SQL

Structured Query Language(SQL):

1. SQL is **developed** by **IBM in 1970.**
2. SQL became a **standard of the American National Standards Institute (ANSI)** in 1986, and of the **International Organization for Standardization (ISO)** in 1987.
3. The **first name** of SQL **is SEQUEL(Simple or Structured English query language).**
4. SQL is **not case sensitive.**
5. SQL – it is a standard language for **dealing with Relational Databases.** Such as **access the data (or) manipulating data from our database.**
6. SQL can be effectively used to **insert, search, update, delete database records.**

Data:

1. Basically **data is a raw material** when we **process it** then it will be **useful information.**
2. Data is the **set of useful information.**
3. Which it is **helps to make a decision.**
4. **Set of information of any entity** is known as data.

Example:

- Employee information.(Name,job,sal etc…)
- Product information,Sales information.

Database:

1. **Place** where you **store data.**
2. A **database** is a systematic **collection of data.**
3. Databases **support storage and manipulation of data.**
4. Databases **make data management easy.**

Ex:

- An online telephone directory.
- Electricity service provider
- Facebook.

Note **:-**

- If we install any of the database related software(s) – we can create our own database, we can create our own tables and we can store the data inside it.
- When we install any database s/w(s) – a part of hard disk will be designated / reserved to perform database related activities
- A database can also contain other database objects like views, indexes, stored procedures, functions, triggers etc, apart from tables.

Some of the database software(s) we have are,
Oracle, SQL Server, DB2, Sybase, Informix, MySQL, MS – Access, Foxbase, FoxPro

Among the above database software – some of them are DBMS and some of them are RDBMS

The s/w which is widely used today is Oracle. The different versions of Oracle starting from the earliest to the latest are – Oracle 2, Oracle 3,

Oracle 4, Oracle 5, Oracle 6, Oracle 7, Oracle 8i, Oracle 9i, Oracle 10g, and the latest to hit the market is Oracle 11g. here ‘i’ stands for Internet and ‘g’ stands for Grid / Grid computing.

DBMS

Database Management System (DBMS):

1. **Data base combined with management system services** is known as DBMS.
2. Database Management System (DBMS) is a **collection of programs** which **enables its users to access database, manipulate data, reporting / representation of data.**

Here the **services are**

1. Entering new data
2. Updating old data with new data
3. Deleting unwanted data
4. Authenticating the users
5. Providing security.

Ex for DBMS:

- FoxPro, FoxBase, Dbase

DBMS

DATABASE:

A **database** is *an organized collection of data*.

Database handlers create database in such a way that only one set of software program provide access of data to all the users.

The **main purpose** of database is to operate large amount of information by storing, retrieving and managing.

There are many **dynamic websites** on the world wide web now a days which are handled through databases. For example, a model to checks the availability of rooms in a hotel. It is an example of dynamic website that uses database.

There are many **database available** like MySQL, Sybase, Oracle, Mango DB, Informix, Postgre, SQL Server etc.

SQL or Structured Query Language is used to perform operation on the data stored in a database. SQL depends on relational algebra and tuple relational calculus.

A cylindrical structure is used to display the image of a database.

SQL CREATE Database

The **SQL CREATE DATABASE** statement is used by a developer to create a database.

Let's see the syntax of SQL CREATE DATABASE:

1. **CREATE DATABASE** database_name;

If you want to add tables in that database, you can use CREATE TABLE statement.

Create Database in MySQL

In MySQL, same command is used to create a database.

1. **CREATE DATABASE** database_name;

Create Database in Oracle

You don't need to create database in Oracle. In Oracle database, you can create tables directly.

You can also rename, drop and select database that is covered in next pages.

> ***We should always remember that database name should be unique in the RDBMS.***

SQL DROP Database

SQL DROP statement is used to delete or remove indexes from a table in the database.

If you want to delete or drop an existing database in a SQL schema, you can use SQL DROP DATABASE

Let's see the syntax of SQL DROP DATABASE:

1. **DROP DATABASE** database_name;

> ***If you delete or drop the database, all the tables and views will also be deleted. So be careful while using this command.***

SQL RENAME Database

SQL RENAME DATABASE is used when you need to change the name of your database. Sometimes it is used because you think that the original name is not more relevant to the database or you want to give a temporary name to that database.

Let's see how to rename MySql and SQL Server databases.

Rename MySQL database

To rename the mysql database, you need to follow the following syntax:

1. RENAME **DATABASE** old_db_name **TO** new_db_name;

Rename SQL server database using T-SQL

This command is useful for SQL server 2005, 2008, 2008R2 and 2012.

1. **ALTER DATABASE** old_name **MODIFY NAME** = new_name

If you are using SQL server 2000, you can also use this command to rename the database. But, Microsoft phased out it.

1. **EXEC** sp_renamedb 'old_name' , 'new_name'

SQL SELECT Database

In MySQL database, you need to select a database first before executing any query on table, view etc. To do so, we use following query:

1. USE **DATABASE** database_name;

In oracle, you don't need to select database.

RDBMS

Relational Database Management System (RDBMS):

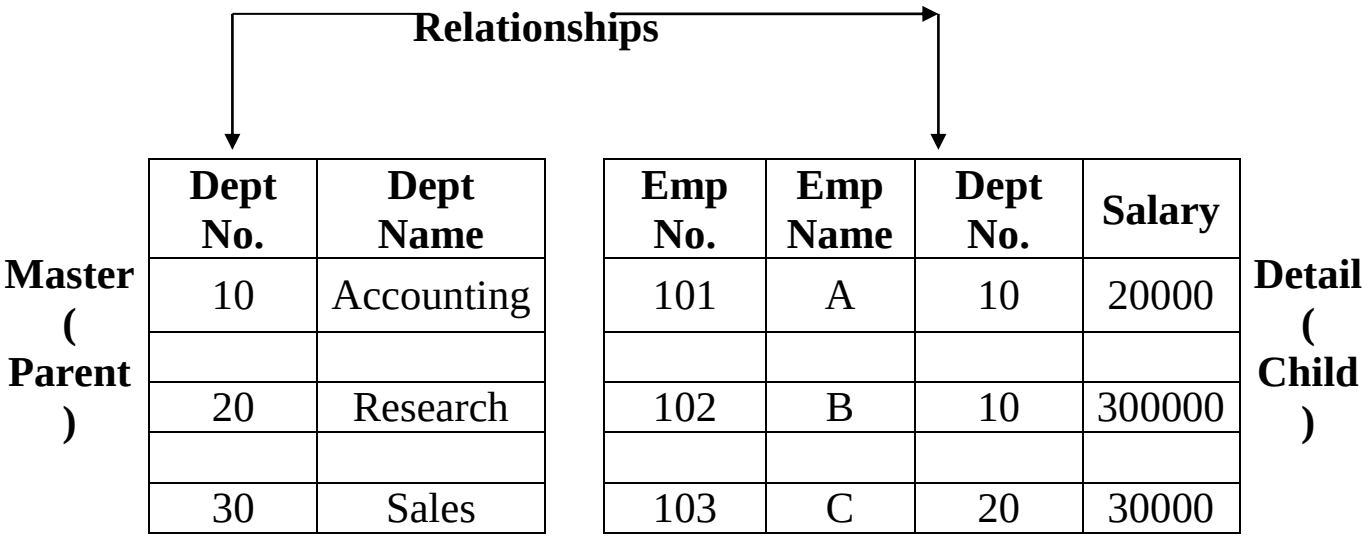
1. RDBMS is also a database s/w which has facility to handle more **data volume, good performance, enhanced security features etc when compared against DBMS.**
2. **Any DBMS to qualify as a RDBMS** should support the Codd rules / Codd laws.
3. It defines **relationships in the form of tables.**
4. **Developed** By **E F Codd** in the year **1970.**
5. All the **latest database products** use this model.
6. Entire **database** is **divided** into number of **tables** and they are **connected through "Key Field".**
7. Data is **stored in the form of table**. Table **consists** of several **rows** and **column.**
8. **Rows** are also called as **record (or) tuple.**it is in **horizontal form.**
9. **Columns** are also called as **field (or) attribute**.it is in **vertical form.**
10. The **intersection of rows and columns** are called as **cell.**
11. **Tables** are also called as **relation, entity and object.**

Ex for RDBMS:

- Oracle, Sybase, DB2, Teradata, SQL Server, MySQL

RELATIONSHIPS:

- A **relationship** is the **association between any two tables** which **preserves data integrity.**



1. Relationship **helps to prevent the incorrect data in the child tables.**
2. Once the **relationship is created, one table becomes master (or parent) and the other one becomes the child (or detail).**
3. Whatever we insert the data **(column in child table like deptno)** into the child should be **present in the master, else the record will be rejected from the child.**
4. The **master table contains** the master **data** which will **not change frequently.**
5. The **child table contains** the transactional **data** which will **change quite often.**

DATA INTEGRITY

DATA INTEGRITY:

Data integrity is **used to restrict invalid data into the table.**

We can **achieve data integrity** by

1. Data type
2. Constraints

EX:

Number	Name
-3446	Mahe

1.DATA TYPE:

1. It is nothing but the **type of data given to every individual column.**

TYPES OF DATA TYPE:

1. NUMBER
2. CHAR/VARCHAR
3. DATE.

NUMBER:

1. It is used to **input the numeric value into a table column.**

EX:

2. Number(2) → -99 to 99
3. Number(4) → -9999 to 9999
4. Number(4,2) → -99.99 to 99.99
5. Number(8,2) → -999999.99 to 999999.99

CHAR/VARCHAR:

CHAR	VARCHAR
Here we can **enter up to 2000 characters.**	Here we can **enter 4000 characters.**
It will **allow null values.**	It will **not allow null values.**
It has **fixed length.**	It has **variable length.**
It **allows only characters.**	It **allows alphanumeric.**
Memory consumption.	**No memory consumption.**

DATE:

- Here we have default method called as **DATE to use to enter** the **different type date format.**

default format :- dd – mon – yy

BLOB:

Stands for – Binary Large Object
It stores binary data (images, movies, music files) within the database. It stores upto 4GB.

CLOB:

Stands for – Character Large Object
It stores plain character data like **varchar** field upto 4GB.

SQL Data Types

The SQL data type defines a kind of value that a column can contain.

In a database table, every column is required to have a name and a data type.

> **Data Type varies from database to database. For example, MySQL supports INT but Oracle supports NUMBER for integer values.**

These are the general data types in SQL.

Data-type	Syntax	Explanation

Integer	INTEGER	The integer data type is used to specify an integer value.
Smallint	SMALLINT	The smallint data type is used to specify small integer value.
Numeric	NUMERIC(P,S)	It specifies a numeric value. Here 'p' is precision value and 's' is scale value.
Real	REAL	The real integer is used to specify a single precision floating point number.
Decimal	DECIMAL(P,S)	It specifies a decimal value. Here 'p' is precision value and 's' is scale value.
Double precision	DOUBLE PRECISION	It specifies double precision floating point number.
Float	FLOAT(P)	It specifies floating-point value e.g. 12.3, 4.5 etc. Here, 'p' is precision value.
Character	CHAR(X)	Here, 'x' is the character's number to store.
Character varying	VARCHAR2(X)	Here, 'x' is the character's number to store
Bit	BIT(X)	Here, 'x' is the number of bits to store
Bit varying	BIT VARYING(X)	Here, 'x' is the number of bits to store (length can vary up to x).
Date	DATE	It stores year, month and days values.
Time	TIME	It stores hour, minute and second values
Timestamp	TIMESTAMP	The timestamp data type is used to store year, month, day, hour, minute and second values.
Time with time zone	TIME WITH TIME ZONE	It is exactly same as time but also store an offset from UTC of the time specified.
Timestamp with time	TIMESTAMP with TIME ZONE	It is same as timestamp but also stores an offset from UTC of the time specified.

zone		

2.Constraints:

1. Constraints are nothing but **condition** which **restricts invalid data** in the tables.
2. It is **provided to the columns** of the table.

Types of constraints:

1. Not Null
2. Unique
3. Check
4. Primary key
5. Foreign key

NULL:

1. NULL is nothing but it is **neither zero nor blank space.**
2. Null **doesn’t occupy any space** in the memory.
3. Null represent **unknown value.**
4. **Two null never ever occur** in oracle.
5. If we **perform any operation** (arithmetic operation) with **null** its **result** in **null** itself.

Ex:

- Null+100=null
- Null*109=null

NOT NULL:

1. Not null value constraint ensures **some value present** in the **column**.

UNIQUE:

1. It **won't accept the duplicate value** in column.
2. It can **take multiple null values.**

CHECK:

1. It is nothing but we are **providing additional validation** as per the **customer requirement specification.**

Ex:

Check(Phoneno=10)

PRIMARY KEY:

1. It is the combination of **not null** and **unique** constraint.
2. It is column(s) which **uniquely identifies** a row.
3. Creation of primary key is **not mandatory** but **highly recommend**.
4. In a table **only one** primary key is allowed.

EX:

Empno	Ename	Sal	Dept	Hiredate
1	Mahe	50000	Dev	18-7-1995
2	Gokul	55000	Dev	18-7-2017
3	Viky	20000	Test	15-3-2017
4	Vicky	30000	Test	16-3-2012

CANDIDATE KEY:

1. **Columns** which are **eligible to become primary key** is called as candidate key.

