

ORACLE 19C

Data:

- Data is a raw collection of facts about people, places, and things.
- It is unprocessed one. It means, it is not meaningful form.
- Data can be divided into 2 types. They are:
 - i. Structured Data
 - ii. Unstructured Data

Structured Data:

It is made up of with letters, digits, and special symbols.

Example:

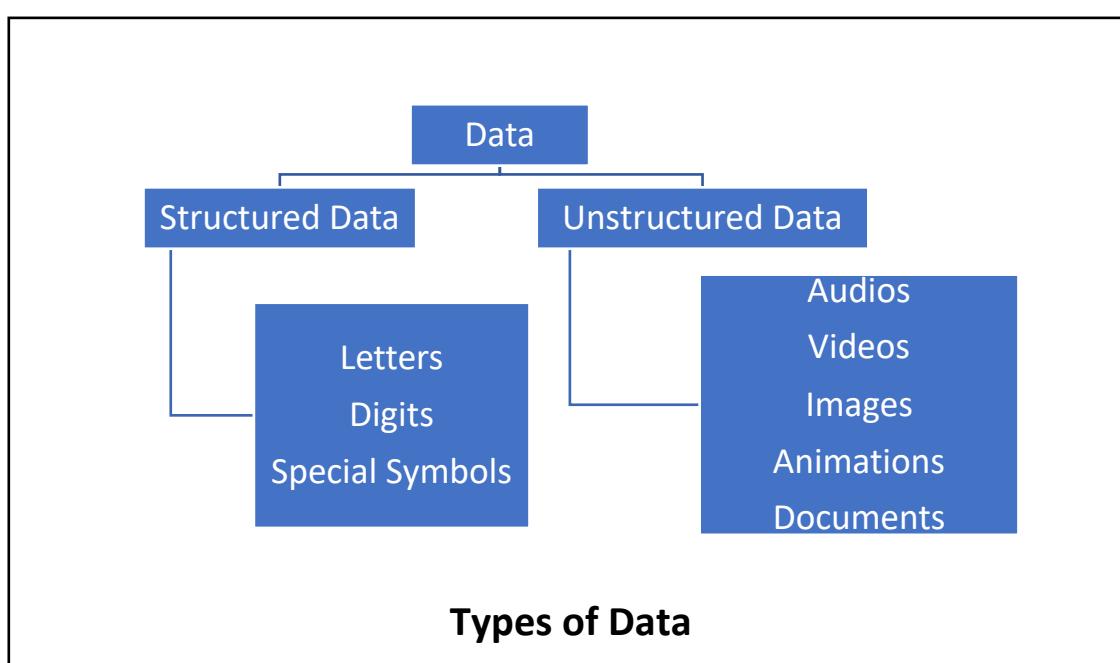
25 1-2-123/A Ramu

Unstructured Data:

Multimedia objects come under Unstructured data.

Example:

Audios
Videos
Images
Animations
Documents



Information:

If the data is arranged in meaningful form, then it called “Information”. It is processed one. It is in meaningful form.

Example:**STUDENT**

STUDENT_ID	STUDENT_NAME	MARKS
25	Ramu	500

Note:

If the data is processed by the process of data processing, it becomes Information.

**Database:**

- Database is a collection of interrelated data.
- It contains records and fields.
- Rows are called records and columns are fields.
- Field holds individual values.
- Record is a collection of field values.
- A row can be also called as record (or) tuple (or) entity instance.
- A column can be also called as field (or) attribute (or) property.
- Database can be divided into two types. They are:
 - i. OLTP
 - ii. OLAP

OLTP:

- OLTP stands for “OnLine Transaction Processing”.
- This database is used for day-to-day operations.
- CRUD operations are performed in day-to-day operations. CRUD operations are:

C = Create

R = Read

U = Update

D = Delete

Example:

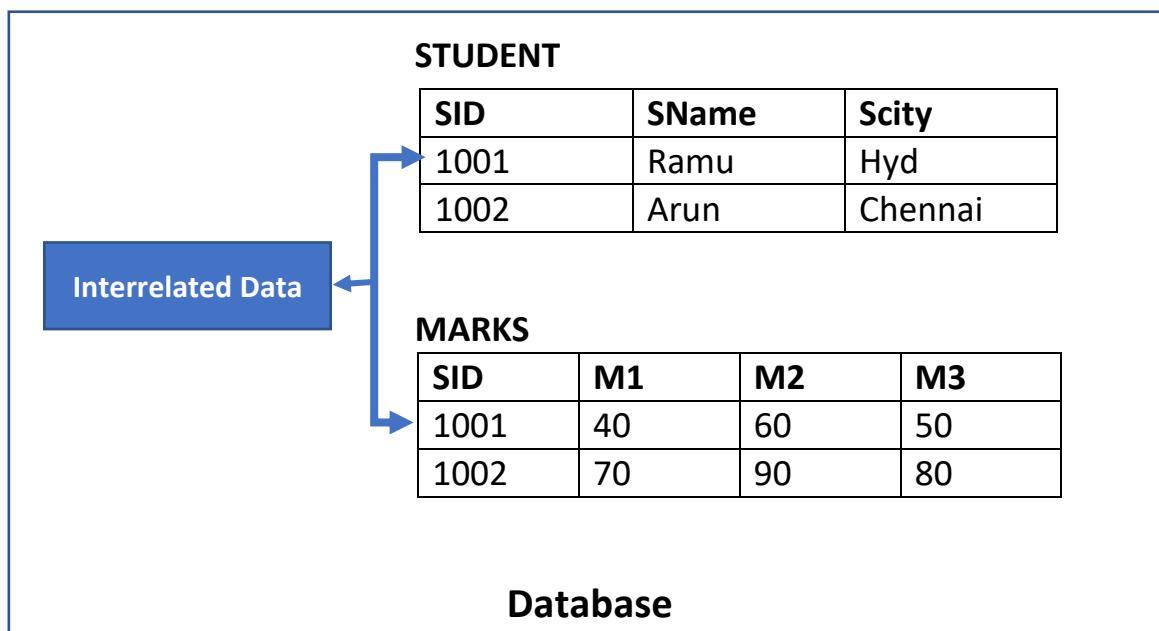
Maintaining day-to-day operations of Bank Customers like withdrawals, deposits, checking balance and fund transfers

OLAP:

- OLAP stands for “OnLine Analytical Processing”.
- This database is used for Data Analysis.
- In this, we write complex queries for data analysis.
- It can be also called “Dataware Housing [DWH]” or “Decision Support System [DSS]”.

Example:

Retrieving sales of past 5 years

**DBMS:**

- DBMS stands for DataBase Management System/Software.
- It is a software that is used to create and maintain the database.
- This software allows us to store, manipulate and retrieve the data of database. “Retrieve” means opening existing data. Manipulate means, Inserting, Deleting and Updating [modifying] the data.

RDBMS:

- RDBMS stands for Relational DataBase Management System / Software.
- In this, we maintain database in the form of tables.

Example: Oracle, SQL Server, MY SQL, PostGre SQL, DB2

Table:

- Table is a collection of rows and columns.
- It can be also called as “Relation” (or) “Entity”.
- Rows are called Records (or) Tuples (or) Entity Instances.
- Columns are called Fields (or) Attributes (Or) Fields.
- Field holds individual values.
- Record is a collection of field values.

**Metadata:**

- Metadata is the data about the data.
- It describes about the data.

Example:

Field Name

Table Name

Data Type

Field Size

File Management System [FMS]:

- It is a software which is used to create and maintain the data in flat files (text files).
- It was introduced in 1960s.
- It was the first method to store the business data in computer.
- In this, data and metadata will be stored in many locations. If we want to make any changes to data (or) metadata, we need to change in many places. If we changed in one place forgot to change in another places Data Inconsistency problem will occur.

Disadvantages of FMS:

- Data Redundancy
- Data Dependence
- No Security
- Limited Data Sharing
- Excessive Program Maintenance
- Lengthy Development Times

DataBase Management System [DBMS]:

- DBMS is a software that is create and maintain the database.
- It allows us to store, manipulate & retrieve the data of Database.
- In this, data and metadata will be stored in one location. If we want to make any changes, we need to change in one place only.

Advantages of DBMS:

- Minimal Data Redundancy
- Data Independence
- Security
- Improved Data Sharing
- Improved Data Consistency
- Improved Data Integrity
- Improved Data Accessibility and Responsiveness
- Increased Productivity

ORACLE:

- Oracle is a Relational Database management software.
- It is used to create & maintain the database.
- In this we maintain database in the form of tables.
- It allows us to store, manipulate and retrieve the data of database.
- Second version of “Oracle” introduced in 1979.
- Latest version is Oracle 19C. [c =Cloud].
- It is a product of “ORACLE” company. This company is established in 1977 by “Larry Ellison” with the name “Software Development Laboratories”. Later it renamed as “Relation Software Inc.” [Incorporation] in 1979. Later it renamed as “ORACLE CORPORATION” in 1983.

Tools of ORACLE:

Oracle provides following tools:

- SQL
- PL/SQL

SQL:

- SQL stands for “Structured Query Language”.
- It is a query language used to write the queries.
- Query is a request that is sent to Database Server.
- Query is used to communicate with the database.
- It is a Non-Procedural Language. We will not write any set of statements.
- It is a Fourth Generation Language [4GL]. In 4GLs, we much concentrate on what to do rather than how to do.
- It is a Unified Language. It is common for many relational databases such as SQL Server, My SQL, Postgre SQL ..etc.
- It provides operators to perform operations like +, -, >, Between and, In Like ..etc.
- It provides built-in functions to perform specific actions.
- It does not provide control structures. Because we cannot write set of statements in SQL. We have no need to control the flow of execution.
- It provides 5 sub languages. Every sub language provides commands to work with the database. They are:
 - i. DDL
 - ii. DML
 - iii. DQL /DRL
 - iv. DCL / ACL
 - v. TCL

List of SQL Commands:

SQL Sub Languages	Commands	Purpose
DDL [Data Definition Language]	Create Alter Drop Truncate Rename Flashback Purge	<ul style="list-style-type: none"> • These commands deal with metadata. • To create Database objects like Tables, Views, Indexes, Synonyms, Sequences we use these commands. • To change the structure of table also we use these commands.
DML [Data Manipulation Language]	Insert Delete Update Insert All Merge	<ul style="list-style-type: none"> • These commands deal with the data. • To insert, delete and update [modify] the records we use these commands.
DQL / DRL [Data Query Language / Data Retrieval Language]	Select	<ul style="list-style-type: none"> • It deals with Data Retrievals. • We retrieve the data as per requirement using clauses, operators, sub queries, joins and built-in functions ...etc.
DCL / ACL [Data Control Language / Accessing Control Language]	Grant Revoke	<ul style="list-style-type: none"> • It deals with data accessibility. • These are used to give permission to other users or cancel the permission from other users.
TCL [Transaction Language]	Commit Rollback Savepoint	<ul style="list-style-type: none"> • These commands are used to deal with the transactions. • These are used to save or cancel the transactions.

Data Types in SQL:

- Data Type is used to specify which type of data a column can hold.
- It tells how much memory should be allocated.

SQL provides following Data Types:

Category	Data Types
Number Related Data Types	Number(p) Number(p,s)
Character Related Data Types	Char(n) Varchar2(n) Long CLOB nChar(n) nVarchar2(n) nCLOB [n = National]
Date & Time Related Data Types	Date Timestamp
Binary-Related Data Types	BFILE BLOB

Number Related Data Types:

Number(p):

- It is used to hold integers.
- “p” stands for “Precision”. Precision means, Total Number of Digits”.
- Valid range of p is: 1 to 38.
- Default size is: 38.
- If we don’t specify the precision default size will be taken.
- Default Valid Range of values: -99999.....99 38 digits to 99999.....99 38 digits.

Example:

- Marks Number(3) => Range: -999 to 999
 Empno Number(4) => Range : -9999 to 9999
 Customer_ID Number(6) => Range: -999999 to 999999

Number(p,s):

- It is used to hold floating point type values.
- “p” stands for Precision. “s” stands for Scale.
- Precision means, Total Number of Decimal Places
- Scale means, Number of Decimal Places.
- Valid Range of Precision is: 1 to 38.
- Valid range of Scale is -84 to 127.

Example:

If maximum average is 100.00 then declare as:

Avrg_Marks Number(5,2) => Range: -999.99 to 999.99

If maximum salary is 100000 .00[1 Lakh] then declare as:

Salary Number(8,2) => -999999.99 to 999999.99

Character Related Data Types:**Char(n):**

- It is used to hold a set of characters [strings].
- “n” means maximum number of characters.
- It is a Fixed Length Character Data Type.
- It is a single byte Character Data Type.
- It is ASCII Code Character Data type.
- Maximum size is: 2000 Bytes
- Default size is: 1
- To hold fixed length characters like state code, country code, PAN Card Number, and Gender ...etc. we can use this data type.

Example:**State_Code Char(2)****STATE CODE**

AP

TS

Country_Code Char(3)**COUNTRY CODE**

IND

AUS

Gender Char(1)**GENDER**

M

F

Varchar(n):

- It is used to hold a set of characters.
- “n” means maximum number of characters.
- It is a Variable Length Character Data Type.
- It is a single byte Character Data Type.
- It is ASCII Code Character Data type.
- Maximum size is: 4000 Bytes
- There is no default size for it. We must specify the size.
- To hold variable length characters like Emp_Name, Student_Name, Product_Name ...etc. we can use this data type.

Example:

Emp_Name Varchar2(20) => it can hold maximum of 20 characters
Job Varchar2(10) => it can hold maximum of 10 characters

Long:

- It is used to hold a set of characters.
- It is a Variable Length Character Data Type.
- It is a single byte Character Data Type.
- It is ASCII Code Character Data type.
- Maximum size is: 2 GB

Example:

Experience_summary LONG
Customer_Feedback LONG

CLOB:

- It is used to hold a set of characters.
- CLOB stands for Character Large Object.
- It is a Variable Length Character Data Type.
- It is a single byte Character Data Type.
- It is ASCII Code Character Data type.
- Maximum size is: 4 GB

Example:

Experience_Summary CLOB
Product_Features CLOB
Remarks CLOB
Customer_Feedback CLOB

nChar(n):

- It is used to hold a set of characters.
- In “nChar”, n means National.
- It is a Fixed Length Character Data Type.
- It is a multi-byte Character Data Type.
- It is UNI Code Character Data type.
- Maximum size is: 2000 Bytes
- Default size is: 1
- To hold English and other language characters, we use it.

nVarchar(n):

- It is used to hold a set of characters.
- In “nVarchar”, n means National.
- It is a Variable Length Character Data Type.
- It is a multi-byte Character Data Type.
- It is UNI Code Character Data type.
- Maximum size is: 4000 Bytes
- No default size for it. We must specify the size.
- To hold English and other language characters, we use it.

nCLOB:

- It is used to hold a set of characters.
- In “nCLOB”, n means National.
- It is a Variable Length Character Data Type.
- It is a multi-byte Character Data Type.
- It is UNI Code Character Data type.
- Maximum size is: 4 GB.
- To hold English and other language characters, we use it.

Date & Time Related Data Types:**Date:**

- It is used to hold date values.
- It can hold time values also.
- It can hold day, month, year, hours, minutes and seconds
- It cannot hold fractional seconds [Milli Seconds].
- To insert date value, we use to_Date() Function. Even if we don't use this function implicitly given value will be converted to Date type.

- Default time value is: 12:00:00 AM
- To insert time value, we use `to_Date()` Function.
- Default Date Format is: DD-MON-RR.
- RR means year last 2 digits. If RR value is between 0 to 49, year value will be prefixed with 20. If RR value is between 50 to 99, year value will be prefixed with 19.
- 7 Bytes Fixed Memory will be allocated for it.
- Valid Range is : 1st January, 4712 BC to 31st December, 9999 AD

Example:

Date_Of_Birth Date
Date_OfJoining Date

Timestamp:

- It is used to hold time values.
- It can hold date values also.
- It can hold day, month, year, hours, minutes, seconds and fractional seconds [Milli Seconds].
- 11 Bytes Fixed Memory will be allocated for it.

Binary Related Data Types:**BFILE:**

- In BFILE, B stands for “Binary”.
- It is used to hold multimedia objects like images, audios, videos and documents.
- It can hold path of multimedia object.
- It acts like a pointer to multimedia object.
- It maintains path of the object & object is stored out of the database. That's why it can be also called as “External Large Object”.
- It is not secured one. Because object is stored out of the database.
- Maximum size is: 4 GB.

Example:

Student_Photo BFILE

BLOB:

- BLOB stands for “Binary Large Object”.
- It is used to hold multimedia objects like images, audios, videos and documents.
- It can hold binary data of the multimedia object and displays to us in hexadecimal format.

- It maintains path of the object & object is stored out of the database. That's why it can be also called as "External Large Object".
- It is secured one. Because object stored with in the database.
- Maximum size is: 4 GB.

Example:

Student_Photo BLOB

List of SQL Data Types & Their Maximum Size:

Data Type	Maximum Size
Number	21 Bytes
Char	2000 Bytes
Varchar2	4000 Bytes
Long	2 GB
CLOB	4 GB
nChar	2000 Bytes
nVarchar2	4000 Bytes
nCLOB	4 GB
Date	7 Bytes
Timestamp	11 Bytes
BFILE	4 GB
BLOB	4 GB

Constraints in SQL:

- Constraint is a rule which is applied on a column.
- Constraint is used to maintain accurate and quality data.
- Constraint restricts the user from entering invalid data.
- The goal of constraint is maintaining accurate and quality data. This feature is called “Data Integrity”.
- SQL provides Data Integrity.

SQL provides following constraints to maintain the Data Integrity:

1. Primary Key
2. Unique
3. Not Null
4. Check
5. Default
6. References [Foreign Key]

1. Primary Key:

- It should not accept duplicate values.
- It should not accept null values.
- If our requirement is not to accept duplicate value and null value in a column then use it.

Example:

SID [Primary Key]	SName	M1
1001	Ravi	45
1002	Arun	70
1003	Ravi	70
1001	Sai	50
	Ramu	90

This record will not be accepted. It creates duplicate value.

This record will not be accepted. It creates null value.

2. Unique:

- It should not accept duplicate values.
- It accepts null values.
- If our requirement is accepting null values but not accepting duplicate values, then use it.

Example:

CUSTOMER

CID	SName	Mobile_Number[Unique]
1001	Ravi	90123 12312
1002	Arun	90123 12313
1003	Ravi	90123 12314
1004	Sai	90123 12312
1005	Ramu	

This record will not be accepted. It creates duplicate value.

This record will be accepted. It creates null value. "Unique" accepts null value.

3. Not Null:

- It should not accept null values.
- It can accept duplicate values.
- When we want to accept duplicate value and demand for entering a value in field then use it.

Example:

SID	SName [Not Null]	M1
1001	Ravi	45
1002		70
1003	Ravi	70

This record will not be accepted. It creates duplicate value.

This record will not be accepted. It creates null value.

4. Check:

It is used to write our own conditions [rules] on a column.

Example:

If M1 value Must be between 0 to 100 only then write:

M1 Number(3) Check(M1>=0 and M1<=100)

If Gender value must be M or F only then write:

Gender Char(1) Check(Gender='M' or 'F')

If salary must be between 5000 to 100000 then write:

Salary Number(8,2) Check(Salary>=5000 and Salary<=100000)

5. Default:

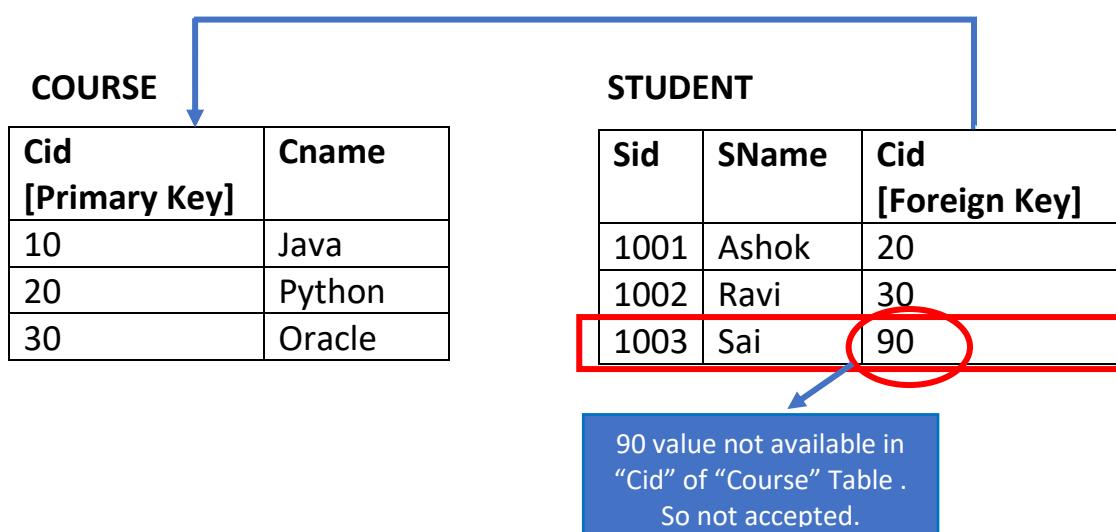
- It is used to apply default value to a column.
- We have no need to pass this value when we insert the record. It will be taken automatically when we don't pass the value.
- If we pass the value, it takes the value. If we don't pass the value, it takes default value.

Example:

Empno	EName	Company_Name [default 'Naresh IT']	Company_City [default 'Hyd']
7001	Raju	Naresh IT	Hyd
7002	Ashok	Naresh IT	Hyd
7003	Kiran	Naresh IT	Hyd

6. References [Foreign Key]:

Foreign Key refers to primary key values of another table.



DDL Commands

- DDL Stands for Data Definition Language
- It deals with meta data [Data Definitions] such as table name, column name, data type, field size, constraint name, constraint type ...etc.
- We have no need to use “commit” command after using a “DDL Command”. All DDL commands are auto committed.
- DDL Commands provided by Oracle-SQL are:
 - Create
 - Alter
 - Drop
 - Truncate
 - Rename
 - Flashback
 - Purge

“Create” Command:

- It is a DDL Command.
- It is used to create the database objects like tables, views, and indexes ...etc.

Syntax to create a Table:

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

Example:

```
Create Table emp(Empno number(4) constraint c1 Primary Key,  
    Ename varchar2(20) constraint c2 Not Null);
```

Examples on creating tables:**Example-1:**

Create a table with following structure, insert records into table & retrieve the records from table:

Student

Rno	Name
-----	------

Creating Table:

```
SQL> Create Table Student(  
      Rno Number(2),  
      Name Varchar2(20) );
```

Output:

Table Created

To See Structure of Table:

```
SQL> Desc Student;
```

Output:

Name	Null?	Type
RNO		NUMBER(2)
NAME		VARCHAR2(20)

Inserting a record into Table:

```
SQL> insert into student values(10,'Raju');
```

Output:

1 row created.

Inserting multiple records using parameters:

```
SQL> insert into student values(&rno,'&name');
```

Enter value for rno: 20

Enter value for name: Vijay

Output:

old 1: insert into student values(&rno,'&name')

new 1: insert into student values(20,'Vijay')

1 row created.

Note: / (or) Run => is used to repeat recent command

Repeating Recent (Above) Command:

SQL> /

Enter value for rno: 30

Enter value for name: Sai

Output:

old 1: insert into student values(&rno,'&name')

new 1: insert into student values(30,'Sai')

1 row created.

Retrieving all columns and all all records from Table:

SQL> Select * from student;

Output:

RNO NAME

10 Raju

20 Vijay

30 Sai

Note: * means, All Columns

Example-2:

Create a table with following structure, Insert the records and retrieve the records:

Emp

Empno	Ename	Job	Sal	DOJ
-------	-------	-----	-----	-----

Creating Table:

```
SQL> Create Table Emp(  
    empno number(4),  
    ename varchar2(20),  
    job varchar2(15),  
    sal number(7,2),  
    DOJ date);
```

Output:

Table created.

Inserting a Record:

```
SQL> Insert into Emp values(1234,'ravi','clerk',5000,'12-Dec-2017');
```

Output:

1 row created.

Inserting Multiple Records using Parameters:

```
SQL> Insert into Emp values(&empno,'&en','&j','&sal','&doj');
```

```
Enter value for empno: 1235  
Enter value for en: arun  
Enter value for j: salesman  
Enter value for sal: 7000  
Enter value for doj: 23-aug-2018
```

Output:

```
old  1: Insert into Emp values(&empno,'&en','&j','&sal','&doj')  
new  1: Insert into Emp values(1235,'arun','salesman','7000','23-aug-  
2018')
```

1 row created.

Repeating Recent Command:

```
SQL> /
Enter value for empno: 1236
Enter value for en: sravan
Enter value for j: accountant
Enter value for sal: 6000
Enter value for doj: 25-Nov-2017
```

Output:

```
old  1: Insert into Emp values(&empno,'&en','&j','&sal','&doj')
new  1: Insert into Emp values(1236,'sravan','accountant','6000','25-
Nov-2017')
```

1 row created.

Retrieving All Columns and All Records:

```
SQL> Select * from Emp;
```

Output:

EMPNO	ENAME	JOB	SAL	DOJ
1234	ravi	clerk	5000	12-DEC-17
1235	arun	salesman	7000	23-AUG-18
1236	sravan	accountant	6000	25-NOV-17

Examples on Creating Tables using Constraint Types:**Example-1:**

Create a Table using following structure:

Student1

Rno	Name	M1
-----	------	----

Set Primary Key for Rno

Set Not Null for Name

Set Check for M1 [Marks must be between 0 to 100]

Creating Table using Constraints:

```
SQL> Create Table Student1(  
      Rno Number(2) Primary Key,  
      Name Varchar2(20) Not Null,  
      M1 Number(3) Check(M1>=0 and M1<=100));
```

Output:

Table created.

Inserting a Record:

```
SQL> Insert into Student1 values(12,'Ravi',45);
```

Output:

1 row created.

Inserting null value in Rno field to check Primary Key does not accept null value:

```
SQL> Insert into Student1 values(null,'Kiran',70);
```

Output:

ERROR at line 1:

ORA-01400: cannot insert NULL into ("SYSTEM"."STUDENT1"."RNO")

Inserting duplicate value in Rno field to check Primary Key does not accept duplicate value:

```
SQL> Insert into Student1 values(12,'Arun',65);
```

Output:

ERROR at line 1:

ORA-00001: unique constraint (SYSTEM.SYS_C007590) violated

Inserting duplicate value in name field to check 'not null' accepts duplicate value:

SQL> Insert into Student1 values(13,'Ravi',56);

Output:

1 row created.

Inserting null value in name field to check 'not null' does not accept null value:

SQL> Insert into Student1 values(14,null,80);

Output:

ERROR at line 1:

ORA-01400: cannot insert NULL into ("SYSTEM"."STUDENT1"."NAME")

Inserting Value >100 in M1 field to test check constraint:

SQL> Insert into Student1 values(15,'ramesh',123);

Output:

ERROR at line 1:

ORA-02290: check constraint (SYSTEM.SYS_C007589) violated

Inserting Value <0 in M1 field to test check constraint:

SQL> Insert into Student1 values(15,'ramesh',-45);

Output:

ERROR at line 1:

ORA-02290: check constraint (SYSTEM.SYS_C007589) violated

Example-2:

Create a Table with following structure:

Student2

StudentId	Name	Fee	College_City
-----------	------	-----	--------------

Set Unique for StudentId

Set Not Null for Name

Set default fee value 15000 for Fee

Set default College_City Value 'Hyd' for College_City

Creating Table using Constraints:

```
SQL> Create Table Student2(
    StudentId number(4) Unique,
    Name varchar2(15) Not Null,
    Fee Number(7,2) Default 15000,
    College_City Varchar2(15) Default 'Hyd');
```

Output:

Table created.

Inserting a record:

```
SQL> Insert into Student2(StudentId,Name) values(1234,'Raju');
```

Output:

1 row created.

Inserting duplicate value in StudentId field to check 'unique' does not accept null value:

```
SQL> Insert into Student2(StudentId,Name) values(1234,'Amar');
```

Output:

ERROR at line 1:

ORA-00001: unique constraint (SYSTEM.SYS_C007592) violated

Inserting null value in StudentId field to check 'unique' accepts null value:

```
SQL> Insert into Student2(StudentId,Name) values(null,'Amar');
```

Output:

1 row created.

Retrieve the records from Table and observe default values inserted in Fee and College_City fields:

SQL> Select * from Student2;

Output:

STUDENTID	NAME	FEE	COLLEGE_CITY
1234	Raju	15000	Hyd
	Amar	15000	Hyd

Example-3:

Create following tables:

Dept

Deptno	Dname

Set Deptno as Primary Key

Emp

Empno	Ename	Job	Sal	DeptNo

Set Empno as Primary Key

Set Dept No as Foreign Key [References].

Creating “Dept” Table:

SQL> Create Table Dept (Deptno Number(2) Primary Key, Dname Varchar2(15));

Output:

Table created.

Creating “Emp” Table:

SQL> Create Table Emp(Empno Number(4) Primary Key, Name Varchar2(20), Job Varchar2(15), Sal Number(7,2), DeptNo Number(2) References Dept(DeptNo));

Output:

Table created.

Inserting multiple records in “Dept” Table using parameters:

SQL> Insert into Dept values(&dno,'&dn');

Enter value for dno: 10

Enter value for dn: Sales

Output:

old 1: Insert into Dept values(&dno,'&dn')

new 1: Insert into Dept values(10,'Sales')

1 row created.

Repeating recent Command:

SQL> /

Enter value for dno: 20

Enter value for dn: HR

Output:

old 1: Insert into Dept values(&dno,'&dn')

new 1: Insert into Dept values(20,'HR')

1 row created.

SQL> /

Enter value for dno: 30

Enter value for dn: Payroll

Output:

old 1: Insert into Dept values(&dno,'&dn')

new 1: Insert into Dept values(30,'Payroll')

1 row created.

```
SQL> /
Enter value for dno: 40
Enter value for dn: Orders
```

Output:

```
old  1: Insert into Dept values(&dno,'&dn')
new  1: Insert into Dept values(40,'Orders')
```

1 row created.

Retrieving all columns and records from “dept” table:

```
SQL> Select * from Dept;
```

DEPTNO	DNAME
10	Sales
20	HR
30	Payroll
40	Orders

Inserting Employee records in “Emp” Table:

```
SQL> Insert into Emp values(1001,'Ravi','clerk',6000,20);
```

Output:

1 row created.

```
SQL> Insert into Emp values(1002,'Sai','Sales Man',5000,10);
```

Output:

1 row created.

**Checking References Constraint of Emp Table’s DeptNo Column
accepts Primary Key values of Dept Table’s DeptNo values:**

```
SQL> Insert into Emp values(1003,'Vijay','Sales Man',5000,90);
```

Output:

ERROR at line 1:

ORA-02291: integrity constraint (SYSTEM.SYS_C007595) violated - parent key not found

Creating Tables with Constraint Names:**Example-1:**

Create a Table with following structure:

Student3

Std_Id	Std_Name	M1
--------	----------	----

Set Primary Key for Std_Id with Constraint Name c1

Set Not Null for Std_Name with Constraint Name c2

Set Check for M1 with Constraint Name c3 [M1 must be b/w 0 to 100]

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

Insert the Records

Test the constraints that are working properly or not.

Example-2:

Create a Table with following Structure:

Emp1

Empno	Ename	MobileNo	Company_City	Job	Salary
-------	-------	----------	--------------	-----	--------

Set Primary Key for Empno with constraint name emp_pk

Set Not Null for Ename with constraint name emp_nn

Set Unique for MobileNo with constraint name emp_u

Set Default value as 'Hyd' for Company_City

Set Check for Salary with constraint name emp_chk (Salary must be between 5000 to 100000)

Creating Table:

```
SQL> Create Table Emp1(  
    Empno Number(4) Constraint Emp_Pk Primary Key,  
    Ename varchar2(20) Constraint Emp_nn Not Null,  
    MobileNo varchar2(10) Constraint Emp_u Unique,  
    Company_City varchar2(12) Default 'Hyd',  
    Salary Number(8,2) Constraint Emp_chk Check(salary>=5000 and  
    salary<=10000));
```

Output:

Table created.

Note: We cannot give name to 'Defualt' Constraint

Insert the records

Test all Constraints working properly or not

Example-3:

Crete following Tables with Constraint Names and Constraint Types:

Course

Course_Id	Course_Name
-----------	-------------

Set Course Id as Primary Key with Constraint Name Course_PK

Student

Student_Id	SName	Course_Id
------------	-------	-----------

Set Course_Id as Foreign Key (References) with Constraint Name

Student_FK

Create above tables with constraint names and constraint types

Enter Course Table Records

Enter Student Table Records

Enter Course_Id value in "Student" Table which is not available in "Course" Table's Course_Id. It will not accept. Because, It accept Primary key values of Course Table's Course ID values.

“Alter” Command:

- It is a DDL Command.
- It is used to change the structure of database objects like tables, views, and indexes ...etc.
- This command can be used for following purposes:
 - To Add the Columns
 - To Modify the Field Sizes and Field Data types
 - To Drop the Columns
 - To Rename the Columns
 - To Rename the Constraints
 - To Disable the Constraints
 - To Enable the Constraints
 - To Add the Constraints
 - To Drop the Constraints

Syntax to Alter a Table:

```
Alter Table <Table_Name>
[add(<field_name> <data_type> [constraint <constraint_name> constraint_type>,...]);
[modify(<field_name> <data_type> [constraint <constraint_name> constraint_type>,...]);
[drop column <column_name>];
[drop(<column_name1>[,<column_name2>,.....]);
[rename column <old_name> to <new_name>];
[rename constraint <old_name> to <new_name>];
[Disable Constraint <Constraint_Name>];
[Enable Constraint <Constraint_Name>];
[Add Constraint <Constraint_Name> <Constraint_Type>(<Field_Name>)];
[Drop Constraint <Constraint_Name>];
```

Example-1 [Adding a Column]:

```
Alter Table emp add aadhar_num varchar2(12);
```

Example-2 [Deleting Columns]:

```
Alter Table emp drop(comm,doj);
```

Example-3 [Modifying field size]:

```
Alter Table emp modify ename varchar2(30);
```

Examples on Altering the Table:

Example-1:

Create a Table with following Structure:

Student5

Rno	Name
-----	------

Creating Table:

```
SQL> Create Table Student5(  
      StdId Number(2),  
      SName Varchar2(20));
```

Table created.

Note:

See Table Structure using “Desc” Command After Altering the table to see whether changes applied or not.

Ex: Desc Student5;

Adding a Column:

```
SQL> Alter Table Student5 add m1 number(3);  
(OR)  
SQL> Alter Table Student5 add(m1 number(3));
```

Output:

Table altered.

Adding Multiple Columns:

```
SQL> Alter Table Student5 add(m2 number(3), m3 number(3));
```

Output:

Table altered.

Dropping a Column:

```
SQL> Alter Table Student5 Drop Column m3;
```

(OR)

```
SQL> Alter Table Student5 Drop(m3);
```

Output:

Table altered.

Dropping Multiple Columns:

```
SQL> Alter table Student5 Drop(m1,m2);
```

Output:

Table altered.

Modifying Field Size:

```
SQL> Alter Table Student5 Modify Sname Varchar2(30);
```

(OR)

```
SQL> Alter Table Student5 Modify(Sname Varchar2(30));
```

Output:

Table altered.

Modifying Multiple Field Sizes:

```
SQL> Alter Table Student5 Modify(stdid number(4), Sname  
Varchar2(35));
```

Output:

Table altered.

Modifying Data Type:

```
SQL> Alter Table Student5 Modify stdid varchar2(10);
```

(OR)

```
SQL> Alter Table Student5 Modify(stdid varchar2(10));
```

Output:

Table altered.

Renaming a Column:

```
SQL> Alter Table Student5 rename Column Sname to Std_Name;
```

Output:

Table altered.

To See Structure of table:

```
SQL> desc Student5;
```

Name	Null?	Type
STDID		VARCHAR2(10)
STD_NAME		VARCHAR2(35)

Adding a Primary Key Constraint:

```
SQL> Alter Table Student5 add constraint std5_pk Primary Key(StdId);
```

Output:

Table altered.

Disabling a Constraint:

```
SQL> Alter Table Student5 disable constraint std5_pk;
```

Output:

Table altered.

Note: Temporarily Primary Key will not work for this field. Now this field can accept duplicate and null values.

```
SQL> Insert into Student5 values(10,'Raju');
```

Output:

1 row created.

SQL> Insert into Student5 values(10,'Sai');

Output:

1 row created.

Note: Duplicate record accepted with above query

SQL> Insert into Student5 values(null,'Vijay');

Output:

1 row created.

Note: Null value accepted in “StdId” field with above query

Enabling Primary Key Constraint:

Note: To enable it, the column should not have duplicate and null values

Remove or update duplicate and null values in Primary Key field to enable primary key.

Removing all Records:

SQL> Delete from Student5;

Output:

3 rows deleted.

Enabling Primary Key Constraint:

SQL> Alter table student5 enable constraint std5_pk;

Output:

Table altered.

Dropping Constraint:

```
SQL> Alter table student5 drop constraint std5_pk;
```

Output:

Table altered.

Note:

Using “Add Constraint”, we can add Table Level Constraints like Primary Key, Check, Foreign Key, and Unique. We cannot add Not Null and Default. These are column level constraints.

To add Not Null and Default, we can use “Modify”.

There are 2 types of Constraints. They are:

Column Level Constraints:

Column Level Constraints can be applied in column definition.

Column Level Constraints are:

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

Table Level Constraints:

Table Level Constraint are applied after all column definitions.

Table Level Constraints are:

- Primary Key
- Unique
- Check
- References [Foreign Key]

Adding Not Null Constraint:

```
SQL> Alter table student5 modify(std_name varchar2(35) constraint std5_nn not null);
```

Output:

Table altered.

Adding Mobile Number Column:

```
SQL> Alter table student5 add mobileno varchar2(10);
```

Output:

Table altered.

Adding Unique Constraint to MobileNo:

```
SQL> Alter table student5 add constraint std5_u unique(mobileno);
```

Table altered.

Adding 'clg_name' column:

```
SQL> Alter table student5 add clg_name varchar2(8);
```

Output:

Table altered.

Adding Default Constraint:

```
SQL> Alter table student5 modify(clg_name varchar2(8) default 'NareshIT');
```

Output:

Table altered.

Adding m1 Column:

```
SQL> Alter table student5 add m1 number(3);
```

Table altered.

Adding Check Constraint:

```
SQL> Alter table student5 add constraint std5_chk Check(m1>=0 and  
m1<=100);
```

Table altered.

Example-2:**Create following Tables:****Dept5**

DeptNo	DName

Emp5

Empno	Ename	DeptNo

Create Tables:

```
SQL> create table dept5(  
    deptno number(2),  
    dname varchar2(15));
```

Output:

Table created.

```
SQL> create table emp5(  
    empno number(4),  
    ename varchar2(20),  
    deptno number(2));
```

Output:

Table created.

Adding Primary Key to DeptNo Field of Dept Table:

```
SQL> Alter Table Dept5 add constraint Dept5_pk primary key(deptno);
```

Output:

Table altered.

Adding Primary Key to Empno Field of Emp5 Table :

```
SQL> Alter Table Emp5 add constraint Emp5_pk primary key(empno);
```

Output:

Table altered.

Adding Foreign Key Constraint to DeptNo Field of Emp5 Table:

```
SQL> Alter Table Emp5 add constraint Emp5_fk foreign key(deptno) references dept5(deptno);
```

Output:

Table altered.

“Drop” Command:

- It is a DDL Command.
- It is used to drop [delete] the database objects like tables, views, and Indexes ...etc.

Syntax to create a Table:

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

Example:

Drop Table Emp; -- Emp table dropped to Recycle bin

“Flashback” Command:

- It is a DDL Command.
- It is used to recollect the dropped database objects from recycle bin.

Syntax to Flashback [Recollect] a Table:

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

Example:

Flashback Table Emp to before drop; --Emp table restored

“Purge” Command:

- It is a DDL Command.
- It is used to remove a dropped database object from Recycle Bin.

Syntax to Purge [Remove] a Table from recycle bin:

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

Example:

Purge Table Emp; -- emp table deleted permanently

Note:

To empty the recycle bin we write:

SQL> Purge recyclebin;

Example on drop, flashback and purge:

Example-1:

Create aTable with following Structure:

Product

pid	pname
-----	-------

Set pid as Primary key with the name prd_pk

Insert the records

Drop the product table

Recollect from recycle bin using flashback

Drop the product table again & purge [remove] from recycle bin.

Create a Table t1 with f1 field of number type.

Drop t1 table permanently.

Creating Table:

```
SQL> create table product(
      pid number(4) constraint prd_pk primary key,
      pname varchar2(20) );
```

Output:

Table created.

Inserting multiple records using parameters:

```
SQL> Insert into product values(&pid,'&pname');
Enter value for pid: 1001
Enter value for pname: Hard Disk
```

Output:

```
old  1: Insert into product values(&pid,'&pname')
new  1: Insert into product values(1001,'Hard Disk')
1 row created.
```

Repeating above command:

```
SQL> /
Enter value for pid: 1002
Enter value for pname: Monitor
```

Output:

```
old  1: Insert into product values(&pid,'&pname')
new  1: Insert into product values(1002,'Monitor')
1 row created.
```

```
SQL> /
```

```
Enter value for pid: 1003
Enter value for pname: RAM
```

Output:

```
old 1: Insert into product values(&pid,'&pname')
new 1: Insert into product values(1003,'RAM')
1 row created.
```

```
SQL> /
Enter value for pid: 1004
Enter value for pname: SMPS
```

Output:

```
old 1: Insert into product values(&pid,'&pname')
new 1: Insert into product values(1004,'SMPS')
1 row created.
```

Retrieving all columns & all records:

```
SQL> select * from product;
```

PID	PNAME
1001	Hard Disk
1002	Monitor
1003	RAM
1004	SMPS

Dropping a Table:

```
SQL> Drop Table product;
```

Output:

```
Table dropped.
```

To see dropped objects of RecycleBin:

```
SQL> show recyclebin;
```

Output:

ORIGINAL NAME	RECYCLEBIN NAME	OBJECT TYPE	DROP TIME
PRODUCT	BIN\$7VyxkH38QPCoCke0Qx/t6Q==\$0	TABLE	2021-06-15:10:49:28

Note:

PRODUCT table dropped. So below query gives error.

```
SQL> select * from product;
```

Output:

ERROR at line 1:

ORA-00942: table or view does not exist

To flashback (Recollect) dropped Table:

```
SQL> flashback table product to before drop;
```

Output:

Flashback complete.

Note:

Product table recollected from recycle bin. Now we can see the records in Product Table.

```
SQL> select * from product;
```

PID	PNAME
1001	Hard Disk
1002	Monitor
1003	RAM
1004	SMPS

Dropping product Table:

```
SQL> Drop Table product;
```

Output:

Table dropped.

To see Recycle Bin:

```
SQL> show recyclebin;
```

Output:

ORIGINAL NAME	RECYCLEBIN NAME	OBJECT TYPE	DROP TIME
PRODUCT	BIN\$5eEJqn+HTF6yEaXYgkeKSg==\$0	TABLE	2021-06-15:11:06:17

To purge [remove] the Table from Recycle Bin:

```
SQL> purge table product;
```

Output:

Table purged.

Note:

product table deleted from recycle bin. It means, product table deleted permanently. We cannot recollect it.

Create a Table as following:

```
SQL> Create Table t1(f1 number(2));
```

Output:

Table Created.

Dropping above table t1 permanently:

```
SQL> Drop Table t1;
```

Output:

Table dropped.

```
SQL> purge table t1;
```

Output:

Table purged.

Note:

Using above two queries we can remove a table permanently.

(OR)

Instead of writing two queries we can delete t1 table permanently using one query as following:

```
SQL> drop table t1 purge;
```

Output:

Table dropped.

```
SQL> show recyclebin;
```

Output:

Empty

Note:

No dropped objects in recycle bin. It is empty.

“Truncate” Command:

- It is a DDL Command.
- It is used to remove all records from a table.
- It does not remove structure of the table.
- It cannot be used to remove one record (or) a set of records from table.

Syntax to truncate a Table:

```
Truncate Table <Table_Name>;
```

Example:

Truncate table emp; -- deleted all records from emp table

“Rename” Command:

- It is a DDL Command.
- It is used to rename the data base objects like tables.

Syntax to Rename a Table:

```
Rename <Old_Name> to <New_Name>;
```

Example:

`Rename emp to employee;` -- emp table renamed to employee

DML Commands

- DML Stands for Data Manipulation Language.
- DML Commands deal with the data.
- DML Commands are used to manipulate the data. Manipulation means, inserting, deleting, and modifying [updating] the records.
- We need to use “commit” command after using a “DML Command”. Because All DML commands are not auto committed.
- DML Commands provided by Oracle-SQL are:
 - Insert
 - Insert All
 - Delete
 - Update
 - Merge

“Insert” Command:

- It is a DML Command.
- It is used to insert the records into table.

