Introduction to Database Systems

DBMS (Database Management System) is a computer software application that interacts with the user, other applications and the database itself to capture and analyze the data.

The DBMS manages three important things:

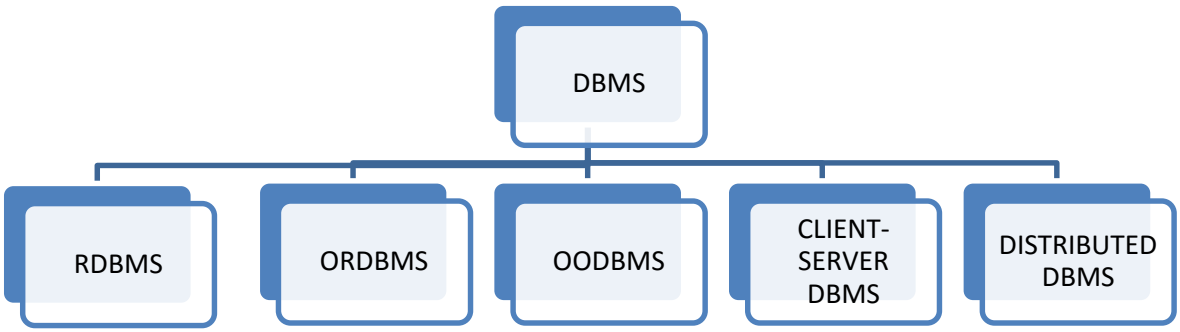
- The data
- The database engine- that allows data to be accessed, locked and modified
- The database schema- which defines the database's logical structure.

A DBMS performs several important functions that guarantee integrity and consistency of data in the database. Most of these functions are transparent to end-users. There are the following important functions and services provided by a DBMS:

- Data Storage Management
- Data Manipulation Management
- Data Definition Services
- Data Dictionary/System Catalog Management
- Database Communication Interfaces
- Authorization **/** Security Management
- Backup and Recovery Management
- Concurrency Control Service
- Transaction Management
- Database Access and Application Programming Interfaces

Shubhkirti Sharma
UE163098
BE CSE G5

Types of DBMS:



1: Types of DBMS

- **RDBMS**

  RDBMS stands for Relational Database Management System. RDBMS data is structured in database tables, fields and records. Each RDBMS table consists of database table rows. Each database table row consists of one or more database table fields.

  RDBMS store the data into collection of tables, which might be related by common fields (database table columns). RDBMS also provide relational operators to manipulate the data stored into the database tables. Most RDBMS use SQL as database query language.

- **ORDBMS**

  An object-relational database (ORD), or object-relational database management system (ORDBMS), is a database management system (DBMS) similar to a relational database, but with an object-oriented database model: objects, classes and inheritance are directly supported in database schemas and in the query language.

- **OODBMS**

  An object-oriented database management system (OODBMS) is a database management system that supports the creation and modelling of data as objects. OODBMS also includes support for classes of objects and the inheritance of class properties, and incorporates methods, subclasses and their objects. Most of the object databases also offer some kind of query

language, permitting objects to be found through a declarative programming approach.

- **CLIENT-SERVER DBMS**

  The Client-Server DBMS Model has emerged as the main paradigm in database computing. The Enhanced Client--Server architecture takes advantage of all the available client resources including their disk managers. However, when updates occur at the server, some of the client data managers may need to not only be notified about them but also obtain portions of the updates as well.

- **DISTRIBUTED DBMS**

  A DDBMS (distributed database management system) is a centralized application that manages a distributed database as if it were all stored on the same computer. The DDBMS synchronizes all the data periodically, and in cases where multiple users must access the same data, ensures that updates and deletes performed on the data at one location will be automatically reflected in the data stored elsewhere.

  In this course we will be focusing on **RDBMS** only.

ORACLE Database

An Oracle database is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. A database server is the key to solving the problems of information management. In general, a server reliably manages a large amount of data in a multiuser environment so that many users can concurrently access the same data. All this is accomplished while delivering high performance. A database server also prevents unauthorized access and provides efficient solutions for failure recovery.

The database has logical structures and physical structures. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting the access to logical storage structures.

Introduction to SQL (Structured Query Language)

Structured Query Language (SQL) is a standard computer language for relational database management and data manipulation. SQL is used to query, insert, update and modify data. Most relational databases support SQL, which is an added benefit for database administrators (DBAs), as they are often required to support databases across several different platforms. SQL offers two main advantages:

- It introduced the concept of accessing many records with one single command
- It eliminates the need to specify how to reach a record (i.e. with or without an index)

Types of SQL Commands

The basic categories of commands used in SQL is to perform various **functions** such as building database objects, manipulating objects, populating database tables with data, updating existing data in tables, deleting data, performing database queries, controlling database access, and overall database administration.

The main categories are:

- **DDL (Data Definition Language)**

  Data Definition Language, DDL, is the part of SQL that allows a database user to create and restructure database objects, such as the creation or the deletion of a table. Some of the most fundamental DDL commands are:
  - CREATE TABLE
  - ALTER TABLE
  - DROP TABLE
  - CREATE INDEX
  - ALTER INDEX
  - DROP INDEX
  - CREATE VIEW
  - DROP VIEW

- **DML (Data Manipulation Language)**

  Data Manipulation Language, DML, is the part of SQL used to manipulate data within objects of a relational database. There are three basic DML commands:
  - INSERT
  - UPDATE
  - DELETE

- **DQL (Data Query Language)**

  Though comprised of only one command, Data Query Language (DQL) is the most concentrated focus of SQL for modern relational database users. The base command is as follows:

  - SELECT

- **DCL (Data Control Language)**

  Data control commands in SQL allow you to control access to data within the database. These DCL commands are normally used to create objects related to user access and also control the distribution of privileges among users. Some data control commands are as follows:

  - ALTER PASSWORD
  - GRANT
  - REVOKE

- **Data Administration Commands**

  Data administration commands allow the user to perform audits and perform analyses on operations within the database. They can also be used to help analyze system performance. Two general data administration commands are as follows:

  - START AUDIT
  - STOP AUDIT

- **Transactional Control Commands**

  In addition to the previously introduced categories of commands, there are commands that allow the user to manage database transactions.

  - COMMIT
  - ROLLBACK
  - SAVEPOINT
  - SET TRANSACTION

DQL - Data Query Language

The commands of SQL that are used to retrieve data from the database are collectively called as DQL. So all Select statements comes under DQL.

SELECT Statement

It is used to view a relation. It shows the extension as well as the schema of the relation. The view can be modified using various conditions to suit the viewer.

Syntax: SELECT <attribute_names> from <table_name>

Example:

```
SQL> select * from employee;

FIRST_NAME LAST_NAME  DEPT           SALARY EMPNO
---------- ---------- ---------- ---------- -----
Shubhkirti Sharma     CEO             95000 EMP01
Varun      Kumar      Technology      40000 EMP02
Anil       Gupta      Research        25000 EMP03
Tanya      Singh      HR              35000 EMP04


SQL> select empno "ECODE", first_name "NAME" from employee;

ECODE NAME
----- ----------
EMP01 Shubhkirti
EMP02 Varun
EMP03 Anil
EMP04 Tanya

SQL> select first_name "NAME" from employee where dept='CEO';

NAME
----------
Shubhkirti

SQL> select empno,name,salary from employee where salary>300000;

EMPNO NAME           SALARY
----- ---------- ----------
E102  David          350000
E101  SHUBHKIRTI    5000000
E103  PHOEBE         350000
E104  Monica         345000
E108  Robin          300500
```

DDL – Data Definition Language

Data Definition Language includes the commands that are used to structure the database or any relation in it. DDL creates, modifies and remove the various database objects such as tables, indexes, and users. It mainly controls the schema of the database.

