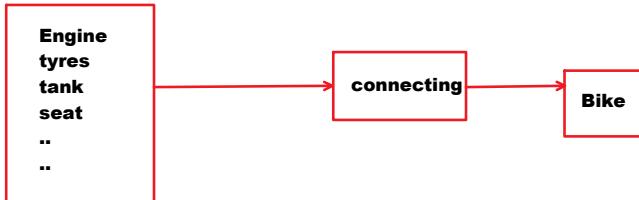


**Data**  
**Information**  
**Database**  
**DBMS**  
**RDBMS**  
**Metadata**

**Data:**

**Data** is a raw collection of facts about people, places & things ...etc.

raw => unordered [not in order]

**Data****Data:**

- is a raw collection of facts about people, places & things ...etc



- is unprocessed one.
- It is not meaningful form

**Information:**

- If the data is arranged in meaningful form it becomes Information.
- It processed one.
- It is in meaningful form.

**Types of Data:****2 Types:**

- **Structured Data** => letters, digits & symbols

Ex: 1234 Ramu h.no.1-2-123

- **Unstructured data** => images, audios, videos, documents

**Database:**  
is a collection of interrelated data in an organized form.

#### College Database

Student		
sid	sname	scity

Marks			
sid	M1	M2	M3

#### Online shopping database

Product				
pid	pname	QIH	Price	
Customer				
cid	cname	ccity		
Order				
oid	cid	pid	ord date	del date

#### Customer

cid	cname	ccity
1001	Arun	Hyd
1002	Kiran	Mumbai

→ column / property / attribute / field

Customer => Table / Relation

Table:  
is a collection of rows and columns

Field:  
holds individual values

row:  
is a set of field values

2 rows  
3 columns

#### A Database contains records & fields

#### Types of database:

2 types:

<b>OLTP</b>	<b>OLAP [DWH / DSS]</b>
<ul style="list-style-type: none"> <li>• <b>Online Transaction Processing</b></li> <li>• used to perform day-to-day operations</li> <li>• <b>C =&gt; Create R =&gt; Read U =&gt; Update D =&gt; Delete</b></li> <li>• <b>CRUD operations will be performed in OLTP</b></li> </ul>	<ul style="list-style-type: none"> <li>• <b>Online Analytics Processing</b></li> <li>• used for data analysis we maintain historical data.</li> <li>• <b>READ operation only</b></li> </ul>

#### **DBMS:**

- Database Management System / Software
- DBMS is a software that is used to create & maintain the database

**FMS => 1960s**



#### **RDBMS:**

- relational Database management System
- in this we maintain database in the form of tables.

**EX: Oracle SQL Server My SQL Postgre SQL dbase**

#### **Oracle:**

**is a Relational Database Management Software.**  
**is used to create & maintain the database in the form of tables.**  
**allows us to store, manipulate & retrieve the data of database.**

**manipulate => add / delete / modify**  
**retrieve => opening existing data**

#### **ORACLE**

**Duration : 3 Months**

**1 and half hour class**

**SQL**  
**PL/SQL**

**Metadata:**

- Data about the data

Student		
sid	sname	percentage
25	Ravi	56.78

**field name => sid**  
**table name => Student**  
**Data Type => Number**  
**field size => 4**

**sid Number(4) => 9999**

**ORACLE 19C**

- Oracle is a Relational Database management Software.
- Relation => Table
- In Oracle, create & maintain the database. In this , we maintain database in the form of tables.
- Store, manipulate & retrieve the data of database.
- manipulate => insert / delete / update
- retrieve => opening existing data
- Oracle 2nd version introduced in 1979.
- It is a product of "ORACLE" Company.
- Latest version is Oracle 19c

**Larry Ellison**

**1977 => Software Development Laboratories**

**1979 => Relational Software Inc.**

**1983 => Oracle Corp.**

**Tools of ORACLE :**

- 2 tools:
- SQL
  - PL/SQL

**SQL:**

- Structured Query Language.
- It is a query language that is used to write the queries.
- Query => is a request that is sent to database server by the client.
- Queries are used to communicate with the database.
- is a unified language. it is common for many relational databases.

**OOPS Concepts**

- Class**  
**Object**  
**Encapsulation**  
**Abstraction**  
**Inheritance**  
**Polymorphism**

Oracle	SQL Server	PostgreSQL
SQL	SQL	SQL

- SQL is 4GL. we much concentrate on what to do rather than how to do.
- Ex: `sqrt(100) => 10`
- is a Non-Procedural Language.
- provides Operators
- does not provide control structures.
- provides built-in functions
- Ex: sum of salaries => `sum(sal)`
- maximum salary => `max(sal)`

**Function =>**

## Task / Action / Job

### **Sub-Languages in SQL:**

**SQL provides 5 sub-languages:**

<b>DDL</b> <b>Data Definition Language</b>	<b>Create</b> <b>Alter</b> <b>Drop</b> <b>Truncate</b> <b>Rename</b> <b>Flashback [Oracle 10g]</b> <b>Purge [Oracle 10g]</b>
<b>DML</b> <b>Data Manipulation Language</b>	<b>Insert</b> <b>Delete</b> <b>Update</b> <b>Insert All [Oracle 9i]</b> <b>Merge [Oracle 9i]</b>
<b>DRL / DQL</b> <b>Data Retrieval Language /</b> <b>Data Query Language</b>	<b>Select</b>
<b>DCL / ACL</b> <b>Data Control Language /</b> <b>Accessing Control Language</b>	<b>Grant</b> <b>Revoke</b>
<b>TCL</b> <b>Transaction Control Language</b>	<b>Commit</b> <b>Rollback</b> <b>Savepoint</b>

### **DDL:**

- **Data Definition Language.**
- **It deals with metadata.**

### **Student**

sid	sname
20	Ramu

### **DML:**

- **Data Manipulation Language.**
- **Manipulation => Insert / Update / Delete**
- **It deals with data**

**fieldname => sid sname**  
**tablename => student**  
**datatype => number**  
**field size => number(4)**

### **DRL / DQL:**

- **Data Retrieval Language /**
- Data Query Language.**
- **Retrieve => opening existing data**
- **It deals with data retrievals.**

### **student**

sid	sname

### **DCL / ACL:**

- **Data Control Language /**
- Accessing Control Language**
- **It deals with data accessibility**

### **OWNER**

#### **HR**

#### **asstmanager**

#### **emp**

empno	ename	job	sal

#### **Read only [DQL]**

### **CLERK**

**DML**  
**DQL**

**TCL:**

- Transaction Control Language
- it deals with the transactions.

withdraw 10000

Rollback [Undo]  
Commit [Save]

**DDL:**

Create  
Alter  
Drop  
Truncate  
Rename  
Flashback  
Purge

**"Create" Command:**

is used to create the database objects  
like tables, view, indexes ...etc

< > => Any  
[ ] => Optional

**Syntax to create the table:**

```
Create Table <Table_name>
(
  <field_name> <data_type> [constraint <constraint_name> <constraint_type>,
  .....,
  .....]
);
```

**Data Types of SQL:****DDL:**

- Data Definition Language.
- It deals with metadata.
- DDL Commands are auto-committed.
- DDL Command = DDL Command + Commit
- Create, Alter, Drop, Truncate, Rename, Flashback, Purge

**Create:**

- is used to create the database objects  
like tables, views, indexes...etc.

DB  
Schema [User]  
DB Objects

**Tables**  
**Views**  
**Indexes**  
**Sequences**  
**Synonyms**  
**Materialized views**  
**functions**  
**procedures**  
**packages**  
**triggers**

**[ ] => Optional**  
**<> => Any**

#### Syntax to create a Table:

```

Create Table <Table_Name>
(
  <field_name> <data_type> [constraint <constraint_name> <constraint_type>,
  <field_name> <data_type> [constraint <constraint_name> <constraint_type>,
  .....,
  .....]
);

```

#### Data Types in SQL:

- Data Type tells the type of data which a column can hold.
- It also tells how much memory has to be allocated for the data.
- It identifies valid range for the data type.

sid	sname	dob
100 1	Ramu Sai	23-dec-2000 12-sep-1998
100 2	Arun	
100 3		

45  
67.89

'M'  
'Ravi'

#### Integer :

Number without decimal places

Ex: 123 78 23 1234

#### Floating Point:

Number with decimal places

Ex: 78.93 5000.00 1500.80

m1  
-----  
67  
SD

Number Related	Number(p) => Integer Number(p,s) => Floating Point
Character Related	Char(n) Varchar2(n) long clob

	<b>nchar(n)</b> <b>nvarchar2(n)</b> <b>nclob</b>
<b>Date &amp; Time Related</b>	<b>Date</b> <b>Timestamp [Oracle 9i]</b>
<b>Binary Related</b>	<b>BFILE</b> <b>BLOB</b>

<b>Number Related</b>	<b>Number(p) =&gt; Integer</b> <b>Number(p,s) =&gt; Floating Point</b>
-----------------------	---

#### Number(p):

- it is used to hold integer type data.
- p => precision => Max number of digits.
- p => range => 1 to 38
- -9999....99999 [-9 38digits] to 999....99 [9 38 digits]

```
age => 23
m1 => 78

tot_marks => 567
```

**number(38)**

#### Examples:

age number(2) => -99 to 99	m1
m1 number(3) => -999 to 999	----
empno number(4) => -9999 to 9999	100
custid number(6) => -999999 to 999999	
mobile_num number(10)	
aadhar_num number(12)	
debit_card_num number(16)	

#### Number(p,s):

- it is used to hold floating point type data.
  - p => Precision => Total Number of digits
  - s => Scale => Number of Decimal Places
- |            |       |
|------------|-------|
| percentage | ----- |
|            | 56.78 |
|            | 89.23 |

#### Example:

per Number(5,2) => -999.99 to 999.99	100.00
sal number(8,2) => -999999.99 to 999999.99	sal
sal number(9,2) =>	-----
	7000.00
	10000.00
p => range => 1 to 38	90,00,000.00
s => range => -84 to 127	

#### **Character related Data Types:**

##### **char(n):**

- it is used to hold a set of characters.
- n => Maximum number of characters
- Fixed length character data type.

##### **Varchar2(n):**

- it is used to hold a set of characters.
- n => Maximum number of characters
- Variable Length Character Data Type

**chardemo**

name1 [char(10)]	name2 [varchar2(10)]
sai7spaces	sai
arun6spaces	arun
kiran5spaces	kiran

**varchar => Oracle 2, 3, 4, 5, 6****varchar2 => introduced in Oracle 7 version****varchar was not following one of the E.F. Codd Rules.****there should be difference b/w space and null**

```
' ' => space
null => unknown
```

**In present versions, varchar and varchar2 both acts as same.**  
**Oracle company convention is, don't use varchar data type.**  
**varchar is deprecated data type.**

#### **Character Related Data Types:**

##### **char(n):**

- n => maximum number of chars.
- Fixed Length Char Data Type.
- maximum size: 2000 Bytes
- default size: 1
- to declare fixed length char fields, use it.
- Example:

State_Code	Country_Code	Gender	PAN char(10)	10 digits
-----	-----	-----	-----	
TS	IND	M		
AP	AUS	F		
MP	USA			
WB				

```

gender char          => size = 1
statecode char(2)   => size = 2

```

**varchar2(n):**

- n => maximum number of chars
- Variable Length Char Data Type.
- Maximum size: 4000 Bytes
- Default size: No default size. we must specify the size.
- Ex:
  - state\_Code char => default size = 1
  - ename varchar2 // ERROR
  - when we want to declare a field as variable length, then use it.

Ex:

Ename

-----

Raju  
Kiran  
Sai

char(n)	varchar2(n)
• Fixed length char data type	• Variable length char data type
• default size: 1	• default size: no default size. we must specify the size
• maximum size: 2000 Bytes	• max size: 4000 Bytes
• Ex: statecode	• Ex: ename

#### Long:

- it is used to hold characters.
- it is outdated.
- Maximum size: 2GB
- we can create only one long column in a table.
- Ex:
  - Cust\_feedback
  - Complaints
  - Experience\_summary

#### CLOB:

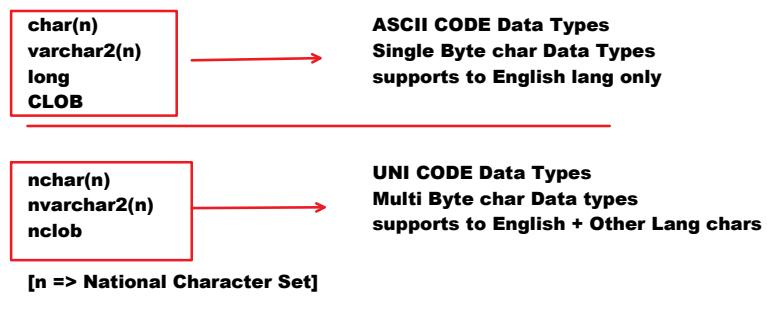
- Character Large Object.
- Maximum size: 4GB
- Ex:
  - Experience\_summary
  - Cust\_Feedback

char(n)  
varchar2(n)

ASCII CODE Data Types  
Single Byte char Data Types

In C-Lang,

char => 1 Byte => ASCII coding system



**In C-Lang,**  
**char => 1Byte => ASCII coding system**

**English**  
**256 chars => 0 to 255**

**A = 65 B=66 ....Z=90**  
**a=97 b=98 ..z=122**

**In Java,**  
**char => 2 Bytes => UNI coding system**

**English + other language chars**

<b>char(n)</b>	<b>2000 Bytes</b>
<b>nchar(n)</b>	<b>2000 Bytes</b>
<b>varchar2(n)</b>	<b>4000 Bytes</b>
<b>nvarchar2(n)</b>	<b>4000 Bytes</b>
<b>CLOB</b>	<b>4 GB</b>
<b>nclob</b>	<b>4 GB</b>

#### Date & Time Related data types:

##### Date:

- It is used to hold date values.
- It can also hold time value.
- It can hold date, month, year, hour, minute & second.
- It cannot hold fractional seconds [ milli seconds]
- Fixed Length Data Type.
- 7 Bytes memory is required.
- Default Date Format: DD-MON-RR Ex: 23-DEC-21
- to change date format use "nls\_date\_format" parameter.
- default time: 12:00:00 AM [ Mid Night]

##### changing date format:

Log in as DBA  
 user name:system  
 password: nareshit

Alter Session set nls\_date\_format = 'dd/mm/yyyy';

##### Timestamp:

- introduced in Oracle 9i.
- It is an extension of Date Data Type.
- It can hold Date, Month, Year, Hour, Minute, Second and Fractional Seconds.
- fixed length data type.
- memory: 11 Bytes

##### Date

##### Timestamp

cannot hold fractional secs  
 7 Bytes  
 23-dec-20 12:30:05 AM

can hold fractional secs  
 11 Bytes  
 23-dec-20 12:30:05.560000 AM

**cannot hold fractional secs**  
**7 Bytes**  
**23-dec-20 12:30:05 AM**

**can hold fractional secs**  
**11 Bytes**  
**23-dec-20 12:30:05.560000 AM**

#### **Binary related Data Types:**

- **BFILE**
- **BLOB**

**BFILE & BLOB data types are used to hold multimedia objects like images, audios, videos, documents ....etc.**

**Text file**  
hello

.txt

**Binary file**  
101100010100011  
1000111

.mp3  
.jpg  
.mp4  
.3gp  
.exe

#### **BFILE:**

- **Binary File**
- **it is used to hold multimedia object path.**
- **it acts like pointer to the multimedia object.**
- **it can be also called as "External Large Object Data Type".**
- **max size: 4 GB**
- **not secured one.**
- **bfilename() is used to maintain the path of the multimedia object.**

#### **BLOB:**

- **Binary Large Object**
- **It is used to maintain multimedia object inside of the database.**
- **Internal LOB data type.**
- **it is secured one.**
- **maximum size: 4GB**

#### **Syntax to create a Table:**

```
Create Table <Table_Name>
(
  <field_name> <data_type> [constraint <constraint_name> <constraint_type>,
  <field_name> <data_type> [constraint <constraint_name> <constraint_type>,
  .....,
  .....];
);
```

**Syntax to insert a record in table:**

```
Insert into <Table_Name>[(<column_list>)]
values(<value_list>);
```

**Examples on creating tables:**

**Example-1:**

**student**

Rno	Name	Marks
10	Ramu	456
20	Sai	789

Rno => Number(3) => 150  
Name => Varchar2(20)  
Marks => Number(4) => 1000

**Creating a Table:**

```
SQL> Create Table Student
(
  Rno Number(3),
  Name varchar2(20),
  Marks Number(4)
);
```

**To see tables list:**

```
SQL> Select * from tab;
```

Object name	Object type	Database	Schema [User]	DB Objects
Student	Table	.....	.....	-Tables -Views
.....	.....	.....	.....	.....

**User\_Tables**  
=> current user tables list

**All\_Tables**  
=> all users tables list

**to see structure of a table:**

-----  
**desc / describe =>**  
**it is used to see the structure of a table**

**Syntax:**  
  **desc <table\_name>;**

**Example:**  
  **desc student;**

**Inserting a record into table:**

**Insert into Student values(10,'Ramu',567);**

**Inserting records using parameters:**

**SQL> Insert into Student values(&rno,'&name',&marks);**

**Output:**

**Enter value for rno: 30**

**Enter value for name: Sravan**

**Enter value for marks: 678**

**Insert into Student values(30,Sravan,678)**

**1 row created**

**Note:**

**Run [or] / => command is used to repeat recent [above] command**

**SQL> /**

**Enter value for rno: 40**

**Enter value for name: A**

**Enter value for marks: 350**

**SQL> /**

**display all columns & all records:**

**Note: \* => All columns**

**SQL> Select \* from student;**

**Display name & marks:**

**SQL> Select name,marks from student; [limited columns]**

**display the student records whose marks are > 500:**

**employee**

empno	ename	job	sal	doj
-------	-------	-----	-----	-----

**Downloading Oracle Database:**

**oracle.com**

**Products => Oracle database**

**Download Oracle 19c**

**Windows 64 bit => ZIP File**

- Create a New Folder in C: Drive with name "Oracle".
- Enter into "Oracle" Folder
- Create new folder with the name "db\_home"
- Extract oracle software into "db\_home" folder.
- enter into db\_home folder.
- right click on last file "setup.exe" & select "run as administrator".

**Creating a User:**

- Log in as DBA
- username: system
- password: nareshit

**SQL>****Syntax to create a User:**

```
Create User <user_name> identified by <password>
default tablespace <table_space_name>
quota <memory> on <table_space_name>;
```

**Example:**

```
Create User c##oracle6pm identified by nareshit
default tablespace Users
quota unlimited on Users;
```

```
Grant connect, resource to c##oracle6pm;
```

**Changing Password:**

```
SQL> alter user c##oracle6pm identified by naresh;
```

to see tablespace names list:

"DBA\_Tablespace"

```
Select tablespace_name from DBA_Tablespace;
```

**Connect  
Resource**

1000000.00

empno => Number(4)  
ename => varchar2(20)  
job => varchar2(10)  
sal => Number(8,2)  
hiredate => date

Example-2:

Employee

empno	ename	job	sal	hiredate
-------	-------	-----	-----	----------

Insert the records

**Creating employee table:**

```
Create Table Employee(
empno number(4),
ename varchar2(20),
job varchar2(10),
sal number(8,2),
hiredate date
);
```

**DB  
Schema [User]  
Table**

**Inserting a Record:**

```
Insert into employee values(1001,'A','CLERK',5000,'25-sep-2020');
```

**Inserting records using parameters:**

```
Insert into employee  
values(&empno,'&ename','&job',&sal,'&hiredate');
```

**display all columns & records:**

```
Select * from employee;
```

display emp records whose salary is > 5000:

```
Select * from employee where sal>5000;
```

display empno, ename and sal:

```
Select empno,ename,sal from employee;
```

display empno, ename,sal for the employees who are earning 5000:

```
Select empno,ename,sal  
from employee  
where sal=5000;
```

**Syntax to create the table:**

```
Create Table <Table_Name>  
(  
    <field_name> <data_type> [constraint <constraint_name> <constraint_type>,  
    <field_name> <data_type> [constraint <constraint_name> <constraint_type>,  
    .....]  
)
```

**Constraints in SQL:**

- Constraint is a Rule which is applied on a column.
- Constraint is used to restrict the user from entering invalid data.
- It is used to maintain data integrity.
- Data integrity means, maintaining accurate & quality data.

**student**

sid	sname	M1
1	A	56
2	b	785

$M1 \geq 0$  and  $M1 \leq 100$

**There are 6 constraints available in SQL:**

- Primary Key
- Unique
- Not Null
- Check
- Default
- References [Foreign Key]

**Primary Key:**

- should not accept duplicate values.
- should not accept null values.

**Example:**

Employee PK			
empno	ename	job	sal
1001	Ramu	Clerk	6000
1002	Srinu	Clerk	5000
1003	Ramu	Manager	9000
1004	Arun	Clerk	5000
1002	Vijay	Manager	10000
	Sai	Clerk	7000

duplicate  

Null 

PK PK => Composite PK

SrNo	Sid	Cid	Fee
1	1001	20	5000
2	1001	30	6000
3	1002	20	5000
4	1001	20	5000

20 => Java  
30 => Python

**Unique:**

- does not accept duplicate values.
- accepts null values.

null  
duplicate

Example: 

cid	cname	mobile
1001	A	90123 45678
1002	B	
1003	C	90123 45678

Null   
Duplicate 

**Not Null:**

- does not accept null values
- can accept the duplicate values.
- it demands for a value.

Not Null

emp 

empno	ename	job	sal
1001		Clerk	5000
1002	Srikanth		
1003	Srikanth		

Null   
duplicate 

Constraint Type	Duplicate	Null
PK	No	No
Unique	No	Yes

Not Null	Yes	No
----------	-----	----

**PK = Unique + Not Null**

#### Check:

- is used to apply our own conditions on a column

#### Example:

##### Student

sid	sname	M1	
11	A	56	↙
12	B	456	X

Max Marks: 100

**Check(M1 >= 0 and M1 <= 100)**

Handwritten annotations:

- $56 \geq 0$  (True, T)
- $456 \geq 0$  (False, F)
- $56 \leq 100$  (True, T)
- $456 \leq 100$  (False, F)

##### Flight

Pid	Pname	seatno
1001	Ramu	506

**Check(seatno >= 1 and seatno <= 100)**

Handwritten annotations:

- $506 \geq 1$  (True, T)
- $506 \leq 100$  (False, F)

##### Gender

-----  
M  
F  
Z  
Q

**Check(Gender = 'M' or Gender = 'F')**

#### Default:

is used to apply default values to a column

**Country**  
-----  
India  
India  
India

Student

1000 xyz hyd  
1000 xyz hyd  
1000 xyz hyd  
1000 xyz hyd

sid	sname	fee	cname	ccity
1000	xyz	1000	hyd	
1000	xyz	1000	hyd	
1000	xyz	1000	hyd	
1000	xyz	1000	hyd	

fee default 1000,  
cname default 'xyz',  
ccity default 'hyd'

#### References [Foreign Key]:

Course

PK

cid	cname
10	Java
20	Python
30	HTML

Student

PK

child =

Sid	Sname	cid
1001	A	20
1002	B	30
1003	C	90

FK

Dept

PK

Deptno	Dname
10	Accounts
20	Research
30	Sales

emp

PK

error

empno	ename	Deptno
5001	A	20
5002	B	10
5003	C	70

FK

Foreign key refers to Primary key values of another table.

#### Null:

- Null is a value
- Null equals to empty or blank.
- When the value is unknown, use null.
- It can be accepted by any data type.
- something + null = null

#### emp

empno	ename	job	sal
1001	Ramu	clerk	5000
1002	Arun		

- something - null = null
- null value cannot be compared using comparison operator [=]
- "Is Null" Operator is used for null comparison.
- Null is not equals to 0 or space                '' 0

**Inserting null values in the table:**

**2 ways:**

- using NULL [Explicit]
- by inserting limited column values [Implicit]

**using NULL:**

**Syntax of Insert Command:**

```
Insert into <Table_Name>[(<column_list>)]
values(<value_list>);
```

**inserting a record using NULLs [Explicit]:**

```
insert into employee values(1004,'D','CLERK',null,null);
```

**inserting nulls by giving limited column values [Implicit]:**

```
insert into employee(empno,ename,job)
values(1005,'E','MANAGER');
```

**Examples on creating tables using constraints:**

**Example-1:**

std

sid	sname	M1
-----	-------	----

```
sid number(4) => set PK
sname varchar2(20) => set Not Null
M1 Number(3)      => set check => 0 to 100
```

```
Create Table std
(
  sid Number(4) primary key,
  sname varchar2(20) not null,
  m1 number(3) check(m1>=0 and m1<=100)
);
```

```
Insert into std values(1001,'A',56);
Insert into std values(null,'B',80); //ERROR
Insert into std values(1001,'B',78); //ERROR
Insert into std values(1002,null,67); //ERROR
Insert into std values(1003,'A',55);
Insert into std values(1004,'D',234); //ERROR
```

**Example-2:****std1**

sid	sname	Fee	Cname	CCity	
					15000.00

**sid => Number(4) => Unique**  
**sname => varchar2(20) => Not Null**  
**Fee => Number(7,2) => default 15000**  
**Cname => Varchar2(10) => default XYZ**  
**Ccity => varchar2(10) => default Hyd**

**Create Table std1**

```

(
    sid Number(4) Unique,
    sname Varchar2(20) Not Null,
    Fee Number(7,2) default 15000,
    Cname varchar2(10) default 'XYZ',
    Ccity varchar2(10) default 'Hyd'
);

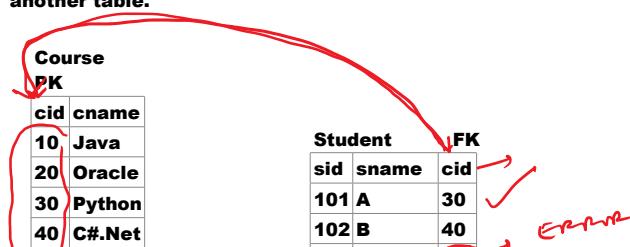
```

**Insert into std1(sid,sname) values(1001,'A');****Insert into std1(sid,sname) values(1002,'B');****Insert into std1(sid,sname) values(1003,'C');**

**PK**  
**Not Null**  
**Check**  
**Unique**  
**Default**

**Foreign Key:**

- is used to establish the relationship between two tables.
- To set Foreign key one common field is required.
- "References" keyword is used to set the foreign key.
- Foreign key refers primary key values of another table.



20	Oracle
30	Python
40	C#.Net

sid	sname	cid
101	A	30
102	B	40
103	C	80

### Course1

cid	cname
10	Java

cid => Number(2) => PK  
 cname => varchar2(10)

### Student1

sid	sname	cid
1001	A	30

sid => Number(4)  
 sname => varchar2(20)  
 cid => Number(2) => FK

#### Create Table Course1

```
(  
  cid Number(2) Primary Key,  
  cname varchar2(10)  
)
```

#### Create Table Student1

```
(  
  sid Number(4),  
  sname varchar2(20),  
  cid Number(2) References Course1(cid)  
)
```

Insert into Course1 values(10,'Java');

Insert into Course1 values(20,'Oracle');

Insert into Course1 values(30,'Python');

Insert into Course1 values(40,'C#.Net');

Insert into Student1 values(1001,'A',30);

Insert into Student1 values(1002,'B',40);

Insert into Student1 values(1003,'C',80);

### Example 2 on references:

deptno	dname
10	Sales
20	Accounts
30	Research

empno	ename	deptno
5001	A	20
5002	B	10
5003	C	90

#### Create Table dept1

```
(  
  deptno Number(2) Primary Key,  
  dname varchar2(10)
```

```
);

Create Table Employee1
(
  empno number(4) primary key,
  ename varchar2(20),
  deptno Number(2) References Dept1(deptno)
);
```

```
Insert into Dept1 values(10,'Sales');

Insert into Dept1 values(20,'Accounts');

Insert into Dept1 values(30,'Research');
```

```
Insert into Employee1 values(5001,'A',30);

Insert into Employee1 values(5002,'B',10);

Insert into Employee1 values(5003,'C',90);
```

#### Syntax to Create a Table:

```
Create Table <Table_Name>
(
<field_name> <data_type> [constraint <constraint_name> <constraint_type>,
<field_name> <data_type> [constraint <constraint_name> <constraint_type>
........................]
);
```

#### Naming Constraints:

- Giving names to the constraints is called "Naming Constraint".
- "Constraint" keyword is used to give the constraint name.

**Note:**

- Constraint name must be unique with in the schema.
- As an SQL Developer, we must define constraint name. If we don't define constraint name, oracle defines the constraint name implicitly by prefixing it with "sys\_c".

```
C##Oracle6PM
=> Student => c1
=> Emp => c1

C##Oracle11AM
=> Student => c1
```

Ex: sys\_c0012654

```
student3

|            |              |           |
|------------|--------------|-----------|
| <b>sid</b> | <b>sname</b> | <b>M1</b> |
|------------|--------------|-----------|

sid => PK => constraint name => c1
sname => Not Null => constraint name =>
c2
M1 => Check => constraint name=> c3
```

```
Create Table Student3
(
  sid Number(4) constraint c1 Primary Key,
  sname varchar2(20) constraint c2 Not Null,
  M1 Number(3) constraint c3 Check(M1>=0 and M1<=100)
);
```

"User\_Constraints" maintains all constraints information.

```
Select Table_name, Constraint_name, Constraint_Type
from User_Constraints;
```

#### **DDL Commands:**

```
Create
Alter
Drop
Truncate
Rename
Flashback
Purge
```

#### **Constraints can be applied in 2 ways:**

- **Column Level Constraints [ Inline Constraints ]**
- **Table Level Constraints [ Out of Line Constraints ]**

##### **• Column Level Constraints [ Inline Constraints ]:**

- If we define the constraint in column definition then it is called "Column Level Constraint".

**Ex:**

```
Create table t1(f1 number(3) primary key);
```

- All 6 constraints can be applied at column level.

**Table Level Constraints [Out of line Constraints]:**

- After defining all columns if define the constraint then it is called "Table Level Constraints".
- We can define 4 constraints at table level. They are:  
**PK, FK, Unique & Check**
- We cannot set not null and default at table level.

**Student5**

sid	sname	M1
-----	-------	----

**sid => Primary Key => Table Level => cc1**  
**sname => Not Null => Column Level => cc2**  
**M1 => Check => Table Level => cc3**

**Create Table student5**

```
(  
    sid number(4),  
    sname varchar2(20) constraint cc2 not null,  
    m1 number(3),  
    Constraint cc1 Primary Key(sid),  
    Constraint cc3 Check(M1>=0 and M1<=100)  
)
```

**course6**

cid	cname
10	C
20	Java
30	Cpp

**cid => PK => Table Level => course\_pk**

**student6**

sid	sname	cid
1201	A	20
1202	B	10

**cid => FK => Table Level => std\_fk**

**Create Table course6**

```
(  
    cid number(2),  
    cname varchar2(10),  
    constraint course_pk Primary Key(cid)  
)
```

**Create Table Student6**

```
(  
    sid number(4),  
    sname varchar2(20),  
    cid number(2),  
    constraint std_fk Foreign Key(cid) references course6(cid)  
)
```

**PK  
Unique  
Not Null  
Check  
Default  
References [FK]**

**DDL Commands:****Create**

**Alter**  
**Truncate**  
**Drop**  
**Rename**  
**Flashback**  
**Purge**

**Alter:**

- **Alter => Change**
- **used to change the structure of a table.**
- **using this command we can**
  - **add the columns => add**
  - **rename the columns => rename column**
  - **drop the columns => drop**
  - **modify the field sizes => modify**
  - **modify the data types => modify**
  - **add the constraints => add constraint**
  - **rename the constraints => rename constraint**
  - **disable the constraints => disable constraint**
  - **enable the constraints => enable constraint**
  - **drop the constraints => drop constraint**

**Syntax:**

```
Alter Table <Table_Name> add
    modify
    rename column
    drop column
    drop
    add constraint
    rename constraint
    disable constraint
    enable constraint
    drop constraint
```

**add:**

**used to add the columns**

**Syntax:**

```
add <column_definition>; => adding 1 column

add(<column_list>); => adding multiple columns
```

**std**

<b>sid</b>	<b>sname</b>
------------	--------------

**adding m1 column:**

```
alter table std add m1 number(3);
```

**adding multiple columns m2,m3:**

```
alter table std add(m2 number(3), m3 number(3));
```

**rename column:**

**is used to rename the columns**

**Syntax:**

**rename column <old\_name> to <new\_name>**

**Ex:**

std	sid	sname	M1	M2	M3
					M3 => Maths

**renaming a column:**

**Alter Table std rename column M3 to Maths;**

**Drop Column:**

**is used to drop [delete] one column.**

**Syntax:**

**drop column <column\_name>**

**Ex:**

std	sid	sname	M1	M2	Maths

**Dropping maths column:**

**Alter table std drop column Maths;**

**drop:**

**it is used to drop the multiple columns.**

**Syntax:**

**drop(<column\_list>);**

**Ex:**

std	sid	sname	M1	M2

**Dropping m1 and m2 columns:**

**Alter Table std drop(m1,m2);**

**modify:**

- we can change the field size
- we can change the data type
- we can add column level constraints

**Syntax:**

**modify <column\_Definition>; => modifying 1 column**

**modify(<column\_list>); => modifying multiple columns**

**std**

std	sid	sname

**sid => Numer(3)**  
**sname => varchar2(20)**

**modifying field size of sname from 20 to 30:**

**Alter Table std modify sname varchar2(30);**

**modifying data type [sid data type from number to varchar2]:**

**Alter Table std modify sid varchar2(20);**

**add constraint:**

- used to add the constraint to existing column
- we can add table level constraints only. It means, we can add PK, FK, Unique and Check.
- We cannot add not null and default.

**Syntax:**

**add constraint <constraint\_name> constraint type;**

**std**

**sid sname**

**add primary key to sid column:**

**Alter Table std add constraint s\_pk Primary Key(sid);**

**Unique(sid)**

**Check(M1>=0 and M1<=100)**

**foreign key(cid) references course(cid)**

**disable constraint:**

**used to disable the constraints.**

**Syntax:**

**disable constraint <constraint\_name>;**

**Ex:**

Select  
Constraint\_name,  
table\_name  
from  
User\_Constraints;

**disabling pk of sid column:**

**Alter table std disable constraint s\_pk;**

**Enable Constraint:**

**is used to enable the constraints**

**Syntax:**

**enable constraint <constraint\_name>;**

**Ex:**

**enabling pk of sid column:**

**Alter table std enable constraint s\_pk;**

**rename constraint:**

**used to rename the constraint**

**Syntax:**

```
rename constraint <old_name> to <new_name>;
```

**modify the constraint name of sid from s\_pk to s:**

```
Alter table std rename constraint s_pk to s;
```

**drop constraint:**

**used to drop the constraint**

**Syntax:**

```
drop constraint <constraint_name>;
```

**Ex:**

**drop the constraint s of sid:**

```
Alter table std drop constraint s;
```

**DDL:**

- Create
- Alter
- Truncate
- Drop
- Flashback
- Purge
- Rename

**Truncate:**

- used to delete complete information from a table.
- It also releases the memory

**Syntax:**

```
Truncate Table <table_name>;
```

**Ex:**

```
Truncate Table Employee;
```



**Block**

**Drop**

**Flashback [Oracle 10g]**

**Purge [Oracle 10g]**

**Drop:**

- is used to drop the database objects like tables, views, indexes...etc.
- When we drop the table it will be moved to recycle bin.
- dropped table address will be maintained in recycle bin

**Syntax for Dropping Table:**

```
Drop Table <Table_Name> [Purge];
```

**Ex:**

```
Drop Table Employee;
```

```
show recyclebin
```

**Flashback:**

- used to recollect dropped tables from recycle bin.

**Syntax:**

```
Flashback Table <Table_Name> to before drop [rename to <new_name>];
```

**Ex:**

```
Flashback Table employee to before drop;
```

```
Flashback table employee  
to before drop  
rename to employee_old;
```

**Purge:**

- used to remove the dropped table from recycle bin.
- It will be removed permanently. we cannot recollect it.

**Syntax:**

```
Purge Table <Table_Name>;
```

**Ex:**

```
Purge Table Employee;
```

```
//Employee table will be deleted from recycle bin
```

```
To clear the recyclebin:
```

```
Purge recyclebin;
```

**Drop**  
**Flashback**  
**Purge**

**Drop std1 table permanently:**

```
Drop table std1;
Purge table std1;
(or)
Drop Table std1 purge;
```

**Rename:**

**used to rename the database objects like  
tables, view, indexes ..etc.**

**Syntax:**

```
Rename <old_name> to <new_name>;
```

**Ex:**

```
Rename student1 to std;
```

**DDL => deals with metadata**

**Create  
Alter  
Truncate  
Drop  
Flashback  
Purge  
Rename**

**Create Table**

**Alter Table  
Truncate Table  
Drop Table  
Flashback Table  
Purge Table  
Rename <old\_name> to <new\_name>**

std	
sno	sname

**Truncate**  
**desc std**

**DDL  
DQL / DRL  
DML  
DCL / ACL  
TCL**

**DQL / DRL:**

- **DQL => Data Query Language**
- **DRL => Data Retrieval Language**
- **Query => is a request which is sent to DB Server**
- **Retrieve => Opening existing data**
- **SQL provides only one DRL Command. i.e. SELECT**
- **SELECT command is used to retrieve the data from existing tables.**

**Syntax:**

```
Select [Distinct] Column_List / *
From <Table_list>
[Where <Condition>]
[Group By <column_list>]
[Having <condition>]
[Order By <column_list> Asc / Desc];
```

English	SQL
Sentences	QUERIES
Words	CLAUSES

**SELECT => used to specify column list**  
**FROM => used to specify the table names**  
**WHERE => used to write the condition on rows**

emp

empno	ename	job	sal	deptno	hiredate
-------	-------	-----	-----	--------	----------

```
Select empno,ename,sal
From emp
Where job='MANAGER';
```

iphone 12 pro

- **Using Select command we can display**  
=> all columns & all records  
=> limited columns  
=> limited rows  
=> limited rows & columns

**Display all columns & all records of emp table:**

**Select \* From Emp;**

**\*=> All Columns**

**Execution Order:**

**From**

**Where**

**Select**

**Display Limited Columns:**

**display empno, ename, job, sal columns of emp table:**

**Select empno, ename, job, sal  
From emp;**

**From emp:**

empno	ename	job	sal	mgr	hiredate	comm	deptno
1001							
1002							
1003							
1004							
1005							

**Select empno, ename, job, sal :**

empno	ename	job	sal
1001			
1002			
1003			
1004			
1005			

**Limited Records:**

**Display the emp records whose sal is >2500:**

**Select \* from emp  
where sal>2500;**

**From emp:**

empno	ename	job	sal	mgr	hiredate	comm	deptno
1001			5000				
1002			2000				

1003		6000					
1004		2300					
1005		2500					

where  $\text{sal} > 2500$ : -  $\text{S} \Rightarrow > 2500$

empno	ename	job	sal	mgr	hiredate	comm	deptno
1001			5000				
1002			2000				
1003			6000				
1004			2300				
1005			2500				

empno	ename	job	sal	mgr	hiredate	comm	deptno
1001			5000				
1003			6000				

Select

empno	ename	job	sal	mgr	hiredate	comm	deptno
1001			5000				
1003			6000				

Display limited rows & columns:

display empno,ename,sal of the employees whose sal <1800:

Select empno,ename,sal  
From emp  
where sal<1800;

Operators in SQL:

- Operator  $\Rightarrow$  is a symbol / text
- Operators are used to perform operations like arithmetic or logical operations

Oracle SQL provides following Operators:

Arithmetic	+ - * /
Relational / Comparison	< > $\geq$ $\leq$ = $\neq$ / $\neq$ not equals
Logical	And Or Not
Special	In Not In Between And Not Between And Like Not Like Is Null Is Not Null  Any Exists All
Set	Union Union All Intersect Minus

**Examples on Arithmetic Operators:**

+	Addition
-	Subtraction
*	Multiplication
/	Division

**Dual:**

- is a temporary table
- to work non-database values, we use "dual"

**Select empno,ename From emp;**

**Select 2+3 From Dual;**

**Output:**

**2+3**

-----

**5**

**Giving alias name to a column:**

**Select 2+3 sum from dual;**

**SUM**

-----

**5**

**Column Alias Name:**

- Column alias names are used display column headings in the output
- If we want to maintain case & spaces then specify alias name in double quotes.

**Exs:**

**Select 2+3 sum from dual;**

**SUM**

-----

**5**

**Select 2+3 "sum of 2 nums" from dual;**

**sum of 2 nums**

-----

**5**

- we can use "as" keyword to specify the alias name. Using it is optional.

**Arithmetic Operators:**

std				M1+M2+M3		
sid	Sname	M1	M2	M3		
1001	A	50	70	60		
1002	B	80	30	50		

```
Create Table std(
sid Number(4),
sname varchar2(20),
M1 Number(3),
M2 Number(3),
M3 Number(3)
);
```

```
Insert into std values(1001,'A',60,80,70);
```

```
Insert into std values(1002,'B',60,30,90);
```

**Calculate Total marks & average marks:****calculating total:**

```
Select sid, sname, M1+M2+M3 as TOTAL from std;
```

sid	sname	TOTAL
.....	.....	.....

```
Select sid,sname, M1+M2+M3 as "TOTAL MARKS"
from std;
```

sid	sname	TOTAL MARKS
.....	.....	.....

**Calculating average marks:**

```
Select sid, sname, (M1+M2+M3)/3 as "Avrg Marks" from std;
```

sid	sname	Avrg Marks
.....	.....	.....

**emp**

empno	ename	job	sal	hiredate	mgr	comm	deptno
7900	A		4500				

**Calculate Annual Salary:**

```
Select empno, ename, sal*12 as "Annual Salary" from emp;
```

empno	ename	Annual Salary
.....	.....	.....

**Calculate TA, HRA, DA, ITAX and GROSS Salary:**

10% on sal as TA => sal\*10/100 (or) sal\*0.1

20% on sal as HRA => sal\*20/100 (or) sal\*0.2

15% on sal as DA => sal\*15/100 (or) sal\*0.15

5% on sal as TAX [Deducting] => sal\*5/100 (or) sal\*0.05

**GROSS => sal+TA+HRA+DA-TAX**

```
Select empno, ename, sal,  
sal*0.1 as TA,  
sal*0.2 as HRA,  
sal*0.15 as DA,  
sal*0.05 as TAX,  
sal+sal*0.1+sal*0.2+sal*0.15-sal*0.05 as GROSS  
from emp;
```

#### **Relational Operators [Comparison Operators]:**

- are used to compare two values

<	less than
>	greater than
<=	less than or equals to
>=	greater than or equals to
=	equals to
!= / <> / ^=	not equals to

**display the emp records whose salary is greater than 2800:**

**Select \* from emp where sal>2800;**

**display the emp records whose salary is less than 1000:**

**Select \* from emp where sal<1000;**

**display the emp records whose sal is 3000 or greater:**

**Select \* from emp where sal>=3000;**

**display the emp records whose sal is 1800 or less:**

**Select \* from emp where sal<=1800;**

**display the emp record whose name is 'SMITH':**

**Select \* from emp where ename='SMITH';**

**display all managers records:**

**Select \* from emp where job='MANAGER';**

**display the emp records except clerks:**

**Select \* from emp where job<>'CLERK';**

**display the emp records who are working in 10th dept:**

**Select \* from emp where deptno=10;**

**display the emp record whose empno is 7900:**

**Select \* from emp where empno=7900;**

**display the emp records who joined after 1982:**

**Select \* from emp where hiredate>'31-DEC-1982';**

**display the emp records who joined before 1982:**

**Select \* from emp where hiredate<'1-JAN-1982';**

#### **Logical Operators:**

**are used to perform logical operations.**

**And**  
**Or**  
**Not**

**And, Or are used to write multiple conditions.**

c1	c2	c1 and c2	c1 Or c2
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F

**And => all conditions should be satisfied**  
**Or => at least one condition should be satisfied**

**std**

sid	sname	M1	M2	M3
1001	A	60	90	80
1002	B	80	30	50

**Max Marks: 100**

**Min Marks: 40 for pass in each subject**

**display passed students records:**

```
Select * from std  
where M1>=40 and M2>=40 and M3>=40;
```

**display failed students records:**

```
Select * from std  
where M1<40 Or M2<40 Or M3<40;
```

**display all Managers and Clerks Records:**

```
Select * from emp  
where job='MANAGER' or job='CLERK';
```

**display the managers records whose sal is less than 2800:**

```
Select * from emp  
where job='MANAGER' and sal<2800;
```

**display the managers records whose sal is > 2800 and joined after 1981 april:**

```
Select * from emp  
where job='MANAGER' and sal>2800 and hiredate>'30-APR-1981';
```

**display the emp records who are working in 10 and 30:**

```
Select * from emp  
where deptno=10 or deptno=30;
```

**display the emp records who are earning more than 1000 and less than 2000:**

```
Select * from emp  
where sal>1000 and sal<2000;
```

**display the emp records whose empnos are 7369, 7900, 7902:**

```
Select * from emp  
where empno=7369 or empno=7900 or empno=7902;
```

**display the emp records whose names are ALLEN, SMITH and BLAKE:**

```
Select * from emp  
where ename='ALLEN' or ename='SMITH' or ename='BLAKE';
```

display all emp records except clerks and managers:

```
Select * from emp  
where not(job='CLERK' or job='MANAGER');
```

#### Special Operators:

- All special operators are comparison operators

In	Not In
Between And	Not Between And
is null	is not null
like	not like

Any
Exists
All

In:

#### Syntax:

`column_name In(val1,val2,val3....valn)`

- "In" is used to compare column value with a list of values.
- It avoids of writing multiple equality conditions using or.

