY	GRADE_LEVEL
1	Vargas	2500	A
2	Matos	2600	A
3	Davies	3100	B
4	Rajs	3500	B
5	Lorentz	4200	B
6	Whalen	4400	B
7	Mourgos	5800	B
8	Ernst	6000	C
9	Fay	6000	C
10	Grant	7000	C
...			

نکته: عملگر BETWEEN همان عملگر AND است.

۶.۶ روش **JOIN** با استفاده از عمگر **OUTER**

براساس استاندارد SQL:1999 تمامی روش های قبلی JOIN از نوع INNER JOIN می باشند به این معنی که در این روش ها فقط سطرهایی از جداول که شرط JOIN در مورد آنها برقرار بود نمایش می یابند. ولی در روش OUTER سطرهای دیگر جدول که مطابق با شرطJOIN نیستند نیز نمایش می یابند. بنابراین برای جداول از نام مستعار و برای ستون ها از پیشوند نام جدول استفاده می شود.

روش OUTER JOIN می تواند به صورت **LEFT**، **RIGHT**یا **FULL**انجام شود.

نکته: در روش LEFT OUTER JOIN علاوه بر حالت عادی، سطرهایی از جدول سمت چپ که شرط JOINدر مورد آنها صدق نمی کند نیز نمایش می یابند. منظور از جدول سمت چپ جدولی می باشد که نام آن در سمت چپ عبارت JOIN ذکر شده است.

مثال: نام، شماره دپارتمان و نام دپارتمان تمام کارمندان نمایش یابد. نام تمام کارمندان از جدول سمت چپ نمایش یابد حتی اگر مقدار شماره دپارتمان یک کارمند از جدول EMPLOYEES با مقدار شماره دپارتمان در جدول DEPARTMENTS برابر نیست.

```
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
1	Whalen	10	Administration
2	Fay	20	Marketing
3	Hartstein	20	Marketing
4	Vargas	50	Shipping
5	Matos	50	Shipping
...			
16	Kochhar	90	Executive
17	King	90	Executive
18	Gietz	110	Accounting
19	Higgins	110	Accounting
20	Grant	(null)	(null)

نکته: در روش RIGHT OUTER JOIN علاوه بر حالت عادی، سطرهایی از جدول سمت راست که شرط JOIN در مورد آنها صادق نمی باشد نیز نمایش می یابند. منظور از جدول سمت راست جدولی می باشد که نام آن در سمت راست عبارت JOIN ذکر شده است.

مثال: نام، شماره دپارتمان و نام دپارتمان تمام کارمندان نمایش یابد. نام دپارتمان و شماره دپارتمان تمام کارمندان از جدول سمت راست نمایش یابد حتی اگر مقدار شماره دپارتمان یک کارمند از جدول EMPLOYEES با مقدار شماره دپارتمان در جدول DEPARTMENTSبرابر نیست

```
SELECT e.last_name, d.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
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Davies	50	Shipping
5	Vargas	50	Shipping
6	Rajs	50	Shipping
7	Mourgos	50	Shipping
8	Matos	50	Shipping

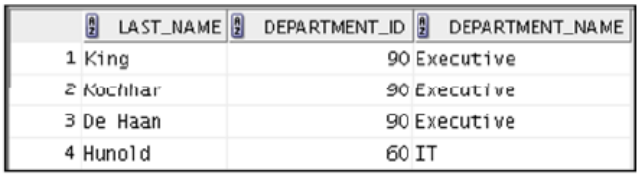
...

18	Higgins	110	Accounting
19	Gietz	110	Accounting
20	(null)	190	Contracting

نکته: در روش FULL OUTER JOIN علاوه بر حالت عادی، سطرهایی از جدول **سمت راست و سمت چپ** که شرط JOIN در مورد آنها صدق نمی کند نیز نمایش می یابند.

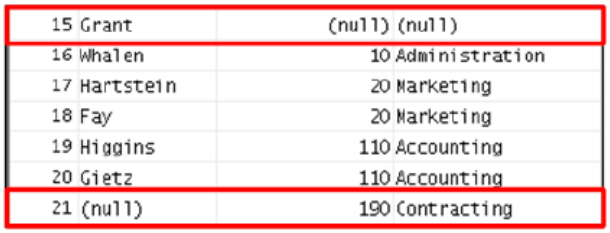
مثال: نام، شماره دپارتمان و نام دپارتمان تمام کارمندان نمایش یابد .اگر مقدار شماره دپارتمان یک کارمند از جدل EMPLOYEES با مقدار شماره دپارتمان در جدول DEPARTMENTS برابر نیست سطرهای متناظر از هر دو جدول سمت چپ و راست نمایش یابند.

```
SELECT e.last_name, d.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
1	King	90	Executive
2	Kochhar	90	Executive
3	De Haan	90	Executive
4	Hunold	60	IT

...



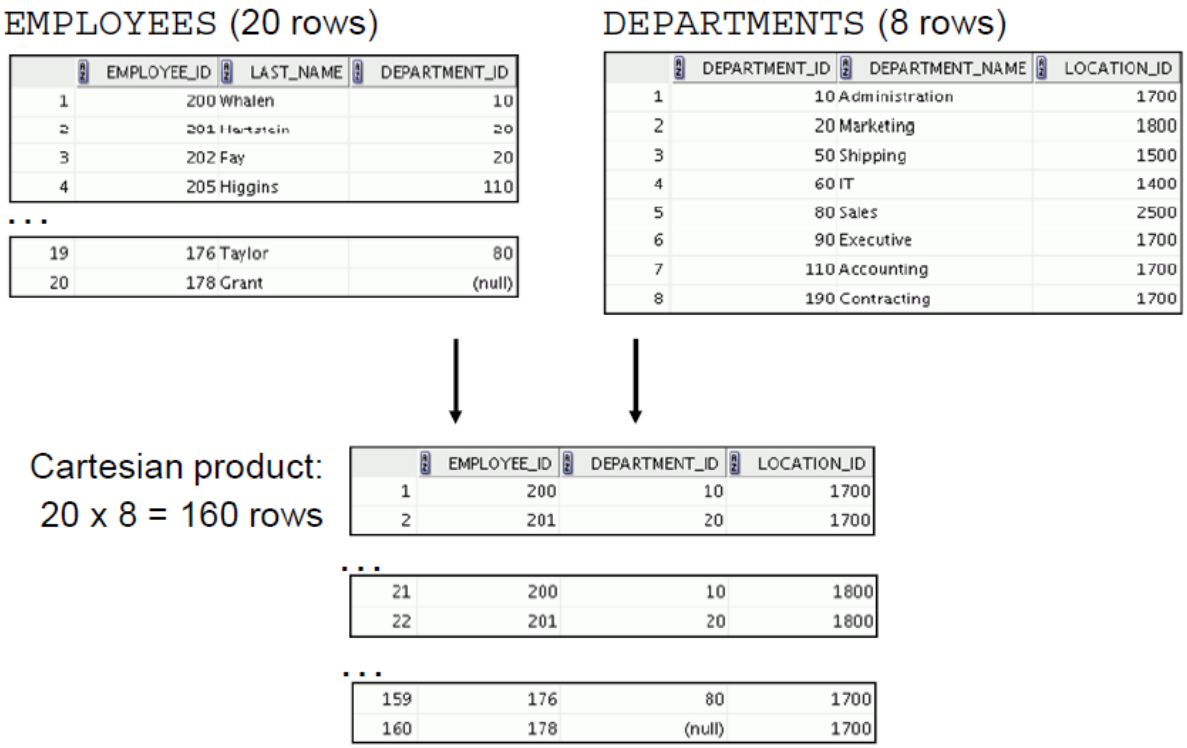
15	Grant	(null)	(null)
16	Whalen	10	Administration
17	Hartstein	20	Marketing
18	Fay	20	Marketing
19	Higgins	110	Accounting
20	Gietz	110	Accounting
21	(null)	190	Contracting

۶.۷ روش CROSS JOIN

اگر در یک دستور SQL از این روش برای JOIN دو جدول استفاده شود تمام سطرهای این دو جدول با هم JOIN می شوند و در خروجی نمایش می یابند. این روش معمولا در برخی موارد تست و آزمون که نیاز به نمایش تعداد زیادی از سطرها می باشد استفاده می گردد.

مثال: جدول EMPLOYEES شامل ۲۰ سطر می باشد و جدول DEPARTMENTS 8 سطر دارد. بنابراین در CROSS JOIN این دو جدول ۱۶۰ سطر نمایش می یابند.

```
SELECT   EMPLOYEE_ID,   DEPARTMENT_ID,   LOCATION_ID   FROM
EMPLOYEES CROSS JOIN DEPARTMENTS;
```



	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	200	Whalen	10
2	201	Hartstein	20
3	202	Fay	20
4	205	Higgins	110
...			
19	176	Taylor	80
20	178	Grant	(null)

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
1	10	Administration	1700
2	20	Marketing	1800
3	50	Shipping	1500
4	60	IT	1400
5	80	Sales	2500
6	90	Executive	1700
7	110	Accounting	1700
8	190	Contracting	1700

	EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
1	200	10	1700
2	201	20	1700
...			
21	200	10	1800
22	201	20	1800
...			
159	176	80	1700
160	178	(null)	1700

۶.۸ روش غیر استاندارد

تمام روش هایی که در این فصل توضیح داده شد مطابق با روش استاندارد است. همچنین می توان از روش غیر استاندراد بجای این روش ها استفاده نمود. در ادامه یکی از مثال های روش USING با روش غیر استاندارد انجام شده است.

روش USING

```
SELECT EMPLOYEE_ID, DEPARTMENT_NAME FROM employees JOIN
Departments USING (DEPARTMENT_ID);
```

روش غیر استاندارد

```
SELECT EMPLOYEE_ID, DEPARTMENT_NAME FROM employees a,
Departments b where a.DEPARTMENT_ID = b.DEPARTMENT_ID;
```

۷ SUBQUERY در SQL

گاهی اوقات لازم است یک دستور SQL طوری نوشته شود که در داخل آن دستوراتSQL ای دیگری قرار بگیرند و با اجرای دستورات درونی، نتیجه لازم به دستور اصلی برگردد. در این حالت دستور اول Main Query نام دارد و دستورات داخلی را SUBQUERY می نامند.هر SUBQUERY نتیجه لازم را برای Main Query ارسال می کند. در این فصل روش های مختلف استفاده از SUBQUERY در دستورات SQL توضیح داده می شود.

می توان در قسمت عبارتWHERE ، FROMو یا HAVINGیک دستورSQL ، از یک SUBQUERY استفاده نمود. سینتکس دستور در روش های مختلف SUBQUERY به شکل زیر است که در آن **expr operator** یک شرط مقایسه ای می باشد.

MAIN QUERY

```
SELECT   select_list
FROM     table
WHERE    expr operator          SUBQUERY
                    (SELECT      select_list
                    FROM         table);
```

نکته: در دستورات غیر از SELECT نیز می توان از SUBQUERY استفاده نمود.

مثال: می خواهیم نام و تاریخ استخدام کارمندانی که بعد از کارمند Davis استخدام شده اند را نمایش دهیم. بنابراین یک SUBQUERY جهت مشخص نمودن زمان استخدام کارمند Davis و یک Main Query برای نمایش نام سایر کارمندانی که بعد از Davis استخدام شده اند تعریف می شود.

```
SELECT last_name, hire_date
FROM   employees                29-JAN-05
WHERE  hire_date > (SELECT hire_date
                    FROM   employees
                    WHERE  last_name = 'Davies');
```

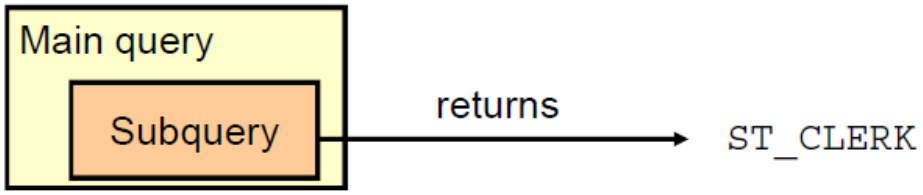
۷.۱ برخی قوانین در استفاده از SUBQUERYها:

- SUBQUERY باید در داخل پرانتز باشد.
- به منظور افزایش خوانایی دستور بهتر است SUBQUERY در سمت راست شرط مقایسه ای قرار گیرد.
- اگر از عملگرهای مقایسه ای که فقط یک سطر را برمی گردانند یعنی Single-Row هستند استفاده می شود از SUBQUERY از نوع Single-Row استفاده شود و در غیر این صورت از SUBQUERY از نوع Multiple-Row استفاده گردد.

۷.۲ انواع روش های SUBQUERY

۷.۲.۱ Single-Row SUBQUERY

- Single-row subquery



درSUBQUERY های از نوع Single-Row فقط یک سطر از جدول به Main QUERY برگرداننده می شود. همچنین باید برای آنها از عملگرهای مقایسه ای Single-Row استفاده نمود. این نوع ازعملگرها در جدول زیر مشخص شده اند.

عملگر	توضیحات
=	برابر با
>	بزرگتر از
>=	بزرگتر از یا مساوی با
<	کوچکتر از
<=	کوچکتر از یا مساوی با
<>	مخالف با

مثال: نمایش نام کارمندانی که job_id آنها برابر با job_id کارمند شماره ۱۴۲ می باشد.

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

	LAST_NAME	JOB_ID
1	Rajs	ST_CLERK
2	Davies	ST_CLERK
3	Matos	ST_CLERK
4	Vargas	ST_CLERK

مثال: نمایش اطلاعات کارمندانی که شغل آنها با شغل کارمند TAYLOR یکسان می باشد ولی حقوق بیشتر از او دریافت می کنند. در این دستور از دو SUBQUERY استفاده شده است.

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id =                          SA_REP
                (SELECT job_id
                 FROM   employees
                 WHERE  last_name = 'Taylor')
AND    salary >                            8600
                (SELECT salary
                 FROM   employees
                 WHERE  last_name = 'Taylor');
```

	LAST_NAME	JOB_ID	SALARY
1	Abel	SA_REP	11000

مثال: نام و حقوق کارمندانی که حقوق آنها برابر با حداقل حقوق دریافتی در شرکت می باشد. در این مثال در داخل SUBQUERY از یک تابع گروهی استفاده شده است.

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

2500

	LAST_NAME	JOB_ID	SALARY
1	Vargas	ST_CLERK	2500

مثال: نمایش نام دپارتمان هایی که در آنها حداقل حقوق دریافتی کمتر از حداقل حقوق در دپارتمان شماره ۳۰ است. در این مثال در عبارت Having از SUBQUERY استفاده شده است.

```
SELECT   department_id, MIN(salary)
FROM     employees
GROUP BY department_id
HAVING   MIN(salary) >
                       (SELECT MIN(salary)
                        FROM   employees
                        WHERE  department_id = 30);
```

2500

	DEPARTMENT_ID	MIN(SALARY)
1	100	6900
2	(null)	7000
3	90	17000
4	20	6000
5	70	10000
6	110	8300
7	80	6100
8	40	6500
9	60	4200
10	10	4400

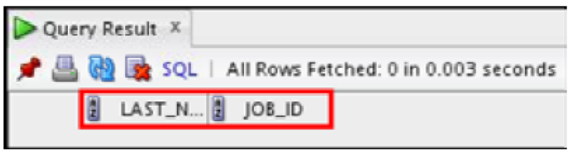
مثال: نمایش شغل هایی که میانگین حقوق دریافتی در آنها برابر با حداقل میانگین حقوق دریافتی شغل ها می باشد.

```
SELECT   job_id, AVG(salary)
FROM     employees
GROUP BY job_id
HAVING   AVG(salary) = (SELECT   MIN(AVG(salary))
                        FROM     employees
                        GROUP BY job_id);
```

نکته: اگر هیچ مقداری برای SUBQUERY وجود نداشته باشد مقدار NULL به MAIN QUERY بازمی گردد.

مثال: هیچ کارمندی با نام 'Haas' وجود ندارد بنابراین مقدار NULL به MAIN QUERY برمی گردد.

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

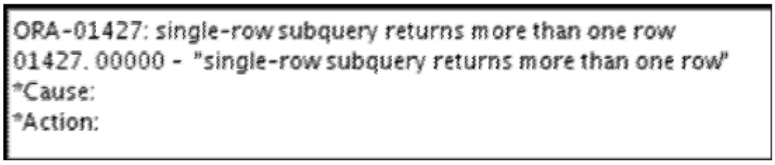


Subquery returns no rows because there is no employee named “Haas.”

نکته: اگر توسط یک Single-Row SUBQUERY بیشتر از یک سطر برگردانده شود خطای اوراکل دریافت می کنیم. بنابراین دراین حالت باید از Multiple-Row SUBQUERY استفاده شود.

مثال: در این مثال بکارگیری عبارت Group By در SUBQUERY سبب می شود به ازای هر گروه یک سطر مجزا برگردد در حالی که عملگر مقایسه ای در MAIN QUERY علامت '=' می باشد که از نوع Single-Row است و نمی تواند چندین مقدار مختلف را بپذیرد .بنابراین باید از Multiple-Row Query و عملگر IN استفاده نمود.

```
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

۷.۲.۲ Multiple-Row SUBQUERY

این روش ازSUBQUERY ها بیشتر از یک سطر از جدول را به Main QUERY برمی گردانند و باید برای آنها از عملگرهای مقایسه ای Multiple-Row استفاده نمود. عملگرهای مقایسه ای از نوع Multiple-Rowرا در جدول زیر می بینید.

عملگر	توضیحات
IN	برابر با هرکدام از مقدارهای داخل لیست IN
ANY	قبل از این عملگر یکی از علامت های =>, =<, <, >, !=, = استفاده می شود باید حداقل یکی از مقدارهای SUBQUERY مطابق شرط مقایسه ای باشد تا مقدار صحیح برگرداننده شود
ALL	قبل از این عملگر یکی از علامت های =>, =<, <, >, !=, = استفاده می شود باید تمام مقدارهای SUBQUERY مطابق شرط مقایسه ای باشد تا مقدار صحیح برگرداننده شود

قبل از عملگرهای مقایسه ای ANY و ALL یکی از عملگرهای =، !=، >، <، <= و >= استفاده می شود. زمانی که از ANY استفاده می شود باید حداقل یکی از مقدارهای SUBQUERY مطابق شرط مقایسه ای باشد تا مقدار صحیح برگرداننده شود ولی در عملگر ALL باید تمام مقدارهای SUBQUERY مطابق شرط مقایسه ای باشد تا مقدار صحیح برگردانده شود.

مثال: نمایش مشخصات کارمندانی که حقوق دریافتی آنها برابر با یکی از حقوق های لسیت IN است.

```
SELECT last_name, salary, department_id
FROM   employees
WHERE  salary IN (2500, 4200, 4400, 6000, 7000, 8300,
                  8600, 17000);
```

مثال: نمایش اطلاعات کارمندانی که در حوزه IT فعالیت نمی کنند و حقوق دریافتی آنها کمتر از هرکدام از **کارمندان حوزه IT کمتر است. در این مثال حقوق کارمندان حوزه IT برابر با ۶۰۰۰ ، ۹۰۰۰ و ۴۲۰۰ است.**

```
SELECT employee_id, last_name, job_id, salary
FROM   employees        9000, 6000, 4200
WHERE  salary < ANY
                      (SELECT salary
                       FROM   employees
                       WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	144	Vargas	ST_CLERK	2500
2	143	Matos	ST_CLERK	2600
3	142	Davies	ST_CLERK	3100
4	141	Rajs	ST_CLERK	3500
5	200	Whalen	AD_ASST	4400
...				
9	206	Gietz	AC_ACCOUNT	8300
10	176	Taylor	SA_REP	8600

نکته: زمانی که در یک SUBQUERY یکی از مقدارهای داخلی برابر با مقدار NULL باشد و از عملگر مقایسه ای NOT IN استفاده شود، تمام مقدارهای بازگشتی برای آن SUBQUERY برابر با مقدار NULLخواهد بود.

مثال: نام کارمندانی که مدیر نیستند نمایش یابد. در این مثال به دلیل اینکه در SUBQUERY یکی از رکوردهای ستون manager_id دارای مقدار NULL می باشد و از عملگر NOT IN استفاده شده است، هیچ سطری برنمیگردد.

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

Query Result

SQL | All Rows Fetched: 0 in 0.051 seconds

LAST_NAME

Subquery returns no rows because one of the values returned by a subquery is null.

مثال: در مثال قبلی با استفاده از یک عبارت WHERE مقدارهای NULL نادیده گرفته می شوند و نتایج مورد نظر برگردانده می شود.

```
SELECT last_name FROM employees
WHERE  employee_id NOT IN
                          (SELECT manager_id
                           FROM   employees
                           WHERE  manager_id IS NOT NULL);
```

مثال: در این مثال با استفاده از عملگر ALL مشخات کارمندانی که حقوق آنها از تمام کارمندان حوزه IT کمتر است نمایش می یابد. حقوق کارمندان حوزه IT برابر با ۶۰۰۰ ، ۹۰۰۰ و ۴۲۰۰ است.

```
SELECT employee_id, last_name, job_id, salary
FROM   employees          9000, 6000, 4200
WHERE  salary < ALL
                    (SELECT salary
                     FROM   employees
                     WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	141	Rajs	ST_CLERK	3500
2	142	Davies	ST_CLERK	3100
3	143	Matos	ST_CLERK	2600
4	144	Vargas	ST_CLERK	2500

۷.۲.۳ Multiple Column SUBQUERY

این دسته از SUBQUERY ها بیشتر از یک ستون از جدول را به Main QUERY برمی گردانند.
همچنین از این روش می توان در قسمت From دستور SELECT نیز استفاده نمود.

مثال: نمایش نام، شماره دپارتمان و حقوق تمام کارمندانی که در هر کدام از دپارتمان ها حداقل حقوق را دریافت می کنند.

```
SELECT first_name, department_id, salary
FROM employees
WHERE (salary, department_id) IN
      (SELECT min(salary), department_id
       FROM employees
       GROUP BY department_id)
ORDER BY department_id;
```

۸ عملگرهای SET در SQL

در دستورات SQL عملگرهای SET سطرهای چندین **دستور SELECT** را ترکیب کرده و ترتیب نمایش آنها را مشخص می کند. در این فصل انواع روش های استفاده از عملگرهای SET توضیح داده می شود.

دستوراتی که در آنها از عملگر SET استفاده می شوند COMPOUND QUERY نام دارند. اولویت اجرای تمام روش های عملگر SET یکسان است. اگر در یک دستور SQL از چند عملگر SET استفاده شود ترتیب اجرای آنها از سمت چپ به راست خواهد بود مگر اینکه از علامت پرانتز استفاده شده باشد.

در جدول زیر، هر یک از عملگرهای SET را مختصرا توضیح دادیم.

نام عملگر	سطرهایی که برمی گرداند
UNION	سطرهای هردو دستور به غیر از سطرهای تکراری بین آنها
UNION ALL	تمام سطرهای هردو دستور شامل سطرهای تکراری بین آنها
INTERSECT	سطرهایی که بین هردو دستور مشترک هستند
MINUS	سطرهایی از دستور اول که در دستور دوم وجود ندارند (تفاضل دستور اول با دوم)

مثال: اتصال دو دستور SELECT با استفاده از یک عملگر SET .

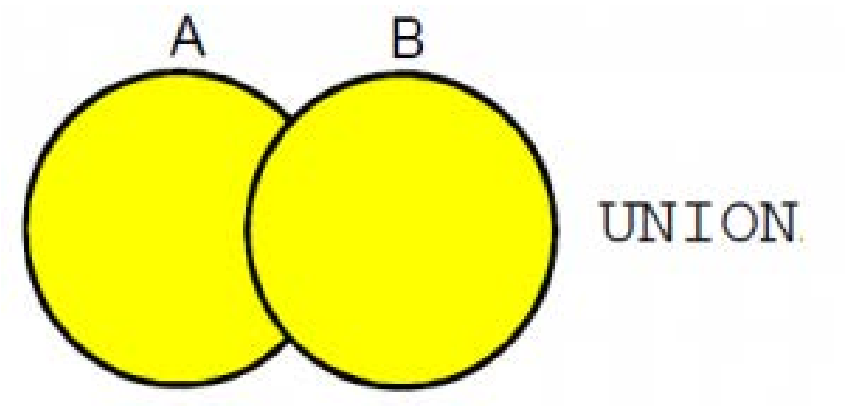
```
SELECT supplier_id FROM suppliers UNION SELECT supplier_id
FROM order_details;
```

۸.۱ نکاتی در مورد عملگرهای SET

- باید دستورات قبل و بعد از عملگر SET از لحاظ **تعداد ستون** و **نوع داده ی آنها** یکسان باشند.
- نام ستون های متناظر در دو دستور SELECT می تواند متفاوت باشد.
- در زمان نمایش اطلاعات، نام ستون ها برابر با نام ستون های انتخاب شده در دستور اول خواهد بود. زمانی که از چند عملگر SET استفاده می شود می توان از علامت پرانتز برای تعیین ترتیب اجرای آنها استفاده نمود.
- می توان فقط یکبار و آن هم در انتهای هر COMPOUND QUERY از عملگر مرتب سازی ORDER BY استفاده کرد.
- می توان از عملگرهای SET در SUBQUERY استفاده نمود.

- در عملگرهای SET به غیر از UNION ALL به صورت پیش فرض خروجی به صورت صعودی نمایش می یابد.
- سطرهای تکراری به صورت اتوماتیک حذف می شوند مگر اینکه از عملگر UNION ALL استفاده شود.
- دیتابیس اوراکل برای دستورات SQL که از عملگرهای SET استفاده می کنند از روش تبدیل نوع داده ی ضمنی استفاده نمی کند. بنابراین اگر نوع داده های ستون های متناظر یکسان نباشد، خطای اوراکل دریافت می کنیم.

۸.۲ عملگر UNION



فرض کنید دو دستور SELECT داریم که هر کدام از آنها تعدادی از سطرهای جدول/جداول را انتخاب می کنند. اگر بین این دو دستور از عملگر UNION استفاده شود تمام سطرهایی که توسط آنها انتخاب شده اند با هم ترکیب شده و نمایش می یابند **البته سطرهایی از دو دستور که یکسان هستند فقط یکبار نمایش می یابند**.

نکته: توجه شود که عملگر UNION فقط برای ستون هایی که در دستورات SELECT انتخاب شده اند عمل می کند.

نکته: مقادیر NULL نیز در زمان بررسی یکسان بودن سطرها در نظر گرفته می شوند.

نکته: در مثال های این متن اگر کارمندان شرکت تغییر شغل داده باشند اطلاعات شغل قبلی آنها در جدول JOB_HISTORY ذخیره می شود.

مثال: اطلاعات مربوط به شغل قبلی و فعلی تمام کارمندان شرکت را نمایش دهید.

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

	EMPLOYEE_ID	JOB_ID
1	100	AD_PRES
2	101	AC_ACCOUNT
...		
22	200	AC_ACCOUNT
23	200	AD_ASST
...		
27	205	AC_MGR
28	206	AC_ACCOUNT

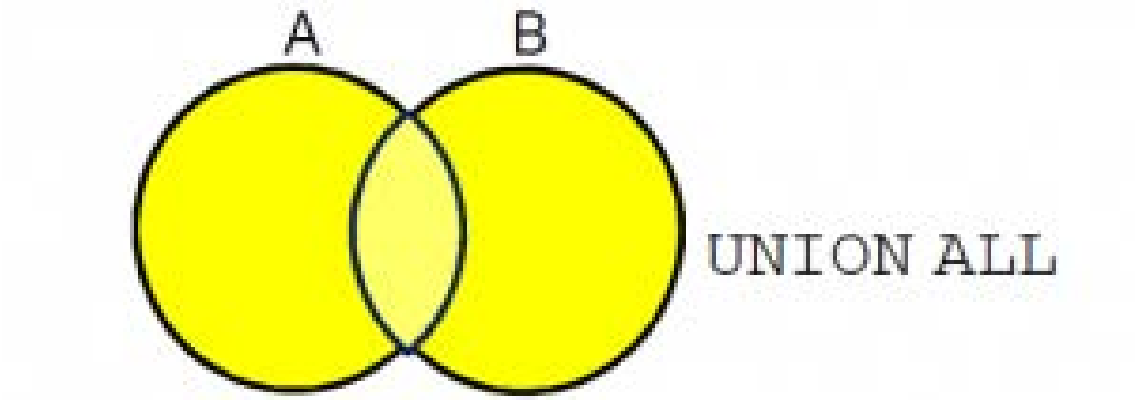
مثال: اگر به در دستورات مثال قبل ستون DEPARTMENT_IDرا اضافه کنیم، اطلاعات کارمندانی که شغل قبلی و فعلی آنها یکسان است ولی قبلا DEPARTMENT_ID متفاوت داشته اند نیز نمایش می یابد.

```
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
1	100	AD_PRES	90
...			
22	200	AC_ACCOUNT	90
23	200	AD_ASST	10
24	200	AD_ASST	90
...			
29	206	AC_ACCOUNT	110

توجه: در مثال های قبلی نتایج بر اساس ستون اول مرتب شده اند.