- Not Null+unique = candidate key (primary key is allowed)

Empno	Ename	Sal	Dept	Hiredate	Phoneno	Email id
1	Mahe	50000	Dev	18-7-1995	9148848975	Maherms95@gmail.com
2	Gokul	55000	Dev	18-7-2017	9312534642	gokulkala@gmail.com
3	Viky	20000	Test	15-3-2017	9346312423	Vicky@gmail.com
4	Vicky	30000	Test	16-3-2012	9325263673	Vicky213@gmail.com

ALTERNATE KEY:

1. **Column** which are **eligible to become primary key but not chosen as primary key.**

Empno	Ename	Sal	Dept	Hiredate	phoneno	Email id
1	Mahe	50000	Dev	18-7-1995	9148848975	Maherms95@gmail.com
2	Gokul	55000	Dev	18-7-2017	9312534642	gokulkala@gmail.com
3	Viky	20000	Test	15-3-2017	9346312423	Vicky@gmail.com
4	Vicky	30000	Test	16-3-2012	9325263673	Vicky213@gmail.com

NOT NULL+unique = alternate key (primary key is not allowed)

FOREIGN KEY:

1. Foreign key is **use to create relationship between two table.**
2. It is also called as **referential integrity constraint.**
3. In a table, we can **have more than one foreign key.**
4. **Foreign key** is **created in child table.**
5. **Foreign key of child table** will be the **primary key of master table.**
6. It can **accept null and duplicate value.**

EX:

Empno	Ename	Sal	Comm	Hiredate	Deptno	dname	Dloc
1	Chandru	10000		15-mar-2017	10	Hr	Chennai
2	Raja	20000	0	5-may-2013	30	Dev	Ban
3	Maha	23000		8-aug-2015	20	Test	Cbe
4	Raji	13000	500	9-feb-2016	20	Dev	Chennai
5	Raja	54000	1000	5-jan-2011	10	Test	Ban
6	Golu	65342		9-apr-2016	30	Hr	Cbe

EX:Employee table

Empno	Ename	Sal	Comm	Hiredate	Deptno	Project id
1	Chandru	10000		15-mar-2017	10	101
2	Raja	20000	0	5-may-2013	30	302
3	Maha	23000		8-aug-2015	20	203
4	Raji	13000	500	9-feb-2016	20	202
5	Raja	54000	1000	5-jan-2011	10	104
6	Golu	65342		9-apr-2016	30	306

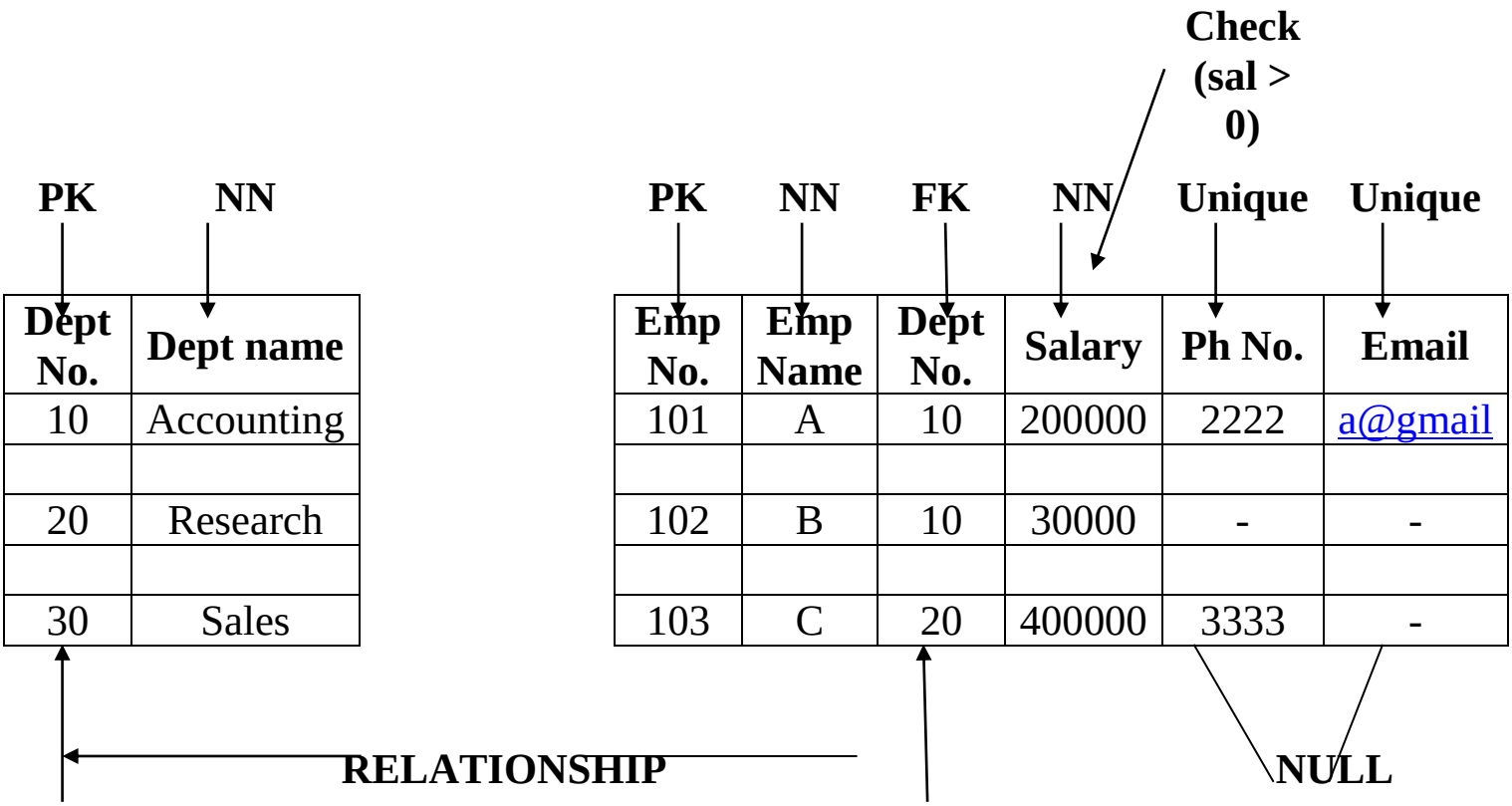
EX: dept table

Deptno	dname	Dloc
10	Hr	Chennai
30	Dev	Ban
20	Test	Cbe
20	Dev	Chennai
10	Test	Ban
30	Hr	Cbe

EX: project table

Deptno	dname	Dloc	Project id
10	Hr	Chennai	101
30	Dev	Ban	302
20	Test	Cbe	203
20	Dev	Chennai	202
10	Test	Ban	104
30	Hr	Cbe	306

EX FOR CONSTRAINT:



Dept No.	Dept name
10	Accounting
20	Research
30	Sales

Emp No.	Emp Name	Dept No.	Salary	Ph No.	Email
101	A	10	200000	2222	a@gmail
102	B	10	30000	-	-
103	C	20	400000	3333	-

SQL COMMANDS

1. **SHOW USER:**

   It is use to display the current user name.

2. **CLEAR SCREEN (or) CL SCR:**

   It is used to clear the screen.

3. **SELECT * FROM TAB;**

   It is use to display the table which are present in that account.

   EX:

TNAME	TABTYPE	CLUSTERID
DEPT	TABLE	
EMP	TABLE	
BONUS	TABLE	
SALGRADE	TABLE	

4. **DESCRIBE DEPT (or) desc dept:**

   It is used to display the table structure

Name	Null?	Type

DEPTNO	NOT NULL	NUMBER(2)
DNAME		VARCHAR2(14)
LOC		VARCHAR2(13)

5. **EXIT:**

   It is used to exit.

6. **Select * from (table name) dept;**

   It is used to retrieve all the records from that table.

7. **Show linesize:**

   It is used to show line size.

8. **Show pagesize:**

   It is used to show the page size.

9. **Set linesize120 pagesize 120** (or) **set lines 120 pages 120**

   It is used to set the line size and page size.

10. **Conn:**

    It is used to switch from one account to another account.

    SQL> conn

    Enter user-name: hr

    Enter password: *****

Connected.

```
SQL> show user
USER is "HR"
SQL> select * from tab;

TNAME                          TABTYPE        CLUSTERID
------------------------------ -------        ----------
REGIONS                        TABLE
COUNTRIES                      TABLE
LOCATIONS                      TABLE
DEPARTMENTS                    TABLE
JOBS                           TABLE
EMPLOYEES                      TABLE
JOB_HISTORY                    TABLE
EMP_DETAILS_VIEW               VIEW

8 rows selected.
```

11. EDIT:

Edit command is used to modify the previously executed query.

SQL> select ename,job,hiredate from emp;

Ans:

It will open one notepad file and data will be store like this →

select ename from emp

/

12. **List:**

List command is used to display previously (only last executed) executed query.

SQL> select dname from dept;

SQL> list

1* select dname from dept

13. **Save:**

Save command is used to save the previously (only last executed) executed query.

SQL> select dname from dept;

SQL> save "E:\mahe.doc"

Created file E:\mahe.doc

14. **Replace:**

Replace command is used to override the existing document.

SQL> select job from emp;

SQL> save "E:\mahe.doc"

SP2-0540: File "E:\mahe.doc" already exists.

Use "SAVE filename[.ext] REPLACE".

SQL> save "E:\mahe.doc" replace;

Wrote file E:\mahe.doc

15. **Append:**

Append is used to save multiple command in same file.

SQL> selct job from emp;

SQL> select loc from dept;

SQL> save "E:\mahe.doc" append;

Appended file to E:\mahe.doc

16. (@):

It is used to execute all the queries present in a single file.

SQL> @ "E:\mahe.doc";

JOB

CLERK

SALESMAN

SALESMAN

MANAGER

SALESMAN

MANAGER

MANAGER

ANALYST

PRESIDENT

SALESMAN

CLERK

CLERK

ANALYST

CLERK

14 rows selected.

JOB

CLERK

SALESMAN

SALESMAN

MANAGER

SALESMAN

MANAGER

MANAGER

ANALYST

PRESIDENT

SALESMAN

CLERK

CLERK

ANALYST

CLERK

14 rows selected.

17. **Spool:**

Spool command is used to save statement along with their output.

SQL> select * from emp;

SQL> spool "E:\mahe1.doc";

SQL> select * from 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 19-APR-87 3000 20

7839 KING PRESIDENT 17-NOV-81 5000 10

7844 TURNER SALESMAN 7698 08-SEP-81 1500 0 30

7876 ADAMS CLERK 7788 23-MAY-87 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

14 rows selected.

SQL> spool off;

SQL* Plus/SQL Plus

- Client tool to access the database
- Installed along with database
- Has specific set of commands
- Interface btn user and data base

Difference between SQL Statement and SQL* Plus command

SL No.	SQL Statement	SQL*Plus
1	A Language	An Environment
2	ANSI Standard	Oracle Proprietary
3	Keywords Cannot be abbreviated	Keywords can be abbreviated
4	Statement manipulate data and table definitions in the database	Commands do not allow manipulation of values in database

SQL STATEMENTS

SQL STATEMENTS:

1. DATA QUERY LANGUAGE
2. DATA DEFINITION LANGUAGE
3. DATA MANIPULATION LANGUAGE
4. TRANSACTION CONTROL LANGUAGE
5. DATA CONTROL LANGUAGE

DATA QUERY LANGUAGE:

1. SELECT

DATA DEFINITION LANGUAGE:

1. CREATE
2. ALTER
3. RENAME
4. TRUNCATE
5. DROP

DATA MANIPULATION LANGUAGE:

1. INSERT
2. UPDATE
3. DELETE

TRANSACTION CONTROL LANGUAGE:

1. COMMIT
2. ROLLBACK
3. SAVEPOINT

DATA CONTROL LANGUAGE:

1. GRANT
2. REVOKE

SQL Statements

1	Select	Data retrieval language
2	Insert	Data manipulation Language
	Update	
	Delete	
3	Create	Data definition language
	Alter	
	Drop	
	Rename	
	Truncate	
4	Commit	Transaction Control Language
	Rollback	
	Save Point	
5	Grant, Revoke	Data Control language

DATA QUERY LANGUAGE

DATA QUERY LANGUAGE:

1. Select

Capability of select statement:

1. Projection
2. Selection
3. Joins

PROJECTION:

Select *|{[distinct] column_name|expression [alias…],….} from table_name;

EX:

Select ename from emp;

DISTINCT:

1. It is used to display the unique value.

EX:

SQL> select deptno from emp;

DEPTNO

20

30

30

20

30

30

10

20

10

30

20

30

20

10

14 rows selected.

SQL> select distinct deptno from emp;

DEPTNO

30

20

10

EXPRESSION:

It is used to perform some specific operation.

EX:

SQL> select 1+4 from emp;

1+4

5

5

5

5

5

5

5

5

5

5

5

5

5

5

14 rows selected.

DUAL:

1. Dual is a dummy table where you can perform independent calculation.
2. It has one column and one row.

EX:

SQL> select 1+4 from dual;

1+4

5

LITERALS:

1. Literals are the value used in select statements.
2. It can be number, character, and date.
3. For character and date literal it should be enclosed by single quotes.

SQL> select qsp from dual;

```
select qsp from dual
       *
ERROR at line 1:
ORA-00904: "QSP": invalid identifier
SQL>  select 'Mahe' as MAHE from dual;
MAHE
----
Mahe
```

Structure of dual:

```
SQL> desc dual;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 DUMMY                                              VARCHAR2(1)
```

```
SQL> select * from dual;

D
-
X
```

EX:

SQL> select ename from emp;

ENAME

SMITH

ALLEN

WARD

JONES

MARTIN

BLAKE

CLARK

SCOTT

KING

TURNER

ADAMS

JAMES

FORD

MILLER

14 rows selected.