#### Examples on "IN":

```
display the emp records whose empnos are 7369, 7900,  
7902;
```

```
Select * from emp  
where empno In(7369,7900,7902);
```

```
Display the emp records who are working in deptno 10 and  
30;
```

```
Select * from emp  
where deptno In(10,30);
```

display the emp records other than managers and clerks:

```
Select * from emp  
where job not in('MANAGER','CLERK');
```

```
display the emp records whose names are ALLEN, BLAKE &  
SMITH;
```

```
Select * from emp  
where ename In('ALLEN','BLAKE','SMITH');
```

```
display all emp records execpt the emp record whose empno  
is 7369;
```

```
Select * from emp  
where empno not in(7369);
```

display the emp records who are not working in deptno 30:

```
Select * from emp  
where deptno not in(30);
```

#### Between And:

#### Syntax:

`column_name Between <min_value> and <max_value>`

**Ex:**

**where sal between 1000 and 2000**

- **is used to compare column value with a range of values**

**display the emp records whose salary is between 1000 and 2000:**

**Select \* from emp  
where sal Between 1000 and 2000;**

**display the emp records who joined in 1982:**

**Select \* from emp  
where hiredate Between '1-JAN-1982' and '31-DEC-1982';\**

**display the emp records who joined in 1981,1982,1983:**

**Select \* from emp  
where hiredate between '1-JAN-1981' and '31-DEC-1983';**

**display the emp records whose salary is not between 2000 and 3000:**

**Select \* from emp  
where sal not between 2000 and 3000;**

**IS NULL:**

**Syntax:**

**<column\_name> IS NULL**

**comm**

**-----**

**500**

- **it us used to compare column value with null value.**

**300**

**400**

**Display the emp records who are getting commission:**

**2000**

**Select \* from emp  
where comm is not null;**

**Display the emp records whose comm is null:**

**Select \* from emp  
where comm is null;**

**Like:**

- **is used to compare column value with text pattern.**
- **It is used to retrieve the records based on text pattern matching.**

**Syntax:**

**column\_name like 'text\_pattern'**

- **Oracle provides two wild card characters for text pattern searching:**

<b>_</b>	<b>replaces 1 char</b>
<b>%</b>	<b>replaces 0 or any no of chars</b>

**Display the emp records whose names are started with 'S':**

**Select \* from emp where ename like 'S%';**

**Display the emp records whose name is ended with 'S':**

**Select \* from emp where ename like "%S";**

**Display the emp records whose name is having 'A' as middle char:**

**Select \* from emp where ename like "%A%";**

**Display the emp records whose names are ended with 'RD':**

**Select \* from emp where ename like "%RD";**

**Display the emp records which are having second character as 'A':**

**Select \* from emp where ename like '\_A%';**

**Display the emp records whose name is starter with D and ended with D:**

**Select \* from emp where ename like 'D%D';**

**Display the emp records whose name is having 4 chars:**

**Select \* from emp where ename like '\_\_\_\_';**

**Display the emp records who are getting 3 digit salary:**

**Select \* from emp where sal like '\_\_\_';**

**Display the emp records who are joined in 1982:**

**Select \* from emp where hiredate like "%82";**

**Display the emp records who are joined in 1982 & 1983:**

**Select \* from emp  
where hiredate like '%82' Or hiredate like '%83';**

**Display the employee records who are joined in December month:**

**Select \* from emp  
where hiredate like '%DEC%';**

**Display the emp records who are joined in first 9 days in the month:**

**Select \* from emp  
where hiredate like '0%';**

**Display the emp records whose names are not started with S:**

**Select \* from emp where ename not like 'S%'**

**Display the emp names whose names are having '\_':**

**Select \* from emp where ename like "%\\_%" Escape '\';**

**name**

**.....**

**sai\_teja => %\\_%**

**Display the emp records whose names are having '%':**

**Select \* from emp  
where ename like "%\%%" escape '\';**

**Example on concatenation operator:**

<b>Fname</b>	<b>Lname</b>
<b>-----</b>	<b>-----</b>

Fname	Lname	
-----	-----	
raj	kumar	raj kumar
sai	krishna	sai krishna

Select fname || ' ' || lname name from emp;

#### **DML Commands:**

- Data manipulation Language.
- Manipulation => Insert / Delete / Update [modify]
- It deals with data.
- All DML Commands are not auto-committed.

**DDL Commands are auto-committed**

**DML Commands are not auto-committed**

#### **TCL Commands:**

- Commit
- Rollback
- SavePoint

#### **Commit [SAVE]:**

- used to make the changes permanent. it means, it is used to save the transactions.
- when we use commit, the changes of oracle instance will be applied to Database.

**ORACLE SERVER = ORACLE INSTANCE + DB**

#### **Rollback [UNDO ALL]:**

- It is used to cancel the previous transactions.
- When we use rollback command, it cancels uncommitted data.
- After commit, we cannot use rollback.

**DDL Commands are auto-committed**

**DDL Command = DDL Command + COMMIT**

**DML Commands are not auto-committed**

#### **DML Commands:**

- Data manipulation Language.
- Manipulation => Insert / Delete / Update [modify]
- It deals with data.
- All DML Commands are not auto-committed.
- After using DML command, to save the transaction use "COMMIT" command.
- After using DML Command, to cancel the transaction use "ROLLBACK" command.

Insert  
Insert  
Insert  
Insert

Create table t2  
Insert  
Insert

<b>Insert</b>	<b>Create table t2</b>
<b>Insert</b>	<b>Insert</b>
<b>Insert</b>	<b>Insert</b>
<b>Insert</b>	<b>Create table t3 =&gt; Create+Commit</b>
<b>Insert</b>	<b>Insert</b>
<b>Rollback</b>	<b>Insert</b>
	<b>Rollback</b>

#### DML Commands are:

**INSERT**  
**DELETE**  
**UPDATE**  
**INSERT ALL**  
**MERGE**

#### Insert:

- used to insert the records into table.
- using this command we can
  - insert all column values
  - insert column values using parameters
  - insert limited column values
  - insert limited column values by changing the order
  - copy the records from existing table

#### Syntax of Insert Command:

```
Insert into <table_name>[(<column_list>)]
values(<value_list>);
```

#### Example:

**Customer**  

<b>cid</b>	<b>cname</b>	<b>ccity</b>
------------	--------------	--------------

**Create Table Customer**  
`(  
 cid Number(4),  
 cname varchar2(10),  
 ccity varchar2(10)  
);`

#### Inserting all column values:

```
Insert into customer values(2001,'A','Hyd');
```

#### Inserting records using parameters:

```
Insert into customer values(&A,'&B','&C');
```

Enter value for A:2002

Enter value for B:B

Enter value for C: mumbai

/

Enter value for A:

..

**Inserting limited column values:**

```
Insert into customer(cid,cname)
values(2005,'E');
```

**Inserting records by changing order:**

```
Insert into customer(cname,cid)
values('F',2006);
```

**Inserting records by copying records from existing table:**

```
copy emp table records into customer;
```

```
Insert into customer(cid,cname)
select empno,ename from emp;
```

```
copy manager records into customer table;
```

```
Insert into customer(cid,cname)
select empno,ename from emp where
job='MANAGER';
```

**Delete:**

- used to delete the records.
- Using this command we can
  - delete single record
  - delete a set of records
  - delete records using parameters
  - delete all records

**Syntax:**

```
Delete from <table_name> [where <condition>];
```

**Delete the emp record whose empno is 7900 [deleting single record]:**

```
Delete from emp where empno=7900;
```

**Delete the emp records whose job title is 'ANALYST' [Deleting a set of records]:**

```
Delete from emp where job='ANALYST';
```

**Delete the employee records who are having more than 40 years experience:**

```
Delete from emp where (sysdate-hiredate)/365>40;
```

**Delete the emp records whose deptnos are 10 & 30:**

```
Delete from emp where deptno in(10,30);
```

**Delete records using parameters:**

```
Delete from emp where empno=&empno;
Enter value for empno: 7369
```

```
/
```

**Enter value for empno: 7499**

**Deleting all records:**

```
Delete from emp;
```

(or)  
Delete emp;

Delete all emp records except CLERK and SALESMAN:

Delete from emp  
where job not in('CLERK','SALESMAN');

Delete from emp; --all records will be deleted

truncate table emp; --all records will be deleted

Differences b/w Delete & truncate:

Delete	Truncate
<ul style="list-style-type: none"><li>• it is DML Command.</li><li>• using it, we can delete single record or a set of records or all records.</li><li>• "where" clause can be used in this command</li><li>• It is not auto-committed.</li><li>• can be rolled back</li><li>• does not release the memory</li><li>• works slower than "truncate"</li><li>• deletes row by row</li></ul>	<ul style="list-style-type: none"><li>• it is DDL Command.</li><li>• we can delete all records only. we cannot delete single record or a set of records.</li><li>• "where" clause cannot be used in this command</li><li>• it is auto-committed</li><li>• It cannot be rolled back</li><li>• releases the memory.</li><li>• works faster than "delete"</li><li>• deletes page by page [page =&gt; block]</li></ul>

Truncate	Drop
<p>deletes all records only. it will not delete table structure</p>	<p>deletes entire table</p>

**Update:**

- used to update (modify) the records.
- Using this command we can
  - update single column value of single record
  - update multiple column values of single record
  - update a set of records
  - update using parameters
  - update all records
- Using this command we can do calculations.

**Syntax:**

```
Update <table_name>
set <column_name>=<value>
[,<column_name>=<value>,.....]
[where <condition>];
```

Increase RS 2000 salary to an employee whose empno is 7902:

Update emp set sal=sal+2000 where empno=7902;

Update the job value as manager and sal value as 8000 to the emp whose empno is 7499:

**Update emp set  
job='MANAGER', sal=8000  
where empno=7499;**

**Increase 10% sal to the employees who are getting commission:**

**Update emp set sal=sal+sal\*0.1  
where comm is not null;**

**Update comm value as null to the employees who are getting commission:**

**For null comparison use "is null"  
For null assignment use " = "**

**Update emp set comm=null where comm is not null;**

**Increase 10% sal and set comm as 500 for the employees who are not getting commission:**

**Update emp set sal=sal+sal\*0.1, comm=500  
where comm is null;**

**Transfer 10th dept employees to 20th dept:**

**Update emp set deptno=20 where deptno=10;**

**Increase 20% sal and 10% comm to the employees who are having more than 40 years experience:**

**Update emp set sal=sal+sal\*0.2,  
comm=comm+comm\*0.1  
where (sysdate-hiredate)/365>40;**

**Increase 15% salary to all clerks:**

**Update emp set sal=sal+sal\*0.15  
where job='CLERK';**

**Increase 10% sal to all employees:**

**Update emp set sal=sal+sal\*0.1;**

student						
sid	sname	M1	M2	M3	Total	Avrg
1001	A	60	80	70		
1002	B	80	50	70		

**Update student set Total=M1+m2+M3;  
Update student set Avrg=Total/3;  
[OR]**

**Update student set Total=M1+M2+M3,  
Avrg=(M1+M2+M3)/3;**

**Emp**

empno	ename	job	sal	ta	hra	Da	ITAX	Gross
-------	-------	-----	-----	----	-----	----	------	-------

**10% on sal => TA  
20% on sal=> HRA  
15% on sal=> DA  
5% on sal => TAX [Deducting]  
GROSS => sal+ta+hra+da-itax**

**DML**

-----

**Insert**  
**Update**  
**Delete**

**Insert All**  
**Merge**

**Update command using parameter:**

**update salary to different employees with  
different salary values as following:**

**7499 => 8000  
7900 => 7000  
7902 => 7500**

**Update emp set sal=&sal where  
empno=&empno;  
enter value for sal:8000  
enter value for empno: 7499**

**/  
enter value for sal:7000  
enter value for empno: 7900**

**/  
enter value for sal:7500  
enter value for empno: 7902**

**SQL:**

<b>DDL</b>	<b>DML</b>	<b>DRL/DQL</b>	<b>TCL</b>	<b>DCL/ACL</b>
<b>Create</b>	<b>Insert</b>	<b>Select</b>	<b>Rollback</b>	<b>Grant</b>
<b>Alter</b>	<b>Update</b>		<b>Commit</b>	<b>Revoke</b>
<b>Drop</b>	<b>Delete</b>		<b>Savepoint</b>	
<b>Truncate</b>	<b>Insert All</b>			
<b>Rename</b>	<b>Merge</b>			
<b>Flashback</b>				
<b>Purge</b>				

**TCL Commands:**

- Transaction Control Language.**

**Commit => Save the transaction**

**Rollback => to cancel the transaction**

**SavePoint:**

**used to set margin for rollback.**

**Syntax:**

**Savepoint <save\_point\_name>;**

**Example:**

```
Savepoint abc;
```

**Insert => 2001**

**Savepoint aaa;**

**Delete => 7900**

**Insert => 2001**

**Update => 2000 sal to 7499**

**Savepoint bbb;**

**Rollback;**

**Delete => 7900**

**Savepoint ccc;**

**Update => 2000 sal to 7499**

**Rollback to bbb;**

**TCL:**

**Rollback**

**Commit**

**Savepoint**

**DCL / ACL Commands:**

- **Data Control Language / Accessing Control Language.**
- **It deals with data accessibility.**

**2 commands:**

**Grant**

**Revoke**

**DBA**

**Grant:**

**is used to grant the permissions on database objects like tables to other users.**

**raju**

**emp =>**

**Grant => select**

**Revoke**

**Syntax:**

**kiran**

**select**

```
Grant <privileges_list>
on <db_object_name>
to <user_name>;
```

**Revoke:**

**used to cancel the permissions on database objects from the users.**

**Syntax:**

```
Revoke <privileges_list>
On <db_object_name>
From <user_name>;
```

<b>privileges_list</b>
<b>Select</b>
<b>Insert</b>
<b>Update</b>
<b>Delete</b>
<b>All</b>

**Create two common users with the names c##userA, c##userB:**

**Syntax to create the User:**

```
Create user <user_name>
identified by <password>
default tablespace <tablespace_name>
quota <memory> on <tablespace_name>;
```

**Note:**

**Common user name must be prefixed with c##.**

**User name is not case sensitive. But password is case sensitive.**

**Log in as DBA:**

```
username: system
password: nareshit
```

**Creating userA:**

```
Create user c##userA
identified by usera
default tablespace users
quota unlimited on users;
```

**Grant connect, resource roles:**

```
connect => for logging in into db we are giving the permission
resource => for creating tables we are giving the permission
```

**Grant connect, resource to c##userA;**

**Creating userB:**

```
Create user c##userB
identified by userb
default tablespace users
quota unlimited on users;
```

**Granting connect & resource roles to userB:**

**Grant connect, resource to c##userB;**

**To see all user names:**

```
select username from all_users;
```

**c##USERA****Create table std**

sid	sname
1	A
2	B

**Select \* from std;****Grant Select on std  
to c##userB;****Grant Insert, Update on std  
to c##userB;****to give all permissions:****Grant all on std  
to c##userB;****userA is granting permission to userB to  
grant permission to other users:****Grant All on std  
to c##userB with grant option;****Revoke all on std  
from c##userB;****c##USERB****Select \* from c##USERA.std;****ERROR => insufficient privileges****Select \* from c##userA.std;****displays the records****Insert into c##userA.std values(5,'Z');  
ERROR => insufficient privileges****Delete from c##useA.std where sid=1;  
ERROR => Insufficient privileges****Update c##userA.std set sid=101  
where sid=1;  
ERROR => Insufficiet privileges****Insert into c##userA.std values(5,'Z');  
row inserted****Delete from c##useA.std where sid=1;  
ERROR => Insufficient privileges****Update c##userA.std set sid=101  
where sid=1;  
row updated****Grant select  
on c##userA.std  
to c##oracle6pm;****ERROR****Grant select  
on c##userA.std  
to c##oracle6pm;****grant succeeded****Select \* from c##userA.std;****ERROR => Insufficient privileges**

#### Insert All & Merge Commands:

- introduced in Oracle 9i.
- These two commands are used to perform ETL Operations.

ETL => Extract - Transfer - Load

#### Insert All:

- This command is insert multiple records in single table or multiple tables.
- we can perform ETL Operations.
- It avoids of writing multiple insert commands.

"Insert All" can be used in 2 ways:

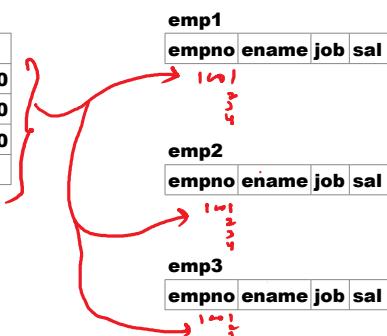
- Unconditional Insert All
- Conditional Insert All

Syntax of Unconditional Insert All:

```
Insert All
into <table-1>(<column_list>) values(<value_list>)
into <table-2>(<column_list>) values(<value_list>)
into <table-3>(<column_list>) values(<value_list>)
.....
Select Statement;
```

Insert Emp Table records into emp1,emp2 & emp3:

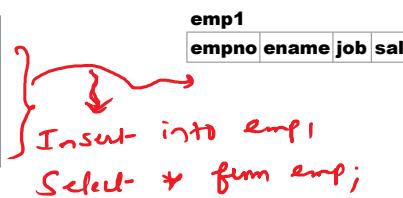
Emp			
empno	ename	job	sal
1001	A	CLERK	10000
1002	B	MANAGER	15000
1003	C	MANAGER	18000
1004	D	CLERK	8000



Copying records from one table to another:

Copying emp table records to emp1:

Emp			
empno	ename	job	sal
1001	A	CLERK	10000
1002	B	MANAGER	15000
1003	C	MANAGER	18000
1004	D	CLERK	8000



Copying Table:

Create Table emp1 as Select \* from emp;

**Syntax for Copying Records:**

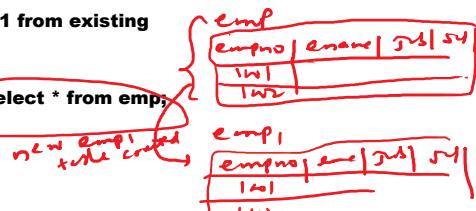
```
Insert into <table_name> <Select Statement>;
```

**Syntax for Copying Table:**

```
Create Table <Table_Name> As <Select Statement>;
```

**Create a new Table emp1 from existing emp [Copying Table]:**

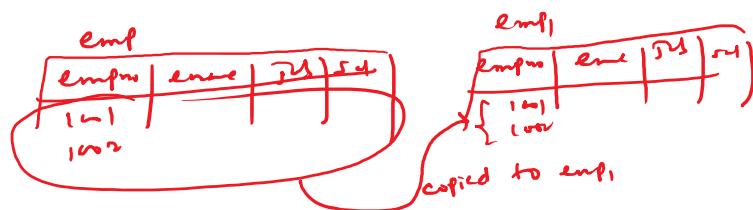
**Create Table emp1 as Select \* from emp;**



In above example, a new emp1 table created with emp table structure & records.

**Copying records from emp table to emp1:**

**Insert into emp1 Select \* from emp;**



In the above ex, emp table records will be copied to emp1

**Create a table from existing table without records [Copy Table Structure Only. Don't Copy Records]:**

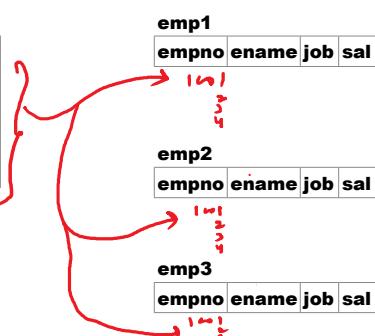
**Create Table emp1 as Select \* from emp where 1=2;**

**Note:**

In where clause, write any false condition to copy table structure.

**Insert Emp Table records into emp1,emp2 & emp3:**

Emp			
empno	ename	job	sal
1001	A	CLERK	10000
1002	B	MANAGER	15000
1003	C	MANAGER	18000
1004	D	CLERK	8000





```
Create Table emp1
As
Select empno,ename,job,sal from emp
where 1=2;
```

```
Create Table emp2
As
Select empno,ename,job,sal from emp
where 1=2;
```

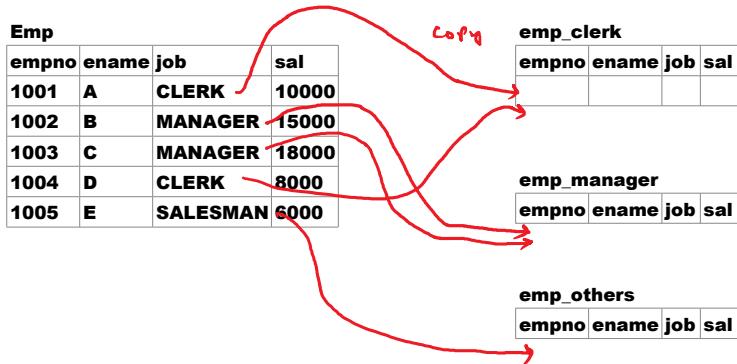
```
Create Table emp3
As
Select empno,ename,job,sal from emp
where 1=2;
```

**Insert All**

```
into emp1(empno,ename,job,sal) values(empno,ename,job,sal)
into emp2(empno,ename,job,sal) values(empno,ename,job,sal)
into emp3(empno,ename,job,sal) values(empno,ename,job,sal)
Select empno,ename,job,sal from emp;
```

#### Syntax of Conditional Insert All:

```
Insert All
When <condition-1> Then
    into <table-1>(<column_list>) values(<value_list>)
When <condition-2> Then
    into <table-2>(<column_list>) values(<value_list>)
.
.
[Else
    into .....]
Select Statement;
```



```

Insert All
When job='CLERK' Then
  into emp_clerk(empno,ename,job,sal) values(empno,ename,job,sal)
When job='MANAGER' Then
  into emp_manager(empno,ename,job,sal) values(empno,ename,job,sal)
Else
  into emp_others(empno,ename,job,sal) values(empno,ename,job,sal)
Select empno,ename,job,sal from emp;

```

#### **Assignment:**

**Copy emp table records into following tables:**

```

copy dept 10 employee records into dept10 table
copy dept 20 employee records into dept20 table
copy dept 30 employee records into dept30 table

```

**Copy the employees joined in 1980 into emp80 table**

**Copy the employees joined in 1981 into emp81 table**

**In case of other years copy into emp\_other table**

#### **Merge:**

**Merge = Update + Insert**

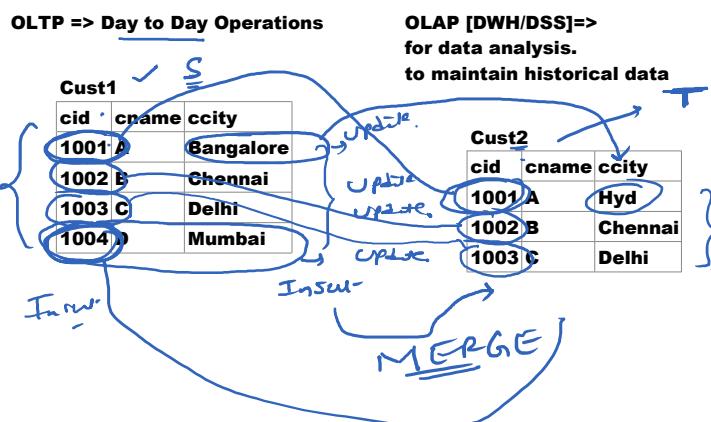
- Merge command is combination of Update & Insert Commands.
- It can be also called as "UPSERT" Command.

#### **Replication:**

**The process of making duplicate copies.**

#### **Replica:**

**Duplicate Copy is called "Replica".**



**Merge:**

- introduced in Oracle 9i
- Combination of Update & Insert Commands.
- It can be also called as "UPSERT" Command.
- Used to apply changes of one table to its replica [duplicate copy].
- It avoids of writing PL/SQL Program.

**Syntax:**

```
Merge into <target_table_name> <alias>
Using <source_table_name> <alias>
On(<condition>)
When Matched Then
    Update Statement
When Not Matched Then
    Insert Statement;
```

**OLTP**

<b>Cust1 S</b>		
<b>cid</b>	<b>cname</b>	<b>ccity</b>
1001	A	Hyd
1002	B	Chennai
1003	C	Delhi
1004	D	Mumbai

**OLAP [DWH]**

<b>Cust2 T</b>		
<b>cid</b>	<b>cname</b>	<b>ccity</b>
1001	A	Bangalore
1002	B	Chennai
1003	C	Delhi

```
Merge into Cust2 T
Using Cust1 S
On(S.cid=T.cid)
When Matched Then
    Update Set T.cname=S.cname, T.cc city=S.cc city
When Not Matched Then
    Insert values(S.cid,S.cname,S.cc city);
```

**SQL**

<b>DDL</b>	<b>DML</b>	<b>DQL/DRL</b>	<b>TCL</b>	<b>DCL / ACL</b>
Create	Insert	Select	Rollback	Grant
Alter	Update		Commit	Revoke
Drop	Delete		SavePoint	
Truncate	Insert All			
Rename	Merge			
Purge				
Flashback				

## Built-In Functions

Thursday, November 11, 2021 7:10 PM

### Built-In Functions:

- **Function is a set of statements that gets executed on calling.**
- **Function can take arguments.**
- **Function can return the result.**
- **Oracle developers defined some functions in Oracle Database. These Functions are called "Built-In Functions" (or) "Predefined Functions".**
- **Every function they defined to perform particular task.**

### Types of Built-In Functions:

- **String Functions [Text / Character Related Functions]**
- **Math Functions [Number / Numeric Functions]**
- **Conversion Functions**
- **Aggregate Functions [Group Functions]**
- **Date Functions**
- **Miscellaneous Functions**

### String Functions:

**upper()**

**lower()**

**initcap()**

**length()**

**concat()**

**substr()**

**instr()**

**ltrim()**

**rtrim()**

**trim()**

**lpad()**

**rpad()**

**replace()**

**translate()**

**soundex()**

**chr()**

**ascii()**  
**reverse()**

**upper():**  
**used to convert the string to upper case.**

**Syntax:**  
**upper(<string>)**

**Example:**  
**upper('raju') => RAJU**

**lower():**  
**used to convert the string to lower case.**

**Syntax:**  
**lower(<string>)**

**Example:**  
**lower('RAJU') => raju**

**initcap():**  
**used to get starting letter of every word  
as capital.**

**Syntax:**  
**initcap(<string>)**

**Raj Kumar**  
**Sachin Tendulkar**

**Steve**

**Example:**  
**initcap('RAJU') => Raju**  
**initcap('RAJ KUMAR') => Raj Kumar**

**length():**  
**used to get length of the string.**

**Syntax:**

**length(<string>)**

**Example:**

**length('SAI') => 3**

**length('RAJU') => 4**

**concat():**

**used to concatenate(combine) two strings.**

**Syntax:**

**concat(<string1>,<string2>)**

**Example:**

**concat('Raj','Kumar') => RajKumar**

**concat(concat('Raj',' ') , 'Kumar')**

Raj -

Raj Kumar

**concat('raj','kumar','varma') => ERROR**

**Select 'Raj' || ' ' || 'Kumar' from dual;**

**Raj Kumar**

**Display the emp names whose names are having 4 characters:**

**Select \* from emp where ename like '\_\_\_\_';**

**[or]**

**Select \* from emp where length(ename) = 4;**

**Display the emp names whose names are having more than 4 characters:**

**Select \* from emp  
where length(ename)>4;**

**Player**

<b>pid</b>	<b>Fname</b>	<b>Lname</b>
<b>1001</b>	<b>Sachin</b>	<b>Tendulkar</b>
<b>1002</b>	<b>Virat</b>	<b>Kohli</b>
<b>1003</b>	<b>Rohit</b>	<b>Sharma</b>
<b>1004</b>	<b>Rahul</b>	<b>Dravid</b>

**Create Table Player**

```
(  
pid number(4),  
fname varchar2(10),  
lname varchar2(10)  
);
```

**Display the player names whose name is having 12 chars:**

**Select \* from player  
where length(pname) = 12;**

**substr():**  
**used to get sub string from the string.**

**Syntax:**

**substr(<string>,<starting\_position>[,<number\_of\_chars>])**

1	2	3	4	5	6	7	8	9
r	a	j		k	u	m	a	r
-9	-8	-7	-6	-5	-4	-3	-2	-1

### Examples:

`substr('raj kumar',5) => kumar`

`substr('raj kumar',6) => umar`

`substr('raj kumar',1,3) => raj`

`substr('raj kumar',1,5) => raj k`

`substr('raj kumar',6,3) => uma`

`substr('raj kumar',-4) => umar`

`substr('raj kumar',-4,3) => uma`

**position value => +ve => From Left Side**

**position value => -ve => From Right Side**

**Display the emp record whose name is blake when we don't know the exact case:**

**Select \* from emp  
where lower(ename) = 'blake';**

lower(ename) = 'blake';  
'blake' = 'BlaKE' ✓      Blake ✓

**Generate email ids for employees by taking emp name's first 3 chars & empno's last 3 chars as username:**

Empno	Ename	Job	Sal	Mail_ID
7369	SMITH	CLERK		SMI369@nareshit.com

**Substr(Ename,1,3) || Substr(Empno,-3,3) || '@nareshit.com'**  
**SM1369@nareshit.com**

**Alter table emp add(mail\_id varchar2(50));**

**Update emp set  
mail\_id=Substr(ename,1,3) || Substr(empno,-3,3) ||  
'@nareshit.com';**

**Display the emp records whose name is  
started with 'S':**

**Select \* from emp  
where ename like 'S%';**

**[or]**

**Select \* from emp  
where Substr(ename,1,1) = 'S';**

**Select \* from emp  
where substr(ename,1,2)='Sc';**

**Display the emp records whose names are  
ended with RD:**

**Select \* from emp**

**where substr(ename,-2,2)='rd';**

### Instr():

This function is used to know the position of sub string.

#### Syntax:

**Instr(<String>,<Sub\_String>[,<start\_position>,<occurrence>])**

#### Examples:

**default**

**start\_position => 1**

**occurrence => 1**

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
T	h	i	s		i	s		h	i	s		w	i	s	h

---

-----

-----

→ Instr('This is his wish','is') => 3

Instr('This is his wish','is',4) => 6

Instr('This is his wish','is',7) => 10

Instr('This is his wish','is',11) => 14

Instr('This is his wish','is',15) => 0

**When substring is not found this function returns 0.**

**When substring is found it returns substring position.**

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
T	h	i	s		i	s		h	i	s		w	i	s	h
-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Instr('This is his wish','is',-1,3) => 6

Instr('This is his wish','is',-4,2) => 6

Instr('This is his wish','is',-8,2) => 3

**Instr('This is his wish','is',1,2) => 6  
Instr('This is his wish','is',1,4) => 14  
Instr('This is his wish','is',4,2) => 10**

**Instr('This is his wish','is',-1,1) => 14**

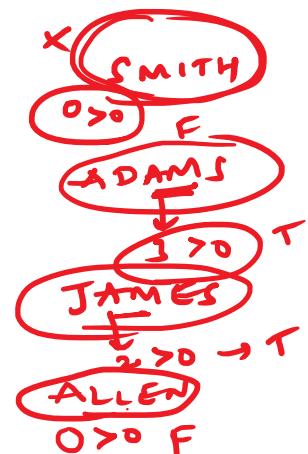
**Display the emp records whose names are having 'AM' chars:**

**Select \* from emp  
where ename like '%AM%';**

**[or]**

**Select \* from emp  
where Instr(ename,'am')>0**

**ADAMS ✓  
JAMES ✓**



**Trim(), Ltrim(), Rtrim():**

**Ltrim():**

**used to remove specified char set from left side.**

**Default character is ' ' [space].**

**It is used to remove unwanted chars from left side.**

**Syntax:**

**Ltrim(<String>[,<char\_set>])**

**Example:**

**Ltrim(' sai') => sai**

**Example:**

**Ltrim(' sai') => sai**

**Ltrim('@@@@@raju@@@','@') => raju@@@@**

**Rtrim()**

**used to remove unwanted chars from right side**

**Syntax:**

**Ltrim(<String>[,<char\_set>])**

**Rtrim(' sai ') => ...sai**

**Rtrim('@@@@@raju@@@','@') => @@@@raju**

**..**

**Trim():**

**used to remove unwanted chars from left side or right side or both sides.**

**Syntax:**

**Trim(Leading/Trailing/Both <char> from <string>)**

**Trim(Leading '@' from '@@@raju@@@')**

**Product**

**Mobile\_Model**

**-----**

**SAMSUNG A71**

**Ltrim(Mobile\_Model,'SAMSUNG ')**

**SAMSUNG A51**

**A71**

**SAMSUNG A21**

**A51**

**SAMSUNG M12**

**A21**

**M12**

**mail\_id**

-----  
**abc@nareshit.com**  
**def@nareshit.com**

**Rtrim(mail\_id,'@nareshit.com')**

**lpad(), rpad():**

**pad => Fill**

**is used to fill specified char set from left side.**

**Default char is space [' ']**

**Syntax:**

**lpad(<string>,<size>[,<char\_Set>])**

**Ex:**

**lpad('raju',10,'\*') => \*\*\*\*\*raju**

**lpad('sai',10,'@') => @@@@@@@@sai**

**lpad('raju',10,'#@') => #@#@#@raju**

**lpad('raju',10) => 6 spacesraju**

**rpad():**

**used to fill specified char set from right side.**

**Default char is space [' '].**

**Syntax:**

**rpad(<string>,<size>[,<char\_Set>])**

**rpad('ravi',12,'\*') => ravi\*\*\*\*\***

**rpad('ravi',10) => ravi6spaces**

**rpad('ravi',10,'#\$') => ravi##\$\$#**

**LPAD('a',10,'\*') => \*\*\*\*\*a**

**LPAD('\*',10,'\*') => \*\*\*\*\***

**LPAD('\*',6,'\*') => \*\*\*\*\***

**1234567890**

**Amount debited from XXXXXX7890**

**Select 'amount debited from ' || Lpad('X',6,'X') ||  
Substr('1234567890',-4,4) from dual;**

**OTP Sent to mobile number XXXXXXXX123**

**Replace():**

**used to replace search string with replace string.**

**Syntax:**

**Replace(<string>,<search\_string>,<replace\_string>)**

**Ex:**

**Replace('ravi teja','teja','kumar') => ravi kumar**

**Replace('abcpqrabcbaaabcbbaabbcc','abc','XYZ')**

**XYZpqrXYZaaXYZbbaabbcc**

**Translate():**

**used to translate corresponding chars with specified chars**

**Transalte('abcpqrabcbaaabcbbaabbcc','abc','XYZ')**  
**XYZpqrXYZXXXXYZYYYYXXYYZZ**

**Replace('abcpqrabcbaaabcbbaabbcc','abc','XYZ')**  
**XYZpqrXYZaaXYZbbaabbcc**

**To provide security for sensitive data we can use translate() Function.**

**Select**

sal	s ↴	0 => ) 1 => ! 2 => @ 3 => # 4 => \$ 5 => % 6 => ^ 7 => & 8 => * 9 => (
5500	%.%.))	0 => )
7200	&(2))	1 => !
3780	#(8*)	2 => @

**Replace salary values with mask characters:**

**Select empno,ename,  
Translate(sal, '0123456789', '!@#\$%^&\*' )  
from emp;**

**ASCII():**

**used to know ASCII value of specified**

**char.**

**Syntax:**

**ASCII(<char>)**

**ASCII('A') => 65**

**ASCII('a') => 97**

**ASCII('Z') => 90**

**ASCII('z') => 122**

**Chr():**

**is used to get character of specified ascii code.**

**Syntax:**

**Chr(<Ascii\_code>)**

**Ex:**

**Chr(65) => A**

**Chr(97) => a**

**Soundex():**

**used to retrieve the records based on pronunciation.**

**Syntax:**

**Soundex(<String>)**

**Ex:**

**Select \* from emp**

**where Soundex(ename) = Soundex('Skat');**

**Select \* from emp**

**where Soundex(ename) = Soundex('SMYT');**

**Reverse():**  
**is used to reverse the string.**

**Syntax:**  
**Reverse(<string>)**

**Reverse('ramu') => umar**

**Select ename, reverse(ename) from emp;**

### **Aggregate [Group] Functions:**

**sum()**  
**avg()**  
**max()**  
**min()**  
**count()**

**sum():**  
**used to find sum of values of a column**

**Syntax:**  
**sum(<column\_name>)**

**emp**

empno	ename	job	sal	deptno
1001	A	CLERK	5000	10
1002	B	MANAGER	8000	10
1003	C	CLERK	4000	20
1004	D	MANAGER	10000	20
1005	E	CLERK	8000	30

### **Examples:**

**find sum of salaries of all employees:**  
**Select Sum(sal) from emp;**

**35000**

**find sum of salaries of deptno 20:**

**Select Sum(Sal) from emp  
where deptno=20;**

**find sum of salaries of all clerks:**

**Select Sum(Sal) from emp  
where job='CLERK';**

**Avg():**

**is used to find average value.**

**Syntax:**

**Avg(<Column\_Name>)**

**Ex: Avg(Sal)**

**Find Average salary of all employees:**

**Select avg(sal) from emp;**

**Find Average salary of Deptno 20 employees:**

**Select avg(sal) from emp where deptno=20;**

**Find Average salary of all clerks:**

**Select avg(sal) from emp where job='CLERK';**

**max():**

**This function is used to find maximum value in a set of numbers.**

**Syntax:**

**max(<column\_name>)**

**Ex: max(Sal)**

**Find max salary in all employees:**

**Select max(Sal) from emp;**

**Find max salary in managers:**

**Select max(Sal) from emp  
where job='MANAGER';**

**Find max sal in deptno 10:  
Select max(Sal) from emp  
where deptno=10;**

**Find the emp name who is getting max salary:**

**Sub Queries / Nested Queries:**

- Writing query in another query is called "Sub Query" (or) "Nested Query".
- Sub query must be written parenthesis.