۸.۳ عملگر UNION ALL



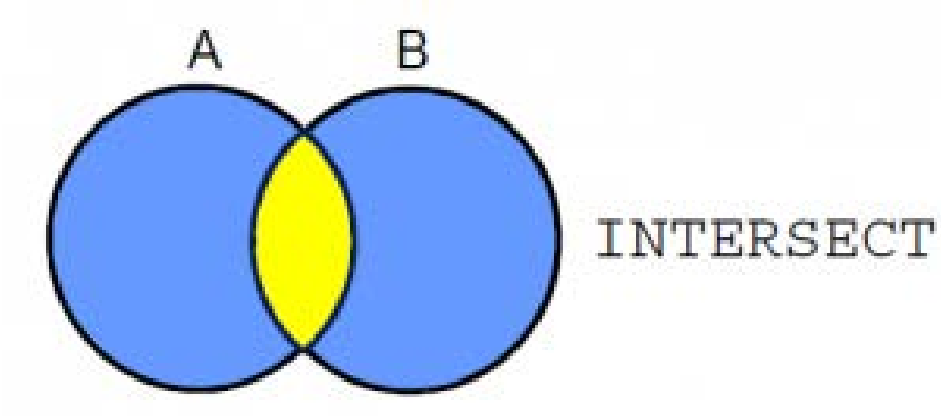
اگر بین دو دستور SELECT از عملگر UNION ALL استفاده شود تمام سطرهای دو دستور حتی آنهایی که یکسان هستند نمایش می یابند.

مثال: تمام اطلاعات کارمندان شرکت حتی اگر شغل متفاوت داشته اند نمایش یابد. در این مثال با استفاده از عملگر UNION ALL سطرهای تکراری نیز نمایش می یابند(سطرهای داخل کادر قرمز رنگ). همچنین در زمان نمایش اطلاعات هیچگونه عمل مرتب سازی انجام نشده است.

```
SELECT employee_id, job_id, department_id
FROM   employees
UNION ALL
SELECT employee_id, job_id, department_id
FROM   job_history
ORDER BY  employee_id;
```

	EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
1	100	AD_PRES	90
...			
17	149	SA_MAN	80
18	174	SA_REP	80
19	176	SA_REP	80
20	176	SA_MAN	80
21	176	SA_REP	80
22	178	SA_REP	(null)
23	200	AD_ASST	10
...			
30	206	AC_ACCOUNT	110

۸.۴ عملگر INTERSECT



این عملگر فقط سطرهایی که توسط هر دو دستور SELECT انتخاب شوند و یکسان باشند را برمی گرداند.

نکته: همانند عملگر های قبلی باید تعداد و نوع داده ی ستون ها یکسان باشد ولی لزومی ندارد نام آنها برابر باشد.

نکته: همانند عملگرهای قبلی مقدارهای NULL در زمان بررسی یکسان بودن سطرها در نظر گرفته می شوند.

نکته: زمانی که از عملگر INTERSECT استفاده می شود بهتر است از پرانتز استفاده گردد تا ترتیب اجرای دستورات به صورت صریح مشخص شود و برای این عملگر اولویت بیشتری درنظر گرفته شود.

مثال: نمایش اطلاعات کارمندانی که حوزه شغلی آنها در زمان گذشته با حال یکسان است. همانطور که مشخص است دو کارمند شماره ۱۷۶ و ۲۰۰ تغییر شغل نداشته اند.

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

	EMPLOYEE_ID	JOB_ID
1	176	SA_REP
2	200	AD_ASST

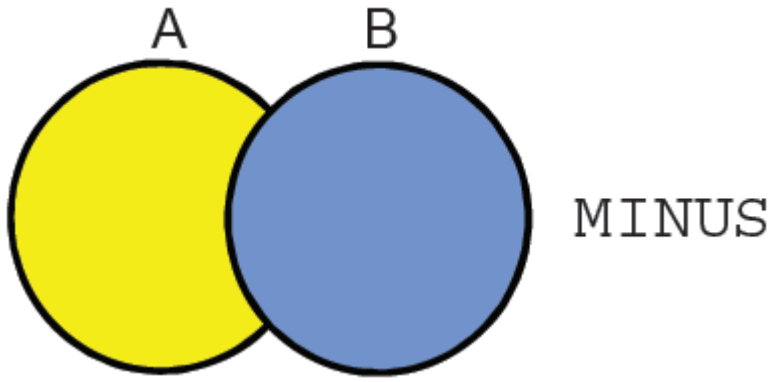
مثال: اگر در مثال قبل فقط اطلاعات کارمندانی را بخواهیم که تغییر شغل و تغییر دپارتمان نداشته اند از دستور زیر استفاده می کنیم.

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

	EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
1	176	SA_REP	80

بنابراین با اضافه کردن ستون DEPARTMENT_ID کارمند شماره ۲۰۰ در خروجی دستور قرار ندارد چون در زمان گذشته در دپارتمان دیگری فعالیت داشته است.

۸.۵ عملگر MINUS



این عملگر تمام سطرهایی که توسط دستور اول انتخاب می شوند ولی در دستور دوم قرار ندارند را برمی گرداند. در واقع عمل تفاضل بین دستور اول و دوم انجام می شود.

مثال: نمایش شماره پرسنلی کارمندانی که تغییر شغل نداشته اند.

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

	EMPLOYEE_ID
1	100
2	103
3	104
...	
13	202
14	205
15	206

۸.۶ یکسان کردن تعداد و نوع داده ستون ها

زمانی که از عملگرهای SET استفاده می شود ولی تعداد ستون های دستورات SELECT یکسان نیستند باید ستون معادل تعریف شود. همچنین اگر نوع داده ستون های متناظر، یکسان نبود باید از توابع تبدیل نوع داده استفاده شود.

مثال: سابقه اطلاعات شغلی کارمندان نمایش یابد. همانطور که مشخص است در هرکدام از دستوات SELECT یک ستون متناظر با نام دلخواه و نوع داده یکسان تعریف شده است.

```
SELECT location_id, department_name "Department",
   TO_CHAR(NULL) "Warehouse location"
FROM departments
UNION
SELECT location_id, TO_CHAR(NULL) "Department",
   state_province
FROM locations;
```

	LOCATION_ID	Department	Warehouse location
1	1400	IT	(null)
2	1400	(null)	Texas
3	1500	Shipping	(null)
4	1500	(null)	California
5	1700	Accounting	(null)
6	1700	Administration	(null)
7	1700	Contracting	(null)

. . .

مثال: نام کارمندان، حوزه فعالیت و درآمد آنها نمایش یابد. در این مثال در دستور دوم از مقدار LITERAL برابر با صفر استفاده شده است تا به عنوان یک ستون معادل برای ستون SALARY در دستور اول باشد. بنابراین مقدار ستون SALARY برای سطرهایی که از طریق دستور دوم انتخاب می شوند برابر با صفر خواهد بود(کادر قرمز رنگ).

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

	EMPLOYEE_ID	JOB_ID	SALARY
1	100	AD_PRES	24000
2	101	AC_ACCOUNT	0
3	101	AC_MGR	0
4	101	AD_VP	17000
5	102	AD_VP	17000

...

29	205	AC_MGR	12000
30	206	AC_ACCOUNT	8300

۸.۷ استفاده از عبارت **ORDER BY** برای عملگرهای **SET**

می توان یک مرتبه و در انتهای دستورات عملگر SET از عبارت ORDER BY استفاده نمود. عبارت ORDER BY بر اساس ستون های دستور اول عمل می کند و می تواند نام مستعار یا نام ستون های دستور اول را بپذیرد.

مثال: مرتب سازی نتایج خروجی بر اساس ستون شماره دو در دستور اول(ستون JOB_ID).

```
SELECT employee_id, job_id,salary
FROM   employees
UNION
SELECT employee_id, job_id,0
FROM   job_history
ORDER BY 2;
```

۹ دستورات DML و کنترل تراکنش ها

در این فصل انواع دستورات DML(Data Manipulation Language) توضیح داده می شوند. دستورهای INSERT، DELETE، UPDATE از نوع DML هستند که باعث می شوند اطلاعات جدول های دیتابیس تغییر یابند. همچنین در ادامه، روش کنترل تراکنش ها بر اساس مفهوم READ CONSISTENCY و عملیات COMMIT، ROLLBACK و SAVEPOINT توضیح داده می شوند.

زمانی که یک دستور از نوع DML اجرا می شود یکی از حالت های زیر رخ می دهد:

- اطلاعات جدید به یک جدول اضافه می شوند(توسط دستور INSERT).
- اطلاعات قبلی تغییر می کند(توسط دستور UPDATE).
- اطلاعات قبلی حذف می شوند(توسط دستور DELETE).

در دیتابیس اوراکل به مجموعه ای از دستورات DML که یک عمل خاصی را انجام می دهند یک **تراکنش(TRANSACTION)** می گویند.

۹.۱ دستور INSERT

سینتکس دستور INSERT را در شکل زیر می بینید. در این دستور کلمه TABLE نام یک جدول از دیتابیس می باشد و کلمات COLUMN نام ستون یا ستون هایی از آن جدول هستند که در لیست VALUES برای هر کدام یک مقدار خاص درنظر گرفته می شود تا در جدول مورد نظر درج شود.

```
INSERT INTO   table [(column [, column...])]
VALUES        (value [, value...]);
```

نکته: در دستور INSERT برای مقدارهای از نوع تاریخ یا کاراکتری از علامت '' استفاده می شود.

مثال: یک سطر جدید شامل اطلاعات دپارتمان شماره ۷۰ به جدول DEPARTMENTS اضافه کنید.

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

نکته: اگر سطری که توسط دستور INSERT به یک جدول اضافه می کنیم، شامل اطلاعات برای تمام ستون های آن جدول است (و ترتیب ستون های جدول رعایت شده باشد) دیگر نیاز نیست در آن دستور از نام ستون ها استفاده کنیم.

مثال: جدول DEPARTMENTS دارای چهار ستون است که در آنها توسط دستور زیر اطلاعات با نوع داده مناسب درج می شوند ولی نام ستون ها استفاده نشده است.

```
INSERT INTO departments
VALUES (70, 'Public Relations', 100, 1700);
1 rows inserted
```

نکته: اگر در دستور INSERT نام و مقدار یکی از ستون های جدول تعریف نشود، برای آن ستون مقدار NULL درج می شود. همچنین می توان در ستون های جدول به روش صریح و با استفاده از لیست VALUES یک مقدار NULL درج کرد.

مثال: جدول DEPARTMENTS چهار ستون دارد که توسط هر دو دستور در ستون سوم و چهارم مقدار NULL درج می شود.

```
INSERT INTO   departments (department_id,
                           department_name)
VALUES        (30, 'Purchasing');
1 rows inserted
```

```
INSERT INTO   departments
VALUES        (100, 'Finance', NULL, NULL);
1 rows inserted
```

نکته: می توان در دستور INSERT و در لیست VALUES از توابع مختلف استفاده نمود.

مثال: با استفاده از تابع SYSDATE تاریخ و زمان فعلی سیستم در ستون HIRE_DATE درج می شود.

```
INSERT INTO employees (employee_id,
                       first_name, last_name,
                       email, phone_number,
                       hire_date, job_id, salary,
                       commission_pct, manager_id,
                       department_id)
VALUES                (113,
                       'Louis', 'Popp',
                       'LPOPP', '515.124.4567',
                       SYSDATE, 'AC_ACCOUNT', 6900,
                       NULL, 205, 110);
1 rows inserted
```

مثال: اطلاعات کارمند جدید در جدول EMPLOYEES درج شود. از تابع تبدیل نوع داده TO_DATE استفاده می شود و تاریخ با فرمت مناسب ذخیره می شود.

```
INSERT INTO employees
VALUES      (114,
             'Den', 'Raphealy',
             'DRAPHEAL', '515.127.4561',
             TO_DATE('FEB 3, 2003', 'MON DD, YYYY'),
             'SA_REP', 11000, 0.2, 100, 60);
1 rows inserted
```

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID
1	114	Den	Raphealy	DRAPHEAL	515.127.4561	03-FEB-03	SA_REP	11000	0.2	100

۹.۲ کُپی اطلاعات با استفاده از دستور INSERT

اگر سه شرط زیر رعایت شوند می توان با استفاده از دستور INSERT اطلاعات یک جدول را در جدول دیگر کُپی کرد:

۱- در داخل دستور INSERT از یک SUBQUERY استفاده شود.

۲- نوع داده و تعداد ستون های دستور INSERT با دستور SUBQUERY برابر باشد.

۳- در دستور INSERT از لیست VALUES استفاده **نشود**.

مثال: تمام سطرهایی که توسط دستور SUBQUERY انتخاب می شوند در جدول SALES_REPS درج شود.

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
  SELECT employee_id, last_name, salary, commission_pct
  FROM   employees
  WHERE  job_id LIKE '%REP%';

5 rows inserted.
```

مثال: یک کُپی از جدول EMPLOYEES در جدول COPY_EMP ذخیره شود (در دیتابیس جدول COPY_EMP با ساختار یکسان با جدول EMPLOYEES وجود دارد).

```
INSERT INTO copy_emp
  SELECT *
  FROM   employees;
```

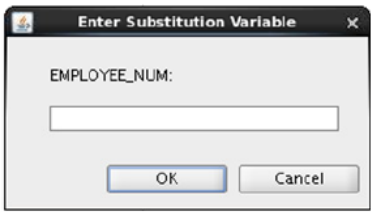
۹.۳ استفاده از متغیرهای تعویضی در دستور INSERT

در زمان درج اطلاعات توسط دستور INSERT می توان از متغیرهای تعویضی استفاده نمود. ابتدا متغیرهای تعویضی را با ذکر چند مثال توضیح می دهیم.

متغیرهای تعویضی در قسمت های مختلف یک دستور SQL از جمله نام جداول، ستون ها و یا عبارات ORDER BY و شروط WHERE استفاده می شوند. با استفاده از این متغیرها می توانیم مقادیر یا نام ها را ار کاربر بپرسیم و یا در بیرون از دستور تعریف کنیم. تا زمانی که این متغیرها تعیین نشود دستور اجرا نمی شود.. برای تعریف متغیر تعویضی باید از علامت & استفاده کنیم.

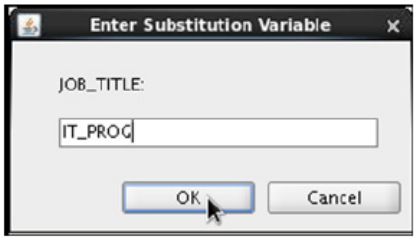
مثال: در این مثال ابتدا مقدار متغیر EMPLOYEE_NUM از کاربر پرسیده می شود و سپس بر اساس آن مقدار دستور SELECT اجرا می شود.

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



نکته: برای مقدارهایی که کاراکتر یا تاریخ هستند در دستور از علامت ‘ ’ استفاده می شود.

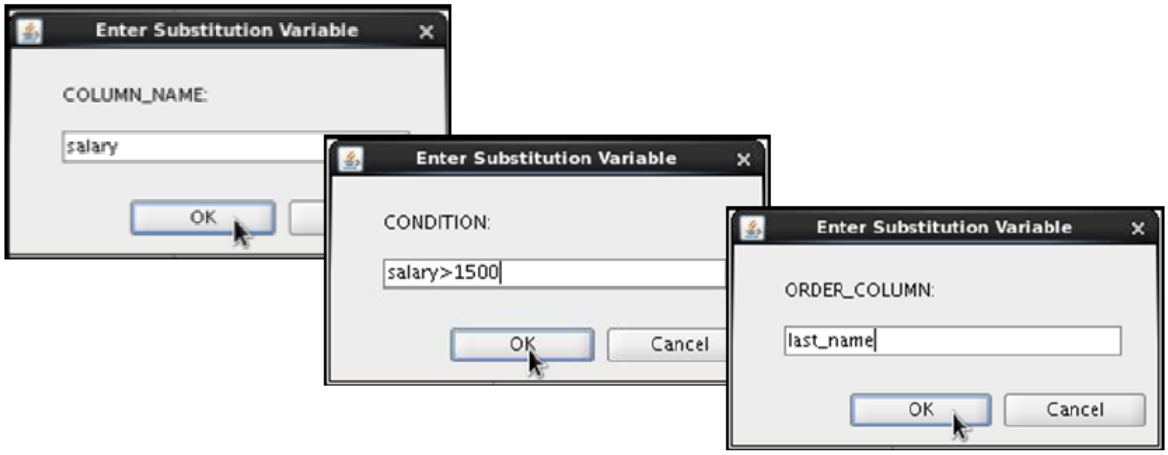
```
SELECT last_name, department_id, salary*12
FROM   employees
WHERE  job_id = '&job_title' ;
```



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

مثال: استفاده از چندین متغیر در یک دستور SELECT:

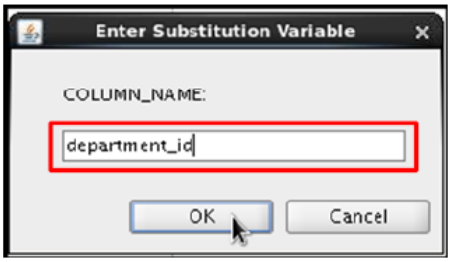
```
SELECT employee_id, last_name, job_id,&column_name
FROM   employees
WHERE  &condition
ORDER BY &order_column ;
```



نکته: اگر نیاز باشد فقط یکبار مقدار یک متغیر از کاربر پرسیده شود و در جاهای دیگر که آن متغیر قرار دارد ار همان مقدار استفاده شود باید در زمان نوشتن متغیر برای مرتبه اول از علامت && استفاده گردد.

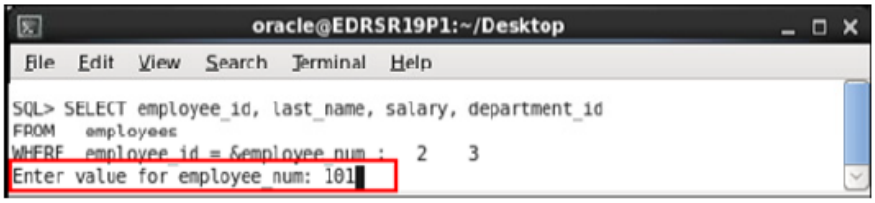
مثال: مقدار متغیر COLUMN_NAME را یک مرتبه از کاربر بپرسید و دستور زیر را اجرا کنید.

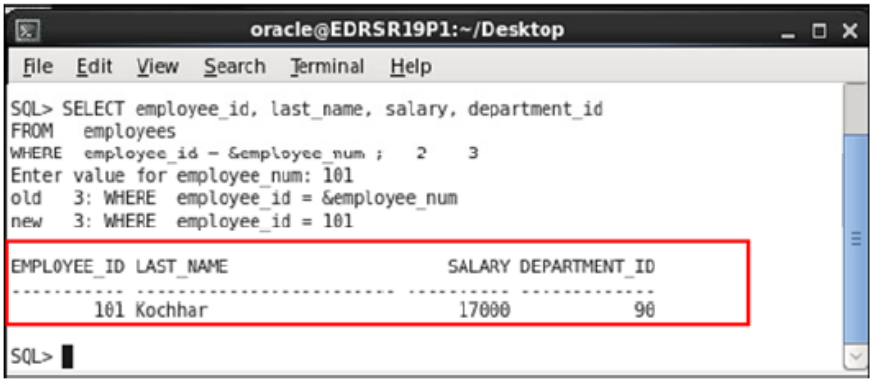
```
SELECT   employee_id, last_name, job_id, &&column_name
FROM     employees
ORDER BY &column_name ;
```



	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	200	Whalen	AD_ASST	10
2	201	Hartstein	MK_MAN	20
3	202	Fay	MK_REP	20

نکته: در محیط SQL*PLUS نیز می توان از متغیرهای تعویضی استفاده نمود. مثال:





نکته: می توان بجای اینکه مقدار متغیر را از کاربر پرسید، از دستور DEFINE قبل از دستور SELECT استفاده نمود.

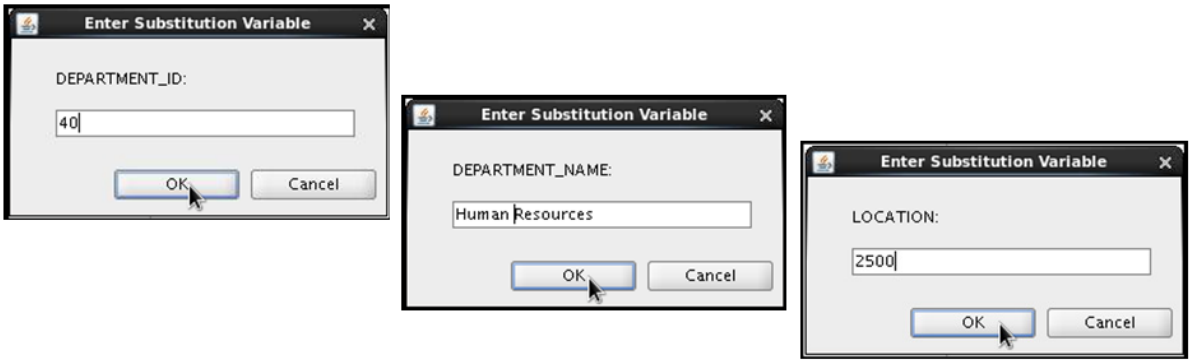
مثال: مقدار متغیر EMPLOYEE_NUM را برابر با ۲۰۰ تنظیم کنید و دستور زیر را اجرا کنید.

```
DEFINE employee_num = 200

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

UNDEFINE employee_num
```

	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	200	Whalen	4400	10

مثال: اطلاعات مورد نیاز دپارتمان جدید از کاربر دریافت شود و در جدول DEPARTMENT درج گردد.

```
INSERT INTO departments
            (department_id, department_name, location_id)
VALUES      (&department_id, '&department_name',&location);
```



۹.۴ دستور UPDATE

یکی دیگر از دستورات DML دستور UPDATE است که از آن برای تغییر مقدار اطلاعات جداول استفاده می شود. در شکل زیر سینتکس مربوط به این دستور را می بینید.

```
UPDATE      table
SET         column = value [, column = value, ...]
[WHERE      condition];
```

نکته: اگر در این دستور از عبارت WHERE استفاده شود عملیات UPDATE فقط برای یک یا چند سطر خاص انجام می شود و در غیر این صورت تمام سطرهای جدول تغییر می کنند.

مثال: مقدار ستون DEPARTMENT_ID برای سطرهایی که مقدار EMPLOYEE_ID آنها برابر با ۱۱۳ است به مقدار ۵۰ تغییر یابد.

```
UPDATE employees
SET    department_id = 50
WHERE  employee_id = 113;
1 rows updated
```

مثال: مقدار ستون DEPARTMENT_ID در تمام سطرها برابر با ۱۱۰ شود.

```
UPDATE   copy_emp
SET      department_id = 110;
```

```
22 rows updated
```

مثال: مقدار حق کمیسیون کارمند شماره ۱۱۴ برابر با مقدار NULL شود.

```
UPDATE employees
SET job_id = 'IT_PROG', commission_pct = NULL
WHERE employee_id = 114;
```

نکته: در دستور UPDATE می توان از دستور SUBQUERY استفاده نمود.

مثال: شغل و حقوق کارمند شماره ۱۰۳ برابر با شغل و حقوق کارمند شماره ۲۰۵ شود.

```
UPDATE   employees
SET      (job_id,salary)   = (SELECT  job_id,salary
                              FROM    employees
                              WHERE   employee_id = 205)
         WHERE    employee_id    =  103;
```

```
1 rows updated
```

مثال: شماره دپارتمان تمام کارمندانی که شغل آنها با شغل کارمند شماره ۲۰۰ برابر است به شماره دپارتمان کارمند شماره ۱۰۰ تغییر یابد.

```
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 rows updated
```

۹.۵ دستور DELETE

دستور DELETE یک دستور از نوع DML است که یک یا چند سطر از جدول را حذف می کند. سینتکس این دستور را در شکل زیر می بینید.

```
DELETE [FROM]   table
[WHERE          condition];
```

در دستور DELETE استفاده از کلمه FROM اختیاری است. همچنین اگر از عبارت WHERE استفاده شود فقط سطرهایی از جدول که مطابق با شرط WHERE هستند حذف می شوند و در غیر این صورت تمام سطرهای جدول حذف خواهند شد.

مثال: سطرهایی از جدول که در آن نام دپارتمان برابر با Finance است را حذف کنید.

```
DELETE FROM departments
WHERE  department_name = 'Finance';
1 rows deleted
```

مثال: تمام سطرهای جدول COPY_EMP را حذف کنید.

```
DELETE FROM  copy_emp;
22 rows deleted
```

نکته: در دستور DELETE می توان از SUBQUERY استفاده نمود.

مثال: سطرهایی از جدول EMPLOYEES که نام دپارتمان آنها شبیه به Public می باشد را حذف کنید.

```
DELETE FROM employees
WHERE  department_id IN
                (SELECT department_id
                 FROM   departments
                 WHERE  department_name
                        LIKE '%Public%');
1 rows deleted
```

نکته: دستور TRUNCATE یک دستور از نوع DDL(DATA DEFINITION LANGUAGE) می باشد که شبیه به دستور DELETE عمل می کند. این دستور تمام سطرهای یک جدول را پاک می کند و فقط ساختار جدول را باقی می گذارد. در شکل زیر سینتکس این دستور را می بینید.

```
TRUNCATE TABLE table_name;
```

مثال: تمام اطلاعات جدول COPY_EMP حذف شود.

```
TRUNCATE TABLE copy_emp;
```

۹.۶ کنترل تراکنش های دیتابیس

هر ارتباطی که یک کاربر با دیتابیس برقرار می کند SESSION نامیده می شود. در دیتابیس اوراکل هر تراکنش در یک SESSION خاص و به وسیله مجموعه ای از دستورات از نوع DML یا DDL رخ می دهد. اگر در یک تراکنش از دستورات DML مانند INSERT، DELETE، UPDATE استفاده شود باید جهت دائمی کردن تغییراتی که این دستورات ایجاد کرده اند از دستور **COMMIT** استفاده شود. زمانی که از دستور COMMIT استفاده می شود این تغییرات از حافظه به دیسک منتقل و در دیتابیس ذخیره می شوند و SESSION های دیگر این تغییرات را خواهند دید. همچنین در صورتی که آن SESSION خاتمه یابد تمام آن تغییرات لغو می شود و اطلاعات به مقدارهای اولیه باز می گردند. البته اگر در یک SESSION بعد از دستورات DML یک دستور از نوع DDL استفاده شود دیتابیس اوراکل به صورت اتوماتیک یک عمل COMMIT انجام می دهد و دیگر نیازی به اجرای دستور COMMIT نیست.

مثال: یک سطر از جدول EMPLOYEES حذف شود و یک سطر در جدول DEPARTMENTS درج گردد. سپس این تغییرات به صورت دائمی در دیتابیس ذخیره شوند

- Make the changes:

```
DELETE FROM EMPLOYEES
WHERE employee_id=113;
1 rows deleted
INSERT INTO departments
VALUES (290, 'Corporate Tax', NULL, 1700);
1 rows inserted
```

- Commit the changes:

```
COMMIT;
commited.
```

اگر بعد از دستورات DML بجای دستور COMMIT از دستور **ROLLBACK** استفاده شود تمام تغییراتی که توسط آن تراکنش در حافظه انجام گرفته است لغو می شود و اطلاعات به مقدارهای اولیه باز می گردند. همچنین اگر قبل از COMMIT به صورت ناخواسته ارتباط با دیتابیس قطع شود به صورت اتوماتیک یک عمل ROLLBACK انجام می شود.

مثال: به اشتباه تمام سطرهای جدول COPY_EMP حذف می شوند ولی در ادامه با دستور ROLLBACK تمام سطرها برگردانده می شوند.