CREATE Statement

The CREATE command can be used to create new tables as well as new databases.

Syntax: CREATE <database/table> <database/table_name> <attributes>

Example:

```
SQL> create table employee
  2 (
  3 first_name varchar(10),
  4 last_name varchar(10),
  5 dept varchar(10),
  6 salary number(5)
  7 );

Table created.
```

RENAME Statement

The RENAME statement is used to rename a table to a new name. Mostly what it does is that is creates a copy of the table with the new name and then drops the previous table.

SYNTAX: RENAME <table_name> to <new_table_name>

EXAMPLE:

```
SQL> rename employee to employee_data;

Table renamed.
```

ALTER Statement

ALTER statement is used to modify the structure or schema of an existing table or view.

Syntax: ALTER <database/table> <database/table_name>

MODIFY <attribute_name> <attributes_new_properties>

Example:

```
SQL> alter table employee
  2  add empno varchar(5);

Table altered.

SQL> alter table employee drop column id;

Table altered.

SQL> alter table employee rename column joiningdate to doj;

Table altered.
```

DESC Statement

DESC statement is used to describe or view the structure or schema of an existing table or view.

Syntax: DESC <table_name>

Example:

```
SQL> desc employee;

 Name                      Null?    Type
 ------------------------- -------- ----------------
 EMPNO                     NOT NULL VARCHAR2(5)
 NAME                               VARCHAR2(10)
 JOB                                VARCHAR2(5)
 JOININGDATE                        DATE
 SALARY                             NUMBER(10)
 MANID                              VARCHAR2(5)
 DEPTNO                             VARCHAR2(6)
```

DML – Data Manipulation Language

Data Manipulation Language includes the commands that are used to modify the contents of the extension of a table. It includes inserting, deleting, modifying and viewing tuples.

INSERT Statement

The INSERT command is used to insert tuples to a table.

Syntax: INSERT into <table_name> values (<parameter_1>, <parameter_2>…)

Example:

```
SQL> insert into employee values('Shubhkirti', 'Sharma', 'CEO',
90000, 'EMP01');

1 row created.
```

UPDATE Statement

The UPDATE command is used to modify the contents of tuples in a relation. It is usually followed by a ‘where’ condition to update the required tuples only.

Syntax: UPDATE <table_name> SET <new_value_to_be_set> WHERE <condition>

Example:

```
SQL> update employee set salary=95000 where empno='EMP01';

1 row updated.
```

DELETE Statement

It is used to delete tuples in a relation. It is usually followed by a 'where' condition to delete only the required tuples only.

Syntax: DELETE from <table_name> WHERE <condition>

Example:

```
SQL> delete from employee where empno='EMP04';

1 row deleted.
```

DCL – Data Control Language

The commands of SQL that are used to control the access to data stored in the database are collectively called as DCL and examples include Grant and Revoke.

GRANT Statement

The GRANT statement is used to grant a user access and privileges to a database and tables.

Syntax: GRANT <permissions_to_operate> ON <database_name>.<table_name> TO <user_name>

Example:

```
SQL> grant select,update on student98.employee to bello;

Grant succeeded.
```

REVOKE Statement

The REVOKE statement is used to revoke permissions given to a user from a table or database.

Syntax: REVOKE <permissions_to_operate> ON <database_name>.<table_name> FROM <user_name>

Example:

```
SQL> revoke select on student98.employee from bello;

Revoke succeeded.
```

TCL – Transaction Control Language

The commands of SQL that are used to control the transactions made against the database are collectively called as TCL and examples include Commit, Rollback and Savepoint.

COMMIT Statement

COMMIT statement is used to save changes permanently to the hard drive. We cannot undo the changes after executing commit statement.

Syntax: COMMIT;

Example:

```
SQL> commit;

Commit complete.
```

SAVEPOINT Statement

SAVEPOINT statement is used to create a Savepoint during a transaction. We can rollback to a savepoint using the rollback command.

Syntax: SAVEPOINT <Savepoint_name>

Example:

```
SQL> savepoint my_savepoint;

Savepoint created.
```

ROLLBACK Statement

ROLLBACK statement is used to undo changes to a savepoint or to the last commit. Rollback executes at physical level.

Syntax: ROLLBACK
ROLLBACK TO <savepoint_name>

Example:

```
SQL> rollback;

Rollback complete.

SQL> rollback to my_savepoint;

Rollback complete.
```

DROP Statement

The DROP command is used to delete or databases along with data.

Syntax: DROP <database/table> <database/table_name>

Example:

```
SQL> drop table employee_temp;

Table dropped.
```

TRUNCATE Statement

The TRUNCATE statement deletes the extension part of a relation. It deletes the all the tuples leaving the schema.

SYNTAX: TRUNCATE table <table_name>

EXAMPLE:

```
SQL> truncate table employee;

Table truncated.
```

DATA CONSTRAINTS

Constraints are used to prevent users from entering fault values or values not supported by the relation.

I/O CONSTRAINTS

1. PRIMARY KEY

A primary key is used to uniquely define the tuple in a relation. This reduces data redundancy in the database. There are many ways to add this constraint. One of them is shown here where you can alter the table to set a column as a primary key. One can define a primary key at the time of creating the table as well.

EXAMPLE:

```
SQL> alter table employee add PRIMARY KEY (empno);

Table altered.
```

2. FOREIGN KEY

A foreign key is an attribute that depends on the primary key of another relation. Here we have shown how one can define a foreign key while creating a table also telling which attribute it refers to.

EXAMPLE:

```
SQL> create table sales_department
  2  (
  3  sales_code varchar(5) REFERENCES employee(empno)
  4  );

Table created.
```

3. NOT NULL

The NOT NULL constraint makes sure that the user does not enter a NULL value for the attribute for which the statement is called. This can be done either by altering the table or at the time of creating the table.

EXAMPLE:

```
SQL> alter table employee modify empno NOT NULL;

Table altered.
```

BUSINESS CONSTRAINTS

Business contraints are implemented using CHECK statements and allow users to put custom check on values entered such as greater then or less than or similar to.

Example:

1. The following constraint only allows users to enter the employee number that starts with E and then followed by anything:

```
SQL> alter table employee add constraint EMP_CHECK check(empno like
'E%');

Table altered.
```

2. The following constraint only allows users to enter the salary of an employee greater than zero.

```
SQL> alter table employee add constraint EMP_SAL_CHECK
check(salary>0);

Table altered.
```

ASSIGNMENT – I

Q1. Create the tables described below:

1. Table Name: CLIENT_MASTER (Used to store client information)

Column Name	Data Type	Size
CLIENTNO	Varchar2	6
NAME	Varchar2	20
ADDRESS1	Varchar2	30
ADDRESS2	Varchar2	30
CITY	Varchar2	15
PINCODE	Number	8
STATE	Varchar2	15
BALDUE	Number	10,2

```
SQL> create table client_master(
  2  clientno varchar2(6),
  3  name varchar2(20),
  4  address1 varchar2(30),
  5  address2 varchar2(30),
  6  city varchar2(15),
  7  pincode number(8),
  8  state varchar(15),
  9  baldue number(10,2)
 10  );

Table created.
```

2. Table Name: PRODUCT_MASTER (Used to store product information)

Column Name	Data Type	Size
PRODUCTNO	Varchar2	6
DESCRIPTION	Varchar2	15
PROFITPERCENT	Number	4,2
UNITMEASURE	Varchar2	10
QTYONHAND	Number	8
REORDERLVL	Number	8
SELLPRICE	Number	8,2
COSTPRICE	Number	8,2

```
SQL> create table product_master(
  2  productno varchar2(6),
  3  description varchar2(15),
  4  profitpercent number(4,2),
  5  unitmeasure varchar(10),
  6  qtyonhand number(8),
```

```
  7  reorderlvl number(8),
  8  sellprice number(8,2),
  9  costprice number(8,2)
 10  );

Table created.
```