Syntax :

```
Insert into <Table_Name>[<column_list>] values(<v1>, <v2>, .....);
```

Example:

```
Insert into emp(empno,ename,job,sal) values(1001, 'Ramu', 'CLERK', 4000);
```

“Insert All” Command:

- It is a DML Command.
- It is introduced in Oracle 9i version.
- It is used to insert the multiple records into a table or multiple tables by writing one query.
- It avoids writing multiple insert commands.

Syntax to insert record into Table:

```

Insert All
into <Table_Name>[<column_list>] values(<v1>,<v2>,...)
into <Table_Name>[<column_list>] values(<v1>,<v2>,...)
into <Table_Name>[<column_list>] values(<v1>,<v2>,...)

Sub Query

```

Example-1 [Inserting Multiple Records in One Table]:

Insert All

```

into emp(empno,ename,job) values(1001, 'Ramu', 'CLERK')
into emp(empno,ename,job) values(1002, 'Vijay', 'MANAGER')
into emp(empno,ename,job) values(1003, 'Arun', 'SALESMAN')
Select * from dual;

```

Example-2 [Inserting Multiple Records in Multiple Tables]:

Insert All

```

Into Customer Values(1001, 'Sravan', 'Hyd')
Into Emp Values(2001, 'Sai', 'CLERK')
Into Product Values(3001, 'Soap', 30.00)
Select * from dual;

```

Example on INSERT and INSERT ALL:

Create a table with following structure and insert records into table & Customer

CID	CNAME	CCITY
-----	-------	-------

Creating Table:

```

SQL> create table customer(
      cid number(4),
      cname varchar2(20),
      ccity varchar2(10) );

```

Output:

Table created.

Inserting a record:

```
SQL> Insert into Customer values(1001,'Ravi','Hyd');
```

Output:

1 row created.

Inserting multiple records using parameters:

```
SQL> Insert into Customer values(&cid,'&cname','&city');
```

Enter value for cid: 1002

Enter value for cname: Krishna

Enter value for city: Mumbai

Output:

```
old 1: Insert into Customer values(&cid,'&cname','&city')
```

```
new 1: Insert into Customer values(1002,'Krishna','Mumbai')
```

1 row created.

```
SQL> /
```

Enter value for cid: 1003

Enter value for cname: Prasad

Enter value for city: Bangalore

Output:

```
old 1: Insert into Customer values(&cid,'&cname','&city')
```

```
new 1: Insert into Customer values(1003,'Prasad','Bangalore')
```

1 row created.

Inserting limited column values:

```
SQL> Insert into customer(cid,cname) values(1004,'Kishore');
```

Output:

1 row created.

Inserting limited column values by changing the order of columns:

```
SQL> Insert into customer(cname,cid) values('Naresh',1005);
```

Output:

1 row created.

Retrieving all customers records:

```
SQL> Select * from Customer;
```

Output:

CID	CNAME	CCITY
1001	Ravi	Hyd
1002	Krishna	Mumbai
1003	Prasad	Bangalore
1004	Kishore	
1005	Naresh	

Creating another table with same fields & records of “CUSTOMER”**Table [Copying a Table]:**

```
SQL> Create Table Customer1 As Select * From Customer;
```

Output:

Table created.

Inserting records from existing Table [Copying records from existing Table]:

```
SQL> Insert into Customer1  
      Select * from Customer;
```

Output:

5 rows created.

Inserting multiple records into “Customer” table using “INSERT ALL”:

```
SQL> INSERT ALL  
      into Customer values(1006,'Ramu','Delhi')  
      into Customer values(1007,'Kiran','Hyd')  
      into Customer values(1008, 'Arun', 'Chennai')  
      Select * from dual;
```

Output:

3 rows created.

Inserting multiple records into multiple tables [“Customer” and “Customer1” tables] using “INSERT ALL”:

```
SQL> INSERT ALL
      into Customer values(1009,'Ramesh','Chennai')
      into Customer1 values(1009,'Ramesh','Chennai')
      into Customer values(1010,'Sridhar','Hyd')
      into Customer1 values(1010,'Sridhar','Hyd')
      Select * from dual;
```

4 rows created.

“Delete” Command:

- It is a DML Command.
- It is used to delete the records.
- Using this command we can delete single record, set of records or all records.

Syntax :

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

Example:

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

Example on “Delete” Command:

Create a table with following structure, Insert the records and delete the records:

Book

BookId	BookName	Author	Price	Publisher
1001	Oracle	Kevin	1200	TATA MH
1002	Oracle	Edward	1500	Techmedia
1003	Java	James	1000	TATA MH
1004	C#.Net	Steve	3500	Wrox
1005	Java	Norton	2000	Techmedia

Creating Table:

```
SQL> Create Table Book(  
      Bookid Number(4),  
      BookName varchar2(10),  
      Author Varchar2(10),  
      price number(7,2),  
      publisher varchar2(10));
```

Output:

Table created.

Inserting multiple records in “Book” Table using “INSERT ALL”:

```
SQL> Insert All  
      into Book values(1001,'Oracle','Kevin',1200,'TATAMH')  
      into Book values(1002,'Oracle','Edward',1500,'Techmedia')  
      into Book values(1003,'Java','James',1000,'TATAMH')  
      into Book values(1004,'C#.Net','Steve',3500,'Wrox')  
      into Book values(1005,'Java','Norton',2000,'Techmedia')  
      Select * from dual;
```

Output:

5 rows created.

Delete the book which bookid is 1005 [Deleting a Record]:

```
SQL> Delete from book where bookid=1005;
```

Output:

1 row deleted.

Deleting all Oracle Books [Deleting multiple records]:

```
SQL> Delete from Book where BookName='Oracle';
```

Output:

2 rows deleted.

Deleting All Records:

```
SQL> Delete from Book;
```

Output:

2 rows deleted.

“Update” Command:

- Is a DML Command.
- It is used to update [modify] the records.
- It can also be used to perform calculations and store the result in tables.
- Using this command we can update single record, a set of records & all records.

Syntax:

```
Update <Table_Name>
Set <column-1>=<value-1>[, <column-2>=<value-2>, ....]
[Where <Condition>];
```

Example:

Update emp set sal=4000,comm=500 where empno=7900;

Examples on “Update” Command:

Example-1:

Create Emp Table with following structure, insert the records & perform update operations on emp table based on different conditions:

Emp

Empno	Ename	Job	MGR	HireDate	Sal	Comm	Deptno
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500		30
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

Creating Table & Inserting Records:

Above Table Table Creation Query and insert queries are available in “emp_dept_salgrade.txt” file [This text file available in google drive].

Increasing 10% salary to an employee whose empno is 7900 [Updating one field value of a record]:

```
SQL> Update emp Set sal=sal+sal*0.1 where empno=7900;
```

Output:

1 row updated.

Updating job value as manager and sal as 5000 to the employee whose empno is 7369 [Updating multiple field values of a record]:

```
SQL> Update emp set sal=5000, job='MANAGER' where empno=7369;
```

Output:

1 row updated.

Increasing 10% sal to all managers [Updating Multiple Records]:

```
SQL> update emp set sal=sal+sal*0.1 where job='MANAGER';
```

Output:

4 rows updated.

Increasing 15% salary to all employees [Updating all records]:

```
SQL> update emp set sal=sal+sal*0.15;
```

Output:

14 rows updated.

Updating records using parameters:

```
SQL> Update emp set sal=&sal where empno=&empno;
```

Enter value for sal: 2000

Enter value for empno: 7900

Output:

```
old  1: Update emp set sal=&sal where empno=&empno
```

```
new  1: Update emp set sal=2000 where empno=7900
```

1 row updated.

```
SQL> /  
Enter value for sal: 3000  
Enter value for empno: 7934
```

Output:

```
old  1: Update emp set sal=&sal where empno=&empno  
new  1: Update emp set sal=3000 where empno=7934
```

1 row updated.

Increasing 10% salary to all managers and clerks:

```
SQL> update emp set sal=sal+sal*0.1 where job='MANAGER' or  
job='CLERK';
```

[OR]

```
SQL> update emp set sal=sal+sal*0.1 where job IN('MANAGER','CLERK');
```

Increasing 10% salary to the employees whose salary is between 1000 and 2000:

```
SQL> update emp set sal=sal+Sal*0.1 where sal>=1000 and sal<=2000;  
[OR]
```

```
SQL> update emp set sal=sal+Sal*0.1 where sal between 1000 and  
2000;
```

Increasing 10% salary to all employees except managers:

```
update emp set sal=sal+sal*0.1 where job!='MANAGER';
```

[OR]

```
update emp set sal=sal+Sal*0.1 where job not in('MANAGER');
```

Increasing 10% sal to the employees whose comm is null:

```
Update emp set sal=sal+sal*0.1 where comm is null;
```

Increasing 10% sal to the employees whose comm is not null:

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

Updating comm as null to the employees whose comm is not null:

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

Updating comm as 500 to all employees:

SQL> Update emp set comm=500;

Updating comm as 1000 for all employees of deptno 20:

Update emp set comm=1000 where deptno=20;

Increasing 10% sal to the employees whose name is started with 'S':

Update emp set sal=sal+sal*0.1 where ename like 'S%';

Increasing 10% sal to the employees whose name's second character is 'A':

update emp set sal=sal+sal*0.1 where ename like '_A%';

Increasing 10% sal to the employees who joined in 1981:

Update emp set sal=sal+sal*0.1 where hiredate like '%81';

Increasing 10% salary to the employees of deptno 20 who joined in 1981:

Update emp set sal=sal+sal*0.1 where hiredate like '%81' and deptno=20;

Increasing 10% sal to the employees who joined in 82 and 83:

SQL> Update emp set sal=sal+sal*0.1 where hiredate like '%82' or hiredate like '%83';

[OR]

SQL> Update emp set sal=sal+sal*0.1 where hiredate between '1-JAN-1982' and '31-DEC-1983';

[OR]

Update emp set sal=sal+sal*0.1 where hiredate>='1-JAN-1982' and hiredate<='31-DEC-1983';

Increasing 10% sal to the employees whose name has 4 characters:

Update emp set sal=sal+sal*0.1 where ename like '____';

Increasing 10% sal to the employees who have greater than 40 years experience:

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

Example-2:

Create a Table with following Structure & Insert the records. Then calculate total and average marks:

Student

StdID	SName	M1	M2	M3	Total	Avrg
1001	Ravi	40	80	60		
1002	Sravan	66	44	77		

Creating “Student” Table:

```
SQL> Create Table Student(
      StdId Number(4),
      SName Varchar2(20),
      M1 Number(3),
      M2 Number(3),
      M3 Number(3),
      Total Number(3),
      Avrg Number(5,2));
```

Output:

Table created.

Inserting Records into “Student” Table:

```
SQL> Insert into Student(StdId,Sname,M1,M2,M3)
      values(1001,'Ravi',40,80,60);
```

Output:

1 row created.

```
SQL> Insert into Student(StdId,Sname,M1,M2,M3)
      values(1002,'Sravan',66,44,77);
```

Output:

1 row created.

Calculating Total:

```
SQL> Update Student Set Total=M1+M2+M3;
```

Output:

2 rows updated.

Calculating Average Marks:

```
SQL> Update Student Set Avrg=Total/3;
```

Output:

2 rows updated.

Retrieving All Records:

```
SQL> Select * from Student;
```

STDID	SNAME	M1	M2	M3	TOTAL	AVRG
1001	Ravi	40	80	60	180	60
1002	Sravan	66	44	77	187	62.33

Example-3:

Create a Table with following structure & insert the records. Calculate TA, DA, HRA and Gross Salary for all employees.

Employee

Empno	Ename	Job	BasicSal	TA	DA	HRA	GROSS
-------	-------	-----	----------	----	----	-----	-------

TA: 10% on Basic Salary

DA: 20% on Basic Salary

HRA: 15% Basic Salary

GROSS: BasicSal+TA+DA+HRA

Creating “Employee” Table:

```
SQL> Create Table Employee(
```

```
    Empno Number(4),
    Ename Varchar2(20),
    Job Varchar2(10),
    BasicSal Number(7,2),
    TA Number(7,2),
    DA Number(7,2),
    HRA Number(7,2),
    GROSS Number(7,2));
```

Output:

Table created.

Inserting Records:

```
SQL> Insert into Employee(Empno,Ename,Job,BasicSal)  
      values(1001,'Sai','CLERK',4000);
```

Output:

1 row created.

```
SQL> Insert into Employee(Empno,Ename,Job,BasicSal)  
      values(1002,'Krishna','MANAGER',8000);
```

Output:

1 row created.

Calculating TA, DA, HRA:

```
SQL> Update Employee  
      Set TA=BasicSal*0.1, DA=BasicSal*0.2, HRA=BasicSal*0.15;
```

Output:

2 rows updated.

Calculating Gross Salary:

```
SQL> Update Employee Set Gross=BasicSal+TA+DA+HRA;
```

Output:

2 rows updated.

Retrieving all Records:

```
SQL> Select * from Employee;
```

Output:

EMPNO	ENAME	JOB	BASICSL	TA	DA	HRA	GROSS
1001	Sai	CLERK	4000	400	800	600	5800
1002	Krishna	MANAGER	8000	800	1600	1200	11600

“Merge” Command:

- Is a DML Command.
- It is a combination of Update and Insert Commands.
- It is used to merge one table records into another table (Replica / Duplicate Copy) based on condition.
- It avoids writing multiple DML Statements.
- It is used to deal with Replicas [Duplicate Copies].

Syntax:

```
Merge into <Target_Table_Name> <Alias>
Using <Source_Table_Name> <Alias>
On(<Condition>)
When Matched Then
<Update Query>
When Not Matched Then
<Insert Query>
```

Example:

BOOK1 [BookID, Book_Name, Price] => Source Table
BOOK2 [BookID, Book_Name, Price] => Target Table

```
Merge into Book2 T
Using Book1 S
On (s.BookId=t.BookId)
When Matched Then
Update Set T.Price=S.Price
When Not Matched Then
Insert (bookid,book_name,price) values(s.bookid, s.book_name, s.price);
```

Example on “Merge” Command:

Create Customer1 Table with following structure. Insert the records and make a copy of customer1 table as customer2 table. Use Customer1 Table perform daily operations. Apply the daily changes to Customer2 using “Merge” command:

Customer1

Cid	Cname	Ccity
1001	Ravi	Hyd
1002	Kiran	Mumbai

Creating “Customer1” Table:

```
SQL> Create table Customer1(
    Cid Number(4),
    CName Varchar2(20),
    CCity Varchar2(10));
```

Output:

Table created.

Inserting Records into “Customer1” Table:

```
SQL> Insert into Customer1 values(1001,'Ravi','Hyd');
```

Output:

1 row created.

```
SQL> Insert into Customer1 values(1002,'Kiran','Mumbai');
```

Output:

1 row created.

Retrieving Customer1 Table Details:

```
SQL> Select * from Customer1;
```

Output:

CID	CNAME	CCITY
1001	Ravi	Hyd
1002	Kiran	Mumbai

Creating Customer2 Table using existing table Customer1 Table**[Copying Table]:**

```
SQL> Create Table Customer2 As Select * from Customer1;
```

Output:

Table created.

Retrieving Customer2 Table Details:

```
SQL> Select * from Customer2;
```

Output:

CID	CNAME	CCITY
1001	Ravi	Hyd
1002	Kiran	Mumbai

Perform daily operations like insert & update on “Customer1” Table:**Insertion:**

```
SQL> Insert into Customer1 values(1003,'Arun','Delhi');
```

Output:

1 row created.

Updation:

```
SQL> Update Customer1 Set Ccity='Bangalore' where Cid=1001;
```

Output:

1 row updated.

Retrieving Customer1 Table Details:

```
SQL> Select * from Customer1;
```

CID	CNAME	CCITY
1001	Ravi	Bangalore
1002	Kiran	Mumbai
1003	Arun	Delhi

Retrieving Customer2 Table Details:

```
SQL> Select * from Customer2;
```

CID	CNAME	CCITY
1001	Ravi	Hyd
1002	Kiran	Mumbai

Apply Customer1 Table Changes to Customer2 Table [Merging Records of Customer1 into Customer2]:

```
SQL> Merge into Customer2 T
      Using Customer1 S
      On(S.cid=T.cid)
      When Matched Then
        Update Set T.CCity=S.CCity
      When Not Matched Then
        Insert values(S.Cid,S.CName,S.CCity);
```

Output:

3 rows merged.

Flashback Query:

- After “commit” we cannot use “rollback” command. So, we cannot recollect the data using “rollback”. But if accidentally any record or records are deleted & committed to recollect the data, we can write Flashback Query.
- A query which returns the past data is called “Flashback Query”.
- “As Of” Clause is used to write the Flashback Query.
- Flashback Query is used to recollect the data which was existed some time ago. After “commit” also we can recollect the data using this query.
- Default “undo_retention” value is 900 seconds. It means, we can recollect the data upto 15 minutes. We can change “undo_retention” value.

Examples on Flashback Query:

Displaying 5 minutes ago data:

```
Select * from emp as of Timestamp(Sysdate-Interval '5' Minute);
```

Recollecting the 5 minutes ago data in a new Table:

```
Create Table Emp1 As
```

```
Select * from emp as of Timestamp(Sysdate-Interval '5' Minute);
```

Recollecting the 5 minutes ago data by inserting records into Table:

Insert into Emp

Select * from emp as of Timestamp(Sysdate-Interval '5' Minute);

Changing undo_retention value:

Log in as DBA

SQL> conn system/nareshit

SQL> show parameters

Output:

Displays parameters list. Check undo_retention value in the list. Default value is 900.

SQL> Alter system set undo_retention=3000; --3000 sec = 50 minutes

Output: System altered.

ACL Commands

- ACL Stands for Accessing Control Language.
- ACL commands deal with Data Accessibility.
- These commands are used to give permission on database objects like tables to other users & cancel the permissions on database objects from other users.
- ACL Commands provided by Oracle-SQL are:
 - Grant
 - Revoke

“Grant” Command:

- It is an ACL Command.
- It is used to give permissions on database objects like tables, views to other users.

Syntax :

```
Grant <Privileges_List> on <Table_Name> to <Users_List>;
```

Example:

```
Grant select on Std to raju;
```

Syntax to create a User:

```
Create User <User_name> identified by <password>
default tablespace <tablespace_name>
quota <memory> on <tablespace_name> ;
```

Example:

```
Create user c##raju identified by kumar
```

```
Default tablespace users
```

```
Quota unlimited on users;
```

“Revoke” Command:

- It is an ACL Command.
- It is used to cancel the permissions on database objects like tables, views from other users.

Syntax :

```
Revoke <Privileges_List> on <Table_Name> from <Users_List>;
```

Example:

```
Revoke select on Std from raju;
```

Example on GRANT and REVOKE:

Example-1:

Create two users with the names c##userA and c##userB. Log in as c##userA. Create a table with following structure, insert the records, give permissions on this table c##userB & Cancel the Permissions:

Student

Sid	SName

Creating a User with the name C##UserA:

Only DBA can create the user. So, Log In as DBA

```
SQL> conn system/nareshit
```

```
SQL> show User      -- checking current user
USER is "SYSTEM"
```

Note:

Give your DBA password in place of nareshit. When Oracle software installed, at that time you had given one password. That is DBA Password.

```
SQL> Create User c##userA identified by userA  
      default tablespace users  
      quota unlimited on users;
```

Output:

User created.

```
SQL> Grant connect, resource to c##userA;
```

Output:

Grant succeeded.

Creating a User with the name C##UserB:

Log in as DBA

```
SQL> conn system/nareshit;
```

Output:

Connected.

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

Output:

User created.

```
SQL> Grant connect, resource to c##userB;
```

Output:

Grant succeeded.

Note:

Connect and Resource are the roles on which DBA is giving permission to the userB.

Connect => This permission for log in

Resource => This permission for creating resources like tables

Log in as c##userA, create a table & insert the records:

SQL> conn c##userA/userA

Note: Password is case sensitive. Username is not case sensitive.

Output:

Connected.

SQL> show user

Output:

USER is "C##USERA"

SQL> Create Table Student(

 2 Sid Number(4),

 3 Sname Varchar2(20));

Output:

Table created.

SQL> Insert into Student values(10,'Ramu');

Output:

1 row created.

SQL> Insert into Student values(20,'Vijay');

Output:

1 row created.

SQL> Select * from Student;

Output:

SID SNAME

10 Ramu
20 Vijay

Give “Select” Permission to C##UserB:

SQL> Grant Select on Student to C##UserB;

Output:

Grant succeeded.

Connect as C##UserB to test the select permission:

SQL> conn C##UserB/userB

Output:

Connected.

SQL> Select * from C##UserA.student;

SID SNAME

10 Ramu
20 Vijay

SQL> Insert into C##UserA.student values(30,'Ravi');

Output:

ORA-01031: insufficient privileges

SQL> Delete from C##UserA.student where sid=10;

Output:

ORA-01031: insufficient privileges

```
SQL> Update c##userA.student set sname='Arun' where Sid=10;
```

Output:

ORA-01031: insufficient privileges

Note:

Above 3queries are giving same errors. Insufficient privileges. With this we can understand that c##userB has select permission only. Insert, Update and delete permissions had not given.

To Grant Insert, Update permissions to c##UserB:

Log in as c##userA

```
SQL> conn c##userA/userA
```

Output:

Connected.

```
SQL> Grant insert, update on student to c##userB;
```

Output:

Grant succeeded.

```
SQL> conn c##userB/userB
```

Connected.

```
SQL> show user
```

USER is "C##USERB"

```
SQL> Insert into c##UserA.student values(30,'Ravi');
```

Output:

1 row created.

```
SQL> Update c##userA.student set sname='Arun' where Sid=10;
```

Output:

1 row updated.

```
SQL> Delete from c##userA.student where sid=10;  
Delete from c##userA.student where sid=10
```

Output:

ORA-01031: insufficient privileges

Note:

With above 3 queries we can understand that c##UserB got permission for updating data and inserting the record. But unable to delete the record. Because, He has no “Delete” permission.

Granting all permissions:

```
SQL> conn c##userA/userA
```

```
SQL> Grant all on student to C##UserB;
```

Output:

Grant succeeded.

```
SQL> conn c##userB/userB
```

Output:

Connected.

```
SQL> Delete from c##userA.student where sid=10;
```

Output:

1 row deleted.

```
SQL> Insert into c##UserA.student values(40,'Sai');
```

Output:

1 row created.

```
SQL> Update c##userA.student set sname='Srinu' where Sid=20;
```

Output:

1 row updated.

Note:

With above 3 queries we can understand that now c##userB has all permissions. He can see the data, insert, delete & update the records.

Granting all permissions with grant option:

```
SQL> conn c##userA/userA
```

Output:

Connected.

```
SQL> Grant all on student to c##userB with Grant Option;
```

Output:

Grant succeeded.

Note:

If C##userA gives permission “with Grant Option” to C##UserB, Now C##UserB can give permission on this table to other users.

```
SQL> conn c##userB/userB
```

Output:

Connected.

```
SQL> Grant Select on c##userA.student to c##oracle7am;
```

Output:

Grant succeeded.

Note:

With above query c##userB is giving permission on c##userA's student table to other user "c##oracle7am".

Cancelling Delete permission from c##userB:

```
SQL> conn c##userA/userA
```

Output:

Connected.

```
SQL> Revoke delete on student from c##userB;
```

Output:

Revoke succeeded.

```
SQL> conn c##userB/userB
```

Output:

Connected.

```
SQL> Select * from c##userA.student;
```

SID	SNAME
20	Srinu
30	Ravi
40	Sai

```
SQL> delete from c##userA.student where sid=40;
```

Output:

ORA-01031: insufficient privileges

Note:

With above query we can understand that c##userB unable to delete the record. Because. “Delete” permission has been cancelled by c##userA.

Cancelling Insert, Update Permissions:

```
SQL> conn c##userA/userA
```

Output:

Connected.

```
SQL> Revoke Update, Insert on student from c##userB;
```

Output:

Revoke succeeded.

```
SQL> conn c##userB/userB
```

Output:

Connected.

```
SQL> Insert into c##userA.student values(50,'kiran');
```

Output:

ORA-01031: insufficient privileges

```
SQL> update c##userA.student set sname='raju' where sid=30;
```

Output:

ORA-01031: insufficient privileges

Note:

With above 2 queries we can understand that insert & update permissions canceled.

To cancel all permissions:

```
SQL> conn c##userA/userA
```

Output:

Connected.

```
SQL> Revoke all on student from c##userB;
```

Output:

Revoke succeeded.

To give select permission to all Users on student table:

```
SQL> Grant select on student to public;
```

Output:

Grant succeeded.

To cancel select permission from all users:

```
SQL> Revoke select on student from public;
```

Output:

Revoke succeeded.

To grant select permission to the users c##oracle7am, c##raju, c##userB:

```
SQL> Grant select on student to c##userB, c##raju, c##oracle7am;
```

Output:

Grant succeeded.

To cancel select permission from the users c##oracle7am, c##raju, c##userB:

```
SQL> Revoke select on student from c##userB, c##raju, c##oracle7am;
```

Output:

Revoke succeeded.

Changing Password:

Login as DBA

```
SQL> Alter user c##raju identified by nareshit;
```

Output:

User altered.

Dropping a User:

Log in as DBA

```
SQL> Drop User c##userB;
```

Output:

User dropped.

TCL Commands

- TCL Stands for Transaction Control Language.
- TCL commands deal with Transactions. A Transaction can be a series of commands (or) a single command.

Example:

Withdrawing Amount

Depositing Amount

Checking Balance

Placing an Order ... etc

- These commands are used to cancel the previous transactions or to save the transactions.
- A Transaction must be successfully completed or aborted [canceled].
- If a Transaction is successfully completed, then use “Commit” Command. If a Transaction is stopped in the middle due to some reason, then we use “roll back” command.
- TCL Commands provided by Oracle-SQL are:
 - Commit
 - Rollback
 - SavePoint

“Commit” Command:

- It is a TCL Command.
- It is used to make the changes permanent. It means, it is used to save the transactions. When we perform Insert or delete or update operation, it will be performed on “Oracle Instance”. To apply these changes to “Oracle Database” permanently, we use “commit” command.
- After commit, we cannot use rollback.

Syntax :

Commit;

Example:

```
Insert into student values(1234, 'Ravi');  
Commit;
```

In the above example, insertion will be done on “Oracle Instance”. To make this change permanent, we used “Commit” command.

Note:

Not only one. But also, all the previous actions will be saved permanently in “Oracle Database” when we use “Commit” command.

“Rollback” Command:

- It is a TCL Command.
- It is used to cancel the previous actions. It means, it will undo the previous actions.
- It must be used before “Commit” to cancel the previous actions. After commit, we cannot use “Rollback” Command.

Syntax :

```
Rollback [To SavePoint_Name];
```

Example:

```
Insert into student values(1234, 'Ravi');  
Rollback;
```

In the above example, insertion will be done on “Oracle Instance”. Then insertion will be canceled in “Oracle Instance” because of “Rollback” Command.

Note:

Not only one. But also, all the previous actions will be canceled when we use “Rollback” command [Uncommitted actions will be cancelled].

“SavePoint” Command:

- It is a TCL Command.
- It is used to set margin for “Rollback”. When rollbacked using the Save_Point_Name, the actions will be cancelled up to this point only.
- Save_Point_name must be unique.
- If we define same Save_Point_Name again, previous SavePoint will be replaced with new SavePoint. This new point will be considered as SavePoint. Previous savepoint [old savepoint] will be gone.

Syntax :

```
SavePoint <Save_Point_name>;
```

Example:

```
Insert into student values(1234, 'Ravi');  
SavePoint NareshIT;  
Insert into student values(1235, 'Sai');  
Rollback To NareshIT;
```

In the above example, rollback will be applied till the SavePoint ‘NareshIT’. It means second insertion will be cancelled.

Example on Rollback, Commit and Savepoint:

Example-1:

Create a table with following structure, perform manipulations, save the manipulations, and cancel the manipulations:

Product

Pid	PName	Qunatity_on_Hand
-----	-------	------------------

Creating “Product” Table:

```
SQL> Create Table Product(  
    PId Number(4),  
    PName Varchar2(15),  
    Quantity_On_Hand Number(5));
```

Output:

Table created.

Inserting 2 records and saving them:

```
SQL> Insert into Product values(1001,'Hard Disk',2000);
```

Output:

1 row created.

```
SQL> Insert into Product values(1002,'Pen Drive',3000);
```

Output:

1 row created.

```
SQL> Commit;
```

Output:

Commit complete.

Note:

Above two insertions will be saved permanently

Inserting two records and canceling those two insertions:

```
SQL> Insert into Product values(1003,'Monitor',1500);
```

Output:

1 row created.

```
SQL> Insert into Product values(1004,'Keyboard',2500);
```

Output:

1 row created.

```
SQL> Select * from Product;
```

Output:

PID	PNAME	QUANTITY_ON_HAND
1001	Hard Disk	2000
1002	Pen Drive	3000
1003	Monitor	1500
1004	Keyboard	2500

```
SQL> Rollback;
```

Output:

Rollback complete.

```
SQL> Select * from Product;
```

Output:

PID	PNAME	QUANTITY_ON_HAND
1001	Hard Disk	2000
1002	Pen Drive	3000

Note:

Previous two insertions are canceled because of “Rollback”

Using Savepoint:

```
SQL> select * from product;
```

Output:

PID	PNAME	QUANTITY_ON_HAND
1001	Hard Disk	2000
1002	Pen Drive	3000

Setting SavePoint p1:

```
SQL> Savepoint p1;
```

Output:

Savepoint created.

Inserting a Record:

```
SQL> Insert into product values(2001,'SMPS',4000);
```

Output:

1 row created.

Setting Savepoint p2:

```
SQL> Savepoint p2;
```

Output:

Savepoint created.

Updating a Record:

```
SQL> Update product set Quantity_On_Hand=5000 where pid=1001;
```

Output:

1 row updated.

Setting Savepoint p3:

```
SQL> SavePoint p3;
```

Output:

Savepoint created.

Deleting a Record:

```
SQL> Delete from product where pid=1002;
```

Output:

1 row deleted.

Retrieving Product Table records:

```
SQL> Select * from product;
```

Output:

PID	PNAME	QUANTITY_ON_HAND
1001	Hard Disk	5000
2001	SMPS	4000

Rollbacking upto savepoint p3:

```
SQL> rollback to p3;
```

Output:

Rollback complete.

```
SQL> Select * from product;
```

Output:

PID	PNAME	QUANTITY_ON_HAND
1001	Hard Disk	5000
1002	Pen Drive	3000
2001	SMPS	4000

Note:

Rollbacked up to p3. Deleted record “1002”. Came again in table. It means, deletion cancelled. But, not cancelled insertion and updation.

Rollbacking upto savepoint p2:

SQL> rollback to p2;

Output:

Rollback complete.

SQL> Select * from product;

Output:

PID	PNAME	QUANTITY_ON_HAND
1001	Hard Disk	2000
1002	Pen Drive	3000
2001	SMPS	4000

Note:

Rollbacked up to p2. Deletion and updation canceled.

Rollbacking upto Savepoint p1:

SQL> rollback to p1;

Output:

Rollback complete.

SQL> Select * from product;

Output:

PID	PNAME	QUANTITY_ON_HAND
1001	Hard Disk	2000
1002	Pen Drive	3000

Note:

Rollbacked up to p1. Deletion, updation and insertion canceled.

Built-In Functions

- Function is a set of statements [or Sub Program] that gets executed on calling.
- A Function can take arguments [Input] and can return the result [Output].
- Every function is defined to perform task (or) action.
- Oracle developers defined some functions in Oracle Database to perform tasks. These are already built in Oracle Database. So, these are called “Built-In Functions”. We can also call them as “Predefined Functions”. Because they are already defined.

Built-In Functions can be categorized as follows:

- Math Functions [Numeric Functions]
- String Functions [Character Functions]
- Aggregate Functions [Group Functions]
- Conversion Functions
- Date Functions
- Miscellaneous [Other] Functions

Math Functions [Numeric Functions]:

SQL provides following Math Functions:

Function Name	Purpose
Sqrt()	<p>Returns square root value of specified number.</p> <p>Syntax: Sqrt(<Number>)</p> <p>Ex: Sqrt(100) => 10 Sqrt(81) => 9</p>
Power()	<p>Returns power value of specified number and power.</p> <p>Syntax: Power(<Number>, <Power>)</p> <p>Ex: Power(2,3) => 8 Power(5,2) => 25</p>
Abs()	<p>Returns Absolute value [Positive Value] of specified number.</p> <p>Syntax: Abs(<Number>)</p> <p>Ex: Abs(5) => 5 Abs(-5) => 5</p>
Sign()	<p>Returns sign of specified number. If number is positive, it returns 1. If number is negative, it returns -1. If number is zero, it returns 0.</p>

	<p>Syntax: Sign(<Number>)</p> <p>Ex: Sign(5) => 1 Sign(-5) => -1 Sign(0) => 0</p>
Mod()	<p>Returns remainder value of specified number and divisor.</p> <p>Syntax: Mod(<Number>, <Divisor>)</p> <p>Ex: Mod(5,2) => 1 Mod(10,6) => 4</p>
Sin()	<p>Returns Sine value of specified angle. Angle must be specified in the form of radians.</p> <p>Syntax: Sin(<Angle>)</p> <p>Ex: Sin(90*3.14/180) => 1</p>
Cos()	<p>Returns Cosine value of specified angle. Angle must be specified in the form of radians.</p> <p>Syntax: Cos(<Angle>)</p> <p>Ex: Cos(0*3.14/180) => 1</p>

Tan()	Returns Tangent value of specified angle. Angle must be specified in radians. Syntax: Tan(<Angle>) Ex: Tan(45*3.14/180) => 1
Ln()	Returns natural logarithmic value of specified number. Syntax: Ln(<Number>) Ex: Ln(7) => 1.94591015 Ln(10) => 2.30258509
Log()	Returns logarithmic value of specified number and base. Syntax: Log(<Number>, <Base>) Ex: Log(10,10) => 1
Trunc()	Returns truncated value. It is used to remove specified number of decimal places from the number. Syntax: Trunc(<Number> [, <Number_Of_Decimal_Places>]) Ex: Trunc(123.4567) => 123 Trunc(123.4567, 1) => 123.4 Trunc(123.4567, 2) => 123.45

	<p>Trunc(123.4567, 3) => 123.456 Trunc(123.4567, -1) => 120 Trunc(123.4567, -2) => 100</p>
Round()	<p>Returns rounded value. This function returns upper value if specified value is equals to 0.5 or greater than 0.5. It returns lower value if specified value is less than 0.5.</p> <p>Syntax: Round(<Number>, <Number of Decimal_Places>)</p> <p>Ex:</p> <p>Round(123.4567) => 123 Round(123.6567) => 124 Round(123.5567) => 124 Round(123.4567,2) => 123.46 Round(123.4547,2) => 123.45 Round(123.4557,2) => 123.46 Round(123.4557,-1) => 120 Round(126.4557,-1) => 130 Round(123.4557,-2) => 100 Round(126.4557,-2) => 100 Round(153.4557,-1) => 150 Round(153.4557,-2) => 200</p>
Ceil()	<p>Always returns upper nearest integer value of specified number.</p> <p>Syntax: Ceil(<Number>)</p> <p>Ex: Ceil(123.4567) => 124</p>
Floor()	<p>Always returns lower nearest integer value of specified number.</p>

	<p>Syntax: Floor(<Number>)</p> <p>Ex: Ceil(123.4567) => 123</p>
--	--

String Functions [Character Functions]:

SQL provides following String Functions:

Function Name	Purpose
Upper()	<p>Returns string in upper case [All Capital Letters].</p> <p>Syntax: Upper(<String>)</p> <p>Ex: Upper('raju') => RAJU</p>
Lower()	<p>Returns string in lower case [All Small Letters].</p> <p>Syntax: Lower(<String>)</p> <p>Ex: Lower('RAJU') => raju</p>
Initcap()	<p>Returns every word's first letter in upper case and remaining characters in lower case.</p> <p>Syntax: Initcap(<String>)</p>

	<p>Ex: Initcap('RAJ KUMAR') => Raj Kumar</p>
Concat()	<p>Returns concatenated [Combined] string. It is used to concatenate [combine] two strings.</p> <p>Syntax: Concat(<String-1>, <String-2>)</p> <p>Ex: Concat('Raj', 'Kumar') => RajKumar Concat(Concat('Raj', ' '), 'Kumar') => Raj Kumar</p>
Length()	<p>Returns length of the string. It is used to find number of characters in string. 'Space' also treated as a character.</p> <p>Syntax: Length(<String>)</p> <p>Ex: Length('Sai') => 3 Length('Arun') => 4 Length('Sai Teja') => 8</p>
Lpad()	<p>Returns String which is filled with specified characters from left side. Default Char is '' [Space].</p> <p>Syntax: Lpad(<String>, <Size> [, <Char / Chars>])</p> <p>Ex: Lpad('raju', 10) => raju -- 6 spaces & raju</p> <p>Lpad('raju', 10, '*') => *****raju Lpad('sai', 10, '@') => @@@@@@@@sai Lpad('raju', 10, '#\$') => #\$\$#\$#\$raju</p>

Rpad()	<p>Returns String which is filled with specified characters from right side. Default Char is '' [Space].</p> <p>Syntax:</p> <p>Rpad(<String>, <Size> [, <Char / Chars>])</p> <p>Ex:</p> <p>Rpad('raju', 10) => raju -- raju & 6 spaces</p> <p>Rpad('raju', 10, '*') => raju*****</p> <p>Rpad('sai', 10, '@') => sai@{@{@{@{@{@{@</p> <p>Rpad('raju', 10, '#\$') => raju#\$\$#\$\$</p>
Ltrim()	<p>Returns the string by removing all characters from left side which are in character set. Default char is '' [Space].</p> <p>Syntax:</p> <p>LTrim(<String> [, <Char_Set>])</p> <p>Ex:</p> <p>LTrim(' raju') => raju -- removes left side spaces</p> <p>LTrim('@@@Raju', '@') => Raju</p> <p>LTrim('abcabcaabbccRaju', 'abc') => Raju</p>
RTrim()	<p>Returns the string by removing all characters from right side which are in character set. Default char is '' [Space].</p> <p>Syntax:</p> <p>RTrim(<String> [, <Char_Set>])</p> <p>Ex:</p> <p>RTrim('raju ') => raju -- removes right side spaces</p> <p>RTrim('Raju@{@{@{@', '@') => Raju</p> <p>RTrim('Rajuabcabcaabbcc','abc') => Raju</p>

Trim()	<p>Returns the string by removing leading characters or trailing characters or both.</p> <p>Syntax:</p> <p>Trim(<Both / Leading / Trailing> <char> from <string>)</p> <p>Ex:</p> <p>Trim(' raju ') => raju -- removes left & right side spaces</p> <p>Trim(Leading '@' from '@@@@raju') => raju</p> <p>Trim(Trailing '@' from 'raju@ @@ @') => raju</p> <p>Trim(Both '@' from '@@@raju@ @@ @') => raju</p>
Substr()	<p>Returns substring from the string. It is used to get characters from position to specified number of chars.</p> <p>Syntax:</p> <p>Substr(<String>, <Position> [, <Number_Of_Chars>])</p> <p>Ex:</p> <p>substr('raj kumar',5) => kumar substr('raj kumar',1,3) => raj substr('raj kumar',1,5) => raj k substr('raj kumar',5,3) => kum substr('raj kumar',-4) => umar substr('raj kumar',-4,3) => uma</p>
Instr()	<p>Returns sub string position in the string. Default position value is 1. Default occurrence value is 1.</p> <p>Syntax:</p> <p>Instr(<String>, <Sub-String> [, <Position>, <Occurrence>])</p> <p>Ex:</p> <p>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</p>

	<pre>Instr('This is his wish','is',-1) => 14 Instr('This is his wish','is',-1,2) => 10 Instr('This is his wish','is',-4) => 10 Instr('This is his wish','is',-4,2) => 6 Instr('This is his wish','is',-4,3) => 3</pre>
Replace()	<p>Returns string by replacing every occurrence of search string with replacement string.</p> <p>Syntax:</p> <p>Replace(<String>, <Search_String>, <Replace_String>)</p> <p>Ex:</p> <p>Replace('raj kumar','raj','sai') => sai kumar</p> <p>Replace('abcabcxyzzaabbccabc','abc','pqr') => pqrpqrxzyzaabbccpqr</p>
Translate()	<p>Returns string by replacing all occurrences of each character in from_string with corresponding character of to_string.</p> <p>Syntax:</p> <p>Translate(<String>, <From_String>, <To_String>)</p> <p>Ex:</p> <p>Translate('abcabcxyzzaabbccabc','abc','pqr') => pqrpqrxzyppqqrrpqr</p>
Soundex()	<p>Returns a string which is pronounced same as specified string. It is used to get the strings which are sounding same when pronouncing.</p> <p>Syntax:</p> <p>Soundex(<String>)</p>

	<p>Ex: Select ename from emp where Soundex(Ename) = Soundex('Smyt'); => SMITH</p>
Ascii()	<p>Returns an ASCII value of specified character.</p> <p>Syntax: ASCII(<Char>)</p> <p>Ex: ASCII('A') => 65 ASCII('a') => 97</p>
Chr()	<p>Returns the character of specified ASCII value.</p> <p>Syntax: Chr(<Number>)</p> <p>Ex: Chr(65) => A Chr(97) => a</p>

Aggregate Functions [Group Functions] :

SQL provides following Aggregate Functions:

Function Name	Purpose
Sum()	<p>Returns sum of values of a column.</p> <p>Syntax: Sum(<Column>)</p>

	Ex: Select Sum(Sal) from emp;
Avg()	Returns average value of a set of values of a column. Syntax: Avg(<Column>) Ex: Select Avg(Sal) from emp;
Max()	Returns maximum value in a column. Syntax: Max(<Column>) Ex: Select Max(Sal) from emp;
Min()	Returns minimum value in a column. Syntax: Min(<Column>) Ex: Select Min(Sal) from emp;
Count()	Returns number of values in a column. Syntax: Select Count(Comm) from emp; -- cannot count nulls Select Count(Empno) from emp; --cannot count nulls Select Count(*) from emp; --counts number of records

Conversion Functions:

SQL provides following Conversion Functions:

Function Name	Purpose																						
To_Char(Date)	<p>It is used to convert a date value to String type. Using this function, we can get date in different formats like mm/dd/yyyy (or) dd/mm/yyyy.</p> <p>Syntax: To_Char(<Date> [, Format])</p> <p>Ex:</p> <table> <tbody> <tr><td>To_Char(Sysdate)</td><td>=> 12-JUL-21</td></tr> <tr><td>To_Char(Sysdate,'dd')</td><td>=> 12</td></tr> <tr><td>To_Char(Sysdate,'mm')</td><td>=> 07</td></tr> <tr><td>To_Char(Sysdate,'yy')</td><td>=> 21</td></tr> <tr><td>To_Char(Sysdate,'yyyy')</td><td>=> 2021</td></tr> <tr><td>To_Char(Sysdate,'hh:mi AM')</td><td>=> 12:53 AM</td></tr> <tr><td>To_Char(Sysdate,'Q')</td><td>=> 3</td></tr> <tr><td>To_Char(Sysdate,'d')</td><td>=> 2</td></tr> <tr><td>To_Char(Sysdate,'dy')</td><td>=> mon</td></tr> <tr><td>To_Char(Sysdate,'day')</td><td>=> Monday</td></tr> <tr><td>To_Char(Sysdate,'CC')</td><td>=> 21</td></tr> </tbody> </table>	To_Char(Sysdate)	=> 12-JUL-21	To_Char(Sysdate,'dd')	=> 12	To_Char(Sysdate,'mm')	=> 07	To_Char(Sysdate,'yy')	=> 21	To_Char(Sysdate,'yyyy')	=> 2021	To_Char(Sysdate,'hh:mi AM')	=> 12:53 AM	To_Char(Sysdate,'Q')	=> 3	To_Char(Sysdate,'d')	=> 2	To_Char(Sysdate,'dy')	=> mon	To_Char(Sysdate,'day')	=> Monday	To_Char(Sysdate,'CC')	=> 21
To_Char(Sysdate)	=> 12-JUL-21																						
To_Char(Sysdate,'dd')	=> 12																						
To_Char(Sysdate,'mm')	=> 07																						
To_Char(Sysdate,'yy')	=> 21																						
To_Char(Sysdate,'yyyy')	=> 2021																						
To_Char(Sysdate,'hh:mi AM')	=> 12:53 AM																						
To_Char(Sysdate,'Q')	=> 3																						
To_Char(Sysdate,'d')	=> 2																						
To_Char(Sysdate,'dy')	=> mon																						
To_Char(Sysdate,'day')	=> Monday																						
To_Char(Sysdate,'CC')	=> 21																						
To_Date()	<p>It is used to convert string to date.</p> <p>Syntax: To_Date(<String> [, <Format>])</p> <p>Ex:</p> <table> <tbody> <tr><td>To_Date('23 december 2020')</td><td>=> 23-DEC-20</td></tr> <tr><td>To_Date('25-Dec-2019')</td><td>=> 25-DEC-19</td></tr> <tr><td>To_Date('25/12/2019','dd/mm/yyyy')</td><td>=> 25-DEC-19</td></tr> </tbody> </table>	To_Date('23 december 2020')	=> 23-DEC-20	To_Date('25-Dec-2019')	=> 25-DEC-19	To_Date('25/12/2019','dd/mm/yyyy')	=> 25-DEC-19																
To_Date('23 december 2020')	=> 23-DEC-20																						
To_Date('25-Dec-2019')	=> 25-DEC-19																						
To_Date('25/12/2019','dd/mm/yyyy')	=> 25-DEC-19																						
To_Number()	It is used to convert String to Number.																						