```
DELETE FROM copy_emp;
ROLLBACK ;
```

نکته: اگر قبل از عمل COMMIT دستورات DML، داده های یک جدول تغییر یابند این تغییرات موقتی در همان SESSION با دستور SELECT نمایش می یابند.

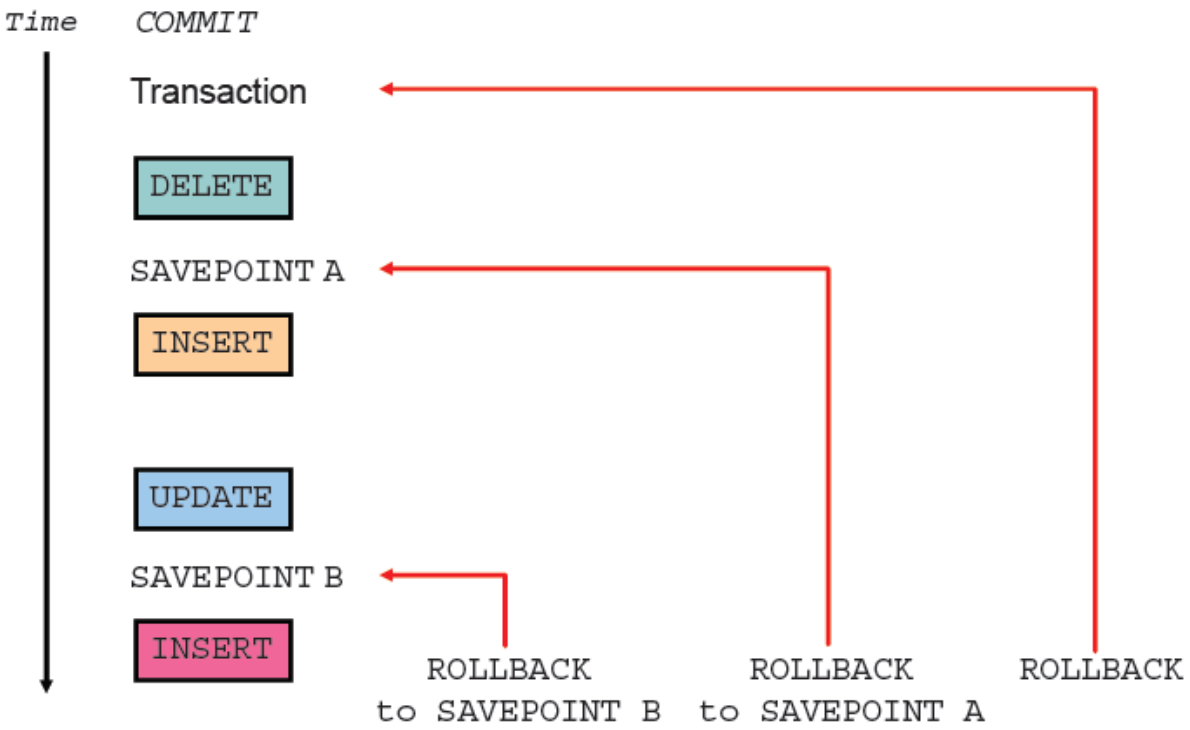
مثال: به اشتباه تمام سطرهای جدول TEST حذف می شوند ولی دستور ROLLBACK تغییرات انجام شده را لغو می کند. همچنین در ادامه یک سطر با ID برابر با ۱۰۰ حذف می شود و عمل COMMIT انجام می شود تا این تغییرات در دیتابیس ذخیره شوند. توجه شود که قبل از عمل COMMIT و در همان SESSION دستور SELECT تغییر انجام شده را نمایش می دهد.

```
DELETE FROM test;
4 rows deleted.

ROLLBACK;
Rollback complete.

DELETE FROM test WHERE  id = 100;
1 row deleted.

SELECT * FROM   test WHERE  id = 100;
No rows selected.

COMMIT;
Commit complete.
```

نکته: زمانی که در یک SESSION توسط دستورات DML سطرهایی از یک جدول تغییر می کنند این سطرها LOCK می شوند بنابراین SESSIONهای دیگر نمی توانند همزمان آنها را تغییر دهند تا زمانی که یک عمل COMMIT یا ROLLBACK در آن SESSION رخ دهد.

نکته: وقتی یک تراکنش در حال اجرا می باشد اگر بعد از یک دستور DML از دستور SAVEPOINT NAME1 استفاده شود و سپس مراحل بعدی تراکنش انجام شود می توان با دستور ROLLBACK TO SAVEPOINT NAME1 به زمان اجرای آن SAVEPOINT بازگشت. البته به شرطی که عمل COMMIT انجام نشده باشد زیرا بعد از هر COMMIT تمامی SAVEPOINT های قبلی حذف می شوند.

مثال: در تراکنش زیر یک SAVEPOINT به نام A بعد از دستور DELETE و یکی دیگر به نام B بعد از دستورات UPDATE و INSERT انجام می شود. بنابراین می توان به هرکدام از آنها ROLLBACK کرد.



۹.۷ اصل READ CONSISTENCY

عملیاتی که توسط کاربران یک دیتابیس انجام می شود یا به صورت READ و با دستور SELECT است یا به صورت WRITE و با دستورات INSERT، UPDATE، DELETE. در تمام این عملیات باید مکانیزمی باشد که هر کاربر مستقل از دیگران به داده های قبل از آخرین COMMIT در دیتابیس دسترسی داشته باشد

دیتابیس اوراکل بر اساس اصل **READ CONSISTENCY** تضمین می کند یک کابر در هر SESSION آخرین اطلاعات COMMIT شده را می بیند زیرا تا زمانی که COMMIT انجام نشود SESSION های دیگر متوجه تغییرات نخواهند شد. بنابراین اگر دو SESSION به منظور نوشتن اطلاعات در سطرهای یکسان اقدام کنند باید منتظر هم بمانند ولی در حالت های دیگر هر SESSION به اطلاعات مورد نیاز خود دسترسی دارد.

نکته: یک کاربر می تواند SESSIONهای مختلف داشته باشد و در سطح SESSION مفهوم READ CONSISTENCY پابرجاست.

نکته: زمانی که در یک SESSION با دستورات DML سطرهای یک جدول تغییر می کند قبل از عمل COMMIT اطلاعات قبلی در بخشی از دیسک به نام UNDO SEGMENT قرار دارد و SESSION های دیگر از طریق این بخش به اطلاعات قبلی دسترسی دارند تا زمانی که عمل COMMIT انجام شود.

۹.۸ عبارت FOR UPDATE در دستور SELECT

اگر یک کاربر به نام ALI، در یک دستور SELECT از عبارت FOR UPDATE استفاده کند سطرهایی که توسط دستور SELECT انتخاب شده اند LOCK می شوند و SESSIONهای دیگر نمی توانند آن سطرها را تغییر دهند تا زمانی که یک عمل COMMIT یا ROLLBACK توسط کاربر ALI اجرا شود. البته اگر آن سطرها از قبل توسط SESSIONهای دیگر LOCK شده باشند دیتابیس منتظر UNLOCK شدن آنها می ماند و سپس دستور SELECT FOR UPDATE را برای کاربر ALI اجرا می کند.

نکته: می توان در دستور SELECT FOR UPDATE از عبارت NOWAIT استفاده کرد تا اگر سطرهایی که می خواهیم LOCK کنیم از قبل LOCK بودند دیتابیس کنترل عملیات را به SESSION برگرداند و در پشت صحنه منتظر UNLOCK شدن آن سطرها بماند.

مثال: سطرهایی از جدول EMPLOYEES که JOB_ID برابر با SA_REP دارند را LOCK کنید.

```
SELECT employee_id, salary, commission_pct, job_id
FROM employees
WHERE job_id = 'SA_REP'
FOR UPDATE
ORDER BY employee_id;
```

مثال: برخی از سطرهای دو جدول EMPLOYEES و DEPARTMENTS را LOCK کنید.

```
SELECT e.employee_id, e.salary, e.commission_pct
FROM employees e JOIN departments d
USING (department_id)
WHERE job_id = 'ST_CLERK'
AND location_id = 1500
FOR UPDATE
ORDER BY e.employee_id;
```

نکته: اگر از عبارت FOR UPDATE **OF** COLUMN_NAME استفاده شود فقط سطرهایی که شامل ستون COLUMN_NAME هستند LOCK می شوند.

مثال: فقط سطرهایی از جدول EMPLOYEES که مطابق با شرط های دستور هستند LOCK شوند (ستون salary در جدول EMPLOYEES می باشد).

```
SELECT e.employee_id, e.salary, e.commission_pct
FROM employees e JOIN departments d
USING (department_id)
WHERE job_id = 'ST_CLERK' AND location_id = 1500
FOR UPDATE OF e.salary
ORDER BY e.employee_id;
```

نکته: اگر از عبارت FOR UPDATE **WAIT X** استفاده شود دیتابیس X ثانیه صبر می کند تا سطرهای مورد نظر UNLOCK شوند و سپس کنترل را به SESSION بازمی گرداند

مثال: ابتدا ۵ ثانیه منتظر UNLOCK شدن سطرها بمانید.

```
SELECT employee_id, salary, commission_pct, job_id
FROM employees
WHERE job_id = 'SA_REP'
FOR UPDATE WAIT 5
ORDER BY employee_id;
```

۱۰ دستورات DDL و CONSTRAINT ها

در این فصل دستورات از نوع DDL(DATA DEFINITION LANGUAGE) معرفی می شوند. مفهوم OBJECT و برخی از نوع داده هایی که در دیتابیس اوراکل استفاده می شوند را توضیح می دهیم. همچنین انواع CONSTRAINTها توضیح داده می شوند و خطاهایی که با رعایت نکردن آنها رخ می دهند را معرفی می کنیم.

۱۰.۱ انواع OBJECT ها در دیتابیس اوراکل

یک دیتابیس مجموعه ای از OBJECT های متفاوت است که متعلق به کاربران دیتابیس هستند. در جدول زیر بعضی از انواع OBJECT های دیتابیس اوراکل و توضیحات آنها را می بینید.

نام object	توضیحات
Table	عنصر اولیه ذخیره سازی می باشد و داده ها را در سطرها(رکوردها)ذخیره می کند
View	مجموعه ای از داده های یک یا چند جدول را به صورت LOGICAL نمایش می دهد
Sequence	مقدارهای عددی ایجاد می کند
Index	سرعت اجرای برخی دستورات را افزایش می دهد
Synonym	یک نام جایگزین برای object ها ایجاد می کند

نکته: در دیتابیس اوراکل OBJECT ها به دو دسته تقسیم می شوند:

۱- SCHEMA OBJECTS که تعدادی از آنها را در جدول بالا می بینید.

۲- NON-SCHEMA OBJECTS مانند دایرکتوری، TABLESPACE و ROLE.

۱۰.۲ نام گذاری OBJECT ها

باید در نام گذاری OBJECT های دیتابیس اوراکل قوانین زیر رعایت شوند:

۱- نام OBJECT باید با حروف انگلیسی آغاز شود.

۲- یک نام می تواند بین ۱ تا ۳۰ کارکتر باشد.

۳- در نام گذاری فقط می توان از کارکترهای A تا Z، a تا z، 0 تا 9، _ ، $ و # استفاده نمود.

۴- نام یک OBJECT باید با نام OBJECT های دیگر که مالک آنها یکسان است متفاوت باشد.

۵- از نام های رزرو شده توسط دیتابیس اوراکل نمی شود استفاده کرد(مثلا TABLE).

نکته: نام یک OBJECT به بزرگی یا کوچکی حروف حساس نیست. البته زمانی که در برخی دستورات به نام OBJECT ها اشاره می شود(از علامت " " استفاده می شود) باید بزرگی و یا کوچکی حروف رعایت شود.

۱۰.۳ نوع داده ها در دیتابیس اوراکل

برای هر ستون از یک جدول باید یک نوع داده مشخص کنیم. در دیتابیس اوراکل نوع داده ها به دو حالت Fixed-Length یا Variable-Length هستند. اگر یک ستون با نوع داده های Variable-Length تعریف شود هر سطر از آن ستون می تواند دارای طول متفاوت باشد. یعنی فضایی که هر سطر از دیتابیس اشغال می کند برابر با اندازه مقدار آن سطر می باشد. ولی در نوع داده های Fixed-Length طول تمام سطرها یکسان است. در ادامه برخی از نوع داده های پرکاربرد در دیتابیس اوراکل توضیح داده می شوند.

- **VARCHAR2(size)**: این نوع داده Variable-Length است و اگر در دیتابیس پارامتر MAX_SQL_STRING برابر با EXTENDED باشد طول پیش فرض آن می تواند بین ۱ تا ۳۲۷۶۷ بایت تعریف گردد(توسط size). اگر پارامتر MAX_SQL_STRING برابر با LEGACY باشد طول پیش فرض می تواند بین ۱ تا ۴۰۰۰ بایت تعریف شود.
- **CHAR[(size)]**: این نوع داده Fixed-Length است و به صورت اختیاری می توان طول آن را تعریف کرد. اگر طول آن تعریف نشود طول هر سطر ۱ بایت خواهد بود. ولی size را می توان ۱ تا ۲۰۰۰ بایت تعریف نمود.
- **NUMBER(p,s)**: این نوع داده Variable-Length است.در آن p تعداد اعداد دسیمال است و این تعداد می تواند بین ۱ تا ۳۸ تعریف شود. همچنین s تعداد اعداد اعشاری است که می تواند بین ۸۴- تا ۱۲۷ تعریف شود.
- **DATE**: نوع داده DATE به منظور ذخیره سازی زمان و تاریخ تعریف می شود.
- **LONG**: این نوع داده Variable-Length است که طول آن در هر سطر می تواند تا حداکثر ۲ گیگابایت باشد. معمولا از نوع داده LONG استفاده نمی شود و بجای آن از CLOB استفاده می گردد.

- **CLOB**: این نوع داده یک CHARACTER LARGE OBJECT است که Variable-Length می باشد و می تواند کاراکترهای از نوع Single-Byte یا Multiple-Byte ذخیره کند. حداکثر طول هر سطر می تواند DB_BLOCK_SIZE * (4gigabyte -1) باشد. از این نوع داده معمولا برای ذخیره صدا، تصویر یا ویدئو استفاده می شود.
- **NCLOB**: همان CLOB است ولی اگر کاراکترست دیتابیس از نوع Multibye نباشد نیز کاراکترهایی که با فرمت Multibyte هستند را نمایش می دهد (مثلا زبان چینی).
- **RAW(size)**: این نوع داده Variable-Length است و برای ذخیره ی داده های از نوع باینری یا رشته ای از کاراکترها استفاده می شود. اگر پارامتر دیتابیس MAX_SQL_STRING برابر با EXTENDED باشد طول پیش فرض آن می تواند بین ۱ تا ۳۲۷۶۷ بایت تعریف گردد(توسط size). اگر پارامتر MAX_SQL_STRING برابر با LEGACY باشد طول پیش فرض می تواند بین ۱ تا ۴۰۰۰ بایت تعریف شود. از RAW برای ذخیره فایل های خواندنی و گرافیکی استفاده می شود.
- **LONG RAW**: همانند RAW است ولی طول آن در هر سطر می تواند تا ۲ گیگابایت باشد.
- BLOB: این BINARY LARGE OBJECT همانند CLOB است ولی ذخیره اطلاعات به صورت باینری می باشد.
- **BFILE**: یک LARGE OBJECT است که برای ذخیره اطلاعات باینری در یک فایل خارج از دیتابیس استفاده می شود و طول آن ۴ گیگابایت می باشد.
- **ROWID**: در این نوع داده یک رشته بر مبنای ۶۴ ذخیره می شود که آدرس هر سطر از یک جدول را مشخص می کند.

نکته: به دلیل محدودیت های زیر بهتر است از نوع داده CLOB بجای نوع داده LONG استفاده نمود.

۱- اگر با استفاده از روش SUBQUERY یک جدول را کُپی کنیم ستون هایی که نوع داده آنها LONG هستند کُپی نمی شوند.

۲- یک ستون از نوع داده LONG را **نمی توان** در عبارت های GROUP BY و ORDER BY استفاده نمود.

۳- در هر جدول فقط می توانیم یک ستون با نوع داده LONG تعریف کنیم.

۴- اگر یک ستون از نوع LONG باشد هیچ CONSTRAINT برای آن ستون نمی توان تعریف کرد.

۱۰.۴ نوع داده برای زمان و تاریخ

برای مقدارهای تاریخ و زمان علاوه بر نوع داده DATE می توان از نوع داده های جدول زیر استفاده نمود.

نوع داده	توضیحات
TIMESTAMP	سال، ماه، روز، ساعت ودقیقه به همراه ثانیه ذخیره می شود
INTERVAL YEAR TO MONTH	تاریخ را به عنوان یک بازه زمانی سال و ماه ذخیره می کند
INTERVAL DAY TO SECOND	بازه زمانی به اندازه روز، ساعت، دقیقه و ثانیه ذخیره می کند

نکته: کاربرد اصلی نوع داده های INTERVAL زمانی است که اختلاف دو تاریخ را می خواهیم محاسبه کنیم.

مثال: یک جدول اطلاعات افراد بسازید و تجربه کاری کارمند Camila را ۱۰ سال و ۲ماه ذخیره کنید.

```
CREATE TABLE candidates (
   candidate_id NUMBER,
   first_name VARCHAR2(50) NOT NULL,
   last_name VARCHAR2(50) NOT NULL,
   job_title VARCHAR2(255) NOT NULL,
   year_of_experience INTERVAL YEAR TO MONTH,
   PRIMARY KEY (candidate_id)
);

INSERT INTO candidates (
   first_name,
   last_name,
   job_title,
   year_of_experience
   )
VALUES (
   'Camila',
   'Kramer',
   'SCM Manager',
   INTERVAL '10-2' YEAR TO MONTH
   );
```

نکته: می توان مقدار LITERAL برای نوع داده های INTERVAL تعریف نمود.

مثال: یک بازه زمانی ۱۰ سال و ۰ ماه تعریف کنید تا در یک دستور خاص استفاده شود.

```
INTERVAL '15' YEAR
```

مثال: یک بازه زمانی ۸ ساعته تعریف کنید تا در یک دستور خاص استفاده شود.

```
INTERVAL '8' HOUR
```

۱۰.۵ تعریف مقدار پیش فرض برای یک نوع داده

اگر در زمان تعریف نوع داده برای یک ستون از کلمه DEFAULT استفاده شود می توان نوع داده پیش فرض آن را تعیین کرد. این نوع داده پیش فرض می تواند یک LITERAL، عبارت یا تابع SQL باشد. بنابراین زمانی که می خواهیم یک سطر را که برای یک ستون خاص هیچ مقداری ندارد به جدول اضافه کنیم اگر برای نوع داده آن ستون خاص از عبارت DEFAULT استفاده شده باشد در آن سطر مقدار DEFAULT درج خواهد شد.

مثال:اگر در دستور INSERT مقداری برای ستون HIRE_DATE درج نشد مقدار SYSDATE(تاریخ و زمان فعلی سیستم) در آن سطر درج شود.

```
CREATE TABLE hire_dates
        (id             NUMBER(8),
        hire date DATE DEFAULT SYSDATE);
table HIRE_DATES created.
```

```
INSERT INTO hire_dates(id) values(35);
```

نکته: بنابراین در جدول بالا فقط زمانی که مقدار NULL در دستور آورده شود در ستون HIRE_DATE یک مقدار NULL درج می شود.

```
INSERT INTO hire_dates values(45, NULL);
```

۱۰.۶ دستور CREATE TABLE

اگر مجوز لازم را داشته باشیم می توانیم با دستور CREATE TABLE یک جدول بسازیم. این دستور از نوع DDL است بنابراین بعد از اجرای آن یک عمل COMMIT به صورت اتوماتیک انجام می شود. سینتکس این دستور را در شکل زیر می بینید. در این سینتکس [schema.] نام کاربری است که مالک جدول است و زمانی که با آن کاربر متصل هستیم استفاده از آن اختیاری است. همچنین نام ستون ها و نوع داده آنها در داخل پرانتز تعریف می شوند.

```
CREATE TABLE [schema.]table
        (column datatype [DEFAULT expr][, ...]);
```

نکته: بعد از ساختن یک جدول با دستور CREATE TABLE اطلاعات این جدول در دیتادیکشنری دیتابیس اوراکل ثبت می شود و می توان با دستور زیر این اطلاعات را نمایش داد.

```
select table_name from user_tables;
```

مثال: برای کاربر NADER یک جدول به اسم EMPLOYEE2 با شش ستون بسازید.

```
1. CREATE TABLE NADER.Employee2
2. (
3. EmployeeID int,
4. FirstName varchar(255),
5. LastName varchar(255),
6. Email varchar(255),
7. AddressLine varchar(255),
8. City varchar(255)
9. );
```

مثال: یک جدول به نام DEPT بسازید و نوع داده ستون CREATE_DATE به صورت DEFAULT برابر با تاریخ سیستم شود. سپس با استفاده از دستور DESC(در SQL*PLUS) مشخصات جدول را نمایش دهید.

- Create the table:

```
CREATE TABLE dept
        (deptno      NUMBER(2),
         dname       VARCHAR2(14),
         loc         VARCHAR2(13),
         create_date DATE DEFAULT SYSDATE);
table DEPT created.
```

- Confirm table creation:

```
DESCRIBE dept
```

```
DESCRIBE dept
Name        Null Type
----------- ---- ------------
DEPTNO           NUMBER(2)
DNAME            VARCHAR2(14)
LOC              VARCHAR2(13)
CREATE_DATE      DATE
```

۱۰.۷ انواع CONSTRAINT ها

زمانی که برای مجموعه ای از داده ها یک CONSRAINT تعریف می شود باید قوانین خاصی در زمان بروزرسانی، درج یا حذف آن داده ها رعایت شوند. در واقع CONSTRAINT ها یکسری محدودیت و شرایط خاص تعیین می کنند که در دستورات DML و DDL در نظر گرفته می شوند. می توان یک CONSTRAINT تعریف کرد که اگر اطلاعات یک جدول به یک جدول اصلی وابسته بود از حذف شدن جدول اصلی جلوگیری کند. بکارگیری CONSTRAINT در جدول ها سبب یکپارچگی در سطح دیتابیس می شود. انواع CONSTRAINT ها و توضیحات آنها را در جدول زیر می بینید.

Constraint	توضیحات
NOT NULL	باعث می شود یک ستون از جدول نتواند مقدار NULL ذخیره کند
UNIQUE	باعث می شود یک ستون یا ترکیبی از ستون های یک جدول مقدار غیرتکراری در تمام سطرهای جدول داشته باشند
PRIMARY KEY	هر سطر از جدول را متفاوت از سطرهای دیگر می کند
FOREIGN KEY	سبب می شود مقداری که در ستون یک جدول درج می شود با مقدار یک ستون در جدول دیگر برابر باشد
CHECK	یک شرط تعریف می کند که باید برقرار باشد

۱۰.۸ نکاتی در مورد CONSTRAINT ها

- اگر برای یک CONSTRAINT هیچ نامی انتخاب نشود سرور اوراکل از قالب SYS_Cn برای نام گذاری آن CONSTRAINT استفاده می کند و یک نام اتوماتیک ایجاد می کند.
- می توان در زمان ایجاد یک جدول با دستور CREATE TABLE همزمان CONSTRAINT های مورد نیاز راتعریف نمود. همچنین می توانیم بعد از ساخته شدن یک جدول، CONSTRAINT های آن را تعریف کنیم.
- بعد از تعریف CONSTRAINT اطلاعات مربوط به آنها در دیتادیکشنری دیتابیس ذخیره می شوند.
- CONSTRAINT یک OBJECT از دیتابیس اوراکل است بنابراین در تعیین نام آنها باید قوانین نام گذاری OBJECT ها را رعایت کنیم.

۱۰.۹ CONSTRAINT در سطح ستون و در سطح جدول

می توان CONSTRAINT ها را در دوسطح مختلف ستون و یا جدول تعریف کرد که از نظر عملکرد هیچ تفاوتی با هم ندارد. ولی اگر در سطح جدول تعریف می شوند باید ستون یا ستون های مورد نظر برای آن CONSTRAINT تعیین شوند.

سینتکس مربوط به تعریف CONSTRAINT در سطح ستون را در شکل زیر می بینید.

```
column [CONSTRAINT constraint_name] constraint_type,
```

سینتکس مربوط به تعریف CONSTRAINT در سطح جدول را در شکل زیر می بینید. ابتدا تمام ستون ها تعریف می شوند سپس CONSTRAINTها تعریف می شوند.

```
column,...
  [CONSTRAINT constraint_name] constraint_type
  (column, ...),
```

مثال: یک CONSTRAINT به نام emp_emp_id_pk در سطح ستون تعریف کنید.

```
CREATE TABLE employees(
  employee_id  NUMBER(6)
    CONSTRAINT emp_emp_id_pk PRIMARY KEY,
  first_name   VARCHAR2(20),
  ...);
```

مثال: CONSTRAINT مثال قبلی را در سطح جدول تعریف کنید.

```
CREATE TABLE employees(
  employee_id  NUMBER(6),
  first_name   VARCHAR2(20),
  ...
  job_id       VARCHAR2(10) NOT NULL,
  CONSTRAINT emp_emp_id_pk
    PRIMARY KEY (EMPLOYEE_ID));
```

نکته: اگر یک CONSTRAINT به صورت ترکیبی باشد و برای بیشتر از یک ستون تعریف شود فقط می توان آن را در سطح جدول تعریف نمود.

نکته: CONSTRAINT های از نوع NOT NULL را فقط در سطح ستون می توان تعریف کرد.

۱۰.۱۰ NOT NULL COSTRAINT

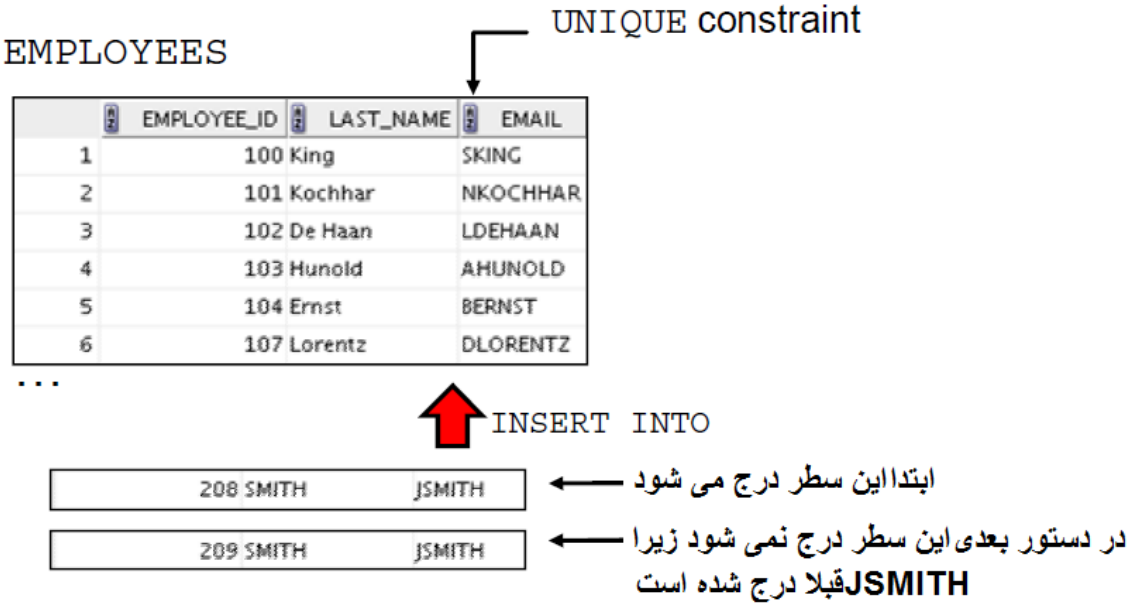
زمانی که برای یک ستون از جدول خاص یک CONSTRAINT از نوع NOT NULL تعریف می شود، اگر توسط دستورات DML مقدار NULL برای آن ستون در نظر گرفته شود خطای اوراکل دریافت می کنیم. بنابراین هیچ مقدار NULL در آن ستون قرار نخواهد گرفت.

مثال: در جدول زیر ستون های EMPLOYEE_ID، LAST_NAME، SALARY و HIRE_DATE از نوع NOT NULL تعریف شده اند بنابراین هیچ مقدار NULL در سطرهای جدول برای این ستون ها وجود ندارد. از طرفی ستون های دیگر مانند COMISSION_PCT بدون CONSTRAINT از نوع NOT NULL هستند بنابراین مقدار NULL ذخیره می کنند.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	COMMISSION_PCT	DEPARTMENT_ID	EMAIL	PHONE_NUMBER	HIRE_DATE
100	Steven	King	24000	(null)	90	SKING	515.123.4567	17-JUN-87
101	Neena	Kochhar	17000	(null)	90	NKOCHHAR	515.123.4568	21-SEP-89
102	Lex	De Haan	17000	(null)	90	LDEHAAN	515.123.4569	13-JAN-93
103	Alexander	Hunold	9000	(null)	60	AHUNOLD	590.423.4567	03-JAN-90
104	Bruce	Ernst	6000	(null)	60	BERNST	590.423.4568	21-MAY-91
107	Diana	Lorentz	4200	(null)	60	DLORENTZ	590.423.5567	07-FEB-99
124	Kevin	Mourgos	5800	(null)	50	KMOURGOS	650.123.5234	16-NOV-99
141	Trenna	Rajs	3500	(null)	50	TRAJS	650.121.8009	17-OCT-95
142	Curtis	Davies	3100	(null)	50	CDAVIES	650.121.2994	29-JAN-97
143	Randall	Matos	2600	(null)	50	RMATOS	650.121.2874	15-MAR-98
144	Peter	Vargas	2500	(null)	50	PVARGAS	650.121.2004	09-JUL-98
149	Eleni	Zlotkey	10500	0.2	00	EZLOTKEY	011.44.1344.429010	29-JAN-00
174	Ellen	Abel	11000	0.3	80	EABEL	011.44.1644.429267	11-MAY-96
176	Jonathon	Taylor	8600	0.2	80	JTAYLOR	011.44.1644.429265	24-MAR-98
178	Kimberely	Grant	7000	0.15	(null)	KGRANT	011.44.1644.429263	24-MAY-99
200	Jennifer	Whalen	4400	(null)	10	JWHALEN	515.123.4444	17-SEP-87
201	Michael	Hartstein	13000	(null)	20	MHARTSTE	515.123.5555	17-FEB-96
202	Pat	Fay	6000	(null)	20	PFAY	603.123.6666	17-AUG-97
205	Shelley	Higgins	12000	(null)	110	SHIGGINS	515.123.8080	07-JUN-94
206	William	Gietz	8300	(null)	110	WGIETZ	515.123.8181	07-JUN-94

۱۰.۱۱ UNIQUE CONSTRAINT

زمانی که از این نوع CONSTRAINT برای یک ستون استفاده می شود تمام سطرهای جدول که برای آن ستون دارای مقدار غیر NULL هستند به صورت غیر تکراری و متفاوت از همدیگر خواهند بود ولی می توان هر تعداد از مقدار NULL برای آن ستون داشته باشیم(البته مطابق تعریف مقدار NULL این مقدارها با هم متفاوت هستند). همچنین می توان همزمان برای یک ستون از دو CONSTRAINT از نوع UNIQUE و NOT NULL استفاده نمود. بنابراین فقط زمانی دستورات DML برای یک ستون از نوع UNIQUE اجرا می شوند که مقدار جدید آن قبلا در جدول وجود نداشته باشد.



نکته: یک CONSTRAINT از نوع UNIQUE را می توان در سطح ستون یا جدول تعریف کرد.

مثال: یک CONSTRAINT از نوع UNIQUE در سطح جدول تعریف کنید.