**Select ename from emp  
where sal= (Select max(Sal) from emp);**

**First inner query gets executed.  
The result of inner query becomes input for outer query**

**Display the emp name who is earning max sal in managers:**

**Select ename from emp  
where sal= (Select max(Sal) from emp  
where job='MANAGER');**

**Find second maximum salary:**

<b>sal</b>	<b>Select max(Sal) from emp =&gt; 5000</b>
<b>-----</b>	
<b>3000</b>	
<b>4500</b>	<b>where sal&lt;5000</b>
<b>4000</b>	
<b>3500</b>	<b>3000</b>
<b>5000</b>	<b>4500</b>

<b>3500</b>	<b>3000</b>
<b>5000</b>	<b>4500</b>
<b>1800</b>	<b>4000</b>
<b>1500</b>	<b>3500</b>
	<b>1800</b>
	<b>1500</b>

**Select max(sal) from emp  
where sal<(Select max(sal) from emp);**

**Display the emp name who is earning  
second max salary:**

**Select ename from emp  
where sal=(Select max(sal) from emp  
where sal<(Select max(sal) from emp));**

**Find third max salary**

**Find the emp name who is earning third  
max salary**

**Min():**

**used to find minimum value in a set of  
numbers.**

**Syntax:**

**Min(<column\_name>)**

**Ex: Min(Sal)**

**Find minimum sal in all employees:  
Select min(Sal) from emp;**

**Find minimum salary in clerks:  
Select min(sal) from emp**

**where job='CLERK';**

**Find min sal in deptno 30:**

**Select min(sal) from emp  
where deptno=30;**

**Display the emp name who is earning min sal:**

**Select ename from emp  
where sal=(Select min(Sal) from emp);**

**Find Second Minimum salary:**

<b>sal</b>	<b>Select min(sal) from emp where sal&gt;(Select min(sal) from emp);</b>
-----	
5000	
6000	
4000	
4500	
3000	

**Display the emp name who is earning second min sal:**

**Select ename from emp  
where sal=(Select min(sal) from emp  
where sal>(Select min(sal) from emp));**

**Count():**

**It is used to count number of records.**

**Syntax:**

**Count(<column\_name>) / Count(\*)**

**Ex: Count(ename) Count(Comm) Count(\*)**

**Count(Sal) =>**

**It counts salary values.**

**It will not count null values.**

**Count(\*) =>**  
**It counts null values.**

**Find number of employees who are getting commission:**

**Select Count(Comm) from emp;**

**Find number of records in emp table:**

**Select Count(\*) from emp;**

**Find number of managers:**

**Select Count(\*) from emp  
where job='MANAGER';**

**Find number of employees in deptno 30:**

**Select Count(\*) from emp  
where deptno=30;**

**Numeric [Math] Functions:**

**power()**  
**sqrt()**  
**sign()**  
**mod()**  
**log()**  
**ln()**

**sin()**  
**cos()**  
**tan()**

**ceil()**  
**floor()**

**trunc()**

**round()**

**abs()**

**Power():**

**is used to find power values.**

**Syntax:**

**Power(Number, Power)**

**Ex:**

$2^3 \rightarrow \text{power}(2,3)$

**Sqrt():**

**is used to find square root value**

**Syntax:**

**Sqrt(Number)**

**Ex:**

$\sqrt{100} \rightarrow \text{Sqrt}(100)$

**Sign():**

**is used to find whether the number is +ve or -ve or zero.**

**Syntax:**

**Sign(number)**

**Ex: Sign(5) => 1 => +ve**

**Sign(-5) => -1 => -ve**

**Sign(0) => 0 => ZERO**

**Mod():**

**It is used to get remainder value.**

**Syntax:**

**Mod(Number, divisor)**

**Ex: Mod(5,3) => 2 [Remainder]**

**sin():**

**used to find sine values**

**specify angle in the form of radians.**

**Syntax:**

**sin(angle)**

**Ex: Sin(90\*3.14/180) => 1**

**Cos() => used to find Cosine values**

**Tan() => Used to find tangent values**

**Cos(0\*3.14/180) => 1**

**Tan(45\*3.14/180) => 1**

**log(): is used to find logarithmic values**

**In(): is used to find natural logarithmic value**

$$\log_{10} 10 \rightarrow \begin{array}{l} \log(10,10) \\ \ln(10) \end{array}$$

**Ceil():**

**is used to get nearest upper integer value.**

**Syntax:**

**Ceil(<Number>)**

**Example:**

**Ceil(123.4567) => 124**

**Ceil(6789.7432) => 6790**

**Floor():**

**is used to get nearest lower integer value.**

**Syntax:**

**Floor(<Number>)**

**Example:**

**Floor(123.4567) => 123**

**Floor(5678.1234) => 5678**

**Trunc():**

**is used to remove the decimal places.**

**Syntax:**

**Trunc(<Number>[,<Number\_of\_Decimal\_places>])**

**Example:**

**Trunc(123.45678) => 123**

**Trunc(123.45678,2) => 123.45**

**Trunc(4567.1234567,3) => 4567.123**

**-1 => It rounds in 10s**

**-2 => rounds in 100s**

**-3 => rounds in 1000s**

<b>Trunc(123.4567,-1) =&gt; 120</b>	<b>[120 to 130]</b>
<b>Trunc(123.4567,-2) =&gt; 100</b>	<b>[100 to 200]</b>

<b>Trunc(5674.1234,-1) =&gt; 5670</b>	<b>5670 to 5680</b>
<b>Trunc(5674.1234,-2) =&gt; 5600</b>	<b>5600 to 5700</b>
<b>Trunc(5674.1234,-3) =&gt; 5000</b>	<b>5000 to 6000</b>

<b>Trunc(456.7892,-1) =&gt; 450</b>	<b>450 to 460</b>
<b>Trunc(456.7892,-2) =&gt; 400</b>	<b>400 to 500</b>

**Note: Trunc function always takes lower value**

**Round():**

**is used to get rounded values.**

**.5 or >.5 => takes upper**

**<.5 => takes lower**

**Syntax:**

**Round(<Number>[,Number\_of\_decimal\_places])**

**Examples**

**Round(123.4567) => 123**

**Round(123.6789) => 124**

**Round(123.56789) => 124**

**Round(123.45678,2) => 123.46**

**Round(123.45478,2) => 123.45**

**Round(456.56789) => 457**

**Round(456.34567) => 456**

**Round(456.56789,2) => 456.57**

**Round(456.56498,2) => 456.56**

**Round(123.4567,-1) => 120 to 130**

120 < 125 >= 130

**Round( 153.4567,-1) => 150**

150 ————— 160  
155

**Round( 153.9492,-1) => 160**

150 ————— 160  
155

**Round(123.4567,-2) => 100**

100 to 200  
avg => 150

**Round(175.7865,-2) => 200**

100 to 200  
avg => 150

**abs():**

**it is used to get absolute value.**

**Syntax:**

**abs(<number>)**

**abs(5) => 5**

**abs(-5) => 5**

**Conversion Functions:**

**Select 100+200 from dual;**

**300**

**Select '100'+'200' from dual;**

**300**

**Select 100+'200' from dual;**

**300**

**Select \* from emp where empno=7499;**

**displays 7499 record**

**Select \* from emp where empno='7499';**

**displays 7499 record**

**There are 2 types of conversions.**

**They are:**

**Implicit Conversion**

**Explicit Conversion**

**Implicit Conversion:**

**Oracle performs this implicit conversion implicitly.**

**Select '100'+'200' from dual;**

**300**

**Select 100+'200' from dual;**

**300**

**Select \* from emp where empno=7499;**

**displays 7499 record**

**Select \* from emp where empno='7499';**

**displays 7499 record**

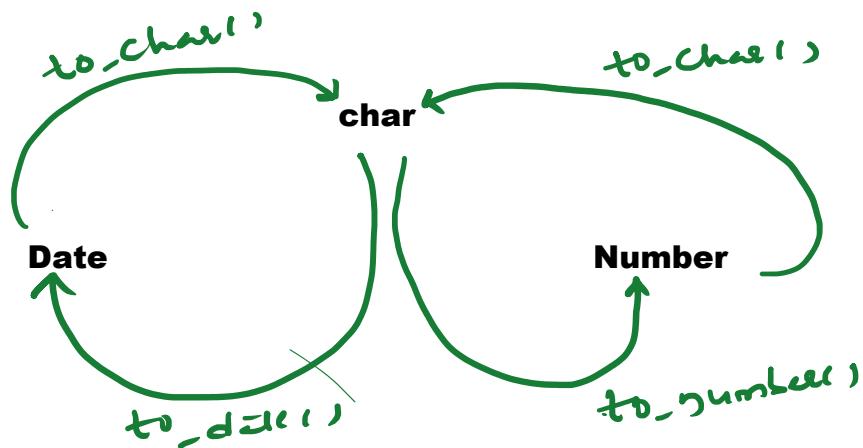
**Insert into emp(empno,ename,hiredate)  
values(1001,'A','12-Jan-2020');**

**Explicit Conversion:**

**We perform conversion explicitly using  
built-in functions.**

**For Explicit Conversion oracle provides following functions:**

- **to\_char()**
- **to\_date()**
- **to\_number()**
- **to\_timestamp()**



#### **to\_char() [date to string]:**

- **It is used to convert date value to string.**
- **Using this function we can change the date formats like mm/dd/yyyy or dd/mm/yyyy**

#### **Syntax:**

**to\_char(<Date>,<Format>)**

<b>Format</b>	<b>Example / PURPOSE</b>
<b>yyyy</b>	<b>2021</b>
<b>yy</b>	<b>021</b>
<b>y</b>	<b>21</b>
<b>year</b>	<b>Twenty Twenty-One</b>

<b>mm</b>	<b>11</b>
<b>mon / MON</b>	<b>nov NOV</b>
<b>month / MONTH</b>	<b>november NOVEMBER</b>
<b>dd</b>	<b>It gives day number in the month 18</b>
<b>ddd</b>	<b>It gives day number in the year 322</b>
<b>d</b>	<b>It gives day number in the week 1 - sun 2 - mon .. 7 - sat</b>
<b>dy</b>	<b>It gives short weekday name sun mon thu</b>
<b>day</b>	<b>sunday monday</b>
<b>q</b>	<b>quarter number jan-mar =&gt; 1st quarter apr-jun =&gt; 2nd quarter jul-sep =&gt; 3rd quarter oct-dec =&gt; 4rd quarter</b>
<b>cc</b>	<b>Century Number 21st century is current century</b>
<b>w</b>	<b>week number in the month</b>
<b>ww</b>	<b>week number in the year</b>
<b>AD / BC</b>	<b>AD / BC</b>
<b>HH / HH12</b>	<b>Gives hours part in 12 hours format</b>
<b>HH24</b>	<b>Gives Hour part in 24 hours format</b>
<b>AM / PM</b>	<b>AM / PM</b>
<b>MI</b>	<b>Minutes</b>
<b>SS</b>	<b>Seconds</b>

**Display year 4 digits from today's date:**

**Select to\_char(sysdate,'yyyy') from dual;**  
**2021**

**Select to\_char(sysdate,'yyy') from dual;**  
**021**

**Select to\_char(sysdate,'yy') from dual;**  
**21**

**Select to\_char(sysdate,'y') from dual;**

**1**

**Select to\_char(sysdate,'year') from dual;**

**Twenty Twenty-One**

**Display current month number:**

**Select to\_char(sysdate,'mm') from dual;**

**11**

**Select to\_char(sysdate,'mon') from dual;**

**nov**

**Select to\_char(sysdate,'MON') from dual;**

**NOV**

**Select to\_char(sysdate,'month') from dual;**

**november**

**Select to\_char(sysdate,'MONTH') from dual;**

**NOVEMBER**

**Display day part from today's date:**

**Select to\_char(sysdate,'dd') from dual;**

**18**

**Select to\_char(sysdate,'ddd') from dual;**

**322**

**Select to\_char(sysdate,'d') from dual;**

**5**

**Select to\_char(sysdate,'dy') from dual;**

**thu**

**Select to\_char(sysdate,'DY') from dual;**  
**THU**

**Select to\_char(sysdate,'day') from dual;**  
**thursday**

**Select to\_char(sysdate,'DAY') from dual;**  
**THURSDAY**

**Find today's date falls under which quarter:**

**Select to\_char(sysdate,'q') from dual;**

**Find to\_day's date falls under which century:**

**Select to\_char(sysdate,'cc') from dual;**

**Display current system time:**

**12 hours format:**

**Select to\_char(sysdate,'HH:MI AM') from dual;**

**24 hours format:**

**Select to\_char(sysdate,'HH24:MI') from dual;**

**Display the emp records who joined in 1982:**

**Select \* from emp  
where to\_char(hiredate,'yyyy')=1982;**

**Display the emp records who joined in 1980,1982,1983:**

**Select \* from emp  
where to\_char(hiredate) in(1980,1982,1983);**

**Display the emp records who joined in december month:**

**Select \* from emp**

**where to\_Char(hiredate,'mm')=12;**

**Display the emp records who joined in Jan, Jun, Dec:**

**Select \* from emp  
where to\_char(hiredate,'mm') in(1,6,12);**

**Display the emp records who joined in 4th quarter:**

**Select \* from emp  
where to\_char(hiredate,'q')=4;**

**Display the emp records who joined in 1,3,4 quarters:**

**Select \* from emp  
where to\_char(hiredate,'q') in(1,3,4);**

**Display the emp records who joined in current century:**

**Select \* from emp  
where to\_char(hiredate,'cc')=21;**

**Display the emp records who joined on Sunday:**

**d dy day  
1 sun Sunday**

**Select \* from emp  
where to\_char(hiredate,'d')=1;**

**or**

**Select \* from emp  
where to\_char(hiredate,'dy')='sun';**

**or**

**Select \* from emp  
where rtrim(to\_char(hiredate,'day'))='sunday';**

**Display the hiredate in US Date Format [mm/dd/yyyy]:**

**Select empno,ename, to\_char(hiredate,'mm/dd/yyyy')  
hiredate from emp;**

**Display hiredate in India Date Format [dd/mm/yyyy]:**

**Select empno,ename,  
to\_char(hiredate,'dd/mm/yyyy') hiredate  
from emp;**

**to\_char() [number to string]:**

- using this function we can convert number to string.**
- Using this we can apply different currency formats.**

**Syntax:**

**to\_char(<number>[,<format>])**

**Select to\_char(123) from dual;**

**123 => '123'**

**123 => string**

**Select to\_char(123.45) from dual;**

**123.45 => string**

<b>Format purpose</b>	
<b>L</b>	<b>Currency Symbol</b>
<b>C</b>	<b>Currency Name</b>
,	<b>Thousand separator</b>
.	<b>Decimal Point</b>
<b>9</b>	<b>Digits</b>

**5000 => \$5000.00**

**Number => String**

**to\_char(5000,'L9999.99') => \$5000.00**

**to\_char(5000,'C9,999.99') => USD5,000.00**

**Display sal column of emp table with currency symbol:**

**sal number(7,2)**

**number of 9s will be decided by field size**

**Select empno,ename,  
to\_char(sal,'L99999.99') salary  
from emp;**

**Parameters:**

**NLS\_TERRITORY = 'AMERICA'  
NLS\_CURRENCY = '\$'**

**Session/System**

**Alter Session => For current session only it is applicable  
Alter System => Permanently it is applicable**

**Log in as DBA:**

**conn system/nareshit**

**show parameters**

**Alter Session Set  
NLS\_TERRITORY='United Kingdom';**

## Alter Session Set **NLS\_CURRENCY='£';**

## **ALTER Session Set NLS\_Territory = 'INDIA';**

**Alter Session Set  
NLS\_Currency='RS';**

**to\_date():**  
**used to convert string to date.**

## Syntax:

**to\_date(<string>[,<format>])**

## **Example:**

**to\_Date('17-oct-2020')**  
**17-oct-20 => Date**

**'17-oct-2020' => String**

**to\_date('17 october 2020')**  
**17-oct-20 => Date**

**to\_date('17/12/2020') => ERROR**

**to\_date('17/12/2020','dd/mm/yyyy')**

**to\_char(sysdate,'yyyy')**

**to\_char('23-dec-2020','yyyy') => ERROR**

**to\_char(to\_date('23-dec-2020'),'yyyy')**

**to\_char(to\_date('23-dec-2020'),'mm')**

**Display the week day of the date when  
india got independence:**

**to\_char('15-AUG-1947','dy') => ERROR**

d => 1  
dy => sun  
day => sunday

**to\_char(to\_date('15-AUG-1947'),'dy') => fri**

**Find week day of sachin's birth date:**

**24 april 1973**

**Select to\_char(to\_date('24-apr-1973'),'day')  
from dual;**

**to\_number():**

**is used to convert string to number.  
This string must be numeric string.**

**Syntax:**

**to\_number(<string>[,<format>])**

**to\_number('123') => 123**

**to\_number('123.45') => 123.45**

**to\_number('\$5000.00') => ERROR**

**to\_number('\$5000.00','L9999.99') => 5000**

**to\_number('USD5,500.00','C9,999.99') => 5500**

**to\_timestamp():**  
**is used to convert string to timestamp value.**

**to\_timestamp('23-nov-2020') => 23-NOV-20 12.00.00.000000000 AM**

**to\_timestamp('23-nov-2020 10:30:50.505') =>  
23-NOV-20 10.30.50.505000000 AM**

**to\_char() [date to string]  
to\_char() [number to string]  
to\_Date() [string to date]  
to\_number() [string to number]  
to\_timestamp() [string to timestamp]**

### **Date Functions:**

**sysdate  
systimestamp  
Add\_Months()  
Last\_Day()  
Next\_Day()  
Months\_Between()**

**sysdate:  
used to get current system date.**

**Ex: Select sysdate from dual;**

**systimestamp:**

**used to get current system date and time**

**Ex: Select systimestamp from dual;**

**Add\_Months():**

**used to add or subtract months from a date**

**Syntax:**

**Add\_Months(<date>,<number\_of\_months>)**

**Examples:**

**After 2 months what will be the date from sysdate:**

**Add\_Months(sysdate,2) => It adds 2 months**

**20-JAN-22**

**Display 2 months ago date from sysdate:**

**Add\_Months(sysdate,-2) => It subtracts 2 months**

**20-SEP-21**

**Display 2 months ago from specific date:**

**Add\_Months('25-AUG-2020',-2) => 25-JUN-20**

**Add 2 months to specific date:**

**Add\_Months('25-AUG-2020',2) => 25-OCT-20**

**Display the emp records who joined today:**

**Select \* from emp  
where trunc(hiredate) = trunc(sysdate);**

**use trunc() function to truncate the time value.**

**Display the emp records who joined yesterday:**

**Select \* from emp  
where trunc(hiredate) = trunc(sysdate-1);**

**Display the emp records who joined 1 month ago:**

**Select \* from emp  
where trunc(hiredate) =  
trunc(add\_months(sysdate,-1));**

**Display the emp records who joined 1 year ago:**

**Select \* from emp  
where trunc(hiredate) =  
trunc(add\_months(sysdate,-12));**

**Sales**

<b>dateid</b>	<b>Amount</b>
<b>20-NOV-2021</b>	<b>15000</b>
<b>19-NOV-2021</b>	<b>12000</b>
<b>20-OCT-2021</b>	<b>18000</b>
<b>20-NOV-2022</b>	<b>16000</b>

**display today's sales:**

**Select \* from sales  
where trunc(dateid) = trunc(sysdate);**

**display yesterday sales:**

**Select \* from sales  
where trunc(dateid) = trunc(sysdate-1);**

**display 1 month ago sales:**

**Select \* from sales  
where trunc(dateid) = trunc(Add\_Months(sysdate,-1));**

**display 1 year ago sales:**

**Select \* from sales  
where trunc(dateid) = trunc(Add\_Months(sysdate,-12));**

**Calculate Date of Retirement of employees. 60 years age  
is the retirement date:**

**Emp**

**DOB => Date\_OF\_Birth  
DOR => Date\_of\_Retirement**

<b>Empno</b>	<b>ename</b>	<b>DOB</b>	<b>DOR</b>
<b>1001</b>	<b>Ravi</b>	<b>23-OCT-2000</b>	<b>23-OCT-2060</b>
<b>1002</b>			

**Update Emp**

**Set DOR = Add\_Months(DOB,60\*12);**

**INDIA\_CMS**

<b>STATE</b>	<b>CM_NAME</b>	<b>term_start_date</b>	<b>term_end_date</b>
<b>TS</b>	<b>KCR</b>	<b>15-NOV-2018</b>	
<b>AP</b>	<b>YS JAGAN</b>	<b>23-MAY-2019</b>	

**Update INDIA\_CMS**

**Set term\_end\_date = Add\_Months(term\_start\_date,5\*12);**

**Last\_Day():**

**This function is used to get last date in  
the month.**

**Syntax:**

**Last\_day(<date>)**

**Ex:**

**Last\_Day(sysdate) => 30-NOV-21**

**Last\_Day('17-OCT-21') => 31-OCT-21**

**Find first date in the next month:**

**Select Last\_day(sysdate)+1 from dual;**

**Find next month last date:**

**Last\_Day(Add\_Months(sysdate,1)) => 31-DEC-21**

**Find last date in the previous month:**

**Last\_Day(Add\_Months(sysdate,-1)) => 31-OCT-21**

**Display current month first date**

**Display previous month first date**

**Next\_Day():**

**it is used to get coming date of specified week day**

**Syntax:**

**Next\_Day(<date>,<weekday>)**

**Ex:**

**Find coming Sunday date:**

**Select Next\_Day(sysdate,'sun') from dual;**

**21-NOV-21**

**Display next month first Sunday date:**

**Select Next\_Day(Last\_day(sysdate)+1,'sun') from dual;**

**Display current month last Sunday date:**

**Select Next\_Day(Last\_Day(sysdate)-7,'sun') from dual;**

**Months\_Between():**

**used to find difference between two dates.**

**Syntax:**

**Months\_Between(<date1>,<date2>)**

**Calculate experience of employees:**

**Select empno,ename,hiredate,  
Trunc(Months\_Between(sysdate,hiredate)/12) exp  
from emp;**

**Calculate age:**

**Select Months\_Between(sysdate,'24-APR-1973')/12  
from dual;**

## **Miscellaneous Functions:**

**User**  
**UID**  
**Greatest()**  
**Least()**  
**NVL()**  
**NVL2()**  
**Rank()**  
**Dense\_Rank()**

### **User:**

**used to get current user name**

#### **Ex:**

**to see current user name:**

**Select User from dual;**  
**[or]**  
**show user;**

**to see all users information:**

**Select username,user\_id from all\_users;**

### **UID:**

**this function is used to get user id.**

#### **Ex:**

**Select uid from dual;**

### **Greatest():**

**used to find maximum value in specified set of numbers. It can be take any number of arguments. Where as max() function can take only one argument.**

**Greatest function is used to find max value in row where as max() function is used to find max value in a column.**

### **student**

sid	sname	M1	M2	M3	
1001	A	40	80	70	→ Greatest()
1002	B	90	50	85	→ max()

**Select Max(M3) from student; --return s m3 sub highest marks**

**Select Greatest(M1,M2,M3) from student; --returns**

**80  
90**

**Select greatest(800,1000,900,400,600) from dual;**

**1000**

### **Least():**

**used to find least value in specified set of values.**

**Using this we can find minimum value in a row.**

**Where as min() function is used to find minimum value in a column.**

**We can pass any number of arguments to least() function where as we pass argument to min() function.**

### **Ex:**

**Select Least(M1,M2,M3) from student;**

**Select Least(100,50,90,20,30) from dual;**

**20**

**NVL():**

**Something+null = null**

**Select sal+comm total\_sal from emp;**

**Syntax:**

**NVL(<first>,<second>)**

**first argument is null, it returns second argument.  
first argument is not null, it returns first argument.**

**NVL(500,1000) => 500**

**NVL(NULL,1000) => 1000**

**This function is used to replace null values with other values.**

**Calculate Total Salary:**

**Select empno,ename,  
Sal+NVL(Comm,0) as "tot sal" from emp;**

**Display the comm value as N/A for the employees who are not getting commission:**

**comm**

**-----**

**N/A**

**500**

**300**

**N/A**

**Select empno,ename,**

**NVL(comm,'N/A') as "comm" from emp; //ERROR**

**Select empno,ename,  
NVL(to\_char(comm),'N/A') as "comm" from emp;**

### **Result**

**Replace null values with 'AB'**

<b>Sid</b>	<b>M1</b>
<b>1001</b>	<b>50</b>
<b>1002</b>	
<b>1003</b>	<b>70</b>
<b>1004</b>	
<b>1005</b>	
<b>1006</b>	<b>90</b>

**Select sid,  
NVL(to\_char(M1),'AB') from student;**

### **NVL2():**

**This function is used to replace null and not null values.**

#### **Syntax:**

**NVL2(<first>,<second>,<third>)**

#### **Ex:**

**NVL2(100,200,300) => if first arg not null, returns second arg  
NVL2(null,200,300) => if first arg null, it returns third arg.**

**Set comm as RS 500 to the employees  
who are not getting commission. Increase  
RS 200 commission to the employees who  
are getting commission:**

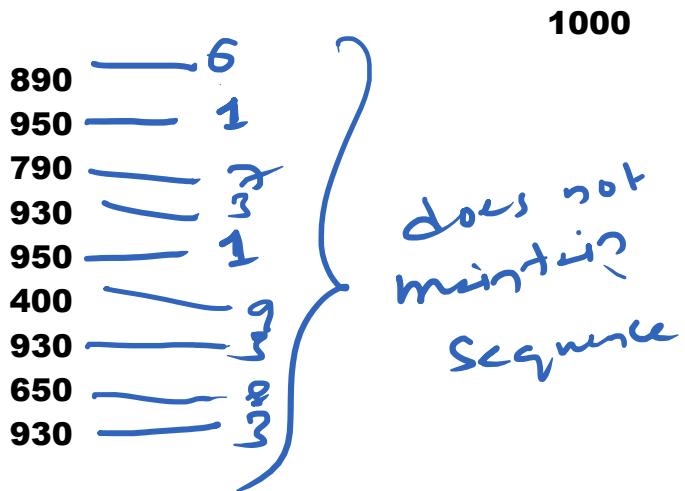
```

Select
NVL2(comm,comm+200,500)
from emp;

```

### **Rank():**

**is used to apply ranks to the column values.**

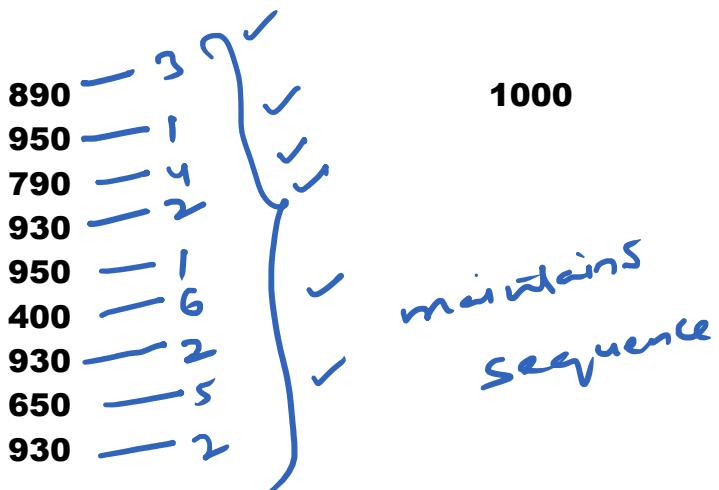


### **Syntax:**

**Rank() Over(order by <column> Asc/Desc)**

### **Dense\_Rank():**

**is used to apply ranks to the column values.**



### **Syntax:**

**Dense\_Rank() Over(order by <column> Asc/Desc)**

**Display the emp records with ranks. Give first rank to highest salary.**

**Select empno,ename,sal,  
Rank() Over(order by sal Desc) as "Rank"  
from emp;**

**Select empno,ename,sal,  
Dense\_Rank() Over(order by sal Desc) as "Rank"  
from emp;**

**Display the emp records with ranks. Give first rank to highest salary. If salary is same give ranking according highest experience.**

**Select empno,ename,sal,hiredate,  
Dense\_Rank() Over(order by sal desc, hiredate Asc) as "Rank"  
from emp;**

## **Interval Expressions**

Tuesday, November 23, 2021 6:19 PM

### **Interval Expressions:**

- are introduced in Oracle 9i.
- are used to add/subtract days, months or years to a date or from a date.
- are also used to add/subtract hours or minutes to a time or from a time.

### **Adding 2 days to sysdate:**

**Select sysdate+interval '2' day from dual;**  
**25-NOV-21**

### **Adding 2 months to sysdate:**

**Select sysdate+interval '2' MONTH from dual;**

### **Adding 2 years to sysdate:**

**Select sysdate+interval '2' year from dual;**

### **Adding 1year 6 months to sysdate:**

**Select sysdate+interval '1-6' Year to Month from dual;**

### **Adding 2 days to the date 20-DEC-19:**

**Select to\_date('20-DEC-2019')+interval '2' day from dual;**

### **display the emp records who joined yesterday:**

**Select \* from emp  
where trunc(hiredate) = trunc(sysdate-interval '1' day);**

### **display the emp records who joined 1 year ago on same day:**

**Select \* from emp  
where trunc(hiredate) = trunc(sysdate-interval '1' year');**

### **Adding 2 hours to system time:**

**Select systimestamp+interval '2' HOUR  
from dual;**

**Adding 2 minutes to system time:**

**Select systimestamp+interval '2' MINUTE  
from dual;**

**Adding 1hour 30 minutes to system time:**

**Select systimestamp+interval '1:30' Hour  
to Minute from dual;**

### **Flashback Query:**

- **introduced in Oracle 10g.**
- **After commit we cannot use rollback. If we delete some records accidentally from the table and if it is committed, we cannot rollback it.**  
**In such type of situations use "FLASHBACK QUERY".**
- **it is used to recollect the past data which was existed some time ago.**
- **"AS OF " Clause is used to write Flashback Query.**

**Delete from emp;  
COMMIT;**

**Rollback;**

### **Example:**

**Delete from emp;  
COMMIT;**

**Rollback;**

### **Flashback Query:**

**Select \* from emp AS OF TIMESTAMP sysdate-interval '5'**

**MINUTE;**

**Above query displays 5 minutes ago data.**

**Insert into emp**

```
Select * from emp AS OF TIMESTAMP  
sysdate-interval '5' MINUTE;
```

**Above query recollects 5 minutes ago data in emp table.**

**undo\_retention => 900 seconds [15 MINUTES]**

**By default upto 15 minutes we can recollect the data using  
FLASHBACK QUERY.**

**We can change this default value by writing following  
command:**

**login as DBA**

**Alter System set undo\_retention = 3600; [60 minutes]**

### **CASE Expressions:**

- **are introduced in Oracle 9i.**
- **are used to implement 'If ..Then..Else'.**
- **It avoids of writing a separate procedure to implement  
'If .. Then ..Else'.**
- **In SQL query we can implement 'If..then..else' using CASE  
expression.**

### **It can be used in 2 ways:**

- **Simple Case => can check equality condition only**

- **Searched Case => can check any type of condition**

### **Simple Case:**

#### **Syntax:**

```
CASE <column_name>
When <value> Then <return_expression>
When <value> Then <return_expression>
.....
.....
[Else <return_expression>]
END
```

### **Searched Case:**

#### **Syntax:**

```
CASE
when <condition> then <return_expression>
when <condition> then <return_expression>
.....
.....
[Else <return_expression>]
END
```

### **Ex on simple case:**

**Increase emp salaries as following:**

**10 dept => increase 10%**  
**20 dept => increase 20%**  
**30 dept => increase 15%**  
**others => increase 5%**

**Update emp set sal=**

```
Case deptno
When 10 Then sal+sal*0.1
When 20 Then sal+sal*0.2
When 30 Then sal+Sal*0.15
Else sal+Sal*0.05
End;
```

### **Ex on Simple Case:**

```
Display clerk as worker
manager as boss
president as big boss
others employee
```

```
Select ename,
CASE JOB
When 'CLERK' Then 'WORKER'
When 'MANAGER' Then 'BOSS'
When 'PRESIDENT' then 'BIG BOSS'
Else 'EMPLOYEE'
END as "JOB"
from emp;
```

```
Display the emp names, salaries and
salary ranges as following:
sal >3000 => HiSal
sal<3000 => LoSal
sal=3000 => AvrgSal
```

emp		
ename	sal	sal_range

<b>SMITH</b>	<b>800</b>	<b>LoSal</b>
<b>ALLEN</b>	<b>4500</b>	<b>HiSal</b>
<b>BLAKE</b>	<b>3000</b>	<b>AvrgSal</b>

```

Select ename, sal,
CASE
  When sal>3000 Then 'HiSal'
  When sal<3000 Then 'LoSal'
  Else 'AvrgSal'
End as "Salary_Range"
from emp;

```

### **Student**

<b>sid</b>	<b>sname</b>	<b>m1</b>	<b>m2</b>	<b>m3</b>

**Max Marks: 100**

**Min marks: 40 in each sub**

**Display sid,sname,m1,m2,m3 and result:**

```

Select sid,sname,m1,m2,m3,
CASE
  when m1>=40 and m2>=40 and m3>=40 then 'PASS'
  Else 'FAIL'
End as "Result"
from student;

```

**In SQL, If..Then..Else can be implemented using 2 ways:**

- **Using Case Expressions**
- **Using Decode() Function**

**Decode():**

- **is used to implement if..then..else in SQL.**
- **It can check equality only.**

**Display job titles as following using decode() function:**

**CLERK => WORKER**

**MANAGER => BOSS**

**PRESIDENT => BIG BOSS**

**OTHERS => EMPLOYEE**

```
Select empno,ename,
Decode(Job,
'CLERK','WORKER',
'MANAGER','BOSS',
'PRESIDENT','BIG BOSS',
'EMPLOYEE') as "Job"
from emp;
```

**Update emp salaries as following:**

**deptno = 10 => increase 10%**

**deptno = 20 => increase 20%**

**deptno = 30 => increase 15%**

**others => increase 5%**

```
Update emp set sal=
Decode(Deptno,
10,sal+sal*0.1,
20,sal+sal*0.2,
30,sal+sal*0.15,
sal+sal*0.05);
```

### **Differences b/w Decode() & CASE expression:**

<b>Decode()</b>	<b>CASE Expression</b>
<b>can check equality condition only</b>	<b>can check any type of condition</b>
<b>It is Not ANSI Standard [Not Portable]</b>	<b>It is ANSI Standard. [Portable]</b>

# Clauses in SQL

Wednesday, November 24, 2021 6:36 PM

## Clauses:

### Syntax:

```
Select [DISTINCT] <column_list>/*
  FROM <table_list>
  [WHERE <condition>]
  [GROUP BY <group_column_list>]
  [HAVING <group_condition>]
  [ORDER BY <column_name> Asc/Desc];
```

English	SQL
Sentences	Queries
Words	Clauses

**Clause => is part of a query**

**SELECT  
FROM  
WHERE  
DISTINCT  
GROUP BY  
HAVING  
ORDER BY**

**SELECT clause:  
is used to specify column list**

**Select empno,ename  
Select \* => \* = All Columns**

**FROM clause:**  
is used to specify the table names.

**Ex:**

**FROM emp**  
**FROM emp,dept**  
**FROM student**

**WHERE clause:**

- is used to specify condition.

**Ex:**

**WHERE sal>3000**  
**WHERE sal between 2000 and 3000**  
**WHERE JOB<>'MANAGER'**

**Display the enames and their salaries.**

**Display emp records whose slary is <3000:**

**SELECT ename,sal  
FROM emp  
WHERE sal<3000;**

**DISTINCT:**

- used to avoid duplicate rows

**JOB**

-----

**CLERK**  
**MANAGER**  
**MANAGER**  
**CLERK**

**display the job titles offered  
by company:**

**SELECT DITINCT job  
from emp;**

**MANAGER**  
**CLERK**  
**CLERK**  
**MANAGER**  
**CLERK**

**SELECT DISTINCT job**  
**from emp;**  
  
**CLERK**  
**MANAGER**

**Display deptnos available in company:**

**deptno**

-----  
**20**  
**30**  
**30**  
**20**  
**20**  
**10**  
**10**  
**20**  
**30**

**Select DISTINCT deptno from emp;**

**10**  
**20**  
**30**

**std**

<b>sid</b>	<b>sname</b>	<b>cname</b>
<b>1001</b>	<b>A</b>	<b>JAVA</b>
		<b>ORACLE</b>
		<b>ORACLE</b>
		<b>JAVA</b>
		<b>JAVA</b>
		<b>CPP</b>

**Display the course names offered by Institute:**

**Select DISTINCT course from std;**

**JAVA  
ORACLE**

**ORDER BY clause:**

**used to arrange the records in Ascending or Descending order according specific column/columns.**

**Alphabet**

**a..z A..Z => Asc  
z..a Z..A => Desc**

**Numbers:**

**1..10 => Asc  
10..1 => Desc**

**Date:**

**1981,1982,1983 => ASC  
1983,1982,1981 => DESC**

**Display the emp names in Alphabetical oreder:**

**Select \* from emp  
ORDER BY ename; //Ascending [default one]**

**[or]**

**Select \* from emp  
ORDER BY ename Asc;**

**Display emp records in descending order according to salary:**

**Select \* from emp  
Order By Sal Desc;**

**Display emp records in descending order according to salary. If salary is same then arrange enames in Ascending order:**

**Select empno,ename,sal  
from emp  
Order by sal desc,ename Asc;**

**Display the emp records in Descending order according to sal. If salary is same check with ename & arrange in alphabetical order. if name is same then arrange them in ascending order according to empno:**

**Select empno,ename,sal  
from emp  
order by sal desc,ename asc,empno asc;**

**Display the emp records in Ascending Order According to Deptno:**

**Select empno,ename,job,sal,deptno  
From emp**

**Order By Deptno;**

**Display the emp records in ascending order according to deptno. Within dept display salaries in descending order:**

**Select empno,ename,deptno,sal  
from emp  
Order By deptno Asc,Sal Desc;**

**Select empno,ename,job,sal  
from emp  
Order By 2; --2 => ename**

**Select empno,ename,job,sal  
from emp  
Order By 4 Desc; --4 => sal**

**Select \* from emp  
order by 2;**

**Group By clause:**

- **Group By clause is used to group the records according to specific column/columns.**
- **If column value is same that will be treated as**

**one group.**

- Using this, we can apply aggregate functions on group of records.
- It converts detailed data to summarized data.

**Emp**      *Detailed DJ<sup>n</sup>*

empno	ename	sal	deptno
1001	A	5000	20
1002	B	8000	20
1003	C	4000	30
1004	D	6000	30
1005	E	10000	10
1006	F	8000	10

*Group By*

*Summarized DJ<sup>n</sup>*

deptno	sum(Sal)
20	13000
30	10000
10	18000

**Select sum(sal) from emp;**

**Display dept wise sum of salaries:**

```
Select deptno,sum(Sal)
from emp
where deptno in(10,30)
Group By deptno
Order By 1;
```

**Execution Order:**

**FROM  
WHERE  
GROUP BY  
HAVING  
SELECT  
ORDER BY**

**FROM emp:**

**Emp**

<b>empno</b>	<b>ename</b>	<b>sal</b>	<b>deptno</b>
1001	A	5000	20
1002	B	8000	20
1003	C	4000	30
1004	D	6000	30
1005	E	10000	10
1006	F	8000	10

**where deptno in(10,30):**

1003	C	4000	30
1004	D	6000	30
1005	E	10000	10
1006	F	8000	10

**GROUP BY deptno:**

30	<table border="1"><tbody><tr><td>1003</td><td>C</td><td>4000</td><td>30</td></tr><tr><td>1004</td><td>D</td><td>6000</td><td>30</td></tr></tbody></table> <p style="margin-left: 20px;"><math>\frac{\text{sum}(\text{Sal})}{10000}</math></p>	1003	C	4000	30	1004	D	6000	30
1003	C	4000	30						
1004	D	6000	30						
10	<table border="1"><tbody><tr><td>1005</td><td>E</td><td>10000</td><td>10</td></tr><tr><td>1006</td><td>F</td><td>8000</td><td>10</td></tr></tbody></table> <p style="margin-left: 20px;"><math>\frac{\text{sum}(\text{Sal})}{18000}</math></p>	1005	E	10000	10	1006	F	8000	10
1005	E	10000	10						
1006	F	8000	10						

**on these group of records, aggregate function will be applied**

**SELECT deptno,sum(Sal):**

<b>deptno</b>	<b>sum(Sal)</b>

<b>30</b>	<b>10000</b>
<b>10</b>	<b>18000</b>

**ORDER BY 1 [deptno]:**

<b>deptno</b>	<b>sum(Sal)</b>
<b>10</b>	<b>18000</b>
<b>30</b>	<b>10000</b>

**Display the number of employees who are joined in different years [Year wise no of emps]:**

<b>1980</b>	<b>3</b>
<b>1981</b>	<b>2</b>
<b>1982</b>	<b>6</b>
<b>1983</b>	<b>3</b>

```
Select to_char(hiredate,'yyyy') YEAR, Count(*)
FROM emp
GROUP BY YEAR
ORDER BY YEAR;
```

**Display the number of emps joined in different quarters [quarter wise no of emps joined organization]:**

<b>quarter</b>	<b>no_of_emps</b>
<b>1</b>	<b>4</b>
<b>2</b>	<b>3</b>

<b>3</b>	<b>2</b>
<b>4</b>	<b>8</b>

```
Select to_Char(hiredate,'q') QUARTER,
Count(*) NO_OF_EMPS
FROM emp
GROUP BY QUARTER
ORDER BY QUARTER; //ERROR
```

**We cannot use alias name in GROUP BY.**  
**Because GROUP BY executed before**  
**SELECT. It does not know the alias name.**

**We can use alias name in ORDER BY.**  
**Because ORDER BY gets executed after**  
**SELECT. It knows the alias name.**

```
Select to_Char(hiredate,'q') QUARTER,
Count(*) NO_OF_EMPS
FROM emp
GROUP BY to_char(hiredate,'q')
ORDER BY QUARTER;
```

**Find deptwise maximum salary &  
minimum sal:**

```
Select deptno, max(sal) "max sal",
min(sal) "min sal"
from emp
Group By deptno
Order By 1;
```

**Find Dept wise Number of employees:**

```
Select deptno, count(*) number_of_emps
FROM emp
```

```
GROUP BY deptno
ORDER BY 1;
```

**Display job wise number of emps:**

```
Select Job, Count(*) Number_Of_Emps
FROM emp
GROUP BY Job
ORDER BY 1;
```

**Find job wise max sal & min sal:**

```
Select Job, max(sal) max_Sal,
min(Sal) min_Sal
FROM emp
GROUP BY Job;
```

**Display job wise sum of salaries:**

```
SELECT job,Sum(Sal) sum_of_sal
FROM emp
GROUP BY job;
```

**Find dept wise avrg sal:**

```
SELECT deptno,avg(sal) avrg_sal
FROM emp
GROUP BY deptno
```

**ORDER BY 1;**

**Find job wise avg sal:**

```
SELECT Job, avg(Sal) avg_Sal  
FROM emp  
GROUP BY Job;
```

### **STUDENT**

<b>sid</b>	<b>sname</b>	<b>COURSE</b>	<b>Fee</b>
<b>1001</b>	<b>A</b>	<b>JAVA</b>	<b>8000</b>
<b>1002</b>	<b>B</b>	<b>ORACLE</b>	<b>5000</b>
<b>1003</b>	<b>C</b>	<b>JAVA</b>	<b>8000</b>
<b>1004</b>	<b>D</b>	<b>ORACLE</b>	<b>5000</b>

**Course wise number of students**

<b>JAVA</b>	<b>2</b>
<b>ORACLE</b>	<b>2</b>

**Display the amount collected by each course:**

**Display number of employees according to salary range:**

```
Select  
CASE  
when sal>3000 then 'HiSal'  
when sal<3000 then 'LoSal'  
when sal=3000 then 'AvrgSal'  
END as sal_range,  
Count(*) as num_of_emps  
FROM emp  
Group By  
CASE  
when sal>3000 then 'HiSal'  
when sal<3000 then 'LoSal'  
when sal=3000 then 'AvrgSal'  
END;
```

sal_range	num_of_emps
hisal	5
losal	6
avrgsal	3

**NOTE:**

- We cannot use aggregate [group ] functions in "WHERE" clause.
- When we use GOUP BY, in SELECT clause we can write grouping columns or aggregate functions. Other than these are not accepted in SELECT clause.

**HAVING clause:**

- is used to write the conditions on groups [group of records].
- We can use aggregate functions in HAVING clause.
- It is applied on result of GROUP BY.
- It is executed after GROUP BY.
- It cannot be used without 'GROUP BY'.