	<p>Syntax: To_Number(<String> [, <Format>])</p> <p>Ex: To_Number('123') => 123 TO_Number('\$1,000', 'L9,999') => 1000</p>
To_Char(Number)	<p>Used to convert Number to String. It can be used to apply number formats like currency symbols, currency names, thousand separators ...etc.</p> <p>Syntax: To_Char(<Number> [, <Format>])</p> <p>Ex: To_Char(123) => 123 To_Char(2000,'L9,999.99') => \$2,000.00 To_Char(2000,'C9,999.99') => USD2,000.00</p>

Date Functions:

SQL provides following Date Functions:

Function Name	Purpose
Sysdate	<p>Returns current system date.</p> <p>Ex: Sysdate => 12-JUL-21</p>
Add_Months()	<p>Used to add months to specified date or subtract from specified date.</p> <p>Syntax: Add_Months(<Date>, <Number_Of_Months>)</p>

	<p>Ex:</p> <p>Add_Months(sysdate,2) => 12-SEP-21 Add_Months(sysdate,-2) => 12-MAY-21</p>
Next_Day()	<p>Returns the date that falls on next weekday given from the date.</p> <p>Syntax: <code>Next_Day(Date, <Week_day_name>)</code></p> <p>Ex: <code>Next_Day(sysdate, 'fri') => 16-JUL-21</code></p>
Last_Day()	<p>Returns last day of specified month in the date.</p> <p>Syntax: <code>Last_Day(<Date>)</code></p> <p>Ex: <code>Last_Day(sysdate) => 31-JUL-21</code></p>
Months_Between()	<p>Returns the number of months between two dates.</p> <p>Syntax: <code>Months_Between(<Date-1>, <Date-2>)</code></p> <p>Ex: <code>Months_Between(sysdate, '12-APR-21') => 3</code></p>

Miscellaneous [Other] Functions:

Function Name	Purpose
User	Returns Current Username. Ex: User => C##Oracle7AM
UId	Returns Current User ID. Ex: UId => 111
Greatest	Returns greatest value in specified values. It can be used to find greatest value in a row. Syntax: Greatest(<n1>,<n2>,.....) Ex: Greatest(10,20,50,30) => 50
Least	Returns Least value in specified values. It can be used to find least value in a row. Syntax: Least(<n1>,<n2>,.....) Ex: Least(10,20,50,40) => 10
NVL()	Returns first value if first value is not null. Returns second value if first value is null. It is used to replace null value with other value. Syntax:

	<p>NVL(<First_Value>, <Second_Value>)</p> <p>Ex:</p> <p>NVL(100,200) => 100 NVL(Null,200) => 200</p>
NVL2()	<p>It returns Second Value if first value is not null. It returns third value if the first value is null. It is used to replace nulls and not nulls.</p> <p>Syntax:</p> <p>NVL2(<first_value>, <second_value>, third_Value)</p> <p>Ex:</p> <p>NVL2(100,200,300) => 200 NVL2(Null,200,300) => 300</p>
VSize()	<p>Returns number of bytes in internal representation of value.</p> <p>Syntax:</p> <p>VSize(<value>)</p> <p>Ex:</p> <p>VSize('raju') => 4</p>
Rank()	<p>Used to apply ranks based on a column or columns. It does not maintain sequence of ranks when multiple values have same rank. It generates gaps between ranks when multiple values have same rank.</p> <p>Syntax:</p> <p>Rank() over(order by <column_name> Asc/ Desc [,...])</p> <p>Ex:</p> <p>Rank() over(order by sal desc) =></p>

	applies ranks based on highest salary. It gives same rank for same values. It generates the gaps between ranks when same rank applied for multiple values.
Dense_Rank()	<p>Used to apply ranks based on a column or columns. It maintains sequence of ranks when even if multiple values have same rank. It will not generate gaps between ranks even if multiple values have same rank.</p> <p>Syntax: <code>Dense_Rank() over(order by <column_name> Asc/ Desc [,...])</code></p> <p>Ex: <code>Dense_Rank() over(order by sal desc) =></code></p> <p>applies ranks based on highest salary. It gives same rank for same values. It will not generate the gaps between ranks even if same rank applied for multiple values.</p>
Row_Number()	<p>Used to apply row number for every record.</p> <p>Syntax: <code>Row_Number() over(order by <column_name> Asc/ Desc [,...])</code></p> <p>Ex: <code>Row_Number() over(order by sal desc) =></code></p> <p>applies record number for every row. Highest salary record number is 1. Second Highest Salary record number is 2 and so on.</p>

Queries on Math Functions:

Finding 2 power 3:

SQL> Select power(2,3) from dual;

Output:

POWER(2,3)

8

Finding 5 power 2 and 3 power 3:

SQL> Select Power(5,2), Power(3,3) from dual;

Output:

POWER(5,2) POWER(3,3)

25 27

Finding Square root of 100:

SQL> Select Sqrt(100) from dual;

Output:

SQRT(100)

10

Finding Square root of 81 and square root of 144:

SQL> Select Sqrt(81), Sqrt(144) from dual;

Output:

SQRT(81) SQRT(144)

9 12

Finding 5 mod 2:

SQL> Select Mod(5,2) from dual;

Output:

MOD(5,2)

1**Finding 10 mod 6 and 20 mod 7:**

SQL> Select Mod(10,6), Mod(20,7) from dual;

Output:

MOD(10,6) MOD(20,7)

4

6**Finding sin 90:**

SQL> Select Sin(90*3.14/180) from dual;

Output:

SIN(90*3.14/180)

.999999683**Finding Cos 0 and Cos 45:**

SQL> Select Cos(0*3.14/180), Tan(45*3.14/180) from dual;

Output:

COS(0*3.14/180) TAN(45*3.14/180)

1

.99920399**Finding natural logarithmic value of 7:**

SQL> Select Ln(7) from dual;

Output:

LN(7)

1.94591015

Finding Log 10 base 10:

SQL> Select Log(10,10) from dual;

Output:

LOG(10,10)

1

Examples on Trunc():

1. SQL> Select Trunc(123.4567) from dual;

Output:

TRUNC(123.4567)

123

Removed all decimal places.

2. SQL> Select Trunc(123.4567,2) from dual;

Output:

TRUNC(123.4567,2)

123.45

Printed up to 2 decimal places. Remaining decimal places truncated.

3. SQL> Select Trunc(123.4567,3) from dual;

Output:

TRUNC(123.4567,3)

123.456

Printed up to 3 decimal places. Remaining decimal places truncated.

4. SQL> Select Trunc(123.4567,-1) from dual;

Output:

TRUNC(123.4567,-1)

120

TRUNC(123.4567,-1) => Rounds to 10s.

120.....123.4567.....130

trunc() always gives lower value. So, 120 printed.

5. SQL> Select Trunc(123.4567,-2) from dual;

Output:

TRUNC(123.4567,-2)

100

TRUNC(123.4567,-1) => Rounds to 100s.

100.....123.4567.....200

trunc() always gives lower value. So, 100 printed.

Examples on Round():

1. SQL> Select Round(123.4567) from dual;

Output:

```
ROUND(123.4567)
```

```
123
```

.4 is there. Below .5 . So, printed below value 123

2. SQL> Select Round(123.6567) from dual;

Output:

```
ROUND(123.6567)
```

```
124
```

.6 is there. Above .5 . So, printed above value 124

3. SQL> Select Round(123.5567) from dual;

Output:

```
ROUND(123.5567)
```

```
124
```

.5 is there. So, printed above value 124

4. SQL> Select Round(123.5567,2) from dual;

Output:

ROUND(123.5567,2)

123.56

After second decimal place 6 is there. It is above .5.
So rounded to 123.56.

5. SQL> Select Round(123.5547,2) from dual;

Output:

ROUND(123.5547,2)

123.55

After second decimal place 4 is there. It is below .5.
So rounded to 123.55.

6. SQL> Select Round(123.5557,2) from dual;

Output:

ROUND(123.5557,2)

123.56

After second decimal place 5 is there. It is equals to .5.
So rounded to 123.56.

7. SQL> Select Round(123.4567,-1) from dual;

Output:

ROUND(123.4567,-1)

120

-1 means, Rounds to 10s
120 123.4567 130
3 is there in 1's position. So, Rounds to 120

8. SQL> Select Round(126.6567,-1) from dual;

Output:

ROUND(126.6567,-1)

130

-1 means, rounds to 10s

120.....126.6567.....130
6 is there at 1's position. >5. So, rounds to 130

9. SQL> Select Round(125.6567,-1) from dual;

Output:

ROUND(125.6567,-1)

130

120.....125.6567.....130
5 is there 1's position. Eqauls to 5. So rounds to 130

10. SQL> Select Round(124.6567,-1) from dual;

Output:

ROUND(124.6567,-1)

120

120.....124.6567.....130
4 is there 1's position. So, rounds to 120

11. SQL> Select Round(123.6567,-2) from dual;

Output:

ROUND(123.6567,-2)

100

-2 means, Rounds in 100s

100.....123.6567.....200

2 is there at 2's position. Less than 5. So, rounds to 100

12. SQL> Select Round(152.6567,-2) from dual;

Output:

ROUND(152.6567,-2)

200

-2 means, Rounds in 100s

100.....152.6567.....200

5 is there at 2's position. Equals to 5. So, rounds to 200

13. SQL> Select Round(150.6567,-2) from dual;

Output:

ROUND(150.6567,-2)

200

-2 means, Rounds in 100s

100.....150.6567.....200

5 is there at 2's position. Equals to 5. So, rounds to 200

14. SQL> Select Round(149.6567,-2) from dual;

Output:

ROUND(149.6567,-2)

100

-2 means, Rounds in 100s

100.....149.6567.....200

4 is there at 2's position. Less Than to 5. So, rounds to 100

Example on Ceil():

SQL> Select Ceil(123.456) from dual;

Output:

CEIL(123.456)

124

Ceil() always gives upper integer value. So, printed 124

Example on Floor():

SQL> Select Floor(123.456) from dual;

Output:

FLOOR(123.456)

123

Floor() function always gives lower integer value. So, Printed 123

Example on Trunc(), Round(), Floor() & Ceil Functions:

Create a table as following:

SQL> Select * from player;

PID	FNAME	LNAME	NO_OF_MATCHES	TOTAL_RUNS
1001	sachin	tendulkar	300	10000
1002	virat	kohli	150	7000
1003	rohit	sharma	130	6800
1004	rahul	dravid	200	9000

SQL> Select Total_Runs/No_Of_Matches as "Avrg Score" from player;

Output:

Avrg Score

33.3333333
46.6666667
52.3076923
45

SQL> Select Round(Total_Runs/No_Of_Matches) as "Avrg Score" from player;

Output:

Avrg Score

33
47
52
45

SQL> Select Round(Total_Runs/No_Of_Matches,2) as "Avrg Score" from player;

Output:

Avrg Score

33.33
46.67
52.31
45

SQL> Select Trunc(Total_Runs/No_Of_Matches) as "Avrg Score" from player;

Output:

Avrg Score

33
46
52
45

SQL> Select Trunc(Total_Runs/No_Of_Matches,2) as "Avrg Score" from player;

Output:

Avrg Score

33.33
46.66
52.3
45

SQL> Select Trunc(Total_Runs/No_Of_Matches,3) as "Avrg Score" from player;

Output:

Avrg Score

33.333
46.666
52.307
45

SQL> Select Floor(Total_Runs/No_Of_Matches) as "Avrg Score" from player;

Output:

Avrg Score

33
46
52
45

SQL> Select Ceil(Total_Runs/No_Of_Matches) as "Avrg Score" from player;

Output:

Avrg Score

34
47
53
45

Queries on Aggregate Functions:

Finding Sum of Salaries of Employees:

```
SQL> Select sum(Sal) as "Sum of Salaries" from emp;
```

Sum of Salaries

```
-----  
29025
```

Finding Average of Salaries of Employees:

```
SQL> Select Avg(Sal) as "Average Salary" from emp;
```

Average Salary

```
-----  
2073.21429
```

Finding Average salary by truncating up to 2 decimal places:

```
SQL> Select Trunc(Avg(Sal),2) as "Average Salary" from emp;
```

Average Salary

```
-----  
2073.21
```

Finding average salary. Take Ceil value as Average Salary:

```
SQL> Select Ceil(Avg(Sal)) as "Average Salary" from emp;
```

Average Salary

```
-----  
2074
```

Finding average salary. Take Floor value as Average Salary:

SQL> Select Floor(Avg(Sal)) as "Average Salary" from emp;

Average Salary

2073

Finding average salary. Take Rounded value as Average Salary:

SQL> Select Round(Avg(Sal)) as "Average Salary" from emp;

Average Salary

2073

Finding average salary. Take Rounded value upto 2 decimal places as Average Salary:

SQL> Select Round(Avg(Sal),2) as "Average Salary" from emp;

Average Salary

2073.21

Finding Minimum Salary:

SQL> Select Min(Sal) as "Min Salary" from emp;

Min Salary

800

Finding Maximum Salary:

SQL> Select Max(Sal) as "Max Salary" from emp;

Max Salary

5000

Counting Comm Column values:

SQL> Select Count(comm) from emp;

COUNT(COMM)

3

Note:

Count(Comm) cannot count nulls.

Counting empno column values:

SQL> Select Count(empno) from emp;

COUNT(EMPNO)

14

Counting Number of Records:

SQL> Select Count(*) from emp;

COUNT(*)

14

Finding Employee name who is getting maximum salary:

```
SQL> select ename from emp  
      where sal=(Select max(Sal) from emp);
```

Output:

ENAME

KING

Finding Employee name who is getting minimum salary:

```
SQL> select ename from emp  
      where sal=(Select min(Sal) from emp);
```

Output:

ENAME

SMITH

Finding Second Maximum Salary:

```
SQL> select max(Sal) from emp  
      where sal<(Select max(sal) from emp);
```

Output:

MAX(SAL)

3000

Finding employee name who is getting second maximum salary:

```
SQL> select ename from emp  
      where sal = (select max(Sal) from emp where sal<(Select max(sal)  
from emp));
```

Output:

ENAME

SCOTT

FORD

Counting Number of Clerks in Emp Table:

```
SQL> Select count(*) as "Number Of Clerks" from emp  
      where job='CLERK';
```

Output:

Number Of Clerks

4

Counting Number of Managers in Emp Table:

```
SQL> Select count(*) as "Number Of Managers" from emp  
      where job='MANAGER';
```

Output:

Number Of Managers

3

Finding Number of Employees in Deptno 20:

```
SQL> Select count(*) as "Number Of Employees" from emp  
      where deptno=20;
```

Output:

Number Of Employees

5

Queries on String Functions:

Using Upper() Function:

```
SQL> Select Upper('raju') from dual;
```

Output:

UPPE

RAJU

Using Lower() Function:

```
SQL> Select Lower('RAJU') from dual;
```

Output:

LOWE

raju

Using Initcap() Function:

```
SQL> Select Initcap('RAJ KUMAR') from dual;
```

Output:

INITCAP('

Raj Kumar

Using Concat() Function:

```
SQL> Select Concat('raj','kumar') from dual;
```

Output:

CONCAT('

rajkumar

Getting space between two names:

```
SQL> Select concat(concat('raj', ' '), 'kumar') from dual;
```

Output:

```
CONCAT(CO
```

```
-----
```

```
raj kumar
```

(Or)**--using || [Concatenation Operator]**

```
SQL> Select 'raj' || ' ' || 'kumar' from dual;
```

Output:

```
'RAJ'||"
```

```
-----
```

```
raj kumar
```

Getting space between two names and display initial letters as capital in name:

```
SQL> Select Initcap(concat(concat('raj', ' '), 'kumar')) from dual;
```

Output:

```
INITCAP(C
```

```
-----
```

```
Raj Kumar
```

Displaying Employee Names in Upper Case:

```
SQL> Select Upper(Ename) from emp;
```

Output:

```
UPPER(ENAM
```

```
-----
```

```
SMITH
```

```
ALLEN
```

```
.
```

```
.
```

Displaying Employee Names in Lower Case:

```
SQL> Select Lower(Ename) from Emp;
```

Output:

```
LOWER(ENAM
```

```
-----  
smith
```

```
allen
```

```
.
```

```
.
```

Displaying Employee Names initial letter as capital:

```
SQL> Select Initcap(Ename) from emp;
```

Output:

```
INITCAP(EN
```

```
-----  
Smith
```

```
Allen
```

```
.
```

```
.
```

Create a Table as following:

```
SQL> Select * from player;
```

PID	FNAME	LNAME
1001	sachin	tendulkar
1002	virat	kohli
1003	rohit	sharma
1004	rahul	dravid

Concatenating First Name and Last Name and getting name's initial letters as capital:

SQL> Select Initcap(Concat(Concat(fname,' '),lname)) as Name from player;

Output:

NAME

Sachin Tendulkar
Virat Kohli
Rohit Sharma
Rahul Dravid

Displaying First name in Upper Case:

SQL> Select upper(Fname) from player;

Output:

UPPER(FNAM

SACHIN
VIRAT
ROHIT
RAHUL

Displaying First name in Lower Case:

SQL> Select lower(Fname) from player;

Output:

LOWER(FNAM

sachin
virat
rohit
rahul

Add a Column “PName” and store player name in “PName” column by concatenating First Name and Last Name and drop fname and lname columns:

Adding PName field:

```
SQL> Alter Table Player add PName Varchar2(30);
```

Output:

Table altered.

Updating Player Names by concatenating fname and lname:

```
SQL> Update Player
```

```
Set PName = Initcap(Concat(Concat(FName,' '),LName));
```

Output:

4 rows updated.

Dropping Fname & Lname columns:

```
SQL> Alter Table Player Drop(Fname, Lname);
```

Output:

Table altered.

Retrieving player table records:

```
SQL> Select PID, PNAME from player;
```

Output:

PID	PNAME
1001	Sachin Tendulkar
1002	Virat Kohli
1003	Rohit Sharma
1004	Rahul Dravid

Finding String Length:

```
SQL> Select Length('raju') from dual;
```

Output:

```
LENGTH('RAJU')
```

```
-----  
4
```

Finding string length of each employee name:

```
SQL> Select ename, Length(ename) "Ename Length" from emp;
```

Output:

```
ENAME    Ename Length
```

```
-----  
SMITH      5  
ALLEN      5  
WARD       4
```

```
...
```

Displaying employee records whose name has 4 characters:

```
SQL> Select ename from emp where ename like '____';
```

Output:

```
ENAME
```

```
-----  
WARD  
KING  
FORD
```

[Or]

```
SQL> Select ename from emp where Length(Ename) = 4;
```

Output:

```
ENAME
```

```
-----  
WARD  
KING  
FORD
```

Retrieving Player Table Records:

SQL> Select PID,PName from Player;

PID PNAME

1001 Sachin Tendulkar
1002 Virat Kohli
1003 Rohit Sharma
1004 Rahul Dravid

Displaying player names whose names are having 12 chars:

SQL> Select pname from player where length(pname)=12;

PNAME

Rohit Sharma
Rahul Dravid

Displaying player names whose names are having more than 12 chars:

SQL> Select pname from player where length(pname)>12;

PNAME

Sachin Tendulkar

Examples on Substr() Function:

SQL> Select substr('raj kumar',5) from dual;

Output:

SUBST

kumar

SQL> Select substr('raj kumar',1,3) from dual;

Output:

SUB

raj

SQL> Select substr('raj kumar',1,5) from dual;

Output:

SUBST

raj k

SQL> Select substr('raj kumar',6,3) from dual;

Output:

SUB

uma

SQL> Select substr('raj kumar',6) from dual;

Output:

SUBS

umar

SQL> Select substr('raj kumar',-4) from dual;

Output:

SUBS

umar

SQL> Select substr('raj kumar',-4,3) from dual;

Output:

SUB

uma

SQL> Select substr('raj kumar',-5) from dual;

Output:

SUBST

kumar

SQL> Select substr('raj kumar',-5,3) from dual;

Output:

SUB

kum

Note:

-ve position => Position Number From Right Side

+ve Position => Position Number From Left Side

Generating Email IDs by taking emp name's first 3 chars and empno's last 3 digits:

```
SQL> Select  
substr(Ename,1,3) || substr(empno,-3,3) || '@nareshit.com'  
As "E-Mail ID" from emp;
```

Output:

E-Mail ID

```
-----  
SMI369@nareshit.com  
ALL499@nareshit.com  
WAR521@nareshit.com
```

...

...

Displaying employee names whose name started with 'S':

```
SQL> Select EName from emp where Ename Like 'S%';
```

Output:

ENAME

SMITH

SCOTT

(Or)

```
SQL> Select EName from emp where substr(Ename,1,1) = 'S';
```

Output:

ENAME

SMITH

SCOTT

Retrieving the emp record whose name is ‘blake’ and when we don’t know whether the name is in upper case or lower case:

SQL> Select ename, sal from emp where lower(ename) = 'blake';

Output:

ENAME	SAL
-----	-----
BLAKE	2850

Displaying Employee names whose names are ended with ‘RD’:

SQL> Select ename from emp where substr(ename,-2,2) = 'RD';

Output:

ENAME

WARD
FORD

(Or)

SQL> Select ename from emp where ename like '%RD';

Output:

ENAME

WARD
FORD

Displaying employee names whose names are started and ended with same letter:

SQL> Insert into emp(empno,ename) values(1001,'DAVID');

Output:

1 row created.

```
SQL> Insert into emp(empno,ename) values(1002,'SRINIVAS');
```

Output:

```
1 row created.
```

```
SQL> Select ename from emp  
      where substr(ename,1,1) = substr(ename,-1,1);
```

Output:

```
ENAME  
-----  
DAVID  
SRINIVAS
```

Examples on Instr() Function:

```
SQL> Select Instr('This is his wish','is') from dual;
```

Output:

```
INSTR('THISISHISWISH','IS')  
-----  
3
```

```
SQL> Select Instr('This is his wish','is',4) from dual;
```

Output:

```
INSTR('THISISHISWISH','IS',4)  
-----  
6
```

```
SQL> Select Instr('This is his wish','is',7) from dual;
```

Output:

```
INSTR('THISISHISWISH','IS',7)  
-----  
10
```

```
SQL> Select Instr('This is his wish','is',11) from dual;
```

Output:

```
INSTR('THISISHISWISH','IS',11)
```

```
-----  
14
```

```
SQL> Select Instr('This is his wish','is',15) from dual;
```

Output:

```
INSTR('THISISHISWISH','IS',15)
```

```
-----  
0
```

Note:

If substring is not found in the string, then it returns 0

```
SQL> Select Instr('This is his wish','is',-1) from dual;
```

Output:

```
INSTR('THISISHISWISH','IS',-1)
```

```
-----  
14
```

Note:

Positive Position => Starts searching from left side

Negative Position => Starts searching from right side

```
SQL> Select Instr('This is his wish','is',-4) from dual;
```

Output:

```
INSTR('THISISHISWISH','IS',-4)
```

```
-----  
10
```

```
SQL> Select Instr('This is his wish','is',-4,2) from dual;
```

Output:

```
INSTR('THISISHISWISH','IS',-4,2)
```

6

Displaying Employee Names whose names are containg 'AM' characters:

```
SQL> Select ename from emp where ename like '%AM%';
```

Output:

```
ENAME
```

ADAMS

JAMES

(or)

```
SQL> Select ename from emp where instr(ename,'AM')>0;
```

Output:

```
ENAME
```

ADAMS

JAMES

Examples on LPad() and RPad Functions:

```
SQL> Select Lpad('raju',10,'*') from dual;
```

Output:

```
LPAD('RAJU
```

*****raju

```
SQL> Select Lpad('raju',15,'@#') from dual;
```

Output:

```
LPAD('RAJU',15,  
-----  
@#@#@#@#@#@#@raju
```

```
SQL> Select Rpad('raju',10,'*') from dual;
```

Output:

```
RPAD('RAJU  
-----  
raju*****
```

```
SQL> Select Rpad('raju',15,'@#') from dual;
```

Output:

```
RPAD('RAJU',15,  
-----  
raju@#@#@#@#@#@#@
```

```
SQL> Select Lpad('raju',10) from dual;
```

Output:

```
LPAD('RAJU  
-----  
raju
```

```
SQL> Select Rpad('raj',10) || 'kumar' from dual;
```

Output:

```
RPAD('RAJ',10)|  
-----  
raj      kumar
```

```
SQL> Select Lpad('*',10,'*') from dual;
```

Output:

```
LPAD('*',1
```

```
-----
```

```
*****
```

```
SQL> Select 'amount credited from the acno ' || LPad('X',6,'X') ||  
Substr('123456789',-4,4) Message from dual;
```

```
MESSAGE
```

```
-----
```

```
amount credited from the acno XXXXXX6789
```

Examples on LTrim(), RTrim() and Trim() Functions:

```
SQL> Select LTrim('    raj    ') || 'kumar' from dual;
```

Output:

```
LTRIM('RAJ')
```

```
-----
```

```
raj kumar
```

```
SQL> Select RTrim('    raj    ') || 'kumar' from dual;
```

Output:

```
RTRIM('RAJ')|
```

```
-----
```

```
rajkumar
```

```
SQL> Select Trim('    raj    ') || 'kumar' from dual;
```

Output:

```
TRIM('RA
```

```
-----  
rajkumar
```

```
SQL> Select LTrim('@@@@@@raju@@@') from dual;
```

Output:

```
LTRIM('@@@@R
```

```
-----  
@@@@@@raju@@@
```

```
SQL> Select LTrim('@@@@@@raju@@@','@') from dual;
```

Output:

```
LTRIM('@
```

```
-----  
raju@@@
```

```
SQL> Select RTrim('@@@@@@raju@@@','@') from dual;
```

Output:

```
RTRIM('@@
```

```
-----  
@@@raju
```

```
SQL> Select LTrim('#@#####@@@raju#@#####@@@#@##','@#')  
from dual;
```

Output:

```
LTRIM('@#@#####@@@R
```

```
-----  
raju#@#@#####@@@#@##
```

```
SQL> Select RTrim('@#@#####@@@raju@#@#####@@@#@##','@#')  
from dual;
```

Output:

```
RTRIM('@#@#####  
-----  
@#@#####@@@raju
```

```
SQL> Select Trim(Leading '@' from '@@@@raju@@@@') from dual;
```

Output:

```
TRIM(LEAD  
-----  
raju@@@@@
```

```
SQL> Select Trim(Trailing '@' from '@@@@raju@@@@') from dual;
```

Output:

```
TRIM(TRA  
-----  
@@@@raju
```

```
SQL> Select Trim(Both '@' from '@@@@raju@@@@') from dual;
```

Output:

```
TRIM  
----  
raju
```

```
SQL> Select LTrim('raju@nareshit.com','@gmail.com') from dual;
```

Output:

```
LTRIM('RAJU@NARES  
-----  
raju@nareshit.com
```

```
SQL> Select RTrim('raju@nareshit.com','@gmail.com') from dual;
```

Output:

```
RTRIM('RAJU@N
```

```
-----  
raju@nareshit
```

```
SQL> Select RTrim('raju@nareshit.com','@nareshit.com') from dual;
```

Output:

```
RTRI
```

```
-----  
raju
```

```
SQL> Select LTrim('OPPO F11 Pro','OPPO') from dual;
```

Output:

```
LTRIM('O
```

```
-----  
F11 Pro
```

Examples on Replace and Translate() Functions:

```
SQL> Select Replace('sai krishna','sai','rama') from dual;
```

Output:

```
REPLACE('SAI
```

```
-----  
rama krishna
```

```
SQL> Select Replace('xyzxyzxxxxyzxyzxyz','xyz','abc') from dual;
```

Output:

```
REPLACE('XYZXYZXXYYZZ
```

```
-----  
abcabccxyyzzabcxyyzz
```

```
SQL> Select Translate('xyzxyzxyzxyzxyzxyz','xyz','abc') from dual;
```

Output:

```
TRANSLATE('XYZXYZXXYY
```

```
-----  
abcabcaabbccabcaabbcc
```

Displaying salary value with mask characters:

```
SQL> Column salary format A10
```

```
SQL> Select ename, Translate(sal,'0123456789','!@#$%^&*+=') Salary  
from emp;
```

Output:

```
ENAME    SALARY
```

```
-----  
SMITH    +!!
```

```
ALLEN    @&!!
```

```
WARD     @#^!
```

```
..
```

Examples on ASCII and Chr() Functions:

```
SQL> Select ASCII('A') from dual;
```

Output:

```
ASCII('A')
```

```
65
```

```
SQL> Select ASCII('a'), ASCII('z') from dual;
```

Output:

```
ASCII('A') ASCII('Z')
```

```
97      122
```

SQL> Select Chr(65) from dual;

Output:

C

-

A

SQL> Select Chr(97), Chr(122) from dual;

Output:

C C

--

a z

Examples on Soundex() Function:

SQL> Select empno,ename from emp where Soundex(ename) = Soundex('SMYT');

Output:

EMPNO ENAME

7369 SMITH

SQL> Select empno,ename from emp where Soundex(ename) = Soundex('SKAT');

Output:

EMPNO ENAME

7788 SCOTT

Examples on Reverse() Function:

SQL> Select Reverse('ramu') from dual;

Output:

REVE

umar

SQL> Select EName, Reverse(Ename) as "Rev Name" from emp;

Output:

ENAME	Rev Name
-------	----------

SMITH	HTIMS
ALLEN	NELLA
WARD	DRAW
JONES	SENOJ

.

.

Example Queries on Conversion Functions:**Converting Date To String:****Getting system date in different formats:****Getting year four digits from today's date:**

SQL> Select to_char(sysdate,'yyyy') from dual;

Output:

TO_C

2021

Getting year two digits from today's date:

```
SQL> Select to_char(sysdate,'yy') from dual;
```

Output:

```
TO  
--  
21
```

Getting year one digit from today's date:

```
SQL> Select to_char(sysdate,'y') from dual;
```

Output:

```
T  
-  
1
```

Getting year three digits from today's date:

```
SQL> Select to_char(sysdate,'yyy') from dual;
```

```
TO_  
---  
021
```

Getting year in words from today's date:

```
SQL> Select to_char(sysdate,'year') from dual;
```

Output:

```
TO_CHAR(SYSDATE,'YEAR')  
-----  
twenty twenty-one
```

Getting month two digits from today's date:

```
SQL> Select to_char(sysdate,'mm') from dual;
```

Output:

```
TO
--
07
```

Getting short month name from today's date:

```
SQL> Select to_char(sysdate,'mon') from dual;
```

Output:

```
TO_CHAR(SYSD
-----
jul
```

Getting full month name from today's date:

```
SQL> Select to_char(sysdate,'month') from dual;
```

Output:

```
TO_CHAR(SYSDATE,'MONTH')
-----
july
```

Getting short month name in upper case:

```
SQL> Select to_char(sysdate,'MON') from dual;
```

Output:

```
TO_CHAR(SYSD
-----
JUL
```

Getting full month name in upper case:

```
SQL> Select to_char(sysdate,'MONTH') from dual;
```

Output:

```
TO_CHAR(SYSDATE,'MONTH')
```

```
JULY
```

Getting date two digits [day number in month] from today's date:

```
SQL> Select to_char(sysdate,'dd') from dual;
```

Output:

```
TO
```

```
--
```

```
15
```

Getting day number in the year of today's date:

```
SQL> Select to_char(sysdate,'ddd') from dual;
```

Output:

```
TO_
```

```
--
```

```
196
```

Getting day number in week of today's date:

```
SQL> Select to_char(sysdate,'d') from dual;
```

Output:

```
T
```

```
-
```

```
5
```

Getting short weekday name of today's date:

```
SQL> Select to_char(sysdate,'dy') from dual;
```

Output:

```
TO_CHAR(SYSD
```

```
-----  
thu
```

Getting full weekday name of today's date:

```
SQL> Select to_char(sysdate,'day') from dual;
```

Output:

```
TO_CHAR(SYSDATE,'DAY')
```

```
-----  
thursday
```

Getting short weekday name in upper case:

```
SQL> Select to_char(sysdate,'DY') from dual;
```

Output:

```
TO_CHAR(SYSD
```

```
-----  
THU
```

Getting weekday full name of today's date:

```
SQL> Select to_char(sysdate,'DAY') from dual;
```

Output:

```
TO_CHAR(SYSDATE,'DAY')
```

```
-----  
THURSDAY
```

Getting time from today's date:

```
SQL> Select to_char(sysdate,'hh:mi AM') from dual;
```

Output:

```
TO_CHAR(
```

```
-----
```

```
11:41 AM
```

Getting quarter number of today's date:

```
SQL> Select to_char(sysdate,'Q') from dual;
```

Output:

```
T
```

```
-
```

```
3
```

Getting week number of today's date in the month:

```
SQL> Select to_char(sysdate,'W') from dual;
```

Output:

```
T
```

```
-
```

```
3
```

Getting weekday number of today's date in the year:

```
SQL> Select to_char(sysdate,'WW') from dual;
```

Output:

```
TO
```

```
--
```

```
28
```

Getting Current Century number:

```
SQL> Select to_char(sysdate,'CC') from dual;
```

TO

--

21

Getting Current Century Number and AD Year or BC Year :

```
SQL> Select to_char(sysdate,'CC AD') from dual;
```

Output:

TO_CH

21 AD

Example Queries on to_Char Function:**Displaying employee records who joined in 1982:**

```
SQL> Select ename,hiredate from emp where hiredate like '%82';
```

Output:

ENAME	HIREDATE
-------	----------

----- -----

SCOTT 09-DEC-82

MILLER 23-JAN-82

[Or]

```
SQL> Select ename,hiredate from emp where  
to_char(hiredate,'yyyy')=1982;
```

Output:

ENAME	HIREDATE
-------	----------

----- -----

SCOTT 09-DEC-82

MILLER 23-JAN-82

Displaying employee records who joined in 1980, 1982,1983:

SQL> Select ename,hiredate from emp where to_char(hiredate,'yyyy') in (1980,1982,1983);

Output:

ENAME	HIREDATE
SMITH	17-DEC-80
SCOTT	09-DEC-82
ADAMS	12-JAN-83
MILLER	23-JAN-82

Displaying employee records who joined on Sunday:

SQL> Select ename, hiredate from emp where to_char(hiredate,'d') =1;

Output:

ENAME	HIREDATE
WARD	22-FEB-81

[Or]

SQL> Select ename, hiredate from emp where to_char(hiredate,'dy')='sun';

Output:

ENAME	HIREDATE
WARD	22-FEB-81

[Or]

SQL> Select ename, hiredate from emp where rtrim(to_char(hiredate,'day'))='sunday';

Output:

ENAME	HIREDATE
WARD	22-FEB-81

Displaying employee records who joined in second quarter:

```
SQL> Select ename, hiredate from emp where to_char(hiredate,'q') = 2;
```

Output:

ENAME	HIREDATE
JONES	02-APR-81
BLAKE	01-MAY-81
CLARK	09-JUN-81

Displaying employee records who joined in Jan, Apr and Dec:

```
SQL> Select ename, hiredate from emp where to_char(hiredate,'mm') in  
(1,4,12);
```

ENAME	HIREDATE
SMITH	17-DEC-80
JONES	02-APR-81
SCOTT	09-DEC-82
ADAMS	12-JAN-83
JAMES	03-DEC-81
FORD	03-DEC-81
MILLER	23-JAN-82

Displaying hiredate in mm/dd/yyyy format:

```
SQL> Select ename, to_Char(hiredate,'mm/dd/yyyy') hiredate from emp;
```

Output:

ENAME	HIREDATE
SMITH	12/17/1980
ALLEN	02/20/1981
.	
.	

Displaying hiredate in dd/mm/yyyy format:

```
SQL> Select ename, to_Char(hiredate,'dd/mm/yyyy') hiredate from emp;
```

Output:

```
ENAME    HIREDATE
```

```
-----  
SMITH   17/12/1980  
ALLEN   20/02/1981
```

Converting String to Date:**Converting string to date:**

```
SQL> Select to_date('20 december 2021') from dual;
```

Output:

```
TO_DATE('
```

```
-----  
20-DEC-21
```

Converting string to date:

```
SQL> Select to_date('2-dec-2021') from dual;
```

Output:

```
TO_DATE('
```

```
-----  
02-DEC-21
```

Converting string to date:

```
SQL> Select to_date('23/12/2020','dd/mm/yyyy') from dual;
```

Output:

```
TO_DATE('
```

```
-----  
23-DEC-20
```

Getting year part from a date:

```
SQL> Select to_char(to_date('23-dec-2020'),'yyyy') from dual;
```

Output:

TO_C

2020

Getting month name from a date:

```
SQL> Select to_char(to_date('23-dec-2020'),'mon') from dual;
```

Output:

TO_

dec

Getting weekday name of a date:

```
SQL> Select to_char(to_date('23-dec-2020'),'day') from dual;
```

Output:

TO_CHAR(T

wednesday

Adding 10 days to sysdate:

```
SQL> Select sysdate+10 from dual;
```

Output:

SYSDATE+1

25-JUL-21

Adding 10 days to a Date:

```
SQL> Select to_date('16-jun-2021')+10 from dual;
```

Output:

```
TO_DATE('  
-----  
26-JUN-21
```

Displaying weekday name of INDIA's Independence Day:

```
SQL> Select to_char(to_date('15-AUG-1947'),'day') from dual;
```

Output:

```
TO_CHAR(T  
-----  
friday
```

Displaying weekday name of SACHIN's Birthday:

```
SQL> Select to_char(to_date('24-APR-1973'),'day') from dual;
```

Output:

```
TO_CHAR(T  
-----  
tuesday
```

Inserting date value in emp table:

```
SQL> Insert into emp(empno, ename, hiredate)  
values(1234,'ccc',to_date('16-nov-2020'));
```

Output:

```
1 row created.
```

Converting Number to String:

Displaying employee salary with currency symbol \$. Also display 2 decimal places:

SQL> Select to_char(5000,'L9999.99') from dual;

Output:

TO_CHAR(5000,'L999

\$5000.00

Displaying currency name:

SQL> Select to_char(5000,'C9999.99') from dual;

Output:

TO_CHAR(5000,'C

USD5000.00

Displaying NLS Parameters [NLS = National Language Support]:

SQL> Select * from v\$NLS_PARAMETERS;

Output:

PARAMETER	VALUE
NLS_LANGUAGE	AMERICAN
NLS_TERRITORY	AMERICA
NLS_CURRENCY	\$
.	

Changing currency and country:

SQL> Alter session set nls_territory='INDIA';

Output:

Session altered.

```
SQL> Alter session set nls_currency='RS';
```

Output:

```
Session altered.
```

```
SQL> Select to_char(5000,'L9999.99') from dual;
```

Output:

```
TO_CHAR(5000,'L999
```

```
-----  
RS5000.00
```

```
SQL> Select to_char(5000,'C9999.99') from dual;
```

Output:

```
TO_CHAR(5000,'C
```

```
-----  
INR5000.00
```

Displaying employee salary with currency symbol & decimal places:

```
SQL> Select ename, to_char(Sal,'L99999.99') Salary from emp;
```

Output:

```
ENAME    SALARY
```

```
-----  
SMITH      RS800.00  
ALLEN     RS1600.00  
WARD      RS1250.00
```

```
.  
. .  
.
```

Displaying employee salary with currency name:

```
SQL> Select ename, to_char(Sal,'C99999.99') Salary from emp;
```

Output:

ENAME	SALARY
CCC	
SMITH	INR800.00
ALLEN	INR1600.00
WARD	INR1250.00
.	
.	

Getting zeros in front of remaining digits places:

```
SQL> Select to_char(1234,'000000') from dual;
```

TO_CHAR

001234

Converting String to Number:

```
SQL> Select to_number('123') from dual;
```

Output:

TO_NUMBER('123')

123

```
SQL> Select to_number('$5000.00','L9999.99') from dual;
```

Output:

TO_NUMBER('\$5000.00','L9999.99')

5000

```
SQL> Select to_number('USD5000.00','C9999.99') from dual;
```

```
TO_NUMBER('USD5000.00','C9999.99')
```

```
-----  
      5000
```

Queries on Date Functions:

Displaying current system date:

```
SQL> Select sysdate from dual;
```

Output:

```
SYSDATE
```

```
-----  
16-JUL-21
```

Displaying current system time:

```
SQL> Select systimestamp from dual;
```

Output:

```
SYSTIMESTAMP
```

```
-----  
16-JUL-21 04.20.34.223000 PM +05:30
```

Displaying time from sysdate:

```
SQL> Select to_char(sysdate,'hh:mi AM') from dual;
```

Output:

```
TO_CHAR(
```

```
-----  
04:21 PM
```

Displaying time in 24Hours Format from sysdate:

```
SQL> Select to_char(sysdate,'hh24:mi') from dual;
```

Output:

```
TO_CH
```

```
-----
```

```
16:21
```

Displaying date from systimestamp:

```
SQL> Select to_char(systimestamp,'dd/mm/yyyy') from dual;
```

Output:

```
TO_CHAR(SY
```

```
-----
```

```
16/07/2021
```

Displaying time from systiestamp:

```
SQL> Select to_char(systimestamp,'hh:mi AM') from dual;
```

Output:

```
TO_CHAR(
```

```
-----
```

```
04:22 PM
```

Displaying time in 24 Hours Format:

```
SQL> Select to_char(systimestamp,'hh24:mi') from dual;
```

Output:

```
TO_CH
```

```
-----
```

```
16:22
```

Adding 2 months to today's date:

```
SQL> Select Add_Months(sysdate,2) from dual;
```

Output:

```
ADD_MONTH
```

```
-----
```

```
16-SEP-21
```

Subtracting 2 months from today's date:

```
SQL> Select Add_Months(sysdate,-2) from dual;
```

Output:

```
ADD_MONTH
```

```
-----
```

```
16-MAY-21
```

Adding two months to a date:

```
SQL> Select Add_Months(to_Date('20-oct-2020'),2) from dual;
```

Output:

```
ADD_MONTH
```

```
-----
```

```
20-DEC-20
```

Subtracting two months to a date:

```
SQL> Select Add_Months(to_Date('20-oct-2020'),-2) from dual;
```

Output:

```
ADD_MONTH
```

```
-----
```

```
20-AUG-20
```

Inserting a record into emp table with hiredate as sysdate:

```
SQL> Insert into emp(empno,ename,job,sal,hiredate)
   values(2001,'Krishna','MANAGER',15000,sysdate);
```

Output:

1 row created.

Displaying employee records who joined today:

```
SQL> Select empno, ename, hiredate from emp where
      to_char(Hiredate,'dd/mm/yyyy') = to_char(Sysdate,'dd/mm/yyyy');
```

Output:

EMPNO	ENAME	HIREDATE
-----	-----	-----
2001	Krishna	16-JUL-21

(Or)

```
SQL> Select empno, ename, hiredate from emp
      where trunc(Hiredate) = trunc(Sysdate);
```

Output:

EMPNO	ENAME	HIREDATE
-----	-----	-----
2001	Krishna	16-JUL-21

Inserting a record with yesterday's date as hiredate:

```
SQL> Insert into emp(empno,ename,job,sal,hiredate)
   values(2002,'Ganesh','MANAGER',15000,sysdate-1);
```

Output:

1 row created.

Displaying emp records who joined yesterday:

```
SQL> Select empno, ename, hiredate from emp where  
to_char(Hiredate,'dd/mm/yyyy') = to_char(Sysdate-1,'dd/mm/yyyy');
```

Output:

EMPNO	ENAME	HIREDATE
-----	-----	-----
2002	Ganesh	15-JUL-21

(Or)

```
SQL> Select empno, ename, hiredate from emp  
2 where trunc(Hiredate) = trunc(Sysdate-1);
```

Output:

EMPNO	ENAME	HIREDATE
-----	-----	-----
2002	Ganesh	15-JUL-21

Inserting a record with one month ago's date:

```
SQL> Insert into emp(empno,ename,job,sal,hiredate)  
values(2002,'Ganesh','MANAGER',15000,Add_Months(sysdate,-1));
```

Output:

1 row created.