```
CREATE TABLE employees(
    employee_id      NUMBER(6),
    last_name        VARCHAR2(25) NOT NULL,
    email            VARCHAR2(25),
    salary           NUMBER(8,2),
    commission_pct   NUMBER(2,2),
    hire_date        DATE NOT NULL,
...
    CONSTRAINT emp_email_uk UNIQUE(email));
```

نکته: زمانی که یک ستون به عنوان UNIQUE تعریف می شود سرور اوراکل به صورت اتوماتیک یک INDEX برای آن ستون می سازد.

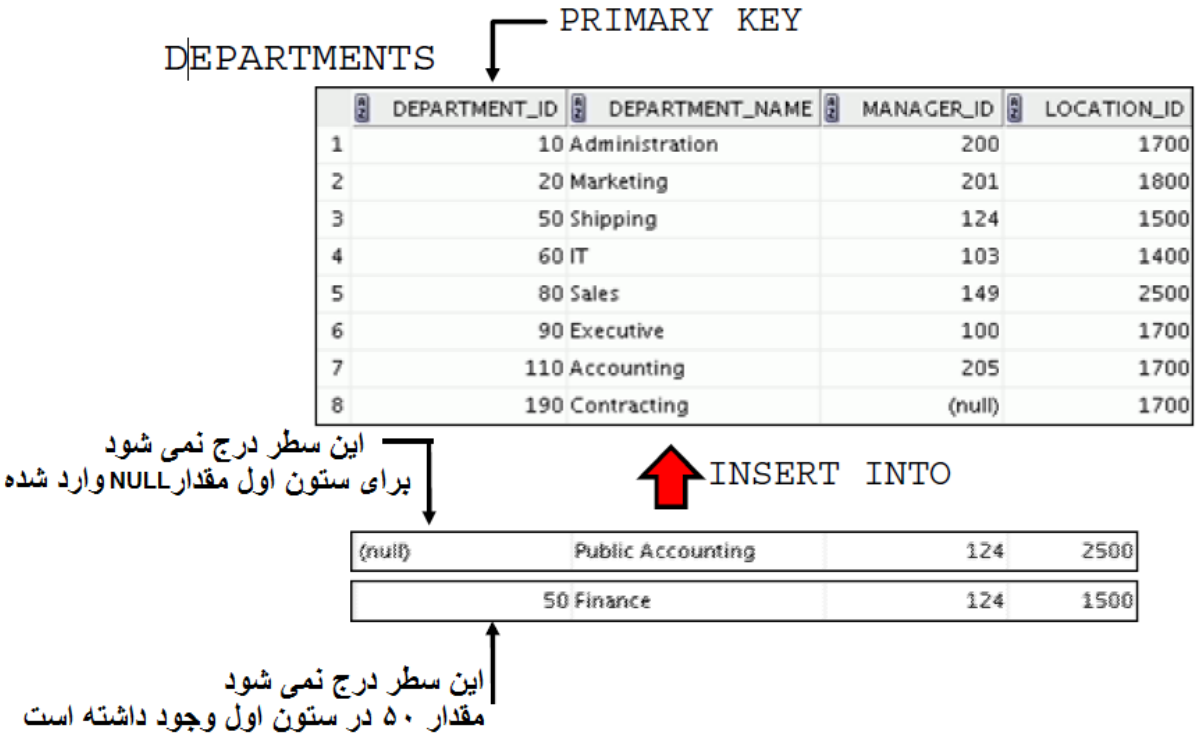
نکته: می توان برای مجموعه ای از ستون های یک جدول به صورت ترکیبی یک CONSTRAINT از نوع UNIQUE تعریف کرد.

PRIMARY KEY CONSTRAINT ۱۰.۱۲

تعریف و کارکرد یکCONSTRAINT از نوع PRIMARY KEY همانند UNIQUE است ولی در دو مورد با هم متفاوت هستند:

۱- در یک ستون از نوع PRIMARY KEY نمی توان مقدار NULL ذخیره کرد. در واقع به صورت اتوماتیک یک CONSTRAINT از نوع NOT NULL در نظرگرفته می شود.

۲- در هر جدول فقط یک PRIMARY KEY می توان تعریف کرد.



نکته: یک CONSTRAINT از نوع PRIMARY KEY را می توان به صورت ترکیبی از ستون ها تعریف کرد.

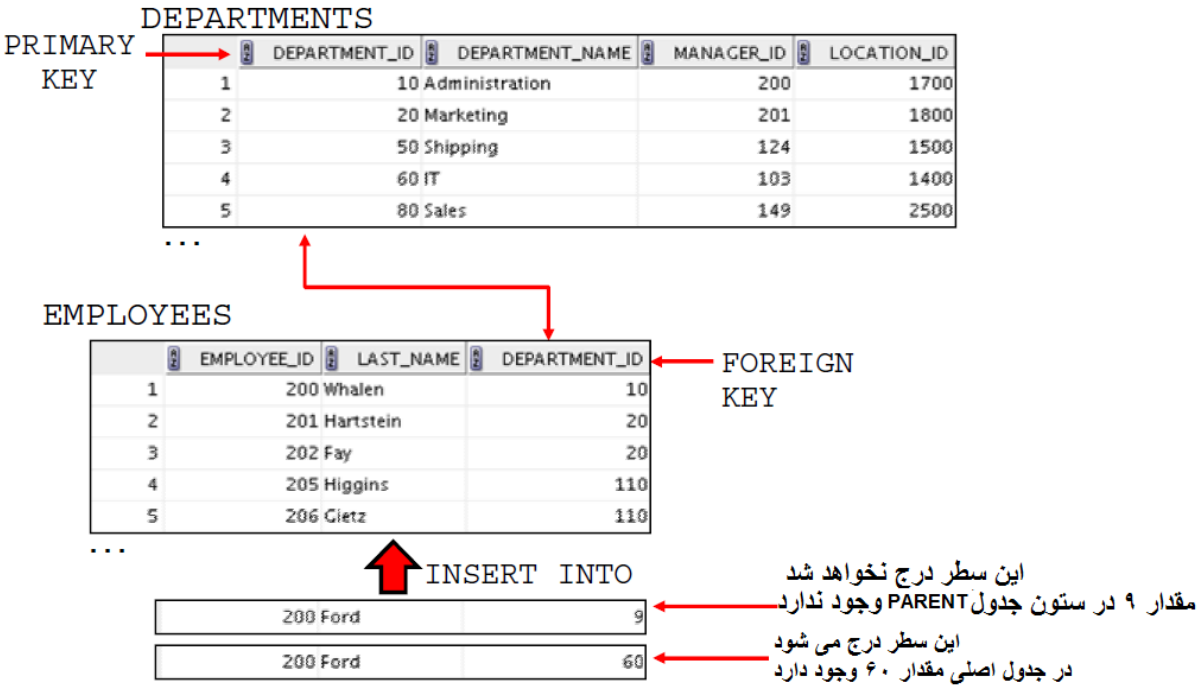
نکته: برای هر ستون از نوع PRIMARY KEY به صورت اتوماتیک توسط سرور توراکل یک INDEX ساخته می شود.

FOREIGN KEY CONSTRAINT ۱۰.۱۳

در این CONSTRAINT ستونی از یک جدول برای مثال جدول B را به عنوان FOREIGN KEY تعریف می کنیم و آن ستون را به ستونی از جدول A که از نوع UNIQUE یا PRIMARY KEY است ارتباط (RELATION) می دهیم. این ارتباط باعث می شود ستون FOREIGN KEY درجدول B فقط مقدارهایی

که قبلا در ستون مرتبط جدول A درج شده است را بپذیرد و درج کند. جدول B جدول CHILD نام دارد و به جدول A، PARENT می گویند. جدول PARENT و ستون مرتبط آن در زمان تعریف FOREIGN KEY و به وسیله عبارت REFRENCES مشخص می شوند. بنابراین هر مقداری که در ستون FOREIGN KEY درج می شود یا باید NULL باشد یا برابر با یکی از مقدارهای درج شده در ستون جدول مرتبط باشد.

نکته: با توجه به تعاریف بالا مقدارهای ستون FOREGN KEY مطابق مقدارهای ستون PARENT می باشند و یک اشاره گر به محل فیزیکی دیتا در جدول PARENT نیستند.



نکته: می توان FOREIGN KEY را در سطح ستون یا در سطر جدول تعریف کرد(همانند CONSTRAINT های قبلی اگر به صورت ترکیبی برای چند ستون تعریف شود باید آن را در سطح جدول تعریف کرد).

مثال: برای شکل بالا یک FOREIGN KEY در سطح جدول تعریف کنید(جدول EMPLYEES جدول CHILD و جدول DEPARTMENTS جدول PARENT است و ستون DEPARTMENT_ID در جدول CHILD به ستون DEPARTMENT_ID در جدول PARENT مرتبط است).

```
CREATE TABLE employees(
    employee_id      NUMBER(6),
    last_name        VARCHAR2(25) NOT NULL,
    email            VARCHAR2(25),
    salary           NUMBER(8,2),
    commission_pct   NUMBER(2,2),
    hire_date        DATE NOT NULL,
...
    department_id    NUMBER(4),
    CONSTRAINT emp_dept_fk FOREIGN KEY (department_id)
      REFERENCES departments(department_id),
    CONSTRAINT emp_email_uk UNIQUE(email));
```

نکته: اگر FOREGN KEY در سطح ستون تعریف شود نیاز نیست از عبارت FOREIGN KEY استفاده شود.

مثال: برای مثال قبلی FOREGN KEY را در سطح ستون تعریف کنید.

```
CREATE TABLE employees
(...
department_id NUMBER(4) CONSTRAINT emp_deptid_fk
REFERENCES departments(department_id),
...
)
```

نکته: اگر در زمان تعریف یک FOREIGN KEY از عبارت ON DELETE CASCADE استفاده شود می توان سطری از جدول PARENT که به جدول CHILD وابسته است را پاک کرد و در این حالت سطرهای وابسته از جدول CHILD نیز به صورت اتوماتیک پاک خواهند شد.

نکته: اگر در زمان تعریف یک FOREIGN KEY از عبارت ON DELETE SET NULL استفاده شود می توان سطری از جدول PARENT که به جدول CHILD وابسته است را پاک کرد و در این حالت ستون های مرتبط از سطرهای وابسته در جدول CHILD به صورت اتوماتیک مقدار NULL می گیرند.

نکته: در دیتابیس اوراکل به صورت پیش فرض از اجرای دستور DELETE و UPDATE بر داده هایی که جدول های دیگر به آنها وابستگی دارند جلوگیری می شود مگر آنکه ON DELETE CASCADE یا SET NULL باشند .

CHECK CONSTRAINT۱۰.۱۴

زمانی که از CONSTRAINT نوع CHECK استفاده شود برای جدول یک شرط تعریف می کنیم که هر تغییری در جدول فقط با داشتن آن شرط امکان پذیر خواهد بود.

نکته: در شرط CHECK نمی توان از تابع SYSDATE استفاده کرد. همچنین نمی توان از دستوری که یک مقدار را در سطرهای دیگر جدول بررسی می کند استفاده کرد.

مثال: در سطح جدول یک CONSTRAINT از نوع CHECK تعریف کنید که هر عمل DML فقط در صورتی اجرا شود که مقدار حقوق آن سطر بیشتر از ۰ باشد.

```
..., salary   NUMBER(2)
     CONSTRAINT emp_salary_min
            CHECK (salary > 0),...
```

نکته: می توان برای هر ستون چندین CHECK CONSTRAINT تعریف نمود.

مثال: CONSTRAINT مثال بالا را در سطح ستون SALARY تعریف کنید.

```
CREATE TABLE employees
    (...
     salary NUMBER(8,2) CONSTRAINT emp_salary_min
                        CHECK (salary > 0),
    ...
```

مثال: ساختن یک جدول به همرا تعریف CONSTRAINT های آن.

```
CREATE TABLE teach_emp (
        empno      NUMBER(5) PRIMARY KEY,
        ename      VARCHAR2(15) NOT NULL,
        job        VARCHAR2(10),
        mgr        NUMBER(5),
        hiredate   DATE DEFAULT (sysdate),
        photo      BLOB,
        sal        NUMBER(7,2),
        deptno     NUMBER(3) NOT NULL
                    CONSTRAINT admin_dept_fkey REFERENCES
                     departments(department_id));
```

۱۰.۱۵ مثال هایی از رعایت نکردن CONSTRAINT ها

در این مثال ها جدول EMPLYEES جدول CHILD و جدول DEPARTMENTS جدول PARENT است و ستون DEPARTMENT_ID در جدول CHILD به ستون DEPARTMENT_ID در جدول PARENT مرتبط است.

مثال ۱: می خواهیم در جدول EMPLOYEES مقدار ستون DEPARTMENT_ID را در سطری که برابر با ۱۱۰ است به ۵۵ تغییر دهیم. از آنجایی که این ستون به جدول PARENT و ستون DEPARTMENT_ID وابسته است و در آن جدول هیچ مقدار ۵۵ در ستون DEPARTMENT_ID وجود ندارد بنابراین این دستور اجرا نمی شود و خطای اوراکل دریافت می کنیم.

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

```
Error starting at line 1 in command:
UPDATE employees
SET    department_id = 55
WHERE  department_id = 110
Error report:
SQL Error: ORA-02291: integrity constraint (ORA1.EMP_DEPT_FK) violated - parent key not found
02291. 00000 - "integrity constraint (%s.%s) violated - parent key not found"
*Cause:    A foreign key value has no matching primary key value.
*Action:   Delete the foreign key or add a matching primary key.
```

مثال ۲: می خواهیم سطری از جدول DEPARTMENT که در آن مقدار ستون DEPARTMENT_ID برابر با ۶۰ است را پاک کنیم. ولی این دستور اجرا نمی شود و خطای اوراکل دریافت می کنیم زیرا در جدول CHILD از این مقدار استفاده شده است.

```
DELETE FROM departments
WHERE department_id = 60;
```

```
Error starting at line 1 in command:
DELETE FROM departments
WHERE department_id = 60
Error report:
SQL Error: ORA-02292: integrity constraint (ORA1.JHIST_DEPT_FK) violated - child record found
02292. 00000 - "integrity constraint (%s.%s) violated - child record found"
*Cause:    attempted to delete a parent key value that had a foreign
           dependency.
*Action:   delete dependencies first then parent or disable constraint.
```

مثال ۳: در مثال قبلی در جدول DEPARTMENTS سطر دارای مقدار ۶۰ را می توانیم پاک کنیم زیرا هیچ سطری از جدول CHILD از این مقدار استفاده نکرده است.

```
DELETE FROM  departments
WHERE department_id = 70;
```

```
1 rows deleted
```

۱۰.۱۶ ساختن جدول با استفاده از SUBQUERY

با استفاده از دستور CREATE TABLE و روش SUBQUERY می توان بطور همزمان یک جدول ساخت و مقدارهای جدول دیگر را در داخل آن درج کرد. البته به شرطی که تعداد ستون ها در دستور CREATE و SUBQUERY یکسان باشد.

```
CREATE TABLE table
            [(column, column...)]
AS subquery;
```

مثال: جدول DEPT80 را همزمان با درج کردن برخی سطرها و ستون های جدول EMPLOYEES در آن بسازید(فقط سطرهایی که مقدار DEPARTMENT_ID برابر با ۸۰ است در جدول جدید درج شوند.).

```
CREATE TABLE  dept80
  AS
    SELECT  employee_id, last_name,
            salary*12 ANNSAL,
            hire_date
    FROM    employees
    WHERE   department_id = 80;
table DEPT80 created.
```

نکته: زمانی که از روش SUBQUERY برای ساخت جدول استفاده می شود به غیر از CONSTRAINT از نوع NULL هیچ CONSTRAINT دیگری از جدول قبلی به جدول جدید منتقل نمی شود. البته اگر CONSTRAINT از نوع NULL به صورت صریح تعریف شده باشد یعنی اگر یک ستون از نوع PRIMARY KEY باشد CONSTRAINT از نوع NULL به جدول جدید منتقل نمی شود.

مثال: در مثال قبل دو ستون از جدول جدید با CONSTRAINT از نوع NULL ساخته شده اند زیرا در جدول قبلی به صورت صریح تعریف شده اند.

```
DESCRIBE dept80
```

Name	Null	Type
EMPLOYEE_ID		NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
ANNSAL		NUMBER
HIRE_DATE	NOT NULL	DATE

نکته: اگر در روش ساخت جدول با SUBQUERY یکی از ستون ها به صورت یک عبارت محاسبه ای بود(SALARY*12 در مثال قبل) باید از یک نام مستعار استفاده کرد(ANNSAL در مثال قبل) وگرنه خطای اوراکل دریافت می کنیم.

مثال: در دستور مثال قبل از نام مستعار استفاده نشده است. بنابراین خطای اوراکل دریافت می کنیم.

```
Error starting at line 1 in command:
CREATE TABLE  dept80
  AS
    SELECT  employee_id, last_name,
            salary*12,
            hire_date
    FROM    employees
    WHERE   department_id = 80
Error at Command Line:4 Column:18
Error report:
SQL Error: ORA-00998: must name this expression with a column alias
00998. 00000 -  "must name this expression with a column alias"
*Cause:
*Action:
```

۱۰.۱۷ دستور ALTER TABLE

دستور ALTER TABLE از نوع DDL است بنابراین بعد از اجرای آن به صورت اتوماتیک توسط دیتابیس یک عمل COMMIT انجام می شود. با استفاده از این دستور می توان عملیات زیر را برای یک جدول انجام داد:

- تغییر نام ستون های یک جدول.
- اضافه کردن یک یا چند ستون به جدول.
- تغییر نام ستون ها.
- تغییر سایز یا نوع داده ستون ها.

نکته: فقط اگر یکی از حالت های زیر برقرار باشد می توان سایز یک ستون را با دستور ALTER TABLE کاهش دهیم:

- جدول هیچ سطری نداشته باشد.
- آن ستون از جدول فقط مقدار NULL داشته باشد.

- کاهش سایز کمتر از سایز تمامی مقدارهای آن جدول برای آن ستون باشد.

نکته: فقط زمانی که تمام مقدارهای یک ستون NULL باشد می توان با دستور ALTER TABLE نوع داده آن ستون را عوض کرد. البته برای تبدیل نوع داده CHAR به VARCHAR2 نیازی به NULL بودن مقدارهای آن ستون نیست.

۱۰.۱۷.۱ عبارت DROP در دستور ALTER TABLE

می توان با استفاده از عبارت DROP COLOUMN در دستور ALTER TABLE ستون های یک جدول را حذف نمود.

مثال: ستون JOB_ID از جدول DEPT80 را حذف کنید.

```
ALTER TABLE  dept80
DROP (job_id);
```

```
table DEPT80 altered.
```

	EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE
1	149	Zlotkey	10500	29-JAN-08
2	174	Abel	11000	11-MAY-04
3	176	Taylor	8600	24-MAR-06

نکته: در هر دستور ALTER TABLE فقط می توان یک ستون را با استفاده از عبارت DROP حذف کرد البته هر جدول باید حداقل یک ستون داشته باشد و آن را نمی شود DROP کرد.

نکته: اگر یک ستون دارای CONSTRAINT از نوع PRIMARY KEY باشد و اطلاعات جدول دیگر به آن ستون وابسته باشد(رابطه CHID و PARENT) نمی توان آن ستون را حذف کرد مگر اینکه CONSTRANT از نوع FOREIGN KEY به روش SET NULL یا CASCADE تعریف شده باشد.

۱۰.۱۷.۲ عبارت SET UNUSED در دستور ALTER TABLE

زمانی که یک دستور ALTER TABLE با عبارت DROP برای حذف کردن یک ستون استفاده می شود اگر سایز آن ستون زیاد باشد منابع سرور مشغول می شوند و ممکن است سبب کندی سیستم شود. بنابراین می توان با استفاده از عبارتSET UNUSED یک ستون را به عنوان ستون حذف شده تعیین کرد و آن را غیر فعال کنیم تا در زمان مناسب DROP شود و فضای ذخیره ساز آزاد گردد. زمانی که یک ستون را SET

UNUSED می کنیم دقیقا مانند زمانی است که آن ستون را حذف می کنیم با این تفاوت که تا قبل از استفاده از یک دستور با عبارت DROP همچنان فضای دیسک را اشغال کرده است. در شکل زیر سینتکس این عبارت را به همراه یک مثال می بینید.

```
ALTER TABLE <table_name>
SET   UNUSED(<column_name> [ , <column_name>]);
OR
ALTER TABLE  <table_name>
SET   UNUSED COLUMN <column_name> [,<column_name>];
```

```
ALTER TABLE <table_name>
DROP  UNUSED COLUMNS;
```

مثال: ستون LAST_NAME را UNUSED کنید و سپس آن ستون را در زمان غیر اداری و کم مشتری از دیتابیس حذف کنید تا فضای آن آزاد شود.

```
ALTER TABLE  dept80
SET   UNUSED (last_name);
table DEPT80 altered.
```

```
ALTER TABLE  dept80
DROP  UNUSED COLUMNS;
table DEPT80 altered.
```

نکته: اگر در یک دستور از کلمه ONLINE استفاده شود به این معنی می باشد که در زمان اجرای آن دستور هرگونه عملیات از نوع DML در جدول به صورت همزمان می تواند اجرا شود و نیاز نیست جدول LOCK شود و دستورات DML منتظر این دستور بمانند. البته وقتی از عبارت ONLINE استفاده می شود یک زمان کوتاه در ابتدا و انتهای دستور عمل LOCK انجام می شود.

مثال: در زمان UNUSED کردن ستون HIRE_DATE عملیات DML انجام شود.

```
ALTER TABLE dept80 SET UNUSED(hire_date)ONLINE;
```

۱۰.۱۷.۳ عبارت READ ONLY در دستور ALTER TABLE

می توانیم با استفاده از عبارت READ ONLY در دستور ALTER TABLE یک جدول را در حالت فقط خواندنی قرار دهیم. هیچ کدام از دستورات DML و DDL برای جدولی که در حالت فقط خواندنی است اجرا نخواهد شد. بنابراین نمی توان اطلاعات آن جدول را تغییر داد.

مثال: جدول EMPLOYEES را READ ONLY کنید و سپس به حالت اول (READ WRITE) برگردانید.

```
ALTER TABLE employees READ ONLY;

-- perform table maintenance and then
-- return table back to read/write mode

ALTER TABLE employees READ WRITE;
```

نکته: می توان جدولی که در حالت READ ONLY قرار دارد را با دستور DROP TABLE به صورت کامل حذف کرد.

۱۰.۱۸ دستور DROP TABLE

دستور DROP TABLE یک دستور از نوع DDL است و برای حذف کردن یک جدول استفاده می شود. زمانی که این دستور برای یک جدول استفاده می شود موارد زیر اتفاق می افتند:

- جدول به RECYCLE BIN یا سطل آشغال دیتابیس منتقل می شود ولی فضای دیسک آن آزاد نمی شود مگر اینکه از عبارت PURGE در دستور DROP TABLE استفاده شود.
- تمام INDEX های مربوط به آن جدول حذف می شوند.
- VIEW ها و SYNONYM های مربوط به آن جدول حذف نمی شوند ولی INVALID می شوند.
- تراکنش های در لیست انتظار مربوط به این جدول COMMIT می شوند.

نکته: دستور DROP TABLE فقط توسط کاربری که مالک آن جدول است یا مجوز DROP ANY TABLE دارد می تواند اجرا شود.

مثال: جدول DEPT80 را حذف کنید.

```
DROP TABLE dept80;