EX 1:

SQL> select 'ename' from emp;

'ENAM

-----

ename

ename

ename

ename

ename

ename

ename

ename

ename

ename

ename

ename

ename

ename

14 rows selected.

EX 2:

SQL> select 'mahe' as mahe from dual;

MAHE

Mahe

ALIAS:

1. Alias is the alternative name given to the column name
2. Alias name should mention next to the column name
3. Between column name and alias name we can use as keyword. Using as keyword is not mandatory.
4. Alias is mainly use for calculation.
5. If they are using any special character or spaces in alias name then it should be enclosed with double quotes.
6. In alias underscore can be consider as a character and hence we won't get any error.

Concatenation Operator:

1. Concatenates columns or character strings to other columns
2. Is represented by two vertical bars (||)

EX:

SQL> Select ename||job||sal from emp;

Selection:

1. Selection is used to select the rows or records.

Syntax:

Select */{[distinct] column_name/expression [alias…]}from table_name where<conditions>;

Ex:

select ename from emp;

DATA DEFINITION LANGUAGE

The various commands in DDL are :-

1. **Create**
2. **Alter**
3. **Rename**
4. **Drop**
5. **Truncate**

CREATE:

1. It creates the table.
2. Before we study the **Create** command, let us first study the some of the basic **data types** we use in SQL.

Syntax:

```
SYNTAX:
CREATE TABLE table_name(
column1 datatype(size) constraints,
column2 datatype(size) constraints,
                    ……
                    .....
columnN datatype(size) constraints);
```

EX:

STUDENT-USN(PK),SNAME,SEMESTER,CNAME

LIBRARY-LID(PK),BNAME,BPRICE,AUTHOR

PROJECT-PID(PK),PNAME,PLOC

Rename:

1. It renames a table.

Syntax:

Rename table_name to new_table_name

EX:

SQL> rename student to stu;

Table renamed.

Syntax rename the column

Alter table table_Name
rename column Column_Name to
New_Column_Name

TRUNCATE:

1. It removes all the data permanently, but the structure of the table remains as it is.

Syntax:

TRUNCATE TABLE <TABLE_NAME> ;

DROP:

It removes both data and the structure of the table permanently from the database.

Ex – SQL > DROP TABLE test ;

Let us understand the difference between **drop & truncate** using the below shown example,

```
SQL> CREATE TABLE test1
  2  AS
  3  select * from dept ;

Table created.
```

```
SQL> CREATE TABLE test2
  2  AS
  3  select * from dept ;

Table created.
```

Let us create 2 tables Test1 and Test2 as shown above.

```
SQL> desc test1 ;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 DEPTNO                                             NUMBER(2)
 DNAME                                              VARCHAR2(14)
 LOC                                                VARCHAR2(13)
```

```
SQL> select * from test1 ;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

The above shows the description of the table test1.

```
SQL> desc test2 ;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 DEPTNO                                             NUMBER(2)
 DNAME                                              VARCHAR2(14)
 LOC                                                VARCHAR2(13)


SQL> select * from test2 ;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

The above gives the description of the table Test2.

Now, let us use the **Truncate query on Test1** and **Drop query on Test2** and see the difference.

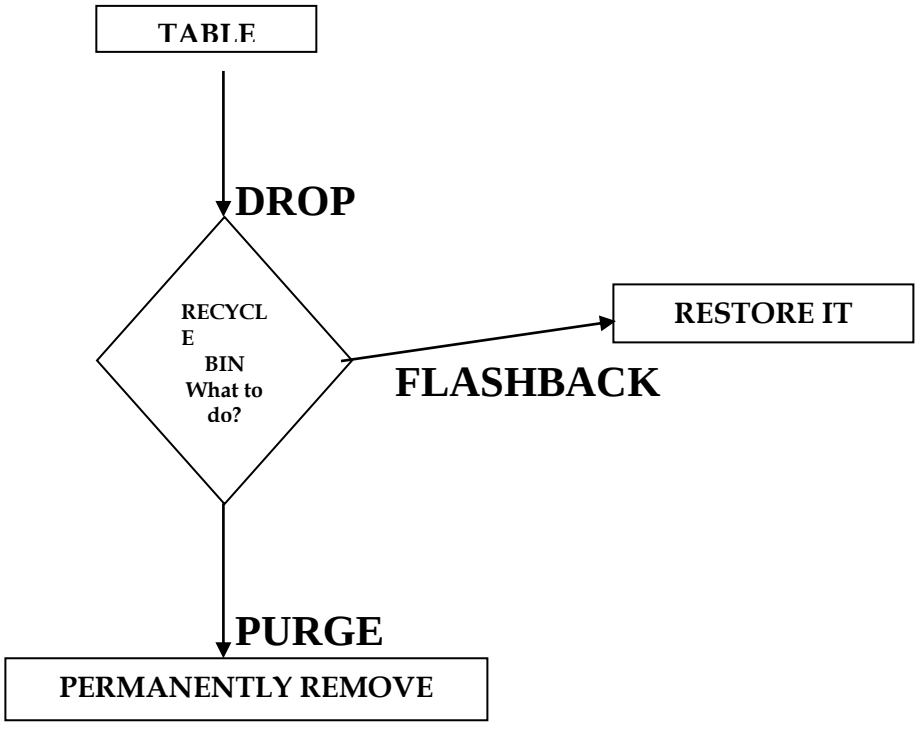
```
SQL> truncate table test1 ;

Table truncated.

SQL> select * from test1 ;

no rows selected

SQL> desc test1 ;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 DEPTNO                                             NUMBER(2)
 DNAME                                              VARCHAR2(14)
 LOC                                                VARCHAR2(13)
```

The above 3 queries show that – 1st query has the table test1 truncated. 2nd query – it shows **no rows selected** – thus only the records from the table has been removed. 3rd query – it shows that the structure of the table is still present. Only the records will be removed.
Thus, this **explains the truncate query.**

```
SQL> drop table test2 ;

Table dropped.

SQL> select * from test2 ;
select * from test2
              *
ERROR at line 1:
ORA-00942: table or view does not exist


SQL> desc test2 ;
ERROR:
ORA-04043: object test2  does not exist
```

Thus from the above queries we can explain how **drop** works. 1st query – it drops the table. Thus – the entire structure and records of the table are dropped.
2nd and 3rd query – since, there is no table – **select & desc** query for **test2** will throw an error.
Thus, this **explains the drop query.**
Hence, we have seen the difference between **drop & truncate** query.

10g Recycle Bin



- The functionality of Recycle Bin was introduced in Oracle 10G version only.
- Thus even though the table has been dropped, we can still restore it using **flashback command** or we can permanently remove it using the **purge** command.
- This concept of Recycle bin was not there in the earlier versions of Oracle.

Recovering Drop Tables (Undo Drop Table):

To recover this dropped table a user can type the command
SQL> Flashback table table_name to before drop;
You can also restore the dropped table by giving it a different name like this
SQL> Flashback table emp to before drop rename to emp2;

Permanently deleting table after drop:

Purge table tableName
Permanently Dropping Tables
If you want to permanently drop tables without putting it into Recycle Bin drop tables with purge command like this
SQL> drop table tablename purge;
This will drop the table permanently and it cannot be restored.

Syntax to drop column

ALTER TABLE table_name DROP COLUMN column_name;

ALTER TABLE table_name DROP (column_name1, column_name2);

SQL Table

Table is a collection of data, organized in terms of rows and columns. In DBMS term, table is known as relation and row as tuple.

> **Note: A table has a specified number of columns, but can have any number of rows.**

Table is the simple form of data storage. A table is also considered as a convenient representation of relations.

Let's see an example of an employee table:

Employee		
EMP_NAME	**ADDRESS**	**SALARY**
Ankit	Lucknow	15000
Raman	Allahabad	18000
Mike	New York	20000

In the above table, "Employee" is the table name, "EMP_NAME", "ADDRESS" and "SALARY" are the column names. The combination of data of multiple columns forms a row e.g. "Ankit", "Lucknow" and 15000 are the data of one row.

Topics of SQL TABLE Statement

SQL TABLE Variable

What TABLE variable can do?

SQL CREATE TABLE

How to create a table using SQL query>

SQL DROP TABLE

How to drop a table?

SQL DELETE TABLE

How to delete all the records of a table?

SQL RENAME TABLE

How to rename a table?

SQL TRUNCATE TABLE

How to truncate a table?

SQL COPY TABLE

How to copy a table?

SQL TEMP TABLE

What is temporary table? What are the advantage of temporary table?

SQL ALTER TABLE

How to add, modify, rename and drop column.

SQL TABLE Variable

The **SQL Table variable** is used to create, modify, rename, copy and delete tables. Table variable was introduced by Microsoft.

It was introduced with SQL server 2000 to be an alternative of temporary tables.

It is a variable where we temporary store records and results. This is same like temp table but in the case of temp table we need to explicitly drop it.

Table variables are used to store a set of records. So declaration syntax generally looks like CREATE TABLE syntax.

1. **create table** "tablename"
2. ("column1" "data type",
3. "column2" "data type",
4. ...
5. "columnN" "data type");

When a transaction rolled back the data associated with table variable is not rolled back.

A table variable generally uses lesser resources than a temporary variable.

Table variable cannot be used as an input or an output parameter.

SQL CREATE TABLE

SQL CREATE TABLE statement is used to create table in a database.

If you want to create a table, you should name the table and define its column and each column's data type.

Let's see the simple syntax to create the table.

```
1. create table "tablename"
2. ("column1" "data type",
3. "column2" "data type",
4. "column3" "data type",
5. ...
6. "columnN" "data type");
```

The data type of the columns may vary from one database to another. For example, NUMBER is supported in Oracle database for integer value whereas INT is supported in MySQL.

Let us take an example to create a STUDENTS table with ID as primary key and NOT NULL are the constraint showing that these fields cannot be NULL while creating records in the table.

```
1. SQL> CREATE TABLE STUDENTS (
2. ID INT                    NOT NULL,
3. NAME VARCHAR (20) NOT NULL,
4. AGE INT                    NOT NULL,
5. ADDRESS CHAR (25),
6. PRIMARY KEY (ID)
7. );
```

You can verify it, if you have created the table successfully by looking at the message displayed by the SQL Server, else you can use DESC command as follows:

SQL> DESC STUDENTS;

FIELD	TYPE	NULL	KEY	DEFAULT	EXTRA
ID	Int(11)	NO	PRI		
NAME	Varchar(20)	NO			

AGE	Int(11)	NO			
ADDRESS	Varchar(25)	YES		NULL	

4 rows in set (0.00 sec)

Now you have the STUDENTS table available in your database and you can use to store required information related to students.

SQL CREATE TABLE Example in MySQL

Let's see the command to create a table in MySQL database.

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

SQL CREATE TABLE Example in Oracle

Let's see the command to create a table in Oracle database.

```
1. CREATE TABLE Employee
2. (
3. EmployeeID number(10),
4. FirstName varchar2(255),
5. LastName varchar2(255),
6. Email varchar2(255),
7. AddressLine varchar2(255),
8. City varchar2(255)
9. );
```

SQL CREATE TABLE Example in Microsoft SQLServer

Let's see the command to create a table in SQLServer database. It is same as MySQL and Oracle.

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

SQL DROP TABLE

A SQL DROP TABLE statement is used to delete a table definition and all data from a table.

This is very important to know that once a table is deleted all the information available in the table is lost forever, so we have to be very careful when using this command.

Let's see the syntax to drop the table from the database.

```
1. DROP TABLE "table_name";
```

Let us take an example:

First we verify STUDENTS table and then we would delete it from the database.

```
1. SQL> DESC STUDENTS;
```

FIELD	TYPE	NULL	KEY	DEFAULT
ID	Int(11)	NO	PRI	
NAME	Varchar(20)	NO		
AGE	Int(11)	NO		

ADDRESS	Varchar(25)	YES		NULL

```
4 rows in set (0.00 sec)
```

This shows that STUDENTS table is available in the database, so we can drop it as follows:

```
SQL>DROP TABLE STUDENTS;
```

Now, use the following command to check whether table exists or not.

```
SQL> DESC STUDENTS;
```

```
Query OK, 0 rows affected (0.01 sec)
```

As you can see, table is dropped so it doesn't display it.

SQL DROP TABLE Example in MySQL

Let's see the command to drop a table from the MySQL database.

```
DROP TABLE table_name;
```

SQL DROP TABLE Example in Oracle

Let's see the command to drop a table from Oracle database. It is same as MySQL.

```
DROP TABLE table_name;
```

SQL DROP TABLE Example in Microsoft SQLServer

Let's see the command to drop a table from SQLServer database. It is same as MySQL.

```
DROP TABLE table_name;
```

SQL DELETE TABLE

The DELETE statement is used to delete rows from a table. If you want to remove a specific row from a table you should use WHERE condition.

1. **DELETE FROM** table_name [**WHERE** condition];

But if you do not specify the WHERE condition it will remove all the rows from the table.

1. **DELETE FROM** table_name;

There are some more terms similar to DELETE statement like as DROP statement and TRUNCATE statement but they are not exactly same there are some differences between them.

Difference between DELETE and TRUNCATE statements

There is a slight difference b/w delete and truncate statement. The **DELETE statement** only deletes the rows from the table based on the condition defined by WHERE clause or delete all the rows from the table when condition is not specified.