Displaying emp records who joined one month ago on same date:

```
SQL> Select empno, ename, hiredate from emp  
where to_char(hiredate,'dd/mm/yyyy') =  
to_char(Add_Months(sysdate,-1),'dd/mm/yyyy');
```

Output:

EMPNO	ENAME	HIREDATE
-----	-----	-----
2002	Ganesh	16-JUN-21

(Or)

```
SQL> Select empno, ename, hiredate from emp  
      where trunc(hiredate) = trunc(Add_Months(sysdate,-1));
```

Output:

EMPNO	ENAME	HIREDATE
-----	-----	-----
2002	Ganesh	16-JUN-21

Inserting a record with one year ago's date:

```
SQL> Insert into emp(empno,ename,job,sal,hiredate)  
      values(2003,'Sai','MANAGER',15000,Add_Months(sysdate,-12));
```

Output:

1 row created.

Displaying employee records who joined one year ago on same date:

```
SQL> Select empno, ename, hiredate from emp  
      where to_char(hiredate,'dd/mm/yyyy') =  
            to_char(Add_Months(sysdate,-12),'dd/mm/yyyy');
```

Output:

EMPNO	ENAME	HIREDATE
-----	-----	-----
2003	Sai	16-JUL-20

(Or)

```
SQL> Select empno, ename, hiredate from emp  
      2 where trunc(hiredate) = trunc(Add_Months(sysdate,-12));
```

Output:

EMPNO	ENAME	HIREDATE
-----	-----	-----
2003	Sai	16-JUL-20

Displaying employee record who joined yesterday using 'Interval'**Expression:**

```
SQL> Select ename, hiredate from emp  
      where trunc(hiredate) = trunc(sysdate-interval '1' day);
```

Output:

ENAME	HIREDATE
Ganesh	15-JUL-21

Displaying employee record who joined one month ago on same date**using 'Interval' Expression:**

```
SQL> Select ename, hiredate from emp  
      where trunc(hiredate) = trunc(sysdate-interval '1' month);
```

Output:

ENAME	HIREDATE
Ganesh	16-JUN-21

Displaying employee record who joined one month ago on same date**using 'Interval' Expression:**

```
SQL> Select ename, hiredate from emp  
      where trunc(hiredate) = trunc(sysdate-interval '1' year);
```

Output:

ENAME	HIREDATE
Sai	16-JUL-20

Example on Add_Months & Interval Expressions:**Create table with following Structure:****Sales**

DatId	Amount
-------	--------

Creating Sales Table:

```
SQL> Create table Sales(Dateld Date,  
Amount Number(7,2));
```

Output:

Table created.

Inserting a record with today's date as DatelD:

```
SQL> Insert into Sales values(sysdate,10000);
```

Output:

1 row created.

Inserting a record with yesterday's date as DatelD:

```
SQL> Insert into Sales values(sysdate-1,10000);
```

Output:

1 row created.

Inserting a record with 1 Month ago's date as DatelD:

```
SQL> Insert into Sales values(sysdate-Interval '1' Month,10000);
```

Output:

1 row created.

Inserting a record with 1 Year ago's date as DatelD:

```
SQL> Insert into Sales values(sysdate-Interval '1' year,10000);
```

Output:

1 row created.

Retrieving Sales Table records:

```
SQL> Select * from sales;
```

Output:

DATEID	AMOUNT
-----	-----
16-JUL-21	10000
15-JUL-21	10000
16-JUN-21	10000
16-JUL-20	10000

Displaying today's sales amount:

```
SQL> Select * from Sales where trunc(dateid) = trunc(sysdate);
```

Output:

DATEID	AMOUNT
-----	-----
16-JUL-21	10000

Displaying yesterday's sales amount:

```
SQL> Select * from Sales where trunc(dateid) = trunc(sysdate-1);
```

Output:

DATEID	AMOUNT
-----	-----
15-JUL-21	10000

Displaying one month ago same date Sales Amount:

```
SQL> Select * from Sales where trunc(dateid) = trunc(sysdate-interval '1' month);
```

Output:

DATEID	AMOUNT
-----	-----
16-JUN-21	10000

(Or)

```
SQL> Select * from Sales where trunc(dateid) =  
trunc(Add_Months(sysdate,-12));
```

Output:

DATEID	AMOUNT
-----	-----
16-JUL-20	10000

Displaying one year ago same date Sales Amount:

```
SQL> Select * from Sales  
where trunc(dateid) = trunc(Add_Months(sysdate,-12));
```

Output:

DATEID	AMOUNT
-----	-----
16-JUL-20	10000

(Or)

```
SQL> Select * from Sales where trunc(dateid) = trunc(sysdate-interval '1'  
year);
```

Output:

DATEID	AMOUNT
-----	-----
16-JUL-20	10000

Example Appln on Add_Months():

Create a table with following structure:

Emp

Empno	DOB	Hiredate	Date_of_Retirement
-------	-----	----------	--------------------

Calculate Date of retirement based on DOB [Date of Birth]. 60 years is retirement age.

Creating a Table with the name “emp3”:

```
SQL> Create table emp3(  
    Empno Number(4),  
    DOB Date,  
    Hiredate Date,  
    Retirement_Date Date);
```

Output:

Table created.

Inserting records into “emp3” Table:

```
SQL> Insert into emp3(Empno,DOB,Hiredate) values(1001,'12-Jan-  
1995','01-JUN-2018');
```

Output:

1 row created.

```
SQL> Insert into emp3(Empno,DOB,Hiredate) values(1002,'12-Jan-  
2000','20-OCT-2020');
```

Output:

1 row created.

```
SQL> Insert into emp3(Empno,DOB,Hiredate) values(1003,'25-Dec-  
1990','20-AUG-2015');
```

Output:

1 row created.

Retrieving all records of emp3 Table:

```
SQL> Select * from emp3;
```

EMPNO	DOB	HIREDATE	RETIREMEN
-----	-----	-----	-----
1001	12-JAN-95	01-JUN-18	

1002	12-JAN-00	20-OCT-20
1003	25-DEC-90	20-AUG-15

Calculating Date of Retirement:

```
SQL> update emp3 set Retirement_Date = DOB + Interval '60' Year;
```

Output:

3 rows updated.

(Or)

```
SQL> Update emp3 set Retirement_Date = Add_Months(dob,60*12);
```

Output:

3 rows updated.

```
SQL> Select * from Emp3;
```

Output:

EMPNO	DOB	HIREDATE	RETIREMEN
1001	12-JAN-95	01-JUN-18	12-JAN-55
1002	12-JAN-00	20-OCT-20	12-JAN-60
1003	25-DEC-90	20-AUG-15	25-DEC-50

Example Application on Add_Months():

Create a Table with following structure:

IndiaCMs

State	CM_Name	Term_Start Date	Term_End Date
-------	---------	-----------------	---------------

Calculate CM Term Ending Date:

Creating “IndiaCMs” Table:

```
SQL> Create Table IndiaCMs(
```

```
    State Varchar2(15),
    CM_Name  Varchar2(20),
    Term_Start Date,
    Term_End Date);
```

Output:

Table created.

Inserting records into IndiaCMs Table:

```
SQL> Insert into IndiaCMs(State, CM_Name, Term_Starting_Date)
   values('Telangana','KCR','13-DEC-2018');
```

Output:

1 row created.

```
SQL> Insert into IndiaCMs(State, CM_Name, Term_Starting_Date)
   values('AP','YS JAGAN','30-MAY-2019');
```

Output:

1 row created.

Displaying all records:

```
SQL> Select * from IndiaCMs;
```

Output:

STATE	CM_NAME	TERM_STAR	TERM_ENDI
Telangana	KCR	13-DEC-18	
AP	YS JAGAN	30-MAY-19	

Calculating CM Term Ending Date:

```
SQL> Update IndiaCMs Set Term_Ending_Date =
   Add_Months(Term_Starting_Date,5*12);
```

Output:

2 rows updated.

(Or)

```
SQL> Update IndiaCMs Set Term_Ending_Date =
   Term_Starting_Date+Interval '5' Year;
```

Output:

2 rows updated.

Displaying all records:

SQL> Select * from IndiaCMs;

STATE	CM_NAME	TERM_STAR	TERM_ENDI
Telangana	KCR	13-DEC-18	13-DEC-23
AP	YS JAGAN	30-MAY-19	30-MAY-24

Example Queries on Last_Day():**Finding current month last date:**

SQL> select last_day(sysdate) from dual;

Output:

LAST_DAY(

31-JUL-21

Finding next month first date:

SQL> select last_day(sysdate)+1 from dual;

Output:

LAST_DAY(

01-AUG-21

Finding previous month first date:

SQL> select last_day(add_months(sysdate,-2))+1 from dual;

Output:

LAST_DAY(

01-JUN-21

Finding Previous month last date:

SQL> select last_day(add_months(sysdate,-1)) from dual;

Output:

LAST_DAY(-----

30-JUN-21

Finding current month first date:

SQL> select last_day(add_months(sysdate,-1))+1 from dual;

Output:

LAST_DAY(-----

01-JUL-21

Example Queries on Next_Day():

Finding next Monday date:

SQL> Select next_day(sysdate,'mon') from dual;

Output:

NEXT_DAY(-----

19-JUL-21

Finding this month last Monday date:

SQL> Select next_day(last_day(sysdate),'mon')-7 from dual;

Output:

NEXT_DAY(-----

26-JUL-21

Finding next month first Monday date:

```
SQL> Select next_day(Last_Day(Sysdate)+1,'mon') from dual;
```

Output:

```
NEXT_DAY(
```

```
-----
```

```
02-AUG-21
```

Finding current month first Monday date:

```
SQL> Select next_day(Last_Day(Add_Months(Sysdate,-2)+1),'mon') from dual;
```

Output:

```
NEXT_DAY(
```

```
-----
```

```
07-JUN-21
```

Example queries on Months_Between():**Finding experience of employees:**

```
SQL> Select ename, Round(Months_Between(sysdate,hiredate)/12)  
Experience from emp;
```

Output:

ENAME	EXPERIENCE
-------	------------

```
-----
```

SMITH	41
ALLEN	40
WARD	40
JONES	40
MARTIN	40
BLAKE	40
CLARK	40
SCOTT	39
KING	40

Finding Sachin's Age:

```
SQL> Select Trunc(Months_Between(Sysdate,'24-APR1973')/12) AGE from dual;
```

Output:

```
AGE
```

```
-----  
48
```

Finding age in the form of years and months:

```
SQL> Select Trunc(Months_Between(Sysdate,'24-APR-1973')/12) Years,  
      Mod(Trunc(Months_Between(Sysdate,'24-APR-1973'))),12) Months  
      from dual;
```

Output:

```
YEARS    MONTHS
```

```
-----  
48       2
```

Finding employee experience in the form of years and months:

```
SQL> Select Trunc(Months_Between(sysdate,hiredate)/12) Years,  
      Mod(Trunc(Months_Between(sysdate,hiredate))),12) Months  
      from emp;
```

Output:

```
YEARS    MONTHS
```

```
-----  
40       7  
40       4  
40       4  
40       3  
39       9  
40       2  
40       1  
38       7  
39       8  
39       10
```

Example Queries on Miscellaneous Functions:

Displaying current user name:

SQL> Select User from dual;

Output:

USER

C##ORACLE7AM

Displaying Current User ID:

SQL> Select UID from dual;

Output:

UID

111

Finding greatest in 3 numbers:

SQL> Select Greatest(100,200,300) from dual;

Output:

GREATEST(100,200,300)

300

Finding least in 3 numbers:

SQL> Select Least(100,200,300) from dual;

Output:

LEAST(100,200,300)

100

Finding size of the string:

```
SQL> Select VSize('raju'), VSize('sai') from dual;
```

Output:

```
VSIZE('RAJU') VSIZE('SAI')
```

```
4      3
```

Examples on NVL() Function:**Replacing not null with same value:**

```
SQL> Select NVL(100,200) from dual;
```

Output:

```
NVL(100,200)
```

```
100
```

Replacing null with 200:

```
SQL> Select NVL(NULL,200) from dual;
```

Output:

```
NVL(NULL,200)
```

```
200
```

Calculating null + 200:

```
SQL> Select null+200 from dual;
```

Output:

```
NULL+200
```

Note: null + something = null

Replacing null with 100:

SQL> Select NVL(null,100)+200 from dual;

Output:

NVL(NULL,100)+200

300

Calculating Total Salary of Employees:

SQL> Select ename,sal,comm,sal+NVL(comm,0) as "Total Salary"
from emp;

Output:

ENAME	SAL	COMM	Total Salary
SMITH	800		800
ALLEN	1600	300	1900
WARD	1250	500	1750
JONES	2975		2975
MARTIN	1250	1400	2650

Replacing null value with N/A [Not Applicable]:

SQL> Select ename, sal, NVL(to_char(comm),'N/A') comm from emp;

Output:

ENAME	SAL	COMM
SMITH	800	N/A
ALLEN	1600	300
WARD	1250	500
JONES	2975	N/A
MARTIN	1250	1400
BLAKE	2850	N/A

Create a Table as following:

```
SQL> Select * from student;
```

Output:

STDID	SNAME	M1	M2	M3
2001	aa	55	77	
1001	Ravi	40	80	60
1002	Sravan	66	44	77

Replacing null with ABSENT:

```
SQL> Select sname, NVL(to_char(m3),'ABSENT') M3 from student;
```

Output:

SNAME	M3
aa	ABSENT
Ravi	60
Sravan	77

Examples on NVL2() Function:**Replacing 100 with 200:**

```
SQL> Select NVL2(100,200,300) from dual;
```

Output:

```
NVL2(100,200,300)
```

```
-----  
200
```

Replacing null with 300:

```
SQL> Select NVL2(NULL,200,300) from dual;
```

Output:

```
NVL2(NULL,200,300)
```

```
-----  
300
```

Setting commission as 1000 to the employees whose commission is null and increase 200 to the employees whose commission is not null:

SQL> Select Ename, NVL2(Comm,comm+200,1000) Comm from emp;

Output:

ENAME	COMM
SMITH	1000
ALLEN	500
WARD	700
JONES	1000
MARTIN	1600
BLAKE	1000
CLARK	1000

Displaying ranks according to salary in descending order using rank():

SQL> Select empno, ename, sal, rank() over(order by sal desc) Rank from emp;

Output:

EMPNO	ENAME	SAL	RANK
7839	KING	5000	1
7902	FORD	3000	2
7788	SCOTT	3000	2
7566	JONES	2975	4
7698	BLAKE	2850	5
7782	CLARK	2450	6
7499	ALLEN	1600	7
7844	TURNER	1500	8
7934	MILLER	1300	9
7521	WARD	1250	10
7654	MARTIN	1250	10
7876	ADAMS	1100	12
7900	JAMES	950	13
7369	SMITH	800	14

Displaying ranks according to salary in descending order using dense_rank():

SQL> Select empno, ename, sal, dense_rank() over(order by sal desc)
Rank from emp;

Output:

EMPNO	ENAME	SAL	RANK
7839	KING	5000	1
7902	FORD	3000	2
7788	SCOTT	3000	2
7566	JONES	2975	3
7698	BLAKE	2850	4
7782	CLARK	2450	5
7499	ALLEN	1600	6
7844	TURNER	1500	7
7934	MILLER	1300	8
7521	WARD	1250	9
7654	MARTIN	1250	9
7876	ADAMS	1100	10
7900	JAMES	950	11
7369	SMITH	800	12

Displaying ranks according to job descending order. If salary is same arrange in ascending order according to experience:

SQL> Select ename, Hiredate, sal, dense_rank() over(order by sal desc, hiredate asc) Rank from emp;

Output:

ENAME	HIREDATE	SAL	RANK
KING	17-NOV-81	5000	1
FORD	03-DEC-81	3000	2
SCOTT	09-DEC-82	3000	3
JONES	02-APR-81	2975	4
BLAKE	01-MAY-81	2850	5
CLARK	09-JUN-81	2450	6

ALLEN	20-FEB-81	1600	7
TURNER	08-SEP-81	1500	8
MILLER	23-JAN-82	1300	9
WARD	22-FEB-81	1250	10
MARTIN	28-SEP-81	1250	11
ADAMS	12-JAN-83	1100	12
JAMES	03-DEC-81	950	13
SMITH	17-DEC-80	800	14

Date Functions:

SQL provides following Date Functions:

Function Name	Purpose
Sysdate	<p>Returns current system date.</p> <p>Ex: Sysdate => 12-JUL-21</p>
Add_Months()	<p>Used to add months to specified date or subtract from specified date.</p> <p>Syntax: Add_Months(<Date>, <Number_Of_Months>)</p> <p>Ex: Add_Months(sysdate,2) => 12-SEP-21 Add_Months(sysdate,-2) => 12-MAY-21</p>
Next_Day()	<p>Returns the date that falls on next weekday given from the date.</p> <p>Syntax: Next_Day(Date, <Week_day_name>)</p> <p>Ex: Next_Day(sysdate, 'fri') => 16-JUL-21</p>
Last_Day()	<p>Returns last day of specified month in the date.</p> <p>Syntax: Last_Day(<Date>)</p> <p>Ex: Last_Day(sysdate) => 31-JUL-21</p>

Months_Between() Syntax: <code>Months_Between(<Date-1>, <Date-2>)</code> Ex: <code>Months_Between(sysdate, '12-APR-21') => 3</code>	<p>Returns the number of months between two dates.</p>
---	--

Miscellaneous [Other] Functions:

Function Name	Purpose
User Ex: <code>User => C##Oracle7AM</code>	<p>Returns Current Username.</p>
UId Ex: <code>UId => 111</code>	<p>Returns Current User ID.</p>
Greatest Syntax: <code>Greatest(<n1>,<n2>,...)</code> Ex: <code>Greatest(10,20,50,30) => 50</code>	<p>Returns greatest value in specified values. It can be used to find greatest value in a row.</p>

Least	Returns Least value in specified values. It can be used to find least value in a row. Syntax: Least(<n1>,<n2>,.....) Ex: Least(10,20,50,40) => 10
NVL()	Returns first value if first value is not null. Returns second value if first value is null. It is used to replace null value with other value. Syntax: NVL(<First_Value>,<Second_Value>) Ex: NVL(100,200) => 100 NVL(Null,200) => 200
NVL2()	It returns Second Value if first value is not null. It returns third value if the first value is null. It is used to replace nulls and not nulls. Syntax: NVL2(<first_value>,<second_value>,<third_value>) Ex: NVL2(100,200,300) => 200 NVL2(Null,200,300) => 300
VSize()	Returns number of bytes in internal representation of value. Syntax: VSize(<value>) Ex: VSize('raju') => 4

Rank()	<p>Used to apply ranks based on a column or columns. It does not maintain sequence of ranks when multiple values have same rank. It generates gaps between ranks when multiple values have same rank.</p> <p>Syntax:</p> <p>Rank() over(order by <column_name> Asc/ Desc [,...])</p> <p>Ex:</p> <p>Rank() over(order by sal desc) =></p> <p>applies ranks based on highest salary. It gives same rank for same values. It generates the gaps between ranks when same rank applied for multiple values.</p>
Dense_Rank()	<p>Used to apply ranks based on a column or columns. It maintains sequence of ranks when even if multiple values have same rank. It will not generate gaps between ranks even if multiple values have same rank.</p> <p>Syntax:</p> <p>Dense_Rank() over(order by <column_name> Asc/ Desc [,...])</p> <p>Ex:</p> <p>Dense_Rank() over(order by sal desc) =></p> <p>applies ranks based on highest salary. It gives same rank for same values. It will not generate the gaps between ranks even if same rank applied for multiple values.</p>

Row_Number()	<p>Used to apply row number for every record.</p> <p>Syntax:</p> <pre>Row_Number() over(order by <column_name> Asc/ Desc [,...])</pre> <p>Ex:</p> <p>Row_Number() over(order by sal desc) => applies record number for every row. Highest salary record number is 1. Second Highest Salary record number is 2 and so on.</p>
---------------------	---

Queries on Math Functions:

Finding 2 power 3:

SQL> Select power(2,3) from dual;

Output:

POWER(2,3)

8

Finding 5 power 2 and 3 power 3:

SQL> Select Power(5,2), Power(3,3) from dual;

Output:

POWER(5,2) POWER(3,3)

25 27

Finding Square root of 100:

SQL> Select Sqrt(100) from dual;

Output:

SQRT(100)

10

Finding Square root of 81 and square root of 144:

SQL> Select Sqrt(81), Sqrt(144) from dual;

Output:

SQRT(81) SQRT(144)

9 12

Finding 5 mod 2:

SQL> Select Mod(5,2) from dual;

Output:

MOD(5,2)

1

Finding 10 mod 6 and 20 mod 7:

SQL> Select Mod(10,6), Mod(20,7) from dual;

Output:

MOD(10,6) MOD(20,7)

4 6

Finding sin 90:

SQL> Select Sin(90*3.14/180) from dual;

Output:

SIN(90*3.14/180)

.999999683

Finding Cos 0 and Cos 45:

SQL> Select Cos(0*3.14/180), Tan(45*3.14/180) from dual;

Output:

COS(0*3.14/180) TAN(45*3.14/180)

1 .99920399

Finding natural logarithmic value of 7:

SQL> Select Ln(7) from dual;

Output:

LN(7)

1.94591015

Finding Log 10 base 10:

SQL> Select Log(10,10) from dual;

Output:

LOG(10,10)

1

Examples on Trunc():

1. SQL> Select Trunc(123.4567) from dual;

Output:

TRUNC(123.4567)

123

Removed all decimal places.

2. SQL> Select Trunc(123.4567,2) from dual;

Output:

TRUNC(123.4567,2)

123.45

Printed up to 2 decimal places. Remaining decimal places truncated.

3. SQL> Select Trunc(123.4567,3) from dual;

Output:

TRUNC(123.4567,3)

123.456

Printed up to 3 decimal places. Remaining decimal places truncated.

4. SQL> Select Trunc(123.4567,-1) from dual;

Output:

TRUNC(123.4567,-1)

120

TRUNC(123.4567,-1) => Rounds to 10s.

120.....123.4567.....130

trunc() always gives lower value. So, 120 printed.

5. SQL> Select Trunc(123.4567,-2) from dual;

Output:

TRUNC(123.4567,-2)

100

TRUNC(123.4567,-1) => Rounds to 100s.

100.....123.4567.....200

trunc() always gives lower value. So, 100 printed.

Examples on Round():

1. SQL> Select Round(123.4567) from dual;

Output:

```
ROUND(123.4567)
```

```
123
```

.4 is there. Below .5 . So, printed below value 123

2. SQL> Select Round(123.6567) from dual;

Output:

```
ROUND(123.6567)
```

```
124
```

.6 is there. Above .5 . So, printed above value 124

3. SQL> Select Round(123.5567) from dual;

Output:

```
ROUND(123.5567)
```

```
124
```

.5 is there. So, printed above value 124

4. SQL> Select Round(123.5567,2) from dual;

Output:

```
ROUND(123.5567,2)
```

```
123.56
```

After second decimal place 6 is there. It is above .5.
So rounded to 123.56.

5. SQL> Select Round(123.5547,2) from dual;

Output:

ROUND(123.5547,2)

123.55

After second decimal place 4 is there. It is below .5.
So rounded to 123.55.

6. SQL> Select Round(123.5557,2) from dual;

Output:

ROUND(123.5557,2)

123.56

After second decimal place 5 is there. It is equals to .5.
So rounded to 123.56.

7. SQL> Select Round(123.4567,-1) from dual;

Output:

ROUND(123.4567,-1)

120

-1 means, Rounds to 10s

120123.4567.....130

3 is there in 1's position. So, Rounds to 120

8. SQL> Select Round(126.6567,-1) from dual;

Output:

ROUND(126.6567,-1)

130

-1 means, rounds to 10s

120.....126.6567.....130

6 is there at 1's position. >5. So, rounds to 130

9. SQL> Select Round(125.6567,-1) from dual;

Output:

ROUND(125.6567,-1)

130

120.....125.6567.....130

5 is there 1's position. Eqauls to 5. So rounds to 130

10. SQL> Select Round(124.6567,-1) from dual;

Output:

ROUND(124.6567,-1)

120

120.....124.6567.....130

4 is there 1's position. So, rounds to 120

11. SQL> Select Round(123.6567,-2) from dual;

Output:

ROUND(123.6567,-2)

100

-2 means, Rounds in 100s

100.....123.6567.....200

2 is there at 2's position. Less than 5. So, rounds to 100

12. SQL> Select Round(152.6567,-2) from dual;

Output:

ROUND(152.6567,-2)

200

-2 means, Rounds in 100s

100.....152.6567.....200

5 is there at 2's position. Equals to 5. So, rounds to 200

13. SQL> Select Round(150.6567,-2) from dual;

Output:

ROUND(150.6567,-2)

200

-2 means, Rounds in 100s

100.....150.6567.....200

5 is there at 2's position. Equals to 5. So, rounds to 200

14. SQL> Select Round(149.6567,-2) from dual;

Output:

ROUND(149.6567,-2)

100

-2 means, Rounds in 100s

100.....149.6567.....200

4 is there at 2's position. Less Than 5. So, rounds to 100

Example on Ceil():

SQL> Select Ceil(123.456) from dual;

Output:

CEIL(123.456)

124

Ceil() always gives upper integer value. So, printed 124

Example on Floor():

SQL> Select Floor(123.456) from dual;

Output:

FLOOR(123.456)

123

Floor() function always gives lower integer value. So, Printed 123

Example on Trunc(), Round(), Floor() & Ceil Functions:

Create a table as following:

SQL> Select * from player;

PID	FNAME	LNAME	NO_OF_MATCHES	TOTAL_RUNS
1001	sachin	tendulkar	300	10000
1002	virat	kohli	150	7000
1003	rohit	sharma	130	6800
1004	rahul	dravid	200	9000

SQL> Select Total_Runs/No_Of_Matches as "Avrg Score" from player;

Output:

Avrg Score

33.3333333
46.6666667
52.3076923
45

SQL> Select Round(Total_Runs/No_Of_Matches) as "Avrg Score" from player;

Output:

Avrg Score

33
47
52
45

SQL> Select Round(Total_Runs/No_Of_Matches,2) as "Avrg Score" from player;

Output:

Avrg Score

33.33
46.67
52.31
45

SQL> Select Trunc(Total_Runs/No_Of_Matches) as "Avrg Score" from player;

Output:

Avrg Score

33
46
52
45

SQL> Select Trunc(Total_Runs/No_Of_Matches,2) as "Avrg Score" from player;

Output:

Avrg Score

33.33
46.66
52.3
45

SQL> Select Trunc(Total_Runs/No_Of_Matches,3) as "Avrg Score" from player;

Output:

Avrg Score

33.333
46.666
52.307
45

SQL> Select Floor(Total_Runs/No_Of_Matches) as "Avrg Score" from player;

Output:

Avrg Score

33
46
52
45

SQL> Select Ceil(Total_Runs/No_Of_Matches) as "Avrg Score" from player;

Output:

Avrg Score

34
47
53
45

Queries on Aggregate Functions:

Finding Sum of Salaries of Employees:

```
SQL> Select sum(Sal) as "Sum of Salaries" from emp;
```

Average Salary

29025

Finding Average of Salaries of Employees:

```
SQL> Select Avg(Sal) as "Average Salary" from emp;
```

Average Salary

2073.21429

Finding Average salary by truncating up to 2 decimal places:

```
SQL> Select Trunc(Avg(Sal),2) as "Average Salary" from emp;
```

Average Salary

2073.21

Finding average salary. Take Ceil value as Average Salary:

```
SQL> Select Ceil(Avg(Sal)) as "Average Salary" from emp;
```

Average Salary

2074

Finding average salary. Take Floor value as Average Salary:

SQL> Select Floor(Avg(Sal)) as "Average Salary" from emp;

Average Salary

2073

Finding average salary. Take Rounded value as Average Salary:

SQL> Select Round(Avg(Sal)) as "Average Salary" from emp;

Average Salary

2073

Finding average salary. Take Rounded value upto 2 decimal places as Average Salary:

SQL> Select Round(Avg(Sal),2) as "Average Salary" from emp;

Average Salary

2073.21

Finding Minimum Salary:

SQL> Select Min(Sal) as "Min Salary" from emp;

Min Salary

800

Finding Maximum Salary:

SQL> Select Max(Sal) as "Max Salary" from emp;

Max Salary

5000

Counting Comm Column values:

SQL> Select Count(comm) from emp;

COUNT(COMM)

3

Note:

Count(Comm) cannot count nulls.

Counting empno column values:

SQL> Select Count(empno) from emp;

COUNT(EMPNO)

14

Counting Number of Records:

SQL> Select Count(*) from emp;

COUNT(*)

14

Finding Employee name who is getting maximum salary:

```
SQL> select ename from emp  
      where sal=(Select max(Sal) from emp);
```

Output:

ENAME

KING

Finding Employee name who is getting minimum salary:

```
SQL> select ename from emp  
      where sal=(Select min(Sal) from emp);
```

Output:

ENAME

SMITH

Finding Second Maximum Salary:

```
SQL> select max(Sal) from emp  
      where sal<(Select max(sal) from emp);
```

Output:

MAX(SAL)

3000

Finding employee name who is getting second maximum salary:

```
SQL> select ename from emp  
      where sal = (select max(Sal) from emp where sal<(Select max(sal)  
from emp));
```

Output:

ENAME

SCOTT

FORD

Counting Number of Clerks in Emp Table:

```
SQL> Select count(*) as "Number Of Clerks" from emp  
      where job='CLERK';
```

Output:

Number Of Clerks

4

Counting Number of Managers in Emp Table:

```
SQL> Select count(*) as "Number Of Managers" from emp  
      where job='MANAGER';
```

Output:

Number Of Managers

3

Finding Number of Employees in Deptno 20:

```
SQL> Select count(*) as "Number Of Employees" from emp  
      where deptno=20;
```

Output:

Number Of Employees

5

Queries on String Functions:

Using Upper() Function:

```
SQL> Select Upper('raju') from dual;
```

Output:

```
UPPE
```

```
----
```

```
RAJU
```

Using Lower() Function:

```
SQL> Select Lower('RAJU') from dual;
```

Output:

```
LOWE
```

```
----
```

```
raju
```

Using Initcap() Function:

```
SQL> Select Initcap('RAJ KUMAR') from dual;
```

Output:

```
INITCAP('
```

```
-----
```

```
Raj Kumar
```

Using Concat() Function:

```
SQL> Select Concat('raj','kumar') from dual;
```

Output:

```
CONCAT('
```

```
-----
```

```
rajkumar
```

Getting space between two names:

```
SQL> Select concat(concat('raj', ' '), 'kumar') from dual;
```

Output:

```
CONCAT(CO
```

```
-----
```

```
raj kumar
```

(Or)**--using || [Concatenation Operator]**

```
SQL> Select 'raj' || ' ' || 'kumar' from dual;
```

Output:

```
'RAJ'||"
```

```
-----
```

```
raj kumar
```

Getting space between two names and display initial letters as capital in name:

```
SQL> Select Initcap(concat(concat('raj', ' '), 'kumar')) from dual;
```

Output:

```
INITCAP(C
```

```
-----
```

```
Raj Kumar
```

Displaying Employee Names in Upper Case:

```
SQL> Select Upper(Ename) from emp;
```

Output:

```
UPPER(ENAM
```

```
-----
```

```
SMITH
```

```
ALLEN
```

```
.
```

Displaying Employee Names in Lower Case:

```
SQL> Select Lower(Ename) from Emp;
```

Output:

```
LOWER(ENAM
```

```
-----  
smith
```

```
allen
```

```
.
```

```
.
```

Displaying Employee Names initial letter as capital:

```
SQL> Select Initcap(Ename) from emp;
```

Output:

```
INITCAP(EN
```

```
-----  
Smith
```

```
Allen
```

```
.
```

```
.
```

Create a Table as following:

```
SQL> Select * from player;
```

PID	FNAME	LNAME
1001	sachin	tendulkar
1002	virat	kohli
1003	rohit	sharma
1004	rahul	dravid

Concatenating First Name and Last Name and getting name's initial letters as capital:

SQL> Select Initcap(Concat(Concat(fname,' '),lname)) as Name from player;

Output:

NAME

Sachin Tendulkar
Virat Kohli
Rohit Sharma
Rahul Dravid

Displaying First name in Upper Case:

SQL> Select upper(Fname) from player;

Output:

UPPER(FNAM

SACHIN
VIRAT
ROHIT
RAHUL

Displaying First name in Lower Case:

SQL> Select lower(Fname) from player;

Output:

LOWER(FNAM

sachin
virat
rohit
rahul

Add a Column “PName” and store player name in “PName” column by concatenating First Name and Last Name and drop fname and lname columns:

Adding PName field:

```
SQL> Alter Table Player add PName Varchar2(30);
```

Output:

Table altered.

Updating Player Names by concatenating fname and lname:

```
SQL> Update Player
```

```
Set PName = Initcap(Concat(Concat(FName,' '),LName));
```

Output:

4 rows updated.

Dropping Fname & Lname columns:

```
SQL> Alter Table Player Drop(Fname, Lname);
```

Output:

Table altered.

Retrieving player table records:

```
SQL> Select PID, PNAME from player;
```

Output:

PID	PNAME
1001	Sachin Tendulkar
1002	Virat Kohli
1003	Rohit Sharma
1004	Rahul Dravid

Finding String Length:

```
SQL> Select Length('raju') from dual;
```

Output:

```
LENGTH('RAJU')
```

```
-----  
4
```

Finding string length of each employee name:

```
SQL> Select ename, Length(ename) "Ename Length" from emp;
```

Output:

ENAME	Ename Length
-------	--------------

SMITH	5
ALLEN	5
WARD	4

```
...
```

Displaying employee records whose name has 4 characters:

```
SQL> Select ename from emp where ename like '____';
```

Output:

```
ENAME
```

```
-----  
WARD  
KING
```

[Or]

```
SQL> Select ename from emp where Length(Ename) = 4;
```

Output:

```
ENAME
```

```
-----  
WARD  
KING
```

Retrieving Player Table Records:

SQL> Select PID,PName from Player;

Output:

PID PNAME

1001 Sachin Tendulkar

1002 Virat Kohli

1003 Rohit Sharma

1004 Rahul Dravid

Displaying player names whose names are having 12 chars:

SQL> Select pname from player where length(pname)=12;

Output:

PNAME

Rohit Sharma

Rahul Dravid

Displaying player names whose names are having more than 12 chars:

SQL> Select pname from player where length(pname)>12;

Output:

PNAME

Sachin Tendulkar

Examples on Substr() Function:

SQL> Select substr('raj kumar',5) from dual;

Output:

SUBST

kumar

SQL> Select substr('raj kumar',1,3) from dual;

Output:

SUB

raj

SQL> Select substr('raj kumar',1,5) from dual;

Output:

SUBST

raj k

SQL> Select substr('raj kumar',6,3) from dual;

Output:

SUB

uma

SQL> Select substr('raj kumar',6) from dual;

Output:

SUBS

umar

SQL> Select substr('raj kumar',-4) from dual;

Output:

SUBS

umar

SQL> Select substr('raj kumar',-4,3) from dual;

Output:

SUB

uma

SQL> Select substr('raj kumar',-5) from dual;

Output:

SUBST

kumar

SQL> Select substr('raj kumar',-5,3) from dual;

Output:

SUB

kum

Note:

-ve position => Position Number From Right Side

+ve Position => Position Number From Left Side

Generating Email IDs by taking emp name's first 3 chars and empno's last 3 digits:

SQL> Select

substr(Ename,1,3) || substr(empno,-3,3) || '@nareshit.com'

As "E-Mail ID" from emp;

Output:

E-Mail ID

SMI369@nareshit.com

ALL499@nareshit.com

..

Displaying employee names whose name started with 'S':

SQL> Select EName from emp where Ename Like 'S%';

Output:

ENAME

SMITH

SCOTT

(Or)

SQL> Select EName from emp where substr(Ename,1,1) = 'S';

Output:

ENAME

SMITH

SCOTT

Retrieving the emp record whose name is 'blake' and when we don't know whether the name is in upper case or lower case:

SQL> Select ename, sal from emp where lower(ename) = 'blake';

Output:

ENAME SAL

----- -----

BLAKE 2850

Displaying Employee names whose names are ended with 'RD':

SQL> Select ename from emp where substr(ename,-2,2) = 'RD';

Output:

ENAME

WARD

FORD

(Or)

SQL> Select ename from emp where ename like '%RD';

Output:

ENAME

WARD

FORD

Displaying employee names whose names are started and ended with same letter:

SQL> Insert into emp(empno,ename) values(1001,'DAVID');

Output:

1 row created.

SQL> Insert into emp(empno,ename) values(1002,'SRINIVAS');

Output:

1 row created.

SQL> Select ename from emp
where substr(ename,1,1) = substr(ename,-1,1);

Output:

ENAME

DAVID

SRINIVAS

Examples on Instr() Function:

SQL> Select Instr('This is his wish','is') from dual;

Output:

INSTR('THISISHISWISH','IS')

SQL> Select Instr('This is his wish','is',4) from dual;

Output:

INSTR('THISISHISWISH','IS',4)

6

SQL> Select Instr('This is his wish','is',7) from dual;

Output:

INSTR('THISISHISWISH','IS',7)

10

SQL> Select Instr('This is his wish','is',11) from dual;

Output:

INSTR('THISISHISWISH','IS',11)

14

SQL> Select Instr('This is his wish','is',15) from dual;

Output:

INSTR('THISISHISWISH','IS',15)

0

Note:

If substring is not found in the string, then it returns 0

SQL> Select Instr('This is his wish','is',-1) from dual;

Output:

INSTR('THISISHISWISH','IS',-1)

14

Note:

Positive Position => Starts searching from left side

Negative Position => Starts searching from right side

SQL> Select Instr('This is his wish','is',-4) from dual;

Output:

INSTR('THISISHISWISH','IS',-4)

10

SQL> Select Instr('This is his wish','is',-4,2) from dual;

Output:

INSTR('THISISHISWISH','IS',-4,2)

6

Displaying Employee Names whose names are containg 'AM' characters:

SQL> Select ename from emp where ename like '%AM%';

Output:

ENAME

ADAMS

JAMES

(or)

SQL> Select ename from emp where instr(ename,'AM')>0;

Output:

ENAME

ADAMS

JAMES

Examples on LPad() and RPad Functions:

SQL> Select Lpad('raju',10,'*') from dual;

Output:

LPAD('RAJU

*****raju

SQL> Select Lpad('raju',15,'@#') from dual;

Output:

LPAD('RAJU',15,

@#@#@#@#@#@#@raju

SQL> Select Rpad('raju',10,'*') from dual;

Output:

RPAD('RAJU

raju*****

SQL> Select Rpad('raju',15,'@#') from dual;

Output:

RPAD('RAJU',15,

raju@#@#@#@#@#@@

SQL> Select Lpad('raju',10) from dual;

Output:

LPAD('RAJU

raju

```
SQL> Select Rpad('raj',10) || 'kumar' from dual;
```

Output:

```
RPAD('RAJ',10)|
```

```
-----  
raj      kumar
```

```
SQL> Select Lpad('*',10,'*') from dual;
```

Output:

```
LPAD('*',1
```

```
-----  
*****
```

```
SQL> Select 'amount credited from the acno ' || LPad('X',6,'X') ||  
Substr('123456789',-4,4) Message from dual;
```

```
MESSAGE
```

```
-----  
amount credited from the acno XXXXX6789
```

Examples on LTrim(), RTrim() and Trim() Functions:

```
SQL> Select LTrim('   raj   ') || 'kumar' from dual;
```

Output:

```
LTRIM('RAJ')
```

```
-----  
raj  kumar
```

```
SQL> Select RTrim('   raj   ') || 'kumar' from dual;
```

Output:

```
RTRIM('RAJ')|
```

```
-----  
rajkumar
```

```
SQL> Select Trim('    raj    ') || 'kumar' from dual;
```

Output:

```
TRIM('RA
```

```
-----  
rajkumar
```

```
SQL> Select LTrim('@@@@@@raju@@@') from dual;
```

Output:

```
LTRIM('@@@@R
```

```
-----  
@@@@@@raju@@@
```

```
SQL> Select LTrim('@@@@@@raju@@@','@') from dual;
```

Output:

```
LTRIM('@
```

```
-----  
raju@@@
```

```
SQL> Select RTrim('@@@@@@raju@@@','@') from dual;
```

Output:

```
RTRIM('@@
```

```
-----  
@@@raju
```

```
SQL> Select LTrim('@#@#####@@@raju@#@#####@@@#@##','@#')  
from dual;
```

Output:

```
LTRIM('@#@#####@@@R
```

```
-----  
raju@#@#####@@@#@##
```

```
SQL> Select RTrim('@#@#####@@@raju@#@#####@@@#@##','@#')  
from dual;
```

Output:

```
RTRIM('@#@#####  
-----  
@#@#####@@@raju
```

```
SQL> Select Trim(Leading '@' from '@@@@raju@@@@') from dual;
```

Output:

```
TRIM(LEAD  
-----  
raju@@@@@
```

```
SQL> Select Trim(Trailing '@' from '@@@@raju@@@@') from dual;
```

Output:

```
TRIM(TRA  
-----  
@@@@raju
```

```
SQL> Select Trim(Both '@' from '@@@@raju@@@@') from dual;
```

Output:

```
TRIM  
----  
raju
```

```
SQL> Select LTrim('raju@nareshit.com','@gmail.com') from dual;
```

Output:

```
LTRIM('RAJU@NARES  
-----  
raju@nareshit.com
```

```
SQL> Select RTrim('raju@nareshit.com','@gmail.com') from dual;
```

Output:

```
RTRIM('RAJU@N
```

```
-----  
raju@nareshit
```

```
SQL> Select RTrim('raju@nareshit.com','@nareshit.com') from dual;
```

Output:

```
RTRI
```

```
-----  
raju
```

```
SQL> Select LTrim('OPPO F11 Pro','OPPO') from dual;
```

Output:

```
LTRIM('O
```

```
-----  
F11 Pro
```

Examples on Replace and Translate() Functions:

```
SQL> Select Replace('sai krishna','sai','rama') from dual;
```

Output:

```
REPLACE('SAI
```

```
-----  
rama krishna
```

```
SQL> Select Replace('xyzxyzxxxxyzxyzxyz','xyz','abc') from dual;
```

Output:

```
REPLACE('XYZXYZXXYYZZ
```

```
-----  
abcabccxyyzzabcxyyzz
```

```
SQL> Select Translate('xyzxyzxyzxyzxyzxyz','xyz','abc') from dual;
```

Output:

```
TRANSLATE('XYZXYZXXYY
```

```
abcabcaabbccabcaabbcc
```

Displaying salary value with mask characters:

```
SQL> Column salary format A10
```

```
SQL> Select ename, Translate(sal,'0123456789','!@#$%^&*+=') Salary  
from emp;
```

Output:

```
ENAME    SALARY
```

```
SMITH    +!!
```

```
ALLEN    @&!!
```

```
WARD     @#^!
```

```
..
```

Examples on ASCII and Chr() Functions:

```
SQL> Select ASCII('A') from dual;
```

Output:

```
ASCII('A')
```

```
65
```

```
SQL> Select ASCII('a'), ASCII('z') from dual;
```

Output:

```
ASCII('A') ASCII('Z')
```

```
97    122
```

SQL> Select Chr(65) from dual;

Output:

C

-

A

SQL> Select Chr(97), Chr(122) from dual;

Output:

C C

--

a z

Examples on Soundex() Function:

SQL> Select empno,ename from emp where Soundex(ename) = Soundex('SMYT');

Output:

EMPNO ENAME

7369 SMITH

SQL> Select empno,ename from emp where Soundex(ename) = Soundex('SKAT');

Output:

EMPNO ENAME

7788 SCOTT

Examples on Reverse() Function:

SQL> Select Reverse('ramu') from dual;

Output:

REVE

umar

SQL> Select EName, Reverse(Ename) as "Rev Name" from emp;

Output:

ENAME	Rev Name
-------	----------

SMITH	HTIMS
ALLEN	NELLA
WARD	DRAW
JONES	SENOJ

.

.