table DEPT80 dropped.
```

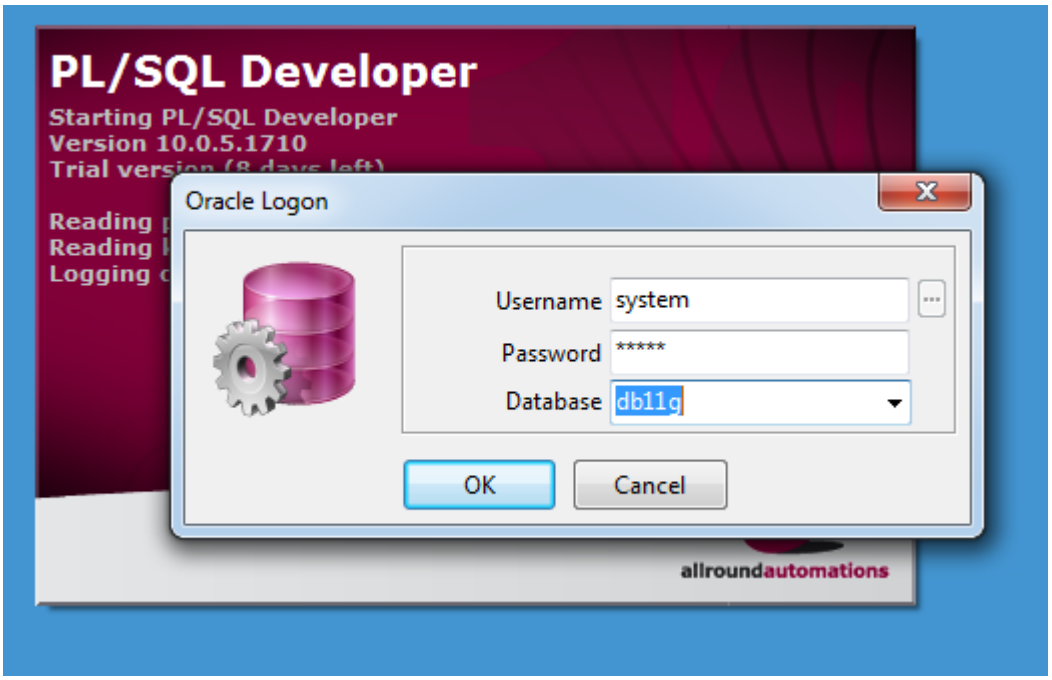
۱۱ ابزارهای گرافیکی برای کار با دیتابیس اوراکل

زمانی که برخی از بانک های اطلاعاتی مانند SQL SERVER را نصب می کنیم، همراه با دیتابیس یک ابزار گرافیکی مناسب نصب می شود که می توانیم با استفاده از آن به بانک اطلاعاتی مقصد متصل شویم و دستورات مختلف را اجرا کنیم. اما در پکیج نصب دیتابیس اوراکل ابزار گرافیکی مناسب وجود ندارد و باید به صورت مجزا این ابزار را تهیه و نصب کنیم. علاوه بر مدیران دیتابیس، برنامه نویسان و توسعه دهندگان هم نیازمند یک ابزار گرافیکی جهت پیش برد پروژه های خود هستند.

در ادامه سه مورد از ابزارهای گرافیکی پرکاربرد را معرفی می کنیم. همچنین روش پیکربندی آنها و نحوه اتصال این ابزار به دیتابیس اوراکل توضیح داده می شود. برای مشاهده کاراکترها و متن های فارسی در دیتابیس اوراکل بهتر است تنظیماتی در سطح سیستم عامل و ابزار گرافیکی انجام شود که آنها را در این فصل عنوان می کنیم.

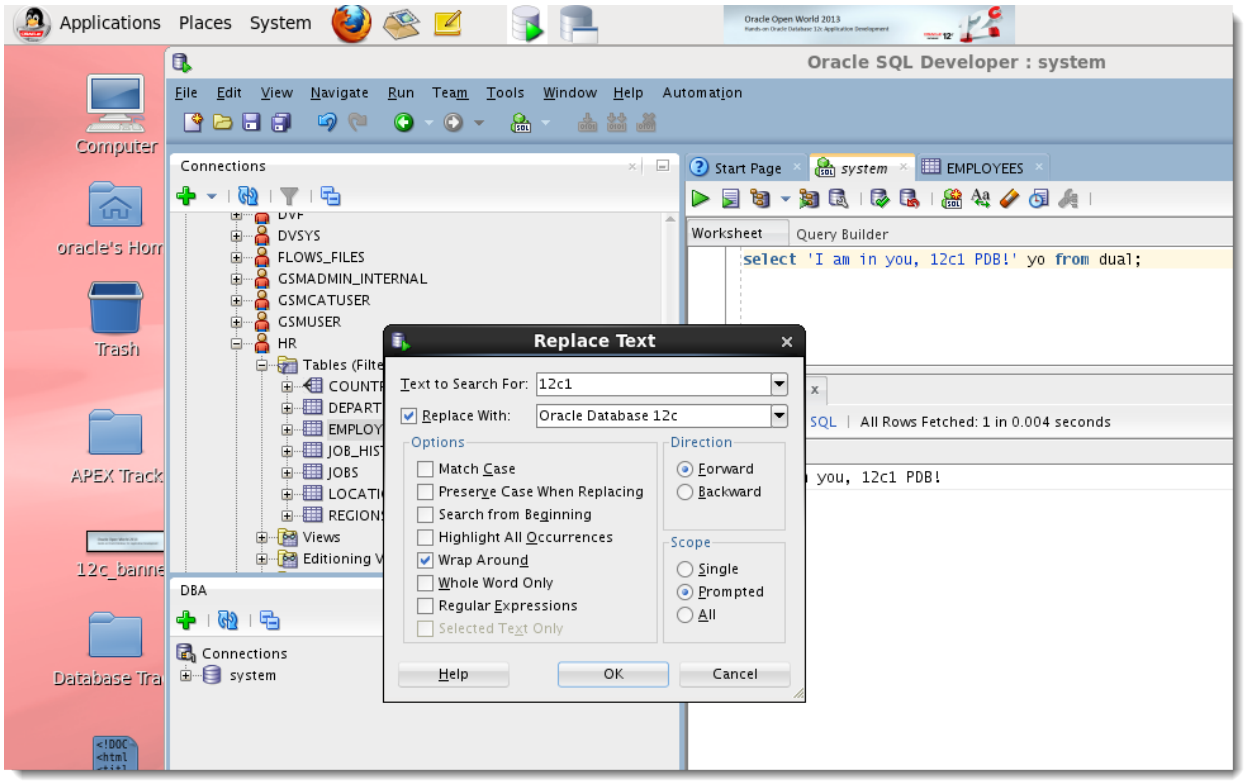
۱۱.۱ PL/SQL DEVELOPER

یکی از پرکاربردترین نرم افزارهای تجاری که امکان کار با دیتابیس اوراکل را فراهم می کند ابزار PL/SQL DEVELOPER است. این نرم افزار توسط شرکت Allround Automations روانه بازار شده است.



ORACLE SQL DEVELOPER ۱۱.۲

یکی از ابزارهایی که توسط شرکت اوراکل تهیه و توزیع گردیده است ابزار SQL DEVELOPER می باشد و می توان آن را به صورت رایگان از سایت اوراکل دانلود کرد.



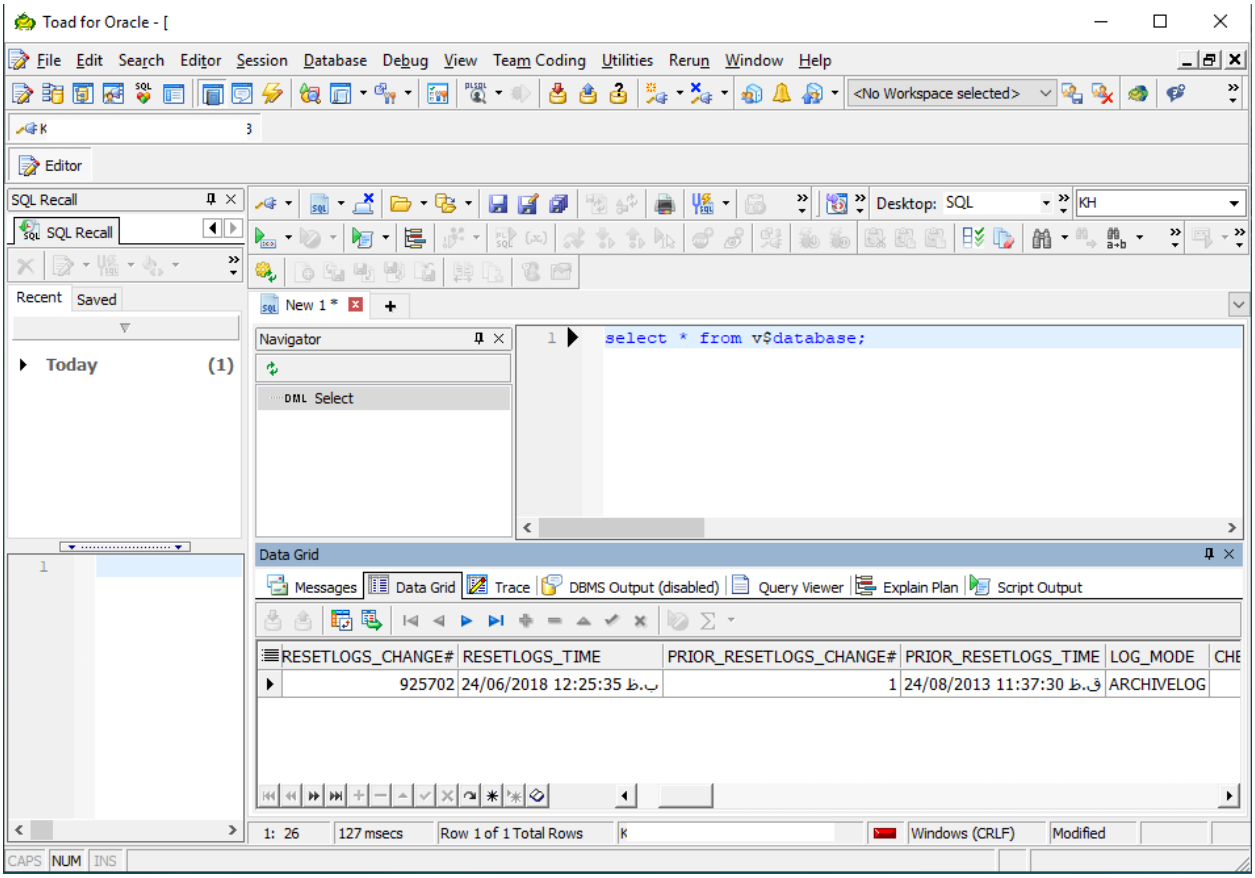
این ابزار قابلیت نصب و اجرا در سیستم عامل های ویندوز و لینوکس را دارد. البته در کنار این نرم افزار باید ابزار JDK نصب شود.

نکته: اگر در اجرای نرم افزار SQL DEVELOPER به مشکل بر می خورید می بایست در فایل پیکربندی sqldeveloper.conf در مسیر نرم افزار و در پوشه bin یکسری از تنظیمات انجام شود و دوباره نرم افزار را اجرا کنید.

نکته: همراه نصب دیتابیس اوراکل یا اوراکل کلایت نگارشی از نرم افزار sql developer نیز نصب می شود.

TOAD FOR ORACLE ۱۱.۳

ابزار TOAD یک نرم افزار از نوع تجاری می باشد که توسط شرکت QUEST SOFTWARE طراحی و توسعه یافته است. این نرم افزار به نسبت دو ابزار قبلی امکانات بیشتری دیگر جهت کار با دیتابیس اوراکل در اختیار می گذارد.



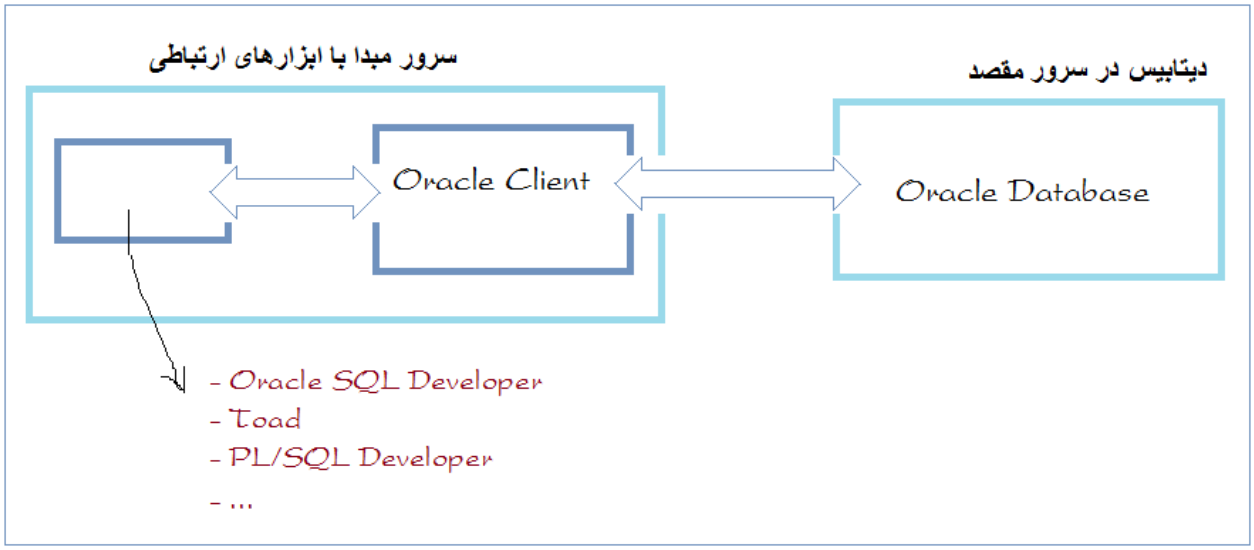
نکته: قبل از نصب این نرم افزار باید NET FRAMEWORK راه اندازی شود.

۱۱.۴ روش های پیکربندی ابزار گرافیکی

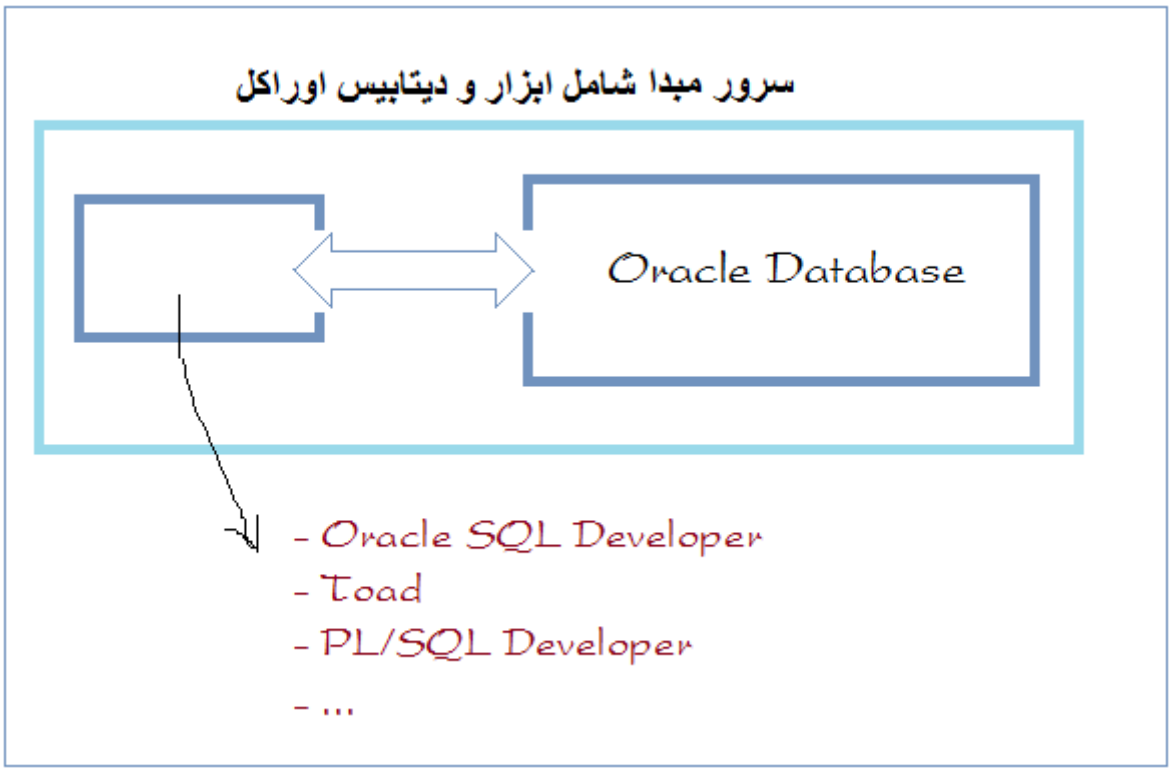
در این قسمت دو مدل مختلف پیکربندی ابزار گرافیکی را توضیح می دهیم که برای دو ابزار TOAD و PL/SQL DEVELOPER صدق می کند.

مدل اول :در این مدل دیتابیس در سرور مقصد است و ابزار گرافیکی در سرور مبدا نصب می باشد. در این حالت باید یک نرم افزار **اوراکل کلاینت** یا **اوراکل دیتابیس** در سرور مبدا و در کنار PL/SQL DEVELOPER و یا TOAD نصب کنیم تا بتوانیم از طریق آن به دیتابیس مقصد متصل شویم.

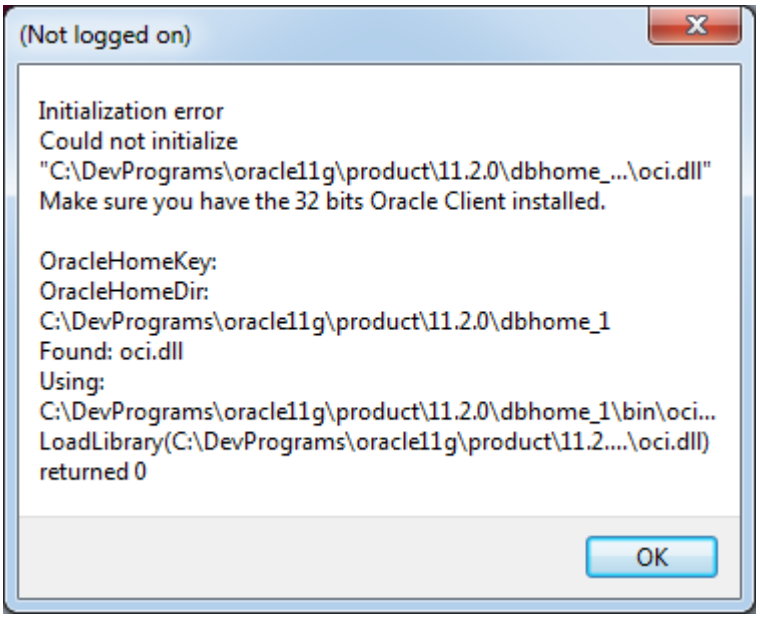
نکته: ابزار SQL DEVELOPER نیازی به نصب اوراکل کلاینت یا اوراکل دیتابیس ندارد.



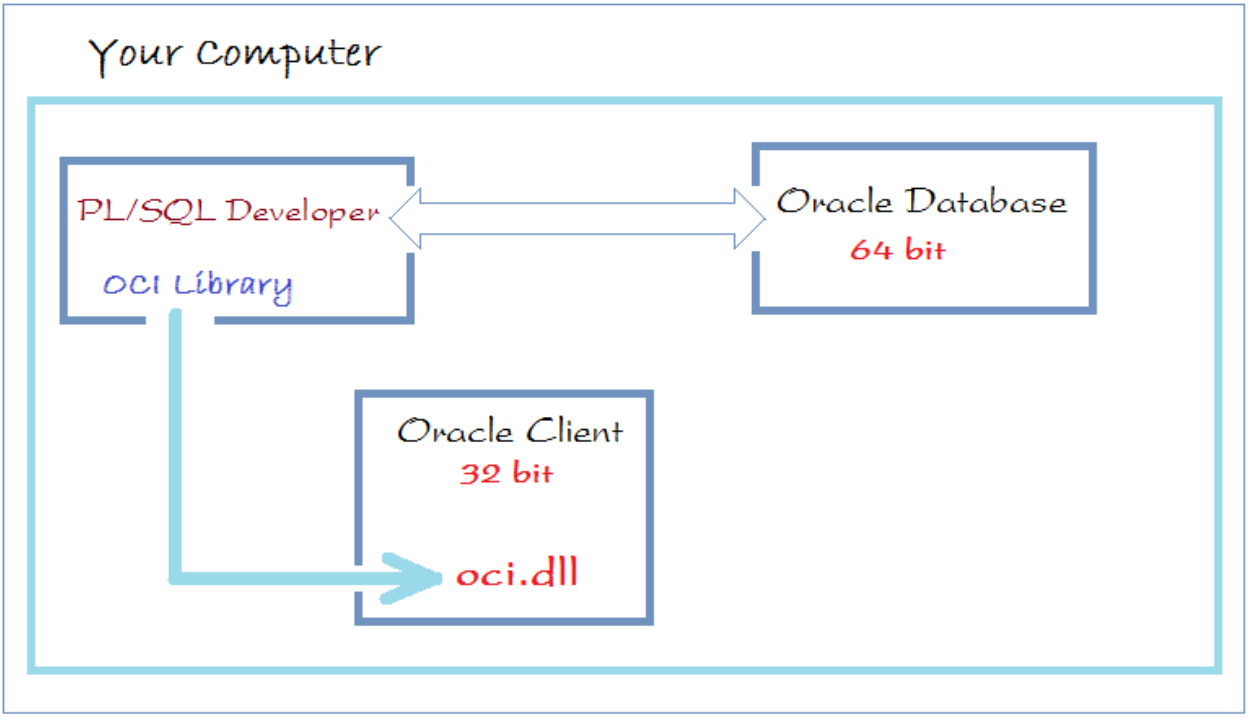
مدل دوم: در این حالت یک دیتابیس در سرور مبدا و در کنار نرم افزارهای گرافیکی داریم که می خواهیم به آن متصل شویم. بنابراین دیگر نیاز نیست در سرور مبدا اوراکل کلاینت یا اوراکل دیتابیس را نصب کنیم.



نکته: اگر نرم افزار PL/SQL DEVELOPER ۳۲ بیتی باشد باید در هرکدام از مدل های ۱ یا ۲، یک اوراکل کلاینت ۳۲ بیتی نصب کنیم وگرنه خطای زیر را دریافت می کنیم که بیانگر ناسازگاری از لحاظ ۳۲ بیتی یا ۶۴ بیتی بودن نرم افزار گرافیکی می باشد.



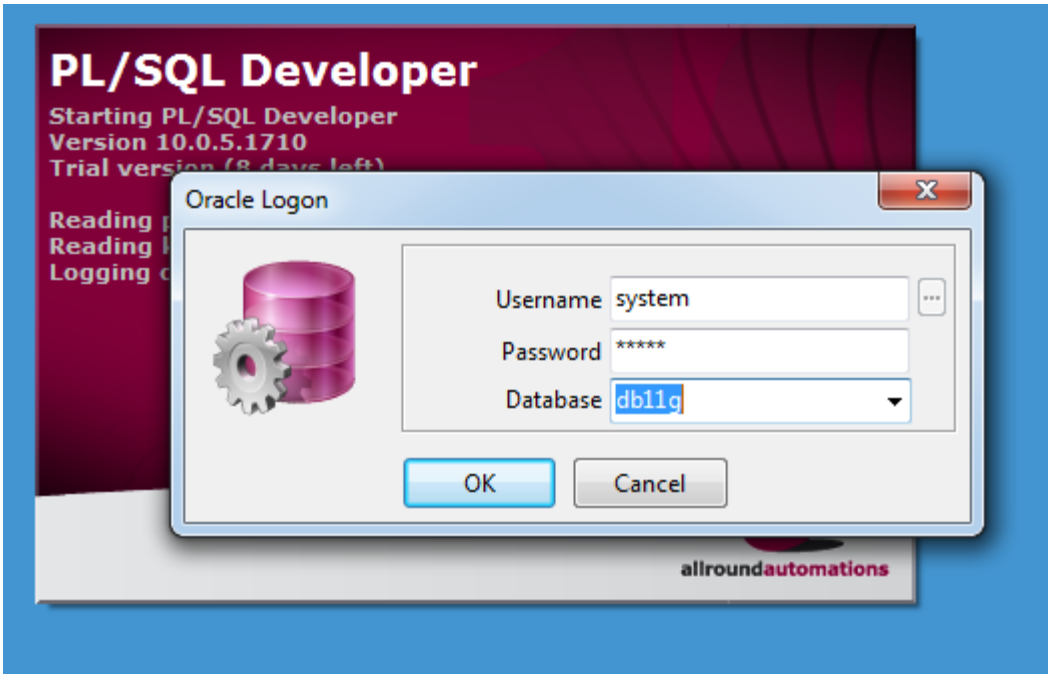
بنابراین هر زمان نرم افزار PL/SQL DEVELOPER ۳۲ بیتی باشد باید یک اوراکل کلاینت ۳۲ بیتی نصب کنیم تا از فایب کتابخانه ای مناسب استفاده شود(OCI.DLL).



۱۱.۵ روش اتصال به دیتابیس اوراکل توسط ابزارهای گرافیکی

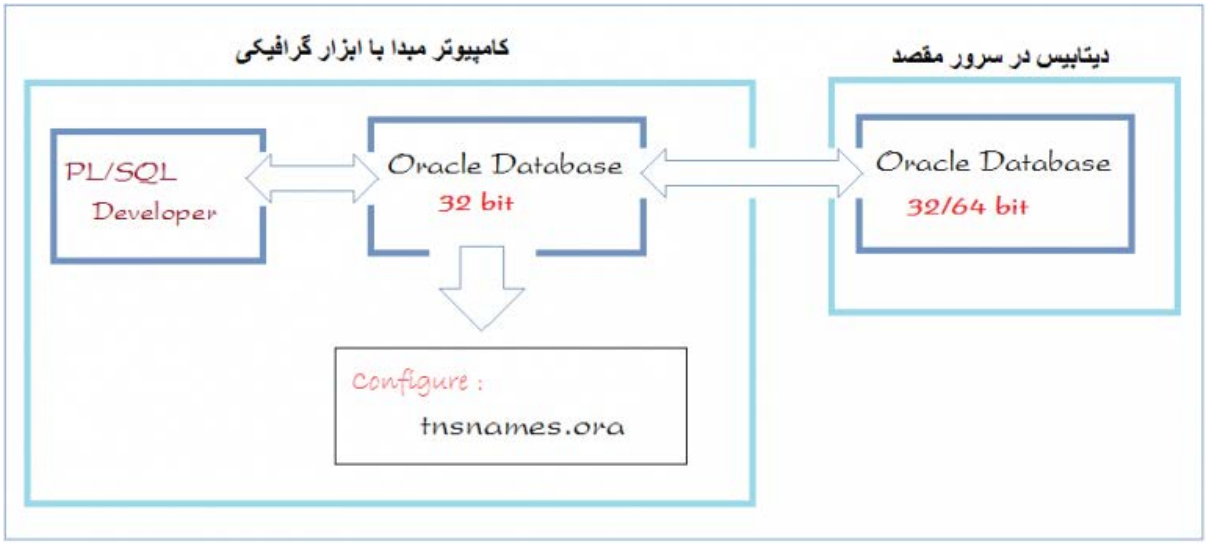
در این قسمت سه مدل ارتباطی بین دیتابیس اوراکل و نرم افزار PL/SQL DEVELOPER را توضیح می دهیم. روش برقراری ارتباط در این مدل ها برای تمام ابزارهای گرافیکی به همین روال است. البته همانطور که در قسمت قبل عنوان شد نرم افزار SQL DEVELOPER برای اتصال به سرور مقصد نیازی به نصب اوراکل کلاینت ندارد.

مدل ۱ :ابتدا حالتی را درنظر می گیریم که در سرور مبدا و در کنار ابزار PL/SQL DEVELOPER یک دیتابیس اوراکل ایجاد شده است و هردوی آنها ۳۲ بیتی هستند(یا هردو ۶۴ بیتی هستند). بنابراین در صفحه اول نرم افزار PL/SQL DEVELOPER نام کاربری، رمزعبور و SID دیتابیس را وارد می کنیم و متصل می شویم.



نکته: اگر ابزار PL/SQL DEVELOPER از نوع ۳۲ بیتی باشد و بخواهیم به یک دیتابیس که در همین سرور قرار دارد و ۶۴ بیتی است متصل شویم باید یک اوراکل کلاینت ۳۲ بیتی نصب کنیم تا ابزار PL/SQL بتواند از فایل کتابخانه ای مناسب استفاده کند.

مدل ۲: اگر در سرور مبدا یک نرم افزار اوراکل دیتابیس ۳۲ بیتی در کنار یک نرم افزار PL/SQL DEVELOPER از نوع ۳۲ بیتی داریم می توانیم به دیتابیس سرور مقصد (دیتابیس مقصد از نوع ۶۴ بیتی یا ۳۲ بیتی) متصل شویم.



همچنین به منظور ایجاد سهولت در متصل شدن به دیتابیس می توانیم در نرم افزار اوراکل دیتابیس در سرور مبدا فایل tnsnames.ora را تنظیم کنیم. آدرس فایل tnsnames.ora در مسیر زیر است که ORACLE_INSTALL_DIRECTORY همان مسیر نصب اوراکل دیتابیس در سرور مبدا می باشد.

<ORACLE_INSTALL_DIRECTORY>/product/11.2.0/dbhome_1/NETWORK/ADMIN

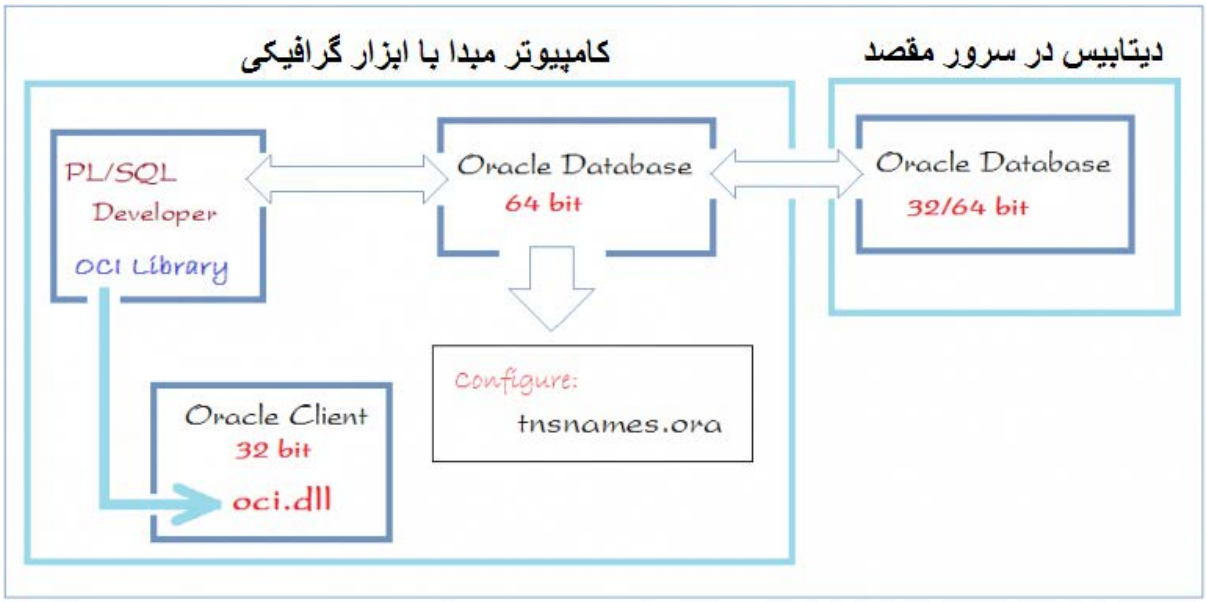
در این فایل برای تنظیم TNS دیتابیس مقصد موارد زیر را اضافه می کنیم.

tnsnames.ora

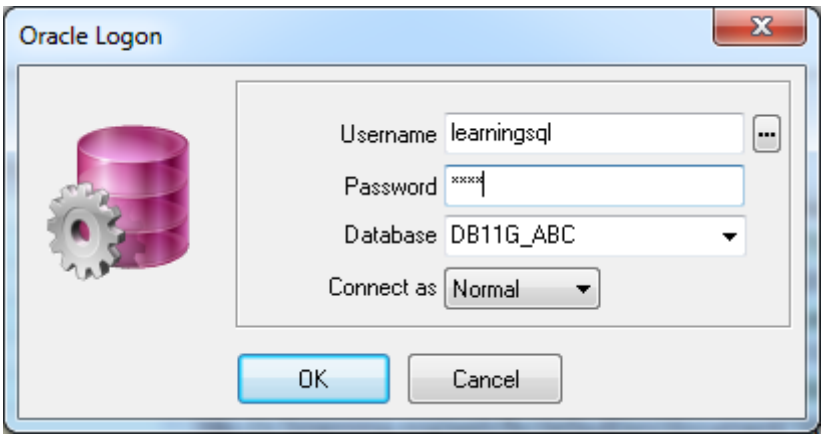
```
1  DB11G_ABC =
2  (DESCRIPTION =
3    (ADDRESS = (PROTOCOL = TCP)(HOST = host_abc)(PORT = 1521))
4    (CONNECT_DATA =
5      (SERVER = DEDICATED)
6      (SERVICE_NAME = SID_ABC)
7    )
8  )
```

- **SID_ABC** : SID دیتابیس مقصد که می خواهیم به آن متصل شویم
- **host_abc** : IP ADDRESS یا HOSTMNAME مقصد که می خواهیم به آن متصل شویم
- **DB11G_ABC** : یک نام برای TNS

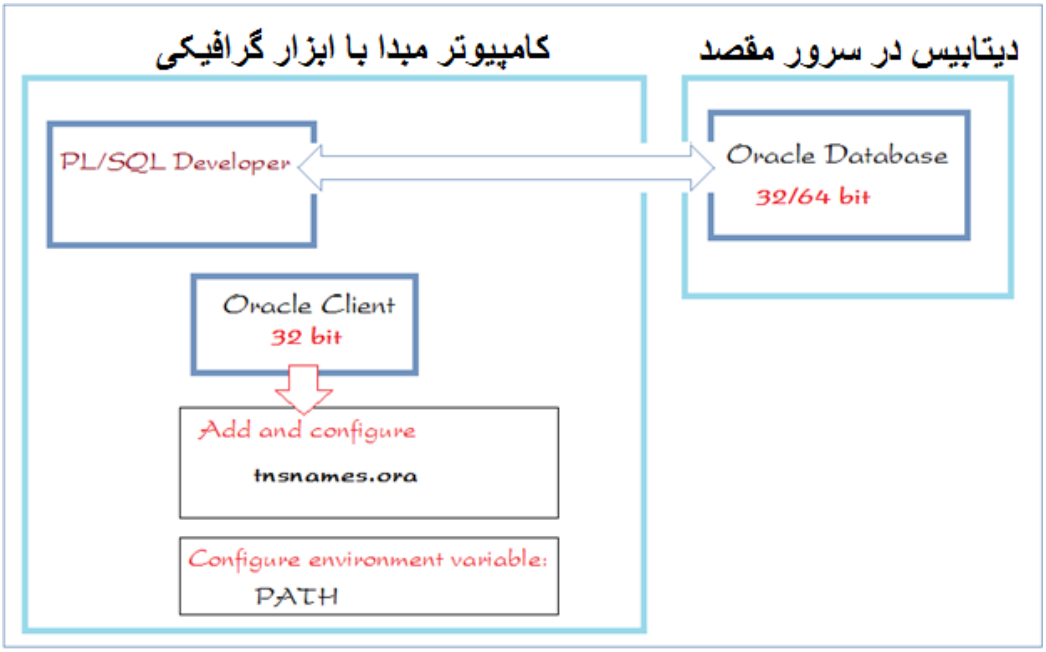
اگر در سرور مبدا اوراکل دیتابیس از نوع ۶۴ بیتی و نرم افزار PL/SQL DEVELOPER از نوع ۳۲ بیتی باشد باید یک اوراکل کلاینت ۳۲ بیتی نصب کنیم تا فایل کتابخانه ای مناسب را داشته باشیم.



در این حالت با وارد کردن نام TNS تنظیم شده در قسمت دیتابیس و نام کاربری و کلمه عبور به دیتابیس متصل می شویم.

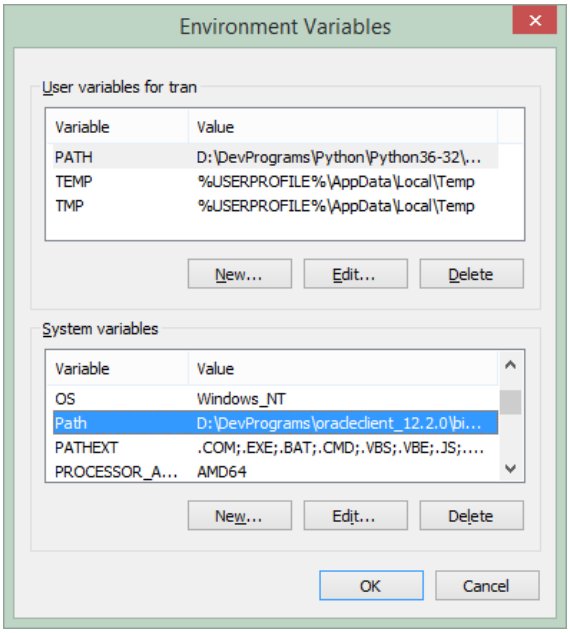


مدل ۳: در این حالت نرم افزار PL/SQL DEVELOPER از نوع ۳۲ بیتی است و یک اوراکل کلاینت ۳۲ بیتی نصب می کنیم و دیگر اوراکل دیتابیس نصب نمی کنیم. بنابراین این اوراکل کلاینت فایل کتابخانه ای OCI.DLL را فراهم می کند. همچنین می توانیم TNS مورد نیاز را در فایل TNSNAMES.ora مربوط به اوراکل کلاینت تنظیم کنیم.

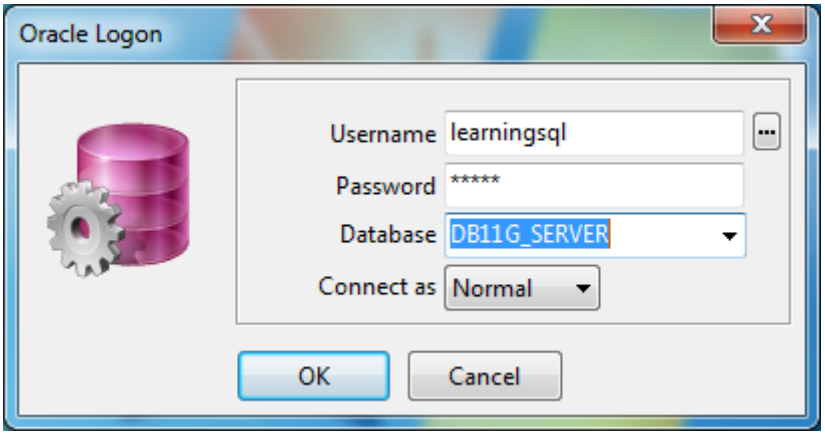


همچنین باید متغیر PATH را تنظیم کنیم تا به ORACLE_HOME اشاره کند(در صورتی که تنظیم نشده باشد). در این متغیر مسیرهای دیگر با علامت ; جدا می شوند. قسمت مربوط به متغیرها در ویندوز در میسر **control panel / system / advanced system settings** و قسمت **environment variables** قرار دارد.

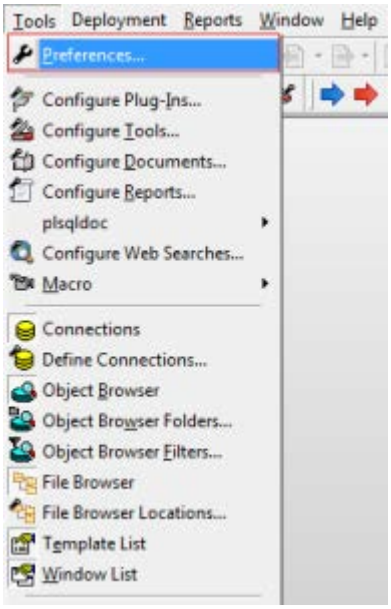
Path=D:\DevPrograms\oracleclient_12.2.0\bin;……



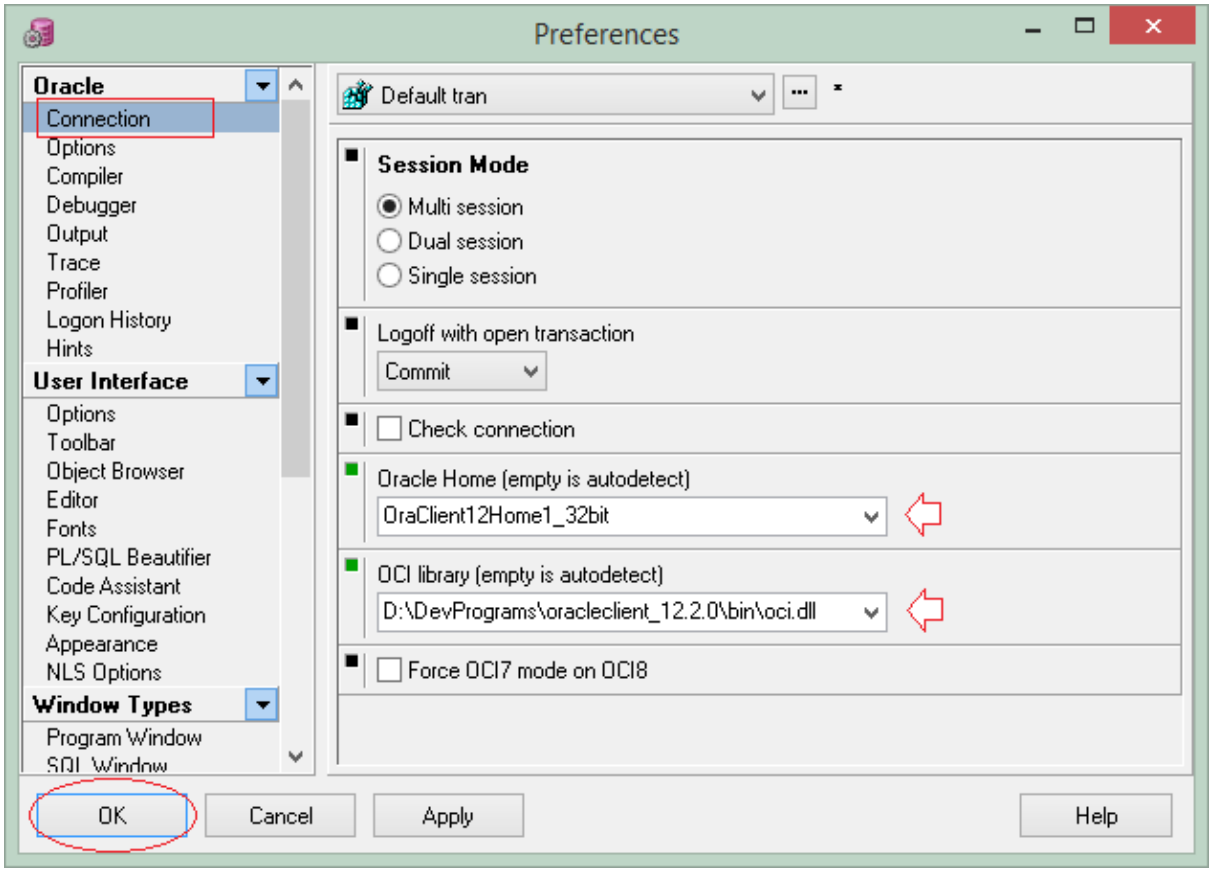
در نهایت می توانیم با وارد کردن نام TNS تنظیم شده، نام کاربری و کلمه عبور به دیتابیس متصل شویم.



نکته: در هرکدام از سه مدل بالا اگر یک اوراکل کلاینت یا اوراکل دیتابیس جدید نصب شود باید مسیر آن در ابزار PL/SQL DEVELOPER تنظیم شود. برای این منظور ابتدا PL/SQL DEVELOPER را باز کنید و سپس از منوی TOOLS گزینه PREFRENCES را انتخاب کنید.



سپس در قسمت ORACLE HOME آدرس اوراکل کلاینت و در قسمت OCI library آدرس فایل OCI.DLL را وارد می کنیم.



۱۱.۶ نکات مهم در فارسی سازی PL/SQL DEVELOPER

اگر تنظیمات زیر رعایت نشود ابزارهای گرافیکی کاراکترهای فارسی را به شکل علامت سوال یا نامفهوم نشان می دهند. به منظور اینکه نرم افزار گرافیکی ما (PL/SQL DEVELOPER و TOAD) بتوانند کارکترهای فارسی ذخیره شده در دیتابیس اوراکل را به درستی نمایش دهند موارد زیر را در سیستم عامل ویندوز انجام می دهیم:

۱- در محیط ویندوز وارد کنترل پنل شده و سپس REGION AND LANGUAGE را انتخاب کنید در قسمت Administrative باید Change System Locale را انتخاب کرده و Current System Locale را در حالت PERSIAN تنظیم کرد.

۲- در محیط PL/SQL DEVELOPER و در منوی Configure گزینه Preferences را می زنیم و در قسمت Fonts برای تمام فونت ها بر روی Select کلیک کرده و درقسمت Script زبان Arabicرا نتخاب می کنیم و Apply می کنیم.

۳- وارد ریجستری کامپیوتر مبدا که در کنار ابزار گرافیکی یک اوراکل کلاینت یا اوراکل دیتابیس نصب کردیم می شویم و وارد قسمت HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE می شویم و سپس بر روی KEY_OraDBversionHOME که مربوط به اوراکل کلاینت یا اوراکل دیتابیس ما می باشد کلیک کرده و مقدار متغیر NSL_LANG را برابر با AR8MSWIN1256 تنظیم می کنیم. توجه شود که KEY_OraDBversionHOME بسته به مدل اوراکل دیتابیس یا اوراکل کلاینت یک نام متفاوت دارد. همچنین می توان رجیستری را برای متغیر NLS_LANG جستجو کرد و بعد از پیدا کردن این متغیر، تنظیم لازم را انجام دهیم.

نکته: در دیتابیس اوراکل هر کاراکترست یکسری از حروف و زبان های خاص را پشتیبانی می کند. موقع ایجاد دیتابیس یک کاراکترست برای آن انتخاب می کنیم که اگر بعدا بخواهیم آن را تغییر دهیم فقط می توانیم کاراکترست هایی که زیرمجموعه آن کاراکترست است را انتخاب کنیم وگرنه ممکن است اطلاعات دیتابیس از بین برود.
بنابرین در رجیستری ویندوز و پس از نصب اوراکل کلاینت یا اوراکل دیتابیس بهتر است برای ارتباط ویندوزی این ابزار با دیتابیس ها یک کاراکترست مناسب انتخاب کنیم که معمولا برای نمایش کاراکترهای فارسی / عربی از AMERICAN_AMERICA.AR8MSWIN1256 استفاده می کنیم.

نکته: در زمان جستجوی LIKE کاراکترهای فارسی اگر از علامت های ٪ استفاده می کنیم باید این علامت را با فونت انگلیسی تایپ کنیم.

۱۲ Sequence در دیتابیس اوراکل

همانطور که می دانید در دیتابیس اوراکل OBJECT های مختلفی وجود دارند که یکی از این OBJECT ها SEQUENCE است که در این فصل معرفی می شود و روش ساخت و بکارگیری آن را به همراه مثال توضیح می دهیم.

گاهی اوقات در دیتابیس نیاز به تولید اعداد جدید و ذخیره آن اعداد است. برای مثال می خواهیم برای هر کارمند که استخدام می شود یک عدد به عنوان شماره پرسنلی تولید شود و در دیتایس ذخیره شود. برای این منظور می توان از SEQUENCE استفاده کرد.

هر SEQUENCE یک عدد صحیح تولید می کند و می تواند به صورت اشتراکی توسط چندین کاربر مورد استفاده قرار گیرد. بعضی از ویژگی های SEQUENCE عبارتند از:

- اعداد تولیدی توسط یک SEQUENCE می تواند صعودی یا نزولی باشد. برای مثال می تواند نزولی باشد و از ۱۰۰۰ شروع شود و در ۲۰۰۰- تمام شود.
- می تواند اعداد واحد (UNIQUE) تولید کند.
- می توان مقادیری که توسط SEQUENCE ایجاد می شود را به عنوان PRIMARY KEY یک جدول در نظر گرفت.
- زمانی که در دیتابیس از یک SEQUENCE استفاده می شود کد نویسی در سطح APPLICATIONکاهش می یابد و در زمان صرفه جویی می شود.
- تولید و ذخیره سازی اعداد توسط SEQUENCE ها مستقل از OBJECT های دیگر است بنابراین می توان از یک SEQUENCE برای چند جدول استفاده نمود.

در شکل زیر سینتکس دستور ساخت SEQUENCE را می بینید.