**Display the department numbers which are spending more than 15000 as sum salaries:**

```
Select deptno,sum(Sal)
FROM emp
GROUP BY deptno
HAVING sum(Sal)>15000
ORDER BY 1;
```

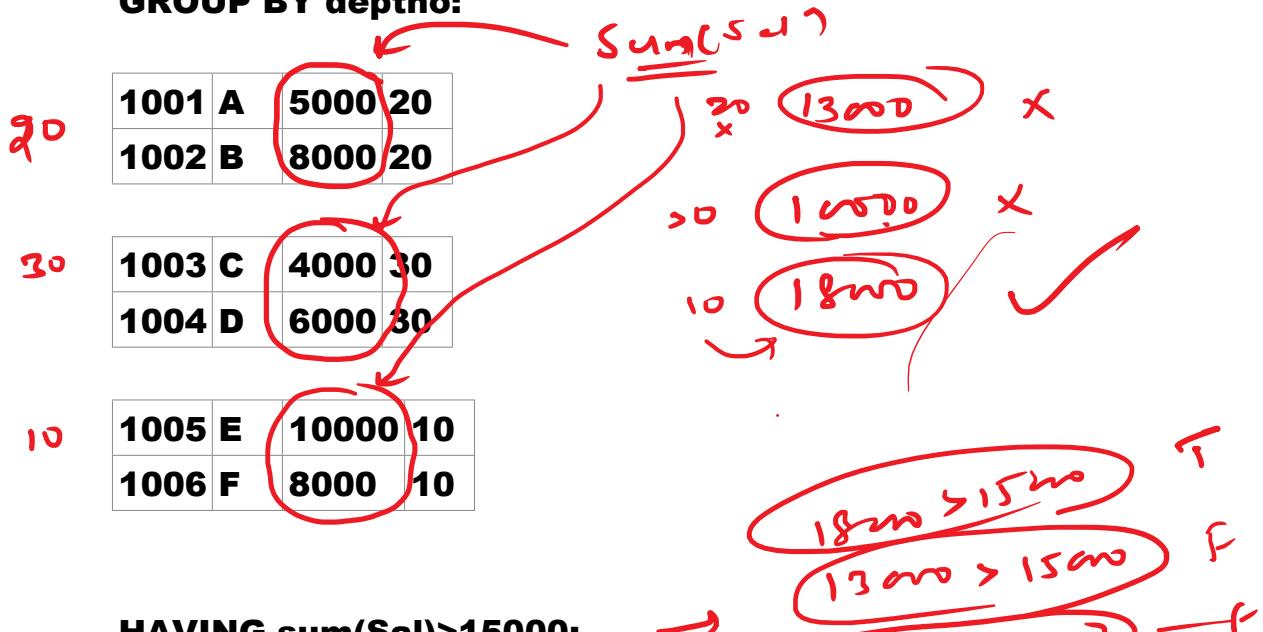
**FROM  
WHERE  
GROUP BY  
HAVING  
SELECT  
ORDER BY**

**FROM emp:**

**Emp**

empno	ename	sal	deptno
1001	A	5000	20
1002	B	8000	20
1003	C	4000	30
1004	D	6000	30
1005	E	10000	10
1006	F	8000	10

**GROUP BY deptno:**



1005	E	10000	10
1006	F	8000	10

**SELECT deptno,sum(Sal):**

<b>10</b>	<b>18000</b>
-----------	--------------

**ORDER BY 1:**

<b>10</b>	<b>18000</b>
-----------	--------------

**Display the dept nums which are having more than 3 emps:**

```
Select deptno,count(*)  
FROM emp  
GROUP BY deptno  
HAVING count(*)>3  
ORDER BY 1;
```

**Display the job titles which are having more than 3 employees:**

```
Select Job, Count(*)  
FROM emp  
GROUP BY Job  
HAVING Count(*)>3;
```

**Display the job titles on which organization is spending more than 7000:**

```
Select Job, Sum(Sal)
FROM emp
GROUP BY Job
HAVING sum(Sal)>7000;
```

### **Differences b/w WHERE clause & HAVING clause:**

<b>WHERE</b>	<b>HAVING</b>
<ul style="list-style-type: none"><li>• it is applied on row</li><li>• we cannot use aggregate functions in WHERE clause</li><li>• It can be used without GROUP BY</li><li>• it is executed before GROUP BY</li></ul>	<ul style="list-style-type: none"><li>• it is applied on group</li><li>• We can use aggregate functions in HAVING clause</li><li>• It cannot be used without GROUP BY</li><li>• it is executed after GROUP BY</li></ul>

### **Grouping the records according to multiple columns:**

**Display dept wise, with in the dept job wise number of employees:**

```
Select Deptno, Job, Count(*)
FROM emp
```

**GROUP BY deptno,job**

**ORDER BY 1;**

**Display Year wise, With in the year quarter wise number of employees joined in organization:**

<b>1980</b>	<b>1</b>	<b>3</b>
	<b>2</b>	<b>2</b>
	<b>3</b>	<b>5</b>
	<b>4</b>	<b>1</b>
<b>1981</b>	<b>1</b>	<b>7</b>
	<b>2</b>	<b>4</b>
	<b>3</b>	<b>3</b>
	<b>4</b>	<b>4</b>

**Select**

**to\_Char(hiredate,'yyyy') YEAR,**

**to\_Char(hiredate,'q') QUARTER,**

**Count(\*) no\_of\_emps**

**FROM emp**

**GROUP BY to\_Char(hiredate,'yyyy'),to\_Char(hiredate,'q')**

**ORDER By YEAR;**

## **SALES**

<b>dateid</b>	<b>amount</b>
<b>25-FEB-2015</b>	<b>10000</b>
<b>26-FEB-2015</b>	<b>20000</b>

<b>2016</b>	
<b>2017</b>	
<b>2018</b>	

**Display year wise, quarter wise sales:**

**Select**

```
to_char(dateid,'yyyy') YEAR,
to_char(dateid,'q') QUARTER,
Sum(amount) Amount
FROM sales
GROUP BY to_char(dateid,'yyyy'),to_char(dateid,'q')
ORDER BY 1;
```

**PERSON**

<b>pid</b>	<b>pname</b>	<b>state</b>	<b>age</b>	<b>gender</b>	<b>aadhar_number</b>
------------	--------------	--------------	------------	---------------	----------------------

**Display state wise, gender wise number of people:**

<b>TELANGANA</b>	<b>M</b>	
	<b>F</b>	
<b>GUJARAT</b>	<b>M</b>	
	<b>F</b>	
<b>MR</b>	<b>M</b>	
	<b>F</b>	

```
Select State, gender, Count(*)  
FROM person  
GROUP BY state, gender  
ORDER BY 1;
```

**Rollup() & Cube() Functions:**  
**used to calculate sub totals & grand totals.**

**Rollup():**

**Group By Rollup(deptno,job)**

**It calculates dept wise sub totals & grand totals.**

**Group By Rollup(job,deptno)**

**It calculates job wise sub totals & grand totals.**

**Rollup() function calculates sub totals & grand totals according first column specified in function.**

**Cube():**

**Group By Cube(deptno,job)**

**It calculates dept wise, job wise sub totals & grand totals.**

**Cube() function calculates sub totals & grand totals according to all columns specified in function.**

**Display dept wise, job wise sum of salaries. Also calculate sub totals & grand totals according to deptno:**

```
Select deptno,job,sum(sal)
FROM emp
GROUP BY Rollup(deptno,job)
ORDER By 1;
```

**Display dept wise, job wise sum of salaries. Also calculate sub totals & grand totals according to deptno and job:**

```
Select deptno, job, sum(Sal)
from emp
group by Cube(deptno,job)
order by 1;
```

**Display dept wise, job wise number of employees. Calculate sub totals & grand totals according to deptno:**

```
Select deptno,job,count(*)
FROM emp
GROUP BY Rollup(deptno,job)
ORDER BY 1;
```

**Display year wise, quarter wise sales.**

**Also display sub total & grand total:**

**SALES**

<b>dateid</b>	<b>amount</b>
<b>1-jan-2015</b>	<b>10000</b>
<b>2-jan-2015</b>	<b>12000</b>
<b>2016 data</b>	
<b>2017 data</b>	

<b>YEAR</b>	<b>QUARTER</b>	<b>AMOUNT</b>
<b>2015</b>	<b>1</b>	<b>25000</b>
	<b>2</b>	<b>16000</b>
	<b>3</b>	<b>20000</b>
	<b>4</b>	<b>30000</b>
<b>2015 sub total</b>		
<b>2016</b>	<b>1</b>	
	<b>2</b>	
	<b>3</b>	
	<b>4</b>	
<b>2016 sub total</b>		
<b>GRAND TOTAL</b>		

```
Select to_char(dateid,'yyyy') YEAR,  
to_char(dateid,'q') QUARTER,  
Sum(amount)  
FROM sales  
GROUP BY  
Rollup(to_char(dateid,'yyyy'),to_char(dateid,'q'))  
ORDER BY 1;
```

**Display state wise, gender wise number of people. Calculate sub total & grand total**

**according to state:**

**PERSON**

pid	pname	state	gender	age	aadhar
-----	-------	-------	--------	-----	--------

State	Gender	no_of_people
<b>TELANGANA</b>	M	
	F	
<b>SUB TOTAL</b>		
<b>GUJARAT</b>	M	
	F	
<b>SUB TOTAL</b>		
<b>GRAND TOTAL</b>		

**Select state, gender, count(\*)  
from PERSON  
GROUP BY Rollup(state, gender)  
ORDER BY 1;**

State	Age_Range	Gender	Number_OF_PEOPLE
<b>TELANGANA</b>	<b>scitizen</b>	M	
		F	
	<b>MIDDLE AGE</b>	M	
		F	
	<b>MINOR</b>	M	
		F	

```
Select State,
CASE
when age>=60 then 'SCITIZEN'
when age between 18 and 59 then 'MIDDLE AGE'
when age<18 then 'MINOR'
END as "age_range",
gender, Count(*)
FROM person
GROUP BY Rollup(State,
CASE
when age>=60 then 'SCITIZEN'
when age between 18 and 59 then 'MIDDLE AGE'
when age<18 then 'MINOR'
END,
GENDER)
ORDER BY 1;
```

## SET OPERATORS

Monday, November 29, 2021 6:29 PM

$$A = \{1, 2, 3, 4, 5, 6\}$$

$$B = \{7, 1, 2, 3, 8, 9\}$$

$$A \cup B = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

$$A \cap B = \{1, 2, 3\}$$

$$A - B = \{4, 5, 6\}$$

$$B - A = \{7, 8, 9\}$$

**CUST1**

cid	cname
1001	A
1002	B
1003	C

**CUST2**

cid	cname
2001	D
1002	B
2002	E

### SET OPERATORS:

To combine rows from multiple tables,  
use SET OPERATORS.

1001  
1002  
1003  
2001  
2002

### Syntax:

```
SELECT STATEMENT
set operator
SELECT STATEMENT;
```

### ORACLE SQL provides 4 SET OPERATORS:

- UNION
- UNION ALL

- **INTERSECT**
- **MINUS**

**CUST1**

<b>cid</b>	<b>cname</b>
<b>1001</b>	<b>A</b>
<b>1002</b>	<b>B</b>
<b>1003</b>	<b>C</b>

**CUST2**

<b>cid</b>	<b>cname</b>
<b>2001</b>	<b>D</b>
<b>1002</b>	<b>B</b>
<b>2002</b>	<b>E</b>

### **UNION:**

- **is used to combine all records from multiple tables uniquely.**
- **It will not give duplicate records.**
- **It gives result in order.**

**SELECT cid, cname from CUST1**

**UNION**

**SELECT cid,cname from CUST2;**

<b>1001</b>	<b>A</b>
<b>1002</b>	<b>B</b>
<b>1003</b>	<b>C</b>
<b>2001</b>	<b>D</b>
<b>2002</b>	<b>E</b>

### **Rules:**

- Number of columns in both SELECT statements should be same.

**Ex:**

```
Select cid from cust1
UNION
Select cid,cname from cust1;
```

**ERROR at line 1:**  
**ORA-01789: query block has incorrect number of result columns**

- Corresponding columns data types of both select statements should be matched.

**Ex:**

```
Select cid,cname from Cust1
UNION
Select cname,cid from Cust2;
```

cid	cname
-----	-------

cname	cid
-------	-----

//ERROR

**ERROR at line 1:**  
**ORA-01790: expression must have same datatype as corresponding expression**

- Corresponding column names need not to be same.

**Ex:**

```
SELECT cid,cname from cust1
UNION
SELECT custid, custname from cust2;
```

cid	cname
-----	-------

custid	custname
--------	----------

**cid    cname**  
-----

**result**

cid	cname
-----	-------

```
SELECT cid,cname from cust
UNION
SELECT bid,bname from product;
```

**cust**

cid	cname
-----	-------

**product**

**UNION**

**SELECT pid,pname from product;**

**cid cname**

**product**

<b>pid</b>	<b>pname</b>
------------	--------------

**UNION ALL:**

- **It is used to combine all records from multiple tables including duplicate records.**
- **It does not give result in the order.**

**SELECT cid,cname from CUST1**

**UNION ALL**

**SELECT cid,cname from CUST2;**

<b>cid</b>	<b>cname</b>
<b>1001</b>	<b>A</b>
<b>1002</b>	<b>B</b>
<b>1003</b>	<b>C</b>
<b>2001</b>	<b>D</b>
<b>1002</b>	<b>B</b>
<b>2002</b>	<b>E</b>

**Differences b/w UNION and UNION ALL:**

## **UNION**

- It gives records uniquely
- gives result in order
- it is slower than UNION ALL

## **UNION ALL**

- \* It gives duplicate record
- does not give result in order
- it is faster than UNION

## **INTERSECT:**

- is used to get common records from multiple tables.

**Select cid, cname from cust1**

**INTERSECT**

**Select cid,cname from cust2;**

<b>cid</b>	<b>cname</b>
<b>1002</b>	<b>B</b>

## **MINUS:**

**it is used to get all records from first table**

**except common records.**

**Ex:**

**Select cid, cname from cust1**

**MINUS**

**Select cid,cname from cust2;**

**1001**

**1003**

**Select cid, cname from cust2**

**MINUS**

**Select cid,cname from cust1;**

**2001**

**2002**

$$\text{Cust1} \cup \text{Cust2} = \text{Cust2} \cup \text{Cust1}$$

$$\text{Cust1} \cup \Delta \text{ Cust2} = \text{Cust2} \cup \Delta \text{ Cust1}$$

$$\text{Cust1} \cap \text{Cust2} = \text{Cust2} \cap \text{Cust1}$$

$$\text{Cust1} \setminus \text{Cust2} \neq$$

↓

All from Cust1  
except Common

$$\text{Cust2} \setminus \text{Cust1}$$

All from Cust2  
except Common

**Display the list of jobs available in deptno  
10 & 20:**

**Select job from emp where deptno=10;**

**CLERK  
MANAGER  
PRESIDENT**

**Select job from emp where deptno=20;**

**CLERK  
MANAGER  
ANALYST  
CLERK  
ANALYST**

**Select job from emp where deptno=10**

**UNION**

**Select job from emp where deptno=20;**

**ANALYST  
CLERK  
MANAGER  
PRESIDENT**

**DISPLAY the common designations  
available in deptno 10 & 20:**

**Select job from emp where deptno=10**

**INTERSECT**

**Select job from emp where deptno=20;**

**Display specific job titles of deptno 10:**

**Select job from emp where deptno=10**

**MINUS**

**Select job from emp where deptno=20;**

**Display the specific jobs of deptno 20:**

**Select job from emp where deptno=20**

**MINUS**

**Select job from emp where deptno=10;**

**emp\_us**

<b>empno</b>	<b>ename</b>	<b>job</b>	<b>sal</b>
<b>1001</b>			
<b>1002</b>			
<b>1003</b>			

**emp\_ind**

<b>empno</b>	<b>ename</b>	<b>job</b>	<b>sal</b>
<b>3001</b>			
<b>3002</b>			
<b>1003</b>			
<b>3003</b>			

**Display all emp records who are working for ind & US:**

**Select \* from emp\_us  
UNION  
Select \* from emp\_ind;**

**Display the emp records who are working for us & ind including duplicates:**

**Select \* from emp\_us  
UNION ALL  
Select \* from emp\_ind;**

**Display the common employees who are working for ind & us:**

**Select \* from emp\_us  
INTERSECT  
Select \* from emp\_ind;**

**Display the specific emps of IND:**

**Select \* from emp\_ind;  
MINUS  
Select \* from emp\_us;**

**Display the specific emps who are working for us:**

**Select \* from emp\_us  
MINUS  
Select \* from emp\_ind;**

## Joins

Tuesday, November 30, 2021 6:28 PM

### Joins:

#### College Database

**Student  
Marks  
Fee  
Library  
Employee**

*JOIN CONDITION*

*Where student.sid = marks.sid*

**STUDENT**

**Marks**

<b>stdid</b>	<b>sname</b>	<b>scity</b>	<b>sid</b>	<b>Maths</b>	<b>Physics</b>	<b>Chemistry</b>
1001	A	Hyd	1001	50	80	70
1002	B	Mumbai	1002	77	88	44
1003	C	Delhi	1003	60	40	90

#### MATHS Faculty

**sid sname Maths**

**STUDENT MARKS**

*sid*      *sname*  
*=*

**PHYSICS**

**sid sname PHYSICS**

**sid sname Maths Physics Chemistry**

#### ORDERS

**oid pid pname quantity ord\_date del\_date cid**

## **CUSTOMERS**

<b>cid</b>	<b>cname</b>	<b>ccity</b>
------------	--------------	--------------

<b>oid</b>	<b>pid</b>	<b>pname</b>	<b>del_date</b>	<b>cid</b>	<b>cname</b>	<b>ccity</b>
------------	------------	--------------	-----------------	------------	--------------	--------------

## **ORDERS**

## **CUSTOMERS**

**where ORDERS.cid = CUSTOMER.cid**

### **Joins:**

- **Join => combine / Connect / Link**
- **Join is an operation that combines one table records with another based on Join Condition.**
- **Join condition decides one table records should be joined with which records in another table.**
- **Normally, to perform join operation a common column is required.**
- **This common column name need not to be same.**
- **Ex: Student.stdid = Marks.sid**

### **Types of Joins:**

- **Equi Join / Inner join**
- **Outer Join**
  - **Left outer join**
  - **Right outer join**
  - **Full outer join**
- **Non-Equi Join**
- **Self-Join**
- **Cartesian Join / Cross Join / Product Join**

### **Equi Join:**

- **If Join Operation is performed based on equality condition then it is called "Equi Join".**

- It can be also called as "Inner Join".
- Inner Join => the records which satisfy the condition will be displayed.

**Emp**

empno	ename	sal	deptno
1001		10	
1002		10	
1003		20	
1004		20	
1005		30	

**Dept**

deptno	dname	loc
10	Accounts	DALLAS
20	RESEARCH	CHICAGO
30	Sales	NEW YORK
40	OPERATIONS ..	

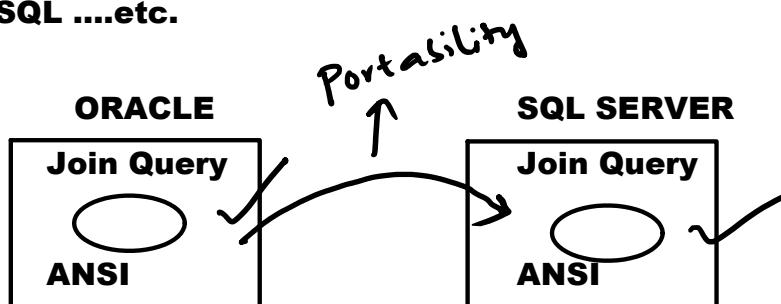
**Display employee details along with department details:**

ename	sal	dname	loc
EMP		DEPT	

**Join Query can be written in 2 styles:**

- Oracle Style / Native Style
- ANSI Style → Best way ✓

**With ANSI style we get Portability feature.**  
**The Join query written in ANSI style can be run in Other RDBMS like SQL Server, MySQL ....etc.**



**Emp**

<b>empno</b>	<b>ename</b>	<b>sal</b>	<b>deptno</b>
<b>1001</b>		<b>10</b>	
<b>1002</b>		<b>10</b>	
<b>1003</b>		<b>20</b>	
<b>1004</b>		<b>20</b>	
<b>1005</b>		<b>30</b>	

**Dept**

<b>deptno</b>	<b>dname</b>	<b>loc</b>
<b>10</b>	<b>Accounts</b>	<b>DALLAS</b>
<b>20</b>	<b>RESEARCH</b>	<b>CHICAGO</b>
<b>30</b>	<b>Sales</b>	<b>NEW YORK</b>
<b>40</b>	<b>OPERATIONS</b>	..

**Display employee details along with department details:**

<b>ename</b>	<b>sal</b>	<b>dname</b>	<b>loc</b>
<b>EMP</b>		<b>DEPT</b>	

### **ORACLE STYLE:**

```
SELECT ename, sal, dname, loc
FROM emp,dept
WHERE emp.deptno=dept.deptno;
```

#### **Note:**

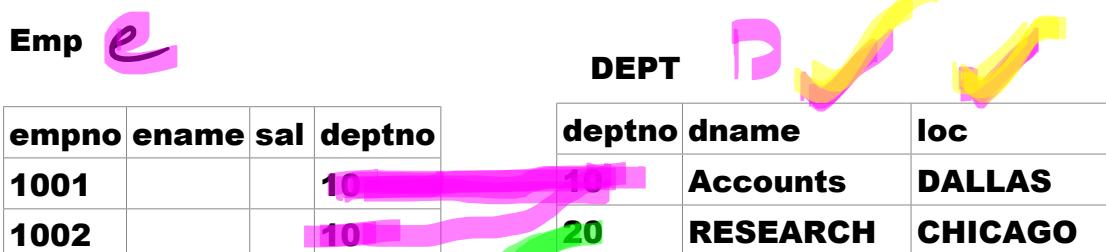
**Its better to use alias names in Join Query.**

#### **Advantages with Alias:**

- It makes table name short
- It improves performance

#### **Equi Join using Alias Names:**

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e,dept d
WHERE e.deptno=d.deptno;
```



1001		10	10	Accounts	DALLAS
1002		10	20	RESEARCH	CHICAGO
1003		20	30	Sales	NEW YORK
1004		20	40	OPERATION	..
1005		30		S	

### ANSI Style:

- To separate table names we use keyword in ANSI style where as in ORACLE Style to separate tables we use ,
- In ANSI Style we write join condition in "ON" Clause or "USING" clause where as in ORACLE Style we write join condition in "WHERE" clause.

### ANSI Style:

ename	sal	dname	loc
emp e		dept d	

### ON Clause [Best way]

```
Select e.ename,e.sal,d.dname,d.loc
From emp e INNER JOIN dept d
ON e.deptno=d.deptno;
```

### USING clause:

#### Note:

Common Column Name must be same.  
If we use USING clause implicitly it will be converted to ON clause

```
Select e.ename,e.sal,d.dname,d.loc
From emp e INNER JOIN dept d
USING(deptno);
```

**Display the emp records along with dept details who are working in CHICAGO:**

**Emp**

empno	ename	sal	deptno
1001			10
1002			10
1003			20
1004			20
1005			30

**Dept**

deptno	dname	loc
10	Accounts	DALLAS
20	RESEARCH	CHICAGO
30	Sales	NEW YORK
40	OPERATIONS ..	

**Oracle Style:**

```
Select e.ename,e.sal,d.dname,d.loc
FROM emp e, Dept d
WHERE e.deptno=d.deptno and d.loc='CHICAGO';
```

**ANSI Style:**

```
Select e.ename, e.sal, d.dname,d.loc
FROM emp e INNER JOIN dept d
ON e.deptno=d.deptno
WHERE d.loc='CHICAGO';
```

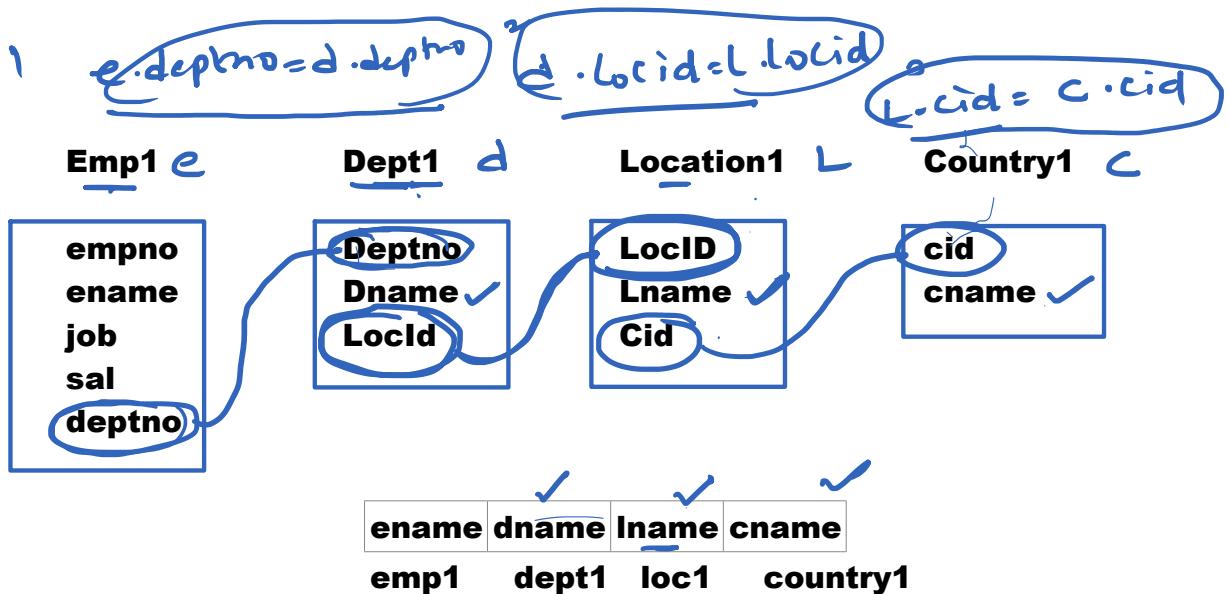
**Display the emp record with dept details whose name is 'BLAKE':**

**Oracle Style:**

```
Select e.ename,e.sal,d.dname,d.loc
FROM emp e, Dept d
where e.deptno = d.deptno and
e.ename='BLAKE';
```

**ANSI Style:**

```
Select e.ename,e.sal,d.dname,d.loc
FROM emp e INNER JOIN dept d
ON e.deptno=d.deptno
WHERE e.ename='BLAKE';
```



### Oracle Style:

```
Select e.ename, d.dname, l.lname, c.cname
FROM emp1 e,Dept1 d,Location1 l,Country1 c
WHERE e.deptno=d.deptno and d.locid=l.locid and l.cid=c.cid;
```

### ANSI Style:

```
SELECT e.ename, d.dname, l.lname, c.cname
FROM emp1 e INNER JOIN dept1 d
ON e.deptno=d.deptno
INNER JOIN location1 l
ON d.locid=l.locid
INNER JOIN country1 c
ON l.cid=c.cid;
```

**Student**

sid	sname	cid
1001	A	10
1002	B	30
1003	C	30
1004	D	20

**COURSE**

cid	cname
10	JAVA
20	C#
30	PYTHON

**Display the student details along with**

**course names:**

sid	sname	cname
1001	SCOTT	ACCOUNTS

**Outer Join:**

- Inner Join can display matching records only. Inner Join cannot display unmatched records.
  - Outer Join is used to display matching & unmatched records.
  - Outer Join = matching records + unmatched records
- 
- In Oracle Style we use Outer Join Operator to perform Join operation.
  - Outer Join symbol is : (+)

There are 3 sub types:

- Left Outer Join
- Right Outer Join
- Full Outer Join

**Left Outer Join:**

- It gives matching records and unmatched records from left table.
- Left Outer Join = matching recs + unmatched from left table

**In Oracle Style:**

Display all employees records along with dept names. Also display the emp records who are not assigned to any dept:

[Left Outer Join]

**Emp**

empno	ename	sal	deptno
1001			10
1002			10
1003			20
1004			20

**Dept**

deptno	dname	loc
10	ACCOUNTS	DALLAS
20	RESEARCH	CHICAGO
30	SALES	NEW YORK
40	OPERATIONS	..

1005			30
1006	E		

**IN Oracle style based on JOIN condition,  
oracle decides left table & right table**

**WHERE e.deptno=d.deptno**

**e => emp => left**  
**d => dept => right**

## In ANSI Style:

**FROM clause we write: emp e INNER JOIN dept d**

**In Oracle Style, For Left Outer Join write  
Outer Join Operator [ (+) ] at Right Side**

**Ex: where e.deptno = d.deptno(+)**

## **ORACLE Style:**

```
Select e.ename,e.sal,d.dname,d.loc  
FROM emp e, dept d  
WHERE e.deptno = d.deptno(+);
```

## **ANSI Style:**

```
SELECT e.ename,e.sal,d.dname,d.loc  
FROM emp e LEFT OUTER JOIN dept d  
ON e.deptno = d.deptno;
```

### **Right Outer Join:**

- It gives matching records and unmatched records from right side table.
- **RIGHT OUTER JOIN = matching recs + unmatched from right table**

**Display the emp records along with dept details. Also display the dept names which are not having any emps:  
[Right Outer Join]:**

**Oracle Style:**

```
SELECT e.ename, e.sal, d.dname,d.loc  
FROM emp e,dept d  
WHERE e.deptno(+) = d.deptno ;
```

**ANSI Style:**

```
SELECT e.ename, e.sal, d.dname, d.loc  
FROM emp e RIGHT OUTER JOIN dept d  
ON e.deptno = d.deptno;
```

**Where e.deptno = d.deptno => Equi Join**

**Where e.deptno = d.deptno(+) => Left Outer Join**

**Where e.deptno(+) = d.deptno => Right Outer Join**

**Where e.deptno(+) = d.deptno(+) => ERROR**

### **FULL OUTER JOIN:**

**Left Outer => Matching Records + unmatched from Left**

## **UNION**

**Right Outer => Matching records + unmatching records Right**

### **Oracle Style:**

```
Select e.ename, e.sal, d.dname, d.loc
FROM emp e, dept d
WHERE e.deptno = d.deptno(+)
UNION
Select e.ename, e.sal, d.dname, d.loc
FROM emp e, dept d
WHERE e.deptno(+) = d.deptno;
```

### **ANSI style:**

```
Select e.ename,e.sal,d.dname,d.loc
FROM emp e FULL OUTER JOIN dept d
ON e.deptno=d.deptno;
```

**Emp**

<b>empno</b>	<b>ename</b>	<b>sal</b>	<b>deptno</b>
<b>1001</b>		<b>10</b>	
<b>1002</b>		<b>10</b>	
<b>1003</b>		<b>20</b>	
<b>1004</b>		<b>20</b>	

**Dept**

<b>deptno</b>	<b>dname</b>	<b>loc</b>
<b>10</b>	<b>Accounts</b>	<b>DALLAS</b>
<b>20</b>	<b>RESEARCH</b>	<b>CHICAGO</b>
<b>30</b>	<b>Sales</b>	<b>NEW YORK</b>
<b>40</b>	<b>OPERATIONS</b>	

1002		10
1003		20
1004		20
1005		30
1006	E	

20	RESEARCH	CHICAGO
30	Sales	NEW YORK
40	OPERATIONS	..

**Display the employees who are not assigned to any dept:  
[Left Outer Join + Condition]**

**Select e.ename,e.sal,d.dname,d.loc  
FROM emp e,Dept d  
Where e.deptno=d.deptno(+)  
and d.dname is null;**

**ANSI Style:**

**Select e.ename,e.sal,d.dname,d.loc  
FROM emp e LEFT OUTER JOIN dept d  
ON e.deptno=d.deptno  
WHERE d.dname is null;**

**display the departments which are not having employees:  
[Right Outer Join + Condition]**

**Select e.ename,e.sal,d.dname,d.loc  
FROM emp e, Dept d  
WHERE e.deptno(+) = d.deptno  
and e.ename is null;**

**ANSI Style:**

**Select e.ename,e.sal,d.dname,d.loc  
FROM emp e RIGHT OUTER JOIN Dept d  
ON e.deptno=d.deptno  
WHERE e.ename is NULL;**

**Display the employee records to whom dept is not assigned. Also display the depts which are not having employees:**

**FULL OUTER JOIN + COnditions**

**ORACLE STYLE:**

```
Select e.ename,e.sal,d.dname,d.loc  
FROM emp e, dept d  
WHERE e.deptno = d.deptno(+) and d.dname is null  
UNION  
Select e.ename,e.sal,d.dname,d.loc  
FROM emp e, dept d  
WHERE e.deptno(+) = d.deptno and e.ename is null;
```

**ANSI Style:**

```
Select e.ename,e.sal,d.dname,d.loc  
FROM emp e FULL OUTER JOIN dept d  
ON e.deptno = d.deptno  
WHERE d.dname is null or e.ename is null;
```

**Equi Join:**

**If Join operation is performed based on equality condition then it is called "Equi Join"**

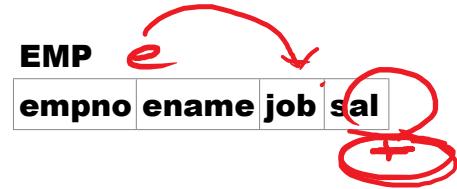
**Non-Equi Join:**

**If Join Operation is performed based on other than equality conditionlike < > between and ..etc.**

**Display the employee details with salary**

**grades:**

salgrade 5			
GRADE	LOSAL	HISAL	
I	700	1200	
empno	ename	sal	grade



Select e.empno,  
e.ename, e.sal,  
s.grade  
from emp e, Salgrade s  
where e.sal  
between s.losal and s.hisal

### **ORACLE STYLE:**

```
Select e.empno,e.ename,e.sal,s.grade  
FROM emp e, Salgrade s  
WHERE e.sal between s.losal and s.hisal;
```

### **ANSI Style:**

```
Select e.empno,e.ename,e.sal,s.grade  
FROM emp e JOIN Salgrade s  
ON e.sal between s.losal and s.hisal;
```

**Display the emp records who are not working in which depts:**

```
Select e.ename,d.dname  
FROM emp e, dept d  
WHERE e.deptno != d.deptno
```

**ORDER By 1;**

**Self-Join:**

- If a table is joined to itself then it is called "Self-Join".
- It can be also called as "Recursive Join".
- A table record will be joined with that table record only.

**emp e**

empno	ename	job	sal	mgr
1001	A	MANAGER		
1002	B	CLERK		1001
1003	C	SALESMAN		1001

**employee**

-----  
B  
C

**manager name**

-----  
A  
A

$$e_1.mgr = e_2.empno$$

**emp e1 [EMPLOYEE]**

empno	ename	job	sal	mgr
1001	A	MANAGER		
1002	B	CLERK		1001
1003	C	SALESMAN		1001

**emp e2 [MANAGER]**

empno	ename	job	sal	mgr
1001	A	MANAGER		
1002	B	CLERK		1001
1003	C	SALESMAN		1001

$e_1.ename = e_2.empno$

$\frac{B}{\boxed{1001}}$

$e_2.ename = Manager$

$\frac{A}{\boxed{1001}}$



```
Select e1.ename EMPLOYEE, e2.ename MANAGER
FROM emp e1, emp e2
WHERE e1.mgr = e2.empno;
```

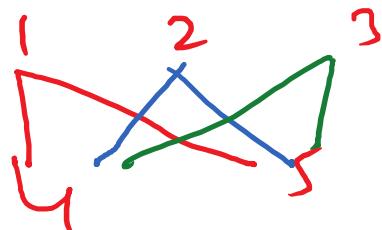
```
Select e1.ename EMPLOYEE, e1.sal EMP_SAL,
e2.ename MANAGER, e2.sal MAN_SAL
FROM emp e1, emp e2
WHERE e1.mgr = e2.empno
```

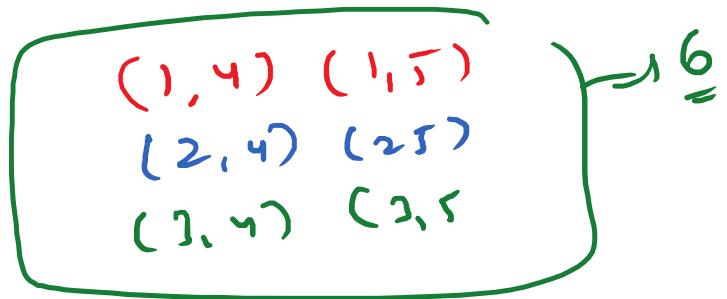
**Display the emp records who are getting salary more than their managers:**

```
Select e1.ename EMPLOYEE, e1.sal EMP_SAL,
e2.ename MANAGER, e2.sal MAN_SAL
FROM emp e1, emp e2
WHERE e1.mgr = e2.empno and e1.sal>e2.sal;
```

**CROSS JOIN / PRODUCT JOIN / CARTESIAN JOIN:**

$\text{A} = \{1,2,3\} \Rightarrow 3$   
 $\text{B} = \{4,5\} \Rightarrow 2$   
 $\text{AXB} = 6 \text{ combinations}$





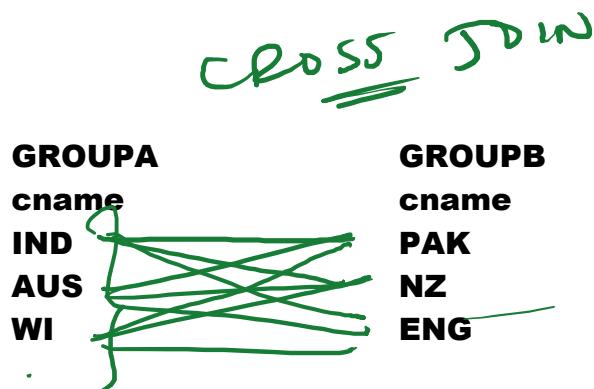
$$A = 3$$

$$B = 4$$

A CROSS JOIN B → 12 Rows

### CROSS JOIN:

**Each record in first table will be joined with each record in second table.**



**Select a.cname, b.cname from groupa a, groupb b;**

**Find dept wise sum of salaries & display along with dept names:**

<b>emp</b>			
<b>empno</b>	<b>ename</b>	<b>sal</b>	<b>deptno</b>

<b>Dept</b>		
<b>deptno</b>	<b>dname</b>	<b>loc</b>

```
Select d.dname, sum(e.sal)
from emp e, dept d
where e.deptno = d.deptno
group by d.dname;
```

dname	sum(Sal)
ACCOUNTING	..
RESEARCH	..
SALES	..

**Display location wise number of employees joined in different years:**

Loc	Year	Count(*)
CHICAGO	1980	3
	1981	2
NEW YORK	1980	5
	1981	4

```
Select d.loc, to_char(e.hiredate,'yyyy') year,
Count(*)
FROM emp e, Dept d
WHERE e.deptno=d.deptno
GROUP BY d.loc,to_char(e.hiredate,'yyyy');
```

## Sub Queries

Saturday, December 4, 2021 6:36 PM

### Sub Queries:

- Writing Query in another query is called "Sub Query / Nested Query".

### Syntax:

```
SELECT <column_list>
FROM <table_list>
WHERE <condition> <operator> (<Select statement>);
```

- Outside query is called "Outer Query / Main Query / Parent Query".
- Inner query is called "Inner Query / Sub Query / Child Query".
- When we don't know the exact value for filter condition in WHERE clause we write Sub Query.
- Inner Query must be "SELECT" statement only.
- Outer Query can be SELECT / INSERT/ UPDATE / DELETE.
- Inner query must be written parenthesis [ (Inner Query) ].
- Normally, First inner query gets executed. then outer query gets executed.
- Result of Inner Query becomes input for the Outer Query.
- We can write maximum of 255 levels of queries.

### Types of Sub Queries:

- Single Row Sub Query
- Multi Row Sub Query
- Correlated Sub Query
- Inline View
- Scalar Sub Query

- **Single Row Sub Query:**  
**If sub query returns single row then it is called "Single Row Sub Query".**

**Display the emp records whose salary is greater than BLAKE's salary:**

```
Select * from emp  
WHERE sal > (Select sal from emp  
where ename='BLAKE');
```

**Display the employee records who are junior to BLAKE [OR]  
Display the employee records who joined after BLAKE:**

```
Select * from emp  
where hiredate > (Select hiredate from  
emp where ename='BLAKE');
```

**Display the emp records who are senior to BLAKE [OR]  
Display the emp records who joined before BLAKE:**

```
Select * from emp  
where hiredate < (Select hiredate from  
emp where ename='BLAKE');
```

**Find max salary in all employees:**

```
Select max(Sal) from emp;
```

**Display the emp name who is earning max salary:**

```
Select * from emp
```

**where sal = (Select max(Sal) from emp);**

**Display the emp name who is senior in all employees:**

**Select ename,hiredate from emp  
where hiredate = (Select min(hiredate)  
from emp);**

**Display the emp name who is junior in all employees:**

**Select ename from emp  
where hiredate = (Select max(hiredate)  
from emp);**

**Find second max salary:**

<b>sal</b>	<b>Select max(sal) from emp where sal&lt;(Select max(Sal) from emp);</b>
-----	
4000	
5000	
4500	
3000	
2500	
3000	
6000	
5500	

**Find the emp name who is earning second max sal:**

**Select ename from emp  
WHERE sal = (Select max(Sal) from emp  
where sal<(Select max(Sal) from emp);**

- **Find third max salary**
- **Find the emp name who is earning third max salary**

**Delete an employee record who has maximum experience:**

**Find max experienced person hiredate:**

**Select min(hiredate) from emp;**

**Delete from emp  
where hiredate = (Select min(hiredate) from emp);**

**Update emp sal as deptno 30's max salary to the empno 7499:**

**Update emp set sal = (Select max(sal)  
from emp where deptno=30)  
where empno=7499;**

**Write a query to swap 2 salaries of the empnos 7499 and 7521:**

**Update emp set sal =  
Case empno  
When 7499 Then (Select sal from emp where empno=7521)  
When 7521 Then (Select sal from emp where empno=7499)  
End  
Where empno in(7499,7521);**

**Display dept wise sum of salaries:**

```
Select deptno,sum(Sal)
from emp
group by deptno;
```

**Display the deptno which is spending max amount on their employees:**

```
Select deptno from emp
group by deptno
having sum(Sal)= (Select max(sum(Sal))
from emp
group by deptno);
```

**Display the dept name which is spending max amount on their emps:**

```
Select dname from dept
where deptno=(Select deptno from emp
group by deptno
having sum(Sal) = (Select max(sum(Sal))
from emp group by deptno))
```

**Display the deptno which is having max number of employees:**

```
Select deptno from emp
group by deptno
having count(*) = (Select max(count(*))
```

```
from emp group by deptno);
```

**Display the dept name which is having  
max number of emps:**

```
Select dname from dept where  
deptno=  
(Select deptno from emp  
group by deptno  
having count(*) =  
(Select max(count(*)) from emp  
group by deptno));
```

**Multi Row Sub Queries:**

- If sub query returns multiple rows then  
it is called "Multi Row Sub Query".**

**Display the emp records whose job titles  
are same as SMITH and BLAKE:**

```
Select * from emp  
where job in(Select job from emp  
where ename in('SMITH','BLAKE') );
```

**Display the emp records whose sal is  
equals to WARD & SCOTT salary:**

<b>WARD</b>	<b>1437.5</b>
<b>MARTIN</b>	<b>1437.5</b>
<b>FORD</b>	<b>3600</b>

<b>Select * from emp where sal in( Select sal from emp where ename in('WARD','SCOTT') );</b>	<b>SCOTT</b>	<b>3600</b>
--	--------------	-------------

**Display the emp names who are earning max salary in their depts:**

```
Select ename,deptno from emp  
where (deptno,sal) in(  
Select deptno, max(Sal) from emp  
Group By deptno);
```

### **Correlated Sub Query:**

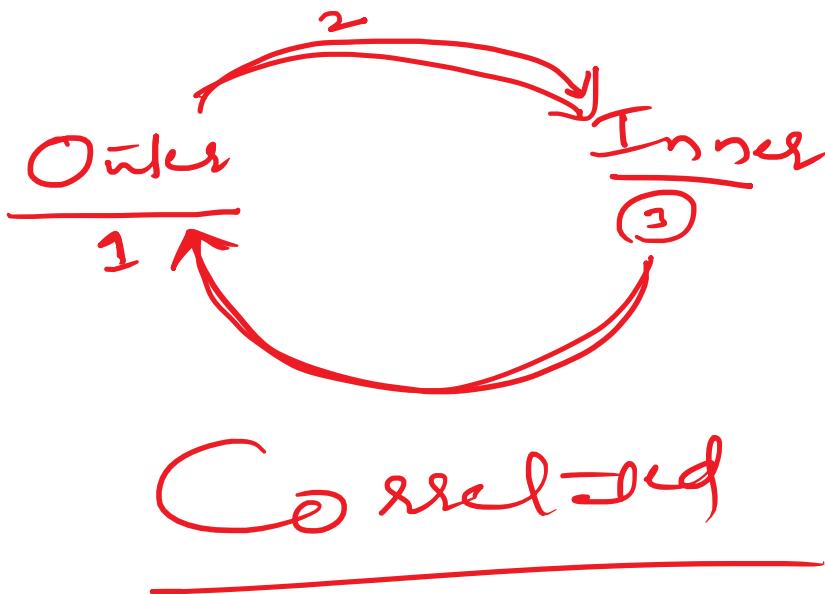
- If outer query passes value to inner query then it is called "Correlated Sub Query".**
- Generally, Inner query gets executed first. Then outer gets executed.**
- Inner query gets executed only once.**

**Ex:**

```
Select * from emp  
where sal > (Select sal from emp  
where ename='BLAKE');
```

- In Correlated Sub Query, First Outer query will be executed. Then inner query will be executed.**
- Inner query gets executed for many times.**

- **number of rows passed to inner query from outer query = number of execution times of inner query.**



### **Execution process of Correlated**

#### **Sub Query:**

- **First outer query gets executed.**
- **Outer query passes value to inner query.**
- **Inner query gets executed.**
- **Inner query gives value to outer query.**
- **Outer query condition will be tested. Based on condition record will be retrieved.**
- **Above steps will be executed repeatedly.**

**Display the emp records whose salary is greater than their dept's avg salary:**

**emp**

empno	ename	sal	deptno
1001	A	6000	10
1002	B	10000	20
1003	C	8000	10
1004	D	12000	20

deptno	avg(Sal)
10	7000
20	11000

**Select empno,ename,sal,deptno from emp e  
where sal > (Select avg(sal) from emp  
where deptno=e.deptno);**

10	C	8000
20	D	12000

**Display the top 3rd salary:  
[Find 3rd max salary]:**

**emp**

empno	ename	sal	deptno
1001	A	6000	10
1002	B	10000	20
1003	C	8000	10
1004	D	12000	20

**Find no of emps > his sal**

**0 emps > emp sal => 1st max sal  
1 emp >emp sal => 2nd max sal  
2 emps > emp sal => 3rd max sal**

**max salary:**

**Select distinct sal  
from emp e  
where 0 = (Select  
count(distinct Sal) from emp**

```
where sal>e.sal);
```

### **Second Max salary:**

```
Select distinct sal  
from emp e  
where 1 = (Select  
count(distinct Sal) from emp  
where sal>e.sal);
```

### **Third max salary:**

```
Select distinct sal  
from emp e  
where 2 = (Select  
count(distinct Sal) from emp  
where sal>e.sal);
```

### **Query to find nth max salary:**

**&n => parameter**

```
Select distinct sal  
from emp e  
where &n-1 = (Select  
count(distinct Sal) from emp  
where sal>e.sal);
```

### **Inline View:**

- Writing Sub Query in "FROM" clause is**

**called "Inline View".**

- **Sub query acts like table.**
- **Using this, we can change order of execution.**

**Display the emp records whose annual salary is > 30000:**

```
Select * FROM
(Select empno,ename,sal, sal*12 an_sal
FROM emp) e
WHERE an_sal>30000;
```

**FROM  
WHERE  
GROUP BY  
HAVING  
SELECT  
ORDER BY**

**Display the top 3 salaries:**

```
Select * from (Select empno,ename,sal,
dense_rank() over(order by sal desc) rnk
from emp) e
where rnk<=3;
```

**Rownum:**

- **is a pseudo column.**
- **to apply row numbers we use it.**
- **On result of select query row**

**pseudo => false**

**numbers will be applied**

**Display first 3 records:**

**Select rownum,empno,ename,sal  
from emp where rownum<=3;**

**Display 3rd record:**

**Select rownum,empno,ename,sal from emp  
where rownum=3;**

**no rows selected**

**Select \* from  
(Select rownum rn, empno,ename,sal  
FROM emp) e  
where rn=3;**

**Display 5th,10th and12th records:**

**Select \* FROM  
(Select rownum  
rn,empno,ename,sal from emp) e  
WHERE rn in(5,10,12);**

**Display the emp records whose  
row numbers are b/w 6 to 10:**

```
Select * FROM
(Select rownum
rn,empno,ename,sal FROM emp) e
WHERE rn between 6 and 10;
```

**even => 2,4,6,8 ... => divide with 2 => remainder 0**

**odd => 1,3,5,7 .. => divide with 2 => 1**

**mod(rn,2)=0 => even record**  
**mod(rn,2)=1 => odd record**

```
Select * FROM
(Select rownum rn,
empno,ename,sal from emp) e
WHERE mod(rn,2)=0;
```

**Display odd number records:**