Example Queries on Conversion Functions:**Converting Date To String:**

Getting system date in different formats:

Getting year four digits from today's date:

SQL> Select to_char(sysdate,'yyyy') from dual;

Output:

TO_C

2021

Getting year two digits from today's date:

```
SQL> Select to_char(sysdate,'yy') from dual;
```

Output:

TO

--

21

Getting year one digit from today's date:

```
SQL> Select to_char(sysdate,'y') from dual;
```

Output:

T

-

1

Getting year three digits from today's date:

```
SQL> Select to_char(sysdate,'yyy') from dual;
```

Output:

TO_

021

Getting year in words from today's date:

```
SQL> Select to_char(sysdate,'year') from dual;
```

Output:

TO_CHAR(SYSDATE,'YEAR')

twenty twenty-one

Getting month two digits from today's date:

```
SQL> Select to_char(sysdate,'mm') from dual;
```

Output:

```
TO
```

```
--
```

```
07
```

Getting short month name from today's date:

```
SQL> Select to_char(sysdate,'mon') from dual;
```

Output:

```
TO_CHAR(SYSD
```

```
-----
```

```
jul
```

Getting full month name from today's date:

```
SQL> Select to_char(sysdate,'month') from dual;
```

Output:

```
TO_CHAR(SYSDATE,'MONTH')
```

```
-----
```

```
july
```

Getting short month name in upper case:

```
SQL> Select to_char(sysdate,'MON') from dual;
```

Output:

```
TO_CHAR(SYSD
```

```
-----
```

```
JUL
```

Getting full month name in upper case:

```
SQL> Select to_char(sysdate,'MONTH') from dual;
```

Output:

```
TO_CHAR(SYSDATE,'MONTH')
```

```
JULY
```

Getting date two digits [day number in month] from today's date:

```
SQL> Select to_char(sysdate,'dd') from dual;
```

Output:

```
TO
```

```
--
```

```
15
```

Getting day number in the year of today's date:

```
SQL> Select to_char(sysdate,'ddd') from dual;
```

Output:

```
TO_
```

```
--
```

```
196
```

Getting day number in week of today's date:

```
SQL> Select to_char(sysdate,'d') from dual;
```

Output:

```
T
```

```
-
```

```
5
```

Getting short weekday name of today's date:

```
SQL> Select to_char(sysdate,'dy') from dual;
```

Output:

```
TO_CHAR(SYSD
```

```
-----  
thu
```

Getting full weekday name of today's date:

```
SQL> Select to_char(sysdate,'day') from dual;
```

Output:

```
TO_CHAR(SYSDATE,'DAY')
```

```
-----  
thursday
```

Getting short weekday name in upper case:

```
SQL> Select to_char(sysdate,'DY') from dual;
```

Output:

```
TO_CHAR(SYSD
```

```
-----  
THU
```

Getting weekday full name of today's date:

```
SQL> Select to_char(sysdate,'DAY') from dual;
```

Output:

```
TO_CHAR(SYSDATE,'DAY')
```

```
-----  
THURSDAY
```

Getting time from today's date:

```
SQL> Select to_char(sysdate,'hh:mi AM') from dual;
```

Output:

```
TO_CHAR(
```

```
-----
```

```
11:41 AM
```

Getting quarter number of today's date:

```
SQL> Select to_char(sysdate,'Q') from dual;
```

Output:

```
T
```

```
-
```

```
3
```

Getting week number of today's date in the month:

```
SQL> Select to_char(sysdate,'W') from dual;
```

Output:

```
T
```

```
-
```

```
3
```

Getting weekday number of today's date in the year:

```
SQL> Select to_char(sysdate,'WW') from dual;
```

Output:

```
TO
```

```
--
```

```
28
```

Getting Current Century number:

```
SQL> Select to_char(sysdate,'CC') from dual;
```

TO

--

21

Getting Current Century Number and AD Year or BC Year :

```
SQL> Select to_char(sysdate,'CC AD') from dual;
```

Output:

TO_CH

21 AD

Example Queries on to_Char Function:**Displaying employee records who joined in 1982:**

```
SQL> Select ename,hiredate from emp where hiredate like '%82';
```

Output:

ENAME	HIREDATE
-------	----------

----- -----

SCOTT 09-DEC-82

MILLER 23-JAN-82

[Or]

```
SQL> Select ename,hiredate from emp where  
to_char(hiredate,'yyyy')=1982;
```

Output:

ENAME	HIREDATE
-------	----------

----- -----

SCOTT 09-DEC-82

MILLER 23-JAN-82

Displaying employee records who joined in 1980, 1982,1983:

SQL> Select ename,hiredate from emp where to_char(hiredate,'yyyy') in (1980,1982,1983);

Output:

ENAME	HIREDATE
SMITH	17-DEC-80
SCOTT	09-DEC-82
ADAMS	12-JAN-83
MILLER	23-JAN-82

Displaying employee records who joined on Sunday:

SQL> Select ename, hiredate from emp where to_char(hiredate,'d') =1;

Output:

ENAME	HIREDATE
WARD	22-FEB-81

[Or]

SQL> Select ename, hiredate from emp where to_char(hiredate,'dy')='sun';

Output:

ENAME	HIREDATE
WARD	22-FEB-81

[Or]

SQL> Select ename, hiredate from emp where rtrim(to_char(hiredate,'day'))='sunday';

Output:

ENAME	HIREDATE
WARD	22-FEB-81

Displaying employee records who joined in second quarter:

```
SQL> Select ename, hiredate from emp where to_char(hiredate,'q') = 2;
```

Output:

ENAME	HIREDATE
JONES	02-APR-81
BLAKE	01-MAY-81
CLARK	09-JUN-81

Displaying employee records who joined in Jan, Apr and Dec:

```
SQL> Select ename, hiredate from emp where to_char(hiredate,'mm') in  
(1,4,12);
```

ENAME	HIREDATE
SMITH	17-DEC-80
JONES	02-APR-81
SCOTT	09-DEC-82
ADAMS	12-JAN-83
JAMES	03-DEC-81
FORD	03-DEC-81
MILLER	23-JAN-82

Displaying hiredate in mm/dd/yyyy format:

```
SQL> Select ename, to_Char(hiredate,'mm/dd/yyyy') hiredate from emp;
```

Output:

ENAME	HIREDATE
SMITH	12/17/1980
ALLEN	02/20/1981
.	
.	

Displaying hiredate in dd/mm/yyyy format:

```
SQL> Select ename, to_Char(hiredate,'dd/mm/yyyy') hiredate from emp;
```

Output:

```
ENAME    HIREDATE
```

```
-----  
SMITH   17/12/1980  
ALLEN   20/02/1981
```

Converting String to Date:**Converting string to date:**

```
SQL> Select to_date('20 december 2021') from dual;
```

Output:

```
TO_DATE('
```

```
-----  
20-DEC-21
```

Converting string to date:

```
SQL> Select to_date('2-dec-2021') from dual;
```

Output:

```
TO_DATE('
```

```
-----  
02-DEC-21
```

Converting string to date:

```
SQL> Select to_date('23/12/2020','dd/mm/yyyy') from dual;
```

Output:

```
TO_DATE('
```

```
-----  
23-DEC-20
```

Getting year part from a date:

```
SQL> Select to_char(to_date('23-dec-2020'),'yyyy') from dual;
```

Output:

TO_C

2020

Getting month name from a date:

```
SQL> Select to_char(to_date('23-dec-2020'),'mon') from dual;
```

Output:

TO_

dec

Getting weekday name of a date:

```
SQL> Select to_char(to_date('23-dec-2020'),'day') from dual;
```

Output:

TO_CHAR(T

wednesday

Adding 10 days to sysdate:

```
SQL> Select sysdate+10 from dual;
```

Output:

SYSDATE+1

25-JUL-21

Adding 10 days to a Date:

```
SQL> Select to_date('16-jun-2021')+10 from dual;
```

Output:

```
TO_DATE('  
-----  
26-JUN-21
```

Displaying weekday name of INDIA's Independence Day:

```
SQL> Select to_char(to_date('15-AUG-1947'),'day') from dual;
```

Output:

```
TO_CHAR(T  
-----  
friday
```

Displaying weekday name of SACHIN's Birthday:

```
SQL> Select to_char(to_date('24-APR-1973'),'day') from dual;
```

Output:

```
TO_CHAR(T  
-----  
tuesday
```

Inserting date value in emp table:

```
SQL> Insert into emp(empno, ename, hiredate)  
values(1234,'ccc',to_date('16-nov-2020'));
```

Output:

```
1 row created.
```

Converting Number to String:

Displaying employee salary with currency symbol \$. Also display 2 decimal places:

SQL> Select to_char(5000,'L9999.99') from dual;

Output:

TO_CHAR(5000,'L999

\$5000.00

Displaying currency name:

SQL> Select to_char(5000,'C9999.99') from dual;

Output:

TO_CHAR(5000,'C

USD5000.00

Displaying NLS Parameters [NLS = National Language Support]:

SQL> Select * from v\$NLS_PARAMETERS;

Output:

PARAMETER	VALUE
NLS_LANGUAGE	AMERICAN
NLS_TERRITORY	AMERICA
NLS_CURRENCY	\$
.	

Changing currency and country:

SQL> Alter session set nls_territory='INDIA';

Output:

Session altered.

```
SQL> Alter session set nls_currency='RS';
```

Output:

```
Session altered.
```

```
SQL> Select to_char(5000,'L9999.99') from dual;
```

Output:

```
TO_CHAR(5000,'L999
```

```
-----  
RS5000.00
```

```
SQL> Select to_char(5000,'C9999.99') from dual;
```

Output:

```
TO_CHAR(5000,'C
```

```
-----  
INR5000.00
```

Displaying employee salary with currency symbol & decimal places:

```
SQL> Select ename, to_char(Sal,'L99999.99') Salary from emp;
```

Output:

```
ENAME    SALARY
```

```
-----  
SMITH      RS800.00  
ALLEN     RS1600.00  
WARD      RS1250.00
```

```
.  
. .  
.
```

Displaying employee salary with currency name:

```
SQL> Select ename, to_char(Sal,'C99999.99') Salary from emp;
```

Output:

```
ENAME    SALARY
-----
CCC
SMITH    INR800.00
ALLEN   INR1600.00
```

Getting zeros in front of remaining digits places:

```
SQL> Select to_char(1234,'000000') from dual;
```

TO_CHAR

```
-----
001234
```

Converting String to Number:

```
SQL> Select to_number('123') from dual;
```

Output:

```
TO_NUMBER('123')
-----
123
```

```
SQL> Select to_number('$5000.00','L9999.99') from dual;
```

Output:

```
TO_NUMBER('$5000.00','L9999.99')
-----
5000
```

```
SQL> Select to_number('USD5000.00','C9999.99') from dual;
```

```
TO_NUMBER('USD5000.00','C9999.99')
-----
5000
```

Queries on Date Functions:

Displaying current system date:

```
SQL> Select sysdate from dual;
```

Output:

```
SYSDATE
```

```
-----  
16-JUL-21
```

Displaying current system time:

```
SQL> Select systimestamp from dual;
```

Output:

```
SYSTIMESTAMP
```

```
-----  
16-JUL-21 04.20.34.223000 PM +05:30
```

Displaying time from sysdate:

```
SQL> Select to_char(sysdate,'hh:mi AM') from dual;
```

Output:

```
TO_CHAR(
```

```
-----  
04:21 PM
```

Displaying time in 24Hours Format from sysdate:

```
SQL> Select to_char(sysdate,'hh24:mi') from dual;
```

Output:

```
TO_CH
```

```
-----  
16:21
```

Displaying date from systimestamp:

```
SQL> Select to_char(systimestamp,'dd/mm/yyyy') from dual;
```

Output:

```
TO_CHAR(SY
```

```
-----  
16/07/2021
```

Displaying time from systimestamp:

```
SQL> Select to_char(systimestamp,'hh:mi AM') from dual;
```

Output:

```
TO_CHAR(
```

```
-----  
04:22 PM
```

Displaying time in 24 Hours Format:

```
SQL> Select to_char(systimestamp,'hh24:mi') from dual;
```

Output:

```
TO_CH
```

```
-----  
16:22
```

Adding 2 months to today's date:

```
SQL> Select Add_Months(sysdate,2) from dual;
```

Output:

```
ADD_MONTH
```

```
-----  
16-SEP-21
```

Subtracting 2 months from today's date:

```
SQL> Select Add_Months(sysdate,-2) from dual;
```

Output:

```
ADD_MONTH
```

```
-----
```

```
16-MAY-21
```

Adding two months to a date:

```
SQL> Select Add_Months(to_Date('20-oct-2020'),2) from dual;
```

Output:

```
ADD_MONTH
```

```
-----
```

```
20-DEC-20
```

Subtracting two months to a date:

```
SQL> Select Add_Months(to_Date('20-oct-2020'),-2) from dual;
```

Output:

```
ADD_MONTH
```

```
-----
```

```
20-AUG-20
```

Inserting a record into emp table with hiredate as sysdate:

```
SQL> Insert into emp(empno,ename,job,sal,hiredate)
   values(2001,'Krishna','MANAGER',15000,sysdate);
```

Output:

```
1 row created.
```

Displaying employee records who joined today:

```
SQL> Select empno, ename, hiredate from emp where  
      to_char(Hiredate,'dd/mm/yyyy') = to_char(Sysdate,'dd/mm/yyyy');
```

Output:

EMPNO	ENAME	HIREDATE
-----	-----	-----
2001	Krishna	16-JUL-21

(Or)

```
SQL> Select empno, ename, hiredate from emp  
      where trunc(Hiredate) = trunc(Sysdate);
```

Output:

EMPNO	ENAME	HIREDATE
-----	-----	-----
2001	Krishna	16-JUL-21

Inserting a record with yesterday's date as hiredate:

```
SQL> Insert into emp(empno,ename,job,sal,hiredate)  
      values(2002,'Ganesh','MANAGER',15000,sysdate-1);
```

Output:

1 row created.

Displaying emp records who joined yesterday:

```
SQL> Select empno, ename, hiredate from emp where  
      to_char(Hiredate,'dd/mm/yyyy') = to_char(Sysdate-1,'dd/mm/yyyy');
```

Output:

EMPNO	ENAME	HIREDATE
-----	-----	-----
2002	Ganesh	15-JUL-21

(Or)

```
SQL> Select empno, ename, hiredate from emp  
2 where trunc(Hiredate) = trunc(Sysdate-1);
```

Output:

EMPNO	ENAME	HIREDATE
-----	-----	-----
2002	Ganesh	15-JUL-21

Inserting a record with one month ago's date:

```
SQL> Insert into emp(empno,ename,job,sal,hiredate)  
values(2002,'Ganesh','MANAGER',15000,Add_Months(sysdate,-1));
```

Output:

1 row created.

Displaying emp records who joined one month ago on same date:

```
SQL> Select empno, ename, hiredate from emp  
where to_char(hiredate,'dd/mm/yyyy') =  
to_char(Add_Months(sysdate,-1),'dd/mm/yyyy');
```

Output:

EMPNO	ENAME	HIREDATE
-----	-----	-----
2002	Ganesh	16-JUN-21

(Or)

```
SQL> Select empno, ename, hiredate from emp  
where trunc(hiredate) = trunc(Add_Months(sysdate,-1));
```

Output:

EMPNO	ENAME	HIREDATE
-----	-----	-----
2002	Ganesh	16-JUN-21

Inserting a record with one year ago's date:

```
SQL> Insert into emp(empno,ename,job,sal,hiredate)
   values(2003,'Sai','MANAGER',15000,Add_Months(sysdate,-12));
```

Output:

1 row created.

Displaying employee records who joined one year ago on same date:

```
SQL> Select empno, ename, hiredate from emp
   where to_char(hiredate,'dd/mm/yyyy') =
     to_char(Add_Months(sysdate,-12),'dd/mm/yyyy');
```

Output:

EMPNO	ENAME	HIREDATE
-----	-----	-----
2003	Sai	16-JUL-20

(Or)

```
SQL> Select empno, ename, hiredate from emp
   where trunc(hiredate) = trunc(Add_Months(sysdate,-12));
```

Output:

EMPNO	ENAME	HIREDATE
-----	-----	-----
2003	Sai	16-JUL-20

Displaying employee record who joined yesterday using 'Interval' Expression:

```
SQL> Select ename, hiredate from emp
   where trunc(hiredate) = trunc(sysdate-interval '1' day);
```

Output:

ENAME	HIREDATE
-----	-----
Ganesh	15-JUL-21

Displaying employee record who joined one month ago on same date using 'Interval' Expression:

SQL> Select ename, hiredate from emp
where trunc(hiredate) = trunc(sysdate-interval '1' month);

Output:

ENAME	HIREDATE
Ganesh	16-JUN-21

Displaying employee record who joined one month ago on same date using 'Interval' Expression:

SQL> Select ename, hiredate from emp
where trunc(hiredate) = trunc(sysdate-interval '1' year);

Output:

ENAME	HIREDATE
Sai	16-JUL-20

Example on Add_Months & Interval Expressions:

Create table with following Structure:

Sales

Dateld	Amount
--------	--------

Creating Sales Table:

SQL> Create table Sales(Dateld Date,
Amount Number(7,2));

Output:

Table created.

Inserting a record with today's date as DatelD:

SQL> Insert into Sales values(sysdate,10000);

Output:

1 row created.

Inserting a record with yesterday's date as DateID:

```
SQL> Insert into Sales values(sysdate-1,10000);
```

Output:

1 row created.

Inserting a record with 1 Month ago's date as DateID:

```
SQL> Insert into Sales values(sysdate-Interval '1' Month,10000);
```

Output:

1 row created.

Inserting a record with 1 Year ago's date as DateID:

```
SQL> Insert into Sales values(sysdate-Interval '1' year,10000);
```

Output:

1 row created.

Retrieving Sales Table records:

```
SQL> Select * from sales;
```

Output:

DATEID	AMOUNT
16-JUL-21	10000
15-JUL-21	10000
16-JUN-21	10000
16-JUL-20	10000

Displaying today's sales amount:

```
SQL> Select * from Sales where trunc(dateid) = trunc(sysdate);
```

Output:

DATEID	AMOUNT
16-JUL-21	10000

Displaying yesterday's sales amount:

```
SQL> Select * from Sales where trunc(dateid) = trunc(sysdate-1);
```

Output:

DATEID	AMOUNT
-----	-----
15-JUL-21	10000

Displaying one month ago same date Sales Amount:

```
SQL> Select * from Sales where trunc(dateid) = trunc(sysdate-interval '1' month);
```

Output:

DATEID	AMOUNT
-----	-----
16-JUN-21	10000

(Or)

```
SQL> Select * from Sales where trunc(dateid) = trunc(Add_Months(sysdate,-12));
```

Output:

DATEID	AMOUNT
-----	-----
16-JUL-20	10000

Displaying one year ago same date Sales Amount:

```
SQL> Select * from Sales  
      where trunc(dateid) = trunc(Add_Months(sysdate,-12));
```

Output:

DATEID	AMOUNT
-----	-----
16-JUL-20	10000

(Or)

SQL> Select * from Sales where trunc(dateid) = trunc(sysdate-interval '1' year);

Output:

DATEID	AMOUNT
-----	-----
16-JUL-20	10000

Example Appln on Add_Months():

Create a table with following structure:

Emp

Empno	DOB	Hiredate	Date_of_Retirement
-------	-----	----------	--------------------

Calculate Date of retirement based on DOB [Date of Birth]. 60 years is retirement age.

Creating a Table with the name “emp3”:

SQL> Create table emp3(
 Empno Number(4),
 DOB Date,
 Hiredate Date,
 Retirement_Date Date);

Output:

Table created.

Inserting records into “emp3” Table:

SQL> Insert into emp3(Empno,DOB,Hiredate) values(1001,'12-Jan-1995','01-JUN-2018');

Output:

1 row created.

```
SQL> Insert into emp3(Empno,DOB,Hiredate) values(1002,'12-Jan-2000','20-OCT-2020');
```

Output:

1 row created.

```
SQL> Insert into emp3(Empno,DOB,Hiredate) values(1003,'25-Dec-1990','20-AUG-2015');
```

Output:

1 row created.

Retrieving all records of emp3 Table:

```
SQL> Select * from emp3;
```

EMPNO	DOB	HIREDATE	RETIREMEN
1001	12-JAN-95	01-JUN-18	
1002	12-JAN-00	20-OCT-20	
1003	25-DEC-90	20-AUG-15	

Calculating Date of Retirement:

```
SQL> update emp3 set Retirement_Date = DOB + Interval '60' Year;
```

Output:

3 rows updated.

(Or)

```
SQL> Update emp3 set Retirement_Date = Add_Months(dob,60*12);
```

Output:

3 rows updated.

SQL> Select * from Emp3;

Output:

EMPNO	DOB	HIREDATE	RETIREMEN
1001	12-JAN-95	01-JUN-18	12-JAN-55
1002	12-JAN-00	20-OCT-20	12-JAN-60
1003	25-DEC-90	20-AUG-15	25-DEC-50

Example Application on Add_Months():

Create a Table with following structure:

IndiaCMs

State	CM_Name	Term_Start Date	Term_End Date
-------	---------	-----------------	---------------

Calculate CM Term Ending Date:

Creating “IndiaCMs” Table:

```
SQL> Create Table IndiaCMs(
      State Varchar2(15),
      CM_Name  Varchar2(20),
      Term_Start Date,
      Term_End Date);
```

Output:

Table created.

Inserting records into IndiaCMs Table:

```
SQL> Insert into IndiaCMs(State, CM_Name, Term_Start Date)
      values('Telangana','KCR','13-DEC-2018');
```

Output:

1 row created.

```
SQL> Insert into IndiaCMs(State, CM_Name, Term_Start Date)
      values('AP','YS JAGAN','30-MAY-2019');
```

Output:

1 row created.

Displaying all records:

```
SQL> Select * from IndiaCMs;
```

Output:

STATE	CM_NAME	TERM_STAR	TERM_ENDI
Telangana	KCR	13-DEC-18	
AP	YS JAGAN	30-MAY-19	

Calculating CM Term Ending Date:

```
SQL> Update IndiaCMs Set Term_Ending_Date =  
      Add_Months(Term_StartDate,5*12);
```

Output:

2 rows updated.

(Or)

```
SQL> Update IndiaCMs Set Term_Ending_Date =  
      Term_StartDate+Interval '5' Year;
```

Output:

2 rows updated.

Displaying all records:

```
SQL> Select * from IndiaCMs;
```

Output:

STATE	CM_NAME	TERM_STAR	TERM_ENDI
Telangana	KCR	13-DEC-18	13-DEC-23
AP	YS JAGAN	30-MAY-19	30-MAY-24

Example Queries on Last_Day():

Finding current month last date:

```
SQL> select last_day(sysdate) from dual;
```

Output:

```
LAST_DAY(
```

```
-----
```

```
31-JUL-21
```

Finding next month first date:

```
SQL> select last_day(sysdate)+1 from dual;
```

Output:

```
LAST_DAY(
```

```
-----
```

```
01-AUG-21
```

Finding previous month first date:

```
SQL> select last_day(add_months(sysdate,-2))+1 from dual;
```

Output:

```
LAST_DAY(
```

```
-----
```

```
01-JUN-21
```

Finding Previous month last date:

```
SQL> select last_day(add_months(sysdate,-1)) from dual;
```

Output:

```
LAST_DAY(
```

```
-----
```

```
30-JUN-21
```

Finding current month first date:

```
SQL> select last_day(add_months(sysdate,-1))+1 from dual;
```

Output:

```
LAST_DAY(
```

```
-----
```

```
01-JUL-21
```

Example Queries on Next_Day():**Finding next Monday date:**

```
SQL> Select next_day(sysdate,'mon') from dual;
```

Output:

```
NEXT_DAY(
```

```
-----
```

```
19-JUL-21
```

Finding this month last Monday date:

```
SQL> Select next_day(last_day(sysdate),'mon')-7 from dual;
```

Output:

```
NEXT_DAY(
```

```
-----
```

```
26-JUL-21
```

Finding next month first Monday date:

```
SQL> Select next_day(last_day(sysdate)+1,'mon') from dual;
```

Output:

```
NEXT_DAY(
```

```
-----
```

```
02-AUG-21
```

Finding current month first Monday date:

```
SQL> Select next_day(Last_Day(Add_Months(Sysdate,-2)+1),'mon') from dual;
```

Output:

```
NEXT_DAY(-----
```

```
07-JUN-21
```

Example queries on Months_Between():**Finding experience of employees:**

```
SQL> Select ename, Round(Months_Between(sysdate,hiredate)/12) Experience from emp;
```

Output:

ENAME	EXPERIENCE
-------	------------

SMITH	41
ALLEN	40
WARD	40
JONES	40
MARTIN	40
BLAKE	40
CLARK	40
SCOTT	39
KING	40

Finding Sachin's Age:

```
SQL> Select Trunc(Months_Between(Sysdate,'24-APR1973')/12) AGE from dual;
```

Output:

AGE

48

Finding age in the form of years and months:

```
SQL> Select Trunc(Months_Between(Sysdate,'24-APR-1973')/12) Years,  
      Mod(Trunc(Months_Between(Sysdate,'24-APR-1973'))),12) Months  
    from dual;
```

Output:

YEARS	MONTHS
-----	-----
48	2

Finding employee experience in the form of years and months:

```
SQL> Select Trunc(Months_Between(sysdate,hiredate)/12) Years,  
      Mod(Trunc(Months_Between(sysdate,hiredate))),12) Months  
    from emp;
```

Output:

YEARS	MONTHS
-----	-----
40	7
40	4
40	4
40	3
39	9
40	2
40	1
38	7
39	8
39	10

Example Queries on Miscellaneous Functions:**Displaying current user name:**

```
SQL> Select User from dual;
```

Output:

```
USER
```

```
C##ORACLE7AM
```

Displaying Current User ID:

SQL> Select UID from dual;

Output:

UID

111

Finding greatest in 3 numbers:

SQL> Select Greatest(100,200,300) from dual;

Output:

GREATEST(100,200,300)

300

Finding least in 3 numbers:

SQL> Select Least(100,200,300) from dual;

Output:

LEAST(100,200,300)

100

Finding size of the string:

SQL> Select VSize('raju'), VSize('sai') from dual;

Output:

VSIZE('RAJU') VSIZE('SAI')

----- -----

4 3

Examples on NVL() Function:

Replacing not null with same value:

SQL> Select NVL(100,200) from dual;

Output:

NVL(100,200)

100

Replacing null with 200:

SQL> Select NVL(NULL,200) from dual;

Output:

NVL(NULL,200)

200

Calculating null + 200:

SQL> Select null+200 from dual;

Output:

NULL+200

Note: null + something = null

Replacing null with 100:

SQL> Select NVL(null,100)+200 from dual;

Output:

NVL(NULL,100)+200

300

Calculating Total Salary of Employees:

SQL> Select ename,sal,comm,sal+NVL(comm,0) as "Total Salary"
from emp;

Output:

ENAME	SAL	COMM	Total Salary
SMITH	800		800
ALLEN	1600	300	1900
WARD	1250	500	1750
JONES	2975		2975
MARTIN	1250	1400	2650

Replacing null value with N/A [Not Applicable]:

SQL> Select ename, sal, NVL(to_char(comm),'N/A') comm from emp;

Output:

ENAME	SAL	COMM
SMITH	800	N/A
ALLEN	1600	300
WARD	1250	500
JONES	2975	N/A
MARTIN	1250	1400
BLAKE	2850	N/A

Create a Table as following:

SQL> Select * from student;

Output:

STID	SNAME	M1	M2	M3
2001	aa	55	77	
1001	Ravi	40	80	60
1002	Sravan	66	44	77

Replacing null with ABSENT:

SQL> Select sname, NVL(to_char(m3),'ABSENT') M3 from student;

Output:

SNAME	M3
aa	ABSENT
Ravi	60
Sravan	77

Examples on NVL2() Function:**Replacing 100 with 200:**

SQL> Select NVL2(100,200,300) from dual;

Output:

NVL2(100,200,300)

200

Replacing null with 300:

SQL> Select NVL2(NULL,200,300) from dual;

Output:

NVL2(NULL,200,300)

300

Setting commission as 1000 to the employees whose commission is null and increase 200 to the employees whose commission is not null:

SQL> Select Ename, NVL2(Comm,comm+200,1000) Comm from emp;

Output:

ENAME	COMM
SMITH	1000

ALLEN	500
WARD	700
JONES	1000
MARTIN	1600
BLAKE	1000
CLARK	1000

Displaying ranks according to salary in descending order using rank():

SQL> Select empno, ename, sal, rank() over(order by sal desc) Rank from emp;

Output:

EMPNO	ENAME	SAL	RANK
7839	KING	5000	1
7902	FORD	3000	2
7788	SCOTT	3000	2
7566	JONES	2975	4
7698	BLAKE	2850	5
7782	CLARK	2450	6
7499	ALLEN	1600	7
7844	TURNER	1500	8
7934	MILLER	1300	9
7521	WARD	1250	10
7654	MARTIN	1250	10
7876	ADAMS	1100	12
7900	JAMES	950	13
7369	SMITH	800	14

Displaying ranks according to salary in descending order using dense_rank():

SQL> Select empno, ename, sal, dense_rank() over(order by sal desc) Rank from emp;

Output:

EMPNO	ENAME	SAL	RANK
7839	KING	5000	1

7902 FORD	3000	2
7788 SCOTT	3000	2
7566 JONES	2975	3
7698 BLAKE	2850	4
7782 CLARK	2450	5
7499 ALLEN	1600	6
7844 TURNER	1500	7
7934 MILLER	1300	8
7521 WARD	1250	9
7654 MARTIN	1250	9
7876 ADAMS	1100	10
7900 JAMES	950	11
7369 SMITH	800	12

Displaying ranks according to job descending order. If salary is same arrange in ascending order according to experience:

SQL> Select ename, Hiredate, sal, dense_rank() over(order by sal desc, hiredate asc) Rank from emp;

Output:

ENAME	HIREDATE	SAL	RANK
KING	17-NOV-81	5000	1
FORD	03-DEC-81	3000	2
SCOTT	09-DEC-82	3000	3
JONES	02-APR-81	2975	4
BLAKE	01-MAY-81	2850	5
CLARK	09-JUN-81	2450	6
ALLEN	20-FEB-81	1600	7
TURNER	08-SEP-81	1500	8
MILLER	23-JAN-82	1300	9
WARD	22-FEB-81	1250	10
MARTIN	28-SEP-81	1250	11
ADAMS	12-JAN-83	1100	12
JAMES	03-DEC-81	950	13
SMITH	17-DEC-80	800	14

Interval Expressions & Case Expressions

Interval Expressions:

- Interval Expressions are introduced in “Oracle 9i”.
- Interval expressions are used to add/subtract the days, months, and years to a date or from a date.
- Interval expressions are also used to add/subtract the hours, minutes, and seconds to a time or from a time.

Examples on Interval Expressions:

Adding 2 days to system date:

```
SQL> Select sysdate + interval '2' Day from dual;
```

Output:

```
SYSDATE+I
```

```
-----
```

```
22-JUL-21
```

Adding 2 months to system date:

```
SQL> Select sysdate + interval '2' Month from dual;
```

Output:

```
SYSDATE+I
```

```
-----
```

```
20-SEP-21
```

Adding 2 years to system date:

```
SQL> Select sysdate + interval '2' Year from dual;
```

Output:

```
SYSDATE+I
```

```
-----
```

```
20-JUL-23
```

Adding 1 year 6 months to system date:

```
SQL> Select sysdate + interval '1-6' Year to Month from dual;
```

Output:

```
SYSDATE+I
```

```
-----
```

```
20-JAN-23
```

Subtracting 2 days from system date:

```
SQL> Select sysdate - interval '2' Day from dual;
```

Output:

```
SYSDATE-I
```

```
-----
```

```
18-JUL-21
```

Subtracting 2 months from system date:

```
SQL> Select sysdate - interval '2' Month from dual;
```

Output:

```
SYSDATE-I
```

```
-----
```

```
20-MAY-21
```

Subtracting 2 years from system date:

```
SQL> Select sysdate - interval '2' Year from dual;
```

Output:

```
SYSDATE-I
```

```
-----
```

```
20-JUL-19
```

Subtracting 1 Year 6 Months from system date:

```
SQL> Select sysdate - interval '1-6' Year to Month from dual;
```

Output:

```
SYSDATE-I
```

```
-----
```

```
20-JAN-20
```

Adding 3 days to a specific date:

```
SQL> Select to_date('20-Dec-2020') + Interval '3' Day from dual;
```

Output:

```
TO_DATE('
```

```
-----
```

```
23-DEC-20
```

Adding 3 months to a specific date:

```
SQL> Select to_date('20-Dec-2020') + Interval '3' Month from dual;
```

Output:

```
TO_DATE('
```

```
-----
```

```
20-MAR-21
```

Adding 3 years to a specific date:

```
SQL> Select to_date('20-Dec-2020') + Interval '3' Year from dual;
```

Output:

```
TO_DATE('
```

```
-----
```

```
20-DEC-23
```

Adding 1 year 2 months to a specific date:

```
SQL> Select to_date('20-Dec-2020') + Interval '1-2' Year to Month from dual;
```

Output:

```
TO_DATE('
```

```
-----  
20-FEB-22
```

Adding 2 Hours to system time:

```
SQL> Select systimestamp + interval '2' Hour from dual;
```

Output:

```
SYSTIMESTAMP+INTERVAL'2' HOUR
```

```
-----  
20-JUL-21 09.56.55.507000000 AM +05:30
```

Adding 2 Minutes to system time:

```
SQL> Select systimestamp + interval '2' Minute from dual;
```

Output:

```
SYSTIMESTAMP+INTERVAL'2' MINUTE
```

```
-----  
20-JUL-21 07.59.15.396000000 AM +05:30
```

Adding 2 Minutess to system time:

```
SQL> Select systimestamp + interval '2' Second from dual;
```

Output:

```
SYSTIMESTAMP+INTERVAL'2' SECOND
```

```
-----  
20-JUL-21 07.57.25.627000000 AM +05:30
```

Adding 1 hour 30 minutes to system time:

SQL> Select systimestamp + interval '1:30' Hour to Minute from dual;

Output:

SYSTIMESTAMP+INTERVAL'1:30' HOURTOMINUTE

20-JUL-21 09.28.18.530000000 AM +05:30

Subtracting 2 hours time from system time:

SQL> Select systimestamp - interval '2' Hour from dual;

Output:

SYSTIMESTAMP-INTERVAL'2' HOUR

20-JUL-21 06.01.52.627000000 AM +05:30

Subtracting 2 minutes time from system time:

SQL> Select systimestamp - interval '2' Minute from dual;

Output:

SYSTIMESTAMP-INTERVAL'2' MINUTE

20-JUL-21 08.00.00.867000000 AM +05:30

CASE Expressions:

- Case Expressions are introduced in “Oracle 9i”.
- Case Expressions are used to implement “If-Then-Else” in SQL Query.
- It avoids writing a separate procedure to implement “If-Then-Else”. In SQL Query we can implement “If-Then-Else”.

CASE Expression can be used in 2 ways. They are:

1. Simple Case Expression
2. Searched Case Expression

Simple Case Expression:

Syntax:

```
Case <Column_Name>
When <Value> Then <Return_Expression>
[When <Value> Then <Return_Expression>]
[.....]
[Else <Return_Expression>]
End
```

“Simple Case” is used to return the expressions based on value matching. When the “value” is matched with “column value”, it returns the “Return_Expression”. When all values are not matched with column values, “Return_Expression” in Else will be returned. Writing “Else” is optional.

Searched Case Expression:

Syntax:

```
Case
When <Condition> Then <Return_Expression>
[When <Condition> Then <Return_Expression>]
[.....]
[Else <Return_Expression>]
End
```

“Searched Case” is used to return the expressions based on the condition. When the “Condition” is TRUE, it returns the “Return_Expression”. When all conditions are FALSE, “Return_Expression” in Else will be returned. Writing “Else” is optional.

Note:

“Simple Case” can check equality condition only.

“Searched Case” can check any type of condition.

Examples on Simple Case Expression:

Displaying clerk as worker, manager as boss, president as big boss and others as employee:

SQL> Select ename,

Case Job

When 'CLERK' then 'WORKER'

When 'MANAGER' then 'BOSS'

When 'PRESIDENT' then 'BIG BOSS'

Else 'EMPLOYEE'

End Job

from Emp;

Output:

ENAME	JOB
SMITH	WORKER
ALLEN	EMPLOYEE
JONES	BOSS
KING	BIG BOSS

Updating salary of employees department wise as follows:

10 Dept => increase 10%

20 Dept => increase 20%

30 Dept => increase 15%

Other Depts => 5%

SQL> 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;

Output:

14 rows updated.

Examples on Searched Case Expression:

Displaying salary range as follows:

If sal>3000 then display as high salary

If sal<3000 then display as low salary

If sal=3000 then display as avrg salary

```
SQL> Select Ename, Sal,
CASE
When sal>3000 Then 'High Salary'
When sal<3000 Then 'Low Salary'
When sal=3000 Then 'Avrg Salary'
End As "Salary Range"
From Emp;
```

Output:

ENAME	SAL	Salary Rang
SMITH	800	Low Salary
ALLEN	1600	Low Salary
SCOTT	3000	Avrg Salary
KING	5000	High Salary

Create a table student with following data:

```
SQL> Select * from student;
```

STID	SNAME	M1	M2	M3
2001	srinu	55	77	25
1001	Ravi	40	80	60
1002	Sravan	66	44	77

Finding result of a student if subject marks are <40 in any subject result is fail. In every subject if marks are >=40 then result is pass:

```
SQL> Select Sname,
      CASE
        When M1>=40 and M2>=40 and M3>=40 Then 'PASS'
        ELSE 'FAIL'
      END RESULT
      From Student;
```

Output:

SNAME	RESU
srinu	FAIL
Ravi	PASS
Sravan	PASS

Decode() Function:

It is used to implement the 'If-Then-Else' in SQL Statement.

Syntax:

```
Decode(<Column>,
       <Value>,<Return_Expression>
       [<Value>,<Return_Expression>,<Value>,<Return_Expression>,<Else_Expression>])
```

In Oracle, 'If-Then-Else' can be implemented in 2 ways. They are:

- Using CASE Expression
- Using DECODE() Function

CASE Expression introduced in Oracle 9i.

Before Oracle 9i we were using DECODE() Function

Displaying clerk as worker, manager as boss, president as big boss and others as employee:

SQL> Select Ename,

```
    DECODE(JOB,  
    'MANAGER','BOSS',  
    'PRESIDENT','BIG BOSS',  
    'CLERK','WORKER',  
    'EMPLOYEE') Job  
From Emp;
```

Output:

ENAME	JOB
SMITH	WORKER
ALLEN	EMPLOYEE
WARD	EMPLOYEE
JONES	BOSS
KING	BIG BOSS

Updating salary of employees department wise as follows:

10 Dept => increase 10%

20 Dept => increase 20%

30 Dept => increase 15%

Other Depts => 5%

SQL> 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);

Output:

14 rows updated.

Note:

Decode() Function can check equality only. It cannot check other conditions.

Group By and Having Clauses in SQL

Example Queries on “Group By” Clause:

Displaying dept wise sum of salaries:

```
SQL> Select Deptno, Sum(Sal)
      from Emp
      Group By Deptno
      Order By Deptno;
```

Output:

DEPTNO	SUM(SAL)
10	8750
20	10875
30	9400

Displaying dept wise average of salaries:

```
SQL> Select Deptno, Round(Avg(Sal),2)
      from Emp
      Group By Deptno
      Order By Deptno;
```

Output:

DEPTNO	ROUND(AVG(SAL),2)
10	2916.67
20	2175
30	1566.67

Displaying Dept wise Maximum and Minimum Salary:

```
SQL> Select Deptno, Max(Sal), Min(Sal)
      from Emp
      Group By Deptno
      Order By Deptno;
```

Output:

DEPTNO	MAX(SAL)	MIN(SAL)
10	5000	1300
20	3000	800
30	2850	950

Displaying number of employees in each department:

```
SQL> Select Deptno, Count(*)  
      from Emp  
     Group By Deptno  
    Order By Deptno;
```

Output:

DEPTNO	COUNT(*)
10	3
20	5
30	6

Displaying number of employees joined in different years:

```
SQL> Select to_char(hiredate,'yyyy') Year, count(*) Number_of_Emps  
      from Emp  
     Group By to_char(hiredate,'yyyy')  
    Order By Year;
```

Output:

YEAR	NUMBER_OF_EMPS
1980	1
1981	10
1982	2
1983	1

Displaying number of employees joined on different weekdays:

```
SQL> Select to_char(hiredate,'day') weekday, count(*) Number_of_Emps  
2 from Emp  
3 Group By to_char(hiredate,'day');
```

Output:

WEEKDAY	NUMBER_OF_EMPS
thursday	4
wednesday	2
friday	2
sunday	1
monday	1
tuesday	3
saturday	1

Displaying Job wise sum of salaries:

```
SQL> Select Job, Sum(Sal)  
      from Emp  
     Group By Job;
```

Output:

JOB	SUM(SAL)
CLERK	4150
SALESMAN	5600
ANALYST	6000
MANAGER	8275
PRESIDENT	5000

Displaying Job wise Average Salary:

```
SQL> Select Job, Round(Avg(Sal))  
      from Emp  
     Group By Job;
```

Output:JOB ROUND(AVG(SAL))

CLERK	1038
SALESMAN	1400
ANALYST	3000
MANAGER	2758
PRESIDENT	5000

Displaying Job wise Maximum salary and Minimum salary:

```
SQL> Select Job, Max(Sal), Min(Sal)  
      from Emp  
     Group By Job;
```

Output:JOB MAX(SAL) MIN(SAL)

CLERK	1300	800
SALESMAN	1600	1250
ANALYST	3000	3000
MANAGER	2975	2450
PRESIDENT	5000	5000

Displaying dept wise number of employees:

```
SQL> Select Job, Count(*)  
      from Emp  
     Group By Job;
```

Output:JOB COUNT(*)

CLERK	4
SALESMAN	4
ANALYST	2
MANAGER	3
PRESIDENT	1

Displaying Dept wise sum of salaries of dept nos 10 and 30:

```
SQL> Select Deptno, Sum(Sal)
      from Emp
      where deptno in(10,30)
      Group By Deptno
      Order By Deptno;
```

Output:

DEPTNO	SUM(SAL)
10	8750
30	9400

Displaying Job wise sum of salaries for managers and clerks:

```
SQL> Select Job, Sum(Sal)
      from Emp
      where Job In('CLERK','MANAGER')
      group by Job;
```

Output:

JOB	SUM(SAL)
CLERK	4150
MANAGER	8275

Example Queries on “Having” Clause:**Displaying the deptnos which are having more than 3 employees:**

```
SQL> Select Deptno, CCount(*)
      From Emp
      Group BY Deptno
      Having Count(*)>3;
```

Output:

DEPTNO	COUNT(*)
--------	----------

30	6
20	5

Displaying the deptnos which are having 3 employees:

```
SQL> Select Deptno, COunt(*)  
      From EMp  
      Group BY Deptno  
      Having Count(*)=3;
```

Output:

DEPTNO	COUNT(*)
--------	----------

10	3
----	---

Displaying the Job Titles which are having more than 3 employees:

```
SQL> Select Job, Count(*)  
      From Emp  
      Group By Job  
      Having Count(*)>3;
```

Output:

JOB	COUNT(*)
-----	----------

CLERK	4
SALESMAN	4

Displaying the Job Titles on which organization is spending more than 5000:

```
SQL> Select Job, Sum(Sal)  
      From Emp  
      Group By Job  
      Having Sum(Sal)>5000;
```

Output:

JOB	SUM(SAL)
-----	-----
SALESMAN	5600
ANALYST	6000
MANAGER	8275

Displaying the deptno which is spending maximum amount on employees:

```
SQL> Select Deptno from emp  
      Group By Deptno  
      Having sum(Sal) = (Select max(sum(Sal)) from emp group by deptno);
```

Output:

DEPTNO

20

Displaying the deptno which is having maximum number of employees:

```
SQL> Select deptno from emp  
      Group By deptno  
      Having count(*) = (Select max(count(*)) from emp group by deptno);
```

Output:

DEPTNO

30

Displaying the job title on which organization is spending maximum amount:

```
SQL> Select job from emp group by job  
      Having sum(Sal) = (Select max(sum(Sal)) from emp group by job);
```

Output:

JOB

MANAGER

Displaying Job titles which are having maximum number of employees:

SQL> Select Job from emp group by Job

Having Count(*)=(Select max(count(*)) from emp group by Job);

Output:

JOB

CLERK

SALESMAN

Example Application on Group By and Having Clauses:

Create a Table with following structure & insert some records:

StudentID	CName	Fee
-----------	-------	-----

Displaying “Course” Table records:

SQL> Select * from Course;

SID	CNAME	FEE
1001	Java	6000
1001	Oracle	5000
1002	Oracle	5000
1002	Python	8000
1003	Python	8000
1004	Java	6000
1005	C	4000
1006	C	4000
1007	C	4000
1008	C	4000

Displaying Course wise Number of Students:

```
SQL> Select CName, Count(*)  
      from Course  
      Group By CName;
```

Output:

CNAME	COUNT(*)
Oracle	2
C	4
Java	2
Python	2

Displaying Course Titles which are having more than 2 students:

```
SQL> Select CName, Count(*)  
      From Course  
      Group By CName  
      Having Count(*)>2;
```

Output:

CNAME	COUNT(*)
C	4

Displaying the amount collected on each course:

```
SQL> Select CName, Sum(Fee)  
      From Course  
      Group By CName;
```

Output:

CNAME	SUM(FEE)
Oracle	10000
C	16000
Java	12000
Python	16000

Display the courses which are collected less than 15000:

```
SQL> Select CName, Sum(Fee)  
      From Course  
     Group By CName  
    Having Sum(Fee)<15000;
```

Output:

CNAME	SUM(FEE)
-------	----------

Oracle	10000
Java	12000

Display the course on which minimum amount collected:

```
SQL> Select CName  From Course  Group By CName  
      Having Sum(Fee) = (Select Min(Sum(Fee)) from Course Group By  
                           CName);
```

Output:

CNAME

Oracle

Example Queries on Grouping Records using Multiple Tables:

Display Dept wise, with in Dept Job wise sum of salaries:

```
SQL> break on deptno skip 1
```

```
SQL> Select Deptno, Job, Sum(Sal)  
      From Emp  
     Group By Deptno,Job  
    Order By Deptno;
```

Output:

DEPTNO	JOB	SUM(SAL)
10	CLERK	1300
	MANAGER	2450
	PRESIDENT	5000
20	ANALYST	6000
	CLERK	1900
	MANAGER	2975
30	CLERK	950
	MANAGER	2850
	SALESMAN	5600

Calculating Sub Total & Grand Total using Rollup() Function:

SQL> Select Deptno, Job, Sum(Sal) From Emp

Group By Rollup(Deptno,Job)

Order By Deptno;

Output:

DEPTNO	JOB	SUM(SAL)
10	CLERK	1300
	MANAGER	2450
	PRESIDENT	5000
20		8750
	ANALYST	6000
	CLERK	1900
30	MANAGER	2975
		10875
	CLERK	950
	MANAGER	2850
	SALESMAN	5600
		9400
		29025

10th Dept Sub Total

20th Dept Sub Total

30th Dept Sub Total

Grand Total

Calculating Sub Total & Grand Total using Cube() Function:

SQL> Select Deptno, Job, Sum(Sal)

```
From Emp
Group By Cube(Deptno,Job)
Order By Deptno;
```

Output:

DEPTNO	JOB	SUM(SAL)
10	CLERK	1300
	MANAGER	2450
	PRESIDENT	5000
		8750
		8750
		→ 10 th Dept Sub Total
20	ANALYST	6000
	CLERK	1900
	MANAGER	2975
		10875
		10875
		→ 20 th Dept Sub Total
30	CLERK	950
	MANAGER	2850
	SALESMAN	5600
		9400
		9400
		→ 30 th Dept Sub Total
	ANALYST	6000
	CLERK	4150
	MANAGER	8275
	PRESIDENT	5000
	SALESMAN	5600
		6000
		4150
		8275
		5000
		5600
		→ Analyst Sub Total
		→ Clerk Sub Total
		→ Manager Sub Total
		→ President Sub Total
		→ Salesman Sub Total
		29025
		→ Grand Total

Examples on Grouping records using multiple columns:

Displaying Year wise, with in year quarter wise number of employees joined in organization:

SQL> break on year skip 1

```
SQL> Select to_char(hiredate,'yyyy') Year, to_char(hiredate,'q') Qrtr,
  Count(*) "Number of Emps"
  From Emp
  Group By to_char(hiredate,'yyyy'), to_char(hiredate,'q')
  Order By Year;
```

Output:

YEAR	Q	Number of Emps
1980	4	1
1981	1	2
	2	3
	3	2
	4	3
1982	1	1
	4	1
1983	1	1

Displaying Year wise, with in year quarter wise number of employees joined in organization and calculate sub totals and grand total according to year:

```
SQL> Select to_char(hiredate,'yyyy') Year, to_char(hiredate,'q') Qrtr,
  Count(*) "Number of Emps"
  From Emp
  Group By rollup(to_char(hiredate,'yyyy'), to_char(hiredate,'q'))
  Order By Year;
```

Output:

YEAR	Q	Number of Emps	
1980	4	1	
1981	1	2	
	2	3	
	3	2	
	4	3	
1982	1	1	
	4	1	
1983	1	1	
	14		

The diagram illustrates the calculation of subtotals and a grand total from the provided output. Red circles highlight specific values: '1' for 1980 Q4, '10' for 1981 total, '2' for 1982 Q4, '1' for 1983 Q1, and '14' for the overall grand total. Arrows point from these circled values to boxes labeled '1980 Sub Total', '1981 Sub Total', '1982 Sub Total', '1983 Sub Total', and 'Grand Total' respectively.