3. Table Name: SALESMAN_MASTER (Used to store salesman information working for the company)

Column Name	Data Type	Size
SALESMANNO	Varchar2	6
SALESMANNAME	Varchar2	20
ADDRESS1	Varchar2	30
ADDRESS2	Varchar2	30
CITY	Varchar2	20
PINCODE	Number	8
STATE	Varchar2	20
SALAMT	Number	8,2
TGTTOGET	Number	6,2
YTDSALES	Number	6,2
REMARKS	Varchar2	60

```
SQL> create table salesman_master(
  2  salesmanno varchar2(6),
  3  salesmanname varchar2(20),
  4  address1 varchar(30),
  5  address2 varchar(30),
  6  city varchar(20),
  7  pincode number(8),
  8  state varchar(20),
  9  salamt number(8,2),
 10  tgttoget number(6,2),
 11  ytdsales number(6,2),
 12  remarks varchar(30)
 13  );

Table created.
```

Q2. Insert the following data into their respective tables:

1. Data for **CLIENT_MASTER**

ClientNo	Name	City	Pincode	State	BalDue
C00001	Ivan Bayross	Mumbai	400054	Maharashtra	15000
C00002	Mamta Mazumdar	Madras	780001	Tamil Nadu	0
C00003	Chhaya Bankar	Mumbai	400057	Maharashtra	5000
C00004	Ashwini Joshi	Bangalore	560001	Karnataka	0
C00005	Hansel Colaco	Mumbai	400060	Maharashtra	2000
C00006	Deepak Sharma	Mangalore	560050	Karnataka	0

QUERIES

```
SQL> insert into client_master values('C0001','Ivan Bayross','123 New Street','321
Old Trafford','Mumbai',400054,'Maharashtra',15000);

1 row created.

SQL> insert into client_master values('C0002','Mamta Mazumdar','421 New
Colony','992 Old District','Madras',780001,'Tamil Nadu',0);

1 row created.

SQL> insert into client_master values('C0003','Chhaya Bankar','41 New Friends
District','99 First Floor','Mumbai',400057,'Maharashtra',5000);

1 row created.

SQL> insert into client_master values('C0004','Ashwini Joshi','C01 Defence
Colony','99 Civil Lines','Bangalore',560001,'Karnataka',0);

1 row created.

SQL> insert into client_master values('C0005','Hansel Colaco','A/3 Lajpat
Nagar','B3 Mahatama Colony','Mumbai',400060,'Maharashtra',2000);

1 row created.

SQL> insert into client_master values('C0006','Deepak Sharma','66G Jasola
Vihar','','Mangalore',560050,'Karnataka',0);

1 row created.
```

2. Data for **PRODUCT_MASTER**

ProductNo	Description	Profit Percent	Unit Measure	QtyOn Hand	Reorder Lvl	Sell Price	Cost Price
P00001	T-Shirts	5	Piece	200	50	350	250
P0345	Shirts	6	Piece	150	50	500	350
P06734	Cotton Jeans	5	Piece	100	20	600	450
P07865	Jeans	5	Piece	100	20	750	500
P07868	Trousers	2	Piece	150	50	850	550
P07885	Pull Overs	2.5	Piece	80	30	700	450
P07965	Denim Shirts	4	Piece	100	40	350	250
P07975	Lycra Tops	5	Piece	70	30	300	175
P08865	Skirts	5	Piece	75	30	450	300

```
SQL> insert into product_master values('P00001','T-
Shirts',5,'Piece',200,50,350,250);

1 row created.

SQL> insert into product_master values('P0345','Shirts',6,'Piece',150,50,500,350);

1 row created.

SQL> insert into product_master values('P06734','Cotton
Jeans',5,'Piece',100,20,600,450);

1 row created.

SQL> insert into product_master values('P07865','Jeans',5,'Piece',100,20,750,500);

1 row created.

SQL> insert into product_master
values('P07868','Trousers',2,'Piece',150,50,850,550);

1 row created.

SQL> insert into product_master values('P07885','Pull
Overs',2.5,'Piece',80,30,700,450);

1 row created.

SQL> insert into product_master values('P07965','Denim
Shirts',4,'Piece',100,40,350,250);

1 row created.

SQL> insert into product_master values('P07975','Lycra
Tops',5,'Piece',70,30,300,175);

1 row created.


SQL> insert into product_master values('P08865','Skirts',5,'Piece',75,30,450,300);

1 row created.
```

2. Data for **SALESMAN_MASTER**

SALESMANNO	SALESMANNAME	ADDRESS1	ADDRESS2	CITY	PINCODE	STATE
S00001	Aman	A/14	Worli	Pune	400002	Maharashtra
S00002	Omkar	65	Nariman	Mumbai	400001	Maharashtra
S00003	Raj	P-7	Bandra	Mumbai	400032	Maharashtra
S00004	Ashish	A/5	Juhu	Mumbai	400044	Maharashtra

SALARY	TGTTOGET	YTDSALES	REMARKS
3000	100	50	Good
3000	200	100	Good
3000	200	100	Good
3500	200	150	Good

```
SQL> insert into salesman_master
values('S00001','Aman','A/14','Worli','Mumbai',400002,'Maharashtra',3000,100,50,
'Good');

1 row created.

SQL> insert into salesman_master
values('S00002','Omkar','65','Nariman','Mumbai',400001,'Maharashtra',3000,200,100,
'Good');

1 row created.

SQL> insert into salesman_master
values('S00003','Raj','P-7','Bandra','Mumbai',400032,'Maharashtra',3000,200,100,
'Good');

1 row created.

SQL> insert into salesman_master
values('S00004','Ashish','A/5','Juhu','Mumbai',400044,'Maharashtra',3500,200,150,
'Good');

1 row created.
```

Q3. Exercise on retrieving records from a table

a. Find out names of all the clients.

```
SQL> select name from client_master;

NAME
--------------------
Mamta Mazumdar
Chhaya Bankar
Ashwini Joshi
Hansel Colaco
Deepak Sharma
Ivan Bayross

6 rows selected.
```

b. Retrieve the entire contents of the Client_Master table.

```
SQL> select * from client_master;

CLIENT NAME                 ADDRESS1                       ADDRESS2
------ -------------------- ------------------------------ ------------------------
C0002  Mamta Mazumdar       421 New Colony                 992 Old District
C0003  Chhaya Bankar        41 New Friends District        99 First Floor
C0004  Ashwini Joshi        C01 Defence Colony             99 Civil Lines
C0005  Hansel Colaco        A/3 Lajpat Nagar               B3 Mahatama Colony
C0006  Deepak Sharma        66G Jasola Vihar
C0001  Ivan Bayross         123 New Street                 321 Old Trafford

CITY               PINCODE STATE               BALDUE
--------------- ---------- --------------- ----------
Madras              780001 Tamil Nadu               0
Mumbai              400057 Maharashtra           5000
Bangalore           560001 Karnataka                0
Mumbai              400060 Maharashtra           2000
Mangalore           560050 Karnataka                0
Mumbai              400054 Maharashtra          15000

6 rows selected.
```

c. Retrieve the list of names, city and state of all the clients.

```
SQL> select name,city,state from client_master;

NAME                 CITY            STATE
-------------------- --------------- ---------------
Mamta Mazumdar       Madras          Tamil Nadu
Chhaya Bankar        Mumbai          Maharashtra
Ashwini Joshi        Bangalore       Karnataka
Hansel Colaco        Mumbai          Maharashtra
Deepak Sharma        Mangalore       Karnataka
Ivan Bayross         Mumbai          Maharashtra

6 rows selected.
```

d. List the various products available from the Product_Master table.

```
SQL> select description from product_master;

DESCRIPTION
---------------
T-Shirts
Shirts
Cotton Jeans
Jeans
Trousers
Pull Overs
Denim Shirts
Lycra Tops
Skirts

9 rows selected.
```

e. List all the clients who are located in Mumbai

```
SQL> select name from client_master where city='Mumbai';

NAME
--------------------
Chhaya Bankar
Hansel Colaco
Ivan Bayross
```

f. Find the names of all the salesmen who have salary equal to Rs.3000

```
SQL> select salesmanname from salesman_master where salamt=3000;

SALESMANNAME
--------------------
Aman
Omkar
Raj
```