But it does not free the space containing by the table.

The **TRUNCATE statement:** it is used to delete all the rows from the table **and free the containing space.**

Let's see an "employee" table.

Emp_id	Name	Address	Salary
1	Aryan	Allahabad	22000
2	Shurabhi	Varanasi	13000
3	Pappu	Delhi	24000

Execute the following query to truncate the table:

1. **TRUNCATE TABLE** employee;

Difference b/w DROP and TRUNCATE statements

When you use the drop statement it deletes the table's row together with the table's definition so all the relationships of that table with other tables will no longer be valid.

When you drop a table:

- Table structure will be dropped
- Relationship will be dropped
- Integrity constraints will be dropped
- Access privileges will also be dropped

On the other hand when we **TRUNCATE** a table, the table structure remains the same, so you will not face any of the above problems.

SQL RENAME TABLE

SQL RENAME TABLE syntax is used to change the name of a table. Sometimes, we choose non-meaningful name for the table. So it is required to be changed.

Let's see the syntax to rename a table from the database.

1. **ALTER TABLE** table_name
2. RENAME **TO** new_table_name;

Optionally, you can write following command to rename the table.

1. RENAME old_table _name **To** new_table_name;

Let us take an example of a table named "STUDENTS", now due to some reason we want to change it into table name "ARTISTS".

Table1: students

Name	Age	City
Amrita gill	25	Amritsar
Amrender sirohi	22	Ghaziabad
Divya khosla	20	Delhi

You should use any one of the following syntax to RENAME the table name:

1. **ALTER TABLE** STUDENTS
2. RENAME **TO** ARTISTS;

   Or

1. RENAME STUDENTS **TO** ARTISTS;
2. **After** that the **table** "students" will be changed **into table name** "artists"

SQL TRUNCATE TABLE

A truncate SQL statement is used to remove all rows (complete data) from a table. It is similar to the DELETE statement with no WHERE clause.

TRUNCATE TABLE Vs DELETE TABLE

Truncate table is faster and uses lesser resources than DELETE TABLE command.

TRUNCATE TABLE Vs DROP TABLE

Drop table command can also be used to delete complete table but it deletes table structure too. TRUNCATE TABLE doesn't delete the structure of the table.

Let's see the syntax to truncate the table from the database.

1. **TRUNCATE TABLE** table_name;

   For example, you can write following command to truncate the data of employee table

1. **TRUNCATE TABLE** Employee;

   Note: The rollback process is not possible after truncate table statement. Once you truncate a table you cannot use a flashback table statement to retrieve the content of the table.

SQL COPY TABLE

If you want to copy a SQL table into another table in the same SQL server database, it is possible by using the select statement.

The syntax of copying table from one to another is given below:

1. **SELECT** * **INTO** <destination_table> **FROM** <source_table>

   For example, you can write following command to copy the records of hr_employee table into employee table.

1. **SELECT** * **INTO** admin_employee **FROM** hr_employee;

> **Note: SELECT INTO is totally different from INSERT INTO statement.**

2. **Create table stu as select * from emp;**

   For example, you can write following command to copy the records of emp table into stu table.

3. **Create table stu as select * from emp where ename='yyyy';**

   For example, you can write following command to copy the structure of emp table into stu table.

4. **Insert into stu select * from emp;**

   For example, you can write following command to copy the result to existing table.

SQL TEMP TABLE

The concept of temporary table is introduced by SQL server. It helps developers in many ways:

Temporary tables can be created at run-time and can do all kinds of operations that a normal table can do. These temporary tables are created inside tempdb database.

There are two types of temp tables based on the behavior and scope.

1. Local Temp Variable
2. Global Temp Variable

Local Temp Variable

Local temp tables are only available at current connection time. It is automatically deleted when user disconnects from instances. It is started with hash (#) sign.

```
1. CREATE TABLE #local temp table (
2. User id int,
3. Username varchar (50),
4. User address varchar (150)
5. )
```

Global Temp Variable

Global temp tables name starts with double hash (##). Once this table is created, it is like a permanent table. It is always ready for all users and not deleted until the total connection is withdrawn.

```
1. CREATE TABLE ##new global temp table (
2. User id int,
3. User name varchar (50),
4. User address varchar (150)
5. )
```

SQL ALTER TABLE

The ALTER TABLE statement is used to add, modify or delete columns in an existing table. It is also used to rename a table.

You can also use SQL ALTER TABLE command to add and drop various constraints on an existing table.

SQL ALTER TABLE Add Column

If you want to add columns in SQL table, the SQL alter table syntax is given below:

```
1. ALTER TABLE table_name ADD column_name column-definition;
```

If you want to add multiple columns in table, the SQL table will be

```
1. ALTER TABLE table_name
```

```
2. ADD (column_1 column-definition,
3.         column_2 column-definition,
4.         .....
5.         column_n column-definition);
```

SQL ALTER TABLE Modify Column

If you want to modify an existing column in SQL table, syntax is given below:

```
1. ALTER TABLE table_name MODIFY column_name column_type;
```

If you want to modify multiple columns in table, the SQL table will be

```
1. ALTER TABLE table_name
2. MODIFY (column_1 column_type,
3.                 column_2 column_type,
4.                .....
5.                 column_n column_type);
```

SQL ALTER TABLE DROP Column

The syntax of alter table drop column is given below:

```
1. ALTER TABLE table_name DROP COLUMN column_name;
```

SQL ALTER TABLE RENAME Column

The syntax of alter table rename column is given below:

```
1. ALTER TABLE table_name
2. RENAME COLUMN old_name to new_name;
```

Data manipulation language

Definition:

1. Data Manipulation Language (DML) statements are used for managing data in database.
2. DML commands are not auto-committed.
3. It means changes made by DML command are not permanent to database, it can be rolled back.

Commands in dml:

1. INSERT
2. UPDATE
3. DELETE

INSERT

It inserts a record to a table.

Syntax:

Insert into table_Name

Values(v1,v2,v3…)

Or

Insert into table_Name (column1,column2,…)

Values (v1,v1,….)

```
SQL>  INSERT INTO products
  2   values (1001, 'CAMERA' , 10, 'Digital') ;

1 row created.

SQL> INSERT INTO products
  2  values (1002, 'Laptop', 23, 'Dell') ;

1 row created.
```

This is how we insert values into a table. All characters and alpha-numeric characters(ex – 10023sdf78) must be enclosed in single quotes (‘ ‘) and each value must be separated by comma. Also we must be careful in entering the data without violating the primary key, foreign key , unique constraints.

Now let us see the table in which the data in has been inserted,

```
SQL> select * from products ;

    PRODID PRODNAME   QTY_AVAILABLE DESCRIPTION
---------- ---------- ------------- --------------------
      1001 CAMERA                10 Digital
      1002 Laptop                23 Dell
```

Now, let us insert data into the table **orders** in which a foreign key is referencing primary key,

```
SQL> INSERT INTO orders
  2  values (1001, 9001, 2, 9867.1, sysdate ) ;

1 row created.
```

Here, we see that 1001 is the same prodid as of the earlier table. Sysdate – it displays the current date set in the system .

```
SQL> INSERT INTO orders
  2  values (1002, 9023, 2, 98756.23, ' 02 - Oct - 2010 ' ) ;

1 row created.
```

Now, let us see the table,

```
SQL> select * from orders ;

    PRODID     ORDERID   QTY_SOLD       PRICE ORDER_DT
---------- ---------- ---------- ---------- ---------
      1001       9001          2     9867.1 06-APR-11
      1002       9023          2   98756.23 02-OCT-10
```

Another way of inserting data into the table is shown below,

```
SQL> INSERT INTO orders (prodid, orderid, qty_sold, price, order_dt)
  2  values (1002, 99, 7, 23678.9, '02 - Oct - 1987' ) ;

1 row created.
```

Now, let us see the table,

```
SQL> select * from orders ;

    PRODID     ORDERID   QTY_SOLD       PRICE ORDER_DT
---------- ---------- ---------- ---------- ---------
      1001       9001          2     9867.1 06-APR-11
      1002       9023          2   98756.23 02-OCT-10
      1002         99          7    23678.9 02-OCT-87
```

<u>UPDATE</u> :-

Update command is used to update one or more rows of a table.

Syntax:

UPDATE *table-name* set column-name = value
where **condition**;

For ex – 1) Let us update salary by increasing it by Rs200 and also give commission of Rs100 where empno = 7369.

```
SQL> select * from 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 19-APR-87       3000                    20
      7839 KING       PRESIDENT            17-NOV-81       5000                    10
      7844 TURNER     SALESMAN        7698 08-SEP-81       1500          0         30
      7876 ADAMS      CLERK           7788 23-MAY-87       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

14 rows selected.
```

Now, let us **update** the said record as shown below,

```
SQL> update emp set sal = sal + 200, comm = 100 where empno = 7369 ;

1 row updated.
```

Let us verify if the record has been updated,

```
SQL> select * from emp ;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL        COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7369 SMITH      CLERK           7902 17-DEC-80       1000        100         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 19-APR-87       3000                    20
      7839 KING       PRESIDENT            17-NOV-81       5000                    10
      7844 TURNER     SALESMAN        7698 08-SEP-81       1500          0         30
      7876 ADAMS      CLERK           7788 23-MAY-87       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

14 rows selected.
```

Thus, the record(empno – 7369) has been updated.

DELETE

Delete command is used to delete data from a table.

Delete command can also be used with condition to delete a particular row.

Syntax:

DELETE from *table-name*

[Where Condition(s)];

```
SQL> select * from test ;

     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 19-APR-87       3000                    20
      7839 KING       PRESIDENT            17-NOV-81       5000                    10
      7844 TURNER     SALESMAN        7698 08-SEP-81       1500          0         30
      7876 ADAMS      CLERK           7788 23-MAY-87       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

14 rows selected.
```

Thus, we have created the table test.

```
SQL> delete from test where empno = 7934 ;

1 row deleted.
```

Thus 1 row, ‘miller’ has been deleted.

```
SQL> select * from test ;

     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 19-APR-87       3000                    20
      7839 KING       PRESIDENT             17-NOV-81       5000                    10
      7844 TURNER     SALESMAN         7698 08-SEP-81       1500          0         30
      7876 ADAMS      CLERK            7788 23-MAY-87       1100                    20
      7900 JAMES      CLERK            7698 03-DEC-81        950                    30
      7902 FORD       ANALYST          7566 03-DEC-81       3000                    20

13 rows selected.
```

Thus, the deletion has been confirmed.

Transaction control language

TCL

Any DML change on a table is not a permanent one.
We need to save the DML changes in order to make it permanent
We can also undo (ignore) the same DML changes on a table.

The DDL changes cannot be undone as they are implicitly saved.

ROLLBACK

It undoes the DML changes performed on a table.
Let us see in the below example how **rollback** works,

```
SQL> delete from emp ;

14 rows deleted.

SQL> select * from emp ;

no rows selected
```

Let us delete the employee table. When we perform **select** operation on emp, we can see that all the rows have been deleted.
We now perform the **rollback** operation,

```
SQL> rollback ;

Rollback complete.
```

Now let us perform the **select** operation,

```
SQL> select * from 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 19-APR-87       3000                    20
      7839 KING       PRESIDENT            17-NOV-81       5000                    10
      7844 TURNER     SALESMAN        7698 08-SEP-81       1500          0         30
      7876 ADAMS      CLERK           7788 23-MAY-87       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

14 rows selected.
```

Thus performing the **rollback** operation, we can retrieve all the records which had been deleted.

COMMIT

It saves the DML changes permanently to the database.

Committing after rollback & vice versa will not have any effect
Let us explain the above statement with an example,

```
SQL> select * from test ;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON

SQL> delete from test ;

4 rows deleted.

SQL> select * from test ;

no rows selected

SQL> rollback ;

Rollback complete.

SQL> commit  ;

Commit complete.

SQL> select * from test ;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

We can see that **commit** has no effect after **rollback** operation.

```
SQL> select * from test ;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON

SQL> delete from test ;

4 rows deleted.

SQL> commit ;

Commit complete.

SQL> rollback ;

Rollback complete.