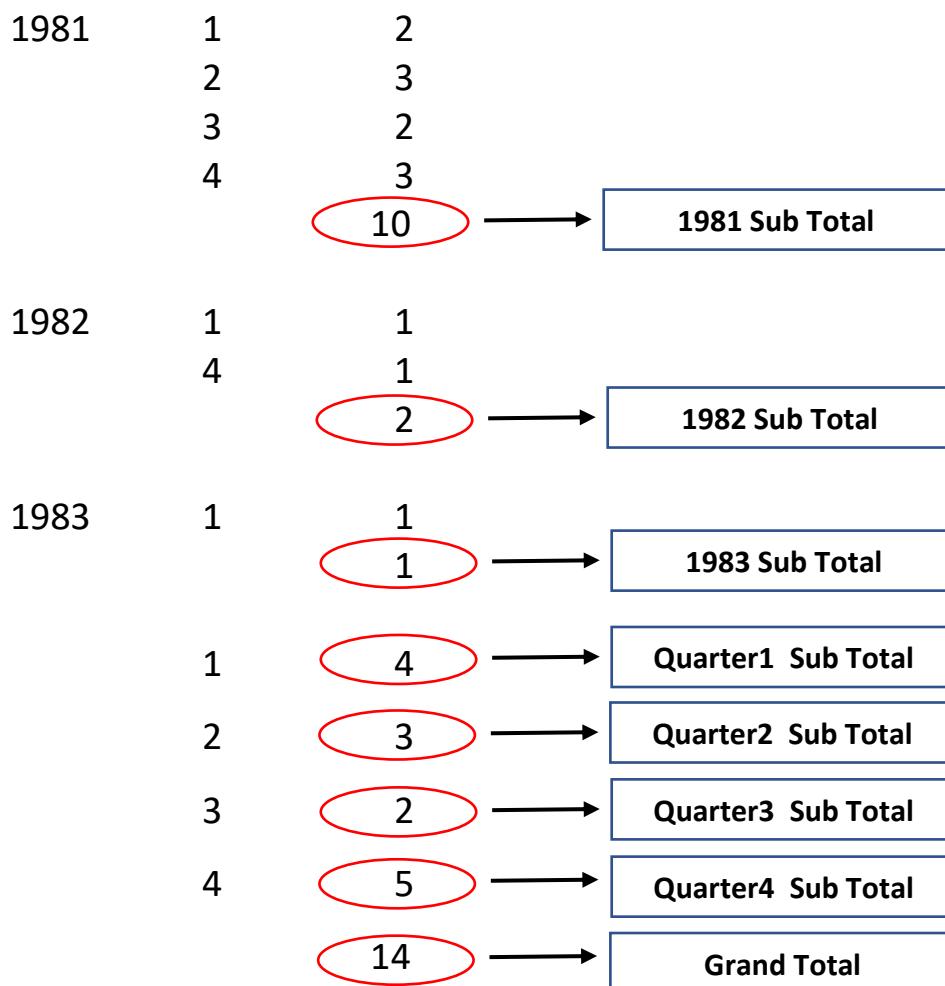
Displaying Year wise, with in year quarter wise number of employees joined in organization and calculate sub totals and grand total according to year and quarter:

```
SQL> Select to_char(hiredate,'yyyy') Year, to_char(hiredate,'q') Qrtr,
  Count(*) "Number of Emps"
  From Emp
  Group By cube(to_char(hiredate,'yyyy'), to_char(hiredate,'q'))
  Order By Year;
```

Output:

YEAR	Q	Number of Emps	
1980	4	1	
		1	

The diagram illustrates the calculation of subtotals and a grand total from the provided output. Red circles highlight the value '1'. An arrow points from this circled value to a box labeled '1980 Sub Total'.



Create a table with the name **sales** and enter 3 years sales as following:

Retrieving “sales” table records:

SQL> Select * from sales;

DATEID	AMOUNT
01-JAN-15	5000
02-JAN-15	5000
03-JAN-15	10000
01-APR-15	8000
02-APR-15	6000
01-MAY-15	6000

01-JUN-15	6000
01-JUL-15	6000
02-JUL-15	7000
02-NOV-15	9000
02-DEC-15	9000
01-JAN-16	4000
02-JAN-16	3000
01-MAY-16	3000
02-MAY-16	6000
01-DEC-16	3000
01-AUG-16	3000
02-AUG-16	2000
01-JAN-17	2000
02-JAN-17	8000
02-MAY-17	8000
03-MAY-17	7000
01-AUG-17	7000
02-AUG-17	5000
01-DEC-17	5000

Displaying year wise, with in year quarter wise sales:

```
SQL> Select to_char(DateID,'yyyy') Year, to_char(DateID,'Q') Qrtr,  
      Sum(Amount) from Sales  
      Group By to_char(DateID,'yyyy'), to_char(DateID,'Q')  
      Order By 1;
```

Output:

YEAR	Q	SUM(AMOUNT)
2015	-	-----
2015	1	20000
	2	26000
	3	13000
	4	18000
2016	1	7000
	2	9000
	3	5000
	4	3000

2017	1	10000
	2	15000
	3	12000
	4	5000

Displaying year wise, with in year quarter wise sales and Calculate sub totals and grand total according to year:

```
SQL> Select to_char(DateID,'yyyy') Year, to_char(DateID,'Q') Qrtr,
      Sum(Amount)
     from Sales
    Group By Rollup(to_char(DateID,'yyyy'), to_char(DateID,'Q'))
   Order By 1;
```

Output:

YEAR	Q	SUM(AMOUNT)
------	---	-------------

---	-	-----
-----	---	-------

2015	1	20000
	2	26000
	3	13000
	4	18000

77000 → **2015 Sub Total**

2016	1	7000
	2	9000
	3	5000
	4	3000

24000 → **2016 Sub Total**

2017	1	10000
	2	15000
	3	12000
	4	5000

42000 → **2017 Sub Total**

143000 → **Grand Total**

Displaying year wise, with in year quarter wise sales and Calculate sub totals and grand total according to year and quarter:

```
SQL> Select to_char(DateID,'yyyy') Year, to_char(DateID,'Q') Qrtr,
      Sum(Amount)
    from Sales
   Group By Cube(to_char(DateID,'yyyy'), to_char(DateID,'Q'))
  Order By 1;
```

Output:

YEAR	Q	SUM(AMOUNT)
------	---	-------------

---	-	-----
-----	---	-------

2015	1	20000
	2	26000
	3	13000
	4	18000
		77000

2016	1	7000
	2	9000
	3	5000
	4	3000
		24000

2017	1	10000
	2	15000
	3	12000
	4	5000
		42000

1	37000	→	Quarter1 Sub Total
---	-------	---	--------------------

2	50000	→	Quarter2 Sub Total
---	-------	---	--------------------

3	30000	→	Quarter3 Sub Total
---	-------	---	--------------------

4	26000	→	Quarter4 Sub Total
---	-------	---	--------------------

143000	→	Grand Total
--------	---	-------------

Create a Table with the name “Person” and enter the data as following:

Retrieving “person” table records:

SQL> Select * from person;

Output:

PID	PNAME	STATE	G	AGE	AADHARNO
1	aa	Telangana	M	45	123456
2	bb	Telangana	M	62	123457
3	cc	Telangana	F	48	123458
4	cc	Telangana	F	65	123459
5	cc	Telangana	M	32	123460
6	dd	Telangana	F	35	123461
7	dd	Telangana	F	23	123462
8	dd	Telangana	M	25	123463
9	ee	Telangana	M	28	123464
10	ff	Gujarat	M	65	123465
11	gg	Gujarat	M	62	123466
12	gg	Gujarat	M	45	123467
12	gg	Gujarat	F	41	123468
13	hh	Gujarat	F	45	123469
14	hh	Gujarat	F	23	123470
14	hh	Gujarat	F	35	123471
15	hh	Gujarat	M	36	123472

Displaying state wise, with in state gender wise number of people:

SQL> break on state skip 1

SQL> Select State, Gender, Count(*)
 2 from Person
 3 Group By State, Gender
 4 Order By State;

Output:

STATE	G	COUNT(*)
Gujarat	F	4
	M	4
Telangana	F	4
	M	5

Displaying state wise, with in state gender wise number of people and calculate subtotals and grand total according to state:

```
SQL> Select State, Gender, Count(*)
      from Person
      Group By Rollup(State, Gender)
      Order By State;
```

Output:

STATE	G	COUNT(*)
Gujarat	F	4
	M	4
		8
Telangana	F	4
	M	5
		9
		17

Displaying state wise, with in state gender wise number of people and calculate subtotals and grand total according to state and gender:

```
Select State, Gender, Count(*)
      from Person
      Group By Cube(State, Gender)
      Order By State;
```

Output:

```
SQL> Select State, Gender, Count(*)
  2 from Person
  3 Group By Cube(State, Gender)
  4 Order By State;
```

STATE	G	COUNT(*)	
Gujarat	-	-----	
	F	4	
	M	4	
		8	→ Gujarat Sub Total
Telangana	F	4	
	M	5	
		9	→ Telangana Sub Total
	F	8	→ Female Sub Total
	M	9	→ Male Sub Total
		17	→ Grand Total

Displaying State wise, age range wise, gender wise number of people:

```
SQL> Select State, CASE
      when age between 1 and 20 then '1 to 20'
      when age between 21 and 40 then '21 to 40'
      when age between 41 and 60 then '41 to 60'
      Else 'Above 60'
    End "Age Range", Gender, Count(*) from Person
    Group By State,
    CASE
      when age between 1 and 20 then '1 to 20'
      when age between 21 and 40 then '21 to 40'
      when age between 41 and 60 then '41 to 60'
      Else 'Above 60'
    End, Gender
    Order By State;
```

Output:

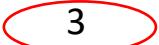
STATE	Age Rang	G	COUNT(*)
-----	-----	-	-----
Gujarat	21 to 40	F	2
		M	1
	41 to 60	F	2
		M	1
	Above 60	M	2
Telangana	21 to 40	F	2
		M	3
	41 to 60	F	1
		M	1
	Above 60	F	1
		M	1

Displaying State wise, Age range wise, gender wise number of people and calculate sub totals & grand total according to state:

```
SQL> Select State, CASE
      when age between 1 and 20 then '1 to 20'
      when age between 21 and 40 then '21 to 40'
      when age between 41 and 60 then '41 to 60'
      Else 'Above 60'
    End "Age Range", Gender, Count(*) from Person
    Group By Rollup(State,
    CASE
      when age between 1 and 20 then '1 to 20'
      when age between 21 and 40 then '21 to 40'
      when age between 41 and 60 then '41 to 60'
      Else 'Above 60'
    End, Gender)
    Order By State;
```

Output:

STATE	Age Rang	G	COUNT(*)
Gujarat	21 to 40	F	2
		M	1
			3
			3
	41 to 60	F	2
		M	1
			3
	Above 60	M	2
			2
			2
			8
Telangana	21 to 40	F	2
		M	3
			5
	41 to 60	F	1
		M	1
			2
	Above 60	F	1
		M	1
			2
			2
			9
			9
			17

 → **Gujarat 21 to 40 age Sub Total**
 → **Gujarat 41 to 60 age Sub Total**
 → **Gujarat above 60 age Sub Total**
 → **Gujarat Sub Total**
 → **Telangana 21 to 40 age Sub Total**
 → **Telangana 41 to 60 age Sub Total**
 → **Telangana Above 60 age Sub Total**
 → **Telangana Sub Total**
 → **Grand Total**

Displaying State wise, Age range wise, gender wise number of people and calculate sub totals & grand total according to state, age range and gender:

```
SQL> Select State,
CASE
when age between 1 and 20 then '1 to 20'
when age between 21 and 40 then '21 to 40'
when age between 41 and 60 then '41 to 60'
Else 'Above 60'
End "Age Range", Gender, Count(*)
from Person
Group By Cube(State,
CASE
when age between 1 and 20 then '1 to 20'
when age between 21 and 40 then '21 to 40'
when age between 41 and 60 then '41 to 60'
Else 'Above 60'
End, Gender)
Order By State;
```

Output:

STATE	Age Rang	G	COUNT(*)
Gujarat	21 to 40	F	2
		M	1
			3
			3 → Gujarat 21 to 40 age Sub Total
	41 to 60	F	2
		M	1
			3
			3 → Gujarat 41 to 60 age Sub Total
	Above 60	M	2
			2
			2 → Gujarat above 60 age Sub Total
		F	4
			4
			4 → Gujarat Female Sub Total
		M	4
			4
			4 → Gujarat Male Sub Total
			8
			8 → Gujarat Sub Total

Telangana 21 to 40

F 2

M 3

5

Telangana 21 to 40 age Sub Total

41 to 60

F 1

M 1

2

Telangana 41 to 60 age Sub Total

Above 60

F 1

M 1

2

Telangana Above 60 age Sub Total

F 4

Telangana Female Sub Total

M 5

Telangana Male Sub Total

9

Gujarat Sub Total

21 to 40

F 4

M 4

8

21 to 40 Sub Total

41 to 60

F 3

M 2

5

41 to 60 Sub Total

Above 60

F 1

M 3

4

Above 60 Sub Total

F 8

Female Sub Total

M 9

Male Sub Total

17

Grand Total

Displaying salary range wise number of employees:

SQL> Select Case

```
    when sal between 700 and 1000 then '700-1000'  
    when sal between 1001 and 2000 then '1001-2000'  
    when sal between 2001 and 3000 then '2001-3000'  
    when sal between 3001 and 4000 then '3001-4000'  
    when sal between 4001 and 5000 then '4001-5000'  
    Else 'Above 5000'  
End "Salary Range", Count(*)  
From Emp  
Group By Case  
when sal between 700 and 1000 then '700-1000'  
when sal between 1001 and 2000 then '1001-2000'  
when sal between 2001 and 3000 then '2001-3000'  
when sal between 3001 and 4000 then '3001-4000'  
when sal between 4001 and 5000 then '4001-5000'  
Else 'Above 5000'  
End;
```

Output:

Salary Ran	COUNT(*)
1001-2000	6
2001-3000	5
700-1000	2
4001-5000	1

Examples on Joins

Example-1:

Create the tables with following structure:

DEPT

Deptno	Dname	Loc

EMP

Empno	Ename	Job	Sal	MGR	Hiredate	Comm	Deptno

SALGRADE

Grade	Losal	Hisal

Creating Emp, Dept and Salgrade Tables:

Emp, Dept and Salgrade table creation Queries and insertion Queries are available in the “Google Drive” [emp_dept_salgrade.txt file]

Dept Table Records:

SQL> select * from dept;

Output:

DEPTNO	DNAME	LOC
-----	-----	-----
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Salgrade Table records:

SQL> select * from salgrade;

Output:

GRADE	LOSAL	HISAL
-----	-----	-----
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

Insert one record in “Emp” Table:

```
SQL> Insert into emp(empno,ename) values(1001,'STEVE');
```

Output:

1 row created.

Display Employee details with department details like department name and department location [EQUI JOIN]:**ORACLE STYLE:**

```
SQL> Select e.ename, e.sal, d.dname, d.loc  
      from emp e, dept d  
     where e.deptno=d.deptno;
```

[OR]

ANSI STYLE:

```
SQL> Select e.ename, e.sal, d.dname, d.loc  
      from emp e Inner Join Dept d  
     On e.deptno=d.deptno;
```

Output:

ENAME	SAL	DNAME	LOC
SMITH	800	RESEARCH	DALLAS
ALLEN	1600	SALES	CHICAGO
.			
.			

Display Employee Salary grades [NON-EQUI JOIN]:**ORACLE STYLE:**

```
SQL> Select e.ename, e.sal, s.grade  
      from emp e, salgrade s  
     where e.sal between s.losal and s.hisal;
```

[OR]

ANSI STYLE:

```
SQL> Select e.ename, e.sal, s.grade  
      from emp e JOIN salgrade s  
      On e.sal between s.losal and s.hisal;
```

Output:

ENAME	SAL	GRADE
SMITH	800	1
JAMES	950	1
ADAMS	1100	1
WARD	1250	2
MARTIN	1250	2
MILLER	1300	2
TURNER	1500	3

Display the dept name in which 'BLAKE' is working [EQUI JOIN]:

ORACLE STYLE:

```
SQL> Select e.ename, d.dname  
      from emp e, dept d  
      where e.deptno=d.deptno and e.ename='BLAKE';
```

[OR]

ANSI STYLE:

```
SQL> Select e.ename, d.dname  
      from emp e INNER JOIN dept d  
      On e.deptno=d.deptno  
      where e.ename='BLAKE';
```

Output:

ENAME	DNAME
BLAKE	SALES

Display the dept names in which ‘BLAKE’ is not working [NON-EQUI JOIN]:

ORACLE STYLE:

```
SQL> Select e.ename, d.dname  
      from emp e, dept d  
     where e.deptno!=d.deptno and e.ename='BLAKE';
```

[OR]

ANSI STYLE:

```
SQL> Select e.ename, d.dname  
      from emp e JOIN dept d  
     On e.deptno!=d.deptno  
    where e.ename='BLAKE';
```

Output:

ENAME	DNAME
-----	-----
BLAKE	ACCOUNTING
BLAKE	RESEARCH
BLAKE	OPERATIONS

Display the emp records who are working in NEW YORK [EQUI-JOIN]:

ORACLE STYLE:

```
SQL> Select e.ename, e.sal, d.dname, d.loc  
      from emp e, dept d  
     where e.deptno=d.deptno and d.loc='NEW YORK';
```

ANSI STYLE:

```
SQL> Select e.ename, e.sal, d.dname, d.loc  
      from emp e INNER JOIN dept d  
     On e.deptno=d.deptno where d.loc='NEW YORK';
```

Output:

ENAME	SAL	DNAME	LOC
-----	-----	-----	-----
CLARK	2450	ACCOUNTING	NEW YORK
KING	5000	ACCOUNTING	NEW YORK
MILLER	1300	ACCOUNTING	NEW YORK

Display all the employees who are assigned to department and not assigned to department [Left Outer Join]:

ORACLE STYLE:

```
SQL> Select e.ename, d.dname
      from emp e, dept d
      where e.deptno = d.deptno(+);
```

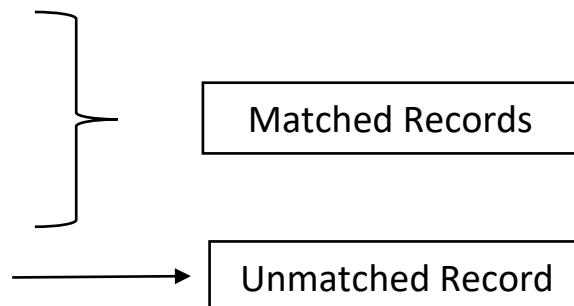
[OR]

ANSI STYLE:

```
SQL> Select e.ename, d.dname
      from emp e LEFT OUTER JOIN dept d
      On e.deptno=d.deptno;
```

Output:

ENAME	DNAME
CLARK	ACCOUNTING
KING	ACCOUNTING
MILLER	ACCOUNTING
SMITH	RESEARCH
JONES	RESEARCH
STEVE	



Display the employees with departments and display the departments which are having no employees [RIGHT OUTER JOIN]:

ORACLE STYLE:

```
SQL> Select e.ename, d.dname
      from emp e, dept d
      where e.deptno(+) = d.deptno;
```

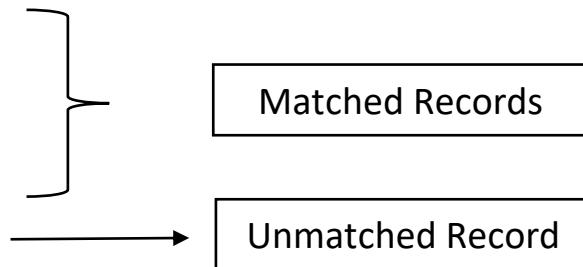
[OR]

ANSI STYLE:

```
SQL> Select e.ename, d.dname
      from emp e RIGHT OUTER JOIN dept d
      On e.deptno=d.deptno;
```

Output:

ENAME	DNAME
SMITH	RESEARCH
ALLEN	SALES
WARD	SALES
JONES	RESEARCH
	OPERATIONS



Display employee details with department details. Display the employees who are not assigned to any department and departments which are not having employees [FULL OUTER JOIN]:

ORACLE STYLE:

```
SQL> Select e.ename, d.dname
      from emp e, dept d
      where e.deptno=d.deptno(+)
      UNION
      Select e.ename, d.dname
      from emp e, dept d
      where e.deptno(+) = d.deptno;
```

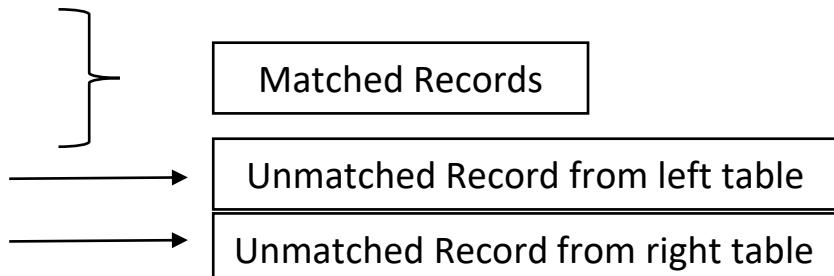
[OR]

ANSI STYLE:

```
SQL> Select e.ename, d.dname
      from emp e FULL OUTER JOIN dept d
      On e.deptno=d.deptno;
```

Output:

ENAME	DNAME
ADAMS	RESEARCH
ALLEN	SALES
SMITH	RESEARCH
STEVE	OPERATIONS



Display the employees who are not assigned to any department [Left Outer Join and Condition]:

ORACLE STYLE:

```
SQL> Select e.ename, d.dname
      from emp e, dept d
      where e.deptno=d.deptno(+) and d.dname is Null;
```

[OR]

```
SQL> Select e.ename, d.dname
      from emp e LEFT OUTER JOIN dept d
      On e.deptno=d.deptno
      Where d.dname is Null;
```

Output:

ENAME	DNAME
STEVE	Unmatched Record from left table

Display the departments which are not having employees [Right Outer Join and Condition]:

ORACLE STYLE:

```
SQL> Select e.ename, d.dname
      from emp e, dept d
      where e.deptno(+) = d.deptno and e.ename is Null;
```

[OR]

ANSI STYLE:

```
SQL> Select e.ename, d.dname
      from emp e RIGHT OUTER JOIN dept d
      On e.deptno=d.deptno
      where e.ename is Null;
```

Output:

ENAME	DNAME
OPERATIONS	Unmatched Record from right table

Display the employees who are not assigned to any department and the departments which are not having employees [Left Outer and Condition UNION Right Outer and Condition]:

ORACLE STYLE:

```
SQL> Select e.ename, d.dname
      from emp e, dept d
      where e.deptno=d.deptno(+) and d.dname is Null
      UNION
      Select e.ename, d.dname
      from emp e, dept d
      where e.deptno(+) = d.deptno and e.ename is Null;
```

[OR]

ANSI STYLE:

```
SQL> Select e.ename, d.dname
      from emp e FULL OUTER JOIN dept d
      On e.deptno=d.deptno
      where d.dname is null or e.ename is null;
```

Output:

ENAME	DNAME	
STEVE		Unmatched Record from left table
	OPERATIONS	Unmatched Record from right table

Example-2:

COURSE

Cid	CName

STUDENT

Sid	Sname	Cid

Course Table Records:

```
SQL> Select * from Course;
```

Output:

CID	CNAME
10	JAVA
20	PYTHON
30	ORACLE
40	HTML

10	JAVA
20	PYTHON
30	ORACLE
40	HTML

Student Table Records:

```
SQL> Select * from Student;
```

SID	SNAME	CID
-----	-------	-----

1005	Vijay	
1001	Ravi	20
1002	Srinu	30
1003	Sai	20
1003	Arun	30
1004	Sravan	10

Display the student records with course details [EQUI JOIN]:**ORACLE STYLE:**

```
SQL> Select s.sname, c cname  
      from student s, course c  
     where s.cid=c.cid;
```

[OR]

ANSI STYLE:

```
SQL> Select s.sname, c cname  
      from student s INNER JOIN course c  
        On s.cid=c.cid;
```

Output:

SNAME	CNAME
Ravi	PYTHON
Srinu	ORACLE
Sai	PYTHON
Arun	ORACLE
Sravan	JAVA

Display all student details including the students registered but not joined in the course [LEFT OUTER JOIN]:

ORACLE STYLE:

```
SQL> Select s.sname, c cname  
      from student s, course c  
     where s.cid=c.cid(+);
```

[OR]

ANSI STYLE:

```
SQL> Select s.sname, c cname  
      from student s LEFT JOIN course c  
     On s.cid=c.cid;
```

Output:

SNAME	CNAME
Sravan	JAVA
Ravi	PYTHON
Sai	PYTHON
Srinu	ORACLE
Arun	ORACLE
Vijay	

Display all student details with course details. Also Display the courses which are not having students [RIGHT OUTER JOIN]:

ORACLE STYLE:

```
SQL> Select s.sname, c cname  
      from student s, course c  
     where s.cid(+) = c.cid;
```

[OR]

ANSI STYLE:

```
SQL> Select s.sname, c cname  
      from student s RIGHT JOIN course c  
     On s.cid=c.cid;
```

Output:

SNAME	CNAME
Ravi	PYTHON
Srinu	ORACLE
Sai	PYTHON
Arun	ORACLE
Sravan	JAVA
	HTML

Display the students who are registered but not joined in the course [Left Outer and Condition]:

ORACLE STYLE:

```
SQL> Select s.sname, c cname  
      from student s, course c  
     where s.cid=c.cid(+) and c cname is Null;
```

[OR]

ANSI STYLE:

```
SQL> Select s.sname, c cname  
      from student s LEFT JOIN course c  
     On s.cid=c.cid  
    where c cname is null;
```

Output:

SNAME CNAME

Vijay

Display the courses which are not having students [Right Outer and Condition]:

ORACLE STYLE:

```
SQL> Select s.sname, c cname  
      from student s, course c  
     where s.cid(+) = c.cid and s.sname is Null;
```

[OR]

ANSI STYLE:

```
SQL> Select s.sname, c cname  
      from student s RIGHT JOIN course c  
        On s.cid=c.cid  
     where s.sname is null;
```

Output:

SNAME CNAME

HTML

Display all students details with course details and display the students who are registered but not joined in any course. Also Display the courses which are not having students [FULL OUTER JOIN]:

ORACLE STYLE:

```
SQL> Select s.sname, c cname  
      from student s, course c  
     where s.cid=c.cid(+)  
      UNION  
      Select s.sname, c cname  
      from student s, course c  
     where s.cid(+) = c.cid;
```

[OR]

ANSI STYLE:

```
SQL> Select s.sname, c cname  
      from student s FULL JOIN course c  
      On s.cid=c.cid;
```

Output:

SNAME	CNAME
Arun	ORACLE
Ravi	PYTHON
Sai	PYTHON
Sravan	JAVA
Srinu	ORACLE
Vijay	HTML

Display the students who are registered but not joined in any course and display the courses which are not having students [Left Outer and Condition Union Right Outer and Condition]:

ORACLE STYLE:

```
SQL> Select s.sname, c cname  
      from student s, course c  
      where s.cid=c.cid(+) and c cname is null  
      UNION  
      Select s.sname, c cname  
      from student s, course c  
      where s.cid(+) = c.cid and s.sname is null;
```

[OR]

ANSI STYLE:

```
SQL> Select s.sname, c cname  
      from student s FULL JOIN course c  
      On s.cid=c.cid  
      Where c cname is null or s.sname is null;
```

Output:

SNAME	CNAME
-------	-------

Vijay	HTML
-------	------

Example-3 [Equi Join on more than two tables]:

Create tables with following structures:

Dept1

Deptno	Dname	LocID
--------	-------	-------

Emp1

Empno	Ename	Job	Sal	Deptno
-------	-------	-----	-----	--------

Location

LocID	Lname	CountryID
-------	-------	-----------

Country

CountryID	CName
-----------	-------

Dept1 Table records:

SQL> select * from dept1;

Output:

DEPTNO	DNAME	LOCID
10	ACCOUNTS	101
20	SALES	102
30	RESEARCH	103
40	OPERATIONS	104

Emp1 Table records:

SQL> select * from emp1;

Output:

EMPNO	ENAME	JOB	SAL	DEPTNO
2001	ravi	clerk	5000	10
2002	kiran	clerk	4000	20
2003	sravan	manager	8000	20
2004	sai	manager	10000	10

Location Table Records:

```
SQL> select * from Location;
```

Output:

LID	LNAME	CID
101	CHICAGO	1002
102	DELHI	1001
103	MUMBAI	1001
104	NEWYORK	1002

Country Table Records:

```
SQL> select * from Country;
```

Output:

CID	CNAME
1001	INDIA
1002	AMERICA
1003	JAPAN
1004	RUSSIA

Display ename, dname, lname, cname:**ORACLE STYLE:**

```
SQL> Select e.ename, d.dname, l.lname, c.cname
   from emp1 e,dept1 d,location l,country c
  where e.deptno=d.deptno and d.locid=l.lid and l.cid=c.cid;
```

[OR]

ANSI STYLE:

```
SQL> Select e.ename, d.dname, l.lname, c.cname
   from emp1 e INNER JOIN dept1 d
  On e.deptno=d.deptno
 INNER JOIN Location l
  On d.locid=l.lid
 INNER JOIN Country c
  On l.cid=c.cid;
```

Output:

ENAME	DNAME	LNAME	CNAME
kiran	SALES	DELHI	INDIA
sravan	SALES	DELHI	INDIA
ravi	ACCOUNTS	CHICAGO	AMERICA
sai	ACCOUNTS	CHICAGO	AMERICA

Sub Queries

- Writing query in another query is called “Sub Query” or “Nested Query”.
- Inside query is called “Inner Query” or “Child Query” or “Sub Query”.
- Outside query is called “Outer Query” or “Parent Query” or “Main Query”.
- Sub query is used when condition is based on unknown value.
- Normally, inner query gets executed first. Then outer query gets executed.
- The result of inner query is input for the outer query.
- Inner query must be written in parenthesis [(.....)].
- Inner query must be select statement only. We cannot write other queries like Insert, Update or Delete as inner query.
- Outer query can be Select, Insert, Update or Delete Query.
- Sub queries can be nested more than one level and can be nested up to 255 levels.

Types of Sub Queries:

Sub queries can be categorized into following types:

- Single Row Sub Query
- Multi Row Sub Query
- Co-related Sub Query
- Inline View
- Scalar Sub Query

Single Row Sub Query:

An inner query which returns one value is called “Single Row Sub Query”.

Syntax:

```
Select <column_list>
From <Table_Name>
Where <column_name> <operator> (<Select statement>);
```

Example queries on Single Row Sub Queries:

- Display the employee records whose salary is greater than 'BLAKE':**

```
SQL> Select ename, sal from emp  
      where sal > (Select sal from emp where ename='BLAKE');
```

Output:

ENAME	SAL
JONES	2975
SCOTT	3000
KING	5000
FORD	3000

- Display the employee records who are senior to 'KING':**

```
SQL> Select ename, hiredate from emp  
      where hiredate < (Select hiredate from emp where  
ename='KING');
```

Output:

ENAME	HIREDATE
SMITH	17-DEC-80
ALLEN	20-FEB-81
.	

- Display the employee's name who has maximum experience:**

```
SQL> Select ename, hiredate from emp  
      where hiredate = (Select min(hiredate) from emp);
```

Output:

ENAME	HIREDATE
SMITH	17-DEC-80

4. Display the employee's name whose salary is maximum:

```
SQL> select ename,sal from emp  
      where sal = (Select max(sal) from emp);
```

Output:

ENAME	SAL
KING	5000

5. Display the second maximum salary:

```
SQL> Select max(Sal) from emp  
      where sal < (Select max(Sal) from emp);
```

Output:

MAX(SAL)
3000

6. Display the employee's name who salary is second maximum:

```
SQL> Select ename from emp  
      where sal = (Select max(Sal) from emp  
      where sal < (Select max(Sal) from emp));
```

Output:

ENAME
SCOTT
FORD

7. Delete an employee record who has maximum experience:

```
SQL> Delete from emp  
      where hiredate = (Select min(hiredate) from emp);
```

Output:

1 row deleted.

8. Update the salary of an employee whose empno is 7499 as Deptno 30's Maximum salary:

```
SQL> Update emp set sal =  
      (Select max(sal) from emp  
       where deptno=30) where empno=7499;
```

Output:

1 row updated.

9. Swapping salary of two employees whose employee numbers are 7369 and 7499:

```
SQL> Update emp set sal =  
      CASE empno  
        when 7369 then (Select sal from emp where empno=7499)  
        when 7499 then (Select sal from emp where empno=7369)  
      End  
      where empno in(7499,7369);
```

Output:

2 rows updated.

10. Display the department number which is spending maximum amount on employees:

```
SQL> Select deptno from emp  
      group by deptno  
      having sum(Sal) = (Select max(sum(sal)) from emp  
      group by deptno);
```

Output:

DEPTNO

20

11. Display the deptno which is having maximum number of employees:

```
SQL> Select deptno from emp  
      group by deptno  
      having count(*) = (Select max(count(*)) from emp  
      group by deptno);
```

Output:

```
DEPTNO
```

```
-----  
30
```

Multi Row Sub Query:

- An inner query which returns more than one value is called “Multi Row Sub Query”.
- We cannot use = operator for Multi row sub queries. Because = operator can check comparison with one value only. But Multi row sub query returns more than one value.
- We can use In, Not In, Any, All operators in Multi Row Sub Queries.

Examples on Multi Row Sub Query:

1. Display the employee records whose Job is equals to SMITH and BLAKE:

```
SQL> Select ename, job from emp  
      where job in (Select job from emp  
      where ename in('SMITH','BLAKE'));
```

Output:

```
ENAME   JOB
```

```
-----  
SMITH   CLERK  
ADAMS   CLERK  
JAMES   CLERK
```

MILLER	CLERK
JONES	MANAGER
BLAKE	MANAGER
CLARK	MANAGER

2. Display the employee names who are earning maximum salary in their department:

```
SQL> Select ename,sal, deptno from emp  
      where(deptno,sal) in(Select deptno,max(Sal) from emp  
                           group by deptno);
```

Output:

ENAME	SAL	DEPTNO
BLAKE	2850	30
KING	5000	10
SCOTT	3000	20
FORD	3000	20

3. Display the employee names who are senior in each department:

```
SQL> Select ename, hiredate  
      from emp  
      where (deptno,hiredate)  
            in(Select deptno,min(hiredate)  
                from emp group by deptno);
```

Output:

ENAME	HIREDATE
ALLEN	20-FEB-81
CLARK	09-JUN-81
SMITH	17-DEC-80

4. Display the employee records whose salary is equals to SCOTT and WARD salaries:

SQL> Select ename, sal from emp where sal in(Select sal from emp where ename in('SCOTT','WARD'));

Output:

ENAME	SAL
WARD	1250
MARTIN	1250
SCOTT	3000
FORD	3000

5. Display the employee records whose salary is greater than BLAKE and WARD salaries:

SQL> Select ename,sal from emp
where sal > All(Select sal from emp
where ename in('BLAKE','WARD'));

Output:

ENAME	SAL
JONES	2975
SCOTT	3000
FORD	3000
KING	5000

6. Display the employee records whose salary is greater than BLAKE or WARD salaries:

SQL> Select ename,sal from emp
where sal > Any(Select sal from emp
where ename in('BLAKE','WARD'));

Output:

ENAME	SAL
KING	5000
FORD	3000
SCOTT	3000
JONES	2975
BLAKE	2850
CLARK	2450
SMITH	1600
TURNER	1500
MILLER	1300

7. Display the employee records who are working in SALEs and RESEARCH Department:

```
SQL> Select Ename, Deptno from emp  
      where deptno in (Select deptno from dept where dname  
      in('SALES','RESEARCH'));
```

[OR]

```
SQL> Select e.ename,e.deptno  
      from emp e INNER JOIN dept d  
      On e.deptno=d.deptno  
      where d.dname in('SALES','RESEARCH');
```

Output:

ENAME	DEPTNO
SMITH	20
JONES	20
ALLEN	30
WARD	30
BLAKE	30
..	
..	

Correlated Sub Query:

- If inner query gets a value from outer query, then it is called “Correlated Sub Query”.
- In Correlated Sub Query, Outer query gets executed first. Then inner query gets executed.
- In this, inner query gets executed multiple times and it depends on number of rows returned by outer query.
- Use correlated sub query when inner query must get value from outer query & it has to be executed for multiple times.

Execution Process of correlated sub query:

- Outer query gets executed first. Returns a row.
- Passes a value to inner query.
- Inner query gets executed.
- Returns a value to outer query.
- Outer query condition gets executed.
- Above steps are performed repeatedly.

Example queries on Correlated Sub Queries:

1. Display the employee records who are getting salary greater than department's average salary:

```
SQL> Select ename,sal,deptno from emp e  
      where sal > (Select avg(Sal) from emp where  
      deptno=e.deptno);
```

Output:

ENAME	SAL	DEPTNO
ALLEN	1600	30
JONES	2975	20
BLAKE	2850	30
SCOTT	3000	20
KING	5000	10
FORD	3000	20

2. Display the employee records who are earning maximum salary in their department:

```
SQL> Select ename,sal,deptno from emp e  
      where sal = (Select max(Sal) from emp where  
      deptno=e.deptno);
```

Output:

ENAME	SAL	DEPTNO
BLAKE	2850	30
SCOTT	3000	20
KING	5000	10
FORD	3000	20

3. Display Top Three Salaries:

```
SQL> Select distinct sal from emp e  
      where 3 > (Select count(distinct sal) from emp where  
      sal>e.sal) Order By sal Desc;
```

Output:

SAL

5000
3000
2975

4. Display third maximum salary:

```
SQL> Select sal from emp e  
      where 2 = (Select count(distinct sal) from emp where  
      sal>e.sal);
```

Output:

SAL

2975

5. Display fourth maximum salary:

```
SQL> Select sal from emp e  
      where 3 = (Select count(distinct sal) from emp where  
                  sal>e.sal);
```

Output:

SAL

2850

6. Display n-th maximum salary:

```
SQL> Select sal from emp e  
      where &n-1 = (Select count(distinct sal) from emp where  
                  sal>e.sal);
```

Enter value for n: 1 -- highest salary

Output:

SAL

5000

SQL> /

Enter value for n: 2 --second maximum salary

Output:

SAL

3000

3000

SQL> /

Enter value for n: 3 --third aximum salary

Output:

SAL

2975

```
SQL> /
Enter value for n: 4          --fourth maximum salary
Output:
SAL
-----
2850
```

7. Display the department names which are having employees:

```
SQL> Select dname from dept d
      where exists(Select * from emp where deptno=d.deptno);
```

Output:

```
DNAME
-----
RESEARCH
SALES
ACCOUNTING
```

8. Display the department names which are not having employees:

```
SQL> Select dname from dept d
      where not exists(Select * from emp where deptno=d.deptno);
```

Output:

```
DNAME
-----
OPERATIONS
```

9. Display the employee records who are having subordinates:

```
SQL> Select empno,ename,job from emp e
      where exists(Select * from emp where mgr=e.empno);
```

Output:

EMPNO	ENAME	JOB
7902	FORD	ANALYST
7698	BLAKE	MANAGER
7839	KING	PRESIDENT
7566	JONES	MANAGER
7788	SCOTT	ANALYST
7782	CLARK	MANAGER

10. Display the employee records who are having subordinates:

SQL> Select empno,ename,job from emp e
where not exists(Select * from emp where mgr=e.empno);

Output:

EMPNO	ENAME	JOB
7876	ADAMS	CLERK
7521	WARD	SALESMAN
7499	ALLEN	SALESMAN
7900	JAMES	CLERK
7369	SMITH	CLERK
7934	MILLER	CLERK
7654	MARTIN	SALESMAN
7844	TURNER	SALESMAN

Inline View:

- A sub query written in “FROM” clause is called “Inline view” subquery.
- This sub query acts like a table.