```
CREATE SEQUENCE [ schema. ] sequence
  [ { START WITH|INCREMENT BY } integer
  | { MAXVALUE integer | NOMAXVALUE }
  | { MINVALUE integer | NOMINVALUE }
  | { CYCLE | NOCYCLE }
  | { CACHE integer | NOCACHE }
  | { ORDER | NOORDER }
];
```

- استفاده از قسمت هایی که در داخل علامت [ ] نوشته شده اند اختیاری است.

- START WITH n :شماره اولین SEQUENCE که تولید می شود برابر با n خواهد بود. اگر از این عبارت در دستور ساخت استفاده نشود به صورت پیش فرض n برابر با یک است.

- INCREMENT BY n: اختلاف میان هر عدد تولیدی برابر با n خواهد بود. مقدار n به طور پیش فرض برابر با یک است.

- MAXVALUE n یا NOMAXVALUE : اگر از عبارت MAXVALUE n استفاده شود بزرگترین عددی که یک SEQUENCE می تواند تولید کند برابر با n خواهد بود. ولی اگر از عبارت NOMAXVALUE استفاده شود مقدار حداکثر برای SEQUENCE ها درنظر گرفته می شود. به طور پیش فرض اگر از هیچ کدام از این عبارات استفاده نشود NOMAXVALUE در نظر گرفته می شود.

- MINVALUE n یا NOMINVALUE :اگر از عبارت MINVALUE n استفاده شود کوچکترین عددی که یک SEQUENCE می تواند تولید کند برابر با n خواهد بود. ولی اگر از عبارت NOMINVALUEاستفاده شود مقدار حداقل برای SEQUENCE ها درنظر گرفته می شود. به طور پیش فرض اگر از هیچ کدام از این عبارات استفاده نشود NOMINVALUE در نظر گرفته می شود.

- CYCLEیاNOCYCLE : اگر از CYCLE استفاده شود زمانی که یک SEQUENCE به آخرین مقدار خود می رسد می تواند مجدد از اولین عدد شروع کند.

- CACHE nیاNOCHACHE : اگر از CACHE n استفاده شود اوراکل به تعداد n عدد را برای SEQUENCEاز قبل تولید می کند و در حافظه نگه می دارد. در دستور ایجاد SEQUENCE به صورت پیش فرض از CACHE 20 استفاده می شود.

- ORDERیا NOORDER : با عبارت ORDER تضمین شود که همیشه مقدار اعداد تولیدی به ترتیب درخواست ها است. برای زمانی مناسب است اعداد به عنوان TIMESTAMP استفاده می شوند. به صورت پیش فرض NOORDER است.

- بنابراین اگر از دستور زیر استفاده شود یک SECUENCE به نام S1 ساخته می شود که از ۱ شروع شده و هربار یک عدد اضافه می شود. مقدار CACHE هم ۲۰ خواهد بود.

```
CREATE sequence s1;
```

مثال :می خواهیم شماره هر دپارتمان را با استفاده از یک SEQUENCE تولید کنیم. این شماره از ۲۸۰ شروع می شود و هربار ۱۰ واحد اضافه می شود. حداکثر شماره دپارتمان نیز می تواند ۹۹۹۹ باشد. این شماره ها به صورت غیرتکراری (NOCYCLE) تولید می شوند SEQUENCE .مورد نظر را بسازید.

```
CREATE SEQUENCE dept_deptid_seq
                START WITH 280
                INCREMENT BY 10
                MAXVALUE 9999
                NOCACHE
                NOCYCLE;
sequence DEPT_DEPTID_SEQ created.
```

نکته :اگر اعداد تولیدی یک SEQUENCE در یک ستون از جدول ذخیره می شوند و می خواهیم آن ستون PRIMARY KEY باشد باید آن SEQUENCE را به صورت NOCYCLE ایجاد کنیم. مگر اینکه مکانیزمی داشته باشیم که سطرهای با شماره قدیمی را پاک میکند.

۱۲.۱ روش استفاده از SEQUENCE

بعد از ساختن یک SEQUENCE می توانیم از شبه ستون های CURRVAL و NEXTVAL استفاده کنیم تا بتوانیم یک عدد جدید توسط آن SEQUENCE تولید کنیم و در جدول ها ذخیره کنیم CURRVAL .و NEXTVALستون های SEQUENCE نیستند ولی ویژگی های آنها شبیه ستونی از جدول است. به همین دلیل به آنها شبه ستون می گویند.

زمانی که در یک دستور SQL از شبه ستون NEXTVAL استفاده شود یک عدد جدید برای SEQUENCE تولید می شود که این عدد با استفاده از CURRVAL قابل مشاهده است. برای اینکه بتوانیم مقدار عددی SEQUENCEرا با استفاده از CURRVAL مشاهدا کنیم باید در آن SESSION یک عمل NEXTVAL انجام شده باشد وگرنه خطای اوراکل دریافت می کنیم. در ادامه با یک مثال فرایند تولید و نمایش عدد جدید با CRRVAL و NEXTVAL را توضیح می دهیم.

نکته :کاربری که می خواهد برای SEQUENCE کاربر دیگر مقدار تولید کند NEXTVAL) اجرا کند (یا مقدار CURVAL را ببیند باید مجوز SELECT ANY SEQUENCE داشته باشد.

مثال:

ابتدا یک SEQUENCE به نام seq1 برای کاربر milad ایجاد می کنیم.

```
SQL> create sequence seq1;

Sequence created.
```

حال اگر بخواهیم مقدار CURRVAL این SEQUENCE را مشاهده کنیم پیغام خطا دریافت می کنیم زیرا در این SESSION هنوز NEXTVAL اجرا نشده است که مقدار SEQUENCE تولید شود.

```
SQL> select seq1.currval from dual;
select seq1.currval from dual

       *

ERROR at line 1:

ORA-08002

ORA-08002: sequence SEQ1.CURRVAL is not yet defined in this session
```

بنابراین ابتدا مقدار NEXTVAL را تولید می کنیم و سپس CURRVAL را مشاهده می کنیم.

```
SQL> select seq1.nextval from dual;
  NEXTVAL

———-

       1

SQL> select seq1.currval from dual;
  CURRVAL

———-

       1
```

در ادامه با همان کاربر milad و در یک SESSION جدید مقدار CURRVAL را درخواست می کنیم ولی از آنجایی که در این SESSION هنوز NEXTVAL انجام نشده است خطا دریافت می کنیم. بنابراین ابتدا یک مقدار NEXTVAL تولید می کنیم.

```
SQL> select seq1.currval from dual;
```

```
select seq1.currval from dual
*

ERROR at line 1:

ORA-08002: sequence SEQ1.CURRVAL is not yet defined in this session

SQL> select seq1.nextval from dual;
  NEXTVAL

———-

         2

SQL> select seq1.currval from dual;
  CURRVAL

———-

         2
```

حال با کاربر ahmad به دیتابیس متصل می شویم و مقدار currval برای SEQUENCE متعلق به کاربر miladرا درخواست می کنیم. ولی خطا دریافت می کنیم. بنابراین ابتدا از nextval استفاده می کنیم.

```
SQL> select milad.seq1.currval from dual;
select milad.seq1.currval from dual

*

ERROR at line 1:

ORA-08002: sequence milad.SEQ1.CURRVAL is not yet defined in this session

SQL> select milad.seq1.nextval from dual;
  NEXTVAL

———-

         3

QL> select milad.seq1.currval from dual;
  CURRVAL

———-

         3
```

نکته :در بخش های زیر از یک دستور SQL می توان از NEXTVAL و CURRVAL استفاده نمود:

- در لیست SELECT یک دستور SELECT که خودش یک SUBQUERY نیست.
- در لیست SELECT یک SUBQUERY به شرطی که بخشی ازیک دستور INSERT باشد.
- در قسمت VALUES یک دستور.INSERT
- در قسمت SET یک دستور.UPDATE

نکته :در بخش های زیر از یک دستور SQL نمی توان از NEXTVAL و CURRVAL استفاده نمود:

- در لیست یک دستور SELECT که در داخل یک VIEW استفاده شده باشد.
- در داخل یک دستور SELECT که از کلمه کلیدی DISTINCT استفاده کرده است.
- در داخل یک دستور SELECT که از عبارت هایGROUP BY ، HAVINGو ORDER BY استفاده می کند.
- هر SUBQUERY که داخل یک دستورDELETE ، UPDATEو یا SELECT اجرا شود.

مثال :یک دپارتمان جدید به جدول DEPARTMENTS اضافه کنید که شماره دپارتمان آن توسط یک SEQUENCE به نام dept_deptid_seq ایجاد می شود.

```
INSERT INTO departments(department_id,
            department_name, location_id)
VALUES      (dept_deptid_seq.NEXTVAL,
            'Support', 2500);
  1 rows inserted
```

مثال :مقدار فعلی SEQUENCE مثال قبل را نمایش دهید.

```
SELECT   dept_deptid_seq.CURRVAL
FROM     dual;
```

مثال :در ادامه مثال قبل یک کارمند جدیدالورود به جدول EMPLOYEES اضافه کنید. این کارمند در دپارتمان جدید مشغول به کار می شود(برای تولید شماره پرسنلی این کارمند جدیدالورورد از یک SEQUENCE به نام employees_seq استفاده شود).

```
INSERT INTO employees (employee_id, department_id,
...)
VALUES (employees_seq.NEXTVAL,
dept_deptid_seq.CURRVAL, ...);
```

۱۲.۲ استفاده از SEQUENCE به عنوان مقدار DEFAULT

همانطور که می دانید زمانی که در دستور INSERT هیچ مقداری برای یک ستون خاص تعیین نشود، اگر برای آن ستون یک مقدار DEFAULT تعریف شده باشد آن مقدار درج خواهد شد. در اوراکل نگارش 12cو بالاتر می توان از مقدارهای NEXVAL و CURRVAL مربوط به یک SEQUENCE به عنوان یک مقدار DEFAULTبرای ستون های جدول استفاده نمود. البته به شرطی که یک مقدار برای آن SEQUENCE تولید شده باشد و مجوز لازم برای دسترسی به SEQUENCE را داشته باشیم.

نکته :اگر یک SEQUENCE که به عنوان مقدار DEFAULT برای یک ستون بکار رفته است DROP شود در زمان درج دستورات DML در جدول هایی که به مقدار DEFAULT از آن SEQUENCE احتیاج دارند خطای اوراکل دریافت می کنیم و آن عمل درج انجام نخواهد شد.

مثال :ستون a1 از جدول emp مقدار DEFAULT خودش را از NEXTVAL مربوط به یک SEQUENCEبه نام s1 دریافت کند.

```
CREATE SEQUENCE s1 START WITH 1;
CREATE TABLE emp (a1 NUMBER DEFAULT s1.NEXTVAL NOT
NULL, a2 VARCHAR2(10));
INSERT INTO emp (a2) VALUES ('john');
INSERT INTO emp (a2) VALUES ('mark');
SELECT * FROM emp;
```

```
sequence S1 created.
table EMP created.
1 rows inserted.
1 rows inserted.
A1 A2
-- ----------
 1 john
 2 mark
```

نکته :زمانی که مقدارهای SEQUENCE در حافظه CACHE می شوند سرعت دسترسی افزایش می یابد.

نکته :در یکی از شرایط زیر ممکن است فاصله یا GAP بین اعداد تولید شده از یک SEQUENCE رخ دهد.

- دستوری که برای آن یک SEQUNCE تولید شده است ROLLBACK شود.

- سیستم CRASH کند و عملیات ROLLBACK و RECOVERY انجام شود.
- یک SEQUENCE توسط جدول یا OBJECT دیگر مورد استفاده قرار گرفته باشد.

۱۲.۳ تغییر دادن مقدار یک SEQUENCE

فرض کنید اعداد تولید شده توسط یک SEQUENCE به مقدار تعریف شده برای MAXVALUE رسیده است و SEQUENCE نیز از نوع NOCYCLE است. بنابراین در زمان تولید مقدار جدید برای آن SEQUENCE، خطای اوراکل دریافت می کنیم. می توان با استفاده از دستور ALTER مقدار MAXVALUE برای آن SEQUENCE را تغییر داد.

```
ALTER SEQUENCE dept_deptid_seq
               INCREMENT BY 20
               MAXVALUE 999999
               NOCACHE
               NOCYCLE;
```

```
sequence DEPT_DEPTID_SEQ altered.
```

نکته: برای ALTER کردن یک SEQUENCE باید مجوز لازم را داشته باشیم و یا مالک آن SEQUENCE باشیم.

نکته: بعد از هرگونه ALTER کردن SEQUENCE فقط می توان روند اعداد بعدی که می خواهند تولید شوند را تغییر داد و اعداد قبلی به همان روال خواهند ماند.

نکته: اگر می خواهیم یک SEQUENCE از ابتدا، اعداد جدید تولید کند باید آن SEQUENCE را DROP کنیم و دوباره با شروط جدید ایجاد کنیم. با دستور ALTER مقدار عبارت START WITH قابل تغییر نیست.

مثال SEQUENCE :به نام dept_deptid_seq را DROP کنید.