SQL> select * from test ;

no rows selected
```

Thus, from above – we can see that **rollback** has no effect after **commit** operation.

During an abnormal exit – i.e, shutdown or if the SQL window is closed by mouse click – then all the DML's will be rolled back automatically.

During a normal exit – **exit ;** - all the DML's will be auto-committed – and there will be no rollback.

Ex – 1) INSERT
UPDATE
ALTER
DELETE
ROLLBACK

When we perform the following operations in the same order for a table – then INSERT, UPDATE will be committed – because ALTER is a DDL – and thus all the DML's above it will also be committed – because DDL operations cannot be undone.
Here – only DELETE will be rolled back because it's a DML.

2) INSERT
UPDATE
DELETE
ROLLBACK

Here, all are rolled back.

SAVEPOINT :

It is like a pointer (break-point) till where a DML will be rolled back.

Ex :-
Insert …
Save point x ;
Update …

Delete ..
Rollback to x ;
…
…

Here, only DELETE & UPDATE are rolled back.
INSERT is neither rolled back nor committed.

Data transaction Language(DTL)

SavePoint:

The SAVEPOINT statement names and marks the current point in the processing of a transaction.

With the ROLLBACK TO statement, savepoints undo parts of a transaction instead of the whole transaction

Ex:Rollback to savepoint

SQL | SEQUENCES

Sequence is a set of integers 1, 2, 3, … that are generated and supported by some database systems to produce unique values on demand.

A sequence is a user defined schema bound object that generates a sequence of numeric values.

Sequences are frequently used in many databases because many applications require each row in a table to contain a unique value and sequences provides an easy way to generate them.

The sequence of numeric values is generated in an ascending or descending order at defined intervals and can be configured to restart when exceeds max_value.

Syntax:

```
CREATE SEQUENCE sequence_name

START WITH initial_value

INCREMENT BY increment_value

MINVALUE minimum value

MAXVALUE maximum value

CYCLE|NOCYCLE ;
```

sequence_name: Name of the sequence.

initial_value: starting value from where the sequence starts.
Initial_value should be greater than or equal
to minimum value and less than equal to maximum value.

increment_value: Value by which sequence will increment itself.
Increment_value can be positive or negative.

minimum_value: Minimum value of the sequence.
maximum_value: Maximum value of the sequence.

cycle: When sequence reaches its set_limit
it starts from beginning.

nocycle: An exception will be thrown
if sequence exceeds its max_value.
Example

Following is the sequence query creating sequence in ascending order.

Example 1:
CREATE SEQUENCE sequence_1

start with 1

increment by 1

minvalue 0

maxvalue 100

cycle;

Above query will create a sequence named sequence_1.Sequence will start from 1 and will be incremented by 1 having maximum value 100. Sequence will repeat itself from start value after exceeding 100.

Example 2:

Following is the sequence query creating sequence in descending order.

CREATE SEQUENCE sequence_2

start with 100

increment by -1

minvalue 1

maxvalue 100

cycle;

Above query will create a sequence named sequence_2.Sequence will start from 100 and should be less than or equal to maximum value and will be incremented by -1 having minimum value 1.

Example to use sequence : create a table named students with columns as id and name.

CREATE TABLE students

(

ID number(10),

NAME char(20)

);

Now insert values into table

INSERT into students VALUES(sequence_1.nextval,'Ramesh');

INSERT into students VALUES(sequence_1.nextval,'Suresh');

where sequence_1.nextval will insert id’s in id column in a sequence as defined in sequence_1.

Output:

_____

| ID | NAME |

| 1 | Ramesh |

| 2 | Suresh |

This article is contributed by ARSHPREET SINGH. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

A transaction is a unit of work that is performed against a database. Transactions are units or sequences of work accomplished in a logical order, whether in a manual fashion by a user or automatically by some sort of a database program.

A transaction is the propagation of one or more changes to the database. For example, if you are creating a record or updating a record or deleting a record from the table, then you are performing a transaction on that table. It is important to control these transactions to ensure the data integrity and to handle database errors.

Practically, you will club many SQL queries into a group and you will execute all of them together as a part of a transaction.

Properties of Transactions

Transactions have the following four standard properties, usually referred to by the acronym **ACID**.

- **Atomicity** – ensures that all operations within the work unit are completed successfully. Otherwise, the transaction is aborted at the point of failure and all the previous operations are rolled back to their former state.
- **Consistency** – ensures that the database properly changes states upon a successfully committed transaction.
- **Isolation** – enables transactions to operate independently of and transparent to each other.
- **Durability** – ensures that the result or effect of a committed transaction persists in case of a system failure.

Transaction Control

The following commands are used to control transactions.

- **COMMIT** – to save the changes.
- **ROLLBACK** – to roll back the changes.
- **SAVEPOINT** – creates points within the groups of transactions in which to ROLLBACK.
- **SET TRANSACTION** – Places a name on a transaction.

Transactional Control Commands

Transactional control commands are only used with the **DML Commands** such as - INSERT, UPDATE and DELETE only. They cannot be used while creating tables or dropping them because these operations are automatically committed in the database.

The COMMIT Command

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database.

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database. The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command.

The syntax for the COMMIT command is as follows.

```
COMMIT;
```

Example

Consider the CUSTOMERS table having the following records –

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Following is an example which would delete those records from the table which have age = 25 and then COMMIT the changes in the database.

```
SQL> DELETE FROM CUSTOMERS

   WHERE AGE = 25;

SQL> COMMIT;
```

Thus, two rows from the table would be deleted and the SELECT statement would produce the following result.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

The ROLLBACK Command

The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database. This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

The syntax for a ROLLBACK command is as follows –

```
ROLLBACK;
```

Example

Consider the CUSTOMERS table having the following records –

```
+----+----------+-----+-----------+----------+

| ID | NAME     | AGE | ADDRESS   | SALARY   |

+----+----------+-----+-----------+----------+

|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |

|  2 | Khilan   |  25 | Delhi     |  1500.00 |

|  3 | kaushik  |  23 | Kota      |  2000.00 |

|  4 | Chaitali |  25 | Mumbai    |  6500.00 |

|  5 | Hardik   |  27 | Bhopal    |  8500.00 |

|  6 | Komal    |  22 | MP        |  4500.00 |
```

```
|  7 | Muffy    |  24 | Indore    | 10000.00 |

+----+----------+-----+-----------+----------+
```

Following is an example, which would delete those records from the table which have the age = 25 and then ROLLBACK the changes in the database.

```
SQL> DELETE FROM CUSTOMERS

   WHERE AGE = 25;

SQL> ROLLBACK;
```

Thus, the delete operation would not impact the table and the SELECT statement would produce the following result.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

The SAVEPOINT Command

A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.

The syntax for a SAVEPOINT command is as shown below.

```
SAVEPOINT SAVEPOINT_NAME;
```

This command serves only in the creation of a SAVEPOINT among all the transactional statements. The ROLLBACK command is used to undo a group of transactions.

The syntax for rolling back to a SAVEPOINT is as shown below.

```
ROLLBACK TO SAVEPOINT_NAME;
```

Following is an example where you plan to delete the three different records from the CUSTOMERS table. You want to create a SAVEPOINT before each delete, so that you can ROLLBACK to any SAVEPOINT at any time to return the appropriate data to its original state.

Example

Consider the CUSTOMERS table having the following records.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following code block contains the series of operations.

```
SQL> SAVEPOINT SP1;

Savepoint created.

SQL> DELETE FROM CUSTOMERS WHERE ID=1;

1 row deleted.

SQL> SAVEPOINT SP2;

Savepoint created.

SQL> DELETE FROM CUSTOMERS WHERE ID=2;

1 row deleted.

SQL> SAVEPOINT SP3;

Savepoint created.

SQL> DELETE FROM CUSTOMERS WHERE ID=3;

1 row deleted.
```

Now that the three deletions have taken place, let us assume that you have changed your mind and decided to ROLLBACK to the SAVEPOINT that you identified as SP2. Because SP2 was created after the first deletion, the last two deletions are undone –

```
SQL> ROLLBACK TO SP2;

Rollback complete.
```

Notice that only the first deletion took place since you rolled back to SP2.

```
SQL> SELECT * FROM CUSTOMERS;

+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+

6 rows selected.
```

The RELEASE SAVEPOINT Command

The RELEASE SAVEPOINT command is used to remove a SAVEPOINT that you have created.

The syntax for a RELEASE SAVEPOINT command is as follows.

```
RELEASE SAVEPOINT SAVEPOINT_NAME;
```

Once a SAVEPOINT has been released, you can no longer use the ROLLBACK command to undo transactions performed since the last SAVEPOINT.

The SET TRANSACTION Command

The SET TRANSACTION command can be used to initiate a database transaction. This command is used to specify characteristics for the transaction that follows. For example, you can specify a transaction to be read only or read write.

The syntax for a SET TRANSACTION command is as follows.

```
SET TRANSACTION [ READ WRITE | READ ONLY ];
```

Data Control language(DCL)

DCL defines two commands,

Grant : Gives user access privileges to database.

Revoke : Take back permissions from user.

To Allow a User to create Table

grant create table to username;

To Grant all privilege to a User

grant sysdba to username;

To Grant permission to Drop any Table

grant drop any table to username;

GRANT SELECT, INSERT, UPDATE, DELETE ON tablename TO username;

grant ALL on emp to hr;

Login as HR

Execute the Bellow query

select * from scott.emp;

Revoke:

REVOKE SELECT, INSERT, UPDATE, DELETE ON tablename FROM username;

Login as Scott

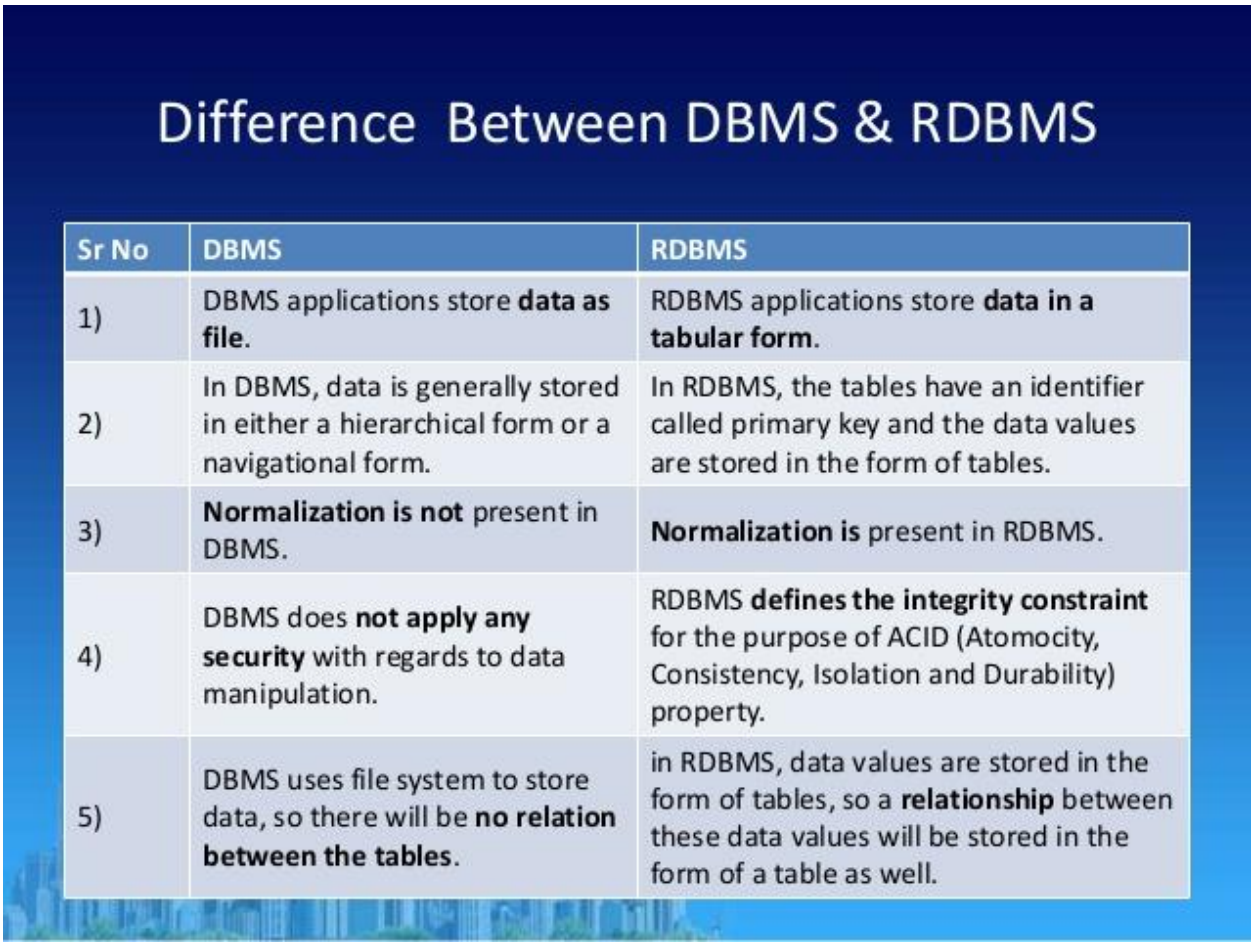
revoke select on emp from hr;

Login as HR

select * from scott.emp;

DIFFERENTIATION

No.	DELETE	TRUNCATE
1)	DELETE is a DML command.	TRUNCATE is a DDL command.
2)	We can use WHERE clause in DELETE command.	We cannot use WHERE clause with TRUNCATE
3)	DELETE statement is used to delete a row from a table	TRUNCATE statement is used to remove all the rows from a table.
4)	DELETE is slower than TRUNCATE statement.	TRUNCATE statement is faster than DELETE statement.



Comparison between DDL and DML in Tabular Form:

	DDL	**DML**
Full Form	Data Definition Language	Data Manipulation Language
Type of	Database Language	Database Language
Subset of	Structured Query Language (SQL)	Structured Query Language (SQL)

Uses	Used for defining the data structures, especially database schemas	Used for data manipulation of data structures
Types		
Related to	Defining data	Inserting, deleting, reviving, and modifying data
Commands	CREATE - to create objects in the database ALTER - alters the structure of the database DROP - delete objects from the database TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed COMMENT - add comments to the data dictionary RENAME - rename an object	SELECT - retrieve data from the a database INSERT - insert data into a table UPDATE - updates existing data within a table DELETE - deletes all records from a table, the space for the records remain MERGE - UPSERT operation (insert or update) CALL - call a PL/SQL or Java subprogram EXPLAIN PLAN - explain access path to data LOCK TABLE - control concurrency

	DELETE	TRUNCATE	DROP
Purpose	Deletes some or all rows of a	Deletes all rows of a table	Removes all rows and also

	table		the table definition, including indexes, triggers, grants, storage parameters
Command Type	DML	DDL	DDL
Space Usage and Release	Uses UNDO space. Released blocks that go to the freelist for the table, to be used for subsequent inserts/updates. Does not deallocate space.	Does not use UNDO space. Deallocates all space used by the table except MINEXTENTS.	Does not use UNDO space. Unless the PURGE clause is specified, does not result in space being released.
Commit required?	Yes	No	No
Undo possible?	Uncommitted deletes can be rolled back	Cannot be rolled back – once truncated, gone forever	A dropped table can be reinstated from the recycle bin *(more on this in a future article)*
Selective deletion possible?	Yes. Filter criteria be specified via WHERE clause	No filter criteria allowed, removes all rows	No filter criteria allowed, removes all rows
Triggers fired?	Yes, DELETE triggers fired	No triggers fired	No triggers fired
What if foreign keys (FKs) based on the table exist?	Can delete data even if FKs are enabled, provided the data violates no FK constraint	Cannot delete data if FKs are enabled; FKs need to be disabled/dropped. Exception: Truncate is possible if the FK is self-referential.	Can drop the table with the CASCADE CONSTRAINTS option. This will also remove the associated FKs

Efficiency	DELETE can be slow especially if the table has many triggers, indexes,and other dependencies	TRUNCATE is most efficient for deleting all rows, even more than dropping and recreating the table using DROP	DROP may not be as efficient as TRUNCATE, as dropping and re-creating the table requires you to re-grant object privileges, re-create indexes, constraints, etc.
Privileges required to issue the command	DELETE privilege. DELETE ANY TABLE allows you to delete rows from *any*table of any schema.	DROP ANY TABLE system privilege.	DROP ANY TABLE system privilege.
Grants	DELETE privilege on a specific table can be granted to another user or role.	TRUNCATE privilege on a specific table cannot be granted to another user or role.	DROP ANY privilege on a specific table cannot be granted to another user or role.
Can work outside the user's schema?	Yes, as long as the user has the DELETE privilege on the object.	No. A table can be truncated in one's own schema only.	No. A table can be dropped in one's own schema only.
Can work on a table that is part of a cluster?	Yes	No. You will have to truncate the whole cluster, or use either DELETE or DROP.	Yes

Operators

Operators:

- Operators are **nothing but symbols** which is used to **perform** some specific **operation**.

Operands:

- Operands are the **inputs required for the operation.**

Operators are classified into,

1. **Arithmetic Operators** (+, - , * , /)
2. **Relational Operators** (> , < , >= , <= , = , < > **or** != - not equals to)
3. **Logical Operators** (NOT, AND, OR)
4. **Special Operators** (IN , LIKE , BETWEEN , IS)

Rules for precedence

Order Evaluated	Operator
1	Arithmetic Operator
2	Concatenation
3	Comparison
4	Is, Like, In
5	Between
6	Not
7	AND
8	OR

SPECIAL OPERATORS

1) IN – it is used for **evaluating multiple values.**

Ex – 1) List the employees in dept 10 & 20