```
Select * FROM
(Select rownum rn,
empno,ename,sal from emp) e
WHERE mod(rn,2)=1;
```

### **Scalar Sub Query:**

- Writing sub query in **SELECT clause** is called "**Scalar Sub Query**".
- In this, Sub Query acts like a column.

**Display the number of records in emp & dept tables:**

```
Select (Select count(*) from emp) emp,
(Select count(*) from dept) dept
FROM dual;
```

<b>emp</b>	<b>dept</b>
-----	-----
<b>14</b>	<b>4</b>

**Display the share of each dept in salaries:**

```
Select deptno, sum(Sal),
(Select sum(Sal) from emp) total_sal,
Round((sum(Sal)/(Select sum(Sal) from emp))*100,1) per
FROM emp
group by deptno;
```

**Single Row Sub Query**  
**Multi Row Sub Query**

**Correlated Sub Query**

**Inline View**

**Scalar Sub Query**

**Operators:**

**Exists**

**Not Exists**

**Any**

**All**

**Exists =>**

**returns true if record is existed.**

**return false when record is not existed**

**Not Exists =>**

**returns true if record is not existed.**

**returns false if record is existed.**

**Display the dept names which are**

**having employees:**

**Correlated Sub Query:**

```
Select dname from dept d  
where exists(Select * from emp  
where deptno=d.deptno);
```

**dept**

<b>deptno</b>	<b>dname</b>
---------------	--------------

**emp**

<b>empno</b>	<b>ename</b>	<b>deptno</b>
--------------	--------------	---------------

dept	
deptno	dname
10	ACCOUNT
20	RESEARCH
30	SALES
40	OPERATION

emp		
empno	ename	deptno
1		10
2		20
3		30

**Display the dept names which are not having employees:**

**Select dname from dept d  
WHERE not exists(select \* from emp  
where deptno=d.deptno);**

**Display the emp names who are having subordinates:  
Correlated Sub Query:**

**Select ename from emp e  
where exists(Select \* from emp  
where mgr=e.empno);**

**Display the emp names who are not having subordinates:**

**Select ename from emp e  
where not exists(Select \* from emp  
where mgr=e.empno);**

=

**IN**

**ename = 'BLAKE'**

**ename IN('BLAKE','SMITH','ALLEN')**

**for single value  
comparison**

**single equality value**

**for multi value comparison**

**Multi equality condition**

> <

**can compare with one value**

**ANY  
ALL**

**can compare with multiple  
values  
for multi greater than,  
multi less than conditions  
we use ANY, ALL**

**ALL => acts like AND  
ANY => acts like OR**

**Select \* from emp  
where sal > All(2000,3000);**

**displays the records whose sal  
is >3000**

**Select \* from emp  
where sal>Any(2000,3000);**

**displays the emp records whose  
salary is greater than 2000**

**Multi Row Sub Queries:**

**WARD  
MARTIN**

**display the emp records  
whose salary is equals  
to WARD & SCOTT salary:**

**FORD  
SCOTT**

**Select \* from emp where  
sal in(Select sal from emp  
where ename in('WARD','SCOTT'));**

**display the emp records whose  
salary is > WARD & SCOTT salary:**

**Select \* from emp  
where sal > All(Select sal from emp  
where ename IN('WARD','SCOTT'));**

**display the emp records whose salary is >  
WARD or SCOTT salary:**

```
Select * from emp  
where sal > Any(Select sal from emp  
where ename IN('WARD','SCOTT'));
```

## **Tables**

### **SQL**

**DDL DML DCL TCL ACL**

### **Built-In Functions**

### **Clauses in SELECT**

#### **Joins**

#### **Set Operators**

#### **Sub Queries**

### **SQL**

#### **Tables**

#### **Views**

#### **Indexes**

#### **Sequences**

#### **Synonyms**

#### **Materialized View**

### **PL/SQL**

**Windows 10, Windows 8 => Oracle 19c  
Minimum RAM => 4GB**

**Windows 7 => Oracle 11g  
RAM => 2GB**

[www.oracle.com](http://www.oracle.com)

**products**

**Oracle db**

**username: sys as sysdba**

**password:**

**Changing DBA password:**

**SQL> Alter user system identified by naresh;**

**Changing User Password:**

**SQL> Alter user c##batch7am identified by  
naresh;**

**edelivery.oracle.com**

**Constraints in SQL:**

**Constraint: is a rule applied on a column**

**Primary Key**

**Unique**

**Not Null**

**Check**

**Default**

**References [Foreign Key]**

**Primary Key => should not accept  
duplicate & null values.**

**Not Null => does not accept null  
values. But it accepts duplicate  
values**

**Unique => does not accept  
duplicate values. but accepts null  
values**

**Check => to write our own  
conditions**

**Default => to apply default value to  
a column**

**References [Foreign Key]**

**It accepts PK values of another  
table**

## **Examples on Constraints:**

**Std**

<b>sid</b>	<b>sname</b>	<b>M1</b>
------------	--------------	-----------

**sid => Number(4) => Primary Key**

**sname => Varchar2(10) => Not Null**

**M1 => Number(3) => Check => b/w 0 to 100**

**Create Table Std**

```
(  
  sid Number(4) Primary Key,
```

## Views

Thursday, December 9, 2021 6:18 PM

### Views:

- View is a Database Object.
- View is a virtual table.
- Virtual Table means, a view does not contain physical data & it does not occupy the memory.
- A view holds SELECT query.
- When we retrieve data through view, implicitly it runs SELECT query.
- A view is created on table.
- A table on which a view is created is called "Base Table".
- A view always gives recent data [committed data].
- Some views can be updated. These are called Updatable Views [Simple Views].  
Some views cannot be updated. These are called "Read-Only Views" [Complex Views].

### Syntax to create a View:

```
Create View <view_name>
As
<Select Query>;
```

Ex:

```
Create view V1
As
Select empno,ename,job,deptno
FROM emp;
```

BASE TABLE



INSERT

```
Select * from v1;
```

empno	ename	job	deptno
-------	-------	-----	--------

**Advantages:**

- Provides Security
- Reduces the complexity & simplifies the queries.

**In Oracle we can provide security  
in 3 levels:**

- DB Level => SCHEMA [USER]
- Table Level => Grant, Revoke
- Data Level => View

**emp**

empno	ename	job	sal	mgr	hiredate	deptno
-------	-------	-----	-----	-----	----------	--------

**Column Level Security:**

```
Create view v1
As
Select empno, ename, job,deptno
from emp;
```

**On this view we give permission to other user. That user works with this view to work with the emp data.**

```
Select * from v1;
```

**Row Level Security:**

**emp**

empno	ename	job	sal	hiredate	mgr	deptno
					10	
					10	
					20	
					20	
					30	
					30	

**We want to retrieve data from 10 tables**

## **9 Join Conditions**

**Create view v1**

**As**

**JOIN QUERY ;**

**Select \* from v1;**

**JAVA developers  
wants to retrieve data from  
10 tables**

**Types of Views:**

**2 Types:**

- **Simple View**
- **Complex View**

**Simple View:**

- **If a view is created based on one table then it is called "Simple View".**
  - **It can be also called as "Updatable View".**
- Because, we can perform DML operations through the view.**

**Granting permission to create the view:**

**Log In as DBA:**

**username => system**

**password => nareshit**

**Grant Create View to c##oracle6pm;**

**Implementing Column Level Security through view:**

**emp**

<b>empno</b>	<b>ename</b>	<b>job</b>	<b>sal</b>	<b>hiredate</b>	<b>deptno</b>	<b>comm</b>	<b>mgr</b>
--------------	--------------	------------	------------	-----------------	---------------	-------------	------------

**create a view with empno,ename,job & deptno columns:**

**C##ORACLE6PM:**

**Create View v1**

**As**

**Select empno, ename, job, deptno  
FROM emp;**

**Create a user:**

**Log in as DBA**

**username: system**

**password:nareshit**

**Create user c##ramu identified by ramu  
default tablespace users  
quota unlimited on users;**

**Grant connect, resource to c##ramu;**

**Log in as c##oracle6pm:**

**Give permission to c##ramu user on v1 view:**

**Grant Select, Insert,Update, Delete on v1  
to c##ramu;**

**Example to implement Row Level Security:**

**emp**

empno	ename	job	sal	deptno	mgr	hiredate	comm
			10				
			10				
			20				
			20				
			30				

**Create a view on deptno 20 records on that give  
permission to 20th dept manager:**

**Create view v2**

**As**

**Select \* from Emp  
where deptno=20;**

**Grant all on v2 to c##ramu;**

**emp**

<b>empno</b>	<b>ename</b>	<b>job</b>	<b>sal</b>	<b>deptno</b>	<b>mgr</b>	<b>hiredate</b>	<b>comm</b>
			10				
			10				
			20				
			20				
			30				

**v2**

<b>empno</b>	<b>ename</b>	<b>job</b>	<b>sal</b>	<b>deptno</b>	<b>mgr</b>	<b>hiredate</b>	<b>comm</b>
			20				
			20				

**Insert into c##oracle6pm.v2(empno,ename,deptno)  
values(1,'A',20);**

**Insert into c##oracle6pm.v2(empno,ename,deptno)  
values(1,'A',10);**

**This record will be inserted. But, this record will not  
be displayed through the view. Because, view  
created on deptno 20 only.**

**With Check Option:**

- "Create Or Replace" is used to make any changes to existing view.
- We cannot use "Alter" command on view.

**Create Or Replace v2****As****Select \* from emp****where deptno=20****With Check Option;**

- "With Check Option" does not allow the records which cannot be displayed by the view.
- "Where" Condition violated records will not be accepted.

### **Complex View:**

- If a view is created based on joins, group by, having, aggregate functions, set operators, expressions or sub queries then it is called "Complex View".
- We cannot perform DML operations on Complex View.
- It can be also called as "Read-Only View".

### **Create view v3**

**As**

```
Select e.empno, e.ename,e.job,e.deptno,  
d.dname,d.loc  
FROM emp e INNER JOIN dept d  
ON e.deptno=d.deptno;
```

### **Differences b/w Simple View & Complex View:**

<b>Simple View</b>	<b>Complex View</b>
<ul style="list-style-type: none"><li>• it is created based on one table</li><li>• We can perform DML operations on it</li><li>• It can be also called as Updatable View</li><li>• it performs simple operations</li></ul>	<ul style="list-style-type: none"><li>• it is created based on multiple tables.</li><li>• We cannot perform DML operations on it</li><li>• It can be also called as Read-Only View.</li><li>• It performs complex operations like Joins, Group By, Set Operators, Aggregate functions, expressions, sub queries.</li></ul>

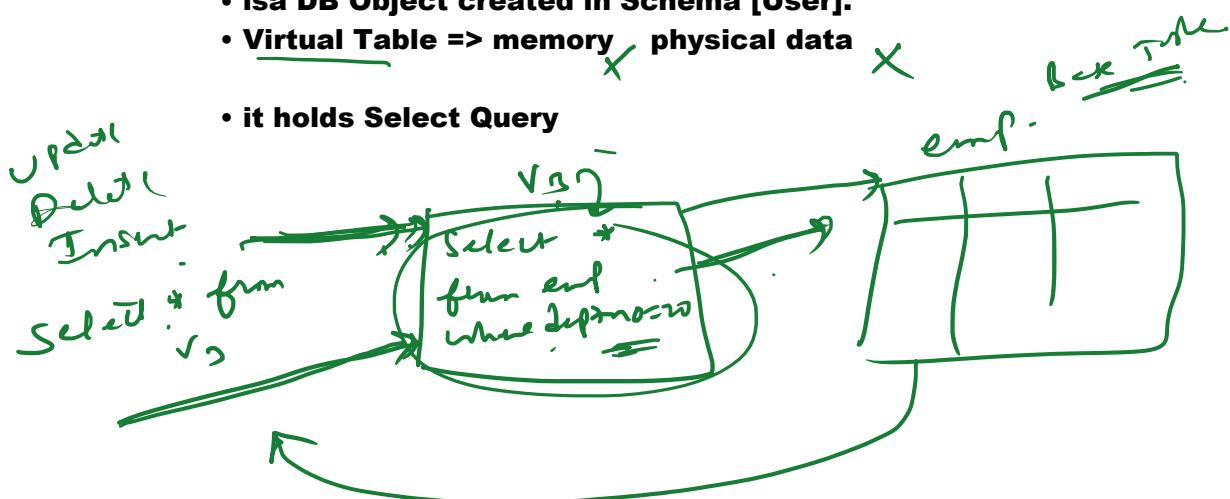
### **View:**

- is a DB Object created in Schema [User].
- Virtual Table → memory → physical data

*AC*

### **View:**

- is a DB Object created in Schema [User].
- Virtual Table => memory ~~physical data~~
- it holds Select Query



### **2 Types**

**Simple View**

**Complex View**

**Can we create a view from another view? Yes. We can.**

**Create view v4**

**As**

**Select ename,dname from v3;**

**After creating the view if we add a column to base table does it reflect to view?**

**No. Because, view holds select query. it runs select query.**

v2
Select * from emp

emp				
empno	ename	job	sal	deptno

**runs select query.**

**v2**

**Select \* from emp  
where deptno=20;**

**Select \* from v2;**

**emp**

empno	ename	job	sal	deptno
-------	-------	-----	-----	--------

## **User\_VIEWS**

**it maintains all view information**

**After creating view if we add  
record to base table does it reflect  
to view?  
Yes. It will display.**

**Can we create a view without table?  
No. But there is a concept FORCE VIEW.**

### **FORCE VIEW:**

**A view which is created forcibly with some  
compilation errors is called "FORCE VIEW".**

**Without existing table we can create FORCE  
VIEW. But, this view will not work until table  
is created.**

**Can we use Alter or Truncate command on view?  
No**

**How can make changes in existing view?  
Create Or Replace View**

**If delete base table does it delete view?  
No. But, these views will not work until you  
recreate those tables.**

**User\_VIEWS:**

**All views information stored in "USER\_VIEWS"**

**Select view\_name, Text from User\_VIEWS;**

## Sequences

Saturday, December 11, 2021 6:24 PM

### Sequence:

- Sequence is a DB Object.
- Sequence is independent of table.
- Sequence is used to generate sequential integers.

**Ex: Order\_ID      Transaction\_ID**

### Syntax to create the Sequence:

```
Create Sequence <sequence_name>
[Increment By <value>]
[Start With <value>]
[MINVALUE <value>]
[MAXVALUE <value>]
[cycle/nocycle]
[cache <size>/nocache];
```

**Create Sequence s1;**

Clause	Default Value
Start With	1
Increment By	1
MINVALUE	1
MAXVALUE	10 power 28
cycle	nocycle
cache	20

**Sequence provides 2 pseudo columns:**

- **NEXTVAL**
- **CURRVAL**

**sequence\_name.pseudo\_column**

**s1.nextval**  
**s1.currval**

**NextVal => returns next value in the sequence**

**CurrVal => returns current value in the sequence**

**Create Sequence s2**

**Start with 1**

**Increment By 1**

**MINVALUE 1**

**MAXVALUE 5;**

**Course**

<b>cid</b>	<b>cname</b>
<b>10</b>	<b>Java</b>
<b>20</b>	<b>Python</b>
<b>30</b>	<b>Oracle</b>
<b>40</b>	<b>HTML</b>
<b>50</b>	<b>C++</b>

**Create Sequence s3**

**START WITH 10**

**Increment By 10**

**MINVALUE 10**

**MAXVALUE 50;**

**cycle / nocycle:**

**cycle:**

**Create Sequence s4**

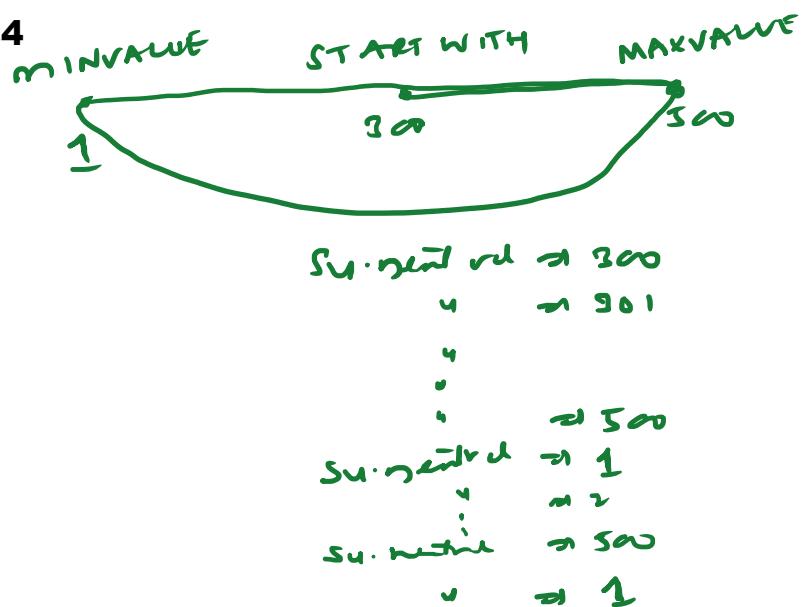
**Start with 300**

**Increment By 1**

**MINVALUE 1**

**MAXVALUE 500**

**cycle;**



- If we create a sequence with "CYCLE", sequence starts from START WITH value. Generates next value up to MAXVALUE. After reaching MAXVALUE it will be reset to MINVALUE.

**nocycle:**

**Create Sequence s4**

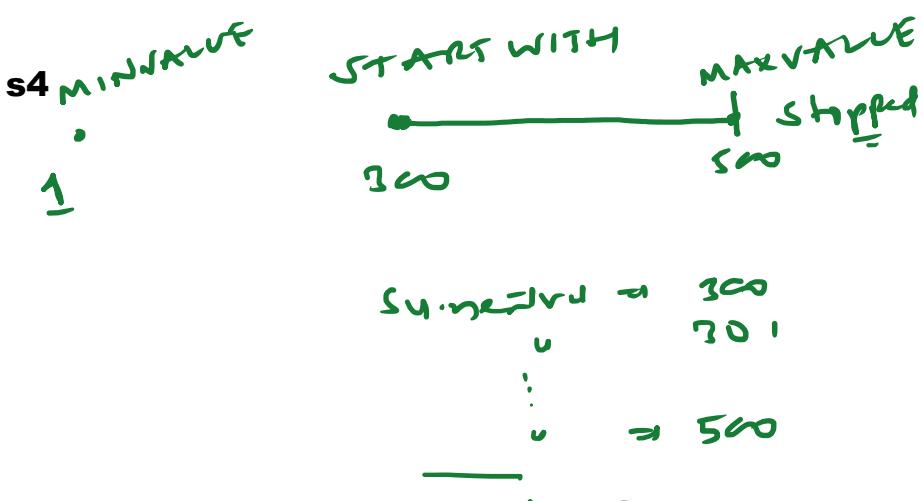
**Start with 300**

**Increment By 1**

**MINVALUE 1**

**MAXVALUE 500**

**nocycle;**



Stop

If we create a sequence with nocycle, sequence will be started from START WITH value, generates next value up to MAXVALUE. After reaching MAXVALUE it will be stopped.

**Create Sequence s5**

**Start With 3**

**default cache size =20**

**Increment By 1**

**MINVALUE 1**

**Cache size must be less than 1 cycle**

**MAXVALUE 5**

**cycle; //ERROR**

**Create Sequence s5**

**Start With 3**

**Create Sequence s5**

**Start With 3**

**Increment By 1**

**Increment By 1**

**MINVALUE 1**

**MINVALUE 1**

**MAXVALUE 5**

**MAXVALUE 5**

**cycle**

**cycle**

**cache 5; //ERROR**

**cache 4; //VALID**

**cache <size> / nocache:**

- Default cache size is 20.
- Always cache size must be less than one cycle.

```
Create Sequence s6
Start With 101
Increment By 1
MAXVALUE 1000
nocache;
```

**When sequence created with nocache, when we call sequence, it goes to DB, identifies current value, adds increment by value. returns value to sequence call.**  
**For every call it goes to DB & performs above process. It degrades the performance.**

**TO improve the performance use CACHE**

**Cache improves the performance.**

```
Create Sequence s6
Start With 101
Increment By 1
MAXVALUE 1000
cache 100;
```

**If we create a sequence with cache size 100, 100 values will be loaded into cache memory.**  
**For every sequence call it will not go to DB. It returns from cache. It improves the performance.**

**Can we call sequence from Create Command?**

**From Oracle 12c version onwards it is possible.**

```
Create Sequence s7
Start with 1
```

```
Increment by 1  
maxvalue 10;
```

```
Create table customer33(  
cid number(4) default s7.nextval,  
cname varchar2(10));
```

### **Altering Sequence:**

```
Alter Sequence s7 MAXVALUE 10;
```

```
Alter Sequence s7 cycle; //ERROR
```

```
Alter Sequence s7 cache 9 cycle;
```

**From Oracle 12c version onwards we can generate sequential numbers using 2 ways:**

- Using Sequence**
- Using Identity [generated always as identity]**

### **Generating sequential numbers Using Identity:**

```
Create Table customer44(  
cid number(4) generated always as identity,  
cname varchar2(10));
```

**Sequence provides more flexibility than Identity.  
We can access currval in sequence whereas it is not possible with**

**identity.**

**We can specify start with value for the sequence. from any number we can start. But identity always starts from 1.**

**we can specify increment by value using sequence whereas it is not possible for identity.**

**Calling sequence from "Update" command:**

**Create sequence s8  
Increment By 1  
Start with 1001  
MAXVALUE 5000;**

**Update emp set empno=s8.nextval;**

**TO make empno in the sequence we are calling sequence in UPDATE command.**

**Dropping the sequence:**

**Drop sequence s8;**

**User\_Sequences:**

**It maintains information about all sequences.**

**Select \* from user\_Sequences;**

## MATERIALIZED VIEW

Monday, December 13, 2021 6:48 PM

### Person

pid	pname	city	state	aadhar_number
-----	-------	------	-------	---------------

**Create view v1**

**As**

**Select state, count(\*) no\_of\_people  
from person  
group by state;**

State	no_of_people
-------	--------------

**Select \* from v1;**

### Disadvantage of view:

- Less Performance.

### Materialized View:

- is a DB Object.
- It is not virtual table. It means, It contains physical data & it occupy the memory.
- It holds result of SELECT query whereas VIEW holds SELECT query.
- It is mainly used in DWH (Data Ware Housing) to maintain summarized data physically.
- It holds precomputed result.
- Materialized view must be refreshed. because, it holds precomputed result.

### Advantages:

- It improves the performance.
- It can be used to maintain local copy of remote database.

### Syntax:

### Syntax:

```
Create Materialized View <name>
As
<Select Query>;
```

**Create view v1**

```
As
Select deptno,
sum(Sal) sum_of_sal
from emp
group by deptno;
```

**v1**

```
Select deptno,
sum(Sal) ....
.....;
```

```
Select * from v1;
5times sum(Sal)
will be calculated
```

**5 times calculated**

**performance will be  
degraded**

**Create materialized view mv1**

```
As
Select deptno,
sum(Sal) sum_of_Sal
from emp
group by deptno;
```

**mv1**

deptno	sum_of_Sal
10	..
20	

```
Select * from mv1;
//calculates sum of salaries
```

```
Select * from mv1; //retrieves
from mv1
```

```
Select * from mv1; //retrieves
from mv1
```

**one time calculated.  
performance will be  
improved/**

**Log in as DBA:  
username: system  
password: nareshit**

**Grant create materialized view to  
c##oracle6pm;**

**Log in as c##oracle6pm:**

**Create Materialized view mv1  
As  
Select deptno, sum(Sal) sum\_of\_sal  
from emp  
group by deptno;**

**After creating view if we add records to base  
table does it reflect to view?  
Yes. Because when we retrieve data through  
view it runs select query. it always gives  
committed data.**

emp				
empno	ename	job	sal	deptno
1001				
1002				
1003				
1004			3000	10
1005			6000	20

**Create view v1  
as  
select deptno,  
sum(Sal) sum\_of\_Sal  
from emp  
group by deptno;**

**Create materialized view mv1  
as  
select deptno,  
sum(Sal) sum\_of\_Sal  
from emp  
group by deptno;**

**v1**

**SELECT query => calc**



**mv1**

deptno	sum_of_Sal
10	..
20	..

**Select \* from mv1;**

### **Refreshing Materialized View:**

- Applying changes of base table to materialized view is called "Refreshing Materialized view".
- The changes made on base table will not be applied to materialized view. that is why we need to refresh the materialized view.

**There are 3 ways to refresh the materialized view.**

- On Demand [Default]
- On Commit
- On Time interval

#### **• On Demand [Default]:**

In this way we need to call the procedure "refresh()" procedure which is available in "dbms\_mview" package.

**dbms\_mview package**

```
procedure refresh(..)
IS
BEGIN
...
END;
```

### **Syntax to call packaged procedure:**

### **Syntax to call packaged procedure:**

```
Execute package_name.procedure_name;
```

```
SQL> Execute dbms_mview.refresh('MV1');
```

**materialized view will be refreshed. It means,  
changes of base table will be applied to  
materialized view.**

#### **On Commit:**

- Materialized view will be refreshed after  
execution of COMMIT.**

#### **Create Materialized View MV5**

**refresh on commit**

**As**

```
Select deptno,sum(sal) sum_of_Sal  
from emp  
group by deptno;
```

#### **On Time Interval:**

**Based on regular interval of time materialized view  
will be refreshed.**

**Ex: monthly once for "MONTH\_SALES"**

**weekly once for "WEEK\_SALES"**

**daily once for "DAY\_SALES"**

#### **Create Materialized view MV6**

**refresh force**

**start with sysdate next sysdate+interval '2' day**

**As**

```
Select deptno,sum(Sal) from emp  
group by deptno;
```

**With above query for every 2 days materialized view will be refreshed**

**Refresh types:**

**3 types:**

- **Complete**
- **Fast**
- **Force**

<b>Complete</b>	<b>It truncates all records from materialized view. runs the query. and fills new result in materialized view</b>
-----------------	---

<b>emp BASE TABLE</b>				
<b>empno</b>	<b>ename</b>	<b>job</b>	<b>sal</b>	<b>deptno</b>
1001			10	
1002			10	
1003			20	
1004			20	
1005			10	
1006			20	

→

<b>MV1</b>	
<b>deptno</b>	<b>sum(Sal)</b>
10	...
20	...

**Create materialized view mv8  
refresh complete  
on commit  
as  
select deptno,sum(Sal)  
from emp group by deptno;**

**FAST** In FAST refresh, the changes made from last refresh to till now will be applied to materialized view. To perform FAST refresh materialized view log file required. If this file is not there it will be failed.

### emp BASE TABLE

empno	ename	job	sal	deptno
1001				10
1002				10
1003				20
1004				20
1005			10	
1006			20	

### MV1

deptno	sum(Sal)
10	...
20	...

### materialized view log file

records  
changes made  
on base table

```
Create materialized view mv8
refresh fast
on commit
as
select deptno,sum(Sal)
from emp group by deptno;
```

**DBA:**

**Create materialized view log on c##oracle6pm.emp;**

**FORCE:**

**FORCE** It depends on ORACLE. Oracle gives first priority for FAST refresh. If materialized view log file is not

**there. then it performs COMPLETE refresh**

### **Enable Query rewrite clause:**

**when we create materialized view using  
"ENABLE QUERY REWRITE" even if we submit  
normal query data will be retrieved from  
MATERIALIZED VIEW.**

### **Java Developer**

```
Select deptno,sum(sal)  
from emp  
group by deptno;
```

```
Select * from MV12;
```

**Set autotrace on explain**

**// It shows execution plan**

```
Create materialized view mv12  
refresh on commit  
Enable query rewrite  
As  
Select deptno,sum(Sal)  
from emp  
group by deptno;
```

**Dropping materialized view:**

**Drop materialized view MV12;**

**user\_mvviews:**

**It maintains information about all  
materialized views.**

**Select MVVIEW\_NAME  
from user\_mvviews;**

**Materialized view:**

- **DB Object**
- **Not virtual table**
- **to improve the performance**

## **Indexes**

Tuesday, December 14, 2021 7:16 PM

### **Indexes:**

- **Index is a DB object.**
- **It is used to improve the performance of data retrieval.**
- **To refer a chapter in book, we use index. With this we can refer that page faster. Similarly If we create index data accessing performance will be improved.**
- **Indexes are created on columns which are frequently used in WHERE clause or in Join Operation.**

### **Syntax to create Index:**

```
Create Index <Index_Name>
On <table_name>(<column_list>);
```

```
Create Index I1 On emp(sal);
```

**When we submit a query ORACLE performs any one of the 2 types of scans:**

- **Table Scan**
- **Index Scan**

### **Indexes:**

- **Index is a DB Object.**
- **Index is used to improve the performance of data retrieval.**
- **Index will be created on columns which are frequently used in where condition or join operation.**

**When we submit a query to ORACLE, it performs any one of the 2 scans:**

- **Table Scan**
- **Index Scan**

**If Index is not created on the column, ORACLE performs Table Scan.**

**If Index is created on column, ORACLE performs Index Scan.**

**Index Scan works faster than Table Scan.**

**Syntax to create the Index:**

```
Create Index <Name> On  
<table_name>(<column_list>);
```

**Select \* from emp where sal<3000; //Table Scan**

**Create Index I1 On emp(sal);**

**Select \* from emp where sal<3000; //Index Scan**

sal	
4500	→ 3000 - T
5000	→ 3000 - T
4000	→ 3000 - T
2500	→ 3000 - F
2000	→ 3000 - F
1800	F
1600	F
3000	F
1500	F
3000	F

where sal > 3000

10 records

10 → Comparisons

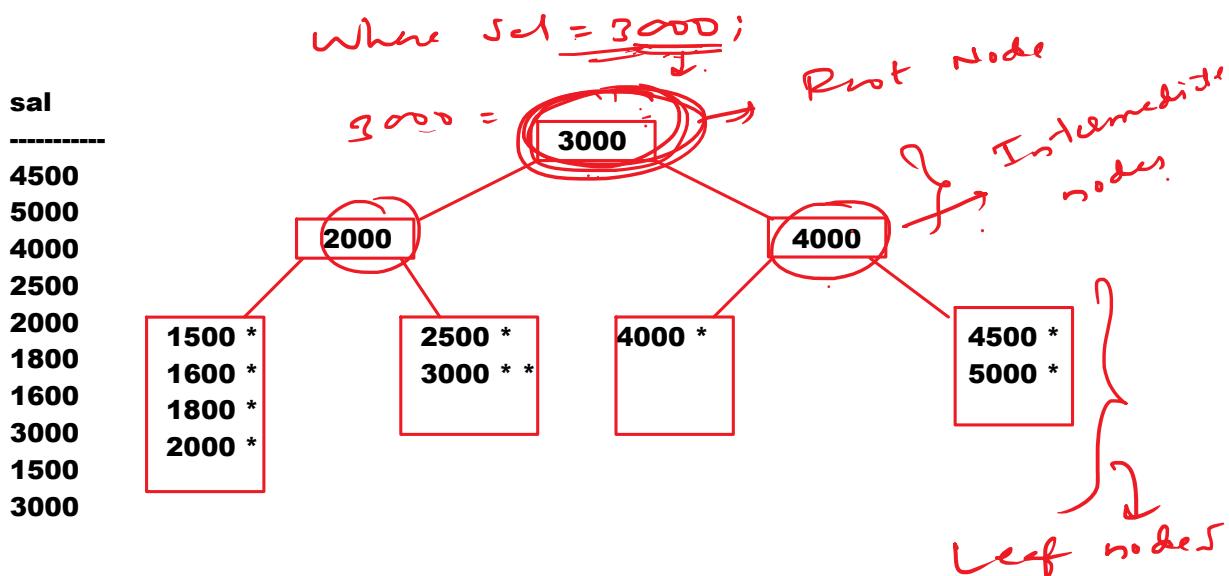
**In Table Scan, it compares filter condition with all values.  
No of comparisons = No of records**

**When we create the Index implicitly one structure will be created. It is called "BTREE STRUCTURE" [Balanced Tree Structure].**

**TREE => collection of nodes**

If value is less than or equals to root node, it will be placed at left side.

If value is greater than root node, it will be placed at right side.



**Root Node & Intermediate Node holds left pointer & right pointer**

**Leaf node holds data & row id.**

**Table Scan =>**

**No of comparisons = No of records**

**Index Scan =>**

**No of Comparisons <= half of the no of records**

### Types of Indexes:

- **BTREE Index / Normal Index**
  - **Simple Index**
  - **Composite Index**
  - **Unique Index**
  - **Function-Based Index**
- **Bitmap Index**

#### • **BTREE Index / Normal Index:**

In this, BTREE will be created when we create the index.

**Sub Types:**

**Simple  
Composite  
Unique  
Function-Based**

**Simple Index:**

**If index is created on one column we call it as  
Simple Index.**

**Ex:**

**Create Index I1 on emp(sal);**

**Composite Index:**

**If index is created on multiple columns then it is called "Composite Index".**

**Create an Index on deptno & job columns:**

**Create Index i2 on emp(deptno,job);**

**Before 9i:**

```
Select * from emp where deptno=10; // Index Scan
Select * from emp where job='MANAGER' //Table Scan
Select * from emp where deptno=10 and job='MANAGER'; //Index Scan
```

**From 9i version onwards:**

```
Select * from emp where deptno=10; //Index Range Scan
Select * from emp where deptno=10 and job='MANAGER';
//Index Range Scan
Select * from emp where job='MANAGER'; //Index skip scan
```

<b>10</b>	<b>CLERK</b>
<b>20</b>	<b>MANAGER</b>
<b>10</b>	<b>CLERK</b>
<b>30</b>	<b>CLERK</b>
<b>20</b>	<b>MANAGER</b>
<b>30</b>	<b>MANAGER</b>

**Unique index:**

**If index is created on a column which is having  
unique values then it is called "Unique Index".**

**Create Unique Index i4 on dept(Dname);**

**When we create a table with  
Primary Key implicitly index will be  
created on PK column.**

**To maintain unique values in a column  
we can use 3 ways:**

- Primary Key
- Unique
- Unique Index

```
create table inddemo1(
f1 number(4),
f2 varchar2(10));
```

**Function Based Index:**

If index is created based on function (or) expression then it is called "Function-Based Index".

**Select \* from emp where ename='SMITH'; // Table Scan**

**Create Index I7 on emp(ename);**

**Select \* from emp where ename='SMITH'; //Index Scan**

**Select \* from emp where lower(ename)='smith'; //Table Scan**

**Create Index I8 on emp(lower(ename)); //Function-Based Index**

**Select \* from emp where lower(ename)='smith'; // Index Scan**

**Expression:**

**column\_names + operators + numbers**

**ex: sal\*12**

**Select \* from emp where sal\*12>30000; //Table Scan**

**Create Index I9 on emp(sal\*12); //Function-Based Index**

**Select \* from emp where sal\*12>30000; //Index Scan**

**Btree Index:**

- **Btree index leaf node stores values and row ids**

**Bitmap Index:**

- **It stores bits 0s and 1s.**
- **In this every bit will be associated with Row ID.**
- **These bits will be converted to Row IDs & retrieves the records.**
- **It is created on low cardinality columns. It means, the columns which are having less distinct values on them we create bitmap index.**

**Low cardinality column:**

A column which has less distinct values is called "Low Cardinality Column".

**Ex: Gender Deptno Job**

Gender	Deptno	Job	distinct values	disstinct values
M			M	
F			F	
F				
M				
M				
F	10	MANAGER		10
M	10	CLERK		20
F	20	CLERK		30
	30			
	20			
	30			
	10			
	20			
	10			

**Job**

Job	distinct values
MANAGER	
CLERK	
CLERK	MANAGER
MANAGER	CLERK
CLERK	
MANAGER	

**High Cardinality Column:**  
**A column which has more distinct values is called "High Cardinality Column".**

**Ex: empno    ename**

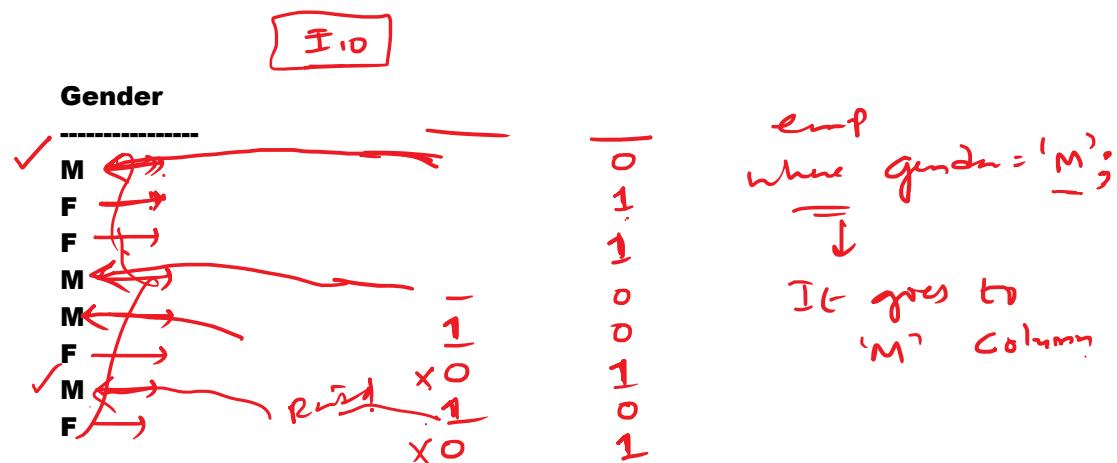
```
Create table emp11(
empno number(4),
ename varchar2(10),
gender char);
```

**Insert 10 records**

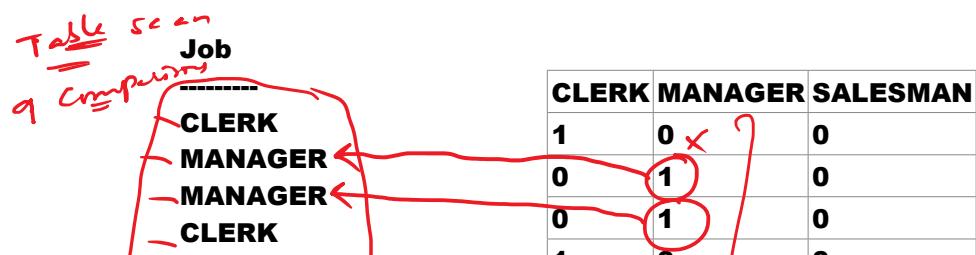
```
Select * from emp11 where gender='M'; //Table Scan
```

```
Create Bitmap Index I10 on emp11(gender);
```

```
Select * from emp11 where gender='M'; //Index Scan
```



```
Create Bitmap Index I11 on emp(job);
```





**In table scan, no of comparisons = no of records.**

**In Bitmap Index scan, no of comparisons = no of records. Then how it improves the performance?**

**Bit comparison is faster than value comparison. So it improves the performance.**

#### Differences b/w BTREE Index & Bitmap Index:

BTREE Index	Bitmap Index
<ul style="list-style-type: none"> <li>• It creates BTREE after creating the index.</li> </ul>	<ul style="list-style-type: none"> <li>• It will not create BTREE</li> </ul>
<ul style="list-style-type: none"> <li>• It stores value &amp; Row Id</li> </ul>	<ul style="list-style-type: none"> <li>• It stores bits 0s and 1s. These bits will be converted to Row Ids</li> </ul>
<ul style="list-style-type: none"> <li>• On high cardinality columns we create BTREE Index</li> </ul>	<ul style="list-style-type: none"> <li>• On low cardinality columns we create Bitmap Index.</li> </ul>

#### Dropping Index:

**Drop Index I1;**

#### User\_Indexes:

**It maintains information about all Indexes**

**Column Index\_name Format a10**

**Select Index\_Name, Index\_Type from User\_Indexes;**

**When we drop the table does it drop rows & columns?**  
**Yes**

**When we drop the table does it drop the views?**  
**No. View will not be deleted. But those will not work.**

**When we drop the table does it drop the indexes  
created on it?**  
**Yes.**

## Synonyms

Thursday, December 16, 2021 7:13 PM

### Synonym:

- It is a DB Object.
- It is used to give alias names ((alternative names) to Database Objects.
- It is permanent. Synonym is permanent whereas column alias name or table alias name is temporary. Column alias or table alias scope is limited to that query only.

### Syntax:

```
Create Synonym <synonym_name>
For <Database_Object>;
```

### Advantages:

- It makes table name short.
- It avoids of writing schema name every time.
- It provides security. Another user cannot know who is the owner & what is the original database object name.

c##oracle11am

-----  
employee\_Details table

c##ramu

-----  
Select \* from  
c##oracle11am.employee\_details

### Types of Synonyms:

- Private Synonym
- Public Synonym

#### • Private Synonym:

- it is created by the user.

c##oracle6pm

-----  
emp

**Log in as DBA:**  
**username: system**  
**password:nareshit**

**Grant create synonym to c##ramu;**

**c##ramu**

**Select \* from c##oracle6pm.emp;**

**Create Synonym e for c##oracle6pm.emp;**

**Select \* from e;**

**Public Synonym:**

- **DBA creates public synonyms gives permission to many users to use it.**

**Syntax:**

**Create Public Synonym <synonym>  
for <object\_name>;**

**c##oracle6pm**

**emp**

**c##usera  
c##userb  
c##userc**

**DBA:**

**grant all on c##oracle6pm.emp  
to c##usera, c##userb, c##userc;**

**usera:**

**select \* from c##oracle6pm.emp;  
create synonym e for c##oracle6pm.emp;  
select \* from e;**

**userb:**  
**select \* from c##oracle6pm.emp;**  
**create synonym e for c##oracle6pm.emp;**  
**select \* from e;**

**userc:**  
**select \* from c##oracle6pm.emp;**  
**create synonym e for c##oracle6pm.emp;**  
**select \* from e;**

**If 3 users wants access c##oracle6pm.emp table with short name, 3 users are creating 3 synonyms.**

**To decrease number of synonyms, DBA creates public synonym. On this he gives permission to many users.**

**log in as DBA:**

**Create public synonym s1 from c##oracle6pm.emp;**

**Grant all on s1 to public;**

**Grant all on s1 to c##usera, c##userb, c##userc;**

**Dropping private synonym:**

**Drop synonym e;**

**Dropping public synonym:**

**Drop public synonym s1;**

**user\_synonyms:**

**It maintains all synonyms information**

**SQL:**

**SQL Commands**

**DDL, DML, DRL/DQL, TCL, DCL**

**Built-in Functions**

**Clauses in SQL**

**Joins**

**Sub Queries**

**Set Operators**

**Views**

**Materialized views**

**Sequences**

**Indexes**

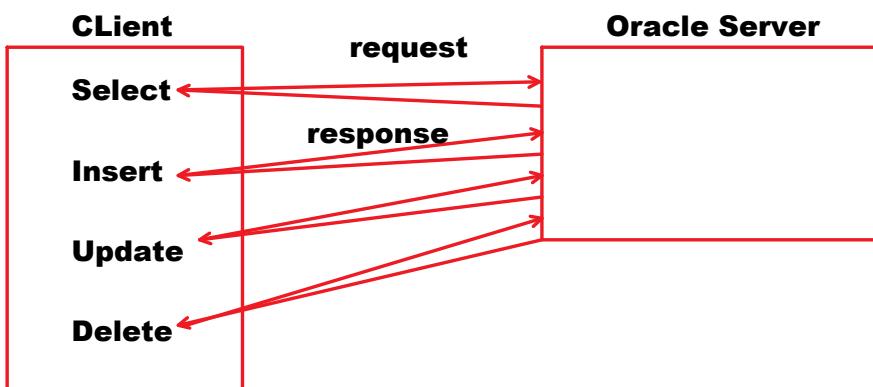
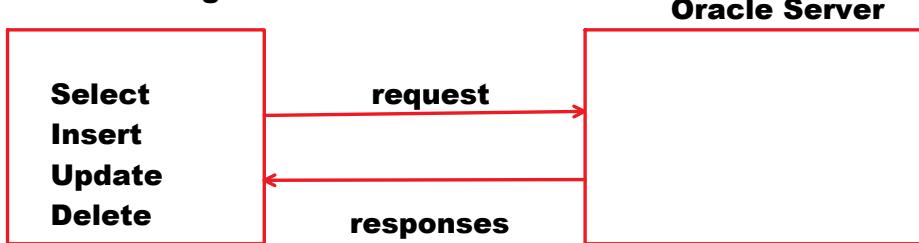
**Synonyms**

**PL/SQL:**

- **Procedural Language / Structured Query Language.**
- **It is an extension of SQL.**
- **It is a Procedural Language. We can write a set of statements. We can define procedures, functions, triggers ..etc.**
- **All SQL queries we can write as statements in PL/SQL program.**

**Advantages:**

- **It improves the performance.**
- **It provides conditional control structures.**
- **It provides looping control structures.**
- **It provides modularity.**
- **It provides Exception Handling.**
- **It provides security.**

**It improves the performance:****PL/SQL Program**

**We can write a set of SQL statements in one PL/SQL program & we can submit to oracle. Now it passes one request to oracle. PL/SQL program reduces no of requests. So, performance will be improved.**

**It provides conditional control structures:**

- **Based on conditions we can perform actions easily using conditional control structures.**
  - **PL/SQL provides conditional control structures like "If..Then", "If..Then..Else", "If..Then..Elsif" ..etc**

### **It provides Looping Control Structures:**

- **Looping control structures are used to execute the statements repeatedly.**
  - **PL/SQL provides looping control structures such as for, while & simple loops.**

- It provides modularity:
  - Modularity => is a programming style

## **Sequential Approach:**

**we write code sequentially**

A series of ten horizontal black dashed lines of varying lengths, decreasing from top to bottom.

## **Disadvantages:**

- Lack of understandability
  - It increases length of code
  - No reusability.

## **Modularity:**

- **Modularity => is a programming style.**

- In this style we divide large program into small parts. This every part is called procedure or function.

#### **Advantages:**

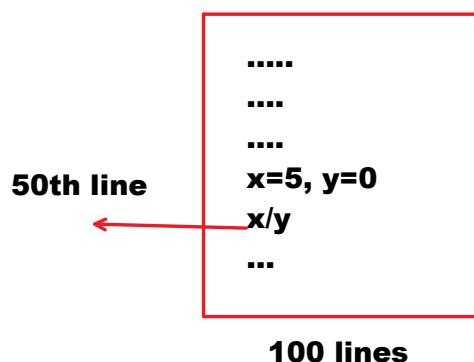
- It improves understandability.
- It provides reusability.
- It decreases length of code.
- Better maintenance.

#### **Exception Handling:**

##### **Exception => Run-Time Error**

**When run-time error occurs we cannot continue our work. Our application will be terminated abnormally in the middle of execution. This is why we need to handle run-time errors.**

##### **Exception-Handling => The mechanism of handling run-time errors**



#### **Security:**

- All Procedures, Functions, Packages are stored in centralized location i.e. server.
- Only authorized users can use them.

## **Types of Blocks:**

**PL/SQL program contains blocks.**

### **2 Types:**

- **Anonymous Blocks**
- **Named Blocks**

#### **Anonymous Block:**

**A Block without the name**

#### **Named Block:**

**A Block with the name**

**Ex:**

**Procedure**  
**Function**  
**Trigger**  
**Package**

#### **Syntax of Anonymous Block:**

```
DECLARE  
    --declaration statements  
BEGIN  
    --executable statements  
END;  
/
```

Diagram illustrating the syntax of an anonymous block:

- The code is enclosed in a red box.
- Red arrows point from the labels to specific parts of the code:
  - An arrow points from "Declaration part" to the "DECLARE" section.
  - An arrow points from "Execution part" to the "BEGIN" section.

## **Data Types in PL/SQL:**

### **Integer Related Data Types:**

Number(n)	}	→ SQL (or) PL/SQL
int		
integer	}	→ specific to PL/SQL
binary_integer		
pls_integer		

### **Floating Point Related Data Types:**

Number(m n)

### **Floating Point Related Data Types:**

**Number(m,n)**

**float**

**real**

**binary\_float**

**binary\_double**

}      SQL (A) PL/SQL

### **Character Related Data Types:**

**char(n)**

**varchar2(n)**

**long**

**clob**

**string** => specific to PL/SQL

**nchar(n)**

**nvarchar2(n)**

**nclob**

}      SQL (A) PL/SQL

### **Date & Time Related Data Types:**

**Date**

**Timestamp**

### **Boolean Type:**

**Boolean** => Specific to PL/SQL

### **Binary Related Data Types**

**BFILE**

**BLOB**

### **Attribute related Data Types:**

**%type**

**%rowtype**

### **Cursor Type:**

**Ref\_Cursor**

}      specific to  
                PL/SQL

### **Variable:**

- Variable is a name of storage location that holds a value.
- It can be changed during program execution.
- A variable can hold only one value at a time.

### **Note:**

To hold the data, variable is required.

To allocate memory for variable, data type is required.

## **Declaring a Variable:**

### **Syntax:**

```
<variable_name> <data_type>;
```

### **Ex:**

```
v_empno number(4);
v_ename varchar2(20);
x number(4);
d date;
```

## **Assigning values:**

:=	<b>Assignment Operator</b>
----	----------------------------

### **Syntax:**

```
<variable> := <constant/variable/expression>;
```

**Ex:**

```
x:=500; constant  

y:=x; variable  

z:=x+y; expression
```



## **Initializing a Variable:**

**Giving value at the time of declaration**

### **Ex:**

```
x number(4):=50;
```

**Text Editors: allow us to write the program.**

**Notepad**

**Edit Plus**

**Notepad++**

**IDEs: allows us to write the program, compile & run the program. Also they provide other**

**features.**

**Toad**  
**SQL Developer**

**put\_line():**  
**package\_name.procedure\_name**

**dbms\_output.put\_line('hello');**

**"dbms\_output" package**

**Procedure put\_line**

.....  
.....  
.....