```
DROP SEQUENCE dept_deptid_seq;
```

```
sequence DEPT_DEPTID_SEQ dropped.
```

نکته :در زمان ALTER کردن یک SEQUENCE عملیات صحت سنجی آن دستور انجام می شود. برای مثال نمی توان مقدار MAXVALUE را به مقداری که SEQUENCE آن را تولید کرده و از آن رد شده است تغییر داد و برای اجرای این دستور خطای اوراکل دریافت می کنیم.

مثال :در دستور ALTER مقدار MAXVALUE از مقدار فعلی SEQUENCE کمتر است(SEQUENCEنیز از نوع صعودی است)بنابراین خطای اوراکل دریافت می کنیم.

```
ALTER SEQUENCE dept_deptid_seq
      INCREMENT BY 20
      MAXVALUE 90
      NOCACHE
      NOCYCLE;
```

- The error:

```
SQL Error: ORA-04009: MAXVALUE cannot be made to be less than the current value
04009. 00000 - "MAXVALUE cannot be made to be less than the current value"
*Cause:    the current value exceeds the given MAXVALUE
*Action:   make sure that the new MAXVALUE is larger than the current value
```

۱۲.۴ مشاهده اطلاعات SEQUENCE در دیتا دیکشنری

اطلاعات هر SEQUENCE که ایجاد شده است در دو VIEW از دیتا دیکشنری قرار می گیرد.

- DBA_OBJECTS
- DBA_SEQUENCES

اطلاعات کامل هر SEQUENCE در DBA_SEQUENCES قرار دارد که ستون های این VIEW را در شکل زیر مشاهده می کنید.

```
DESCRIBE user_sequences
```

```
DESCRIBE user_sequences
Name            Null     Type
--------------- -------- ------------
SEQUENCE_NAME   NOT NULL VARCHAR2(128)
MIN_VALUE                NUMBER
MAX_VALUE                NUMBER
INCREMENT_BY    NOT NULL NUMBER
CYCLE_FLAG               VARCHAR2(1)
ORDER_FLAG               VARCHAR2(1)
CACHE_SIZE      NOT NULL NUMBER
LAST_NUMBER     NOT NULL NUMBER
PARTITION_COUNT          NUMBER
SESSION_FLAG             VARCHAR2(1)
KEEP_VALUE               VARCHAR2(1)
```

نکته : ستون LAST_NUMBER در DBA_SEQUENCES یکی بیشتر از آخرین شماره تولید شده توسط هر SEQUENCE است. یعنی عددی که با یک NEXTVAL تولید خواهد شد. البته اگر یک SEQUENCE از CACHING استفاده کند عدد مربوط به CACHING به این عدد اضافه می شود.

مثال :اطلاعات SEQUENCE ها را نمایش دهید.

```
SELECT   sequence_name, min_value, max_value,
         increment_by, last_number
FROM     user_sequences;
```

۱۳ Synonym در دیتابیس اوراکل

یکی ازOBJECT های دیتابیس اوراکل SYNONYM است که به عنوان یک نام جایگزین برای OBJECTهای دیگر دیتابیس (جدول ها ، VIEWو...)عمل می کند. در این فصل SYNONYM و روش استفاده از آن را توضیح می دهیم.

۱۳.۱ استفاده از synonym

همانطور که گفته شد SYNONYM مانند ALIAS عمل می کند و سبب راحتی در نوشتن دستورات SQL می شود.همچنین استفاده از SYNONYM می تواند از لحاظ امنیتی هم سودمند باشد زیرا می تواند نام و محل اصلی(یک OBJECTدر کدام SCHEMA قرار دارد) را مخفی کند.

برایOBJECT های زیر می توان SYNONYM تعریف کرد:

- **Table**
- **View**
- **sequence**
- **stored procedure – function – package**
- **materialized view**
- **java class schema object**
- **user-defined object**
- **synonym**

نکته: SYNONYM نیاز به فضای ذخیره سازی ندارد و فقط تعاریف مربوط به آنها در دیتادیکشنری دیتابیس اوراکل ثبت می شود.

۱۳.۲ ایجاد و حذف SYNONYM

در شکل زیر سینتکس ایجاد یک SYNONYM را می بینید

```
CREATE [PUBLIC] SYNONYM synonym
FOR    object;
```

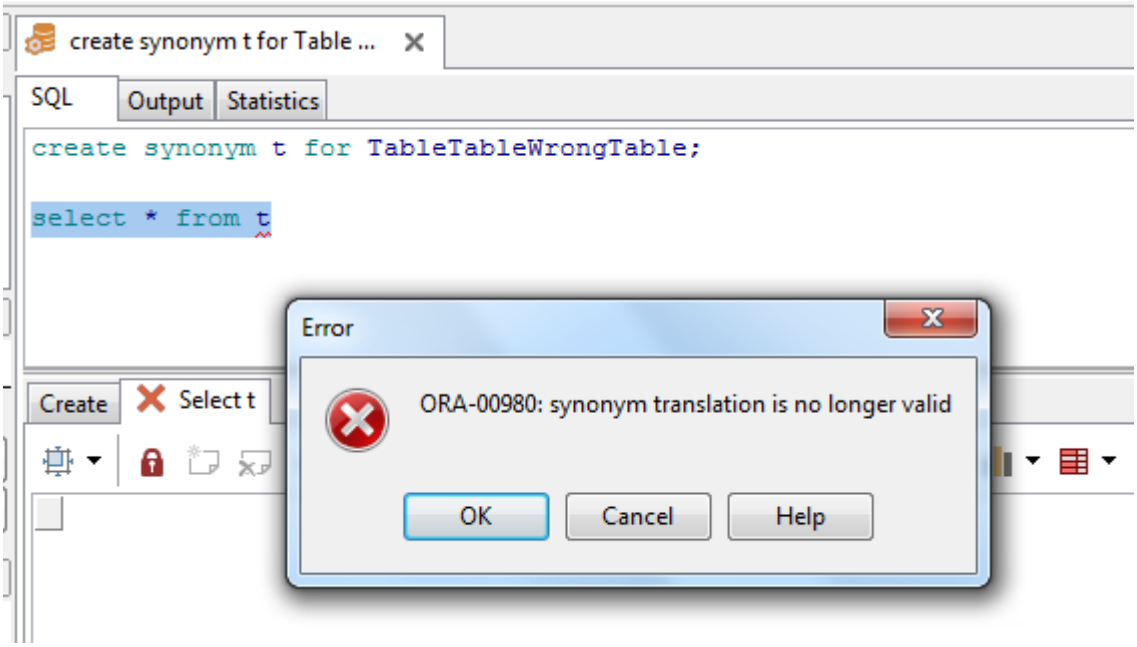
در دستور ایجاد یک SYNONYM اگر از کلمه PUBLIC استفاده شود آن SYNONYM می تواند توسط تمام کاربران دیتابیس مورد دسترسی قرار گیرد. البته به شرطی که یک کاربر مجوز لازم برای OBJECT آن SYNONYMرا داشته باشد.

برای ساختن SYNONYM نیاز به مجوز CREATE SYNONYYM داریم.

```
SQL> grant create synonym to milad;

Grant Succeeded.
```

نکته: ممکن است یک SYNONYM برای یک OBJECT که وجود ندارد ایجاد شود یا آن OBJECT حذف شده یا تغییر نام داده باشد در این صورت در زمان استفاده از SYNONYM با خطای اوراکل مواجه می شویم.

مثال: جدول TableTableWrongTable وجود ندارد ولی یک SYNONYM برای آن ایجاد می شود. در زمان دسترسی به آن SYNONYM متوجه خطای اوراکل می شویم.



نکته: برای ساخت SYNONYM از نوع PUBLIC نیاز به مجوز CREATE PUBLIC SYNONYM داریم.

نکته: زمانی که یک SYNONYM که PUBLIC نیست می سازیم (PRIVATE است)باید نام آن متفاوت از نام تمامOBJECT های آن کاربر باشد.

نکته: برای یک OBJECT که داخل یک PACKAGE قرار دارد نمی توان از SYNONYM استفاده کرد.

مثال: برای VIEW با نام DEPT_SUM_VU یک SYNONYM بسازید.

```
CREATE SYNONYM  d_sum
FOR  dept_sum_vu;
  synonym D_SUM created.
```

مثال: SYNONYM مثال قبل را از دیتابیس حذف کنید.

```
DROP SYNONYM d_sum;
  synonym D_SUM dropped.
```

مثال: یک SYNONYM از نوع PUBLIC به نام DEPT برای جدول DEPARTMENTS که مالک آن کاربر ALICE است بسازید. مجوز لازم برای ساخت PUBLIC SYNONYM را داریم.

```
CREATE PUBLIC SYNONYM dept FOR alice.departments;
```

مثال: SYNONYM مثال قبل را حذف کنید.

```
DROP PUBLIC SYNONYM dept;
```

مثال:

مجوز لازم جهت ساختن و DROP کردن PUBLIC SYNONYM را به کاربر ALI می دهیم.

SQL> grant create public synonym to ali;

Grant Succeeded.

SQL> grant drop public synonym to ali;

Grant Succeeded.

توسط کاربر ALI یک PUBLIC SYNONYM به نام t برای جدول ali.table1 می سازیم و سپس آن را DROPمی کنیم.

```
CREATE PUBLIC SYNONYM t FOR ali.table1;
DROP PUBLIC SYNONYM t;
```

دوباره آن SYNONYM را ایجاد می کنیم. اگر دیتادیکشنری USER_SYNONYMS را در کاربر ALI نمایش دهیم اطلاعات این SYNONYM که از نوع PUBLIC است را نمی توانیم ببینیم.

در ادامه مجوز لازم برای SELECT این PUBLIC SYNONYM را به کاربر MILAD می دهیم. یکی از این دو مجوز کافیست.

SQL> grant select on t to milad;

Grant Succeeded.

یا

SQL> grant select on ali.table1 to milad;

Grant Succeeded.

در نهایت با کاربر milad متصل می شویم و مقدارهای این SYNONYM را مشاهده می کنیم.

```
Select * from t;
```

نکته: اگر یک OBJECT دارای DATABASE LINK است از سینتکس زیر می توان استفاده کرد.

```
CREATE [PUBLIC] SYNONYM [schema .] synonym_name
 FOR [schema .] object_name [@ dblink];
```

نکته: برای تغییر یک SYNONYM می توان از سینتکس زیر استفاده کرد.

```
CREATE OR REPLACE [PUBLIC] SYNONYM [schema .] synonym_name
FOR [schema .] object_name [@ dblink];
```

مثال: جدول SYNONYM را عوض کنید.

```
CREATE OR REPLACE PUBLIC SYNONYM t FOR ali.table22222;
```

۱۳.۳ اطلاعاتی در مورد SYNONYM های تعریف شده

مشخصات تمام SYNONYM های تعریف شده در دیتابیس اوراکل را می توانیم در ویوی دیتادیکشنری DBA_SYNONYMS ببینیم. اگر مجوز کافی نداریم با ویوی USER_SYNONYMS فقط آنهایی که مربوط به کاربر خودمان هستند را می بینیم.

```
DESCRIBE user_synonyms
```

```
DESCRIBE user_synonyms
Name         Null     Type
------------ -------- -------------
SYNONYM_NAME NOT NULL VARCHAR2(128)
TABLE_OWNER           VARCHAR2(128)
TABLE_NAME   NOT NULL VARCHAR2(128)
DB_LINK               VARCHAR2(128)
```

مثال: مشخصات تمام SYNONYMهای کاربر جاری را نمایش دهید.

```
SELECT *
FROM   user_synonyms;
```

	SYNONYM_NAME	TABLE_OWNER	TABLE_NAME	DB_LINK
1	D_SUM	ORA21	DEPT_SUM_VU	(null)

۱۴ VIEW در دیتابیس اوراکل

VIEW یکی از OBJECT های دیتابیس اوراکل است که در این فصل روش ساخت و استفاده از آن را توضیح می دهیم.

۱۴.۱ VIEW چیست؟

هر VIEW یک دستور SELECT ذخیره شده در دیتابیس است که اطلاعات یک یا چند جدول را جهت نمایش یا ویرایش در کنار هم قرار می دهد. برای مثال می توان بخشی از اطلاعات دو جدول را با یک دستور SELECT، باهم JOIN کرد و این دستور را در قالب یک VIEW ذخیره کرد. بنابراین یک کاربر دیتابیس با SELECT کردن این VIEW در واقع یک دستور JOIN دو جدول را اجرا خواهد کرد بدون اینکه از نام اصلی جدول ها، ستون ها و روش استفاده از دستور JOIN مطلع باشد.

بعد از اینکه یک VIEW ساخته شد می توانیم در محدوده اطلاعات آن VIEW از دستورات SELECT یا DML استفاده کنیم. برای هر VIEW (بر خلاف MATERILIZED VIEW) فقط دستور SELECT داخلی آن در دیتادیکشنری ذخیره می شود و احتیاج به هیچ فضای ذخیره سازی دیگری ندارد. به جدول یا جدول هایی که در دستور داخلی VIEW استفاده شده اند BASE TABLE می گویند.

نکته: وقتی یک VIEW را برای نمایش یا ویرایش انتخاب می کنیم مراحل زیر اتجام می شود:

۱. دستور داخلی آن VIEW از جدول دیتادیکشنری استخراج می شود.
۲. مجوزهای لازم بررسی می شوند.
۳. .عملیات دستور داخلی VIEW برای دستیابی به اطلاعات جدول ها انجام می شود.

۱۴.۲ مزایای استفاده از VIEW

- محدود سازی دسترسی به داده ها و جدول ها :نام اصلی جداول و ستون ها را مخفی می کنیم.
- آسان نمودن استفاده از QUERY های سنگین :برای مثال یک QUERY طولانی و پیچیده برای نمایش فضای استفاده شده ازTABLESPACE های دیتابیس داریم که آن را در قالب یک VIEW ذخیره می کنیم و هربار با یک SELECT ساده از آن VIEW آن QUERY اجرا خواهد شد.
- ایجاد نمایش های مختلف برای مجموعه ای از داده ها
- مناسب برای استفاده در APPLICATION ها

۱۴.۳ انواع VIEW

VIEWها به دو دسته تقسیم بندی می شوند:

۱. SIMPLE VIEW
- اطلاعات از یک جدول بدست می آید
- در دستور SELECT داخلی از توابع یا گروه بندی اطلاعات استفاده نمی شود.
- معمولا بعد از ایجاد VIEW علاوه بر SELECT از عملیات DML در محدوده اطلاعات آن VIEW استفاده می شود.

۲. COMPLEX VIEW
- اطلاعات VIEW معمولا از چند جدول استخراج می شود.
- در دستور داخلی VIEW از توابع یا روش های گروه بندی اطلاعات استفاده می شود.
- معمولا از عملیات DML برای این دسته ازVIEW ها استفاده نمی شود.

۱۴.۴ روش ساخت VIEW

سینتکس ساخت یک VIEW را در شکل زیر می بینید. هر قسمت از این سینتکس را در ادامه توضیح می دهیم.

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
  [(alias[, alias]...)]
 AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

- عبارت OR REPLACE : زمانی که یک VIEW با یک نام خاص ایجاد شده است می توان با این عبارت دستور داخلی آن VIEW را تغییر داد و ذخیره کرد. بنابراین اگر می خواهیم یک VIEW را تغییر دهیم نیاز نیست آن را DROP کنیم، مجددا تعریف کنیم و دوباره مجوزهای لازم را به آن بدهیم.
- FORCE : حتی اگر جدول های مورد استفاده در دستور داخلی وجود ندارد آن VIEW تعریف خواهد شد

- NO FORCE: در حالت پیش فرض این مورد استفاده می شود. یعنی اگر جدولی که در دیتابیس وجود ندارد در دستور داخلی استفاده شده باشد آن VIEW ایجاد نخواهد شد

- ALIAS: می توان لیست ALIAS های مورد استفاده شده در دستور داخلی را در این قسمت تعریف کرد. البته تعداد و ترتیب این ALIAS ها باید برابر با تعداد و ترتیب ستون ها و عبارت استفاده شده در دستور داخلی باشد.

- SUBQUERY :همان دستور SELECT داخلی VIEW است.

- WITH CHECK OPTION : یک نوع CONSTRAINT است که تضمین می کند عملیات DML فقط برای سطرهایی که در حوزه دسترسی آن VIEW قرار دارد امکان پذیر است. اگر هیچ نامی برای این CONSTRAINT انتخاب نشود از نام های پیش فرض دیتابیس (SYS_Cn) استفاده می شود.

- WITH READ ONLY : اگر از این عبارت استفاده شود نمی توان هیچگونه عملیات DML آن VIEWرا اجرا کرد. یعنی فقط می توان آن VIEW را SELECT کرد.

مثال: یک VIEW ایجاد کنید که شماره پرسنلی، نام و حقوق تمام پرسنل دپارتمان ۸۰ را انتخاب کند. سپس ساختار آن VIEW را نمایش دهید.

```
CREATE VIEW   empvu80
 AS SELECT  employee_id, last_name, salary
    FROM    employees
    WHERE   department_id = 80;
view EMPVU80 created.
```

```
DESCRIBE empvu80
Name        Null     Type
----------- -------- ------------
EMPLOYEE_ID NOT NULL NUMBER(6)
LAST_NAME   NOT NULL VARCHAR2(25)
SALARY               NUMBER(8,2)
```

مثال: یک دستور بنویسید که شماره پرسنلی، نام و حقوق تمام پرسنل دپارتمان ۸۰ را نمایش دهد.

```
SELECT * FROM empvu80;
```

نکته: همانند یک جدول می توان فقط بخشی از اطلاعات که در محدوده اطلاعات یک VIEW قرار دارد را انتخاب کرد.

مثال: یک دستور بنویسید که شماره پرسنلی، نام و حقوق پرسنلی که در دپارتمان ۸۰ حقوق بالای ۴۰۰۰ می گیرند را نمایش دهد.

```
SELECT * FROM empvu80 where salary > 4000;
```

مثال: یک VIEW تعریف کنید که حقوق سالیانه نام و شماره پرسنلی کارمندان دپارتمان شماره ۵۰ را انتخاب کند.

```
CREATE VIEW   salvu50
 AS SELECT   employee_id ID_NUMBER, last_name NAME,
             salary*12 ANN_SALARY
    FROM     employees
    WHERE    department_id = 50;
 view SALVU50 created.
```

مثال: VIEW مثال قبل را تغییر دهید به نحوی که ALIAS ها در لیست دستور ساخت قرار داشته باشند.

```
CREATE OR REPLACE VIEW salvu50 (ID_NUMBER, NAME, ANN_SALARY)
AS SELECT employee_id, last_name, salary*12 FROM employees
WHERE department_id = 50;
```

مثال: یک VIEW به نام EMPVU80 ایجاد کنید. برای چهار ستون دستور داخلی ALIAS تعریف کنید.

```
CREATE OR REPLACE VIEW empvu80
  (id_number, name, sal, department_id)
AS SELECT  employee_id, first_name || ' '
           || last_name, salary, department_id
   FROM    employees
   WHERE   department_id = 80;
view EMPVU80 created.
```

نکته: اگر در دستور داخلی یک VIEW برای یک ستون از یک تابع یا یک EXPRESSION استفاده شود باید یک نام مستعار برای آن ستون استفاده شود وگرنه در زمان ساخت VIEW خطای اوراکل دریافت می کنیم.

مثال: ایجاد یک VIEW COMPLEX که در آن از توابع و عمل JOIN استفاده شده است.

```
CREATE OR REPLACE VIEW dept_sum_vu
  (name, minsal, maxsal, avgsal)
AS SELECT   d.department_name, MIN(e.salary),
           MAX(e.salary),AVG(e.salary)
   FROM     employees e JOIN departments d
   USING    (department_id)
   GROUP BY d.department_name;
view DEPT_SUM_VU created.
```

۱۴.۵ نمایش اطلاعات VIEW ها در دیتابیس اوراکل

برای نمایش اطلاعاتVIEW های موجود در دیتابیس می توان از دیتادیکشنری DBA_VIEWS یا USER_VIEWSاستفاده کرد. برای یک VIEW دستور داخلی آن در ستون TEXT ذخیره می شود و ستون TEXT_LENGTH تعداد کاراکترهای آن دستور را نشان می دهد.

1
```
DESCRIBE user_views
```

Name	Null	Type
VIEW_NAME	NOT NULL	VARCHAR2(30)
TEXT_LENGTH		NUMBER
TEXT		LONG()

...

2
```
SELECT view_name FROM user_views;
```

	VIEW_NAME
1	EMP_DETAILS_VIEW
2	SALVU50
3	EMPVU80
4	DEPT_SUM_VU

3
```
SELECT text FROM user_views
WHERE view_name = 'EMP_DETAILS_VIEW';
```

	TEXT
1	SELECT e.employee_id, e.job_id, e.manager_id, e.department_id, d.location_id, l.co

۱۴.۶ تغییر داده های یک VIEW با عملیات DML

همانطور که گفته شد برای یک VIEW می توان علاوه بر دستور SELECT از دستورات DML نیز استفاده کرد.

نکته: نمی توان یک سطر از یک VIEW را حذف کرد (DELETE) اگر در دستور داخلیVIEW

۱. یک عبارت GROUP BY استفاده شده باشد.
۲. از کلمه کلیدی DISTINCT استفاده شده باشد.
۳. از تابع گروهی استفاده شود.
۴. از کلمه کلیدی ROWNUM (شبه ستون ROWNUM)استفاده شود.

نکته: نمی توان داده های یک VIEW را تغییر داد (UPDATE) اگر در دستور داخلیVIEW

۱. یک عبارت GROUP BY استفاده شده باشد.
۲. از کلمه کلیدی DISTINCT استفاده شده باشد.
۳. از تابع گروهی استفاده شود.
۴. از کلمه کلیدی ROWNUM (شبه ستون ROWNUM)استفاده شود.
۵. از یک EXPRESSION برای یک ستون استفاده شده باشد

نکته: نمی توان به جدول های یک VIEW سطر جدید اضافه کرد (INSERT) اگر در دستور داخلیVIEW

۱. یک عبارت GROUP BY استفاده شده باشد.

۲. از کلمه کلیدی DISTINCT استفاده شده باشد.
۳. از تابع گروهی استفاده شود.
۴. از کلمه کلیدی ROWNUM (شبه ستون ROWNUM)استفاده شود.
۵. از یک EXPRESSION برای یک ستون استفاده شده باشد
۶. ستون هایی در جدول BASE TABLE داشته باشیم که دارای CONSTRAINT از نوع NOT NULLباشند ولی در دستور داخلی VIEW استفاده نشده اند. در ضمن این ستون ها مقدار DEFAULTندارند.

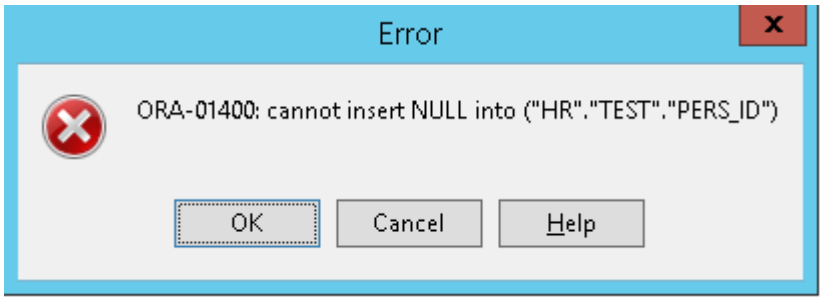
مثال: ابتدا یک جدول با دو ستون می سازیم. سپس VIEW را با ستونی که از نوع NOT NULL نیست می سازیم

```
create table test(name varchar2(10),pers_id integer not null)
create or replace view testVU as select name from test;
```

برای VIEW از دستور INSERT استفاده می کنیم.

```
insert into testVU values('milad')
```

خطای اوراکل دریافت می کنیم.



برای ستون جدول که NOT NULL است یک مقدار DEFAULT تعریف می کنیم.

```
alter table test modify (pers_id integer default 1)
```

دستور INSERT را دوباره اجرا می کنیم. مقدار مورد نظر در جدول اصلی درج شده است.

```
insert into testVU values('milad')
select * from testVU
```

	NAME
1	milad

نکته: اگر در زمان ساخت یک VIEW از عبارت WITH CHECK OPTION استفاده شود تضمین می شود که فقط در حوزه اطلاعات آن VIEW دستورات DML اجرا خواهد شد. یعنی دستورات INSERT و UPDATEفقط روی سطرهایی که آن VIEW می تواند انتخاب کند انجام می شود و هر سطری که توسط دستور داخلی SELECT انتخاب شود را می توان اضافه کرد ولی نمی توان حذف کرد.

مثال: فقط سطر جدیدی که شماره دپارتمان آن برابر با ۲۰ است به VIEW اضافه خواهد شد و هیچ مقدار در ستون شماره دپارتمان تغییر نخواهد کرد. چون اگر ۲۰ ها تغییر کنند دیگر توسط VIEWانتخاب نمی شوند و غیر ۲۰ ها هم در حوزه آن VIEW نیستند.

```
CREATE OR REPLACE VIEW empvu20
AS SELECT     *
   FROM       employees
   WHERE      department_id = 20
   WITH CHECK OPTION CONSTRAINT empvu20_ck ;
view EMPVU20 created.
```

مثال: در VIEW مثال قبل کارمند با شماره پرسنلی ۲۰۱ در دپارتمان ۲۰ مشغول به کار است بنابراین نمی توان مقدار شماره دپارتمان آن را تغییر داد.

```
UPDATE empvu20
SET    department_id = 10
WHERE  employee_id = 201;
```

Error:

```
SQL Error: ORA-01402: view WITH CHECK OPTION where-clause violation
01402. 00000 - "view WITH CHECK OPTION where-clause violation"
*Cause:
*Action:
```

نکته: اگر در تعریف یک VIEW از عبارت WITH READ ONLY استفاده شود نمی توان هیچگونه عملیات DML روی آن VIEW انجام داد.

مثال: یک VIIEW تعریف کنید که عملیات DML را نپذیرد. یک عمل DML انجام دهید.

```
CREATE OR REPLACE VIEW empvu10
    (employee_number, employee_name, job_title)
AS SELECT    employee_id, last_name, job_id
   FROM     employees
   WHERE    department_id = 10
   WITH READ ONLY ;
   view EMPVU10 created.
```

```
DELETE FROM empvu10
WHERE  employee_number = 200;
```

```
Error report:
SQL Error: ORA-42399: cannot perform a DML operation on a read-only view
```

نکته: اگر یک VIEW حذف (DROP)شود جدول های آن VIEW حذف نمی شوند. اگر هر کدام از BASE TABLEها حذف شوند یا یک ستون که در VIEW استفاده شده است حذف گردد آن VIEW به صورت INVALID می شود و در زمان اجرا خطای اوراکل HAS ERRORS دریافت می کنیم.

مثال: ابتدا جدول را ایجاد کرده و یک VIEW از یکی از ستون های آن ایجاد می کنیم.

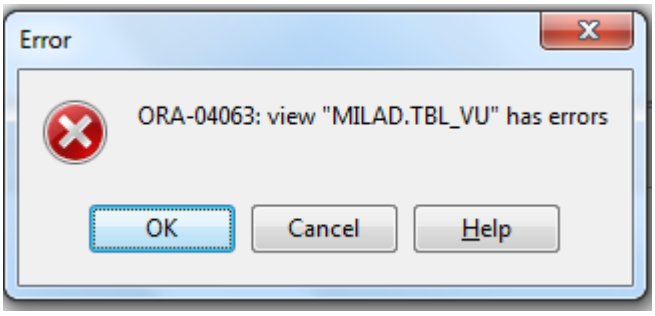
```
create table milad.mytable(name varchar2(10),pers_id integer);

create or replace view milad.tbl_VU as select pers_id from
milad.mytable;
```

ستونی که از روی آن VIEW ساختیم را حذف می کنیم.

```
alter table milad.mytable drop(pers_id);
```

آن VIEW انتخاب می کنیم و خطای HAS ERRORS دیافت می کنیم.

```
select * from milad.tbl_VU;
```



۱۵ NDEX در دیتابیس اوراکل

۱۵.۱ INDEX چیست؟

INDEX یکی از OBJECTهای دیتابیس اوراکل است که می توان به منظور افزایش سرعت اجرای دستورات از آن استفاده نمود. ایندکسها روی ستون های جدول تعریف می شوند و معمولا آدرس فیزیکی رکورد(rowid) را هم در خود ذخیره می کنند. زمانی که در دستورات SELECT و یا حتی در مواردی دستورات DMLای بر روی یک ستون از جدول برای دسترسی به سطر های خاص جستجو می شود می توان با استفاده از rowid سرعت دستیابی به آن سطر را افزایش می یابد.

اگر در دستورات SQL برای یک ستون خاص از INDEX استفاده نشده باشد برای هربار دستیابی به داده های جدول بر اساس آن ستون، از عمل FULL TABLE SCAN استفاده می شود. یعنی به ترتیب تمام سطرهای جدول از دیسک به حافظه منتقل می شوند و توسط آن دستور SQL بررسی می شوند.

بنابراین استفاده از INDEX سبب دسترسی مستقیم به سطرهای جدول می شود و عملیات I/O را کاهش می دهد.

۱۵.۲ روش ساخت INDEX

INDEXها مستقل از داده های یک جدول هستند. بنابراین می توان در هر زمان آنها را ساخت یا DROP کرد. زمانی که یک جدول را DROP کنیم INDEX های وابسته به آن نیز DROP می شوند.

یک INDEX ممکن است به صورت اتوماتیک توسط دیتابیس اوراکل ساخته شود. همچنین هر کاربر می تواند به صورت صریح برای یک ستون از جدول INDEX بسازد. بنابراین دو دسته INDEX داریم:

- UNIQUE INDEX :زمانی که در یک جدول یک ستون از نوع PRIMARY KEY یا UNIQUE KEYتعریف می شود دیتابیس اوراکل به صورت اتومالتیک یک INDEX برای این ستون ها با نام پیش فرض می سازد. توجه شود که اینINDEX ها را به روش دستی هم می توان ساخت ولی بهتر است اجازه دهیم دیتابیس این INDEX ها را بسازد.
- NONUNIQUE INDEX :این دسته ازINDEX ها توسط کاربران برای ستون های جدول ایجاد می شوند تا سرعت دستورات SQL بهبود یابد.

در شکل زیر سینتکس ایجاد INDEX برای یک یا چند ستون از یک جدول را می بینید.