```
SQL> select * from emp where deptno in (10 , 20 ) ;

     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
      7782 CLARK      MANAGER         7839 09-JUN-81       2450                    10
      7788 SCOTT      ANALYST         7566 19-APR-87       3000                    20
      7839 KING       PRESIDENT            17-NOV-81       5000                    10
      7876 ADAMS      CLERK           7788 23-MAY-87       1100                    20
      7902 FORD       ANALYST         7566 03-DEC-81       3000                    20
      7934 MILLER     CLERK           7782 23-JAN-82       1300                    10

8 rows selected.
```

2) LIKE – used for pattern matching

% (percentage) - matches 0 or ‘n’ characters
_ (underscore) - matches exactly one character

Ex – 1) List all the employees whose name starts with ‘S’

```
SQL> select * from emp where ename like 'S%' ;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7369 SMITH      CLERK           7902 17-DEC-80        800                    20
      7788 SCOTT      ANALYST         7566 19-APR-87       3000                    20
```

Whenever we use % or _ , always ensure that it is preceded by the word **‘like’**

BETWEEN:

1. It is used to **evaluate the range of value**

Syntax:

Select * from table_name where columnname between value1 and value2;

IS:

1. It is used to **compare null value.**

Syntax:

Select * from table_name where columnname is null;

LOGICAL OPERATORS

1) List all the salesmen in dept 30

```
SQL> select * from emp where job = 'SALESMAN' and deptno = 30 ;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7499 ALLEN      SALESMAN         7698 20-FEB-81       1600        300         30
      7521 WARD       SALESMAN         7698 22-FEB-81       1250        500         30
      7654 MARTIN     SALESMAN         7698 28-SEP-81       1250       1400         30
      7844 TURNER     SALESMAN         7698 08-SEP-81       1500          0         30
```

2) List all the salesmen in dept number 30 and having salary greater than 1500

```
SQL> select * from emp
  2  where job = 'SALESMAN'
  3  and deptno = 30
  4  and sal > 1500 ;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7499 ALLEN      SALESMAN        7698 20-FEB-81       1600        300         30
```

3) List all the employees whose name starts with ‘s’ or ‘a’

```
SQL> select * from emp
  2  where ename like 'S%' or ename like 'A%' ;

     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
      7788 SCOTT      ANALYST         7566 19-APR-87       3000                    20
      7876 ADAMS      CLERK           7788 23-MAY-87       1100                    20
```

Order by:

1. Order by class is used to **sort the data ascending or descending order.**
2. Order by class should be **last statement in query.**
3. If we use **asc keyword** in order by class then data will be sorted in ascending order.
4. If we use **desc keyword** in order by class then data will be sorted in descending order.
5. If we **won’t provide asc/desc** keyword in order by class the by default data will be sorted in **ascending order.**

Syntax:

```
Select column_list
From table_name
Order by column_name asc/desc;
```

Ex:

1. **Arrange ename, sal, job, empno and sort by descending order of salary**

```
SQL> select ename, sal, job, empno
  2  from emp
  3  order by 2 desc ;

ENAME              SAL JOB              EMPNO
---------- ---------- --------- ----------
KING             5000 PRESIDENT         7839
FORD             3000 ANALYST           7902
SCOTT            3000 ANALYST           7788
JONES            2975 MANAGER           7566
BLAKE            2850 MANAGER           7698
CLARK            2450 MANAGER           7782
ALLEN            1600 SALESMAN          7499
TURNER           1500 SALESMAN          7844
MILLER           1300 CLERK             7934
WARD             1250 SALESMAN          7521
MARTIN           1250 SALESMAN          7654
ADAMS            1100 CLERK             7876
JAMES             950 CLERK             7900
SMITH             800 CLERK             7369

14 rows selected.
```

In the above query we have – **order by 2** – thus it arranges only the 2nd column ‘salary’ in the descending order.
Thus to arrange the specific columns in order – we must have to specify the column number.

SQL | ALL and ANY

ALL & ANY are logical operators in SQL. They return boolean value as a result.

ALL

ALL operator is used to select all tuples of SELECT STATEMENT. It is also used to compare a value to every value in another value set or result from a subquery.

The ALL operator returns TRUE iff all of the subqueries values meet the condition. The ALL must be preceded by comparison operators and evaluates true if all of the subqueries values meet the condition.

ALL is used with SELECT, WHERE, HAVING statement.

ALL with SELECT Statement:

Syntax:

SELECT ALL field_name

FROM table_name

WHERE condition(s);

ALL with WHERE or HAVING Statement:

Syntax:

SELECT column_name(s)

FROM table_name

WHERE column_name comparison_operator ALL

(SELECT column_name

FROM table_name

WHERE condition(s));

Example:

Consider the following Products Table and OrderDetails Table,

Products Table

OrderDetails Table

Queries

Find the name of the all the product.

SELECT ALL ProductName

FROM Products

WHERE TRUE;

Output:

Find the name of the product if all the records in the OrderDetails has Quantity either equal to 6 or 2.

```
SELECT ProductName

FROM Products

WHERE ProductID = ALL (SELECT ProductId

                FROM OrderDetails

                WHERE Quantity = 6 OR Quantity = 2);
```

Output:

Find the OrderID whose maximum Quantity among all product of that OrderID is greater than average quantity of all OrderID.

```
SELECT OrderID

FROM OrderDetails

GROUP BY OrderID

HAVING max(Quantity) > ALL (SELECT avg(Quantity)

                FROM OrderDetails

                GROUP BY OrderID);
```

Output:

ANY

ANY compares a value to each value in a list or results from a query and evaluates to true if the result of an inner query contains at least one row.

ANY return true if any of the subqueries values meet the condition.

ANY must be preceded by comparison operators.

Syntax:

SELECT column_name(s)

FROM table_name

WHERE column_name comparison_operator ANY

(SELECT column_name

FROM table_name

WHERE condition(s));

Queries

Find the Distinct CategoryID of the products which have any record in OrderDetails Table.

```
SELECT DISTINCT CategoryID
FROM Products
WHERE ProductID = ANY (SELECT ProductID
                       FROM OrderDetails);
```

Output:

Finds any records in the OrderDetails table that Quantity = 9.

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY (SELECT ProductID
                       FROM OrderDetails
                       WHERE Quantity = 9);
```

This article is contributed by Anuj Chauhan. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to

contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

FUNCTION

1. Functions are **nothing but reusable program** which returns some output.
2. In order to call the function we sure pass input (inputs maybe called as for parameters or arguments).

Types of function:

1. Build in function (pre-defined function)
2. User defined function

Functions are very powerful **features** of SQL and can be used to do the following

- Perform Calculations on data
- Modify individual data items
- Manipulate output for group of rows
- Format dates and number for display
- Converts column data types

User defined in function:

1. It used in plsql and sql.
2. Plsql is **an extension of sql** for further study of sql.
3. We can **use control statement looping and oops concept.**

Build in function:

1. It is also called as **pre-defined function**, where functions are already exist. We are **using that function just by calling.**

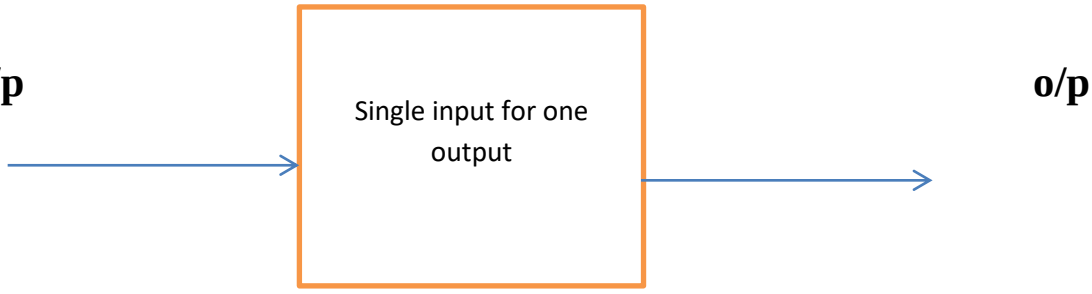
Types of build in function:

1. Single row function
2. Multi row function

Single row functions:

1. Manipulate data items
2. Accept arguments and return one value
3. Act on each row returned
4. Can be nested
5. Accept the arguments which can be column or expression

EX:



Types of single row function:

1. Character function
2. Number function
3. Date function
4. General function
5. Conversion function

Character function:

1. Case manipulation function
2. Character manipulation function

Types of case manipulation function:

1. Upper
2. Lower
3. Inticap

Types of character manipulation function:

1. Concate
2. Substring
3. Instr
4. Length
5. Reverse
6. Replace
7. Trim

Types of number function:

1. Round
2. Trunc
3. Mod
4. Power
5. Sqrt
6. Abs

Types of conversion function:

1. Implicit conversion function
2. Explicit conversion function

Explicit Conversion function:

1. TO_NUMBER
2. TO_CHAR
3. TO_DATE

Types of date function:

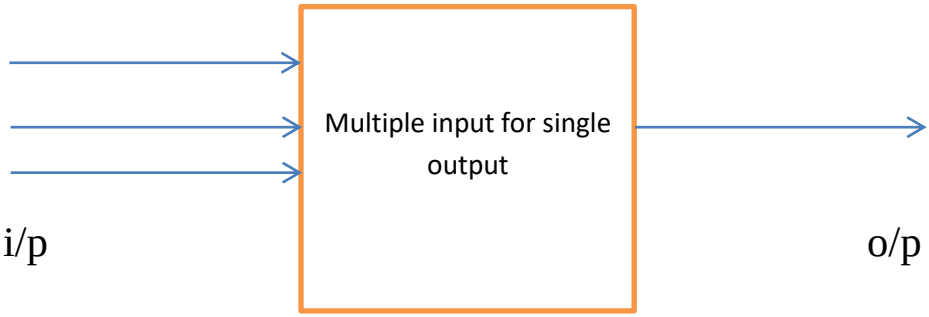
1. Sysdate
2. Currentdate
3. Systimestamp
4. Last_day
5. Add_months
6. Months_between

Types of general function:

1. Nvl
2. Nvl2

Multi row function:

1. It returns one output for multiple input.



Multi row function:

1. Max
2. Min
3. Avg
4. Sum
5. count

Character function

Case manipulation function:

1. This function can act as only in letters.

1.Upper:

1. It is used **to convert** all the letters to **upper case.**
2. If there are characters in the string that are not letters, they are **unaffected by this function.**

Syntax:

1. Select upper(‘mahe’) from dual;
2. Select upper(‘mahe’),upper(‘usha’) from dual;
3. Select * from emp where ename=upper(‘allen’);

2.Lower:

1. Lower function is used to convert all the letters to lower case.

Syntax:

1. Select lower(‘mahe’) from dual;
2. Select lower(‘mahe’),lower(‘usha’) from dual;
3. Select * from emp where ename=lower(‘allen’);

3.Initcap:

1. It is used to convert first letter of all the word in a string to uppercase and rest of the words to lower case.

Syntax:

1. Select initcap(‘mahe’) from dual;
2. Select initcap(‘mahe’),initcap(‘usha’) from dual;
3. Select * from emp where ename=initcap(‘allen’);

Character manipulation function:

1. This function can act as only in letters.
2. But it has more pre-defined method than case manipulation function.

1.Concat:

- CONCAT function **allows you to concatenate two strings together.**
- Concat functions **accept only two arguments.**

SYNTAX:

CONCAT(string1, string2)

Ex:

SQL> select ename,job, concat(ename,job) from emp;

Concatenate Single Quotes

SELECT CONCAT('Let''s', ' learn Oracle') FROM dual;

Syntax:

```
SQL> select concat('hi',' bye')
   from dual;
      • select concat('hi','hello','bye')   from dual
SQL> /
select concat('hi','hello','bye')
       *
ERROR at line 1:
ORA-00909: invalid number of arguments
```

2.Length:

1. It is used to **find the length.**

Syntax:

Select ename,length(ename) from emp;

3.Reverse:

1. It is **used to reverse the even string.**

Syntax:

Select ename,reverse(ename) from emp;

4.Replace:

1. It is **used to replace the string to be replaced.**

Syntax:

Replace(String,String-to-be-replaced,replacement-string)

SQL> select replace('jspider','j','q') from dual;

REPLACE

Qspider

SQL> select replace('jspiders','s','m') from dual;

REPLACE(

Jmpiderm

SQL> select replace('jspiders','sp','n') from dual;

REPLACE

Jniders

SQL> select replace('jspiders','sp','mahe') as result from dual;

RESULT

Jmaheiders

5.Substring:

1. It is **used to extract substring in the given string.**
2. Parameters or Arguments

3. String the source string.
4. Start position the starting position for extraction.
5. The **first position in the string is always 1.**
6. Length Optional. It is the number of characters to extract. If this parameter is omitted, the SUBSTR function will return the entire string where position and length are both integers.

Syntax:

Substr(string,start_position,length);

Note:

If starting position is don't know then we did not perform substring operation. But we can extract that using instr function.

6.Trim:

1. It is used to remove some spaces or character in a given string.

Types:

1. LTRIM
2. RTRIM
3. TRIM

LTRIM:

1. It is **used to remove some spaces or a character** in a given string from **left hand side.**

RTRIM:

1. It is **used to remove some spaces or a character** in a given string from **right hand side.**

TRIM:

1. It is **used to remove some spaces or a character** in a given string from **both sides.**

SYNTAX:

TRIM(LEADING/TRAILING/BOTH TRIM_CHARACTER FROM STRING);

Leading-ltrim

Trailing-rtrim

Both-trim both side

SQL> select trim(both 'ee' from 'eewelcome') from dual;

select trim(both 'ee' from 'eewelcome') from dual

*

ERROR at line 1:

ORA-30001: trim set should have only one character

Note:

In trim we cannot have more than one character as a trim character.

7.INSTR:

1. Instring is **used to find the position of the substring** in a given string.

2. If substring is not present in the given string then **it returns zero [0].**

Syntax:

INSTR(STRING,SUB_STRING,START_POSITION,LENGTH(nth occurance))

NUMBER FUNCTION

Round:

1. It is used to **round the value from nearest digit.**

EX:

- Select round(55.4) from dual;

o/p:

55

- Select round(55.5) from dual;

o/p:

56

Trunc:

1. It is used to **remove all the specified decimal values.**

EX:

- Select trunc(55.44444) from dual;

o/p:

55

- Select trunc(55.5) from dual;

o/p:

55

Mod:

1. It is **used to find remainder**

EX:

- Select mod(50,2) from dual;

o/p:

0

- Select mod(55,2) from dual;

o/p:

1

Power:

1. It is used to **find the power value of particular value.**

EX:

- Select power(4,2) from dual;

o/p:

16

- Select power(5,3) from dual;

o/p:

125

Sqrt:

1. It is **used to find the square root of particular value.**

EX:

- Select sqrt(25) from dual;

o/p:

5

- Select sqrt(4) from dual;

o/p:

2

Abs:

1. It is used to **find absolute value.**

EX:

- Select abs(-55) from dual;

o/p:

55

- Select round(55.5) from dual;

o/p:

55.5

Conversion function

- It is used to **convert one data type to another data type.**

Types of conversion function:

1. Implicit conversion function
2. Explicit conversion function

Implicit conversion function:

1. It is used to convert **convert one data type to another data type without using conversion function.**

Explicit conversion function

1. It is used to convert **convert one data type to another data type with using conversion function.**

Conversion function:

1. TO_NUMBER
2. TO_CHAR
3. TO_DATE

Date function

Working with Dates:

- ✓ Oracle database stores dates in an internal numeric format:

Century, Year, Month, day, hours, minutes, Seconds.

- ✓ The default date display format is DD-MON-YY

Ex: Select ename, hiredate from emp

Sysdate:

1. It is used to show our system date.

Ex:

- Select sysdate from dual;

Currentdate:

1. It is used to show our system date.

EX:

- Select currentdate from dual;

Systimestamp:

1. It is a function that returns date, time including milliseconds and time zone

EX:

- Select systimestamp from dual;

Arithmetic With dates:

1. Add or subtract a number to or from a date for a resultant date value
2. Subtract two dates to find the number of days between those date.

Ex :

To get future or past date

Select sysdate+100 , sysdate-70 from dual;

Ex:

Experience with years;

Select ename,hiredate, round((sysdate-hiredate)/365) from emp;

ADD_MONTHS(date, n) :

1. **Adds the specific number of months (n) to a date**. The ‘n’ can be both negative and positive:

EX:

Select add_months(sysdate, -1) as prev_month , sysdate, add_months (sysdate, 1) as next_month from dual;

LAST_DAY(date):

1. It **returns the last day in the month of the specified date.**

EX:

```
select sysdate, last_day(sysdate) as last_day_curr_month,
last_day(sysdate) + 1 as first_day_next_month from dual;
```

The number of days until the end of the month.

```
select last_day(sysdate) - sysdate as days_left
from dual
```

MONTHS_BETWEEN(date, date) :

1. It calculates the **number of months between two dates.**

Ex:

```
select MONTHS_BETWEEN ('31-MAR-2016', '28-FEB-2015')
from dual;
```

Let's select the number of months an employee has worked for the company.

Select months_between (sysdate, HIREDATE)

from emp;

Elements of date format:

1. YYYY- It is used to **display full year in number.**
2. YEAR- year **spelled out.**
3. MM- **two digit value for month.**
4. MONTH- it will display **full name of the month.**
5. MON- it will display **three letter of the month.**
6. DY- it will display **three letter of the day.**
7. DAY- **full name of the day.**
8. DD – it will display **numeric day of the month.**

General function

NVL:

1. Nvl function **have two arguments**

Syntax:

NVL(arg 1,arg 2)

Rules for NVL:

1. First it will look for **arg 1** if it **is null then it will return arg 2.**
2. If the **arg 1 is not null** it will **return arg 1 itself.**

Converts a null to actual values:

Ex:

1.select ename,sal,comm,nvl(comm,0) from emp

2. SQL> select ename,sal,comm,sal+comm, sal+nvl(comm,0) totalsalary from emp;

NVL2:

1. It will take **three arguments.**

Syntax:

Nvl2(arg 1,arg 2,arg 3);

Rules for NVL2:

1. First it will look for **arg 1** if it **is null** then **it will return arg 3.**
2. If the **arg 1 is not null** then **it will return arg 2.**

Ex: SELECT ENAME,SAL, NVL2(COMM,COMM+SAL,SAL) "TOTAL SALARY" FROM EMP;

Group function/multi row function

Group Functions operate on **group of rows to give one row result** per group.

We have **5 GROUP** functions,

1. Sum
2. Max
3. Min
4. Avg
5. Count

Sum :

It returns **total value**

Max:

It returns **maximum value**

Min:

It returns **minimum value**

Avg:

It returns **average value**

Count:

It returns **number of records**

Ex – 1) display the maximum salary, minimum salary and total salary from employee

```
SQL> select max(sal), min(sal), sum(sal) from emp;

  MAX(SAL)   MIN(SAL)   SUM(SAL)
---------- ---------- ----------
      5000        800      29025
```

2) The below query gives the total number of employees

```
SQL> select count(*), count(empno)
  2  from emp ;

  COUNT(*) COUNT(EMPNO)
---------- ------------
        14           14
```

GROUP BY CLAUSE

DEFINITION:

1. Group by clause is used in select statement **to collect data across multiple records and group the result by one or more column.**
2. Grouping is done by using **'group by'** clause.

Syntax:

SELECT <COLUMN_LIST><AGGREGATE_FUNCTION>

FROM TABLE_NAME

WHERE <CONDITION/S>

GROUP BY <COLUMN_LIST>

HAVING <CONDITION_BASED_ON_GROUP BY>;

RULES:

1. Whatever the column present in the select statement that should be present in group by clause
2. Whatever the column present in the group by clause need not to be present in select statement
3. Where condition is used to check the condition before grouping the data.it is used to filter non group data
4. We cannot use aggregate function
5. Having clause is used to restrict group data and it should be used after group by clause.

EX:

1) Display the maximum of each job

```
SQL> select job, max(sal)
  2  from emp
  3  group by job ;

JOB         MAX(SAL)
--------- ----------
CLERK           1300
SALESMAN        1600
PRESIDENT       5000
MANAGER         2975
ANALYST         3000
```

HAVING

1. **'Having'** is used to filter the grouped data.
2. **'Where'** is used to filter the non grouped data.
3. '**Having'** should be used after **group by** clause
4. **'Where'** should be used before **group by** clause

EX:

1) Display job-wise highest salary only if the highest salary is more than Rs1500

```
SQL> select job, max(sal)
  2  from emp
  3  group by job
  4  having max(sal) > 1500 ;

JOB         MAX(SAL)
--------- ----------
SALESMAN        1600
PRESIDENT       5000
MANAGER         2975
ANALYST         3000
```

RESTRICTIONS ON GROUPING

1. We can select only the columns that are part of '**group by**' statement.
2. If we try selecting other columns, we will get an error as shown below,

```
SQL> select deptno, job, sum(sal), sum(comm)
  2  from emp
  3  group by deptno ;
select deptno, job, sum(sal), sum(comm)
               *
ERROR at line 1:
ORA-00979: not a GROUP BY expression
```

The above query is an error because '**job**' is there in the **select** query but not in the **group by** query.
If it is enclosed in any of the **group functions like sum(sal)** etc – then it is not an error. But whatever table is included in the **select** query must also be included in the **group by** query.

The above problem can be overcome with the following query as shown below,

```
SQL>  select deptno, job, sum(sal), sum(comm)
  2   from emp
  3   group by deptno, job ;

    DEPTNO JOB         SUM(SAL)  SUM(COMM)
---------- --------- ---------- ----------
        20 CLERK           1900
        30 SALESMAN        5600       2200
        20 MANAGER         2975
        30 CLERK            950
        10 PRESIDENT       5000
        30 MANAGER         2850
        10 CLERK           1300
        10 MANAGER         2450
        20 ANALYST         6000

9 rows selected.
```

The below query is also correct to rectify the above error,

```
  1  select deptno, sum(sal), sum(comm)
  2  from emp
  3  group by deptno, job
  4* order by deptno
SQL> /

    DEPTNO   SUM(SAL)  SUM(COMM)
---------- ---------- ----------
        10       1300
        10       2450
        10       5000
        20       6000
        20       1900
        20       2975
        30        950
        30       2850
        30       5600       2200

9 rows selected.
```

Whatever is there in the **select** statement must be there in the **group by** statement. But, whatever is there in the **group by** statement need not be present in the **select** statement. This is shown in the above two corrected queries.

1).Display the maximum salary for each of the job excluding all the employees whose name ends with 'S'

```
SQL>  select ename, job, min(sal)
  2   from emp
  3   where ename not like '%S'
  4   group by ename, job
  5  order by 3 ;

ENAME      JOB         MIN(SAL)
---------- --------- ----------
SMITH      CLERK            800
MARTIN     SALESMAN        1250
WARD       SALESMAN        1250
MILLER     CLERK           1300
TURNER     SALESMAN        1500
ALLEN      SALESMAN        1600
CLARK      MANAGER         2450
BLAKE      MANAGER         2850
FORD       ANALYST         3000
SCOTT      ANALYST         3000
KING       PRESIDENT       5000

11 rows selected.
```

SUB QUERY

Definition:

1. **Query with in a query** is called sub query.
2. The **inner most sub query is executed first.**
3. We can write **maximum 255 query in a single sub query.**
4. It is **also called as nested query.**
5. Each queries are **separated by () parenthesis.**
6. It is used to **find unknown value.**
7. The output of **inner query** is passed as input to the **outer query.**

Syntax of a sub-query

[OUTER QUERY]

Select …

From …

Where … (Select …

From …

Where …

)

[INNER QUERY]

Types of sub query:

1. Single row sub query
2. Multi row sub query

Single row sub query:

1. It is used to return only one output.
2. In this case in between the outer and inner queries we can use = operator.

Ex:

List the employees working in 'Research' department.