Examples on Inline View:**1. Display the emp records whose annual salary is greater than 30000:**

```
SQL> Select * from (Select ename, sal, sal*12 an_Sal from emp) e  
      where an_Sal>30000;
```

Output:

ENAME	SAL	AN_SAL
JONES	2975	35700
BLAKE	2850	34200
SCOTT	3000	36000
KING	5000	60000
FORD	3000	36000

2. Display Top three salaries:

```
SQL> Select * from (Select ename,sal,dense_rank() over(order by  
      sal desc) rnk from emp) e where rnk<=3;
```

Output:

ENAME	SAL	RNK
KING	5000	1
SCOTT	3000	2
FORD	3000	2
JONES	2975	3

3. Display third record in emp table:

```
SQL> Select * from (Select rownum rn, ename, sal from emp) e  
      where rn=3;
```

Output:

RN	ENAME	SAL
3	WARD	1250

4. Display 3rd,5th and 11th records:

SQL> Select * from (Select rownum rn, ename, sal from emp) e
where rn in(3,5,11);

Output:

RN	ENAME	SAL
3	WARD	1250
5	MARTIN	1250
11	ADAMS	1100

5. Display record numbers from 6 to 10:

SQL> Select * from (Select rownum rn, ename, sal from emp) e
where rn between 6 and 10;

Output:

RN	ENAME	SAL
6	BLAKE	2850
7	CLARK	2450
8	SCOTT	3000
9	KING	5000
10	TURNER	1500

6. Display even row number records:

SQL> Select * from (Select rownum rn, ename, sal from emp) e
where mod(rn,2)=0;

Output:

RN	ENAME	SAL
2	ALLEN	1600

4	JONES	2975
6	BLAKE	2850
8	SCOTT	3000
10	TURNER	1500
12	JAMES	950
14	MILLER	1300

7. Display odd row number records:

```
SQL> Select * from (Select rownum rn, ename, sal from emp) e  
      where mod(rn,2)=1;
```

Output:

RN	ENAME	SAL
1	SMITH	800
3	WARD	1250
5	MARTIN	1250
7	CLARK	2450
9	KING	5000
11	ADAMS	1100
13	FORD	3000

Scalar Sub query:

- A sub query written in “SELECT” clause is called “Scalar Sub Query”.
- This sub query acts like a column.

Examples on Scalar Sub Query:

1. Display number of rows in emp and dept tables:

```
SQL> Select (Select count(*) from emp) emp,  
      (Select count(*) from dept) dept  
      from dual;
```

Output:

EMP	DEPT
-----	-----
14	4

- 2. Display the share of each department how much percentage spending on total salaries:**

```
SQL> Select deptno, sum(Sal) dept_tot_sal,
      (Select sum(Sal) from emp) tot_sal,
      Round((sum(Sal)/(Select sum(Sal) from emp))*100,2) percentage
      from emp
      group by deptno;
```

Output:

DEPTNO	DEPT_TOT_SAL	TOT_SAL	PERCENTAGE
-----	-----	-----	-----
30	9400	29025	32.39
10	8750	29025	30.15
20	10875	29025	37.47

Views

View:

- “View” is a Database Object like Table.
- It is a virtual table. It means, it does not contain physical data. It does not occupy the memory.
- It is created on Tables.
- The Table on which view is created is called “Base Table”.
- It holds Select Query. It does not hold query result.
- When we retrieve the data through view, Oracle runs select query which is in the view. So, it retrieves the data from the base table.
- It gives most recent data [Committed Data]. Because, it retrieves data from base table.

Advantages:

- View provides security for the data.
- It reduces complexity & simplifies the usage of query.

Syntax to create a View:

```
Create View <View_Name>
As
<Select Query> ;
```

Note:

Database security can be implemented in three levels.

- To implement Database Level Security, use Schemas [Creating Users]
- To implement Table Level Security, use privileges [Granting permissions]
- To implement data level [row level & column level] security, use views.

Types of Views:

Views can be divided into two types according to the number of Base Tables. They are:

- Simple View
- Complex View

Simple View:

- A view which is created based on one table is called “Simple View”.
- We can perform DML operations like Insert, Delete and Update on this view.
- It can be updated. So, it can be also called as “Updatable View”.

Examples on Simple View:**Example-1 [Implementing Column Level Security]:**

Create a user with the name c##ravi. Create “emp” table.

Create a view on “Emp” Table for the columns Empno, Ename, Job and Deptno

Grant permission on view to the user “c##oracle7am”

Connect as “c##oracle7am” user & perform DML Operations:

Creating a user:

Log in as DBA

User Name: system

Password: Type password Given at the time of Installation

Type following query.

```
SQL> Create User c##Ravi identified by nareshit  
      default tablespace users  
      quota unlimited on users;
```

Grant Permission to connect and create the table:

```
SQL> Grant connect, resource to c##ravi;
```

Output:

Grant succeeded.

Connect as C##ravi:

```
SQL> conn c##ravi/nareshit
```

Output:

Connected.

Create a Table “Emp” and Insert the records.

These commands are available in “emp_dept_salgrade.txt” file which is available in Google Drive.

Connect as DBA to give permission to c##ravi for creating the view:

```
SQL> conn system/nareshit
```

Output:

Connected.

```
SQL> Grant create view to c##ravi;
```

Output:

Grant succeeded.

Connect as “Ravi”, create a view and Grant permission on view to “oracle7AM”:

```
SQL> conn c##ravi/nareshit
```

Output:

Connected.

```
SQL> Create view v1
```

As

```
    Select empno, ename, job, deptno from emp;
```

Output:

View created.

Retrieving data through view:

```
SQL> Select * from v1;
```

Output:

EMPNO	ENAME	JOB	DEPTNO
7369	SMITH	CLERK	20
7499	ALLEN	SALESMAN	30
....			

Granting permission on view “V1” to the user “C##Oracle7AM”:

```
SQL> Grant Select, Insert, Update, Delete on V1 to c##oracle7am;
```

Output:

Grant succeeded.

Connect as “c##oracle7am” and Perform DML Operations:

```
SQL> conn c##oracle7am/nareshit
```

Output:

Connected.

Retrieving data through view:

```
SQL> Select * from c##ravi.v1;
```

Output:

EMPNO	ENAME	JOB	DEPTNO
7369	SMITH	CLERK	20
7499	ALLEN	SALESMAN	30
....			

Inserting record through view:

```
SQL> Insert into c##ravi.v1 values(1001,'STEVE','CLERK',20);
```

Output:

1 row created.

Updating record through view:

```
SQL> Update c##ravi.v1 set Job='MANAGER' where empno=1001;
```

Output:

1 row updated.

Deleting record through view:

```
SQL> Delete from c##ravi.v1 where empno=1001;
```

Output:

1 row deleted.

Note:

In the above example, “c##ravi” user is owner of emp table. He created a view “v1” & given permission to “c##oracle7am” user.

“c##oracle7am” user can access 4 columns of emp table. He cannot access remaining columns like sal, comm ...etc. Hence Column Level security implemented.

“c##oracle7am” user can select, insert, update and delete the records through view.

Example-2 [Implementing Row Level Security]:

Log in as “c##ravi” user.

Create a view on deptno 20 employees of “emp” table for all columns.

Grant permission on this view to the user “c##oracle7am”.

Logging as c##ravi and creating view:

```
SQL> conn c##ravi/nareshit
```

Output:

Connected.

```
SQL> Create view v2  
As  
Select * from emp where deptno=20;
```

Output:

View created.

Retrieving data through view:

```
SQL> Select * from v2;
```

Output:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20
7902	FORD	ANALYST	7566	03-DEC-81	3000		20

Granting permission on view “V2” to the user “c##oracle7am”:

```
SQL> Grant Select, Insert, Delete, Update on V2 to c##oracle7am;
```

Output:

Grant succeeded.

Log in as “c##oracle7am” and perform DML operations on View “V2”:

```
SQL> conn c##oracle7am/nareshit
```

Output:

Connected.

Retrieving data through view:

```
SQL> Select * from c##ravi.v2;
```

Output:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20
7902	FORD	ANALYST	7566	03-DEC-81	3000		20

Inserting record through view:

```
SQL> Insert into c##ravi.v2(empno,ename,job,sal,deptno)
   2 values(1001,'STEVE','CLERK',4000,20);
```

Output:

1 row created.

Updating record through view:

```
SQL> Update c##ravi.v2 set job='MANAGER' where empno=1001;
```

Output:

1 row updated.

Deleting record through view:

```
SQL> Delete from c##ravi.v2 where empno=1001;
```

Output:

1 row deleted.

Inserting other department employee's record through view:

```
SQL> Insert into c##ravi.v2(empno,ename,job,sal,deptno)
   2 values(1002,'JOHN','CLERK',4000,10);
```

Output:

1 row created.

Note:

“c##oracle7am” user has no permission to view the deptno 10 records. Still he is able to insert deptno 10 emp record. To prevent this, recreate the view using “WITH CHECK OPTION” clause.

Even if he inserts the record, he cannot see that record through view.

To check it write following query:

```
SQL> Select * from c##ravi.v2;
```

Output:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20
7902	FORD	ANALYST	7566	03-DEC-81	3000		20

Recreating the view using “WITH CHECK OPTION” clause:

Connect as “c##ravi”:

```
SQL> conn c##ravi/nareshit
```

Output:

Connected.

Recreationg view “v2” using “with check option” clause:

```
SQL> Create or Replace view v2
```

As

```
  Select * from emp where deptno=20
```

```
  With Check Option;
```

Output:

View created.

Connect as “c##oracle7am” user & insert department number 10 employee’s record:

```
SQL> Insert into c##ravi.v2(empno,ename,job,sal,deptno)
   values(1003,'STEVE','CLERK',6000,10);
```

Output:

ERROR at line 1:

ORA-01402: view WITH CHECK OPTION where-clause violation

Note:

Above record is violating where condition of view “V2”. It cannot be accepted by view.

“WITH CHECK OPTION” clause:

If a view is created using “With Check Option” clause, this view cannot accept the records which cannot be seen by the user through this view. It means, the records which are violating “where” clause of view will not be accepted.

Complex View:

- A view which is created based on multiple tables is called “Complex View”.
- A view which is created using **joins or group by or having or aggregate functions or expressions or set operators or sub queries** can be called as “Complex View”.
- This view cannot be updated. So, it can be also called as “Read-Only View”.

Example on Complex View:

```
SQL> Create view v3
```

As

```
  Select e.ename,e.job,e.deptno,d.dname,d.loc
    from emp e INNER JOIN dept d
      On e.deptno=d.deptno;
```

Output:

View created.

Retrieving records through view:

```
SQL> Select * from v3;
```

ENAME	JOB	DEPTNO	DNAME	LOC
SMITH	CLERK	20	RESEARCH	DALLAS
ALLEN	SALESMAN	30	SALES	CHICAGO
..				
..				

Inserting record through view:

```
SQL> Insert into V3 values('STEVE','CLERK',20,'RESEARCH','DALLAS');
```

Output:

ERROR at line 1:

ORA-01779: cannot modify a column which maps to a non key-preserved table

Note:

With above query, we can understand that insertion operation cannot be performed on “Complex View”.

Deleting record through view:

```
SQL> Delete from v3 where ename='SMITH';
```

Output:

ERROR at line 1:

ORA-01752: cannot delete from view without exactly one key-preserved table

Note:

With above query, we can understand that deletion operation cannot be performed on “Complex View”.

Updating record through view:

```
SQL> Update v3 set job='MANAGER' where ename='SMITH';
```

Output:

ERROR at line 1:

ORA-01779: cannot modify a column which maps to a non key-preserved table

Note:

With above query, we can understand that updation operation cannot be performed on “Complex View”.

Differences between Simple View & Complex View:

SIMPLE VIEW	COMPLEX VIEW
A Simple View is created based on one table.	A Complex View is created based on more than one table.
We can perform DML operations like Insert, Delete and Update on this view.	We cannot perform DML operations like Insert, Delete and Update operations on this view.
It can be also called as “Updatable View”.	It can be also called as “Read-Only View”.
It performs simple operations	It performs complex operations like Joins, Group By, Having, Set Operators, Aggregate functions, Expressions and Sub Queries.

Can we create a view from another view?

Yes. We can.

Example:

SQL> Create view v4 as select * from v3;

Output:

View created.

SQL> Select * from v4;

Output:

ENAME	JOB	DEPTNO	DNAME	LOC
SMITH	CLERK	20	RESEARCH	DALLAS
ALLEN	SALESMAN	30	SALES	CHICAGO

After creating a view if base table data is modified does it reflect to view?

Yes. It will be reflected to view. Because when we retrieve data through view it runs select query. It gives recent data from the base table.

After creating view, if we add a column to the base table does it reflect to view?

No. It will not be reflected to view. If we want altered table through view, we must recreate the view using “Create or Replace View”.

Note:

If data of base table modified, we can see the changes through view.

If Structure of base table modified, we cannot see the changes through view.

Can we create a View without Table?

No. To create a View, Base Table is required. But there is a concept “FORCE VIEW” with which we can create a view without existing table.

FORCE VIEW:

A view which is created without existing table with some compilation errors is called “Force View”.

Even if we create the view, this view will not work until we create the base table.

Example:

SQL> Create Force View FV1

As

Select * from ABC;

Output:

Warning: View created with compilation errors.

SQL> Select * from FV1;

Output:

ERROR at line 1:

ORA-04063: view "C##RAVI.FV1" has errors

Note:

After creating ABC table above view will work.

Can we add a column to the view?

No. we cannot use “Alter” and “Truncate” commands on “View”.

If we delete Base Table, does Oracle deletes view?

No. If we delete base table view will not be deleted. But this view will not work till recreates the base table.

Retrieving list of Views created by the user:

“User_VIEWS” table stores the information about the views created by the user.

SQL> Select view_name from user_views;

Output:

VIEW_NAME

V1
V2
V3
V4
FV1

Dropping a View:

“Drop” command is used to drop the view.

Syntax:

Drop view <view_name>;

Example:

SQL> Drop view v1;

Output:

View dropped.

PL / SQL

- In PL/SQL, PL stands for “Procedural Language. SQL stands for “Structured Query Language”.
- It is a Procedural Language. We can define a set of statements to perform actions.

Features of PL/SQL:

Improves Performance:

In PL/SQL, SQL commands can be grouped into one block, and we can submit this block to Oracle. Oracle executes this block and gives response to user. PL/SQL reduces number of requests and response. So, performance will be improved.

Provides Control Structures:

It provides Control Structures which are used to control the flow of execution of statements. These are useful to execute the statements based on conditions. These are also useful to execute the statements repeatedly.

Provides Exception Handling:

The mechanism of handling run-time errors is called “Exception Handling”. When run-time error occurs, program will be terminated in the middle of execution. It will not execute remaining code. To continue the execution even if run-time error occurs, we must handle the exception.

Provides Modular Programming:

Modular programming is a programming style. In this, we divide large program into small parts. These small parts are called Procedures and Functions. It improves understandability. Better maintenance will be provided like easy to debug, test and edit ...etc.

Provides Reusability:

PL/SQL provides Functions, Procedures, Triggers and Packages. Because of these database objects we can reuse the code. We define code only once. But we can use that code for any number of times by calling.

Provides Security:

PL/SQL programs are stored in database. So, these are centralized. Only authorized users can execute these programs.

PL/SQL Blocks:

PL/SQL program contains blocks. These PL/SQL blocks are 2 Types. They are:

- Anonymous Blocks
- Named Blocks

Anonymous Block:

A PL/SQL block without any name is called “Anonymous Block”.

Named Block:

A PL/SQL block with name is called “Named Block”.

Ex: Procedure, Function, Package, Trigger

Syntax of PL/SQL Program:

```
DECLARE
    <declare the variables>      --DECLARATION PART
BEGIN
    <Statements>                --EXECUTION PART
END;
/
```

Dbms_output.put_line():

“Dbms_output” is a package. It provides put_line() procedure. Put_line() procedure is used to print the messages on output.

Example: dbms_output.put_line('hello'); --prints hello

Set ServerOutput On:

By default, messages are not sent to output screen. To send messages to output execute “SET SERVEROUTPUT ON” command.

Program to print ‘hello’:

```
Begin  
    Dbms_output.put_line('hello');
```

```
End;
```

```
/
```

Output:

```
Hello
```

Execution Process:

- Type above program in Notepad / Edit Plus or any other text editor.
- Save it in d: drive oracle7am folder with the name “hello.sql”.
- Open SQL Command Prompt and type as following:
SQL>@d:\oracle7am/hello.sql
--Above compiles & runs “hello.sql” program.

Data Types in PL/SQL:

Number(p)

Number(p,s)

Integer

Int

Pls_integer

Binary_integer

Binary_Float

Binary_Double

Boolean

Date

Timestamp

Char(n)

Varchar2(n)

CLOB

Nchar(n)

NVarchar2(n)

nCLOB

BFILE

BLOB

Declaring a Variable:

Syntax:

```
<variable> <data_type>;
```

Example:

```
x Number(4);  
d Date;  
a varchar2(20);
```

Assigning value to variable:

`:=` is the assignment operator provided by PL/SQL. It is used to assign the values.

Syntax:

```
<variable> := <value>;
```

Example:

```
x := 20;  
d := '12-AUG-2021';  
a := 'Raju';
```

Reading value at run-time:

Prefix variable name with `&` to read the input at run-time.

Example:

```
x := &x;
```

Program to add two numbers:

DECLARE

```
x number(4);  
y number(4);  
z number(4);
```

BEGIN

```
x := 20;  
y := 30;
```

```
z := x+y;

    dbms_output.put_line('sum=' || z);
    dbms_output.put_line('sum of ' || x || ' and ' || y || ' is ' || z);
END;
/
```

Output:

sum=50
sum of 20 and 30 is 50

Program to add two numbers by reading two numbers from keyboard:

```
DECLARE
    x number(4);
    y number(4);
    z number(4);
BEGIN
    x := &x;
    y := &y;

    z := x+y;

    dbms_output.put_line('quotient=' || z);
    dbms_output.put_line('sum of ' || x || ' and ' || y || ' is ' || z);
END;
/
```

Output:

Enter value for x: 20
old 6: x := &x;
new 6: x := 20;
Enter value for y: 30
old 7: y := &y;
new 7: y := 30;
quotient=50
sum of 20 and 30 is 50

To avoid old and new when we read data from output, execute the following command:

```
SQL> SET VERIFY OFF
```

Program to display weekday name of given date:

```
DECLARE
    x Date;
BEGIN
    x := '&x';
    dbms_output.put_line(to_char(x,'day'));-- d => weekdaynumber
=> 1   dy => sun  day => sunday
END;
/
```

Output:

Enter value for x: 15-aug-1947

Friday

Control Structures

- Control Structures are used to control the flow of execution of statements.
- To transfer the control to our desired location, we use control structures.

PL/SQL provides following control structures:

Conditional Control Structures	If ...Then If ...Then...Else If...Then..Elsif Nested If Case	Used to execute the statements based on condition.
Looping Control Structures	Simple Loop While Loop For Loop	Used to executes the statements repeatedly.
Jumping Control Structures	Goto Exit Exit When	Used to jump out of loop or to specified label.

Conditional Control Structures:

Conditional Control Structures are used to execute the statements based on condition.

PL/SQL provides following Conditional Control Structures:

- If ...Then
- If ...Then...Else
- If...Then..Elsif
- Nested If
- Case

IF...THEN:

The statement in 'IF' block get executed when the condition is TRUE. It skips the statements when the condition is FALSE.

Syntax:

```
If <condition> Then  
    //Statements  
End If;
```

Program to check whether the person is eligible to vote or not:

```
DECLARE  
    age integer;  
BEGIN  
    age := &age;  
  
    IF age>=18 THEN  
        dbms_output.put_line('Eligible to vote');  
    END IF;  
  
END;  
/
```

Output:

```
Enter value for age: 25  
Eligible to vote
```

If-Then-Else:

The statements in 'If' block get executed when the condition is TRUE. The statements in 'Else' block get executed when the condition is FALSE.

Syntax:

```
If <condition> Then  
    //Statements  
Else  
    //Statements  
End If;
```

Program to check whether the given number is even or odd:

```
DECLARE
    n int;
BEGIN
    n:=&n;

    IF mod(n,2)=0 THEN
        dbms_output.put_line('EVEN');
    ELSE
        dbms_output.put_line('ODD');
    END IF;

END;
```

Output:

Enter value for n: 7
ODD

If..Then..Elsif:

It is used to execute the statements based on any one of the multiple conditions. 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.

Syntax:

```
If <condition-1> Then
    //Statements
Elsif <condition-2> Then
    //Statements
.
.
Else
    //Statements
End If;
```

Program to check whether the given number is positive or negative or zero:

```
DECLARE
    n int;
BEGIN
    n:=&n;

    IF n>0 THEN
        dbms_output.put_line('POSITIVE');
    ELSIF n<0 THEN
        dbms_output.put_line('NEGATIVE');
    ELSE
        dbms_output.put_line('ZERO');
    END IF;

END;
```

/

Output:

Enter value for n: 8

POSITIVE

Neste If:

Writing ‘IF’ block in another ‘IF’ Block is called “Nested If”. First outer condition is tested. If outer condition is TRUE, inner condition will be tested. If inner condition is also TRUE, it executes the statements in Inner If. If Outer Condition is False, it comes out of Outer If.

Syntax:

```
If <condition-1> Then
    If <condition-2> Then
        //Statements
    End If;
End If;
```

Program to find biggest in three different numbers:

```
DECLARE
    x int;
    y int;
    z int;
BEGIN
    x:=&x;
    y:=&y;
    z:=&z;

    IF x>y THEN
        IF x>z THEN
            dbms_output.put_line('x is big');
        ELSE
            dbms_output.put_line('z is big');
        END If;
    ELSE
        IF y>z THEN
            dbms_output.put_line('y is big');
        ELSE
            dbms_output.put_line('z is big');
        END IF;
    END IF;

    END;
/

```

Output:

```
Enter value for x: 100
Enter value for y: 200
Enter value for z: 150
y is big
```

Case:

“Case” can be used in two ways. They are:

- Simple Case
- Searched Case

Simple Case:

“Simple Case” can check equality condition only.

Syntax:

```
Case <expression>
    When <value> Then
        <Statements>;
    When <value> Then
        <Statements>;
    When <value> Then
        <Statements>;
    .
    .
    Else
        <Statements>
End Case;
```

Program to check whether the given number is even or odd using SIMPLE 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;
/
```

Output:

Enter value for n: 8

EVEN

Searched Case:

It can check non-equality conditions also.

Syntax:

```
Case
    When <value> Then
        <Statements>;
    When <value> Then
        <Statements>;
    When <value> Then
        <Statements>;
    .
    .
    Else
        <Statements>
End Case;
```

Program to check whether the given number is positive or negative or zero using searched CASE:

```
DECLARE
    n integer;
BEGIN
    n := &n;
    CASE
        WHEN n>0 THEN
            dbms_output.put_line('POSITIVE');
        WHEN n<0 THEN
            dbms_output.put_line('NEGATIVE');
        ELSE
            dbms_output.put_line('ZERO');
    END CASE ;
END ;
/
```

Output:

Enter value for n: -4

NEGATIVE

Looping Control Structures:

Looping Control Structures are used to execute the statements repeatedly.
PL/SQL provides following Looping Control Structures:

- Simple Loop
- While Loop
- For Loop

Simple Loop:

Syntax:

```
Loop
    //Statements;
    Exit when <condition>;
End Loop;
```

Program to print numbers from 1 to 10 using Simple Loop:

DECLARE

i INT := 1;

BEGIN

LOOP

dbms_output.put_line(i);

i := i+1;

EXIT WHEN i>10;

END LOOP;

END ;

/

Output:

1
2
3
4
5
6
7
8
9
10

While Loop:**Syntax:**

```
While <Condition>
Loop
  //Statements;
End Loop;
```

Program to print first n numbers using while loop:

```
DECLARE
  i integer := 1;  --Initialization
  n integer;
BEGIN
  n := &n;
  WHILE i<=n
    Loop
      dbms_output.put_line(i); --1
      i := i+1;
    END LOOP;
END ;
/
```

Output:

Enter value for n: 5

1
2
3
4
5

For Loop:**Syntax:**

```
for <variable> in [reverse] <lower> .. <upper>
Loop
  //Statements
End Loop;
```

Program to print numbers from 1 to 10 using for loop:

```
BEGIN
    FOR i IN 1..10
    LOOP
        dbms_output.put_line(i);
    END LOOP;
END ;
/
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Program to print numbers from 5 to 1 using for loop:

```
BEGIN
    FOR i IN REVERSE 1..5
    LOOP
        dbms_output.put_line(i);
    END LOOP;
END ;
/
```

Output:

```
5
4
3
2
1
```

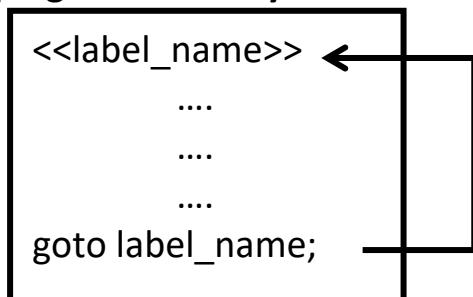
Jumping Control Structures:

Goto:

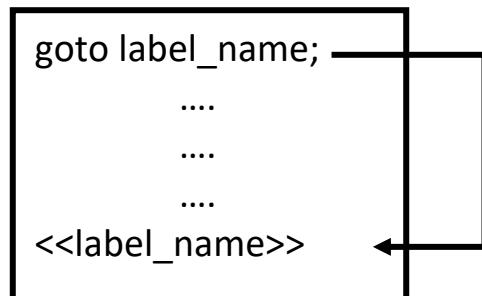
It is used to transfer the control to specified label. It can be used in two ways. They are:

- Jumping Backward
- Jumping Forward

Jumping Backward Syntax:



Jumping Forward Syntax:



Program to print numbers from 1 to 4 using goto:

```

DECLARE
    i INT := 1;
BEGIN
    <<nareshit>>
        dbms_output.put_line(i);
        i := i+1;
        IF i<=4 THEN
            GOTO nareshit;
        END IF;
END ;
/
  
```

Output:

1
2
3
4

Program to demonstrate jumping forward:

```
BEGIN
    dbms_output.put_line('hello');
    GOTO xyz;
        dbms_output.put_line('hi');
    <<xyz>>
        dbms_output.put_line('welcome');
```

END ;

/

Output:

hello
welcome

Exit and Exit When:

- These can be used in Loop only.
- These are used to terminate the loop in the middle of execution.

Program to demonstrate EXIT WHEN:

```
BEGIN
    FOR i IN 1..10
        Loop
            dbms_output.put_line(i);
            EXIT WHEN i=5;
        END loop;
END ;
```

/

Output:

1
2
3
4
5

Program to demonstrate EXIT:

```
BEGIN
    FOR i IN 1..10
    Loop
        dbms_output.put_line(i);
        If i=5 Then
            EXIT;
        End If;
    END loop;
END ;
/
```

Output:

```
1
2
3
4
5
```

Using SQL Statements in PL/SQL:

We can use DML, DQL and TCL commands only in PL/SQL Program. If we want to use DDL commands in PL/SQL program, use Dynamic SQL.

All DML and TCL Commands syntax is same as SQL. But DQL command i.e Select command syntax is different from SQL.

Syntax of Select Command in PL/SQL :

```
Select <column_list> into <variable_list>
from <table_name>
where <condition>;
```

Example:

```
Select ename,sal into x,y
from emp
where empno=7369;
```

x = SMITH, y=800

Program to display employee name and salary of given employee number:

```

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;
/

```

Output:

Enter value for empno: 7499
ALLEN 1600

Attribute Related Data Types:

There are two attribute related data types provided by PL/SQL. They are:

- %type
- %rowtype

%type:

- It is used to take the data type, field size of the table column.
- It avoids mismatch b/w data type & field size of variable & column

Advantage:

- We have no need to check data type and field size of a column every time when we declare.

Syntax:

<variable_name> <table_name>.<column_name>%type;

Example:

v_ename emp.ename%type;

In above example, v_ename is a variable. Its data type will be taken as “ename” column type of “emp” table.

%rowtype:

- It is used to hold entire row of a table.
- It can hold one record only at a time.

Advantages:

- It decreases number of variables
- It reduces complexity.

Example:

```
v_emprec emp%rowtype;
```

In the above example, v_emprec is variable of row type. It can hold record of emp table.

Write a program to calculate and display the experience of given employee number:

```
DECLARE
```

```
    v_empno emp.empno%type;
    v_hiredate emp.hiredate%type;
    v_exp integer;
```

```
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('emp exprience is:' || v_exp || ' years');
```

```
END ;
```

```
/
```

Output:

Enter value for empno: 7369

emp exprience is:41 years

Program to delete an employee record who has greater than 40 years experience:

```

DECLARE
    v_empno emp.empno%type;
    v_hiredate emp.hiredate%type;
    v_exp integer;
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('emp exprience is:' || v_exp || ' years');

    IF v_exp>40 THEN
        DELETE FROM emp WHERE empno=v_empno;
        COMMIT ;
        dbms_output.put_line(v_empno || ' employee record deleted');
    END IF ;

END ;
/

```

Output:

```

Enter value for empno: 7499
emp exprience is:41 years
7499 employee record deleted

```

Program to increase the salary of given empno. If salary exceeds 5000 then cancel the transaction:

```

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>5000 THEN
        Rollback;
        dbms_output.put_line('Rolled back successfully..');
    ELSE
        COMMIT ;
        dbms_output.put_line('committed successfully...');

    END IF ;
END ;
/

```

Output:

Enter value for empno: 7369
 Enter value for amount: 2000
 committed successfully...

Program to display employee record of given employee number:

```

-- Display ename, job, sal, hiredate, deptno of given empno
DECLARE
    v_empno emp.empno%type;
    v_emprec emp%rowtype;
BEGIN
    v_empno := &empno;
    SELECT * INTO v_emprec FROM emp WHERE empno=v_empno;
    dbms_output.put_line(v_emprec.ename || ' ' || v_emprec.job || '
' || v_emprec.sal || ' ' || v_emprec.hiredate || ' ' || v_emprec.deptno);
END ;
/

```

Output:

Enter value for empno: 7900
 JAMES CLERK 950 03-DEC-81 30

Program to increase the salary of an employee based on experience. If experience is greater than or equals to 40 then increment by 20%. Otherwise increment by 10%:

```
DECLARE
    v_empno emp.empno%type;
    v_hiredate emp.hiredate%type;
    v_exp integer;
    v_per float;
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('emp exprience is:' || v_exp || ' years');

    IF v_exp>=40 THEN
        v_per := 0.2;
    ELSE
        v_per := 0.1;
    END IF ;

    UPDATE EMP SET SAL = SAL + SAL*v_per WHERE empno=v_empno;
    COMMIT ;

    dbms_output.put_line(v_empno || 'employee record updated....');
    dbms_output.put_line(v_per || ' on sal increased');

END ;
/
```

Output:

```
Enter value for empno: 7900
emp exprience is:40 years
7900employee record updated....
.2 on sal increased
```

Program to increase salary of given employee number based on job as follows:

Manager => increment by 20%

Clerk => increment by 15%

Others => increment by 10%

DECLARE

 v_empno emp.empno%type;

 v_job emp.job%type;

 v_per float;

BEGIN

 v_empno := &empno;

 SELECT job INTO v_job FROM emp WHERE empno=v_empno;

 IF v_job='MANAGER' THEN

 v_per := 0.2;

 elsif v_job='CLERK' THEN

 v_per := 0.15;

 ELSE

 v_per := 0.1;

 END IF ;

 UPDATE emp SET sal=sal+sal*v_per WHERE empno=v_empno;

 COMMIT ;

 dbms_output.put_line('Salary Updated.....'|| v_per);

END ;

/

Output:

Enter value for empno: 7369

Salary Updated.....15

Create following tables and write a program to calculate total, average and result of given student id. Then Insert these values into RESULT table:
std

sid	sname	m1	m2	m3
1001	Ravi	45	70	50
1002	Kiran	60	80	30

Result

sid	tot	avrg	result

DECLARE

```
v_sid std.sid%type;
v_stdrec std%rowtype;
v_resrec result%rowtype;
```

BEGIN

```
v_sid := &sid;
```

```
SELECT * INTO v_stdrec FROM std WHERE sid=v_sid;
```

```
v_resrec.total := v_stdrec.m1 + v_stdrec.m2 + v_stdrec.m3;
v_resrec.avrg := v_resrec.total/3;
```

```
IF v_stdrec.m1>=40 AND v_stdrec.m2>=40 AND v_stdrec.m3>=40 THEN
```

```
    v_resrec.result := 'PASS';
```

```
ELSE
```

```
    v_resrec.result := 'FAIL';
```

```
END IF ;
```

```
INSERT INTO result VALUES(v_sid, v_resrec.total, v_resrec.avrg,
v_resrec.result);
```

```
dbms_output.put_line('Record inserted .....');
```

```
END ;
```

```
/
```

Output:

Enter value for sid: 1001

Record inserted

Cursors

- Cursor is used to hold multiple rows.
- It is a pointer [reference] to the memory location of oracle instance [RAM] where rows are stored.
- At a time, we can access one row only. To access multiple rows, use Loop.

Steps to use Cursor:

To use cursor, we follow four steps. They are:

- Declaring the Cursor
- Opening the Cursor
- Fetching the Records from Cursor
- Closing the Cursor

Declaring the Cursor:

Syntax:

```
Cursor <Cursor_Name> Is <Select Statement>;
```

Example:

```
Cursor c1 Is Select empno,ename,sal from emp;
```

A cursor variable will be created with above statement.

Opening the Cursor:

Syntax:

```
Open <Cursor_Name>;
```

Example:

```
Open c1;
```

- When cursor is opened, select query will be submitted to Oracle.
- Oracle executes query & result will be copied into oracle instance.
- Now this memory location address will be given to the cursor c1.

Fetching the records from cursor:

Syntax:

Fetch <cursor_name> into <variable_list>;

Example:

Fetch c1 into x,y,z;

- Above statement fetches records and copies into x,y and z variables.
- One fetch statement can fetch one record only.
- To fetch multiple records, write fetch statement in loop.

Closing the Cursor:

Syntax:

Close <cursor_name>;

Example:

Close c1;

Above statement closes the cursor. It means reference to memory location will be gone.

Cursor Attributes:

Cursor attributes are:

- %found
- %notfound
- %rowcount
- %isopen

Syntax to use cursor attribute:

<cursor_name><attribute_name>;

%found:

- It returns Boolean value true or false.
- If fetch is successful, it returns true.
- If fetch is unsuccessful, it returns false.

%notfound:

- It returns Boolean value true or false.
- It returns true when fetch is unsuccessful.
- It returns false when fetch is successful.

%RowCount:

- It returns number of rows fetched successfully.

%isopen:

- It returns Boolean value true or false.
- It returns true when cursor is open.
- It returns false when cursor is closed.

Program to display emp name and salary:

```
DECLARE
    CURSOR c1 IS SELECT ename,sal FROM emp; --declare
    v_ename emp.ename%type;
    v_sal emp.sal%type;
BEGIN
    OPEN c1; --open cursor
    LOOP
        FETCH c1 INTO v_ename, v_sal; --fetch
        EXIT WHEN c1%NOTFOUND;
        dbms_output.put_line(v_ename || ' ' || v_sal);
    end LOOP;

    CLOSE c1; --closing cursor

END ;
/
```

Output:

ALLEN 1600
WARD 1250
JONES 2975
MARTIN 1250

Program to display empno, ename, job and salary:

```
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 ;
/
```

Output:

```
7499 ALLEN SALESMAN 1600
7521 WARD SALESMAN 1250
7566 JONES MANAGER 2975
7654 MARTIN SALESMAN 1250
```

Program to find sum of salaries of all employees:

```
DECLARE
    CURSOR c1 IS SELECT sal FROM emp;
    v_sal emp.sal%type;
    v_sum INT := 0;
BEGIN
    OPEN c1;

    loop
        FETCH c1 INTO v_sal;
        EXIT WHEN c1%notfound;
        v_sum := v_sum+v_sal;
    end loop;
    dbms_output.put_line('sum of salaries=' || v_sum);
```

```
CLOSE c1;  
END ;  
/
```

Output:

sum of salaries=32225

Cursor For Loop:**Syntax:**

```
for <var> in <cursor_name>  
Loop  
    //Statements  
End Loop;
```

- Cursor For Loop is used to fetch the records from Cursor.
- We have no need to declare loop variable. It will be declared implicitly as %rowtype.
- We have no need to open the cursor, fetch the records from cursor and close the cursor. All these will be done implicitly.

Program to find sum of salaries of all employees using cursor for loop:

```
DECLARE  
    CURSOR c1 IS SELECT sal FROM emp;  
    v_sum INT := 0;  
BEGIN  
  
    FOR r IN c1  
    loop  
        v_sum := v_sum+r.sal;  
    end loop;  
    dbms_output.put_line('sum of salaries=' || v_sum);  
END ;  
/
```

Output:

sum of salaries=32225

Program to display emp name & salary using cursor for loop:

```
DECLARE
    CURSOR c1 IS SELECT ename, sal FROM emp;
BEGIN
    FOR r IN c1
    Loop
        dbms_output.put_line(r.ename || ' ' || r.sal);
    END Loop;
END ;
/
```

Output:

```
ALLEN 1600
WARD 1250
JONES 2975
MARTIN 1250
```

Display all emp names by separating them using comma:

```
DECLARE
    CURSOR c1 IS SELECT ename FROM emp;
    s varchar2(500);
BEGIN
    FOR r IN c1
    Loop
        s := s || r.ename || ',';
    END Loop;

    dbms_output.put_line(RTRIM(s, ','));
END ;
/
```

Output:

```
ALLEN, WARD, JONES, MARTIN
```

Program to demonstrate rowcount attribute:

Create raise_Sal table as follows then write the program:

Empno Per

7844	15
7876	20
7900	10

DECLARE

```
CURSOR c1 IS SELECT * FROM raise_Sal;
```

```
r raise_sal%rowtype;
```

BEGIN

```
OPEN c1;
```

Loop

```
    FETCH c1 INTO r;
```

```
    EXIT WHEN c1%NotFound;
```

```
    UPDATE emp1 SET sal=sal+(sal*r.per/100) WHERE
empno=r.empno;
```

```
END Loop;
```

```
    dbms_output.put_line(c1%ROWCOUNT || ' records
updated...');
```

```
CLOSE c1;
```

END ;

/

Output:

14 records updated...

Program to calculate total marks, average marks and result:

Create std table with following structure and insert the records:

Std

Sid	Sname	M1	M2	M3

Create Result table with following structure:

Sid	Tot	Avrg	Result

```
DECLARE
    CURSOR c1 IS SELECT * FROM std;
    sr std%rowtype;
    rr result%rowtype;
BEGIN
    OPEN c1;

    Loop
        FETCH c1 INTO sr;
        EXIT WHEN c1%notfound ;
        rr.tot := sr.m1+sr.m2+sr.m3;
        rr.avrg := rr.tot/3;
        IF sr.m1>=40 AND sr.m2>=40 AND sr.m3>=40 THEN
            rr.result := 'PASS';
        ELSE
            rr.result := 'FAIL';
        END IF ;

        INSERT INTO result VALUES(sr.sid, rr.tot, rr.avrg, rr.result);

    END LOOP;
    dbms_output.put_line(c1%ROWCOUNT || ' rows inserted...');
    CLOSE c1;
END ;
/
```

Output:

2 rows inserted...

Ref Cursor:

- Using Ref Cursor, select statement can be changed during program execution.
- Normal Cursor Select Statement is static. We cannot change it. Whereas Ref Cursor Select statement is dynamic. We can change it during program execution.
- To declare Ref Cursor variable, we use sys_refcursor data type.
- We can use it as procedure parameter.

Advantages:

- Ref Cursor can be used for multiple select statements.
- It can be used as procedure parameter.

Declaring Ref Cursor:

“sys_refcursor” data type is used to declare the ref cursor variable.

Syntax:

`<variable_name> sys_refcursor;`

Example:

```
c1 sys_refcursor;
```

Assigning Cursor:**Syntax:**

`Open <cursor_name> for <Select_Statement>;`

Example:

```
Open c1 for Select * from emp;
```

.....

.....

```
Open c1 for Select * from Dept;
```

....

....

```
Open c1 for Select * from salgrade;
```

....

Program to display emp table records and dept table records using ref cursor:

```
DECLARE
    c1 sys_refcursor;
    r1 emp%rowtype;
    r2 dept%rowtype;
BEGIN
    OPEN c1 FOR SELECT * FROM emp;
```

```
Loop
    FETCH c1 INTO r1;
    EXIT WHEN c1%NOTFOUND;
    dbms_output.put_line(r1.ename || ' ' || r1.sal);
END Loop;
CLOSE c1;

OPEN c1 FOR SELECT * FROM dept;
Loop
    FETCH c1 INTO r2;
    EXIT WHEN c1%notfound;
    dbms_output.put_line(r2.deptno || ' ' || r2.dname || ' ' ||
r2.loc);
END Loop;
CLOSE c1;
END ;
/
```

Output:

JONES 4975
CLARK 4450
SCOTT 5000
KING 7000
10 ACCOUNTING NEW YORK
20 RESEARCH DALLAS
30 SALES CHICAGO
40 OPERATIONS BOSTON

Types of Cursor:

There are 2 types of cursors. They are:

- Implicit Cursor
- Explicit Cursor

Implicit Cursor:

- The cursor created implicitly to process SELECT or DML Statements.
- Implicit Cursor name is => SQL

- We use cursor attributes as follows for implicit cursor:
SQL%NOTFOUND
SQL%FOUND
SQL%ISOPEN
SQL%ROWCOUNT

Explicit Cursor:

The cursor is declared and defined by developer.

Explicit cursors are 2 types. They are:

- Normal Cursor => can be used for one select statement.
- Ref Cursor => can be used for multiple select statements.

Exception Handling

Types of Errors:

There are 3 types of errors. They are:

- Compile Time Errors
- Logical Errors
- Run-Time Errors

Compile Time Errors:

The errors occur at compile time are called “Compile Time Errors”. These errors occur due to syntax mistakes.

Example:

missing parenthesis [] , missing semicolon [;] , missing single quotes ['] ...etc.

Logical Errors:

The errors occur due to mistakes in logic is called “Logical Error”. It will not be checked by the Oracle. Developer is responsible to write the correct logic. Otherwise, we get wrong results.

Example:

For Depositing amount logic is add amount to balance. If we subtract, it is logic error.

Run-Time Errors:

The errors occur at run-time are called “Run-Time errors”. Run-Time errors are very dangerous. When run-time error occurs, application will be terminated abnormally in the middle of execution. It will not execute remaining code.

Example:

Divide with zero

Record not found in table

Inserting duplicate value in primary key field

Exception Handling:

Exception means Run-Time Error. The mechanism of handling run-time errors is called “Exception Handling”.

Advantages:

- It avoids abnormal termination.
- Executes remaining code.

In PL/SQL, define “Exception” block in PL/SQL program for Exception Handling.

Syntax for Exception Handling:

```
DECLARE
    <declaration_part>
BEGIN
    <execution_part>

    EXCEPTION
        When <Exception_Name> Then
            --Handling Code
        When <Exception_Name> Then
            --Handling Code
        .
        .
        When Others Then
            --Handling Code
    End;
    /
```

Types of Exceptions:

There are 2 types of exceptions. They are:

- System-Defined Exceptions
- User-Defined Exceptions

System-Defined Exceptions:

The exception raised implicitly by the Oracle is called “System-Defined Exception”.

Examples:

ZERO_DIVIDE

VALUE_ERROR
NO_DATA_FOUND
DUP_VAL_ON_INDEX
TOO_MANY_ROWS

ZERO_DIVIDE:

This exception will be raised when we try to divide with zero.

VALUE_ERROR:

This exception will be raised when size is exceeded, or data type is mismatched.

NO_DATA_FOUND:

This exception will be raised when data is not found in the table.

DUP_VAL_ON_INDEX:

This exception will be raised when we try to insert duplicate value in primary key column.

TOO_MANY_ROWS:

This exception will be raised when select query returns multiple rows.

User-Defined Exceptions:

The exception declared and raised by the user is called "User-Defined Exception".

Examples:

abc
xyz
comm_is_null
one_divide

Defining User-Defined Exceptions:

We follow 2 steps for user-defined exception. They are:

1. Declare the Exception
2. Raise the Exception

1. Declare the Exception:

“Exception” type is used to declare the Exception.

Syntax:

```
Exception_name Exception;
```

Example:

```
abc Exception;  
one_divide Exception;
```

2. Raise the Exception:

“Raise” keyword is used to raise the exception.

Syntax:

```
Raise Exception_Name;
```

Example:

```
Raise abc;  
Raise one_divide;
```

RAISE_Application_Error:

In PL/SQL we can raise user-defined exception in 2 ways. They are:

- Using RAISE keyword
 - Using RAISE_APPLICATION_ERROR
-
- RAISE_APPLICATION_ERROR is a procedure that is used to raise the error with user defined message & code.
 - Oracle provides error numbers ranges from -20000 to -20999 for user-defined exceptions.

Syntax:

```
Raise_Application_Error(<Error_Code>, <Error_Message>);
```

Example:

```
Raise_Application_Error(-20050,'u cannot divide with one');
```

pragma exception_init:

There are 2 types in system-defined exceptions. They are:

- Named Exceptions
- Unnamed Exceptions

Named Exceptions:

The exceptions which are having names are called “Named Exceptions”.

Example:

- No_DATA_FOUND
- ZERO_DIVIDE
- DUP_VAL_ON_INDEX
- VALUE_ERROR
- TOO_MANY_ROWS

Unnamed Exceptions:

The errors which are not having any names are called “Unnamed Exceptions”. Error code & messages are there but name is not there for unnamed exceptions.