```
CREATE [UNIQUE]INDEX index
ON table (column[, column]...);
```

در این سینتکس اگر ستون های مورد نظر UNIQUE باشند می توان از کلمه UNIQUE استفاده کرد تا نوع INDEX نیز UNIQUE باشد.

مثال: برای ستون LAST_NAME یک INDEX بسازید.

```
CREATE INDEX  emp_last_name_idx
ON employees(last_name);
```

```
index EMP_LAST_NAME_IDX created.
```

نکته: می توان همزمان با ساخت جدولINDEX ها را نیز ساخت.

مثال: در زمان ساخت جدول یک INDEX برای ستون EMPLOYEE_ID بسازید.

```
CREATE TABLE NEW_EMP
(employee_id NUMBER(6)
             PRIMARY KEY USING INDEX
             (CREATE INDEX emp_id_idx ON
             NEW_EMP(employee_id)),
first_name   VARCHAR2(20),
last_name    VARCHAR2(25));
```

```
table NEW_EMP created.
```

مثال: مشخصات INDEX مثال قبل را از دیتادیکشنری استخراج کنید.

```
SELECT INDEX_NAME, TABLE_NAME
FROM   USER_INDEXES
WHERE  TABLE_NAME = 'NEW_EMP';
```

	INDEX_NAME	TABLE_NAME
1	EMP_ID_IDX	NEW_EMP

نکته: می توان یک INDEX که UNIQUE نیست برای یک ستون که UNIQUE هست تعریف کرد و از آن استفاده نمود.

مثال: ابتدا یک INDEX برای ستون EMPLOYEE_ID تعریف کنید. سپس این ستون را PRIMARY KEYکنید.

```
CREATE INDEX emp_id_idx2 ON
new_emp2(employee_id);

ALTER TABLE new_emp2 ADD PRIMARY KEY (employee_id)
USING INDEX
emp_id_idx2;
```

۱۵.۳ INDEX های از نوع FUNCTION-BASED

اگر در دستورات SQL برای یک ستون از جدول از توابع SQL استفاده شده است برای سرعت بخشیدن به این دستورات می توان INDEX آنها را از نوع FUNCTION-BASED تعریف کرد. یعنی تابع مورد نظر در INDEX تعریف شود. بنابراین دیتابیس اوراکل زمانی از این INDEX استفاده می کند که در دستورات از این تابع استفاده کرده باشیم.

مثال: یک INDEX تعریف کنید تا در دستور زیر از آن استفاده شود.

```
CREATE INDEX upper_dept_name_idx
ON dept2(UPPER(department_name));
```

```
index UPPER_DEPT_NAME_IDX created.
```

```
SELECT *
FROM   dept2
WHERE  UPPER(department_name) = 'SALES';
```

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	80	Sales	145	2500

۱۵.۴ انواع INDEX ها

در دیتابیس اوراکل می توانINDEX ها را به دو مدل B-TREE و BITMAP ساخت. در حالت پیش فرض یک INDEX با مدل B-TREE ایجاد می شود که مشابه ساختمان داده درختی است. در مقابل BITMAP را داریم که برای ستون هایی که دارای داده های تکراری زیادی هستند مناسب تر است. اگر چه این دو مدل از لحاظ ساختار فیزیکی با هم متفاوت هستند ولی هردو به منظور افزایش سرعت دستورات SQL استفاده

می شوند. برای تعریف یک INDEX با مدل BITMAP از کلمه BITMAP در سینتکس ساخت استفاده می شود.

نکته: می توان برای مجموعه ای از ستون ها(بیش از یک ستون) INDEX ترکیبی تعریف کرد.

نکته: می توان روی مجموعه ای از ستون ها(چند ستون) بیش از یک INDEX تعریف کرد به شرطی که:

- آن INDEX از لحاظ b-TREE یا BITMAP بودن متفاوت باشد.
- از لحاظ UNIQUE یا NON-UNIQUE بودن متفاوت باشد.

البته فقط یکی از این INDEX ها را می توان به عنوان INDEX در دسترس یا VISIBLE تعیین کرد تا توسط دیتابیس استفاده شود.

مثال : دو INDEX ترکیبی روی ستون های FIRST_NAME و EMPLOYEE_ID تعریف کنید و یکی از آنها را VISIBLE کنید.

```
CREATE INDEX emp_id_name_ix1
ON employees(employee_id, first_name);
```

```
index EMP_ID_NAME_IX1 created.
```

```
ALTER INDEX emp_id_name_ix1 INVISIBLE;
```

```
index EMP_ID_NAME_IX1 altered.
```

```
CREATE BITMAP INDEX emp_id_name_ix2
ON employees(employee_id, first_name);
```

```
bitmap index EMP_ID_NAME_IX2 created.
```

۱۵.۵ تغییر دادن یا حذف کردن INDEX

می توان با دستور DROP یک INDEX را حذف نمود.

مثال: INDEX را حذف کنید.

```
DROP INDEX emp_last_name_idx;
```

```
index EMP_LAST_NAME_IDX dropped.
```

نکته: نمی توان با استفاده از دستور ALTER یک INDEX را تغییر دهیم. بلکه برای این منظور باید آن را DROPکرده و از ابتدا بسازیم.

نکته: اگر در زمان ساختن یا DROP کردن یک INDEX از عبارت ONLINE استفاده شود، عملیات DML می توانند همزمان با دستورات ما روی آن جدول اجرا شوند.

نکته: اگر یک جدول DROP شود، INDEXها و CONSTRAINT های آن جدول نیز DROP می شوند.

۱۵.۶ مشاهده اطلاعات INDEX ها در دیتادیکشنری

دو دیتادیکشنری DBA_INDEXES و DBA_IND_COLUMNS اطلاعات اصلی هرکدام ازINDEX های دیتابیس را ذخیره می کنند.

مثال :اطلاعاتINDEX های جدول EMPLOYEES را نمایش دهید.

a
```
SELECT index_name, table_name, uniqueness
FROM   user_indexes
WHERE  table_name = 'EMPLOYEES';
```

	INDEX_NAME	TABLE_NAME	UNIQUENESS
1	EMP_NAME_IX	EMPLOYEES	NONUNIQUE
2	EMP_MANAGER_IX	EMPLOYEES	NONUNIQUE
3	EMP_JOB_IX	EMPLOYEES	NONUNIQUE
4	EMP_DEPARTMENT_IX	EMPLOYEES	NONUNIQUE
5	EMP_EMP_ID_PK	EMPLOYEES	UNIQUE
6	EMP_EMAIL_UK	EMPLOYEES	UNIQUE

مثال: اطلاعات LNAME_IDX را نمایش دهید.

```
DESCRIBE user_ind_columns
```

```
DESCRIBE user_ind_columns
Name             Null Type
---------------- ---- --------------
INDEX_NAME            VARCHAR2(128)
TABLE_NAME            VARCHAR2(128)
COLUMN_NAME           VARCHAR2(4000)
COLUMN_POSITION       NUMBER
COLUMN_LENGTH         NUMBER
CHAR_LENGTH           NUMBER
DESCEND               VARCHAR2(4)
```

```
SELECT index_name, column_name,table_name
FROM   user_ind_columns
WHERE  index_name = 'LNAME_IDX';
```

	INDEX_NAME	COLUMN_NAME	TABLE_NAME
1	LNAME_IDX	LAST_NAME	EMP_TEST

۱۶ GLOBAL TEMPORART TABLE در دیتابیس اوراکل

گاهی اوقات در اجرای دستورات پیچیده SQL نیاز به نگه داری موقت اطلاعات برای محاسبه نتایج نهایی می باشد که می توان در این مواقع از جدول های TEMPORARY در دیتابیس اوراکل کمک گرفت. استفاده از این جدول ها می تواند سبب افزایش سرعت اجرای دستورات SQL شود. این جداول با نام مخفف GTT(GLOBAL TEMPORARY TABLE)شناخته می شوند.

زمانی که یک GTT ساخته می شود اطلاعات ساختاری جدول در دیتابیس به صورت دائمی ذخیره می شود ولی اطلاعات درونی جدول فقط در طول زمان برقراری یک SESSION و یا تا قبل از خاتمه یک تراکنش نگه داری می شود(در حافظه و یاtemporary tablespace ها).

در GTT ساختار جدول در هر زمان توسط تمامSESSION ها قابل دسترسی است ولی هر SESSION فقط رکوردهای متعلق به خودش در این جدول را می بیند.

نکته: اوراکل نگارش 18C، PRIVATE TEMPORARY TABLEرا معرفی کرده است که در این دسته از جدول ها ساختار جدول نیز موقتی است و در انتهای SESSION یا تراکنش DROP می شود. یعنی هر PRIVATE TEMPORARY TABLEفقط برای همان SESSION که آن را ایجاد کرده قابل رویت است.

نکته: از آنجایی که برای محتویات GTT هیچ اطلاعات REDO به صورت مستقیم تولید نمی شود استفاده از GTT برای اطلاعات از نوعtemp ، می تواند سبب بهبود کارایی دیتابیس شود.

سینتکس ساخت GLOBAL TEMPORARY TABLE را در ادامه می بینید.

```
CREATE GLOBAL TEMPORARY TABLE table_name (
    column_definition,
    ...,
    table_constraints
) ON COMMIT [DELETE ROWS | PRESERVE ROWS];
```

این سینتکس مشابه دستور ساخت جدول معمولی است با این تفاوت که عبارت ON COMMIT در انتهای آن استفاده می شود. عبارت ON COMMIT تعیین می کند که اطلاعات جدول از نوع وابسته به SESSIONباشد یا از نوع وابسته به تراکنش.

- ON COMMIT DELETE ROWS : به این معنی می باشد که دیتای جدول از نوع وابسته به تراکنش است. یعنی بعد از هر عملCOMMIT ، دیتابیس اوراکل آن جدول را TRUNCATE می کند(در سطح session تمام رکوردها را پاک می کند)

- ON COMMIT PRESERVE ROWS :در این نوع ازGGT ها زمانی که یک SESSION خاتمه می یابد اطلاعات جدول حذف می شود(زمانی که COMMIT رخ می دهد اطلاعات حذف نمی شوند) بنابراین این نوع جدول وابسته به SESSION است.

نکته: به صورت پیش فرض دیتابیس اوراکل از ON COMMIT DELETE ROWS استفاده می کند.

مثال: یک GTT به نام tbl1 ایجاد کنید.

```
CREATE GLOBAL TEMPORARY TABLE tbl1(
    id INT,
    description VARCHAR2(100)
) ON COMMIT DELETE ROWS;
```

یک عمل INSERT در این جدول انجام دهید و محتویات جدول را نمایش دهید.

```
INSERT INTO tbl1(id,description)
VALUES(1,'Transaction specific global temp table');

select * from tbl1;
```

	ID	DESCRIPTION
1	1	Transaction specific global temp table

عمل COMMIT را انجام دهید و محتویات جدول را نمایش دهید.

```
commit
SELECT id, description
FROM tbl1;
```

	ID	DESCRIPTION

همانطور که مشخص است هیچ سطری در جدول وجود ندارد زیرا این جدول از نوع وابسته به تراکنش بوده است و بعد از عمل COMMIT تمام رکوردها حذف شده اند.

مثال: یک GTT وابسته به SESSION می سازیم و مراحل بالا را تکرار می کنیم.

```
CREATE GLOBAL TEMPORARY TABLE tbl2(
    id INT,
    description VARCHAR2(100)
) ON COMMIT PRESERVE ROWS;
```

```
INSERT INTO tbl2(id,description)
VALUES(1,'Session specific global temp table');

COMMIT;

SELECT id, description
FROM tbl2;
```

	ID	DESCRIPTION
1	1	Session specific global temp table

حال از این SESSION خارج می شویم و با یک SESSION جدید وارد می شویم و مجددا اطلاعات جدول را نمایش می دهیم.

```
SELECT id, description
FROM tbl2;
```

	ID	DESCRIPTION

همانطور که مشخص است در SESSION جدید هیچ دیتایی برای نمایش وجود ندارد.

نکته: اوراکل اجازه می دهد برای ستون های جدول GTT از INDEX استفاده کنیم. اگرچه آن INDEX در محدوده اطلاعات GTT برای آن SESSION ایجاد می شود..

نکته: به صورت پیش فرض دیتابیس اوراکل اطلاعات GTT را در DEFAULT TEMPORARY TABLE ذخیره می کند. البته می توان در دستور ساخت، TABLESPACEمورد نظر را مشخص کنیم.

```
CREATE GLOBAL TEMPORARY TABLE table_name (
    column_definition,
    ...,
    table_constraints
) ON COMMIT [DELETE ROWS | PRESERVE ROWS]
TABLESPACE tablespace_name;
```

نکته: اگر یک یا چند SESSION دیگر در حال استفاده از GTT باشند نمی توان عملیات DDL (بجز TRUNCATE) روی آن GTT انجام داد.

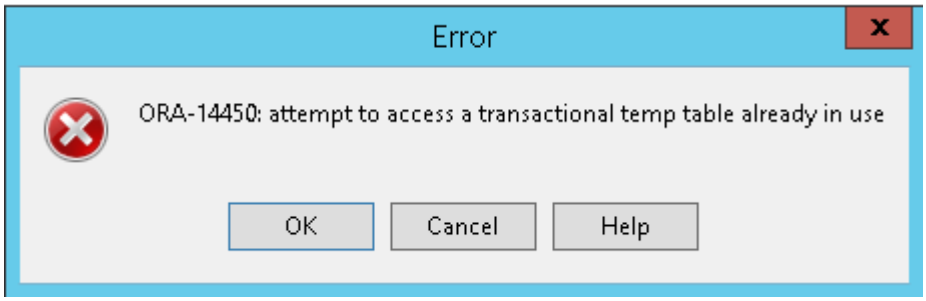
مثال: یک جدول از نوع وابسته به تراکنش بسازید و در آن INSERT کنید.

```
CREATE GLOBAL TEMPORARY TABLE tbl3(
    id INT
```

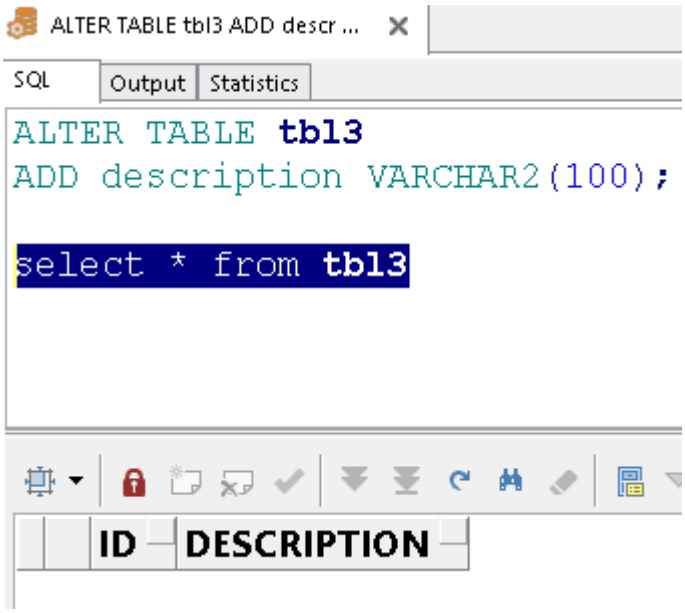
```
) ON COMMIT DELETE ROWS;

INSERT INTO tbl3(id) VALUES(1);
```

حال با یک SESSION دیگر به دیتابیس متصل شوید و دستور ALTER را اجرا کنید.

```
ALTER TABLE tbl3
ADD description VARCHAR2(100);
```



حال در SESSION قبلی دستور COMMIT را اجرا کنید تا اطلاعات آن SESSION در GTT حذف شود و دوباره دستور ALTER را اجرا کنید.



بنابراین دستور ALTER بدون خطا اجرا خواهد شد.

نکته: اگر از دستور ROLLBACK استفاده کنیم تمام اطلاعاتی که تاکنون توسط آن SESSION در یک GTTاز نوع وابسته به تراکنش ثبت شده است از بین می رود.

نکته: نمی توان از جدول های GTT بکاپ تهیه کرد و یا آنها را ریکاور نمود. بنابراین اگر سیستم دچار خرابی شود تمام اطلاعات این نوع جدول ها از بین می روند.

نکته: اگر چه اطلاعات یک GTT در TEMPORARY TABLESPACE ذخیره می شود و به صورت مستقیم از آنها REDO تولید نمی شود ولی همچنان اطلاعات UNDO مربوط به GTT در UNDO TABLESPACEذخیره می شود (تغییر اطلاعات UNDO در فایل های REDO ذخیره می شوند) بنابراین یک GTT از این لحاظ با جدول معمولی تفاوتی ندارد.

نکته: در اوراکل نگارش ۱۲ می توان اطلاعات UNDO یک جدول GTT را در TEMPORARY TABLESPACEذخیره کرد. بنابراین در این حالت با استفاده از GTT ذخیره اطلاعات UNDO و REDO در دیتابیس اوراکل کاهش می یابد. یعنی در اوراکل نگارش ۱۲ می توان عملیات تولید REDO از یک جدول GTT را حذف نمود ولی در نگارش های قبل این عملیات کاهش می یابد اما حذف نمی شوند زیرا همچنان اطلاعات UNDO تولید می شود.

نکته: اگر از دستور TRUNCATE در یک SESSION برای یک GTT استفاده شود فقط اطلاعات مربوط به آن SESSION حذف می شود.

نکته: می توان برای یک GTT یک VIEW ساخت و یا یک VIEW برای ترکیبی از جدولهای معمولی و GTTساخت.

نکته: می توان برای GGT TRIGGER تعریف نمود.

۱۷ زبان های SQL و PL/SQL و تفاوت آنها

۱۷.۱ زبان SQL

SQLمخفف (STRUCTURED QUERY LANGUAGE) یک زبان قدرتمند ولی ساده برای کار با بانک های اطلاعاتی است SQL .در ابتدا توسط شرکت IBM پیاده سازی شده است. در ادامه موسسه جهانی استاندارد، زبان SQL را به عنوان یک زبان رابطه ای برای کار با دیتابیس های از نوع رابطه ای تعیین کرده است. بنابراین زبان SQL به طور کامل مطابق با استانداردهای جهانی است.

این زبان در دیتابیس های رابطه ای داده ها را تغییر می دهد یا جهت نمایش برمی گرداند. یک دیتابیس از نوع رابطه ای، نوعی از دیتابیس است که اطلاعاتی که به هم مرتبط هستند را ذخیره سازی می کند و می توان اطلاعات مختلف را از طریق ارتباط بین آنها مورد دستیابی قرار داد. به عنوان مثال دیتابیس های اوراکل، SQL SERVERو MYSQL از این نوع هستند. بنابراین پیشنهاد می شود به منظور فراگیری بهتر زبان SQL با یکی از دیتابیس های رابطه ای کار کنیم.

نکته: توجه شود که بعضی از دستوراتی که در ابزارهای پشتیبانی دیتابیس اجرا می شوند و جزو استانداردهای زبان SQL نیستند. مانند SHUDOWN یا CONNECT یا STARTUP در SQLPLUS ولی از آنجایی که این ابزارها برای کار با زبان SQL طراحی شده اند به عنوان دستور SQL شناخته می شوند.

۱۷.۲ انواع دستورات SQL

انواع دستورات SQL عبارتند از:

۱. دستورات DML یا :DATA MANIPULATION LANGUAGE با استفاده از دستورات DML می توانیم عملیات حذف کردن اطلاعات(DELETE) ، اضافه کردن (INSERT) و بروزرسانی (UPDATE)را انجام دهیم.
۲. دستورات DDL یا : DATA DEFINITION LANGUAGE با دستوراتDDL ای می توان OBJECTای را ایجاد کرد و ساختار آن را تغییر داد و یا آن شی را حذف نمود. مانند دستورات ALTER، CREATE، DROP، TRUNCATEو.RENAME
۳. دستورات DCL یا DATA CONTROL LANGUAGE که بیشتر مربوط به امنیت داده ها است. مانند GRANT و.REVOKE

۴. DQLیا DATA QUERY LANGUAGE همان دستور پرکاربرد SELECT است که برای دستیابی داده های دیتابیس استفاده می شود.
۵. دستورات کنترل تراکنش مانندROLLBACK ، COMMITو SAVEPOINT

۱۷.۳ PL/SQL چیست؟

PL/SQL یک زبان از نوع رویه ای یا PROCEDURAL است مانند C++ یا JAVA این زبان توسط شرکت اوراکل و به منظور توسعه زبان SQL طراحی و تولید شده است. با استفاده از PL/SQL می توانیم دستورات SQL را به شکل PROCEDURAL ایجاد کرده و اجرا کنیم. در این زبان بلاک های برنامه (مانند PROCEDURE و FUNCTION) تعریف و اجرا می شوند.

در PL/SQL، بلاک ها به دو روش تعریف می شوند:

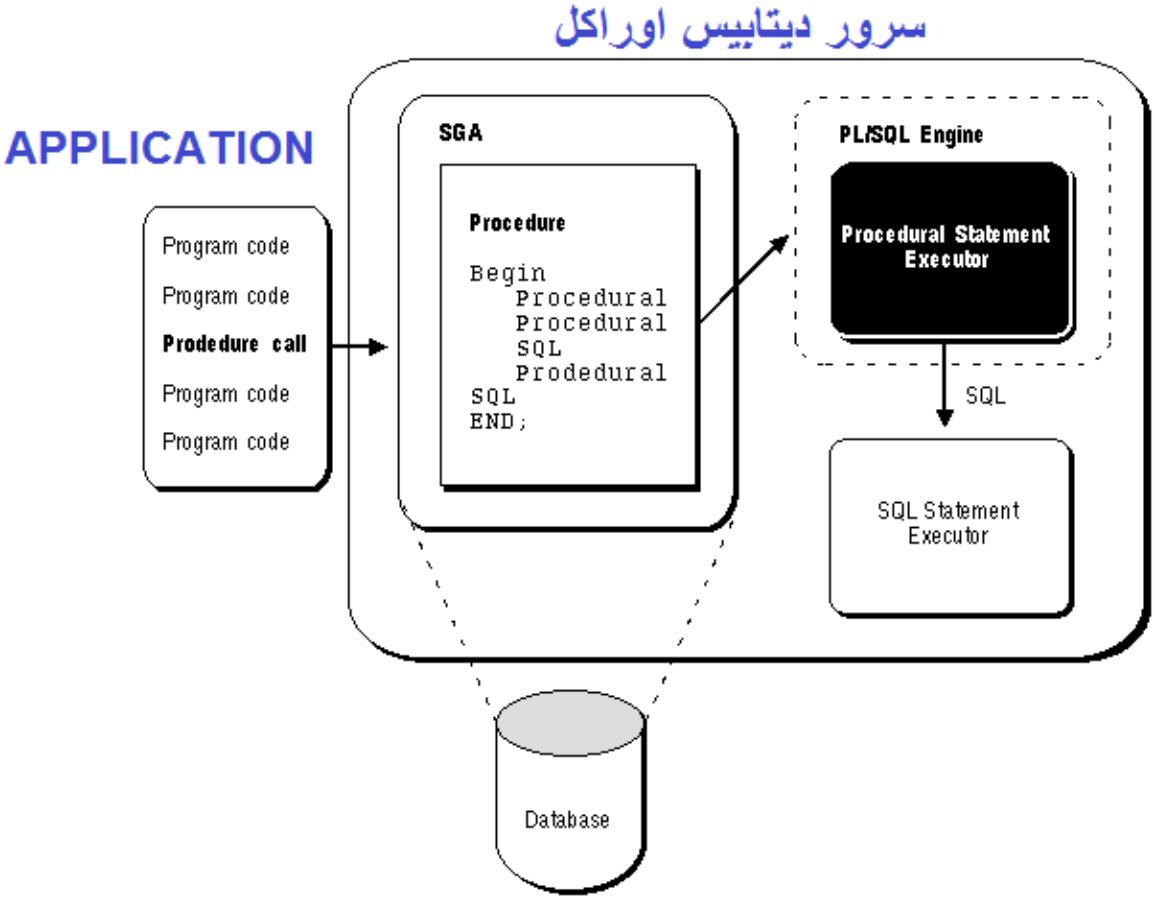
- ANONYMOUS BLOCK: در این روش بلاک به صورت بی نام تعریف می شوند ولی در دیتابیس ذخیره نمی شوند.
- STORED PROCEDURE : در این روش بلاک های برنامه در دیتابیس اوراکل ذخیره می شوند و بر اساس نام، توسط کاربران و یا APPLICATIONها فراخوانی و اجرا می شوند. این STORED PROCEDUREها می توانند به صورت FUNCTION، PROCEDURE یا PACKAGE تعریف شوند.

نکته: در دیتابیس اوراکل عملیات PARSING و اجرای دستورات SQL برای بلاک های PL/SQL و دستورات SQL درونی آنها نیز انجام می شود.

نکته: وقتی یک STORED PROCEDURE در دیتابیس اوراکل ایجاد می کنیم دیتابیس در زمان ذخیره سازی این بلاک عمل کامپایل را انجام می دهد. یعنی بلاک تعریف شده از لحاظ سینتکس و سمانتک بررسی می شوند و اگر خطایی پیدا نشد عمل ذخیره سازی انجام می شود.

۱۷.۴ چگونه STORED PROCEDURE اجرا می‌شود ؟

مراحل اجرای برنامه های PL/SQL که در دیتابیس ذخیره شده اند را در شکل زیر می بینید.



زمانی که APPLICATION یک برنامه ذخیره شده در دیتابیس را فراخوانی می کند دیتابیس اوراکل نسخه کامپایل شده این برنامه را در قسمت SHARED POOL از SGA ذخیره می کند. در ادامه، اجراکننده های PL/SQL و SQL(EXECUTOR)دستورات این برنامه ها را پردازش می کنند. توجه شود که سرور اوراکل اجرای برنامه های PL/SQL را با استفاده از PL/SQL ENGINE انجام می دهد.

نکته: می توان یک STORED PROEDURE را از درون یک بلاک دیگر چه از نوع ANONYMOUS یا STORED PROCEDUREفراخوانی کرد.

نکته: چند مورد از اجزای زبان PL/SQL عبارتند از:

- VARIABLES AND CONSTANTS: متغیرها و مقادیر ثابت که در داخل برنامه تعریف می شوند.
- CRUSORS : اشاره گرهایی که به صورت صریح یا ضمنی تعریف می شوند.

- EXECPTION :در زمان رخداد یک خطا، اجرای عادی بلاک PL/SQL متوقف می شود و EXECPTION HANDLERاجرا می شود.

نکته: می توان از DYNAMIC SQL در داخل بلاک های PL/SQL چه از نوع ANONYMOUS و یا STORED PROCEDUREاستفاده کرد.

نکته: SQL ENGINE دستورات SQL که در داخل یک بلاک قرار دارد را به صورت همزمان اجرا می کند. بنابراین PERFORMANCE بهتری خواهیم داشت.

۱۷.۵ تفاوت PL/SQL با SQL

در جدول زیر تفاوت های این دو زبان را مشاهده می کنید:

PL/SQL	SQL
بلاک یا مجموعه ای از کد که برای نوشتن یک بلاک برنامه یا PROCEDURE استفاده می شود.	در زبان SQL ما QUERY های مجزا را داریم که عملیات DML و DDL انجام می دهند.
این زبان از نوع PROCEDURAL است یعنی روش اجرای عملیات را مشخص می کند.	این زبان از نوع DECLARATIVE است. یعنی فقط کاری که باید انجام شود تعریف می شود.
تمام اجزای یک بلاک برنامه، همزمان اجرا می شود.	هر دستور به صورت مجزا اجرا می شود.
معمولا برای ایجاد APPLICATION استفاده می شود	معمولا برای دستکاری داده ها استفاده می شود.
می تواند شامل کدهای SQL باشد	نمی تواند شامل کدهای PL/SQL باشد
می توان از VARIABLE، LOOP و ... استفاده نمود.	دستورات کنترل ساختار (LOOP و ...) پشتیبانی نمی شوند.
توسط شرکت ORACLE طراحی شده است.	در ابتدا توسط شرکت IBM طراحی شد.