Q4. Exercise on updating records in a table.

a. Change the city of client no 'C0005' to 'Bangalore'.

```
SQL> update client_master set city='Bangalore' where clientno='C0005';

1 row updated.
```

b. Change the BalDue of ClientNo 'C0001' to 1000.

```
SQL> update client_master set baldue=1000 where clientno='C0001';

1 row updated.
```

c. Change the cost price of 'Trousers' to Rs.950

```
SQL> update product_master set costprice='950' where description='Trousers';
```

```
1 row updated.
```

c. Change the city of salesman 'Raj' to 'Pune'.

```
SQL> update salesman_master set city='Pune' where salesmanname='Raj';

1 row updated.
```

Q5. Exercise on deleting records in a table.

a. Delete all salesmen from the Salesman_Master whose salaries are equal to Rs.3500

```
SQL> delete from salesman_master where salamt=3500;

1 row deleted.
```

b. Delete all products from Product_Master where QtyOnHand is equal to 100.

```
SQL> delete from product_master where qtyonhand=100;

3 rows deleted.
```

c. Delete from Client_Master where column state holds the value 'Tamil Nadu'.

```
SQL> delete from client_master where state='Tamil Nadu';

1 row deleted.
```

Q5. Exercise on altering the table structure.

a. Add a column called 'Telephone' of 'number' type and size '10' to the Client_Master table.

```
SQL> alter table client_master add telephone number(10);

Table altered.
```

b. Change the size of 'SellPrice' in Product_Master to 10,2.

```
SQL> alter table product_master modify sellprice number(10,2);

Table altered.
```

Q7. Exercise on deleting a table structure along with the data.

a. Destroy the table Client_Master along with its data.

```
SQL> drop table client_master;

Table dropped.
```

Q8. Exercise on renaming a table.

a. Change the name of 'Salesman_Master' to 'sman_mast'.

```
SQL> rename salesman_master to sman_mast;

Table renamed.
```

Q9. Implement the Intergrity and Business Constraints based on the tables above.

a. PRIMARY KEYS

```
SQL> alter table client_master add primary key(clientno);

Table altered.

SQL> alter table product_master add primary key(productno);

Table altered.

SQL> alter table salesman_master add primary key(salesmanno);

Table altered.

SQL> alter table sales_order add primary key(orderno);

Table altered.
```

b. FOREIGN KEYS

```
SQL> alter table sales_order add constraint CLIENT_SALES_REL foreign key(clientno)
references client_master(clientno);

Table altered.

SQL> alter table sales_order add constraint SALESMAN_SALES_REL foreign
key(salesmanno) references salesman_master(salesmanno);

Table altered.

SQL> alter table sales_order_details add constraint SALES_ORDER_REL foreign
key(orderno) references sales_order(orderno);

Table altered.

SQL> alter table sales_order_details add constraint SALES_ORDER_PRODUCT_REL
foreign key(productno) references product_master(productno);

Table altered.
```

c. NOT NULL CONSTRAINT

```
SQL> alter table client_master modify clientno varchar2(6) NOT NULL;

Table altered.
```

```
SQL> alter table product_master modify description varchar2(15) NOT NULL;

Table altered.

SQL> alter table product_master modify profitpercent number(4,2) NOT NULL;

Table altered.

SQL> alter table product_master modify unitmeasure varchar2(10) NOT NULL;

Table altered.

SQL> alter table product_master modify qtyonhand number(10,2) NOT NULL;

Table altered.

SQL> alter table product_master modify reorderlvl number(8) NOT NULL;

Table altered.

SQL> alter table product_master modify sellprice number(10,2) NOT NULL;

Table altered.

SQL> alter table product_master modify costprice number(10,2) NOT NULL;

Table altered.

SQL> alter table sales_order modify orderdate date NOT NULL;

Table altered.
```

d. DEFAULT CONSTRAINT

```
SQL> alter table sales_order modify delytype default 'F';

Table altered.
```

e. BUSINESS CONSTRAINTS

```
SQL> alter table client_master add constraint CHK_CLIENT_FORMAT check (clientno
like 'C%');

Table altered.

SQL> alter table product_master add constraint CHK_PRODUCT_FORMAT check(productno
like 'P%');

Table altered.

SQL> alter table salesman_master add constraint CHK_SALESMAN_FORMAT
check(salesmanno like 'S%');

Table altered.

SQL> alter table sales_order add constraint CHK_ORDER_FORMAT check(orderno like
'O%');

Table altered.
```

```
SQL> alter table sales_order add constraint CHK_DELYTYPE_FORMAT check(delytype in
('P','F'));

Table altered.

SQL> alter table sales_order add constraint CHK_DELYDATE_STATUS check(delydate >
orderdate);

Table altered.

SQL> alter table sales_order add constraint CHK_ORDERSTATUS_VALUES
check(orderstatus in ('In Process','Fulfilled','Back Order','Cancelled'));

Table altered.
```

BUILT IN FUNCTIONS

Oracle DBMS provides a number of built in functions that can operate on data values to provide users more accessibility and easy management of data.

Many of the functions can be applied and operated on the DUAL table also, which is a special one row and one column table present in data dictionary. The data type of the cell is varchar2(1). Every operation that has a single row output suvh as trimming, padding can be performed on the dual table.

Some of them are discussed here:

1. IS NULL

The IS NULL command shows the tuples where the mentioned attribute entered is NULL.

EXAMPLE:

```
SQL> select * from employee where salary IS NULL;

FIRST_NAME LAST_NAME  DEPT           SALARY EMPNO
---------- ---------- ---------- ---------- -----
Drake      Trooper    Intern                EMP09
Carl       Johnson    Intern                EMP10
```

2. NULL VALUES SUBSTITUTION (NVL STATEMENT)

The NVL command replaces a NULL value wherever with the provided value.

SYNTAX: SELECT NVL (<attribute_name>,<value_to_be_replaced_with>) FROM <table_name>

EXAMPLE:

```
SQL> select NVL(salary, 5000) from employee;

NVL(SALARY,5000)
----------------
           95000
           40000
           25000
           35000
           30000
           28500
           28500
           25000
            5000
            5000

10 rows selected.
```

3. CHARACTER STRING FUNCTIONS

Oracle Application Express provides a number of inbuilt functions to manipulate strings so that the DBA can provide the user or viewer in any way they want to see.