```
printf("hello");
System.out.println("hello");
cout<<"hello";
print('hello')
```

- **put\_line() is a procedure defined in dbms\_output package.**
- **It is used to print data on screen**

**Program to print 'hello' on screen:**

**Open Notepad & in new file type following program**

```
BEGIN
  dbms_output.put_line('hello');
END;
/
```

**Save above program in d:\oracle6pm folder with the name "hello.sql"**

**Open sql plus:**

**SQL> @d:\oracle6pm\hello.sql => compiles & runs PL/SQL program**

**SERVERTOUTPUT:**

**By default messages cannot be sent to output. To send messages to output we have to set serveroutput as on.**

**SQL> set SERVEROUTPUT On**

**Program to add 2 numbers:**

```
x  y  
40 20
```

**sum=60**

**|| => Concatenation Operator**

**Enter value for x:5**

```
old x:=&x  
new x:=5
```

**to avoid "old" , "new" write following command**

**SQL> set verify off**

**Program to display weekday of  
given date:**

**d Date;**

**d:='&Date';**

**Enyter date:18-DEC-2021  
Saturday**

**s:=to\_char(d,'day');**

**Formats => d,dy,day**

**print s**

**Program to read & print student details:**

**stdid**

```

sname
sec
avrg

DECLARE
    sid Number(4);
    sname Varchar2(10);
    sec char;
    avrg Number(5,2);
BEGIN
    sid := &sid;
    sname := '&sname';
    sec := '&section';
    avrg := &average;

    dbms_output.put_line(sid || ' ' ||
    sname || ' ' || sec || ' ' ||
    average);
END ;
/

```

### **SQL:**

**DDL DML DRL TCL DCL**

- In PL/SQL We can use DML / DRL / TCL commands directly.
- DDL commands we cannot use directly in PL/SQL program. To use DDL Commands in PL/SQL program, we use "Dynamic SQL".

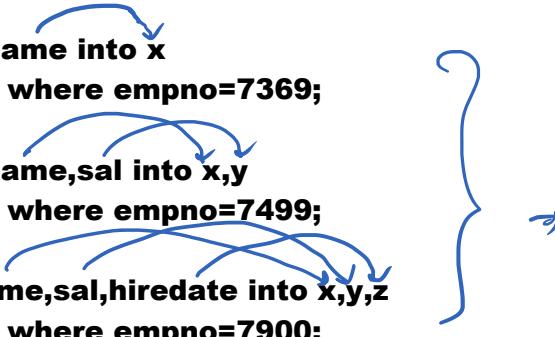
### **Using DML Commands in PL/SQL Program:**

**All DML Commands Syntaxes are same in PL/SQL.**

### **Syntax of DRL Command in PL/SQL:**

### Syntax of DRL Command in PL/SQL:

```
SELECT ename into x  
FROM emp where empno=7369;  
  
SELECT ename,sal into x,y  
FROM emp where empno=7499;  
  
Select ename,sal,hiredate into x,y,z  
FROM emp where empno=7900;
```



**Program to display emp name and salary of given employee number:**

**Enter empno: 1001**  
**SMITH 5000**

**Enter empno: 1002**  
**ALLEN 6840**

```
DECLARE  
    v_empno Number(4);  
    v_ename Varchar2(10);  
    v_sal Number(7,2);  
BEGIN  
    v_empno := &empno;  
  
    SELECT ename, sal into v_ename,v_sal  
    FROM emp where empno=v_empno;  
  
    dbms_output.put_line(v_ename || ' ' || v_Sal);  
END ;  
/
```

**%type:**

```
emp  
empno  Number(4)
```

- is an attribute related data type.
- It is used to declare a variable with table column's data type.

```
emp1  
empid varchar2(10);
```

- is an attribute related data type.
- It is used to declare a variable with table column's data type.  
emp1  
empid varchar2(10);
- It avoids mismatch b/w the table column data type, field size & program variable data type,  
PL/SQL Program:  
v\_Empno Number(2);  
field size.

v\_empid number(3);

**Syntax:**

variable table_name.column_name%type;
---------------------------------------

**Ex:**

```
v_empno emp.empno%type;
v_sal emp.sal%type;
```

```
v_sid std.sid%type;
```

**Program to insert an emp record into emp table:**

emp			
empno	ename	job	sal

**DECLARE**

```
v_empno emp.empno%type;
v_ename emp.ename%type;
v_job emp.job%type;
v_sal emp.sal%type;
```

**BEGIN**

```
v_empno := &empno;
v_ename := '&ename';
v_job := '&job';
v_sal := &sal;
```

```
Insert into emp(empno,ename,job,sal)
values(v_empno,v_ename,v_job,v_sal);
```

```
dbms_output.put_line('record inserted..!');
```

**END;**

/

**Program to delete a record of given empno from emp table:**

**Read empno**

**Delete query**

```
DECLARE
    v_empno emp.empno%type;
BEGIN
    v_empno := &empno;

    Delete from emp where empno=v_empno;

    dbms_output.put_line('record deleted..!');
END;
/
```

**Program to increase sal of an employee with specific amount:**

```
Enter empno:1001
Enter amount: 2000
sal is: 5000
Salary updated..
sal is: 7000
```

**Program to find experience of an given employee number:**

**v\_empno:=&empno;**

```
Select hiredate into v_hiredate from emp
where empno=v_Empno;
```

```
v_exp:=(sysdate-v_hiredate)/365;
```

```
do.pl(v_exp);
```

**Data Types**

**Declare**

**Assign**

**Initialize**

**print**

**read**

**Performing DB operations**

## Control Structures

Tuesday, December 21, 2021 6:29 PM

### Sequential Statements

```
do.pl('hello');  
  
do.pl('hi');  
  
do.pl('welcome');  
  
hello  
hi  
welcome
```

### Control Statements

```
n:=5;  
if n>5 then  
    do.pl('hello');  
end if;  
IF N<5 THEN  
    do.pl('hi');  
END IF;  
IF N=5 THEN  
    do.pl('welcome');  
END IF;
```

welcome

**Sequential Statements:**  
get executed sequentially.

**Control Statements:**  
get executed based on the condition.

### Control Structures:

- Normally, PL/SQL program gets executed sequentially. To change the sequential execution we use "Control Structures".
- Control Structures are used to control the flow of execution of statements.

**PL/SQL provides 3 types of Control Structures:**

<b>Conditional</b>	<b>If .. Then</b> <b>If ..Then ..Else</b> <b>If .. Then ..Elsif</b> <b>Nested If</b> <b>CASE</b>
<b>Looping / Iterative</b>	<b>While</b> <b>For</b> <b>Simple loop</b>
<b>Jumping</b>	<b>goto</b> <b>exit</b> <b>exit when</b> <b>continue</b>

**Conditional Control Structures:**  
are used to execute the statements based on condition

- **If ..Then**
- **If .. Then ..Else**
- **If .. Then .. EIsif**
- **Nested If**
- **Case**

**If ..Then:**

-- => single line comment

**Syntax:**

/\* \*/ => multi line comment

```
If <Condition> Then
  -- Statements
End If;
```

**The statements in "if" block get executed when condition is TRUE.  
It will not execute the statements when the condition is FALSE**

**Program to check whether the person is eligible for vote or not:**

**Read age**

```
If age>=18 Then
    do.pl('Eligible for voting');
End If;
```

**If ..Then .. Else:**

**Syntax:**

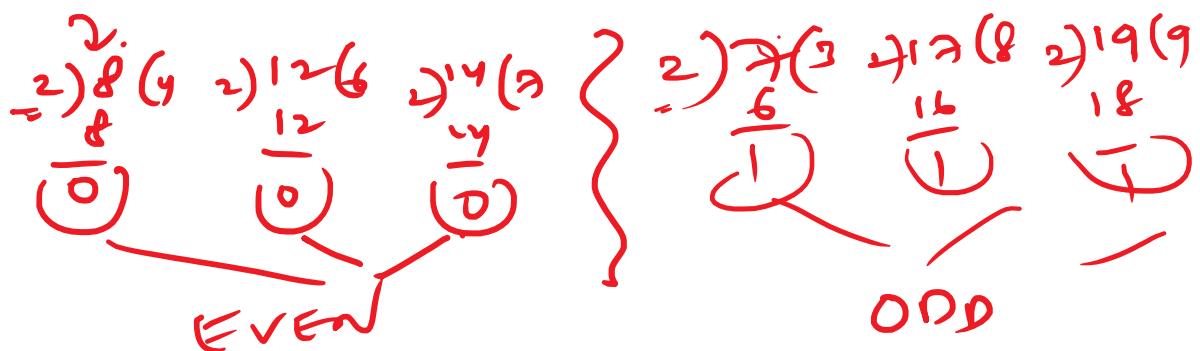
```
If <condition> Then
    -- Statements
Else
    -- Statements
End If ;
```

**The statements in "if" block get executed when condition is TRUE  
The statements in "else" block get executed when condition is FALSE.**

## **Program to check whether the given number is even or odd:**

**EVEN => 2,4,6,8, .... => divide with 2 => remainder 0**

**ODD => 1,3,5,7, .... => divide with 2 => remainder 1**



**Read n**

```
If mod(n,2)=0 Then
    do.pl('EVEN');
else
    do.pl('ODD');
End If;
```

**If .. Then .. Elsif:**  
**Used to write multiple conditions.**

**Syntax:**

```
If <condition-1> Then
    -- Statements
Elsif <condition-2> Then
    -- Statements
```

```

-- Statements
Elsif <condition-2> Then
  -- Statements
Elsif <condition-3> Then
  -- Statements
  .
  .
Else
  -- Statements
End If ;

```

- **The statements in "If .. Then .. Elsif" get executed when corresponding condition is TRUE.**
- **When All conditions are FALSE, it executes "else" block statements.**
- **Writing "else" block is optional.**

### **Program to check whether the given number is +ve or -ve or ZERO:**

**Read n**

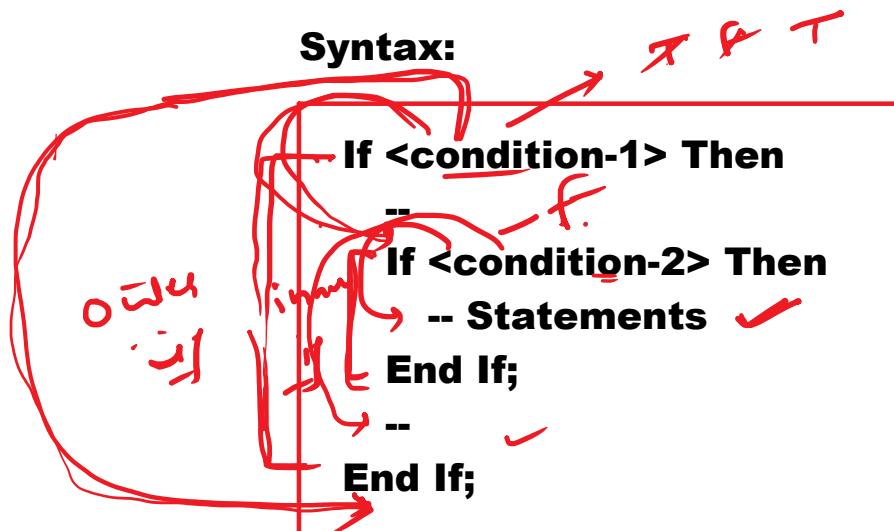
```

If n>0 then
  do.pl('+ve');
elsif n<0 then
  do.pl('-ve');
else
  do.pl('ZERO');
End If;

```

### **Nested If:**

**Writing if in another if is called "Nested If".**



**The statements in INNER IF get executed when outer condition & inner condition are TRUE.**

**Program to find biggest in 3 different numbers:**

x	y	z
80	50	70

**x>y**

**x>z**

**Program to delete an employee record of given empno. If experience is >40 then delete the record.**

**Read empno**

**Calculate experience**

**display experience**

**If exp>40 Then  
    delete query  
end if;**

**DECLARE**

**v\_empno emp.empno%type;  
v\_hiredate emp.hiredate%type;  
v\_exp INT ;**

**BEGIN**

**v\_empno := &empno;**

**SELECT hiredate INTO v\_hiredate  
FROM emp  
WHERE empno=v\_empno;**

**v\_exp := (sysdate-  
v\_hiredate)/365;**

**dbms\_output.put\_line('experienc**

```
e is: ' || v_exp);  
  
IF v_exp>40 THEN  
    DELETE FROM emp WHERE  
    empno=v_empno;  
    COMMIT ;  
    dbms_output.put_line('record  
    deleted..!');  
    END IF ;  
END ;  
/
```

**Program to update sal of given employee  
number with given amount. after update if  
salary exceeds more than 10000 cancel it.**

**Read empno**

**Read amount**

**Update the salary**

**Select the salary**

```
if salary>10000 then  
    Rollback;  
    else  
        Commit;  
end if ;
```

```

DECLARE
  v_empno emp.empno%type;
  v_amount FLOAT ;
  v_sal emp.sal%type;
BEGIN
  v_empno := &empno;
  v_amount := &amount;

  UPDATE emp SET
    sal=sal+v_amount
  WHERE empno=v_empno;

  SELECT sal INTO v_sal FROM
    emp
  WHERE empno=v_empno;

  IF V_SAL>10000 THEN
    ROLLBACK ;
    dbms_output.put_line('Rolled
      Back..!');
  ELSE
    COMMIT ;
    dbms_output.put_line('Committ
      ed..!');
  END IF ;

END ;
/

```

**Program to increase salary of employees as following:**

**deptno=10 => increase 10% on sal  
deptno=20 => increase 20% on sal  
deptno=30 => increase 15% on sal  
others => increase 5% on sal**

**Read empno**

**Find deptno of that employee**

**If v\_Deptno=10 then  
    10%  
elsif v\_deptno=20 then  
    20%**

**DECLARE**

**v\_empno emp.empno%type;  
v\_deptno emp.deptno%type;  
v\_per FLOAT ;**

**BEGIN**

**v\_empno := &empno;**

**SELECT deptno INTO v\_deptno  
FROM emp  
WHERE empno=v\_empno;**

**IF v\_deptno=10 THEN  
    v\_per := 10;  
elsif v\_deptno=20 THEN  
    v\_per := 20;  
elsif v\_deptno=30 THEN  
    v\_per := 15;  
ELSE**

```

v_per := 5;
END IF ;

UPDATE emp SET
sal=sal+(sal*v_per/100)
WHERE empno=v_empno;
COMMIT;

dbms_output.put_line('deptno:' ||
v_deptno || ' ' || 'per:' || v_per);
dbms_output.put_line('salary
updated..!');

END ;
/

```

**Program update the salary of given  
empno as following:**

**If job=clerk => increase 10%**  
**if job=manager => increase 20%**  
**if job=salesman => increase 15%**  
**others              => increase 5%**

**Program to perform deposit & withdraw transactions:**

**Create a Table with following  
structure:**

### **Account**

<b>Acno</b>	<b>Name</b>	<b>Balance</b>
<b>1001</b>	<b>A</b>	<b>100000</b>
<b>1002</b>	<b>B</b>	<b>200000</b>

**Read Acno**

**Read Transaction Type => w d**

**Read amount**

**If v\_type='w' then**

**DECLARE**

**v\_acno account.acno%type;**  
**v\_type CHAR ;**  
**v\_amount FLOAT ;**  
**v\_balance account.balance%**  
**type;**

**BEGIN**

**v\_acno := &acno;**  
**v\_type := '&trans\_type';**  
**v\_amount := &amount;**

**SELECT balance INTO v\_balance**  
**FROM account**  
**WHERE acno=v\_acno;**

**IF v\_type='w' THEN**

**IF v\_amount>v\_balance THEN**

```

dbms_output.put_line('Insufficient balance');
ELSE
    UPDATE account SET
        balance=balance-v_amount
    WHERE acno=v_acno;
    COMMIT;
    dbms_output.put_line('amount debited..!');
END IF ;
elsif v_type='d' then
    UPDATE account SET
        balance=balance+v_amount
    WHERE acno=v_acno;
    COMMIT ;
    dbms_output.put_line('amount credited..!');
ELSE
    dbms_output.put_line('Invalid Transaction..!');
END IF ;
END ;
/

```

### **CASE:**

**It is used to implement "If-Then-Else".**

**it can be used in 2 ways:**

- **Simple Case**
- **Searched Case**

- **Simple Case =>**
- **can check equality condition only**
- **It can be used to execute the statements based**

**on value matching.**

**Syntax:**

```
Case <Expression>
  When <value> Then
    --Statements
  When <value> Then
    --Statements
  .
  .
Else
  --Statements
End Case;
```

**Program to check whether the given number is even or odd using CASE:**

**Read n**

```
CASE mod(n,2)
  When 0 then
    do.pl('EVEN');
  When 1 then
    do.pl('ODD');
END CASE;
```

```
DECLARE
  n INT ;
```

```

BEGIN
  n := &n;
  
  CASE mod(n,2)
    WHEN 0 THEN
      dbms_output.put_line('EVEN')
      ;
    WHEN 1 THEN
      dbms_output.put_line('ODD');
  END CASE;
END ;
/

```

### **Searched Case:**

- It can check any type of condition.

### **Syntax:**

```

Case
  When <condition> Then
    --Statements
  When <condition> Then
    --Statements
  .
  .
  Else
    --Statements
End Case;

```

**Program to check whether the given number is +ve or -ve or zero using CASE:**

**Read n**

**Case**

**When n>0 Then**

**do.pl('+ve');**

**When n<0 Then**

**do.pl('-ve');**

**Else**

**do.pl('ZERO');**

**End Case;**

**DECLARE**

**n INT;**

**BEGIN**

**n := &n;**

**CASE**

**WHEN n>0 THEN**

**dbms\_output.put\_line('+ve');**

**WHEN n<0 THEN**

**dbms\_output.put\_line('-ve');**

**ELSE**

**dbms\_output.put\_line('ZERO')**

**;**

**END CASE ;**

**END ;**

**/**

**Looping Control Structures:**  
**are used to execute the statements repeatedly.**

**PL/SQL provides 3 Looping Control Structures:**

- **while loop**
- **for loop**
- **simple loop**

**while loop:**

**Syntax:**

```
while <condition>
Loop
  --Statements
End Loop;
```

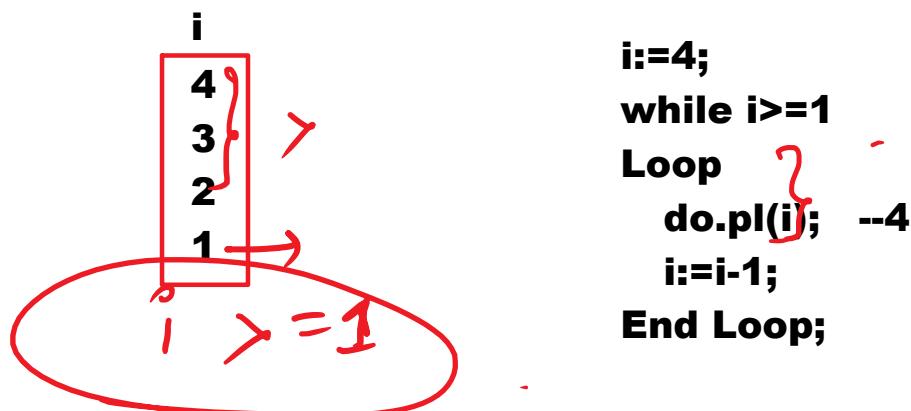
**Program to print numbers from 1 to 4:**

<b>i</b>  <div style="border: 2px solid red; padding: 5px; width: fit-content; margin-left: auto; margin-right: 0;"><b>1</b> <b>2</b> <b>3</b> <b>4</b></div>	<b>i:=1;</b>  <b>i:=1;</b> <b>while i&lt;=4</b> <b>Loop</b> <b>do.pl(i);</b> <b>i:=i+1;</b> <b>End Loop;</b>	<b>i:=1;</b>  <b>do.pl(i); --1</b> <b>i:=i+1; --i=2</b>  <b>do.pl(i); --2</b> <b>i:=i+1; -- i=3</b>
---	---	---

```
do.pl(i); --3  
i:=i+1; --i=4
```

```
do.pl(i); --4
```

### Program to print numbers from 4 to 1:



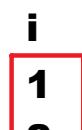
### for loop:

#### Syntax:

```
for <variable> IN [REVERSE]<lower>..<upper>  
Loop  
--Statements  
End Loop;
```

### Program to print numbers from 1 to 4:

```
for i IN 1..4  
Loop  
do n/i/n.
```



```
Loop
  do.pl(i);
End Loop;
```

<b>1</b>
<b>2</b>
<b>3</b>
<b>4</b>

### **Program to print numbers from 4 to 1:**

<b>i</b>
<b>4</b>
<b>3</b>
<b>2</b>
<b>1</b>

```
for i in REVERSE 1..4
Loop
  do.pl(i);
End Loop;
```

<b>1</b>
<b>2</b>
<b>3</b>
<b>4</b>

### **Note:**

- We have no need to declare loop variable in case of "for". Implicitly it will be declared as number type.**

### **Simple Loop:**

```
Loop
  --Statements
  EXIT / EXIT WHEN <condition>;
End Loop;
```

## **Program to print numbers from 1 to4 using simple loop**

i

1  
2  
3  
4

i:=1;

**Loop**

**do.pl(i); -- 1 2 3 4 5**

**exit when i=5;**

**i:=i+1;**

**End Loop;**

**continue:**

**is used to skip current iteration & continue the next iteration**

```
BEGIN
  for i IN 1..10
    Loop
      if i=7 then
        continue ;
      END IF;
      dbms_output.put_line(i);
    End Loop;

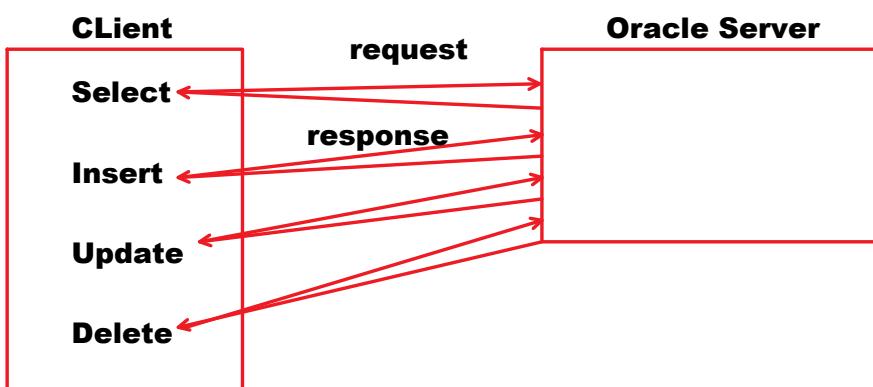
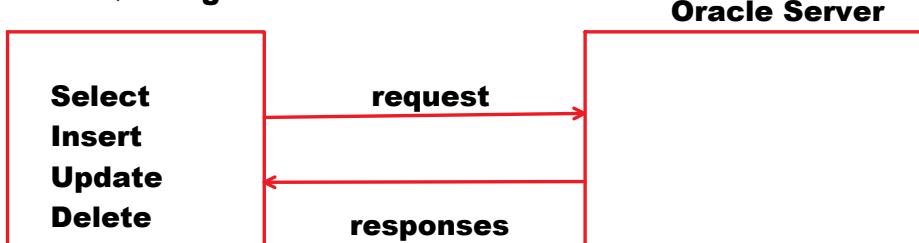
  END ;
/
```

**PL/SQL:**

- **Procedural Language / Structured Query Language.**
- **It is an extension of SQL.**
- **It is a Procedural Language. We can write a set of statements. We can define procedures, functions, triggers ..etc.**
- **All SQL queries we can write as statements in PL/SQL program.**

**Advantages:**

- **It improves the performance.**
- **It provides conditional control structures.**
- **It provides looping control structures.**
- **It provides modularity.**
- **It provides Exception Handling.**
- **It provides security.**

**It improves the performance:****PL/SQL Program**

**We can write a set of SQL statements in one PL/SQL program & we can submit to oracle. Now it passes one request to oracle. PL/SQL program reduces no of requests. So, performance will be improved.**

**It provides conditional control structures:**

- **Based on conditions we can perform actions easily using conditional control structures.**
  - **PL/SQL provides conditional control structures like "If..Then", "If..Then..Else", "If..Then..Elsif" ..etc**

### **It provides Looping Control Structures:**

- **Looping control structures are used to execute the statements repeatedly.**
  - **PL/SQL provides looping control structures such as for, while & simple loops.**

- It provides modularity:
  - Modularity => is a programming style

## **Sequential Approach:**

**we write code sequentially**

A series of ten horizontal dashed lines of varying lengths, decreasing from top to bottom.

## **Disadvantages:**

- Lack of understandability
  - It increases length of code
  - No reusability.

## **Modularity:**

- **Modularity => is a programming style.**

- In this style we divide large program into small parts. This every part is called procedure or function.

#### **Advantages:**

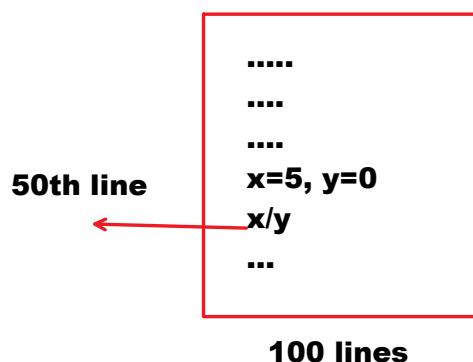
- It improves understandability.
- It provides reusability.
- It decreases length of code.
- Better maintenance.

#### **Exception Handling:**

##### **Exception => Run-Time Error**

**When run-time error occurs we cannot continue our work. Our application will be terminated abnormally in the middle of execution. This is why we need to handle run-time errors.**

##### **Exception-Handling => The mechanism of handling run-time errors**



#### **Security:**

- All Procedures, Functions, Packages are stored in centralized location i.e. server.
- Only authorized users can use them.

## **Types of Blocks:**

**PL/SQL program contains blocks.**

### **2 Types:**

- **Anonymous Blocks**
- **Named Blocks**

#### **Anonymous Block:**

**A Block without the name**

#### **Named Block:**

**A Block with the name**

**Ex:**

**Procedure**  
**Function**  
**Trigger**  
**Package**

#### **Syntax of Anonymous Block:**

```
DECLARE  
    --declaration statements  
BEGIN  
    --executable statements  
END;  
/
```

Diagram illustrating the syntax of an anonymous block:

- The code is enclosed in a red box.
- Braces on the left side group the code into two main sections:
  - The first brace groups the **DECLARE** section and the **--declaration statements**.
  - The second brace groups the **BEGIN** section and the **--executable statements**.
- Red arrows point from the braces to handwritten labels:
  - An arrow from the first brace points to the text "Declaration part".
  - An arrow from the second brace points to the text "Execution part".

## **Data Types in PL/SQL:**

### **Integer Related Data Types:**

**Number(n)**      } → **SQL (or) PL/SQL**  
**int**                  }  
**integer**               }  
**binary\_integer**      } → **specific to PL/SQL**  
**pls\_integer**          }

### **Floating Point Related Data Types:**

**Number(m n)**      ^

### **Floating Point Related Data Types:**

**Number(m,n)**

**float**

**real**

**binary\_float**

**binary\_double**

}      *SQL (A) PL/SQL*

### **Character Related Data Types:**

**char(n)**

**varchar2(n)**

**long**

**clob**

**string** => specific to PL/SQL

**nchar(n)**

**nvarchar2(n)**

**nclob**

}      *SQL (A) PL/SQL*

### **Date & Time Related Data Types:**

**Date**

**Timestamp**

### **Boolean Type:**

**Boolean** => Specific to PL/SQL

### **Binary Related Data Types**

**BFILE**

**BLOB**

### **Attribute related Data Types:**

**%type**

**%rowtype**

### **Cursor Type:**

**Ref\_Cursor**

}      *specific to PL/SQL*

### **Variable:**

- Variable is a name of storage location that holds a value.
- It can be changed during program execution.
- A variable can hold only one value at a time.

### **Note:**

To hold the data, variable is required.

To allocate memory for variable, data type is required.

## **Declaring a Variable:**

### **Syntax:**

```
<variable_name> <data_type>;
```

### **Ex:**

```
v_empno number(4);
v_ename varchar2(20);
x number(4);
d date;
```

## **Assigning values:**

:=	<b>Assignment Operator</b>
----	----------------------------

### **Syntax:**

```
<variable> := <constant/variable/expression>;
```

**Ex:**

```
x:=500; constant
y:=x; variable
z:=x+y; expression
```



## **Initializing a Variable:**

**Giving value at the time of declaration**

### **Ex:**

```
x number(4):=50;
```

**Text Editors: allow us to write the program.**

**Notepad**

**Edit Plus**

**Notepad++**

**IDEs: allows us to write the program, compile & run the program. Also they provide other**

**features.**

**Toad**  
**SQL Developer**

**put\_line():**  
**package\_name.procedure\_name**

**dbms\_output.put\_line('hello');**

**"dbms\_output" package**

**Procedure put\_line**

.....  
.....  
.....

**printf("hello");**  
**System.out.println("hello");**  
**cout<<"hello";**  
**print('hello')**

- **put\_line() is a procedure defined in dbms\_output package.**
- **It is used to print data on screen**

**Program to print 'hello' on screen:**

**Open Notepad & in new file type following program**

**BEGIN**  
    **dbms\_output.put\_line('hello');**  
**END;**  
**/**

**Save above program in d:\oracle6pm folder with the name "hello.sql"**

**Open sql plus:**

**SQL> @d:\oracle6pm\hello.sql => compiles & runs PL/SQL program**

**SERVERTOUTPUT:**

**By default messages cannot be sent to output. To send messages to output we have to set serveroutput as on.**

**SQL> set SERVEROUTPUT On**

**Program to add 2 numbers:**

```
x    y  
40  20
```

**sum=60**

**|| => Concatenation Operator**

**Enter value for x:5**

```
old x:=&x  
new x:=5
```

**to avoid "old" , "new" write following command**

**SQL> set verify off**

**Program to display weekday of  
given date:**

**d Date;**

**d:='&Date';**

**Enyter date:18-DEC-2021  
Saturday**

**s:=to\_char(d,'day');**

**Formats => d,dy,day**

**print s**

**Program to read & print student details:**

**stdid**

```

sname
sec
avrg

DECLARE
    sid Number(4);
    sname Varchar2(10);
    sec char;
    avrg Number(5,2);
BEGIN
    sid := &sid;
    sname := '&sname';
    sec := '&section';
    avrg := &average;

    dbms_output.put_line(sid || ' ' ||
    sname || ' ' || sec || ' ' ||
    average);
END ;
/

```

### **SQL:**

**DDL DML DRL TCL DCL**

- In PL/SQL We can use DML / DRL / TCL commands directly.
- DDL commands we cannot use directly in PL/SQL program. To use DDL Commands in PL/SQL program, we use "Dynamic SQL".

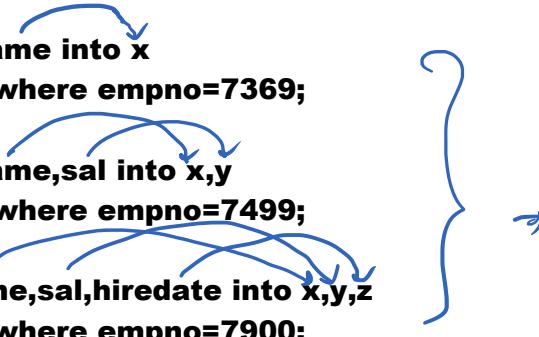
### **Using DML Commands in PL/SQL Program:**

**All DML Commands Syntaxes are same in PL/SQL.**

### **Syntax of DRL Command in PL/SQL:**

### Syntax of DRL Command in PL/SQL:

```
SELECT ename into x  
FROM emp where empno=7369;  
  
SELECT ename,sal into x,y  
FROM emp where empno=7499;  
  
Select ename,sal,hiredate into x,y,z  
FROM emp where empno=7900;
```



**Program to display emp name and salary of given employee number:**

**Enter empno: 1001**  
**SMITH 5000**

**Enter empno: 1002**  
**ALLEN 6840**

```
DECLARE  
    v_empno Number(4);  
    v_ename Varchar2(10);  
    v_sal Number(7,2);  
BEGIN  
    v_empno := &empno;  
  
    SELECT ename, sal into v_ename,v_sal  
    FROM emp where empno=v_empno;  
  
    dbms_output.put_line(v_ename || ' ' || v_Sal);  
END ;  
/
```

**%type:**

```
emp  
empno  Number(4)  
  
• is an attribute related data type.  
• It is used to declare a variable with table  
column's data type.
```

```
emp1  
empid varchar2(10);
```

- is an attribute related data type.
- It is used to declare a variable with table column's data type.  
emp1  
empid varchar2(10);
- It avoids mismatch b/w the table column data type, field size & program variable data type,  
PL/SQL Program:  
v\_Empno Number(2);  
field size.

v\_empid number(3);

**Syntax:**

variable table_name.column_name%type;
---------------------------------------

**Ex:**

```
v_empno emp.empno%type;
v_sal emp.sal%type;
```

```
v_sid std.sid%type;
```

**Program to insert an emp record into emp table:**

emp			
empno	ename	job	sal

**DECLARE**

```
v_empno emp.empno%type;
v_ename emp.ename%type;
v_job emp.job%type;
v_sal emp.sal%type;
```

**BEGIN**

```
v_empno := &empno;
v_ename := '&ename';
v_job := '&job';
v_sal := &sal;
```

```
Insert into emp(empno,ename,job,sal)
values(v_empno,v_ename,v_job,v_sal);
```

```
dbms_output.put_line('record inserted..!');
```

**END;**

/

**Program to delete a record of given empno from emp table:**

**Read empno**

**Delete query**

```
DECLARE
    v_empno emp.empno%type;
BEGIN
    v_empno := &empno;

    Delete from emp where empno=v_empno;

    dbms_output.put_line('record deleted..!');
END;
/
```

**Program to increase sal of an employee with specific amount:**

```
Enter empno:1001
Enter amount: 2000
sal is: 5000
Salary updated..
sal is: 7000
```

**Program to find experience of an given employee number:**

**v\_empno:=&empno;**

```
Select hiredate into v_hiredate from emp
where empno=v_Empno;
```

```
v_exp:=(sysdate-v_hiredate)/365;
```

```
do.pl(v_exp);
```

**Data Types**

**Declare**

**Assign**

**Initialize**

**print**

**read**

**Performing DB operations**

## Control Structures

Tuesday, December 21, 2021 6:29 PM

### Sequential Statements

```
do.pl('hello');
do.pl('hi');
do.pl('welcome');

hello
hi
welcome
```

### Control Statements

```
n:=5;
if n>5 then
  do.pl('hello');
end if;
IF N<5 THEN
  do.pl('hi');
END IF;
IF N=5 THEN
  do.pl('welcome');
END IF;
```

welcome

**Sequential Statements:**  
get executed sequentially.

**Control Statements:**  
get executed based on the condition.

### Control Structures:

- Normally, PL/SQL program gets executed sequentially. To change the sequential execution we use "Control Structures".
- Control Structures are used to control the flow of execution of statements.

**PL/SQL provides 3 types of Control Structures:**

<b>Conditional</b>	<b>If .. Then</b> <b>If ..Then ..Else</b> <b>If .. Then ..Elsif</b> <b>Nested If</b> <b>CASE</b>
<b>Looping / Iterative</b>	<b>While</b> <b>For</b> <b>Simple loop</b>
<b>Jumping</b>	<b>goto</b> <b>exit</b> <b>exit when</b> <b>continue</b>

**Conditional Control Structures:**  
are used to execute the statements based on condition

- If ..Then
- If .. Then ..Else
- If .. Then .. Elsif
- Nested If
- Case

If ..Then: -- => single line comment

Syntax: /\* \*/ => multi line comment

```
If <Condition> Then  
    -- Statements  
End If;
```

The statements in "if" block get  
executed when condition is TRUE.  
It will not execute the statements when  
the condition is FALSE

**Program to check whether the person is  
eligible for vote or not:**

**Read age**

```
If age>=18 Then  
    do.pl('Eligible for voting');  
End If;
```

If ..Then .. Else:

Syntax:

```
If <condition> Then  
    -- Statements  
Else  
    -- Statements
```

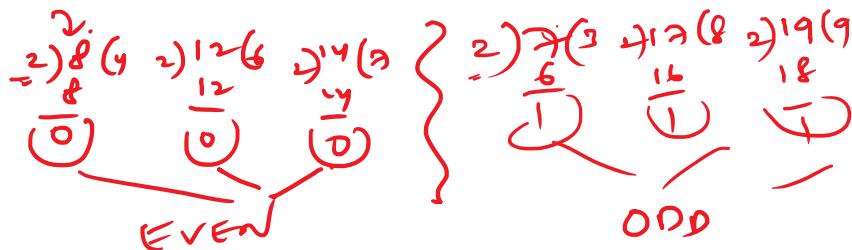
The statements in "if" block  
get executed when condition is  
TRUE  
The statements in "else" block  
get executed when condition is  
FALSE

```
-- Statements  
Else  
-- Statements  
End If ;
```

**The statements in "else" block get executed when condition is FALSE.**

**Program to check whether the given number is even or odd:**

**EVEN => 2,4,6,8, .... => divide with 2 => remainder 0**  
**ODD => 1,3,5,7, .... => divide with 2 => remainder 1**



## **Read n**

```
If mod(n,2)=0 Then  
    do.pl('EVEN');  
else  
    do.pl('ODD');  
End If;
```

**If .. Then .. Elsif:**  
**Used to write multiple conditions.**

## Syntax:

```
If <condition-1> Then
    -- Statements
Elsif <condition-2> Then
    -- Statements
Elsif <condition-3> Then
    -- Statements
.
.
Else
    -- Statements
```

```
Else  
-- Statements  
End If ;
```

- The statements in "If .. Then .. Elsif" get executed when corresponding condition is TRUE.
- When All conditions are FALSE, it executes "else" block statements.
- Writing "else" block is optional.

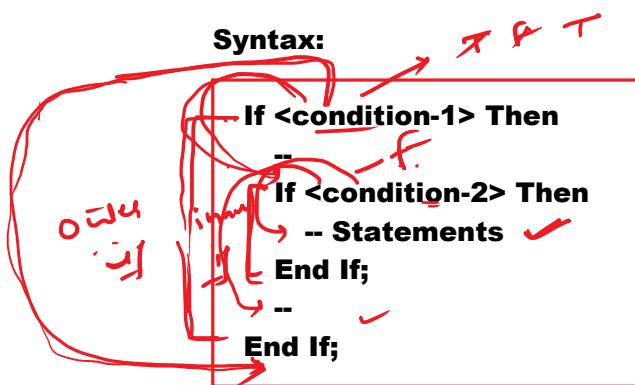
**Program to check whether the given number is +ve or -ve or ZERO:**

**Read n**