Example:

Check Constraint

For exception handling, exception block will be defined. In exception block with the help of exception name we handle the exception as following:

Exception

```
when zero_divide then  
    dbms_output.put_line('cannot divide with zero');
```

“pragma exception_init” is used to define the name for unnamed errors

pragma => is a directive [command]

exception_init() => is a function

Syntax:

```
pragma exception_init(<exception_name>, <error code>);
```

Example:

```
abc Exception;  
pragma exception_init(abc,-2990);
```

Example Programs on Exception Handling:**Program-1:****Program to demonstrate ‘ZERO_DIVIDE’ Exception:**

```
DECLARE  
    x number(4);  
    y number(4);  
    z FLOAT ;  
BEGIN  
    x := &x;  
    y := &y;  
  
    z := x/y;  
  
    dbms_output.put_line('quotient=' || z);  
  
EXCEPTION  
    WHEN zero_divide THEN  
        dbms_output.put_line('u cannot divide with zero');  
    WHEN value_error THEN  
        dbms_output.put_line('check the input. size or data type  
mistake');  
    WHEN others THEN  
        dbms_output.put_line('run time error occurred');  
END ;  
/
```

Output:**Case-1:**

Enter value for x: 10

Enter value for y: 2

quotient=5

Case-2:

Enter value for x: 10

Enter value for y: 0

u cannot divide with zero

Case-3:

Enter value for x: 10

Enter value for y: 'ravi'

check the input. size or data type mistake

Program-2:**Program to demonstrate 'NO_DATA_FOUND' Exception:**

DECLARE

```
v_empno emp.empno%type;  
v_ename emp.ename%type;  
v_sal emp.sal%type;
```

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);
```

EXCEPTION

```
WHEN no_data_found THEN
```

```
    dbms_output.put_line('this employee record not  
available');
```

```
WHEN others THEN
```

```
    dbms_output.put_line('run time error occurred');
```

END ;

/

Output:**Case-1:**

Enter value for empno: 7369

SMITH 18996.16

Case-2:

Enter value for empno: 9001

this employee record not available

Program-3:**Program to demonstrate DUP_VAL_ON_INDEX Exception:**

Create a table with the name t1 with the field f1 as number type & set it as primary key:

Create table t1(f1 number(4) primary key);

DECLARE

 v_f1 t1.f1%type;

BEGIN

 v_f1 := &f1;

 INSERT INTO t1 VALUES(v_f1);

 COMMIT ;

 dbms_output.put_line('row inserted');

EXCEPTION

 WHEN DUP_VAL_ON_INDEX THEN

 dbms_output.put_line('PK should not accept
duplicate values');

END ;

/

Output:**Case-1:**

Enter value for f1: 1001

row inserted

Case-2:

Enter value for f1: 1001

PK should not accept duplicate values

Program-4:**Program to demonstrate TOO_MANY_ROWS Exception:**

DECLARE

```
v_job varchar2(10) ;  
v_ename varchar2(100) ;  
v_sal FLOAT ;
```

BEGIN

```
v_job := '&job';
```

```
SELECT ename, sal INTO v_ename, v_Sal FROM emp  
WHERE job=v_job;
```

```
dbms_output.put_line(v_ename || ' ' || v_Sal);
```

EXCEPTION

```
WHEN too_many_rows THEN
```

```
    dbms_output.put_line('multiple rows selected..!');
```

END ;

/

Output:**Case-1:**

Enter value for job: CLERK

SMITH 18996.16

Case-2:

Enter value for job: MANAGER

multiple rows selected..!

Program-5:**Program to demonstrate User-Defined Exception:**

```
DECLARE
    x INT ;
    y INT ;
    z FLOAT ;
    one_divide Exception;
BEGIN
    x := &x;
    y := &y;
    IF y=1 THEN
        raise one_divide;
    ELSE
        z := x/y;
        dbms_output.put_line('quotient=' || z);
    END IF ;

    EXCEPTION
        WHEN one_divide THEN
            dbms_output.put_line('u cannot divide with 1');
        WHEN zero_divide THEN
            dbms_output.put_line('u    cannot    divide    with
zero');

END ;
/
```

Output:**Case-1:**

Enter value for x: 10

Enter value for y: 2

quotient=5

Case-2:

Enter value for x: 10

Enter value for y: 1

u cannot divide with 1

Case-3:

Enter value for x: 10
Enter value for y: 0
u cannot divide with zero

Program-6:

Program to update salary of given empno. If employee commission is null don't update the salary and raise the exception.

```
DECLARE
    v_empno emp.empno%type;
    v_comm emp.comm%type;
    comm_is_null Exception;
BEGIN
    v_empno := &empno;

    SELECT comm INTO v_comm FROM emp WHERE
        empno=v_empno;

    IF v_comm IS NULL THEN
        raise comm_is_null;
    ELSE
        UPDATE emp SET sal=sal+2000 WHERE empno=v_empno;
        COMMIT ;
        dbms_output.put_line('record updated..');
    END IF ;

    EXCEPTION
        WHEN comm_is_null THEN
            dbms_output.put_line('record not updated. because
comm is null');

END ;
/
```

Output:**Case-1:**

Enter value for empno: 7521

record updated..

Case-2:

Enter value for empno: 7369

record not updated. because comm is null

Program-7:**Program to demonstrate RAISE_APPLICATION_ERROR:**

DECLARE

```
v_empno emp.empno%type;
v_comm emp.comm%type;
comm_is_null Exception;
```

BEGIN

```
v_empno := &empno;
```

```
SELECT comm INTO v_comm FROM emp
WHERE empno=v_empno;
```

```
IF v_comm IS NULL THEN
```

```
    Raise_Application_Error(-20050,'COMM IS NULL. cannot
update..!');
```

```
ELSE
```

```
    UPDATE emp SET sal=sal+2000 WHERE empno=v_empno;
```

```
    COMMIT ;
```

```
    dbms_output.put_line('record updated..!');
```

```
END IF ;
```

```
END ;
```

```
/
```

Output:**Case-1:**

Enter value for empno: 7369

DECLARE

*

ERROR at line 1:
ORA-20050: COMM IS NULL. cannot update..!
ORA-06512: at line 12

Case-2:

Enter value for empno: 7521
record updated..!

Program-8:

Program to not to allow the user update on Sunday. If user tries to update on Sunday then Raise the error using RAISE_APPLICATION_ERROR:

```
DECLARE
    v_empno emp.empno%type;

BEGIN
    v_empno := &empno;

    IF to_char(sysdate,'dy')='sun' THEN
        raise_Application_error(-20070,'Sunday      updation      not
allowed..!');

    ELSE
        UPDATE emp SET sal=sal+2000 WHERE empno=v_empno;
        COMMIT ;
        dbms_output.put_line('record updated..!');
    END IF ;

END ;
/
```

Output:**Case-1 [On Sunday]:**

Enter value for empno: 7369

```
DECLARE
```

```
*
```

ERROR at line 1:
ORA-20070: Sunday updation not allowed..!
ORA-06512: at line 6

Case-2 [Monday to Saturday]:

Enter value for empno: 7369

record updated..!

Program-9:**Program to demonstrate PRAGMA EXCEPTION_INIT:**

Create a table with the name t1 as following:

T1

Sid => Primary Key

Sname => Not Null

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

Create table t1

```
(  
    Sid number(4) Primary Key,  
    Sname Varchar2(10) Not Null,  
    M1 Number(3) Check(M1>=0 and M1<=100)  
);
```

DECLARE

```
    v_sid t1.sid%type;  
    v_sname t1.sname%type;  
    v_m1 t1.m1%type;  
    check_violate EXCEPTION;  
    pragma exception_init(check_violate,-2290);
```

BEGIN

```
    v_sid := &stdid;  
    v_sname := '&sname';  
    v_m1 := &m1;
```

```
    INSERT INTO t1 VALUES(v_sid, v_sname, v_m1);  
    COMMIT ;  
    dbms_output.put_line('row inserted..!');
```

Exception

```
    WHEN dup_val_on_index THEN
```

```
        dbms_output.put_line('dup values not allowed in PK..!');  
WHEN check_violate THEN  
        dbms_output.put_line('check constraint violated..!');  
  
END ;  
/
```

Output:**Case-1:**

```
Enter value for stdid: 1001  
Enter value for sname: Ravi  
Enter value for m1: 50  
row inserted..!
```

Case-2:

```
Enter value for stdid: 1001  
Enter value for sname: Kiran  
Enter value for m1: 75  
dup values not allowed in PK..!
```

Case-3:

```
Enter value for stdid: 1002  
Enter value for sname: Sai  
Enter value for m1: 123  
check constraint violated..!
```

Stored Procedures

- Procedure is a named PL/SQL block of statements that gets executed on calling.
- Every procedure is defined to perform DML operation.
- We cannot call a procedure from SQL Command.
- “Execute” command is used to call the procedure from SQL Prompt.
- A procedure cannot return the value. But we can send the values out of a procedure using OUT parameters.

Types of Procedures:

There are 2 types of procedures. They are:

- Stored Procedures
- Packaged Procedures

Stored Procedure:

A procedure which is defined in schema[user] is called “Stored Procedure”.

Packaged Procedure:

A procedure which is defined in package is called “Packaged Procedure”.

Syntax to define a procedure:

```
Create or Replace procedure <name>(<argument_list>) IS/AS  
    <declararion-part>  
    BEGIN  
        <execution-part>  
    END;  
    /
```

A Procedure can be called from 3 places:

- From PL/SQL program
- From SQL Prompt
- From Front-End Applications

Types of Parameters:

The local variable which is declared in procedure header is called “Parameter”. It can be also called as “Argument”.

There are 3 types of parameters. They are:

- IN Parameters [default]
- OUT Parameters
- IN OUT Parameters

Syntax to define a parameter:

`<parameter_name> [<parameter_type>] <data_type>`

Example:

x IN Number
y OUT Number
z IN OUT Number

IN parameter:

- It is default one.
- It is used to get value into procedure from out of the procedure.
- It is READ-ONLY Parameter. We cannot change this parameter value in the procedure.
- ‘IN’ keyword is used to define it.
- It can be variable or constant value from the procedure call.

OUT Parameter:

- It is used to send the value out of the procedure.
- We must use ‘OUT’ keyword to define it.
- It must be variable from the procedure call. It cannot be constant.

IN OUT Parameter:

- It is used to get value into procedure & same variable can be used to send value out of the procedure.
- We must use ‘IN OUT’ keyword to define it.
- It must be variable from the procedure call. It cannot be constant.

Parameter mapping techniques:

There are 3 parameter mapping techniques. They are:

- Positional Notation
- Named Notation
- Mixed Notation

Positional Notation:

In this, parameters are mapped based on the positions.

Named Notation:

In this, parameters are mapped based on the names.

Mixed Notation:

In this, parameters are mapped based on positions and names. A positional parameter cannot be followed by named parameter.

Example:

```
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;
/
```

Positional Notation:

SQL> execute addnum(10,20,30);

Output:

sum=60

Named Notation:

SQL> execute addnum(y=>10,z=>20,x=>30);

Output:

sum=60

Mixed Notation:

```
SQL> execute addnum(10,z=>20,y=>30);
```

Output:

```
sum=60
```

Note:

```
SQL> execute addnum(z=>20,30,x=>10);
```

Output:

```
ERROR at line 1:
```

```
ORA-06550: line 1, column 20:
```

```
PLS-00312: a positional parameter association may not follow a named  
association
```

Example Programs on Procedures:**Program-1:****Procedure to add two numbers:**

```
CREATE OR REPLACE PROCEDURE addition(x number,y number)  
IS  
    z number;  
BEGIN  
    z:=x+y;  
    dbms_output.put_line('sum=' || z);  
End;  
/
```

--MAIN Program:

```
DECLARE  
    a number;  
    b number;  
BEGIN  
    a:=&a;  
    b:=&b;  
    addition(a,b); --calling from main program  
END ;  
/
```

Output:

Enter value for a: 20

Enter value for b: 30

sum=50

Calling from SQL prompt:

SQL> execute addition(10,20);

sum=30

Program-2:**Procedure to add two number & send result out of the procedure:**

CREATE OR REPLACE PROCEDURE

```
addnum(  
x IN number,  
y IN number,  
z OUT number)
```

IS

BEGIN

 z := x+y;

END ;

/

--MAIN Program:

DECLARE

```
    a number;  
    b number;  
    c number;
```

BEGIN

 a := &a;

 b := &b;

 addnum(a,b,c); --calling from main program

 dbms_output.put_line('sum=' || c);

END ;

/

Output:

Enter value for a: 50

Enter value for b: 40

sum=90

Calling from SQL prompt:

SQL> variable z number

SQL> execute addnum(5,4,:z);

PL/SQL procedure successfully completed.

SQL> print z

z

9

Program-3:

Procedure to find biggest in two different numbers & send biggest number out of the procedure:

CREATE OR REPLACE PROCEDURE

bigin2(
x IN number,
y IN number,
big OUT number)

IS

BEGIN

IF x>y THEN
 big := x;

ELSE
 big := y;

END IF ;

END;

/

Calling from SQL Prompt:

SQL> variable big number

```
SQL> execute begin2(10,20,:big);
```

```
PL/SQL procedure successfully completed.
```

```
SQL> print big
```

```
BIG
```

```
-----  
20
```

Program-4:**Procedure to insert a record into table & return status:****Create table as following:**

```
Create Table std11
```

```
(
```

```
    Sid number(4) Primary Key,  
    Sname varchar2(10) Not Null,  
    M1 number(3) Check(M1>=0 and M1<=100)
```

```
);
```

```
Create or Replace Procedure
```

```
insert_std(p_sid std11.sid%type,
```

```
          p_sname std11.sname%type,
```

```
          p_m1 std11.m1%type,
```

```
          p_msg OUT varchar2)
```

```
IS
```

```
BEGIN
```

```
    Insert into std11 values(p_sid,p_sname,p_m1);
```

```
    p_msg := 'Record Inserted';
```

```
Exception
```

```
    WHEN others THEN
```

```
        p_msg:=SQLERRM;
```

```
END ;
```

```
/
```

Calling from SQL Prompt:

```
SQL> variable s varchar2(100)
SQL> execute insert_std(1,'A',45,:s);
```

PL/SQL procedure successfully completed.

```
SQL> print s
```

S

Record Inserted

```
SQL> execute insert_std(1,'A',45,:s);
```

PL/SQL procedure successfully completed.

```
SQL> print s
```

S

ORA-00001: unique constraint (C##RAJU.SYS_C008461) violated

Program-5:**Procedure to update salary of an employee with specific amount:**

```
CREATE OR REPLACE PROCEDURE
salary_increment(p_empno emp.empno%type, p_amt number)
IS
BEGIN
    UPDATE emp SET sal=sal+p_amt WHERE empno=p_empno;
    COMMIT ;
    dbms_output.put_line('salary incremented..!');
END;
/
```

--MAIN Program:

```
DECLARE
    v_empno emp.empno%type;
    v_amt number;
BEGIN
    v_empno:=&empno;
    v_amt:=&amount;
    salary_increment(v_empno,v_amt);
END ;
/
```

Output:

```
Enter value for empno: 7876
Enter value for amount: 1000
salary incremented..!
```

Calling from SQL Prompt:

```
SQL> execute salary_increment(7566,1500);
salary incremented..!
```

Program-6:**Procedure to update salary of an employee with specific amount & send updated salary out of the procedure:**

```
CREATE OR REPLACE PROCEDURE
salary_increment(p_empno emp.empno%type, p_amt number,
p_sal OUT emp.sal%type)
IS
BEGIN
    UPDATE emp SET sal=sal+p_amt WHERE empno=p_empno;
    COMMIT ;

    SELECT sal INTO p_sal FROM emp WHERE empno=p_empno;
END ;
/
```

--MAIN Program:

```

DECLARE
    v_empno emp.empno%type;
    v_amt number;
    v_sal emp.sal%type;
BEGIN
    v_empno:=&empno;
    v_amt:=&amount;
    salary_increment(v_empno,v_amt,v_sal);
    dbms_output.put_line('updated salary is:' || v_Sal);
END ;
/

```

Output:

Enter value for empno: 7369
 Enter value for amount: 2000
 updated salary is:17996.16

Calling from SQL Prompt:

```

SQL> variable sal number
SQL> execute salary_increment(7844,1000,:sal);
      PL/SQL procedure successfully completed.
SQL> print sal

```

SAL

 15517.08

Program-7:**Procedure to deposit the amount in account:****Create Table & Insert records as following:**

Create Table Account

(

 Acno Number(4),
 Name Varchar2(10),
 Balance Number(8,2)

);

```
Insert into Account values(1001,'A',100000);
Insert into Account values(1002,'B',200000);
```

Procedure:

```
CREATE OR REPLACE PROCEDURE
deposit(p_acno account.acno%type, p_amt number,
p_balance OUT account.balance%type)
IS
BEGIN
    UPDATE account SET balance=balance+p_amt
    WHERE acno=p_acno;
    COMMIT ;
    dbms_output.put_line('Deposit is successful..!');
    SELECT balance INTO p_balance FROM account
    WHERE acno=p_acno;
END ;
/
```

Calling from SQL Prompt:

```
SQL> variable bal number
SQL> execute deposit(1001,25000,:bal);
Deposit is successful..!
```

PL/SQL procedure successfully completed.

```
SQL> print bal
```

```
BAL
```

```
-----
```

```
125000
```

```
SQL> select * from account;
```

ACNO	NAME	BALANCE
1001	A	125000
1002	B	200000

Program-8:**Procedure to withdraw the amount from account:**

Create Or Replace Procedure

```
withdraw(p_acno account.acno%type,  
p_amt number,  
p_balance OUT account.balance%type)  
IS  
BEGIN
```

```
    Select balance into p_balance from ACCOUNT where acno=p_acno;
```

```
    if p_amt>p_balance then  
        RAISE_APPLICATION_ERROR(-20050,'Insufficient Balance');  
    else  
        Update account set balance=balance-p_amt  
        where acno=p_acno;  
        COMMIT ;  
        dbms_output.put_line('Successful withdrawl..!');
```

```
    Select balance into p_balance from ACCOUNT where acno=p_acno;
```

```
    end if ;  
END ;
```

```
/
```

Calling from SQL Prompt:

```
SQL> variable bal number
```

```
SQL> execute withdraw(1001,5000,:bal);
```

PL/SQL procedure successfully completed.

```
SQL> print bal
```

```
BAL
```

```
-----
```

```
120000
```

SQL> select * from account;

ACNO	NAME	BALANCE
1001	A	120000
1002	B	200000

Program-9:

Procedure to demonstrate pragma autonomous_transaction:

CREATE OR REPLACE PROCEDURE

```
salary_increment(p_empno emp.empno%type, p_amt number)
IS
```

```
    pragma autonomous_transaction;
```

```
BEGIN
```

```
    UPDATE emp SET sal=sal+p_amt
```

```
    WHERE empno=p_empno;
```

```
    ROLLBACK ;
```

```
    dbms_output.put_line('rolled back');
```

```
END ;
```

```
/
```

--MAIN Program:

```
BEGIN
```

```
    UPDATE emp SET sal=sal+2000 WHERE empno=1234;
```

```
    salary_increment(7499,1000);
```

```
    COMMIT ;
```

```
END ;
```

```
/
```

Output:

rolled back

Note:

Above program rollbacks 7499 emp salary updating. Commits 1234 emp salary updating. Two different transactions are created for procedure and main program because of “pragma autonomous_transaction”.

Stored Functions

- Function is a named block of statements that gets executed on calling.
- Functions are defined to perform calculations or fetching operations.
- A Function can return the value. It can return one value only. It cannot return multiple values.
- Function can be called from SQL command.

Types of Functions:

There are 2 types of functions. They are:

- Stored Functions
- Packaged Functions

Stored Function:

A function which is defined in schema[user] is called “Stored Function”.

Packaged Function:

A function which is defined in package is called “Packaged Function”.

Syntax to define a function:

```
Create or Replace Function <name>(<argument_list>)
Return <type> IS/AS
    <declararion-part>
BEGIN
    <execution-part>
END;
/
```

A Function can be called from 3 places:

- From PL/SQL program
- From SQL Prompt
- From Front-End Applications

Example Programs on Functions:

Program-1:

Define a Function to perform Arithmetic Operations:

Create or Replace Function calc(x number,y number, op char)

Return Number

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 ;

/

--MAIN Program:

DECLARE

```
a number;
b number;
op char ;
r number;
```

BEGIN

```
a:=&a;
b:=&b;
op:='&op';
```

r:=calc(a,b,op);

dbms_output.put_line(r);

END ;

/

Output:**Case-1:**

```
Enter value for a: 10
Enter value for b: 2
Enter value for op: +
12
```

Case-2:

```
Enter value for a: 10
Enter value for b: 2
Enter value for op: -
8
```

Case-3:

```
Enter value for a: 10
Enter value for b: 2
Enter value for op: *
20
```

Case-4:

```
Enter value for a: 10
Enter value for b: 2
Enter value for op: /
5
```

Calling from SQL Prompt:

```
SQL> select calc(5,4,'+') from dual;
```

Output:

```
CALC(5,4,'+')
```

```
-----  
9
```

```
SQL> select calc(5,4,'*') from dual;
```

Output:

```
CALC(5,4,'*')
```

```
-----  
20
```

Program-2:**Define a function to get 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 ;

/

Calling from SQL prompt:

SQL> select check_balance(1001) from dual;

CHECK_BALANCE(1001)

120000**Program-3:****Define a function to get experience of an employee:**

Create or Replace Function experience(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 ;

/

Calling from SQL Prompt:

SQL> select experience(7369) from dual;

Output:

EXPERIENCE(7369)

41**Program-4:****Define a function to display employees of a dept:**

Create or Replace Function

getdeptemps(p_deptno emp.deptno%type)

Return sys_refcursor

IS

c1 sys_refcursor;

Begin

Open c1 for select * from emp where deptno=p_deptno;

Return c1;

End;

/

Calling from SQL Prompt:

SQL> select getdeptemps(10) from dual;

Output:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7782	CLARK	MANAGER	7839	09-JUN-81	18424		10
7839	KING	PRESIDENT		17-NOV-81	21987.75		10

Program-5:**Define a function to display top n employees as per salary:**

CREATE OR REPLACE FUNCTION

getttopn(n number) 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 ;
/

```

Calling from SQL Prompt:

SQL> select gettopn(3) from dual;

Output:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7521	WARD	SALESMAN	7698	22-FEB-81	33549.06	500	30
7369	SMITH	CLERK	7902	17-DEC-80	22996.16		20
7839	KING	PRESIDENT		17-NOV-81	21987.75		10

Program-6:**Define a Function to check an year is leap or not:**

Create or Replace Function
isleap(y number) return varchar2
IS

```

        d Date;
BEGIN
        d := '29-FEB-' || y; --29-feb-2020
        return 'Leap Year';
    Exception
        when others then
            return 'Not Leap Year';
End ;
/

```

Calling from SQL Prompt:

SQL> select isleap(2020) from dual;

ISLEAP(2020)

Leap Year

SQL> select isleap(2021) from dual;
ISLEAP(2021)

Not Leap Year

Program-7:

Define a function to calculate total amount of an order:

Create Table & Insert records as following:

Orders o

oid	pid	qty
1001	2001	3
1001	2002	5
1002	2001	6
1002	2002	8

Products p

pid	pname	price
2001	A	500
2002	B	200

Create Or Replace Function
Order_Amount(p_oid orders.oid%type)

Return Number

IS

```
Cursor c1 IS Select o.qty, p.price
FROM orders o INNER JOIN Products p
ON o.pid=p.pid
WHERE o.oid=p_oid;
t number;
```

BEGIN

```
t:=0;
for r in c1
Loop
```

```
t:=t+r.qty*r.price;  
End Loop;  
Return t;  
END;  
/
```

Calling from SQL Prompt:

```
SQL> select order_amount(1001) from dual;
```

```
ORDER_AMOUNT(1001)
```

```
-----  
2500
```

```
SQL> select order_amount(1002) from dual;
```

```
ORDER_AMOUNT(1002)
```

```
-----  
4200
```

Packages

- Package is a named PL/SQL block.
- It is a collection of procedures, functions, variables ..etc.
- It is used to group the related procedures & functions.

Example:

Package Bank

```
opening_Account procedure  
closing_Account procedure  
deposit procedure  
withdraw procedure  
checkbal function  
transaction_statement function
```

Advantages:

- We can group related procedures & functions.
- We can declare global variables. A global variable can be accessed anywhere.
- It provides better maintenance.
- It provides better performance.
- We can overload the functions & procedures in a package.
- It provides Reusability.

Creating a Package:

To define a package we follow 2 steps. They are:

- Define package specification
- Define package body

Package Specification:

In this we declare the procedures and functions.

Syntax for Package Specification:

```
Create Or Replace Package <name>
IS / AS
    Procedure Declarations
    Function Declarations
End;
/
```

Package Body:

In this we define body of procedures and functions.

Syntax for Package Body:

```
Create Or Replace Package Body <name>
IS/AS
    DEFINE PROCEDURES
    DEFINE FUNCTIONS
END;
/
```

Example Programs on Packages:**Program-1:**

Define a package with following procedures and functions:

- Hiring an employee
- Firing an employee
- Updating Salary
- Getting Experience
- Getting top-n salaried employees

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 salary_increment(p_empno  emp.empno%type,p_amt
number);
FUNCTION gettopn(n number) 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)
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 salary_increment(p_empno emp.empno%type,p_amt
number)**

IS

```

BEGIN
    UPDATE emp SET sal=sal+p_amt
    WHERE empno=p_empno;
    COMMIT ;
    dbms_output.put_line('record updated..!');
END salary_increment;

FUNCTION gettopn(n number) Return sys_refcursor
IS
    c1 sys_refcursor;
BEGIN
    OPEN c1 FOR SELECT * FROM emp
    ORDER BY sal DESC nulls last 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;
/

```

Calling Packaged procedures and Packaged functions from SQL prompt:
SQL> select hr.gettopn(3) from dual;

Output:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7521	WARD	SALESMAN	7698	22-FEB-81	33549.06	500	30
7369	SMITH	CLERK		7902 17-DEC-80	22996.16		20

```
SQL> select hr.getexp(7369) from dual;
```

Output:

```
HR.GETEXP(7369)
```

```
-----  
41
```

```
SQL> execute hr.hire(5001,'Ramu');
```

Output:

```
record inserted..!
```

```
SQL> execute hr.salary_increment(7369,1000);
```

Output:

```
record updated..!
```

```
SQL> execute hr.fire(5001);
```

Output:

```
record deleted..!
```

Program-2:

Create a package with the name ‘BANK’ with following procedures and functions:

--creating package specification

```
CREATE OR REPLACE package BANK
```

```
IS
```

```
    PROCEDURE deposit(p_acno account.acno%type, p_amt number);
```

```
    PROCEDURE withdraw(p_acno account.acno%type, p_amt number);
```

```
    FUNCTION checkbalance(p_acno account.acno%type) RETURN  
number;
```

```
END ;
```

```
/
```

--creating package body

```
CREATE OR REPLACE package body BANK
IS
PROCEDURE deposit(p_acno account.acno%type, p_amt number)
IS
BEGIN
UPDATE Account SET balance=balance+p_amt WHERE acno=p_acno;
COMMIT ;
dbms_output.put_line('amount deposited...');
END deposit;

PROCEDURE withdraw(p_acno account.acno%type, p_amt number)
IS
v_balance account.balance%type;
BEGIN
SELECT balance INTO v_balance FROM account WHERE
acno=p_acno;
IF p_amt>v_balance THEN
raise_application_error(-20050,'Insufficient Balance...!');
END IF ;
UPDATE account SET balance=balance-p_amt WHERE acno=p_acno;
COMMIT ;
dbms_output.put_line('amount withdrawn...!');
END withdraw;

FUNCTION checkbalance(p_acno account.acno%type) RETURN
number
IS
v_balance account.balance%type;
BEGIN
SELECT balance INTO v_balance FROM account WHERE
acno=p_acno;
RETURN v_balance;
END checkbalance;
END ;
/
```

Calling Packaged procedures and Packaged functions from SQL prompt:

```
SQL> execute bank.deposit(1001,30000);
```

Output:

amount deposited...

```
SQL> select * from account;
```

ACNO	NAME	BALANCE
1001	A	150000
1002	B	200000

```
SQL> execute bank.withdraw(1001,20000);
```

Output:

amount withdrawn...!

```
SQL> select * from account;
```

ACNO	NAME	BALANCE
1001	A	130000
1002	B	200000

Program-3:**Program to demonstrate Overloading:****--creating package specification**

```
CREATE OR REPLACE package p1
```

```
IS
```

```
    FUNCTION addn(x number, y number) RETURN number;
```

```
    FUNCTION addn(x number, y number,z number) RETURN number;
```

```
END ;
```

```
/
```

--creating package body

```
CREATE OR REPLACE package body p1
IS
    FUNCTION addn(x number, y number) RETURN number
    IS
        BEGIN
            RETURN x+y;
        END addn;

    FUNCTION addn(x number, y number,z number) RETURN number
    IS
        BEGIN
            RETURN x+y+z;
        END addn;

END ;
/
```

Calling from SQL Prompt:

```
SQL> select p1.addn(10,20) from dual;
```

Output:

```
P1.ADDN(10,20)
```

```
30
```

```
SQL> select p1.addn(10,20,30) from dual;
```

Output:

```
P1.ADDN(10,20,30)
```

```
60
```

Triggers

- Trigger is a named block of statements that gets executed automatically when user submits DML/DDL Command.
- Procedure must be called explicitly. We have no need to call the Trigger. Oracle calls the Trigger implicitly when we submit the DML/DDL Command.

Trigger is mainly used for 3 purposes. They are:

- Trigger can be used to control the DMLs.
- It can be used for auditing. Auditing means, which user is performing which action on which table at which time can be recorded to implement the security.
- It can be used to implement complex rules or data validations.

Types of Triggers:

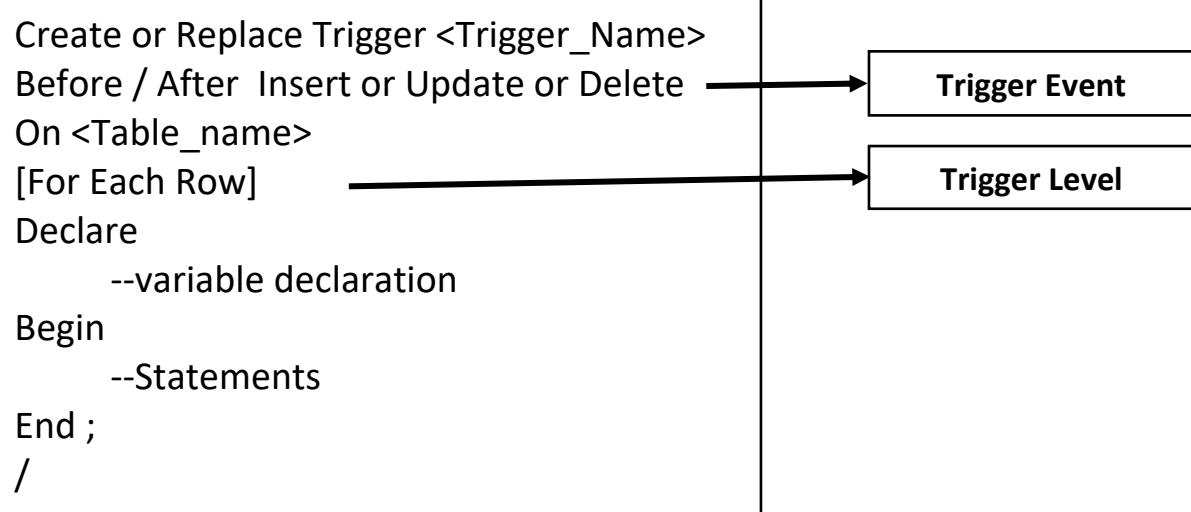
There are 3 types of Triggers. They are:

- Table Level Triggers
- Schema Level Triggers
- Database Level Triggers

Table Level Triggers:

A trigger which is created on table is called “Table Level Trigger”.

Syntax:



Before Trigger:

- Before Trigger gets executed before performing DML operation.
- First trigger gets executed.
- Then DML operation will be performed.

After Trigger:

- After Trigger gets executed after performing DML operation.
- First DML operation will be performed.
- Then trigger gets executed.

There are 2 Trigger Levels in Table Level Triggers. They are:

- Statement Level Trigger
- For Each Row Trigger

Statement Level Trigger:

A trigger gets executed only once for a Statement [Command] is called “Statement Level Trigger”.

For Each Row Trigger:

A trigger gets executed once for each row affected by DML operation is called “For Each Row Trigger”.

Schema Level Triggers:

A trigger which is created on schema [user] is called “Schema Level Trigger”. DBA creates Schema Level Triggers. It is used to control one user.

Syntax:

```
Create or Replace Trigger <Trigger_Name>
Before / After <trigger_event>
On <User_name>.schema
Declare
    --variable declaration
Begin
    --Statements
End ;
/
```

Database Level Triggers:

A trigger which is created on multiple users is called “Database Level Trigger”. DBA creates Database Level Triggers. It is used to control multiple users.

Syntax:

```
Create or Replace Trigger <Trigger_Name>
Before / After <trigger_event>
On Database
Declare
    --variable declaration
Begin
    --Statements
End ;
/

```

:new & :old variables:

- These are the bind variables.
- These are %rowtype variables.
- These can be also called as pseudo records.
- These can be used in trigger only.
- To access the table data in trigger, use these bind variables.
- These can be used in row level triggers only. Cannot be used in statement level triggers.

DML operation	:New	:Old
INSERT	Record will be copied to :new variable	Null
DELETE	Null	Record will be copied to :old variable
UPDATE	New record will be copied to :new variable	Old Record will be copied to :old variable

Compound Trigger:

- It is introduced in Oracle 11g.
- It is a set of triggers.
- It is used to define multiple actions in one trigger.
- Instead of defining multiple different triggers on one table, we can define multiple actions in one 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;
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>;
/
```

System Variables:

System Variable	Purpose
ora_dict_obj_type	returns object type like TABLE, VIEW, INDEX ..etc
ora_dict_obj_name	returns object name like EMP, DEPT ..etc
ora_login_user	returns user name
ora_sysevent	returns the action CREATE, DROP, TRUNCATE

Example Programs on Triggers:**Program-1:****Define a Trigger not to allow the user to update emp data on sunday:**

```

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,'sunday      updation      not
allowed');
    END IF ;
END ;
/

```

Testing [On Sunday]:

SQL> update emp set sal=sal+2000 where empno=7369;

Output:

update emp set sal=sal+2000 where empno=7369

*

ERROR at line 1:

ORA-20050: sunday updation not allowed

ORA-06512: at "C##RAJU.T1", line 3

ORA-04088: error during execution of trigger 'C##RAJU.T1'

Program-2:

Define a Trigger to allow the user to update emp data as following:

From Monday to Friday => between 10 to 4 only

On Saturday => between 10 to 2 only

On Sunday => don't allow for updation

CREATE OR REPLACE TRIGGER t2

Before INSERT OR DELETE OR UPDATE

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,'u can update b/w 10 to 4
only from mon to fri');

 elsif wd=7 AND h NOT BETWEEN 10 AND 14 THEN

 RAISE_APPLICATION_ERROR(-20070,'u can update b/w
10 to 2 only on sat');

 elsif wd=1 THEN

 RAISE_APPLICATION_ERROR(-20090,'sunday not allowed');

 END IF ;

END ;

/

Testing [On Saturday after 2 or before 10]:

SQL> update emp set sal=sal+1000 where empno=7369;

Output:

update emp set sal=sal+1000 where empno=7369

*

ERROR at line 1:

ORA-20070: u can update b/w 10 to 2 only on sat

ORA-06512: at "C##RAJU.T2", line 11

ORA-04088: error during execution of trigger 'C##RAJU.T2'

Program-3:**Define a trigger not to allow the user to update the empno:**

```
Create Or Replace Trigger t3
Before Update of empno
On Emp
Begin
    raise_Application_error(-20050,'u cannot update empno');
End;
/
```

Testing:

```
SQL> update emp set empno=1001 where empno=7369;
```

Output:

```
update emp set empno=1001 where empno=7369
*
```

ERROR at line 1:

```
ORA-20050: u cannot update empno
ORA-06512: at "C##RAJU.T3", line 2
ORA-04088: error during execution of trigger 'C##RAJU.T3'
```

Program-4:**Define a trigger to maintain deleted employee records in 'emp_resign' table:**

```
Create Or Replace Trigger t4
AFTER Delete
On emp
For Each Row
Begin
    insert into emp_resign(empno,ename,job,sal)
    values(:OLD.empno,:OLD.ename,:OLD.job,:OLD.sal);

    dbms_output.put_line('deleted record stored in emp_resign table');

End;
/
```

Testing:

SQL> delete from emp where empno=7876;

Output:

deleted record stored in emp_resign table

Program-5:**Define a trigger to audit emp table:**

Create a table 'emp_audit' as following:

```
create table emp_audit(
    uname varchar2(20),
    op_type varchar2(15),
    op_date_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 UPDATE OR DELETE

ON emp

FOR each row

DECLARE

op varchar2(10);

BEGIN

IF inserting THEN

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 ;
/
```

Testing:

```
SQL> select * from emp_audit;
```

Output:

```
no rows selected
```

```
SQL> insert into emp(empno,ename,sal)
values(5001,'Ravi',6000);
```

Output:

```
rec stored in emp_Audit
```

```
1 row created.
```

```
SQL> update emp set sal=sal+2000 where empno=5001;
```

Output:

```
rec stored in emp_Audit
```

```
1 row updated.
```

```
SQL> delete from emp where empno=5001;
```

Output:

```
rec stored in emp_Audit
```

```
1 row deleted.
```

SQL> select * from emp_audit;

UNAME NEW_EMPNO OLD_SAL	OP_TYPE NEW_ENAME	OP_DATE_TIME NEW_SAL OLD_EMPNO OLD_ENAME
C##RAJU 5001 Ravi	INSERT 6000	25-DEC-21 10.08.01.463000 PM
C##RAJU 5001 Ravi	UPDATE 8000	25-DEC-21 10.08.29.456000 PM
C##RAJU 5001 Ravi	DELETE 8000	25-DEC-21 10.08.39.293000 PM

Program-6:

Define a trigger not to allow the user to decrease the salary:

CREATE OR REPLACE TRIGGER t6

Before UPDATE

ON emp

FOR each row

BEGIN

IF :new.sal < :old.sal THEN

 raise_application_error(-20050,'u cannnot decrease the sal..!');

END IF ;

END ;

/

Testing:

SQL> update emp set sal=sal-1000 where empno=7369;

Output:

update emp set sal=sal-1000 where empno=7369

ERROR at line 1:

ORA-20050: u cannnot decrease the sal..!

ORA-06512: at "C##RAJU.T6", line 3

ORA-04088: error during execution of trigger 'C##RAJU.T6'

Program-7:**[Program to demonstrate schema level trigger]****Define a trigger not to allow c##raju user to drop the table:**

Create Or Replace Trigger t7

Before Drop

On c##oracle11am.schema

BEGIN

IF ora_dict_obj_type = 'TABLE' Then

Raise_Application_Error(-20080,'u cannot drop the db object');

END If;

End;

/

Testing:

SQL> drop table emp;

Output:

drop table emp

*

ERROR at line 1:

ORA-04088: error during execution of trigger 'C##RAJU.T7'

ORA-00604: error occurred at recursive SQL level 1

ORA-20080: u cannot drop the db object

ORA-06512: at line 3

Program-8:**[Program to demonstrate Database Level Trigger]****Define a trigger not to allow many users to drop the table:**

CREATE OR REPLACE TRIGGER t8

Before DROP

ON DATABASE

BEGIN

 IF ora_login_user IN('C##ORACLE11AM','C##ORACLE6PM',
 'C##ORACLE7AM') AND ora_dict_obj_type = 'TABLE' THEN

raise_application_error(-20050,'u cannot drop the table..!');

END IF ;

END ;

/

Testing:

```
SQL> conn c##oracle7am/nareshit
```

Output:

```
Connected.
```

```
SQL> drop table emp;
```

Output:

```
drop table emp  
*
```

```
ERROR at line 1:
```

```
ORA-04088: error during execution of trigger 'C##RAJU.T8'
```

```
ORA-00604: error occurred at recursive SQL level 1
```

```
ORA-20050: u cannot drop the table..!
```

```
ORA-06512: at line 4
```

Program-9:**Define a trigger to perform auditing on DDL commands:**

```
Create a table as following:
```

```
Create table ddl_audit  
(  
    Uname varchar2(10),  
    Date_time timestamp,  
    Action varchar2(10),  
    Obj_name varchar2(10),  
    Obj_Type varchar2(10)  
);
```

```
CREATE OR REPLACE TRIGGER t9  
After DROP OR CREATE OR ALTER OR TRUNCATE  
ON DATABASE  
BEGIN  
    INSERT INTO ddl_audit values(user, systimestamp,  
        ora_sysevent, ora_dict_obj_name, ora_dict_obj_type);  
END ;  
/
```

Testing:

```
SQL> drop table student;
```

Output:

Table dropped.

```
SQL> select * from ddl_audit;
```

UNAME	DATE_TIME	ACTION	OBJ_NAME	OBJ_TYPE
C##RAJU	25-DEC-21 10.42.16.097000 PM			DROP
STUDENT	TABLE			

Program-10:**Program to demonstrate Compound Trigger:**

```
CREATE OR REPLACE TRIGGER t10
```

```
FOR UPDATE ON emp
```

```
compound TRIGGER
```

```
    BEFORE STATEMENT IS
```

```
        BEGIN
```

```
            dbms_output.put_line('before statement');
```

```
        END BEFORE STATEMENT;
```

```
    BEFORE EACH ROW IS
```

```
        BEGIN
```

```
            dbms_output.put_line('before each row');
```

```
        END BEFORE EACH ROW;
```

```
    AFTER EACH ROW IS
```

```
        BEGIN
```

```
            dbms_output.put_line('after each row');
```

```
        END AFTER EACH ROW;
```

```
    AFTER STATEMENT IS
```

```
        BEGIN
```

```
        dbms_output.put_line('after statement');
END AFTER STATEMENT;
END ;
/
```

Testing:

```
SQL> update emp set sal=sal+1000 where job='MANAGER';
```

Output:

```
before statement
before each row
after each row
before each row
after each row
before each row
after each row
after statement
```

3 rows updated.

Dynamic SQL

- We can use DML, DRL, TCL commands directly in PL/SQL program.
- We cannot use DDL commands directly in PL/SQL program.
- If we want to use DDL commands in PL/SQL program, use Dynamic SQL.
- Dynamic SQL means, The SQL command which builds at run time is called "Dynamic SQL".
- When we don't know exact table names & field names when we develop the program then we can use "Dynamic SQL".
- "EXECUTE IMMEDIATE" command is used to execute Dynamic SQL Command.
- We take SQL command as string to use it in "EXECUTE IMMEDIATE".

Programs on Dynamic SQL:

Program-1:

Define a procedure to drop the table:

```
CREATE OR REPLACE PROCEDURE
drop_table(n varchar2)
IS
BEGIN
    EXECUTE immediate 'drop table ' || n;
    dbms_output.put_line(n || ' table dropped..!');
END ;
/
```

Calling from SQL Prompt:

```
SQL> execute drop_table('cust');
cust table dropped..!
```

Program-2:

Define a procedure to drop any kind of database object:

Create Or Replace Procedure

```
drop_object(t varchar2,n varchar2)
IS
Begin
    Execute Immediate 'Drop ' || t || '' || n;
    dbms_output.put_line(t || ' dropped..!');
End ;
/
```

Calling from SQL Prompt:

```
SQL> execute drop_object('view','v1');
view dropped..!
```

Program-3:**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 n in c1
```

```
        Loop
```

```
            execute immediate 'Drop view ' || n.view_name;
```

```
            dbms_output.put_line('All views dropped..!');
```

```
        End Loop;
```

```
    End;
```

```
/
```

Calling from SQL Prompt:

```
SQL> execute drop_All_views;
All views dropped..!
```

Program-4:**Define a procedure to display table wise no of rows:**

```
Create or Replace Procedure
```

```
no_of_rows_in_tables
```

```
IS
```

```
Cursor c1 is Select * from user_tables;
s varchar2(500);
nr number;
Begin
    for n in c1
    Loop
        s:='Select count(*) from ' || n.table_name;
        execute immediate s into nr;
        dbms_output.put_line(rpad(n.table_name,20) || ' ' || nr);
    End Loop;
End ;
/
```

Calling from SQL Prompt:

```
SQL> execute no_of_rows_in_tables
```

```
EMP          14
DEPT         4
....
```

Program-5:**Define a procedure to add a column:**

```
Create Or Replace Procedure
```

```
add_column(tn varchar2,fn varchar2,dt varchar2)
```

```
Is
```

```
Begin
```

```
    execute immediate 'Alter Table ' || tn || ' add ' || fn || '' || dt;
    dbms_output.put_line('column added..!');
```

```
End ;
```

```
/
```

Calling from SQL Prompt:

```
SQL> execute add_column('emp','gender','char');
```

```
column added..!
```