INITCAP Function
Example:

```
SQL> select INITCAP('my name is shubhkirti sharma') "Capital Words" from
DUAL;

Capital Words
---------------------------
My Name Is Shubhkirti Sharma
```

INSTR Function
Example:

```
SQL> select INSTR('I am an Assassin', 'ss', 2, 2) "In String Value" from
DUAL;

In String Value
---------------
             13
```

SUBSTR Function
Example:

```
SQL> select SUBSTR('My name is Shubhkirti Sharma', 4, 13) "SUB STRING" from
DUAL;

SUB STRING
-------------
name is Shubh
```

UPPER & LOWER Functions
Example:

```
SQL> select UPPER(first_name) "First Name", LOWER(last_name) "Last Name"
from employee;

First Name Last Name
---------- ----------
SHUBHKIRTI sharma
VARUN      kumar
ANIL       gupta
TANYA      singh
KARAN      singh
BISWA      chaudhary
TINA       oberoi
SHIVAM     singhania
DRAKE      trooper
CARL       johnson

10 rows selected.
```

LENGTH Function

Example:

```
SQL> select LENGTH(first_name) from employee;

LENGTH(FIRST_NAME)
------------------
                10
                 5
                 4
                 5
                 5
                 5
                 4
                 6
                 5
                 4

10 rows selected.
```

||‘ ’|| Function

Example:

```
SQL> select last_name || ', ' || first_name "Full Name" from employee;

Full Name
----------------------
Sharma, Shubhkirti
Kumar, Varun
Gupta, Anil
Singh, Tanya
Singh, Karan
Chaudhary, Biswa
Oberoi, Tina
Singhania, Shivam
Trooper, Drake
Johnson, Carl

10 rows selected.
```

RTRIM & LTRIM Function

Example:

```
SQL> select join_month "JOINING MONTH", LENGTH(join_month) "UNTRIMMED",
LENGTH(RTRIM(join_month)) "TRIMMED" from employee;

JOINING MONTH    UNTRIMMED    TRIMMED
--------------- ---------- ----------
JANUARY                 15          7
FEBRUARY                15          8
DECEMBER                15          8
MAY                     15          3
JULY                    15          4
APRIL                   15          5
JANUARY                 15          7
JANUARY                 15          7
SEPTEMBER               15          9
NOVEMBER                15          8

10 rows selected.
```

```
SQL> select LENGTH('     Hello from left side') "UNTRIMMED", LENGTH(LTRIM('
Hello from left side')) "TRIMMED" from DUAL;

 UNTRIMMED    TRIMMED
---------- ----------
        25         20
```

RPAD & LPAD Function
Example:

```
SQL> select RPAD('Hello ', 12 ,'I am added to right') "RPAD" from DUAL;

RPAD
------------
Hello I am a

SQL> select LPAD('Hello', 10 ,'Wow') "LPAD" from DUAL;

LPAD
----------
WowWoHello
```

REVERSE Function
Example:

```
SQL> select REVERSE('Bello from SQL') "REVERSE" from DUAL;

REVERSE
--------------
LQS morf olleB
```

4. NUMBER FUNCTIONS

Oracle Application Express also provides a number of inbuilt functions to manipulate numbers by allowing us to operate certain functions on it. Some of them are discussed below:

ABSOLUTE Function
Example:

```
SQL> select abs(-5) "ABSOLUTE VALUE" from DUAL;

ABSOLUTE VALUE
--------------
             5
```

CEILING Function
Example:

```
SQL> select ceil(0.75) from DUAL;

CEIL(0.75)
----------
         1
```

FLOOR Function
Example:

```
SQL> select floor(0.75) from DUAL;

FLOOR(0.75)
-----------
          0
```

EXPONENTIAL Function
Example:

```
SQL> select exp(5) from DUAL;

    EXP(5)
----------
148.413159
```

SQUARE ROOT Function
Example:

```
SQL> select sqrt(10) from DUAL;

  SQRT(10)
----------
3.16227766
```

MODULUS Function
Example:

```
SQL> select mod(19,5) from DUAL;

 MOD(19,5)
----------
         4
```

POWER Function
Example:

```
SQL> select power(5,3) from DUAL;

POWER(5,3)
----------
       125
```

ROUND Function
Example:

```
SQL> select round(15.274758, 3) from DUAL;

ROUND(15.274758,3)
------------------
            15.275
```

TRUNC Function
Example:

```
SQL> select trunc(15.274758, 3) from DUAL;

TRUNC(15.274758,3)
------------------
            15.274
```

GREATEST Function
Example:

```
SQL> select greatest(23,34,45,54,12,9) from DUAL;

GREATEST(23,34,45,54,12,9)
--------------------------
                        54
```

LEAST Function
Example:

```
SQL> select least(23,34,45,54,12,9) from DUAL;

LEAST(23,34,45,54,12,9)
-----------------------
                      9
```

5. AGGREGATE FUNCTIONS

These functions allow users to perform aggregate functions on the data. Some of them are discussed below:

MIN Function
Example:

```
SQL> select MIN(salary) from employee;

MIN(SALARY)
-----------
      25000
```

MAX Function
Example:

```
SQL> select MAX(salary) from employee;

MAX(SALARY)
-----------
      95000
```

AVG Function
Example:

```
SQL> select AVG(salary) from employee;

AVG(SALARY)
```

```
-----------
      38375
```

SUM Function
Example:

```
SQL> select SUM(salary) from employee;

SUM(SALARY)
-----------
     307000
```

COUNT Function
Example:

```
SQL> select COUNT(*) from employee;

  COUNT(*)
----------
        10
```

6. LOGICAL OPERATORS

Logical operators are mostly used with the where condition to test something along with another. These are AND and OR.

EXAMPLE:

```
SQL> select first_name, last_name from employee where
  2  salary > 30000
  3  AND
  4  dept='Research'
  5  OR
  6  dept='Technology';

FIRST_NAME LAST_NAME
---------- ----------
Varun      Kumar
Karan      Singh
Biswa      Chaudhary
```

7. ARITHMETIC OPERATORS

The various arithmetic operators (+, -, /, *) can also be used to view the relation in a desired format. These do not alter the original relation but are only for visualization.

EXAMPLE:

+ OPERATOR

```
SQL> select salary+2500 "INCR SALARY" from employee;
```

```
INCR SALARY
-----------
      97500
      42500
      27500
      37500
```

- OPERATOR

```
SQL> select salary-12000 "DECR SALARY" from employee;

DECR SALARY
-----------
      83000
      28000
      13000
      23000
```

ASSIGNMENT – II

Create table employee and department as shown with the given constraints. Put random correct values/data in the tables to operate on.

Table Employee

```
SQL> CREATE TABLE employee(empno VARCHAR2(4) PRIMARY KEY CHECK(empno    LIKE
'E%'),ename VARCHAR2(20),ejob VARCHAR2(20),joiningdate DATE,esal  NUMBER(10) CHECK
(esal>0),manid VARCHAR2(4),deptno VARCHAR2(4));

Table created.
```

Table Department

```
SQL> CREATE TABLE department (deptno VARCHAR2(4) PRIMARY KEY CHECK(deptno LIKE
'D%'),dname VARCHAR2(20),dloc VARCHAR2(10));

Table created.
```

Q1. Give total salary issued by each department.

```
SQL> select deptno, sum(salary) from employee group by deptno;

DEPTNO SUM(SALARY)
------ -----------
D110        255000
D115        300500
D102        525500
D100       5000000
D101       1045000
```

Q2. Give average salary of each job.

```
SQL> select job, avg(salary) from employee group by job;

JOB   AVG(SALARY)
----- -----------
AM     348333.333
HRM        300500
SM     260166.667
CEO       5000000
```

Q3. List total salary in each job in each department.

```
SQL> select deptno,job,sum(salary) from employee group by deptno,job;

DEPTNO JOB   SUM(SALARY)
------ ----- -----------
D100   CEO       5000000
D102   SM         525500
D101   AM        1045000
D110   SM         255000
D115   HRM        300500
```