```
If n>0 then  
    do.pl('+ve');  
elsif n<0 then  
    do.pl('-ve');  
else  
    do.pl('ZERO');  
End If;
```

**Nested If:**

Writing if in another if is called "Nested If".



The statements in INNER IF get

**executed when outer condition &  
inner condition are TRUE.**

**Program to find biggest in 3 different numbers:**

<b>x</b>	<b>y</b>	<b>z</b>
<b>80</b>	<b>50</b>	<b>70</b>

**x>y  
x>z**

**Program to delete an employee record of given empno. If experience is >40 then delete the record.**

**Read empno**

**Calculate experience**

**display experience**

**If exp>40 Then  
    delete query  
end if;**

```
DECLARE
    v_empno emp.empno%type;
    v_hiredate emp.hiredate%type;
    v_exp INT ;
BEGIN
    v_empno := &empno;

    SELECT hiredate INTO v_hiredate
    FROM emp
    WHERE empno=v_empno;

    v_exp := (sysdate-
    v_hiredate)/365;

    dbms_output.put_line('experienc
```

```

e is: ' || v_exp);

IF v_exp>40 THEN
    DELETE FROM emp WHERE
    empno=v_empno;
    COMMIT ;
    dbms_output.put_line('record
    deleted..!');
END IF ;
END ;
/

```

**Program to update sal of given employee number with given amount. after update if salary exceeds more than 10000 cancel it.**

**Read empno**

**Read amount**

**Update the salary**

**Select the salary**

```

if salary>10000 then
    Rollback;
else
    Commit;
end if ;

```

**DECLARE**

```

v_empno emp.empno%type;
v_amount FLOAT ;
v_sal emp.sal%type;

```

**BEGIN**

```

v_empno := &empno;
v_amount := &amount;

```

```

UPDATE emp SET
sal=sal+v_amount
WHERE empno=v_empno;

```

```

SELECT sal INTO v_sal FROM

```

```

emp
WHERE empno=v_empno;

IF V_SAL>10000 THEN
    ROLLBACK ;
    dbms_output.put_line('Rolled
    Back..!');
ELSE
    COMMIT ;
    dbms_output.put_line('Committ
    ed..!');
END IF ;

END ;
/

```

**Program to increase salary of employees as  
following:**

**deptno=10 => increase 10% on sal**  
**deptno=20 => increase 20% on sal**  
**deptno=30 => increase 15% on sal**  
**others => increase 5% on sal**

**Read empno**

**Find deptno of that employee**

```

If v_Deptno=10 then
    10%
elsif v_deptno=20 then
    20%

```

```

DECLARE
    v_empno emp.empno%type;
    v_deptno emp.deptno%type;
    v_per FLOAT ;
BEGIN
    v_empno := &empno;

    SELECT deptno INTO v_deptno
    FROM emp
    WHERE empno=v_empno;

```

```

IF v_deptno=10 THEN
    v_per := 10;
elsif v_deptno=20 THEN
    v_per := 20;
elsif v_deptno=30 THEN
    v_per := 15;
ELSE
    v_per := 5;
END IF ;

UPDATE emp SET
sal=sal+(sal*v_per/100)
WHERE empno=v_empno;
COMMIT;

dbms_output.put_line('deptno:' ||
v_deptno || ' ' || 'per:' || v_per);
dbms_output.put_line('salary
updated..!');

END ;
/

```

**Program update the salary of given  
empno as following:**

**If job=clerk => increase 10%**  
**if job=manager => increase 20%**  
**if job=salesman => increase 15%**  
**others              => increase 5%**

**Program to perform deposit & withdraw transactions:**

**Create a Table with following  
structure:**

**Account**

<b>Acno</b>	<b>Name</b>	<b>Balance</b>
<b>1001</b>	<b>A</b>	<b>100000</b>
<b>1002</b>	<b>B</b>	<b>200000</b>

**Read Acno**  
**Read Transaction Type => w d**  
**Read amount**

**If v\_type='w' then**

```
DECLARE
  v_acno account.acno%type;
  v_type CHAR ;
  v_amount FLOAT ;
  v_balance account.balance%
type;
BEGIN
  v_acno := &acno;
  v_type := '&trans_type';
  v_amount := &amount;

  SELECT balance INTO v_balance
  FROM account
  WHERE acno=v_acno;

  IF v_type='w' THEN
    IF v_amount>v_balance THEN
      dbms_output.put_line('Insufficient balance');
    ELSE
      UPDATE account SET
        balance=balance-v_amount
      WHERE acno=v_acno;
      COMMIT;
      dbms_output.put_line('amount debited..!');
    END IF ;
  elsif v_type='d' then
    UPDATE account SET
      balance=balance+v_amount
    WHERE acno=v_acno;
    COMMIT ;
    dbms_output.put_line('amount credited..!');
  ELSE
    dbms_output.put_line('Invalid Transaction..!');
  END IF ;
END ;
/
```

**CASE:**

**It is used to implement "If-Then-Else".**

**it can be used in 2 ways:**

- **Simple Case**
- **Searched Case**

- **Simple Case =>**
- **can check equality condition only**
- **It can be used to execute the statements based on value matching.**

**Syntax:**

```
Case <Expression>
  When <value> Then
    --Statements
  When <value> Then
    --Statements
  .
  .
  Else
    --Statements
End Case;
```

**Program to check whether the given number is even or odd using CASE:**

**Read n**

```
CASE mod(n,2)
  When 0 then
    do.pl('EVEN');
  When 1 then
    do.pl('ODD');
END CASE;
```

```
DECLARE
  n INT ;
BEGIN
  n := &n;
```

```

CASE mod(n,2)
  WHEN 0 THEN
    dbms_output.put_line('EVEN')
    ;
  WHEN 1 THEN
    dbms_output.put_line('ODD');
END CASE;
END ;
/

```

#### **Searched Case:**

- It can check any type of condition.

#### **Syntax:**

```

Case
  When <condition> Then
    --Statements
  When <condition> Then
    --Statements
  .
  .
  Else
    --Statements
End Case;

```

#### **Program to check whether the given number is +ve or -ve or zero using CASE:**

**Read n**

```

Case
  When n>0 Then
    do.pl('+ve');
  When n<0 Then
    do.pl('-ve');
  Else
    do.pl('ZERO');
End Case;

```

```

DECLARE
    n INT;
BEGIN
    n := &n;

    CASE
        WHEN n>0 THEN
            dbms_output.put_line('+ve');
        WHEN n<0 THEN
            dbms_output.put_line('-ve');
        ELSE
            dbms_output.put_line('ZERO')
        ;
    END CASE ;

END ;
/

```

**Looping Control Structures:**  
are used to execute the statements repeatedly.

**PL/SQL provides 3 Looping Control Structures:**

- **while loop**
- **for loop**
- **simple loop**

**while loop:**

**Syntax:**

```

while <condition>
Loop
    --Statements
End Loop;

```

**Program to print numbers from 1 to 4:**

i

1  
2  
3

i:=1;  
while i<=4  
loop

i:=1;  
do.pl(i); --1  
i:=i+1; --i=2

```

1      i:=1;
2      while i<=4           do.pl(i); --1
3      Loop                  i:=i+1; --i=2
4          do.pl(i);
           i:=i+1;
      End Loop;            do.pl(i); --2
                           i:=i+1; -- i=3
                               do.pl(i); --3
                               i:=i+1; --i=4
                                   do.pl(i); --4

```

### Program to print numbers from 4 to 1:



### for loop:

#### Syntax:

```

for <variable> IN [REVERSE]<lower>..<upper>
Loop
--Statements
End Loop;

```

### Program to print numbers from 1 to 4:

```

for i IN 1..4
Loop
do.pl(i);
End Loop;

```

i
1
2
3
4

### Program to print numbers from 4 to 1:

i	
4	1
3	2
2	3
1	4

```

for i in REVERSE 1..4
Loop
    do.pl(i);
End Loop;

```

**Note:**

- We have no need to declare loop variable in case of "for".
- Implicitly it will be declared as number type.**

**Simple Loop:**

```

Loop
--Statements
EXIT / EXIT WHEN <condition>;
End Loop;

```

**Program to print numbers from 1 to4 using simple loop**

i	
1	i:=1;
2	
3	
4	

```

Loop
    do.pl(i); -- 1 2 3 4 5
    exit when i=5;
    i:=i+1;
End Loop;

```

**continue:**

**is used to skip current iteration & continue the next iteration**

```

BEGIN
for i IN 1..10
Loop
    if i=7 then

```

```

continue ;
END IF;
dbms_output.put_line(i);
End Loop;

```

```

END ;
/

```

**goto:**

**is used to transfer the control to specified label.**

**2 ways:**

- **Jumping Backward**
- **Jumping Forward**

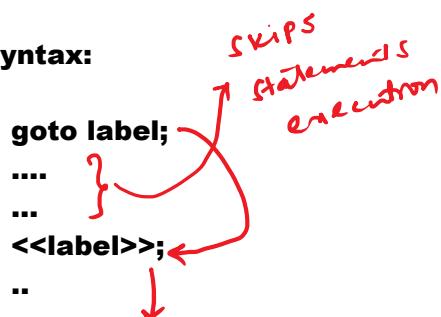
**Jumping Backward**

**Syntax:**



**Jumping Forward:**

**Syntax:**



**Program to print numbers from 1 to 4 using**

**goto:**

i
1
2
3
4

```

i:=1;

<<nareshit>>
do.pl(i); -- 1
i:=i+1; -- i=2
if i<=4 then
  goto nareshit;
end if ;

```

**Program to demonstrate jumping forward:**

```
do.pl('hi');
goto xyz;
  do.pl('hello');
<<xyz>>
do.pl('welcome');
```

hi  
welcome

**Program to print comm of given empno:**

**Declare**

```
v_empno emp.empno%type;
v_comm emp.comm%type;
```

**Begin**

```
  v_empno := &empno;
```

enter empno:7369  
1000

```
  Select comm into v_comm
  from emp
  where empno=v_empno;
```

enter empno: 7521

```
  if v_comm is null then
    do.pl('comm is null');
  else
    do.pl('comm is:' || v_comm);
  end if ;
```

**End ;**

/

**Program to print emp details of given empno:**

**emp**

empno	ename	job	sal	mgr	comm	hiredate	deptno
1002	ALLEN	MANAGER	5000	1008	300	12-MAR-81	30

```
enter empno: 1002
1002 ALLEN MANAGER 5000 1008 300 12-MAR-81 30
```

```
Select ename,sal into v_ename,v_Sal from emp  
where empno=v_empno;
```

**%rowtype:**

- is an attribute related data type.
- It is used to hold one row of a table.
- It can hold one row only at a time. It cannot hold multiple rows.

int  
number(7,2)

varchar2(10)

date

**Syntax:**

```
variable table_name%rowtype;
```

**%type => used to declare a variable with table column's data type & size;**

**Ex:**

```
v_empno emp.empno%type;
```

**Ex:**

```
r emp%rowtype;
```

**%rowtype**

**r**

empno	ename	job	sal	mgr	comm	hiredate	deptno
1002	ALLEN	MANAGER	5000	1008	300	12-MAR-81	30

```
Select * into r from emp  
where empno=1002;
```

r.empno => 1002  
r.ename => ALLEN  
r.job => MANAGER

**Advantages of %rowtype:**

- It decreases number of variables
- It reduces the complexity

```
class Student  
{  
    rno  
    name  
    m1  
    m2  
    m3  
}
```

```
Student s1 = new Student();
```

### Student

sid	sname	m1	m2	m3
1001	Ravi	60	80	70
1002				
1003				

enter sid: 1001

1001 Ravi 60 80 70

r student%rowtype;

Select \* into r

from student

where sid=1001;

sid	sname	m1	m2	m3
1001	Ravi	60	80	70

l. sname  $\rightarrow$  Ravi

l. m2  $\Rightarrow$  80 ✓

Display the emp record of given empno:

### emp

empno	ename	job	sal	mgr	comm	hiredate	deptno
1002	ALLEN	MANAGER	5000	1008	300	12-MAR-81	30

enter empno: 1002

1002 ALLEN MANAGER 5000 1008 300 12-MAR-81 30

### DECLARE

v\_empno emp.empno%type;

r emp%rowtype;

begin

v\_Empno := &empno;

Select \* into r from emp

```

where empno=v_empno;

do.pl(r.ename || ' ' || r.sal || ' ' || r.hiredate);
end ;
/

```

**Program to calculate total, avrg & result of a student:**

**STUDENT**

sid	sname	M1	M2	M3
1001	Ravi	50	90	70
1002	Srinu	60	30	80

**RESULT**

sid	Tot	Avrg	Result
1001			

**Create Table Student**

```

(
    sid number(4),
    sname varchar2(10),
    m1 number(3),
    m2 number(3),
    m3 number(3)
);

```

**SQL> insert into student  
values(1001,'A',60,50,90);**

**1 row created.**

**SQL> insert into student  
values(1002,'B',50,30,60);**

**1 row created.**

**Create table result**

```

(
    sid number(4),
    tot number(3),
    avrg number(5,2),
    result varchar2(10)
);

```

```

v_sid
DECLARE
    v_sid student.sid%type;
    r1 student%rowtype;
    r2 result%rowtype;
BEGIN
    v_sid := &sid; --1001
    Select * into r1
    from student where
    sid=v_sid;
    r2.tot := r1.m1+r1.m2+r1.m3;
    r2.avrg := r2.tot/3;
    if r1.m1>=40 and r1.m2>=40 and r1.m3>=40 then
        r2.result:='PASS';
    else
        r2.result:='FAIL';
    end if ;
    Insert into result values(v_sid,r2.tot, r2.avrg, r2.result);
    commit ;
END ;
/

```

r1

sid	sname	m1	m2	m3
01	A	60	50	90

r2

sid	tot	avrg	result

result

sid	tot	avrg	result

### Program to print calendar:

```

DECLARE
    d1 date;
    d2 date;
BEGIN
    d1 := '1-JAN-2022';
    d2 := '31-DEC-2022';

    while d1<=d2
    loop
        do.pl(d1 || ',' || to_Char(d1,'dy'));
        d1:=d1+1;
    end loop;
END;

```

1-jan-22, sat	i
2-jan-22, sun	1
	2
	3
	4
i:=1;	
i:=i+1;	

## Cursors

Monday, December 27, 2021 7:06 PM

- to hold one value use %type.

Ex:

```
v_empno emp.empno%type;
```

emp			
empno	ename	job	sal
7369	SMITH	clerk	4000
7900	ALLEN	mgr	8000

v\_empno

7369

- to hold entire row use %rowtype.

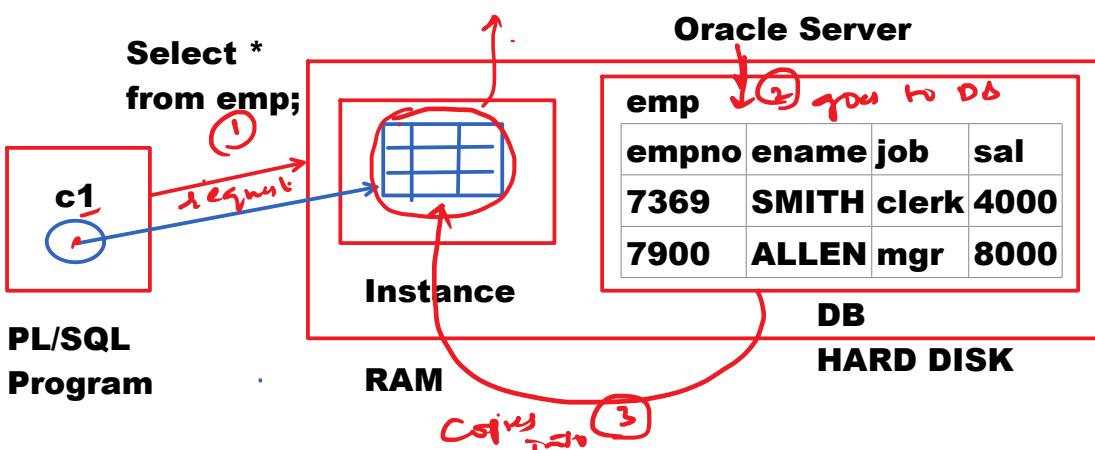
Ex:

```
r emp%rowtype;
```

r

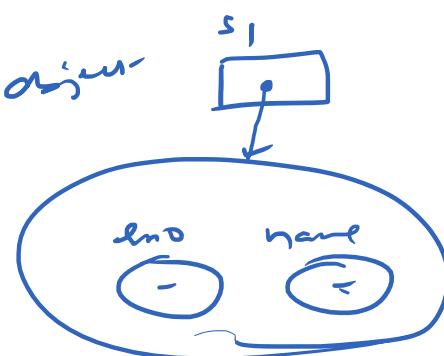
empno	ename	job	sal
7369	SMITH	clerk	4000

- to hold a set of records, use Cursor.



```
class Student
{
    rno
    name
}

Student s1;
s1 = new Student();
```



- **Cursor is a pointer to the memory location in Oracle Instance.**
- **Cursor is used to process multiple rows.**
- **Cursor can process one row at a time. To process multiple rows use loop.**

**To use Cursor, we follow 4 steps:**

- **Declare**
- **Open**
- **Fetch**
- **Close**

**Declaring Cursor:**

**Syntax:**

**Cursor <cursor\_name> IS <Select statement>;**

**Ex:**

**Cursor c1 IS Select ename,sal from emp;**

**When we declare the cursor, oracle identifies the cursor name & associated select query.**

**Opening Cursor:**

**Syntax:**

**Open <cursor\_name>;**

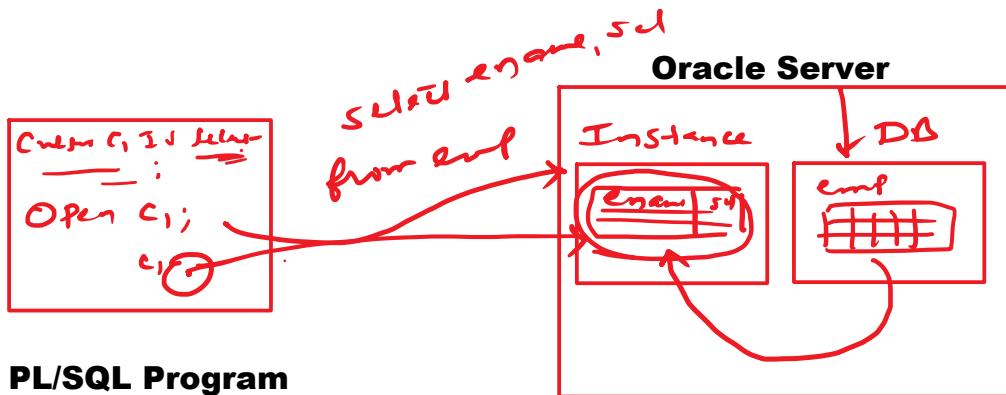
**Ex:**

**Open c1;**

**When cursor is opened, implicitly oracle runs select query associated with the**

**cursor.**

**Oracle goes to database, picks the data & copies into oracle instance. This memory location reference will be given to cursor variable.**



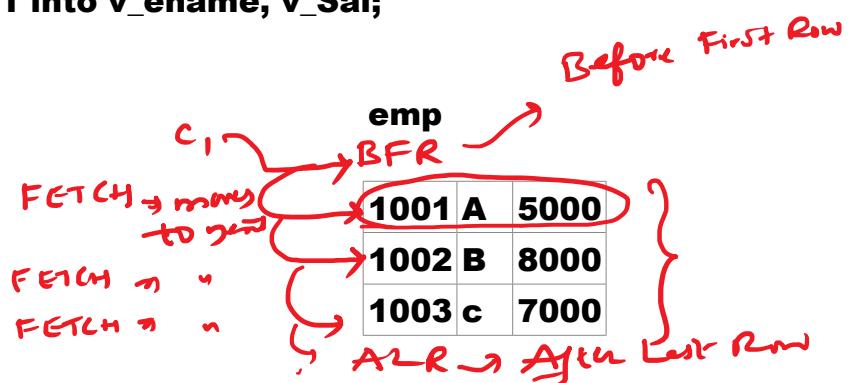
### Fetching records:

**Syntax:**

**Fetch <cursor\_name> into <variable\_list>;**

**Ex:**

**Fetch c1 into v\_ename, v\_Sal;**



**When fetch is executed for first time, it moves to first record.**

**When fetch is executed for second time, it moves to second record.**

**When FETCH is executed, it moves to next record.**

**At a time a cursor can access one row only. To access multiple rows**

**write FETCH statement in loop.**

### **Closing Cursor:**

**Syntax:**

**Close <cursor\_name>;**

**Ex:**

**Close c1;**

**When cursor is closed, reference to memory location in oracle instance will be gone.**

### **Cursor Attributes:**

**4 attributes:**

- **%FOUND**
- **%NOTFOUND**
- **%ROWCOUNT**
- **%ISOPEN**

**TO use cursor attributes we use following syntax:**

**Cursor\_name%attribute\_name**

**Ex:**

**c1%FOUND**

**c1%NOTFOUND**

**c1%ROWCOUNT**

**c1%ISOPEN**

**%FOUND:**

- **It returns TRUE or FALSE.**
- **When fetch is successful, it returns TRUE.**

- When fetch is unsuccessful, it returns FALSE.

**%NOTFOUND:**

- It returns TRUE or FALSE.
- When fetch is unsuccessful, returns TRUE.
- When fetch is successful, it returns FALSE.

**%ROWCOUNT:**

- returns number of rows affected by the query.

**%ISOPEN:**

- returns TRUE or FALSE.
- When cursor is open, it returns TRUE.
- when cursor is not open, it returns FALSE.

**Program to display all emp names and salaries:**

```
DECLARE
  CURSOR c1 IS Select ename,sal from emp; --DECLARE
BEGIN
  OPEN c1;

  LOOP
    FETCH c1 into v_ename, v_sal;
    EXIT WHEN c1%NOTFOUND;
    do.pl(v_ename || ' ' || v_sal);

  END LOOP;

  Close c1;

END ;
```

/

### **Program to display all emp records:**

```
DECLARE
  CURSOR c1 IS SELECT * FROM
  emp;
  r emp%rowtype;
BEGIN
  OPEN c1;

  Loop
    FETCH c1 INTO r;
    EXIT WHEN c1%NOTFOUND;
    dbms_output.put_line(r.empno
    || ' ' || r.ename || ' ' ||
    r.job || ' ' || r.sal);
  END Loop;

  CLOSE c1;

END ;
/
```

### **Cursor For Loop:**

#### **Syntax:**

```
for <variable> IN <cursor_name>
Loop
  --Statements
End Loop;
```

```
for i IN 1..10
Loop
  do.pl(i);
End Loop;
```

```
for r IN c1
Loop
  do.pl(r.ename || ' ' || r.sal);
End Loop;
```

**we have no need to declare r  
r data type will be taken as  
%rowtype implicitly.**

**If we use, Cursor for loop we have no need to open the cursor, fetch records from cursor & close the cursor.**

**These 3 things will be done implicitly.**

**Program to display all emp records using cursor for loop:**

```
DECLARE
    CURSOR c1 IS SELECT * FROM
        emp;
BEGIN

    FOR r IN c1
        Loop
            dbms_output.put_line(r.empno
                || ' ' || r.ename || ' ' || r.sal);
        END Loop;
    END ;
/
```

**Program to find sum of salaries all employees using cursor:**

sal	
5000	
3000	
8000	
6000	

```
DECLARE
    Cursor c1 IS Select sal from emp;
        v_sal emp.sal%type;
        v_sum FLOAT ;
BEGIN
    Open c1;
        v_sum:=0;

    Loop
        FETCH c1 into v_sal;
        exit when c1%notfound;
        v_sum:=v_sum+v_sal;
    End Loop;
        do.pl('sum of salaries=' || v_sum);
    END ;
/
```

**Find max salary in emp table.**

**assume that first sal is max sal**

**compare second sal with max sal  
if second sal is > max sal then  
second sal should become max sal.**

**Program to display all emp names by separating them using ,:**

**ALLEN, WARD, SMITH, ..**

**In Oracle 11g version,**

**Select  
Listagg(ename,', ') within group  
(order by empno)**

**Program to increase the salaries of emps as following:**

**emp**

<b>empno</b>	<b>ename</b>	<b>job</b>	<b>sal</b>
1001			8000
1002			7000
1003			10000

**hike**

empno	per
1001	10
1002	20
1003	15

**Program to find result of the student:**

**student**

sid	sname	M1	M2	M3
1001				
1002				

**Result**

sid	tot	avrg	result
1001			
1002			

**Program to display top 5 salaried emps:**

**declare****cursor c1 is Select \* from emp order by sal****desc;****r emp%rowtype;****begin****open c1;****loop****fetch c1 into r;****exit when c1%notfound;**

```
if c1%rowcount<=5 then
    do.pl(r.empno || ' ' || r.ename || ' ' || r.sal);
end if ;
end loop;
close c1;
end ;
/  

```

**Display the emp record whose sal is  
5th maximum:**

**Oracle 12c version,**

**FETCH clause:  
is used to get top rows**

**Select \* from emp order by sal desc  
fetch first five rows only;**

**display 5th max sal emp record:**

**Select \* from emp order by sal desc  
offset 4 rows fetch next 1 rows only;**

**Ref Cursor [Dynamic Cursor]:**

- Simple cursor can be used for one SELECT statement only. It cannot be used for many SELECT statements.
- To use Same Cursor for many select statements, we use "Ref Cursor".
- "sys\_refcursor" data type is used to declare ref cursor variable.
- At the time of opening the cursor we specify select statement.

### **Declaring ref cursor variable:**

**Syntax:**

```
<cursor_variable_name> sys_refcursor;
```

**Ex:**

```
c1 sys_refcursor;
```

### **Opening ref cursor:**

**Syntax:**

```
open <cursor_name> for <select_Statement>;
```

**Ex:**

```
open c1 for select * from emp;
```

**Program to demonstrate ref cursor.**

**Same cursor we want to use to process emp table records & dept table records:**

<b>Simple Cursor</b>	<b>Ref Cursor</b>
<ul style="list-style-type: none"> <li>• <b>it can be used for 1 select statement only</b></li>   <li>• <b>No data type will be used</b></li>   <li>• <b>Select statement will be specified at the time of declaring the cursor.</b></li>   <li>• <b>static</b></li>     <li>• <b>it cannot be used as parameter for procedure or function</b></li> </ul>	<ul style="list-style-type: none"> <li>• <b>it can be used for multiple statements</b></li>   <li>• <b>"sys_refcursor" data type is used.</b></li>   <li>• <b>Select statement will be specified at the time of opening the cursor.</b></li>   <li>• <b>dynamic</b></li>     <li>• <b>can be used as parameter in case of procedures &amp; functions</b></li> </ul>

```
add(int x, int y)
{
}
```

```
add(10,20);
```

```
procedure p1(c sys_refcursor)
```

#### **Cursor with Parameters:**

- **A cursor which is declared using parameters is called "Parameterized cursor".**
- **Parameter value will be passed when we open the cursor.**

```

DECLARE
  Cursor c1(d INT) is Select * from emp where
  deptno=d;
  r emp%rowtype;
BEGIN
  Open c1(20);

  loop
    fetch c1 into r;
    exit when c1%notfound;
    do.pl(r.empno || ' ' || r.ename || ' ' || r.deptno);
  end loop;

  close c1;

END ;
/

```

### Inline Cursor:

```

BEGIN
  for r in (Select * from emp)
  loop
    do.pl(r.empno || ' ' || r.ename || ' ' || r.sal);
  end loop;
END;
/

```

**Cursor for loop:**  
**no need to**  
**open**  
**fetch**  
**close**

**Types of Errors:****3 Types:**

- **Compile Time Errors**
- **Logical Errors**
- **Run-Time Errors**

**• Compile Time Errors:**

- These errors occur due to syntax mistakes like missing ( ; '
- end if missed
- end loop missed

**Logical Errors:**

- These errors occur due to mistake in the logic.
- It leads to wrong output.
- Ex: Withdraw => bal + amount => logical error

**Run-Time Errors:**

These errors occur at run time due to several reasons like:

Ex:

- divide with zero
- no data found
- wrong input
- size is exceeded

Run-Time errors are very dangerous. When run-time error occurs our application will be closed in the middle of execution. It cannot execute remaining code.  
So, user cannot continue his work. It raises inconvenience to the user.

**Run-Time Error must be handled.**

**Exception => Run-Time Error****Exception Handling:**

- The mechanism of handling run-time errors is called "Exception Handling".
- For handling Exception, we write "Exception" block in PL/SQL.

**Syntax for Exception Handling:**

PL/SQL API

... when PULL

```
try
{
    //code which raises RTE
}
catch(ArithmeticException e)
{
```

### Syntax for Exception Handling:

```
DECLARE
  <declaration statements>
}
BEGIN
  <executable statements>
}
EXCEPTION
  When <Exception_Name> Then
    --Handling Code
  When <Exception_Name> Then
    --Handling Code
  .
  .
END ;
/
```

Declaration Part  
Execution Part

```
}
catch(ArithmeticException e)
{
  //handling code
}
```

### Program to demonstrate Exception Handling:

There are 2 types of Exceptions:

**Predefined exceptions =>**

**These exceptions already declared by oracle developers & it will be raised implicitly by the oracle.**

**x:**

```
ZERO_DIVIDE
VALUE_ERROR
NO_DATA_FOUND
TOO_MANY_ROWS
DUP_VAL_ON_INDEX
```

**user-defined exceptions =>**

**These exceptions are declared by the user & it will be raised explicitly.**

**Ex:**

```
comm_is_null
ONE_DIVIDE
```

**ZERO\_DIVIDE:**

**it will be raised when we try to divide with ZERO.**

**VALUE\_ERROR:**

**it will be raised when size is exceeded or data type mismatched.**

**NO\_DATE\_FOUND:**

**It will be raised when record is not found in the table.**

**OTHERS:**

**It can handle any type of exception.**

**TOO\_MANY\_ROWS:**

**It will be raised when a select query returns multiple rows**

**Program to display emp details of given empno. If record is not found run time error will occur. Handle it:**

**Create table t1(f1 number(4) primary key);**

**program to insert records into t1 table. if user tries to insert duplicate value RTE will occur. Handle it.**

**DUP\_VAL\_ON\_INDEX:**

**It will be raised when user tries to insert duplicate values in PK column.**

**ZERO\_DIVIDE  
VALUE\_ERROR  
NO\_DATA\_FOUND  
TOO\_MANY\_ROWS  
DUP\_VAL\_ON\_INDEX**

**User-Defined Exceptions:**

- We can define our own exceptions like predefined exceptions [ZERO\_DIVIDE].**
- This exception will be raised explicitly whereas predefined exception will be raised implicitly by the oracle.**

**Defining user-defined exception:**

**2 steps:**

- Declare the Exception**
- Raise the Exception**

- **Declare the Exception:**  
To declare the exception name use "Exception" Type.

Syntax:

```
<exception_name> Exception;
```

Ex:

```
comm_is_null Exception;
xyz Exception;
ONE_DIVIDE Exception;
```

- **Raise the Exception:**  
We can raise the exception using "RAISE" keyword.

Syntax:

```
RAISE <exception_name>;
```

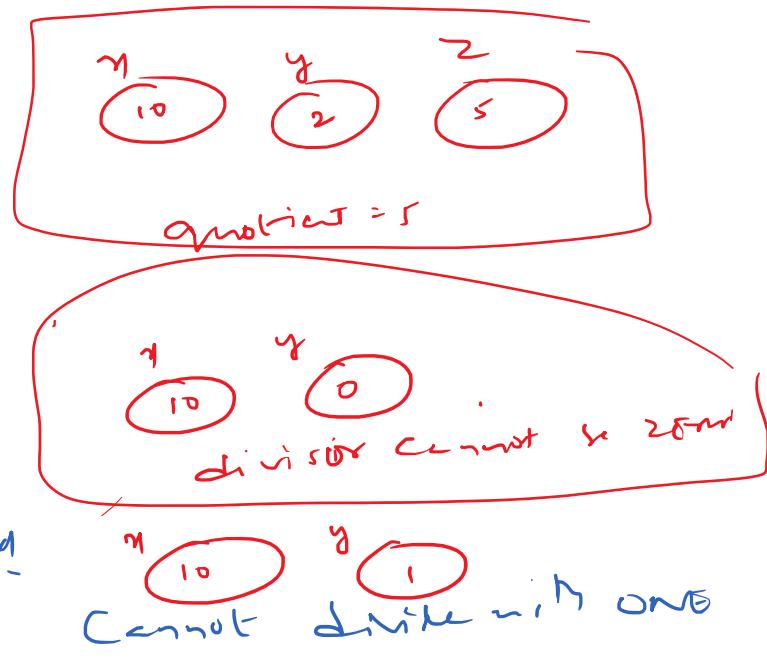
Ex:

```
RAISE comm_is_null;
RAISE xyz;
RAISE ONE_DIVIDE;
```

**Program to demonstrate user-defined exception:**

Define user-defined exception ONE\_DIVIDE:

```
DECLARE
    x number(4);
    y number(4);
    z number(4);
    one_divide Exception; -- ①
    declaring exception
BEGIN
    x := &x; →
    y := &y;
    IF y=1 THEN → ②
        RAISE one_divide;
    END IF ;
    z := x/y;
    dbms_output.put_line('quotient='
    || z);
EXCEPTION
    WHEN zero_divide THEN
        dbms_output.put_line('divisor
        cannot be ZERO');
    WHEN one_divide THEN
        dbms_output.put_line('cannot
        divide with ONE');
END ;
/
```



**Program to update salary of given empno with given amount. If emp comm is null don't update his salary & raise the exception:**

**RAISE\_APPLICATION\_ERROR():**

- It is a procedure.
- It is used to raise the error.
- It does not handle the error.

**Exception can be raised in 2 ways:**

- using RAISE keyword
- using RAISE\_APPLICATION\_ERROR procedure

**Syntax:**

**RAISE\_APPLICATION\_ERROR(<error\_number>,<error\_message>);**

**Using RAISE\_APPLICATION\_ERROR we define our own error numbers & error messages.**

**Error will be displayed as oracle error style.**

```
DECLARE
    x Number(4);
    y Number(4);
    z Number(4);
BEGIN
    x := &x;
    y := &y;
    z := x/y;
    dbms_output.put_line('quotient='
    || z);
END ;
```

**Note:** /  
**For error, error number & error message will be there. For some errors names will be there.**

**Program to update employee salary**

**of given empno with given amount.**

**If comm is null raise the error**

**using RAISE\_APPLICATION\_ERROR:**

**for user-defined error numbers valid range  
is: -20000 to -20999**

RAISE	RAISE_APPLICATION_ERROR
<b>is a keyword</b>	<b>is a procedure</b>
<b>it raises the error &amp; it can be handled</b>	<b>Just it raises the error. it cannot be handled.</b>
<b>exception is raised using name</b>	<b>exception is raised using error number.</b>

**Program to not to allow the user to update on  
Sunday. If user tries to update the data on  
Sunday raise\_application\_error:**

**Pragma Exception\_Init():**

- Every error has error number & error message.
- Not all errors are having names.
- Some errors are having names.
- Ex: unique constraint violation => DUP\_VAL\_ON\_INDEX
- Some errors are not having names.
- Ex: check constraint violation => No Exception name
- To handle Run-Time Error in exception block name is required. If name is not there we cannot handle it in exception block. Because of this reason, to define names to unnamed errors, we use "Pragma Exception\_Init()".

**Pragma Exception\_Init():**

**Pragma => Directive [ Command ]**

**It tells that before going to compile execute this line.**

**Exception\_Init() => It is a function used to define names to  
error numbers.**

**Pragma Exception\_Init(<Exception\_name>,<error\_number>);**

```
check_violate Exception;
Pragma Exception_Init(check_violate,-2290);
```

**Before going to compile the program, "check\_violate"  
exception name will be defined to the  
error number -2290.**

**Program to insert arecord into std table:**

```
std
sid [primary key] sname [not null] m1 [check]
```

**To define names to unnamed  
errors, use Pragma Exception\_Init()**

## **Stored Procedures**

Monday, January 3, 2022 6:16 PM

**In PL/SQL, there are 2 types of blocks:**

- **Anonymous Blocks => Block without name**
- **Named blocks => Block with the name**

**Named Blocks:**

- **Procedures**
- **Functions**
- **Packages**
- **Triggers**

**Sub Programs:**

**Procedures & functions are called sub programs.**

**Advantages of sub-programs:**

- **Modularity**
  - **improves understandability**
  - **decreases length of code**
  - **provides reusability.**
- **provides security**
- **improves the performance**
- **we can call from front-end applications like java or .net appln.**

**• Modularity:**

- **it is programming style**
- **We divide large program into small parts. Every part is called "procedure" or "function".**

**Bank Project**

**opening\_Account**  
**deposit**  
**withdraw**  
**check\_balance**

### **Advantages:**

- improves understandability.
- decreases length of code.
- it provides reusability.

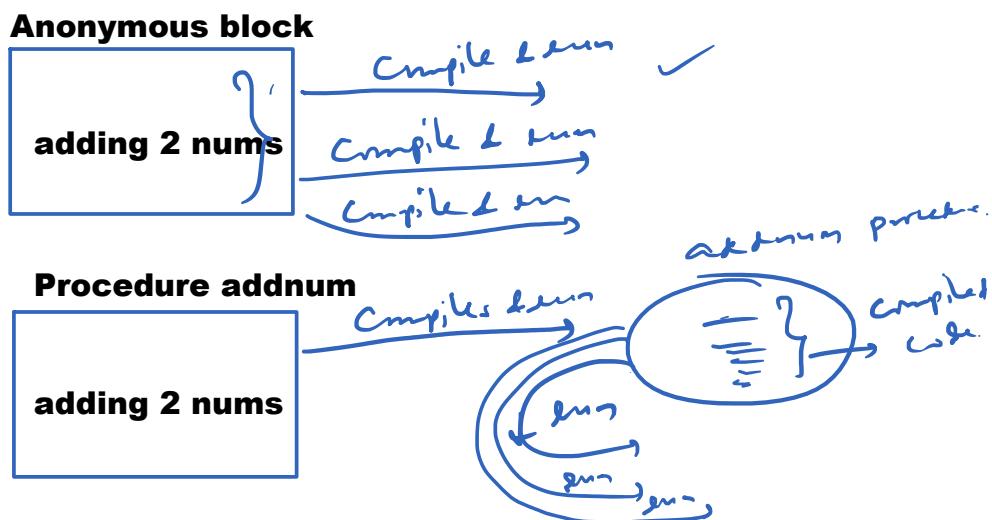
**dbms\_output.put\_line()**

**dbms\_output => package  
put\_line() => procedure**

### **provides security:**

- procedures & functions are stored in centralized location.
- Only authorized users can use them.

### **Improves the performance:**



**Procedure contains compiled code. From 2nd time onwards this compiled code will be executed. Every time code will not be compiled. So, automatically performance will be improved.**

**We can call from front-end applications like java, .net apps.**

### **Procedure:**

- Procedure is a named block of statements that

**gets executed on calling.**

**There are 2 types of procedures:**

- **Stored Procedures**
- **Packaged Procedures**

**Stored Procedure:**

**A procedure which is defined inside of schema then it is called "Stored Procedure".**

**Packaged Procedure:**

**A procedure which is defined inside of a package is called "Packaged Procedure".**

**Ex:**

**dbms\_output => package**  
**put\_line() => procedure**

**dbms\_output package**

**Create or Replace Procedure**  
**put\_line ...**  
**IS**  
**Begin**  
**...**  
**End ;**  
**/**

**put\_line() is packaged procedure**

**dbms\_mview**

**Procedure refresh**

**refresh() is packaged procedure**

**Stored Procedure**

**c##Oracle6PM**

**Procedure update\_Salary => Stored Procedure**

**c##OraclePM**

**Package HR**

**Schema [user]**

**Schema [user]**

**Package**

c##OraclePM → Schema [user]  
↳ Package HR → Package.  
↳ Procedure update\_Salary => **Packaged procedure**

### Stored Procedure:

#### Syntax:

```
Create Or Replace Procedure <name>([<parameter_list>])
IS / AS
  <declaration_part>;
BEGIN
  <execution-part>;
END ;
/
```

### Define a procedure to add 2 numbers:

#### Parameter [Argument]:

A local variable declared in procedure header is called "Parameter".

#### Syntax:

```
<parameter_name> [<parameter_type>] <data_type>;
```

**Note: Don't specify field size for parameters**

```
Create Or replace Procedure addnum(x number,y number)
IS
  z number(4);
Begin
  z:=x+y;
  dbms_output.put_line('sum=' || z);
End;
/
```

**A procedure can be called from 3 places:**

- **From SQL Prompt**
- **From Another PL/SQL Program**
- **From Front-End Appln like Java, .NET Appln.**

**Calling from SQL Prompt:**

**Execute (or) exec command is used to call the procedure from SQL Prompt.**

```
SQL> execute addnum(2,3);
sum=5
```

**Calling From Another PL/SQL Program:**

**DECLARE**

```
a int ;
b int ;
```

**BEGIN**

```
a := &a;
b := &b;
```

```
addnum(a,b); --calling from PL/SQL program
```

**END;**

**/**

**Parameter [Argument]:**

**A local variable declared in procedure header is called "Parameter".**

**Syntax:**

```
<parameter_name> [<parameter_type>] <data_type>;
```

**There are 3 types of parameters:**

- **IN [defualt]**

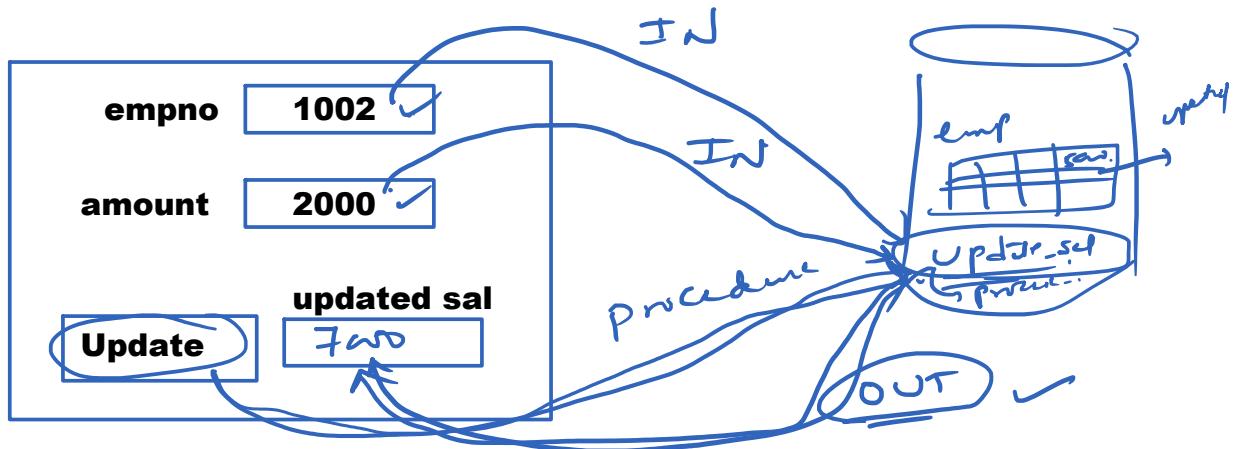
- OUT
- IN OUT

### IN parameter:

- It can be also called as "Read-Only Parameter".
- To get the value into procedure, we use it.

### OUT parameter:

- it is used to send value out of the procedure.
- In procedure call, it must be variable only. it cannot be constant.

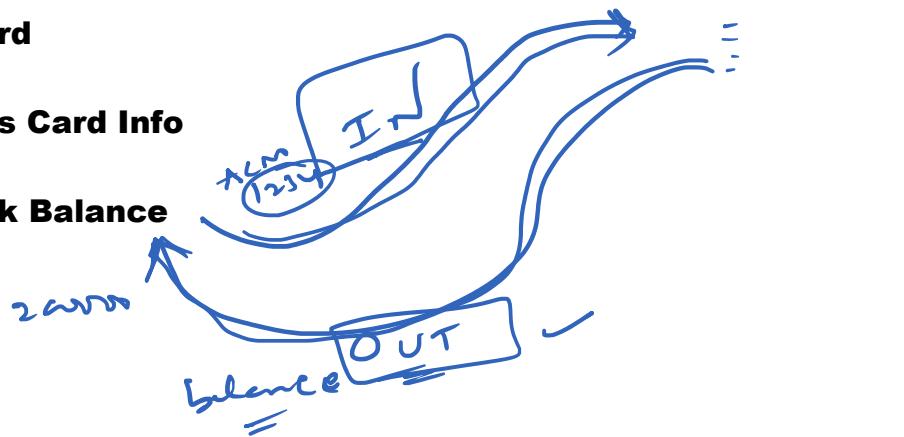


ATM

Insert Card

Reads Card Info

Check Balance



### IN OUT Parameter:

- it can be used to get value into procedure & send the value out of procedure.

**Define a procedure to add 2 nums. And the result out of procedure using OUT parameter:**