Q4. List total, max, min, avg, salary of department no 'D006' job wise.

```
SQL> SELECT job,sum(salary),max(salary),min(salary),avg(salary) from employee
where (deptno='D102') group by job;

JOB   SUM(SALARY) MAX(SALARY) MIN(SALARY) AVG(SALARY)
----- ----------- ----------- ----------- -----------
SM         525500      275500      250000      262750
```

Q5. List jobs and average salary of jobs having average salary greater than 1,000,000.

```
SQL> select job, avg(salary) from employee having (avg(salary)>1000000) group by
job;

JOB   AVG(SALARY)
----- -----------
CEO       5000000
```

NESTED QUERIES

Q6. List name and salary of employee whose salary is greater than minimum salary of department 'Services'.

```
SQL> select name,salary from employee where salary>(select min(salary) from
employee natural join department where deptname='Services');

NAME           SALARY
---------- ----------
David          350000
Chandler       275500
Shubh         5000000
Alex           350000
Monica         345000
Robin          300500

6 rows selected.
```

Q7. List name and salary of employee whose salary is greater than maximum salary of department 'Tech'.

```
SQL> select name,salary from employee where salary>(select max(salary) from
employee natural join department where deptname='Tech');

NAME           SALARY
---------- ----------
Shubh         5000000
```

Q8. List the details of department where Manager ID is 'E104'.

```
SQL> select * from department where deptno=(select deptno from employee where
manid='E104');
```

```
DEPTN DEPTNAME             DEPTLOC
----- -------------------- ------------
D110  Services             Moscow
```

Q9. List the employees belonging to the department 'Shubh'.

```
SQL> select name from employee where deptno=(select deptno from department where
deptname='Shubh');

NAME
----------
Ross
Chandler
```

Q10. List the name of the employees who do the same job as of employee with employee no 'E102'.

```
SQL> select name from employee where job=(select job from employee where
empno='E102');

NAME
----------
David
Alex
Monica
```

Q11. List the name of the employees who do the same job as of employee with employee no 'E102' and whose salary is greater than 'E110'.

```
SQL> select name from employee where job=(select job from employee where
empno='E102') and salary>(select salary from employee where empno='E104');

NAME
----------
David
Alex
```

Q12. List the name and salary of employee whose salary is greater than the salary of the department 'Tech'.

```
SQL> select name,salary from employee where salary>(select sum(salary) from
employee natural join department where deptname='Management');

NAME           SALARY
---------- ----------
David          350000
Shubh         5000000
Alex           350000
Monica         345000
```

Q13.List the details of department where Manager ID is 'E101'.

```
SQL> select * from department where deptno=(select deptno from employee where
manid='E104');

DEPTN DEPTNAME             DEPTLOC
----- -------------------- ------------
D110  Services             Moscow
```

Q14. List the employees who do not manage any employee.

```
SQL> select name from employee where empno not in (select manid from employee);

NAME
----------
Robin
Chandler
Barney
Ross
```

Q15. List the employees who manage atleast one employee.

```
SQL> select name from employee where empno in (select manid from employee);

NAME
----------
Shubh
David
Monica
Alex
```

Q16. List the employees whose salary is less than minimum salary of the department 'HR' and they must be of different department.

```
SQL> select name from employee natural join department where salary<(select
min(salary) from employee natural join department where deptname='HR') and
deptname not in 'HR';

NAME
----------
Chandler
Ross
Barney
```

Q17. List the top 5 and bottom 5 employees with highest salaries and lowest salaries respectively.

```
SQL> select * from (select * from employee order by salary desc) where rownum<=5;

EMPNO NAME       JOB   JOININGDA     SALARY MANID DEPTNO
----- ---------- ----- --------- ---------- ----- ------
E101  SHUBHKIRTI CEO   09-DEC-13    5000000 E100  D100
```

```
E102  David      AM    09-DEC-13     350000 E101  D101
E103  PHOEBE     AM    15-DEC-13     350000 E101  D101
E104  Monica     AM    15-DEC-13     345000 E101  D101
E108  Robin      HRM   20-JUN-14     300500 E103  D115

SQL> select * from (select * from employee order by salary asc) where rownum<=5;

EMPNO NAME       JOB   JOININGDA     SALARY MANID DEPTNO
----- ---------- ----- --------- ---------- ----- ------
E200  GARY       Int   02-FEB-15      40000 E100  D100
E105  Ross       SM    30-DEC-13     250000 E102  D102
E107  Barney     SM    20-JUN-14     255000 E104  D110
E106  Chandler   SM    20-JUN-14     275500 E102  D102
E108  Robin      HRM   20-JUN-14     300500 E103  D115
```

GROUP BY STATEMENT

This statement groups the attribute specified based on similar tuple values. Some examples are shown below:

Example:

```
SQL> select dept from employee group by dept;

DEPT
----------
Research
Technology
HR
Intern
CEO

SQL> select count(*) "NO OF EMPLOYEES", dept from employee group by dept;

NO OF EMPLOYEES DEPT
--------------- ----------
              2 Research
              3 Technology
              2 HR
              2 Intern
              1 CEO
```

ORDER BY STATEMENT

The order by statement is used to order a particular attribute in either ascending or descending order while displaying. Some examples are shown below:

Example:

```
SQL> select empno "EMPLOYEE NUMBER", first_name from employee order by
empno DESC;

EMPLO FIRST_NAME
----- ----------
EMP07 Tina
EMP06 Biswa
EMP05 Karan
EMP04 Tanya
EMP03 Anil
EMP02 Varun
EMP01 Shubhkirti

7 rows selected.
```

SUBQUERY

A subquery is a SQL query within a query. They are nested queries that provide data to the enclosing query. They can either return individual values or a list of records.

JOINING OF TABLES

Data can be viewed by joining various tables based on certain common columns as shown:

1. CROSS JOIN

Displays all combination between the two tables based on common column.

```
SQL> select * from student cross join address_student;

ROLLN NAME       ROLLN ADDRESS    CITY       STATE
----- ---------- ----- ---------- ---------- ----------
S001  Shubhkirti S001  123        New Delhi  Delhi
S002  Tanveer    S001  123        New Delhi  Delhi
S003  Pranav     S001  123        New Delhi  Delhi
S005  Shivank    S001  123        New Delhi  Delhi
S004  Prateek    S001  123        New Delhi  Delhi
S001  Shubhkirti S002  A/5        Ambala     Punjab
S002  Tanveer    S002  A/5        Ambala     Punjab
S003  Pranav     S002  A/5        Ambala     Punjab
S005  Shivank    S002  A/5        Ambala     Punjab
S004  Prateek    S002  A/5        Ambala     Punjab
S001  Shubhkirti S003  D-15       Patna      Bihar
S002  Tanveer    S003  D-15       Patna      Bihar
S003  Pranav     S003  D-15       Patna      Bihar
S005  Shivank    S003  D-15       Patna      Bihar
S004  Prateek    S003  D-15       Patna      Bihar
S001  Shubhkirti S004  M-10       Chandigarh Chandigarh
S002  Tanveer    S004  M-10       Chandigarh Chandigarh
S003  Pranav     S004  M-10       Chandigarh Chandigarh
S005  Shivank    S004  M-10       Chandigarh Chandigarh
S004  Prateek    S004  M-10       Chandigarh Chandigarh
S001  Shubhkirti S005  K/10       Ambala     Punjab
S002  Tanveer    S005  K/10       Ambala     Punjab
S003  Pranav     S005  K/10       Ambala     Punjab
S005  Shivank    S005  K/10       Ambala     Punjab
S004  Prateek    S005  K/10       Ambala     Punjab

25 rows selected.
```

2. NATURAL JOIN

Displays all non-repeated columns based on common columns with matching rows.

```
SQL> select * from student natural join address_student;

ROLLN NAME       ADDRESS    CITY       STATE
----- ---------- ---------- ---------- ----------
S001  Shubhkirti 123        New Delhi  Delhi
S002  Tanveer    A/5        Ambala     Punjab
S003  Pranav     D-15       Patna      Bihar
S004  Prateek    M-10       Chandigarh Chandigarh
S005  Shivank    K/10       Ambala     Punjab
```

```
SQL> select * from student left natural join address_student;

ROLLN NAME       ADDRESS    CITY       STATE
----- ---------- ---------- ---------- ----------
S001  Shubhkirti 123        New Delhi  Delhi
S002  Tanveer    A/5        Ambala     Punjab
S003  Pranav     D-15       Patna      Bihar
S004  Prateek    M-10       Chandigarh Chandigarh
S005  Shivank    K/10       Ambala     Punjab

SQL> select * from student right natural join address_student;

ROLLN NAME       ADDRESS    CITY       STATE
----- ---------- ---------- ---------- ----------
S001  Shubhkirti 123        New Delhi  Delhi
S002  Tanveer    A/5        Ambala     Punjab
S003  Pranav     D-15       Patna      Bihar
S004  Prateek    M-10       Chandigarh Chandigarh
S005  Shivank    K/10       Ambala     Punjab
```

3. OUTER JOIN

Display repeated columns also, but user needs to specify the columns based on which the join is to be performed.

```
SQL> select * from student full outer join address_student on
student.rollno=address_student.rollno;

ROLLN NAME       ROLLN ADDRESS    CITY       STATE
----- ---------- ----- ---------- ---------- ----------
S001  Shubhkirti S001  123        New Delhi  Delhi
S002  Tanveer    S002  A/5        Ambala     Punjab
S003  Pranav     S003  D-15       Patna      Bihar
S004  Prateek    S004  M-10       Chandigarh Chandigarh
S005  Shivank    S005  K/10       Ambala     Punjab
```

VIEWS

A VIEW is a virtual table, through which a selective portion of the data from one or more tables can be seen. Views do not contain data of their own. They are used to restrict access to the database or to hide data complexity.

Creating a view

A view can be created by selecting certain columns orall the columns from a table, similar to creating a table from another table.

```
SQL> create view employee_data as select empno,name from employee;

View created.
```

Displaying a view

A view is displayed just as a table in the area.

```
SQL> select * from employee_data;

EMPNO NAME
----- ----------
E102  David
E105  Ross
E106  Chandler
E107  Barney
E200  GARY
E101  SHUBHKIRTI
E103  PHOEBE
E104  Monica
E108  Robin

9 rows selected.
```

Dropping a view

A view can be dropped similarly to any other table as follows:

```
SQL> drop view employee_data;

View dropped.
```

INTRODUCTION TO PL/SQL

PL/SQL is a procedural programming language designed specifically that allows user to run SQL commands within its program, allowing more feasibility in accessing and editing database. If offers procedural commands such as IF statements, loops and assignment operators, that are organized in blocks to be executed.

A PL/SQL program is combined of three major blocks:

Declarative (DECLARE) : Statements that declare variables, constants, and other elements

Executable (BEGIN…END) : Statements that are run when the block is executed

Exception Handling (EXCEPTION…END) : Statements to execute when programs catches an exception

PROGRAMMING IN PL/SQL

Q1. Take the radius of a circle as input from user and display area and circumference of the circle.

```
SQL> declare
  2  rad number(20);
  3  area number(5,2);
  4  cf number(5,2);
  5  pi number(5,2):=3.14;
  6  begin
  7  rad:='&rad';
  8  area:=pi*power(rad,2);
  9  cf:=2*pi*rad;
 10  dbms_output.put_line('Circumference: '||cf);
 11  dbms_output.put_line('Area: '||area);
 12  end;
 13  /
Enter value for rad: 3
Circumference: 18.84
Area: 28.26

PL/SQL procedure successfully completed.
```

Q2. Take a random number from user as input and display its factorial.

```
SQL> declare
  2  num number(5);
  3  fact number(20):=1;
  4  begin
  5  num:='&num';
  6  for cntr in reverse 1..num
  7  loop
  8  fact:=fact*cntr;
  9  end loop;
```

```
 10  dbms_output.put_line('Factorial: '||fact);
 11  end;
 12  /
Enter value for num: 7
Factorial: 5040

PL/SQL procedure successfully completed.
```

Q3. Take a number as input from user and display that numbers of Fibonacci Series.

```
SQL> declare
  2  num number(5);
  3  var1 number(5):=1;
  4  var2 number(5):=1;
  5  temp number(5);
  6  begin
  7  num:='&numbers_to_display';
  8  dbms_output.put_line('1');
  9  dbms_output.put_line('1');
 10  for cntr in 1..num
 11  loop
 12  temp:=var1+var2;
 13  dbms_output.put_line(temp);
 14  var1:=var2;
 15  var2:=temp;
 16  end loop;
 17  end;
 18  /
Enter value for numbers_to_display: 12
1
1
2
3
5
8
13
21
34
55
89
144
233
377

PL/SQL procedure successfully completed.
```

Q4. Take a year as input from user and check if it is a leap year or not.

```
SQL> declare
  2  year number(20);
  3  var1 number(20);
  4  var2 number(20);
  5  var3 number(20);
  6  begin
  7  year:='&year';
  8  var1:=mod(year,4);
  9  var2:=mod(year,100);
 10  var3:=mod(year,400);
```

```
 11  if var1=0 and var2=0 and var3=0 then
 12  dbms_output.put_line('Leap year!');
 13  elsif  var1=0 and not var2=0 and not var3=0 then
 14  dbms_output.put_line('Leap year!');
 15  else
 16  dbms_output.put_line('Not a Leap year!');
 17  end if;
 18  end;
 19  /
Enter value for year: 2014
Not a Leap year!

PL/SQL procedure successfully completed.
```

Q5. Take a number as input from user and check if it is prime or not.

```
SQL> declare
  2  num number(5);
  3  temp number(5);
  4  var number(5);
  5  flag number(5):=0;
  6  begin
  7  num:='&num';
  8  temp:=num/2;
  9  for cntr in 2..temp
 10  loop
 11  var:=mod(num,cntr);
 12  if var=0 then
 13  flag:=1;
 14  end if;
 15  end loop;
 16  if flag=1 then
 17  dbms_output.put_line('Not a Prime Number');
 18  else
 19  dbms_output.put_line('Prime Number');
 20  end if;
 21  end;
 22  /
Enter value for num: 13
Prime Number

PL/SQL procedure successfully completed.
```

Q6. Take two numbers as inputs from user and a basic operation from +,-,,/ and display the result of the operation between them (Calculator).*

```
SQL> declare
  2  num1 number(5);
  3  num2 number(5);
  4  result number(5);
  5  op varchar(1);
  6  begin
  7  num1:='&num1';
  8  num2:='&num2';
  9  op:='&Operation';
 10  case op
 11  when '+' then result:=num1+num2;
 12  when '-' then result:=num1-num2;
```

```
 13  when '*' then result:=num1*num2;
 14  when '/' then result:=num1/num2;
 15  else dbms_output.put_line('Wrong Option Selected!');
 16  end case;
 17  dbms_output.put_line('The result of operation is '||result);
 18  end;
 19  /
Enter value for num1: 12
Enter value for num2: 4
Enter value for operation: *
The result of operation is 48

PL/SQL procedure successfully completed.

Enter value for num1: 45
Enter value for num2: 15
Enter value for operation: ]
Wrong Option Selected!
The result of operation is

PL/SQL procedure successfully completed.
```

Cursors

A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set.