```
Create Or Replace Procedure
addnum(x IN number, y IN number,z OUT number)
IS
BEGIN
    z := x+y;
END ;
/
```

**Bind Variable:**

- A variable which is declared in SQL prompt is called "Bind Variable".
- To access bind variable we use : operator.

**Calling from SQL Prompt:**

```
SQL> variable a number
SQL> execute addnum(5,4,:a);
SQL> print a
```

```
A
-----
9
```

**Parameter mapping techniques**  
**[Different ways of passing parameters]:**

```
Create Or Replace Procedure
addnum(x number,y number,z number)
IS
    a number;
BEGIN
    a:=x+y+z;
    dbms_output.put_line('sum=' || a);
```

**END ;**

/

**There are 3 Parameter mapping techniques:**

- **Positional Notation**
- **Named Notation**
- **Mixed Notation**

**Positional Notation:**

**addnum(x number,y number,z number)**  
SQL> exec addnum(10,20,30);  
**Output:**  
**sum=60**

*Based on position of arguments, values are mapped*

• **Named Notation:**

**addnum(x number,y number,z number)**  
SQL> exec addnum(y=>10,z=>20,x=>30);

*Based on names, values are mapped*

**Mixed Notation:**

**addnum(x number,y number,z number)**  
SQL> addnum(10,z=>20,y=>30);

**SQL> addnum(y=>10,20,30); //ERROR**

**Positional argument cannot be followed by named argument.**

### **Procedure to increase salary of an employee:**

```
Create Or Replace Procedure
Update_Salary(p_empno emp.empno%type,
p_amt float)
IS

BEGIN
    update emp set sal=sal+p_amt where
    empno=p_empno;
    commit ;

    dbms_output.put_line('salary updated..!');
END ;
/  
  

Update_Salary(1002,2000);
  

Update_Salary(1005,3000);
```

### **Procedure to increase the salary of an employee & send updated salary out of the procedure:**

```
Create Or Replace Procedure
update_salary(p_empno emp.empno%type,
p_amt float, p_sal OUT emp.sal%type)
IS
BEGIN
    Update emp set sal=sal+p_amt where
    empno=p_empno;
    commit ;
    dbms_output.put_line('salary updated..');

    Select sal into p_sal from emp where
    empno=p_Empno;
END ;
/
```

```
Create table std
(
    sid number(4) primary key,
    sname varchar2(10) not null,
    m1 number(3) check(m1>=0 and m1<=100)
);
```

**Define a procedure to insert records into std table:**

```
Create Or Replace Procedure
insert_std(p_sid std.sid%type,
p_sname std.sname%type,
p_m1 std.m1%type,
p_msg OUT varchar2)
IS
BEGIN
    Insert into std values(p_sid,p_sname,p_m1);
    p_msg:='record inserted..!';
```

```
EXCEPTION
    When others then
        p_msg:=SQLERRM;
END ;
/
```

```
SQL> variable s varchar2(500)
SQL> exec insert_std(1005,'C',56,:s);
SQL> print s
s
-----
record inserted..!
```

```
SQL> print s
s
-----
check constraint violated
```

**SQLERRM:**

**is a built-in function that returns error message.**

## **Account**

<b>Acno</b>	<b>Name</b>	<b>Balance</b>
<b>1001</b>	<b>A</b>	<b>110000</b>
<b>1002</b>	<b>B</b>	<b>200000</b>

**Define a procedure to deposit amount into an account:**

```
Create Or Replace Procedure
deposit(p_acno account.acno%type,p_amt float)
is
begin
  Update account set balance=balance+p_amt
  where acno=p_acno;
  commit ;

  dbms_output.put_line('amount credited..!');
end ;
/
```

**Define a procedure to perform withdraw operation:**

```
Create Or Replace Procedure
withdraw(p_Acno account.acno%type,
p_amt float,
p_bal OUT account.balance%type)
IS
  v_bal account.balance%type;
BEGIN
  Select balance into v_bal from account
  where acno=p_Acno;
```

```

If p_amt>v_bal then
  Raise_Application_Error(-20050,'insufficient balance..!');
else
  update account set balance=balance-p_amt where
  acno=p_acno;
  commit;
  dbms_output.put_line('amount debited..!');
end if ;

Select balance into p_bal from account
where acno=p_acno;
END ;
/

```

### **Pragma Autonomuous\_Transaction:**

#### **Pragma => Directive [Command]**

- **By default a separate transaction will not be created for a procedure.**
- **The transaction started in main program is applied for procedure also.**
- **To create separate transaction for the procedure, we use "pragma autonomous\_transaction".**

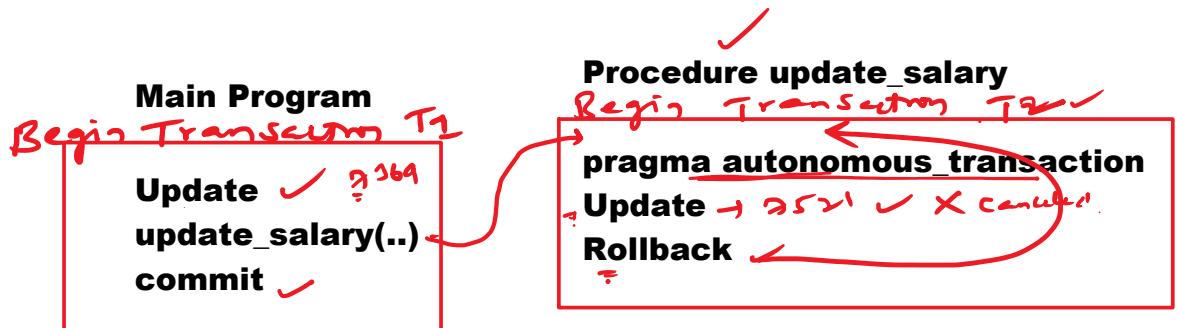
**Transaction:**  
**is a series of actions**

**Deposit Transaction**  
**Withdraw Transaction**  
**Checkbalance Transaction**  
**placing\_order Transaction**

**Transaction must be successfully completed or aborted [canceled].**

If Transaction is successfully completed => COMMIT

If Transaction should be canceled => Rollback



## User-Defined Functions

Wednesday, January 5, 2022 6:52 PM

### Built-In Functions:

Oracle developers already defined some functions. these are called "built-in functions".

Ex:

`lower() upper()  
sum() avg() min()`

### User-defined functions:

we can define our own functions.  
these are called "user-defined functions".

### 2 Types:

**Stored Functions**

**Packaged Functions**

#### Stored Function:

A function which created inside of schema [user] then it is called **Stored function**.

Ex:

```
c##oracle6pm  
      addnum => stored function
```

#### Packaged Function:

A function which is defined inside of package is called "Packaged Function".

Ex:

```
c##oracle6pm  
      package mymath  
          function addnum => packaged function
```

### Syntax to define a function:

**Create Or Replace Function <name>([<parameter\_list>])**

**Return <type>**

**IS / AS**

**<declaration\_part>**

**Begin**

```
<declaration_part>
Begin
  <execution_part>
End;
/
```

**Procedure:**

- **is a set of statements that gets executed on calling.**
- **is used to perform DML Operations [insert / Update /Delete ]**
- **Procedure cannot be called from SQL Command**

**Function:**

- **is a set of statements that gets executed on calling.**
- **is used to perform fetch operations [SELECT] or calculations.**
- **Function can be called from SQL Command.**

<b>Opening_Account</b>	<b>=&gt; Insert =&gt; Procedure</b>
<b>Withdraw</b>	<b>=&gt; UPDATE =&gt; Procedure</b>
<b>Deposit</b>	<b>=&gt; UPDATE =&gt; Procedure</b>
<b>Check_balance</b>	<b>=&gt; SELECT =&gt; Function</b>
<b>trans_statement</b>	<b>=&gt; SELECT =&gt; Function</b>
<b>Closing_Account</b>	<b>=&gt; DELETE =&gt; PROCEDURE</b>

**Define a function to add 2 numbers:**

**Create Or Replace Function**  
**addition(x number, y number)**  
**return number**  
**is**  
**begin**  
    **return x+y;**  
**end ;**  
**/**

**Function => must return the value**

**Function can be called in 3 ways:**

- **From SQL prompt**
- **From another SQL program**
- **From Front-End Applns like java, .net applns**

• **From SQL prompt:**

```
Select addition(5,4) from dual;
```

**From another PL/SQL program:**

```
DECLARE
  a number;
  b number;
  c number;
begin
  a:=&a;
  b:=&b;
  c:=addition(a,b);
  dbms_output.put_line('sum=' || c);
end ;
/
```

**Define a function to perform arithmetic operations:**

```
Create or Replace Function
calc(x number,y number,op char)           calc(5,2,'+');
return number                                calc(5,2,'-');
is
begin
  if op='+' then
    return x+y;
  elsif op='-' then
    return x-y;
  elsif op='*' then
    return x*y;
  elsif op='/' then
    return x/y;
  end if ;
```

```
end ;
```

```
/
```

**Define a function to get experience of an employee:**

**Create Or Replace Function**

```
getexp(p_empno emp.empno%type)
```

```
return number
```

```
is
```

```
v_hiredate date ;
```

```
begin
```

```
Select hiredate into v_hiredate from emp
```

```
where empno=p_empno;
```

```
return trunc(sysdate-v_hiredate/365);
```

```
end ;
```

```
/
```

**Define a function to check the balance of an account number:**

**Create or Replace Function**

```
check_balance(p_acno account.acno%type)
```

```
return number
```

```
is
```

```
v_bal account.balance%type;
```

```
begin
```

```
select balance into v_bal from account
```

```
where acno=p_acno;
```

```
return v_bal;
```

```
end ;
```

```
/
```

**define a function to to top-n salaried employees:**

**Create or Replace Function**

```
gettopn(n int) return sys_refcursor
```

```
is
```

```

c1 sys_refcursor;
begin
  Open c1 for Select * from emp order by sal desc
  fetch first n rows only;

  return c1;
end ;
/

```

**Define a function to display employees of a deptno:**

```

Create or Replace Function
getdept(p_Deptno emp.deptno%type)
Return sys_refcursor
IS
  c1 sys_refcursor;
BEGIN
  Open c1 for Select deptno,empno,ename from
  emp where deptno=p_Deptno;

  return c1;
END ;
/

```

**Define a function to check whether the year is leap year or not:**

```

Create Or Replace Function
isleap(y int) return Varchar2
is
  d date;
begin
  d := '29-feb-' || y;
  return 'Leap year';
Exception
  when others then
    return 'not a leap year';

```

```
end ;  
/
```

**Procedure:**

**to perform DML operations define 'PROCEDURE'**

**Function:**

**to perform FETCH [Select] operation or calculations**

**define FUNCTION**

**Can we perform DML operations using Function?**

**YES. It is not recommended.**

**If we define DML statement in function, we cannot  
call it from SQL prompt.**

**We can use it in another PL/SQL program.**

**Can we use OUT parameters in Function?**

**YES. It is not recommended. Because,  
function meaning will be changed.**

**Differences b/w Procedure & Function:**

<b>Procedure</b>	<b>Function</b>
<ul style="list-style-type: none"> <li><b>Procedure cannot return the value. But, to send values out of procedure, we use OUT parameters.</b></li> <li><b>used to perform DML operations</b></li> <li><b>To send values out of procedure, OUT parameter is used.</b></li> <li><b>it cannot be called from SELECT query</b></li> <li><b>To call a procedure from sql prompt "execute" command is used.</b></li> </ul>	<ul style="list-style-type: none"> <li><b>Function can return the value. Here, return is mandatory.</b></li> <li><b>used to perform fetch operations or calculations</b></li> <li><b>to send value out of the function, we use RETURN.</b></li> <li><b>It can be called from SELECT query</b></li> <li><b>We use SELECT query to call the function from SQL prompt.</b></li> </ul>

**Ex: Withdraw**

**Ex: check\_balance**

**Where functions, procedures, packages & triggers information will be stored?  
"user\_source"**

**to see list of procedures, functions, triggers & packages:**

```
select distinct name,type from user_Source;
```

**to see a procedure or function code:**

```
select text from user_Source  
where name='GETTOPN';
```

**dropping procedure:**

**drop procedure addnum;**

**dropping function:**

**drop function calc;**

## Packages

Friday, January 7, 2022 6:18 PM

### Package:

- is a DB Object.
- Package is a collection procedures, functions, variables, cursors ...etc.

### package Bank

opening_account	=> INSERT => PROCEDURE
withdraw	=> UPDATE => PROCEDURE
deposit	=> UPDATE => PROCEDURE
check_balance	=> SELECT => FUNCTION
tras_statement	=> SELECT => FUNCTION

### Advantages:

- We can group related functions & procedures.
- Better maintenance
- Better performance
- we can declare global variables. Global variables can be accessed from anywhere.
- We can overload packaged procedures & packaged functions whereas stored procedures & stored functions cannot be overloaded.
- provides security for members by making them private. private members can be accessed inside of package only. they cannot be accessed out of the package.

10 stored procedures

10 stored functions

Edit =>

5 stored procedures

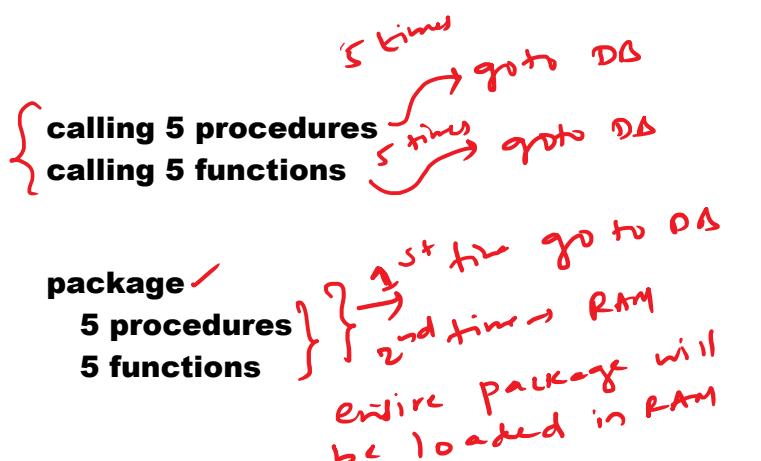
5 stored functions

Package

10 stored procedures

10 stored functions

open the package

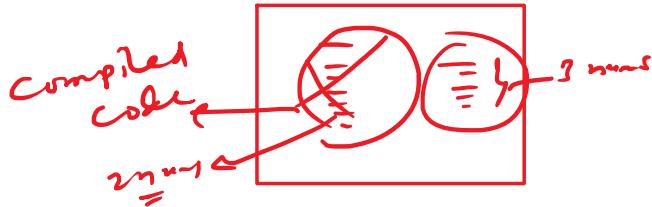


Create Procedure addition(x number,y number)

Create Or Replace Procedure addition(x number,y number,z number)

Procedure addition

## Procedure addition



**There are 2 types of Packages:**

- **Built-in Packages**
- **User-defined Packages**

- **Built-in Packages:**  
defined by oracle developers

**Ex: dbms\_output, dbms\_mview**

- **User-defined Packages:**  
We can define our own packages

**Ex: Bank HR nareshit**

**Defining User-Defined package:**

**2 steps:**

- **Define Package specification**
- **Define Package Body**

**Define Package specification:**

- all public members will be specified in this package specification.

**Syntax:**

```
Create Or Replace Package <package_name>
IS / AS
  PROCEDURE declarations
  FUNCTION declarations
END ;
/
```

```
END ;
```

```
/
```

### **Define Package Body:**

**The procedures & functions declared in package specification will be defined here.**

#### **Syntax:**

```
Create Or Replace Package Body <package_name>
IS / AS
define PROCEDURES
define FUNCTIONS
END ;
/
```

#### **Define a package with name "mymath" :**

```
package mymath
procedure addition
function product
```

#### **Define Package Specification:**

```
Create Or Replace package mymath
IS
procedure addition(x number, y number);
function product(x number,y number) return number;
END;
/
```

#### **Defining Package Body:**

```
Create Or Replace Package Body mymath
IS
procedure addition(x number, y number)
```

```

IS
BEGIN
  dbms_output.put_line('sum=' || (x+y));
END addition;

function product(x number,y number) return number
IS
BEGIN
  return x*y;
END product;

END ;
/

```

**Calling Packaged procedure:**

**exec package\_name.procedure\_name**

**dbms\_output.put\_line()**

**exec mymath.addition(2,3);**

**Calling Package Function:**

**Select package\_name.function\_name from dual;**

**Select mymath.product(2,3) from dual;**

**Define a package with the name "HR" with  
following procedures & functions:**

**package HR**

<b>hire =&gt; INSERT</b>	<b>=&gt; PROCEDURE</b>
<b>fire =&gt; DELETE</b>	<b>=&gt; PROCEDURE</b>
<b>update_salary =&gt; UPDATE</b>	<b>=&gt; PROCEDURE</b>
<b>getTopN =&gt; SELECT</b>	<b>=&gt; FUNCTION</b>

<b>update_salary =&gt; UPDATE</b>	<b>=&gt; PROCEDURE</b>
<b>getTopN =&gt; SELECT</b>	<b>=&gt; FUNCTION</b>
<b>getExp =&gt; calculation</b>	<b>=&gt; FUNCTION</b>

### **Defining Package Specification:**

```
Create Or replace Package HR
IS
  procedure hire(p_empno emp.empno%type, p_ename
  emp.ename%type);
  procedure fire(p_empno emp.empno%type);
  procedure update_salary(p_empno emp.empno%type,
  p_amt float);
  function getTopN(n int) return sys_refcursor;
  function getExp(p_empno emp.empno%type)
  return number;
END ;
/
```

### **Defining Package Body:**

```
CREATE OR REPLACE package
body HR
IS
  procedure hire(p_empno
  emp.empno%type, p_ename
  emp.ename%type)           exec hr.hire(1,'A');
  IS
  BEGIN
    INSERT INTO
    emp(empno,ename)
    VALUES(p_empno, p_ename);
    COMMIT ;
    dbms_output.put_line('record
    inserted..!');
  END hire;

  procedure fire(p_empno
  emp.empno%type)
  IS
  BEGIN
```

```

DELETE FROM emp WHERE
empno=p_empno;
COMMIT ;
dbms_output.put_line('record
deleted..!');
END fire;

procedure
update_salary(p_empno
emp.empno%type,
p_amt float)
IS
BEGIN
    UPDATE emp SET
    sal=sal+p_amt WHERE
    empno=p_empno;
    COMMIT ;
    dbms_output.put_line('record
    updated..!');
END update_salary;

function getTopN(n int) return
sys_refcursor
IS
    c1 sys_refcursor;
BEGIN
    OPEN c1 FOR SELECT
    empno,ename,sal FROM emp
    ORDER BY sal DESC FETCH
    first n rows only;

    RETURN c1;
END getTopN;

function getExp(p_empno
emp.empno%type)
return number
IS
    v_hiredate DATE ;
BEGIN
    SELECT hiredate INTO
    v_hiredate FROM emp
    WHERE empno=p_empno;

    RETURN trunc((sysdate-
v_hiredate)/365);
END getExp;

```

**END;**

/

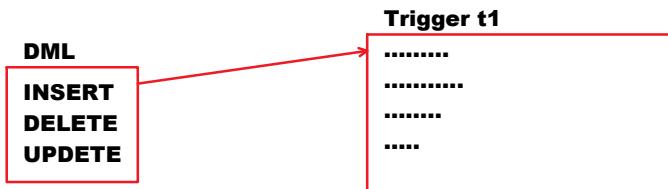
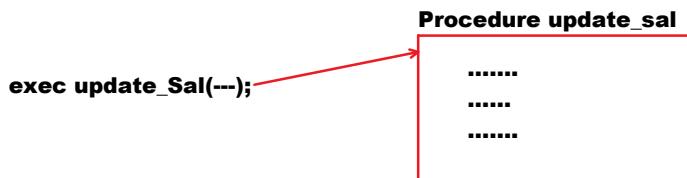
**Function Overloading:**

**Defining multiple functions with same name & different signature.**

**different signature means,**  
**difference in number of parameters**  
**difference in data type**  
**difference in order of parameters**

**procedure show(x int, y varchar2)**  
**procedure show(x int,y date,z varchar2)**  
**procedure show(x varchar2,y int)**

**package demo**  
**function addition => add 2 nums**  
**function addition => add 3 nums**

**Trigger:**

- Is a DB Object.
- Trigger is a named block of statements that gets executed automatically when we perform the DML operation.
- Procedure requires explicit call for execution. Whereas trigger gets executed automatically when we submit DML query.

**Trigger can be used for 3 purposes:**

- Trigger can be used to control the DML operations.
  - Ex:
    - don't allow the user to perform DML operations on sundays
    - Note:
    - to perform DML operations define PROCEDURE
    - to control DML operations define TRIGGER
- Trigger can be used for auditing the tables (or) databases.
  - Ex:
    - Which user, Which time, Which table, Which operation is performed can be recorded.
    - With this we can implement security for the data.
- We can implement our own rules on tables
  - Ex:
    - don't allow the user to decrease the salary

update\_Salary => performing DML operation => Procedure  
don't allow DMLs on Sunday => Controlling DMLs => Trigger

**3 purposes:**

- |                         |
|-------------------------|
| control the DMLs        |
| Audit the tables        |
| implement our own rules |

### Syntax to define a Trigger:

```
Create Or Replace Trigger <trigger_name>
  Before / After INSERT or UPDATE or DELETE
  On <table_name>
  [For Each Row]
  DECLARE
    <declaration_part>
  BEGIN
    <execution_part>
  END;
/
```

### Types of Triggers:

#### 3 Types:

- **Table Level Triggers [DML Triggers]**
- **Schema Level Triggers [DDL Triggers]**
- **Database Level Triggers**

#### • Table Level Triggers [DML Triggers]:

##### 2 Levels:

- **Statement Level trigger**
- **Row level trigger**

#### • Statement Level trigger:

**For every DML statement trigger gets executed only once**

**Update emp set sal=sal+2000; =>  
trigger gets executed only once**

#### Row level trigger:

**For every row affected by DML operation, trigger  
gets executed**

**Ex: if 14 rows updated =>  
trigger gets executed for 14 times**

#### Before Trigger:

- **First trigger gets executed**
- **DML operation will be performed**

#### After Trigger:

- **First DML operation will be performed**

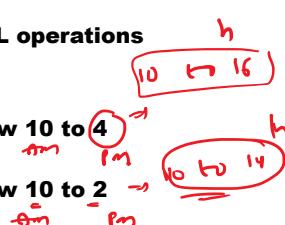
- Then Trigger gets executed

**Define a trigger to not to allow the user to perform DML operations on sundays on emp table:**

```
Create Or Replace Trigger t1
Before Insert Or Update Or Delete
On emp
BEGIN
  IF to_char(sysdate,'dy')='sun' THEN
    RAISE_APPLICATION_ERROR(-20050,'u cannot update on Sunday');
  END IF;
END;
/
```

**Define a trigger to control the DML operations as following:**

Mon to Fri => [2 to 6] <sup>wd</sup>  
 allow the user to perform DMLs b/w 10 to 4  
 Sat => [7]  
 allow the user to perform DMLs b/w 10 to 2  
 Sun =>  
 don't allow the DML operations



```
wd := to_Char(sysdate,'d');           1 => sun
                                         2 => mon
                                         3
                                         4
                                         5
                                         6 => fri
                                         7 => sat
h := to_Char(sysdate,'hh24');
```

**Create Or Replace Trigger t2**

**Before Insert Or Update Or Delete**

**On emp**

**DECLARE**

```
  wd int;
  h int;
```

**BEGIN**

```
  wd := to_char(sysdate,'d');
  h := to_char(sysdate,'hh24');
```

```
IF wd BETWEEN 2 AND 6 AND h NOT BETWEEN 10 AND 16 THEN
  RAISE_APPLICATION_ERROR(-20050,'From mon to fri updation allowed b/w 10AM to 4PM');
```

```
ELSIF wd=7 AND h NOT BETWEEN 10 and 14 THEN
  RAISE_APPLICATION_ERROR(-20070,'On sat, updation allowed b/w 10AM to 2PM');
```

```
ELSIF wd=1 THEN
  RAISE_APPLICATION_ERROR(-20060,'On sun updation not allowed..!');
```

```
    END IF;
END ;
/
```

**Define a trigger to not to allow the user to update empno:**

```
Create Or Replace Trigger t3
Before UPDATE of empno      --empno,ename
On emp
BEGIN
  RAISE_APPLICATION_ERROR(-20040,'u cannot
update empno');
END ;
/
```

**:new & :old bind variables:**

- These are predefined bind variables.
- Implicitly these are declared as "%rowtype".
- These can be used in row level triggers only.  
These cannot be used in Statement Level Triggers.

DML	:new	:old
INSERT	new row	NULL
DELETE	NULL	old row
UPDATE	new row	old row

**Statement Level Trigger:**

**For a DML statement, trigger gets executed once**

**Row Level Trigger:**

**For every row affected by DML stmt, trigger gets executed.**

**5 rows updated => 5 times trigger gets executed**

**Define a trigger to record deleted employees in "emp\_Resign" table:**

**Emp\_Resign**

empno	ename	hiredate	dor
1002	ALLEN	...	..

**dor => date of retirement**

```
Create Table emp_resign(
empno number(4),
ename varchar2(10),
hiredate date,
dor date
);
```

**Create Or Replace Trigger t4**

**After Delete**

**On emp**

**FOR EACH ROW**

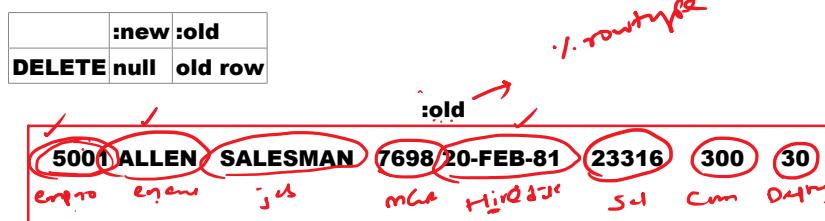
**Declare**

**Begin**

**Insert into emp\_resign values( :old.empno, :old.ename,**  
**:old.hiredate, :old.sal );**

**End;**

**/**



**Trigger can be used for 3 purposes:**

- **Controlling DML operations**
- **Auditing the tables**
- **implementing our own constraints [rules]**

**Define a trigger for auditing the emp table:**

**emp\_audit**

uname	op_type	op_time	new_empno	new_ename	new_sal	old_empno	old_ename	old_Sal
c##oracle6pm	INSERT	9-jan-22 6:57	7001	Ravi	5000			
c##oracle6pm	UPDATE	9-JAN22 7.00	7001	Ravi	8000	7001	Ravi	5000
c##oracle6pm	DELETE	9-jan-22 7:01				7001	Ravi	8000

**Create Table emp\_audit**

```
(  
  uname varchar2(20),  
  op_type varchar2(10),  
  op_time timestamp,  
  new_empno number(4),  
  new_ename varchar2(10),  
  new_sal number(7,2),  
  old_empno number(4),  
  old_ename varchar2(10),  
  old_sal number(7,2)  
);
```

**Create Or Replace Trigger t5**

**After INSERT or DELETE or UPDATE**

**On Emp**

**FOR EACH ROW**

**DECLARE**

    op varchar2(10);

**BEGIN**

```

IF inserting THEN -- inserting => predefined keyword => returns boolean value
  op := 'INSERT';
ELSIF deleting THEN
  op := 'DELETE';
ELSIF updating THEN
  op := 'UPDATE';
END IF:

Insert into emp_audit values(user, op, systimestamp,
:new.empno, :new.ename, :new.sal,
:old.empno, :old.ename, :old.sal);
END ;
/

```

**Define a trigger not to allow the user to decrease the salary:**

```

Create Or Replace Trigger t6
Before Update
On Emp
For Each Row
Declare

Begin
  if :new.sal < :old.sal Then
    raise_application_error(-20060,'u cannot decrease the
    sal..!');
  end if ;
End;
/

```

**Types of Triggers:**

**3 Types:**

- **Table Level Triggers [ DML Triggers]**
  - **Statement Level**
  - **Row Level**
- **Schema Level Triggers [DDL Triggers]**
- **Database Level Triggers**

- **Schema Level Triggers [DDL Triggers]**
- **Database Level Triggers**  
are defined by DBA

- **Schema Level Triggers [DDL Triggers]:**
- **If trigger is created on schema [user] then it is called "schema level trigger".**
- **to restrict one user, define schema level trigger**
- **It is defined by DBA**

**Syntax to define schema level trigger:**

```

Create Or Replace Trigger <trigger_name>
BEFORE / AFTER DROP or ALTER or TRUNCATE or CREATE
On <schema_name>.schema
DECLARE
    declaration_part
BEGIN
    execution_part
END ;
/

```

**Program to demonstrate schema level trigger:**

**Define a trigger to not to allow the "c##oracle11am"**  
**user to drop the database objects like tables or views:**

**Log in as DBA:**

**username: system**  
**password: nareshit**

```

Create Or Replace Trigger t8
Before Drop
On c##oracle11am.schema
BEGIN
    RAISE_APPLICATION_ERROR(-20050,'u cannot drop
    the db object..!');
END;
/

```

**System Variables:**

<b>ora_dict_obj_type</b>	<b>returns object type like table or view</b>
<b>ora_dict_obj_name</b>	<b>returns object name like emp or v1</b>
<b>ora_login_user</b>	<b>returns the user name like c##oracle6pm</b>
<b>ora_sysevent</b>	<b>returns the action like drop (or) alter</b>

**Ex on schema level trigger:**

**Define a trigger to not to allow c##oracle11am to drop the db**  
**objects:**

**Log in as DBA:**

**username: system**  
**password: nareshit**

```

Create Or Replace Trigger t9
Before Drop
On c##oracle11am.schema
Begin
    If ora_dict_obj_type='TABLE' then
        raise_Application_error(-20040,'u cannot drop table..!');
    end if ;
End ;
/

```

**Schema level trigger: to raise trigger on one user**

**DB level trigger: to raise trigger on many users**

**Ex on Db level trigger:**

defie the trigger not to allow c##oracle7am user, c##arun, c##oracle6pm to drop the db objects:

**Log in as DBA**

```
Create or replace trigger t10
Before Drop
On DATABASE
begin
  If ora_login_user in('C##ORACLE6PM','C##ORACLE11AM','C##ARUN') Then
    RAISE_APPLICATION_ERROR(-20045,'u cannot drop db object..!');
  end if;
end ;
/
```

**3 types of triggers:**

- **Table level trigger => ON emp**
  - statement level
  - row level => FOR EACH ROW
- **Schema level => On c##oracle11am.schema**
- **DB level => On Database**

**emp table:**

```
trigger tt1=> before => statement
trigger tt2=> after => statement
trigger tt3=> before => for each row
trigger tt4=> after => for each row
```

**Compound Trigger:**

- In Oracle 11g, Compound Trigger is introduced
- It is a set of triggers. We have no need to define multiple triggers on one table. Define multiple triggers in one trigger i.e. "Compound Trigger".

**Syntax to define Compound Trigger:**

```
CREATE OR REPLACE TRIGGER <trigger-name>
FOR <trigger-action> ON <table-name>
  COMPOUND TRIGGER
  -- Global declaration.
  g_global_variable VARCHAR2(10);
  BEFORE STATEMENT IS
    BEGIN
      NULL; -- Do something here.
    END BEFORE STATEMENT;
  BEFORE EACH ROW IS
    BEGIN
      NULL; -- Do something here.
    END BEFORE EACH ROW;
```

```

END BEFORE STATEMENT;
BEFORE EACH ROW IS
BEGIN
    NULL; -- Do something here.
END BEFORE EACH ROW;
AFTER EACH ROW IS
BEGIN
    NULL; -- Do something here.
END AFTER EACH ROW;
AFTER STATEMENT IS
BEGIN
    NULL; -- Do something here.
END AFTER STATEMENT;
END <trigger-name>;
/

```

**Log in as DBA:**

**DDL\_AUDIT**

<b>uname</b>	<b>op_type</b>	<b>op_time</b>	<b>obj_type</b>	<b>obj_name</b>
c##ora6pm	<b>DROP</b>	<b>10-jan-227:10 PM</b>	<b>TABLE</b>	<b>DEPT</b>

<b>user</b>	<b>ora_sysevent</b>	<b>systimestamp</b>	<b>ora_dict_obj_type</b>	<b>ora_dict_obj_name</b>
-------------	---------------------	---------------------	--------------------------	--------------------------

```

Create Or Replace Trigger t12
After Create Or Alter Or Drop Or Truncate Or Rename
On Database
BEGIN
    Insert into ddl_audit
    values(user,systimestamp,ora_sysevent,ora_dict_obj_name,ora_dict_obj_type);
END ;
/

```

## **Dynamic SQL**

Monday, January 10, 2022 7:05 PM

### **Dynamic SQL:**

- **The SQL query which is built at run time is called "Dynamic SQL".**
- **DML, TCL, DRL commands can be used directly in PL/SQL program. But DDL, DCL commands cannot be used directly in PL/SQL program. To use DDL & DCL commands in PL/SQL program we use "Dynamic SQL".**
- **"Execute Immediate" command is used to execute the dynamic SQL query.**
- **We have to pass SQL query as string to Execute Immediate command**

**Drop Table emp; => static query**

**EXECUTE IMMEDIATE 'Drop Table ' || n ; => Dynamic Query**

### **Define a procedure to drop the tables:**

```
Create Or Replace Procedure drop_table(n varchar2)
is
begin
  EXECUTE IMMEDIATE 'drop table ' || n;
  dbms_output.put_line('table dropped..!');
end ;
/
exec drop_table('dept');
```

### **define procedure to drop any kind of db object:**

**Drop Table t1;**

**Drop view v1;**

**Drop index i1;**

**Drop trigger t1;**

```
Create Or Replace Procedure drop_object(t varchar2,n varchar2)
IS
BEGIN
EXECUTE IMMEDIATE 'Drop ' || t || ' ' || n;
dbms_output.put_line(n || ' ' || t || ' dropped..!');
END ;
/
```

```
exec drop_object('view','v1');
```

```
exec drop_object('table','t1');
```

```
exec drop_object('index','i1');
```

**Define a procedure to drop all views:**

```
Create Or Replace Procedure drop_all_views
IS
Cursor c1 IS Select view_name from user_views;
BEGIN
for r in C1
loop
    EXECUTE IMMEDIATE 'Drop view ' || r.view_name ;
end loop;
dbms_output.put_line('all views dropped..!');
END;
/
```

**Display all table names & no of records:**

**emp 14**  
**dept 4**  
**salgrade 5**

**Create Or Replace Procedure all\_tables\_no\_of\_records**  
**IS**  
**Cursor c1 IS Select table\_name fro User\_Tables;**  
**n int;**  
**BEGIN**  
**for r in c1**  
**Loop**  
**EXECUTE IMMEDIATE 'Select count(\*) from ' || r.table\_name**  
**INTO n;**  
**dbms\_output.put\_line(r.table\_name || '                ' || n);**  
**End Loop;**  
  
**END ;**  
**/**

## Working with LOBS

Tuesday, January 11, 2022 6:16 PM

### LOB Data Types:

- BFILE
- BLOB

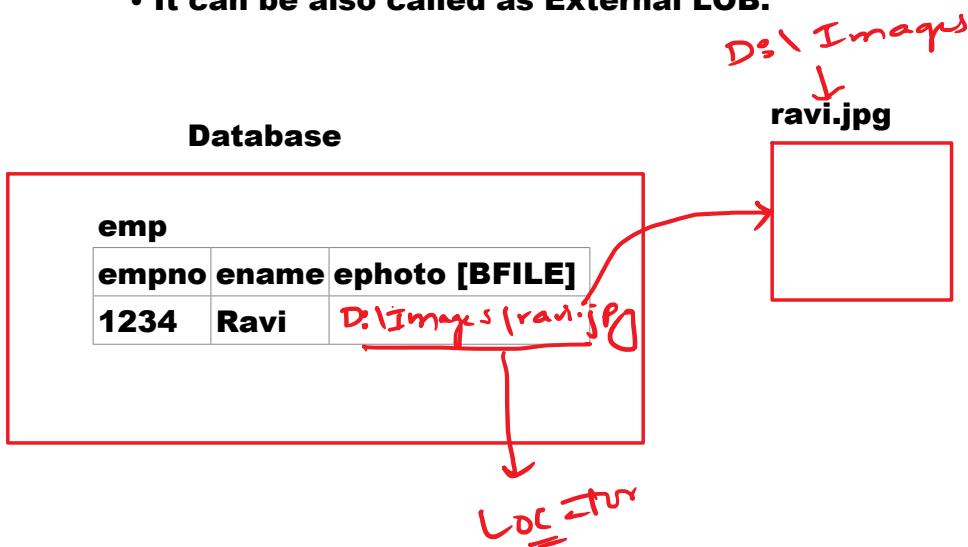
binary file

1001101010
11001

These 2 data types are used to maintain multimedia objects like images, audios, videos ..etc.

### BFILE:

- Binary Large Object File.
- It can be also called as External LOB.



### BFILE:

- Binary Large Object File.
- It can be also called as External LOB.
- it is not secured one.
- to maintain the locator [path of multimedia object]
- we use "bfilename()".

### Syntax:

bfilename(directory\_object,file\_name);

### Ex:

bfilename('D1','ravi.jpg');

D:\Images

D1 is pointer directory in OS

### Creating Directory Object:

- Directory object is an object in the Database that points to a directory in OS.
- DBA creates Directory Object.

**Syntax:**

```
Create Directory <name> As <folder_path>;
```

**Ex:**

```
Create Directory d1 As 'C:\photos';
```

```
Grant read,write on directory d1 to c##oracle6pm;
```

**Create a Table as following:****Customer1**

cid	cname	cphoto [BFILE]
1234	Ravi	bfilename('D1','taj.jpg')

**Create Table Customer1**

```
(  
    cid number(4),  
    cname varchar2(10),  
    cphoto BFILE  
)
```

```
Insert into Customer1 values(1234,'Ravi',  
bfilename('D1','taj.jpg'));
```

**BLOB:**

- **Binary Large Object**
- **It is used to maintain multimedia objects like images, audios, videos ..etc.**
- **Internal Large Object.**

**customer2**

cid	cname	cphoto [BLOB]
1234	Ravi	A1234BCD128765FE

**Create Table Customer2**

```
(  
    cid number(4),  
    cname varchar2(10),  
    cphoto BLOB  
)
```

Function  
=

**Insert into Customer2 values(1234,'Ravi',empty\_blob());**

↑  
Inserting  
null in Blob  
type

**Define a procedure to update customer photo in table:**

```
dbms_lob.open()  
dbms_lob.getlength()  
dbms_lob.LoadFromFile()  
dbms_lob.close()
```

"dbms\_lob" package

Create Or Replace Procedure  
update\_photo(p\_cid customer2.cid%type,n varchar2)  
IS

s BFILE;  
t BLOB;  
l number;  
BEGIN  
s := bfilename('D1',n); --s=> filepath

Select cphoto into t from customer2  
where cid=p\_cid for update; --locks record

dbms\_lob.open(s,dbms\_lob.readonly);

l:=dbms\_lob.getlength(s);

dbms\_lob.LoadFromFile(t,s,l);

Update customer2 set cphoto=t  
where cid=p\_cid;

dbms\_output.put\_line('photo updated..!');

1011040  
1100011001

↑ opens file in read mode  
↑ finds size of file.  
↑ l no. of bytes of s will  
be copied into t.

```
dbms_lob.close(s);  
END;  
/
```

## Collections

Tuesday, January 11, 2022 7:09 PM

### Collection:

- is a set of elements of same type
- All elements share same name.
- To identify elements uniquely, we use indexes.
- Index can be any data type.
- Indexing can start from anywhere.

**Associative Array [or] Index By Table:**  
It maintains a table of elements & indexes.

Index	Element
0	10
1	50
2	40

Index	Element
A	Ramu
B	Kiran
C	Raju

We need to follow 2 steps:

- Define collection type
  - Declare variable
- Define collection type

### Syntax:

Type <name> IS Table OF <element\_data\_type>  
INDEX BY <index\_data\_type>;

### Ex:

Type num\_array IS TABLE OF number(4)  
INDEX BY binary\_integer;

## **Declaring varibale:**

### **Syntax:**

**variable datatype;**

**x num\_array;**

## **Program to demonstrate collection:**

**DECLARE**

```
type num_array is table of number(4)
index by binary_integer;
x num_array;
```

**BEGIN**

```
x(0) := 10;
x(1) := 50;
x(2) := 30;
```

**for I in 0..2**

**loop**

```
dbms_output.put_line(x(i));
```

**end loop;**

**END;**

**/**

## **Define a collection to main dept names:**

<b>index</b>	<b>element</b>
<b>0</b>	<b>ACCOUNTING</b>
<b>1</b>	<b>RESEARCH</b>
<b>2</b>	
<b>3</b>	<b>OPERATIONS</b>

**Declare**

```
Type dept_array IS TABLE OF varchar2(20)
index by binary_integer;
d dept_array;
```

**Begin**

**End ;**

**/**

Select dname into d(0) from dept where deptno=10;  
Select dname into d(1) from dept where deptno=20;  
Select dname into d(2) from dept where deptno=30;  
Select dname into d(3) from dept where deptno=40;

*Instead of this code  
write 2*



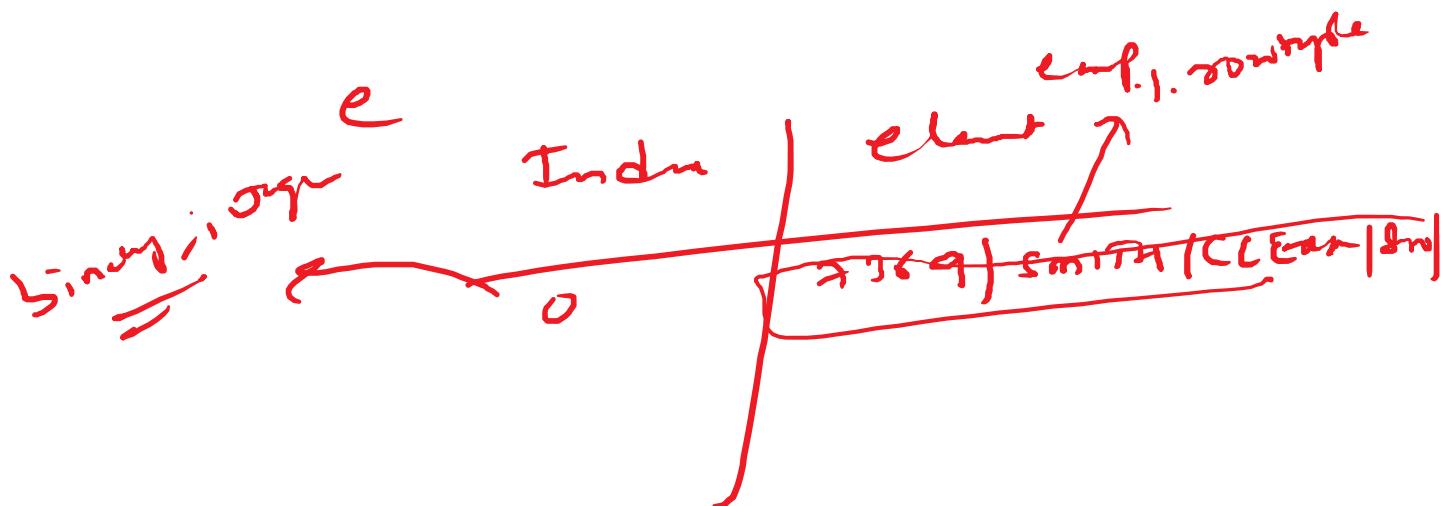
**Select dname BULK collect into d from dept;**

#### **BULK COLLECT:**

- collects entire data and copies into collection.
- It reduces no of requests to server
- Performance will be improved.

**Cursor**  
**can hold a set of records**

**Collection**  
**can hold a set of records**



<b>Cursor</b>
<b>processes row by row</b>
<b>cannot go backward.</b>
<b>can move forward only</b>
<b>random accessing is not possible</b>
<b>poor performance</b>

<b>Collection</b>
<b>collects entire data and copies into collection</b>
<b>can move backward or forward.</b>
<b>random accessing is possible</b>
<b>better performance</b>