Implicit Cursor

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed. Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement.

Program to count records using implicit cursor

```
SQL> declare
  2  total_rows number(2);
  3  begin
  4  update employee set salary = salary + 50000;
  5  if sql%notfound then
  6  dbms_output.put_line('No employees updated!');
  7  elsif sql%found then
  8  total_rows:=sql%rowcount;
  9  dbms_output.put_line(total_rows||' employees updated ');
 10  end if;
 11  end;
 12  /

9 employees updated

PL/SQL procedure successfully completed.
```

Explicit Cursor

Explicit cursors are programmer-defined cursors for gaining more control over the context area. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement.

Displaying rows present in cursor area

```
SQL> declare
  2  cursor c1 is select * from employee order by empno;
  3  rec c1%rowtype;
  4  begin
  5  open c1;
  6  loop
  7  fetch c1 into rec;
  8  exit when c1%notfound;
  9  dbms_output.put_line('Employee No: '|| rec.empno || ' Name: ' || rec.name);
 10  end loop;
 11  end;
 12  /
Employee No: E101 Name: Shubh
Employee No: E102 Name: David
Employee No: E103 Name: Alex
Employee No: E104 Name: Monica
```

```
Employee No: E105 Name: Ross
Employee No: E106 Name: Chandler
Employee No: E107 Name: Barney
Employee No: E108 Name: Robin
Employee No: E200 Name: Gary

PL/SQL procedure successfully completed.
```

Triggers

Triggers are stored programs, which are automatically executed or fired when some events occur.

Triggering update in table while updating or inserting

```
SQL> create or replace trigger uppercase
  2  before insert or update on employee
  3  for each row
  4  begin
  5  :new.name:=UPPER(:new.name);
  6  end uppercase;
  7  /

Trigger created.

SQL> update employee set name='Phoebe' where name='Alex';

1 row updated.

SQL> select * from employee where job='AM';

EMPNO NAME       JOB   JOININGDA     SALARY MANID DEPTNO
----- ---------- ----- --------- ---------- ----- ------
E102  David      AM    09-DEC-13     350000 E101  D101
E103  PHOEBE     AM    15-DEC-13     350000 E101  D101
E104  Monica     AM    15-DEC-13     345000 E101  D101
```

Triggering update in table while updating

```
SQL> create or replace trigger salary_difference
  2  before update on employee
  3  for each row
  4  declare
  5  sal_dif number;
  6  begin
  7  sal_dif:= :new.salary-:old.salary;
  8  dbms_output.put_line('Salary difference: '||sal_dif);
  9  end salary_difference;
 10  /

Trigger created.

SQL> update employee set salary=40000 where job='Int';
Salary difference: 5000

1 row updated.
```

Sequences

Sequences help us create a sequence for a column that uses current value or can also us the next value. This feature in normal SQL acts like AUTO_INCREMENT. The major difference is that a sequence has many extra features like MAXVALUE, MINVALUE, START WITH, CYCLE, CACHE etc.

Creating a sequence to increment salary of next employee

```
SQL> create sequence salary_intern
  2  increment by 1500
  3  start with 30000
  4  maxvalue 300000
  5  minvalue 30000
  6  cycle
  7  cache 20;

Sequence created.

SQL> declare
  2  sal number;
  3  begin
  4  select salary_intern.nextval into sal from dual;
  5  insert into employee values
  6  ('E200','Gary','Int','02-02-2015',sal,'E100','D100');
  7  end;
  8  /

Statement Processed.
```

Deleting or Dropping a sequence

```
SQL> drop sequence salary_intern;

Sequence dropped.
```

Correlated Subqueries

SQL Correlated Subqueries are used to select data from a table referenced in the outer query. The subquery is known as a correlated because the subquery is related to the outer query.

Example:

```
SQL> select name from employee emp where salary=(select avg(salary) from employee
where deptno=emp.deptno);

NAME
----------
Barney
Shubh
Robin
```

Package

Packages are schema objects that groups logically related PL/SQL types, variables, and subprograms.
A package will have two mandatory parts −

- Package specification
- Package body or definition

Creating Package or Writing Package Specification

```
SQL> create package employee_management as
  2  procedure hire_emp(empno varchar,name varchar,job varchar,joiningdate
date,salary number,manid varchar,deptno varchar);
  3  procedure fire_emp(empno varchar);
  4  procedure raise_salary(empno varchar,increment number);
  5  end employee_management;
  6  /

Package created.
```

Creating Package Functions or Writing Package Body

```
SQL> create or replace package body employee_management as
  2  procedure hire_emp(empid varchar,ename varchar,ejob varchar,ejoiningdate
date,esalary number,emanid varchar,edeptno varchar) is
  3  begin
  4  insert into employee
  5  values(empid,ename,ejob,ejoiningdate,esalary,emanid,edeptno);
  6  end hire_emp;
  7  procedure fire_emp(empid in varchar) is
  8  begin
  9  delete from employee where empno=empid;
 10  if sql%notfound then
 11  dbms_output.put_line('No record found!');
 12  end if;
 13  end fire_emp;
 14  procedure raise_salary(empid varchar,increment number) is
 15  begin
 16  update employee set salary=salary+increment where empno=empid;
 17  if sql%notfound then
 18  dbms_output.put_line('No record found!');
 19  end if;
 20  end raise_salary;
 21  end;
 22  /

Package body created.
```

Deleting or Dropping Package

```
SQL> drop package employee_management;

Package dropped.
```

SELECT CASE

Used to set cases where multiple possibilities arise.

```
SQL> select empno,name,job,joiningdate,salary,manid,deptno,case (job)
  2  when 'AM' then 'AMT'
  3  when 'SM' then 'SMT'
  4  else job
  5  end as NEWT
  6  from employee;

EMPNO NAME       JOB   JOININGDA     SALARY MANID DEPTNO NEWT
----- ---------- ----- --------- ---------- ----- ------ -----
E102  David      AM    09-DEC-13     350000 E101  D101   AMT
E105  Ross       SM    30-DEC-13     250000 E102  D102   SMT
E106  Chandler   SM    20-JUN-14     275500 E102  D102   SMT
E107  Barney     SM    20-JUN-14     255000 E104  D110   SMT
E200  Gary       Int   02-FEB-15      20000 E100  D100   Int
E101  Shubh      CEO   09-DEC-13    5000000 E100  D100   CEO
E103  Alex       AM    15-DEC-13     350000 E101  D101   AMT
E104  Monica     AM    15-DEC-13     345000 E101  D101   AMT
E108  Robin      HRM   20-JUN-14     300500 E103  D115   HRM

9 rows selected.
```

ROWID

Returns the address of each row displayed in the query.

```
SQL> select ROWID,empno from employee;

ROWID              EMPNO
------------------ -----
AAAE+PAAEAAAAOHAAA E101
AAAE+PAAEAAAAOGAAA E102
AAAE+PAAEAAAAOHAAB E103
AAAE+PAAEAAAAOHAAC E104
AAAE+PAAEAAAAOGAAB E105
AAAE+PAAEAAAAOGAAC E106
AAAE+PAAEAAAAOGAAD E107
AAAE+PAAEAAAAOHAAD E108
AAAE+PAAEAAAAOGAAE E200

9 rows selected.
```

INDEX

Creates an indexed entry for each value in the table. By default Oracle creates B-Tree indexes.

Creating an Index

```
SQL> create index INDEX_ROLLNO on student(name);

Index created.
```

Creating an Unique Index

```
SQL> create unique index INDEX_EMPNUMBER on employee(name);

Index created.
```

Dropping an Index

```
SQL> drop index INDEX_EMPNUMBER;

Index dropped.
```