For ex :-

1) List the employees working in 'Research' department.

```
SQL> select * from emp
  2  where deptno = (select deptno
  3                  from dept
  4                  where dname = 'RESEARCH'
  5                  ) ;

     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 19-APR-87       3000                    20
      7876 ADAMS      CLERK           7788 23-MAY-87       1100                    20
      7902 FORD       ANALYST         7566 03-DEC-81       3000                    20
```

Multi row sub query:

1. It is used to **return more than one output.**

In this case **in between outer and inner queries we can use IN operator.**

Ex:

List the employees in Research and Sales department

```
SQL>  select * from emp
  2   where deptno IN (select deptno
  3                    from dept
  4                    where dname IN ('RESEARCH','SALES')
  5                    )
  6  order by deptno ;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7369 SMITH      CLERK           7902 17-DEC-80        800                    20
      7876 ADAMS      CLERK           7788 23-MAY-87       1100                    20
      7566 JONES      MANAGER         7839 02-APR-81       2975                    20
      7788 SCOTT      ANALYST         7566 19-APR-87       3000                    20
      7902 FORD       ANALYST         7566 03-DEC-81       3000                    20
      7900 JAMES      CLERK           7698 03-DEC-81        950                    30
      7844 TURNER     SALESMAN        7698 08-SEP-81       1500          0         30
      7654 MARTIN     SALESMAN        7698 28-SEP-81       1250       1400         30
      7521 WARD       SALESMAN        7698 22-FEB-81       1250        500         30
      7499 ALLEN      SALESMAN        7698 20-FEB-81       1600        300         30
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30

11 rows selected.
```

1) Display the 2nd maximum salary

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

  MAX(SAL)
----------
      3000
```

2) Display the 3rd maximum salary

```
SQL> select max(sal) from emp
  2  where sal < (select max(sal) from emp
  3  where sal < (select max(sal) from emp) ) ;

  MAX(SAL)
----------
      2975
```

3) Display the 4th least salary

```
SQL> select min(sal) from emp
  2  where sal > (select min(sal) from emp
  3  where sal > (select min(sal) from emp
  4  where sal > (select min(sal) from emp ) ) ) ;

  MIN(SAL)
----------
      1250
```

This method is not efficient to find the maximum and minimum salary. The limit is 32. This is not efficient if you want to find the 100th maximum salary.

We can have upto 32 levels of sub-queries only.

Co – related Queries :

→ They work on the principles of both **sub – queries & JOIN(s)**.
→ Sub query are dependent on the output of the outer query then we call that as a co-related sub query.
→ They are special type of sub – queries
→ Here, both outer & inner queries are inter-dependent
→ For each & every record of outer query, the entire inner query will be executed
→ The outer query output will be fast to the sub query and this sub query execute.
→ Then the output of the sub query pass to the outer query through outer query it displays the output for the user.

Note:

- Most of the time we use self join in co-related sub query.

For ex, Display the employee who is earning the highest salary

```
SQL> select * from emp A
  2  where 0 = (select count(distinct(B.sal)) from emp B
  3  where A.sal < B.sal ) ;
```

```
     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7839 KING       PRESIDENT            17-NOV-81       5000                    10
```

Thus, if an outer query column is being accessed inside the inner query, then that query is said to be co-related.

Let us see the logic i.e, how we get the 1st max salary :-

Emp (A)		
EmpNo	**Ename**	**Sal**
101	Scott	3000
102	Blake	4000
103	King	5000
104	Smith	2000
105	Jones	1000

Emp (B)		
EmpNo	**Ename**	**Sal**
101	Scott	3000
102	Blake	4000
103	King	5000
104	Smith	2000
105	Jones	1000

Since co-related queries are a combination of Joins and sub-queries.
It follows the concept of Joins and creates multiple copies of the same table.
Then it takes 1st record i.e, - Scott – sal is 3000. It starts comparing with the sal in the emp table,
3000 = 3000 - count starts from 0 – thus, 0 = 0
3000 < 4000 – thus, 0 ! = 1
3000 < 5000 – thus, 0 ! = 2
3000 > 2000 – thus , 0! = 4
3000 > 1000 – thus, 0 ! = 5 if the condition becomes false, then the count increments by 1. Here 3000 is less than 4000 & 5000, thus 0 ! = 2. Thus , Scott does not have the highest salary.

Similarly, it does for the next records,
Blake – salary of 4000 – but 4000 < 5000 – thus, 0 ! = 1. This is also false.
King – salary of 5000 – it is greater than everything – thus, 0 = 0. Thus, King has the highest salary.
But the query doesn't stop here, it checks for Smith & Jones as well.

Similarly, if we want to find the 2nd maximum salary,
Then in the query, change '0' to '1' & here, the logic is – it compares until it gets 1 = 1.
For 3rd maximum salary – change 0 to 2 and so on – here, the logic is – it compares until it gets 2 = 2.

For any highest, always put it as '0' in the query.

If you want n(th) salary, pass (n-1).

In interview – this is a definite question. They will ask you what is co-related queries. And then they'll ask you find, 1st or max or 3rd maximum salary – after you write the query – they will ask you to explain the logic as to how it gets the same – draw the table and explain it to them just as shown above.

Assignment

1) Display the least salary from the employee table.

```
SQL> select * from emp A
  2  where 0 = (select count(distinct(B.sal)) from emp B
  3  where A.sal > B.sal ) ;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7369 SMITH      CLERK           7902 17-DEC-80        800                    20
```

2) Display top 3 person's salaries from the employee table.

```
SQL> select * from emp A
  2  where 2 >= (select count(distinct(B.sal)) from emp B
  3  where A.sal < B.sal ) ;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7566 JONES      MANAGER         7839 02-APR-81       2975                    20
      7788 SCOTT      ANALYST         7566 19-APR-87       3000                    20
      7839 KING       PRESIDENT            17-NOV-81       5000                    10
      7902 FORD       ANALYST         7566 03-DEC-81       3000                    20
```

3) Write a query to display bottom 3 salaries

```
SQL> select * from emp A
  2  where 2 >= (select count(distinct(B.sal)) from emp B
  3  where A.sal > B.sal )
  4  order by sal asc ;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7369 SMITH      CLERK           7902 17-DEC-80        800                    20
      7900 JAMES      CLERK           7698 03-DEC-81        950                    30
      7876 ADAMS      CLERK           7788 23-MAY-87       1100                    20
```

4) Display 1st and 4th maximum salary

```
SQL> select * from emp A
  2  where 0 = (select count(distinct(B.sal)) from emp B
  3  where A.sal < B.sal )
  4  UNION
  5  select * from emp A
  6  where 3 = (select count(distinct(B.sal)) from emp B
  7  where A.sal < B.sal )
  8  /

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30
      7839 KING       PRESIDENT            17-NOV-81       5000                    10
```

5) Display 1st, 4th & 6th highest salaries in a single query

```
SQL> select * from emp A
  2  where 0 = (select count(distinct(B.sal)) from emp B
  3  where A.sal < B.sal )
  4  UNION
  5  select * from emp A
  6  where 3 = (select count(distinct(B.sal)) from emp B
  7  where A.sal < B.sal )
  8  UNION
  9  select * from emp A
 10  where 5 = (select count(distinct(B.sal)) from emp B
 11  where A.sal < B.sal )
 12  /

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7499 ALLEN      SALESMAN        7698 20-FEB-81       1600        300         30
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30
      7839 KING       PRESIDENT            17-NOV-81       5000                    10
```

Inline view

An **inline view** is a **SELECT** statement in the **FROM** clause. As mentioned in the **View** section, a view is a virtual table that has the characteristics of a table yet does not hold any actual data. In an inline view construct, instead of specifying table name(s) after the **FROM** keyword, the source of the data actually comes from the inline view.

Inline view is sometimes referred to as **derived table**. These two terms are used interchangeably.

Syntax

The syntax for an inline view is,

```
SELECT "column_name" FROM (Inline View);
```

Example

Assume we have two tables: The first table is ***User_Address***, which maps each user to a ZIP code; the second table is ***User_Score***, which records all the scores of each user. The question is, how to write a SQL query to find the number of users who scored higher than 200 for each ZIP code?

Without using an inline view, we can accomplish this in two steps:

Query 1

```
CREATE TABLE User_Higher_Than_200
SELECT User_ID, SUM(Score) FROM User_Score
GROUP BY User_ID
HAVING SUM(Score) > 200;
```

Query 2

```
SELECT a2.ZIP_CODE, COUNT(a1.User_ID)
FROM User_Higher_Than_200 a1, User_Address a2
WHERE a1.User_ID = a2.User_ID
GROUP BY a2.ZIP_CODE;
```

In the above code, we introduced a temporary table, ***User_Higher_Than_200***, to store the list of users who scored higher than 200. ***User_Higher_Than_200*** is then used to **join** to the ***User_Address*** table to get the final result.

We can simplify the above SQL using the inline view construct as follows:

Query 3

```
SELECT a2.ZIP_CODE, COUNT(a1.User_ID)
FROM
(SELECT User_ID, SUM(Score) FROM User_Score GROUP BY
User_ID HAVING SUM(Score) > 200) a1,
User_Address a2
WHERE a1.User_ID = a2.User_ID
GROUP BY a2.ZIP_CODE;
```

The code that is in **red** represents an inline view. There are two advantages on using inline view here:

1. We do not need to create the temporary table. This prevents the database from having too many objects, which is a good thing as each additional object in the database costs resources to manage.

2. We can use a single SQL query to accomplish what we want.

Notice that we treat the inline view exactly the same as we treat a table. Comparing Query 2 and Query 3, we see that the only difference is that we replace the temporary table name in Query 2 with the inline view statement in Query 3. Everything else stays the same.

ROWNUM:

For each row returned by a query, the ROWNUM pseudocolumn returns a number indicating the order in which Oracle selects the row from a table or set of joined rows. The first row selected has a ROWNUM of 1, the second has 2, and so on.

You can use ROWNUM to limit the number of rows returned by a query, as in this example:

```
SELECT * FROM employees WHERE ROWNUM < 10;
```

If an ORDER BY clause follows ROWNUM in the same query, then the rows will be reordered by the ORDER BY clause. The results can vary depending on the way the rows are accessed. For example, if the ORDER BY clause causes Oracle to use an index to access the data, then Oracle may retrieve the rows in a different order than without the index. Therefore, the following statement will not have the same effect as the preceding example:

```
SELECT * FROM employees WHERE ROWNUM < 11 ORDER BY last_name;
```

If you embed the ORDER BY clause in a subquery and place the ROWNUM condition in the top-level query, then you can force the

ROWNUM condition to be applied after the ordering of the rows. For example, the following query returns the employees with the 10 smallest employee numbers. This is sometimes referred to as top-N reporting:

```
SELECT * FROM
  (SELECT * FROM employees ORDER BY employee_id)
  WHERE ROWNUM < 11;
```

In the preceding example, the ROWNUM values are those of the top-level SELECT statement, so they are generated after the rows have already been ordered by employee_id in the subquery.

Conditions testing for ROWNUM values greater than a positive integer are always false. For example, this query returns no rows:

```
SELECT * FROM employees
   WHERE ROWNUM > 1;
```

The first row fetched is assigned a ROWNUM of 1 and makes the condition false. The second row to be fetched is now the first row and is also assigned a ROWNUM of 1 and makes the condition false. All rows subsequently fail to satisfy the condition, so no rows are returned.

You can also use ROWNUM to assign unique values to each row of a table, as in this example:

```
UPDATE my_table
   SET column1 = ROWNUM;
```

Joins

Joins are used when we need to **fetch the data from multiple tables.**

Types of Join(s):

1. Cartesian Join (product)
2. Inner (Equi) Join
3. Outer Join - Left Outer Join, Right Outer Join, Full Outer Join
4. Self Join

CARTESIAN JOIN

It is based on **Cartesian product theory.**

Cartesian Product Theory in Mathematics states that :-

Let there be two sets – A {1, 2, 3} & B {4, 5}

Thus the Cartesian product (A*B) will be,
A * B = { (1,4), (1,5), (2,4), (2,5), (3,4), (3,5) }

Thus there are 6 sets – order of A is 3 & order of B is 2.
Therefore, 2*3 = 6 is the Cartesian product.

Ex:

Display employee name along with the department name

Here, each and every record of the 1st table will combine with each and every record of the 2nd table.
If a table A is having 10 records & B is having 4 records – the Cartesian join will return 10*4 = 40 records.

```
SQL> select A.ename, A.sal, B.dname
  2  from emp A, dept B ;
```

ENAME	SAL	DNAME
SMITH	800	ACCOUNTING
ALLEN	1600	ACCOUNTING
WARD	1250	ACCOUNTING
JONES	2975	ACCOUNTING
MARTIN	1250	ACCOUNTING
BLAKE	2850	ACCOUNTING
CLARK	2450	ACCOUNTING
SCOTT	3000	ACCOUNTING
KING	5000	ACCOUNTING
TURNER	1500	ACCOUNTING
ADAMS	1100	ACCOUNTING
JAMES	950	ACCOUNTING
FORD	3000	ACCOUNTING
MILLER	1300	ACCOUNTING
SMITH	800	RESEARCH
ALLEN	1600	RESEARCH
WARD	1250	RESEARCH
JONES	2975	RESEARCH
MARTIN	1250	RESEARCH
BLAKE	2850	RESEARCH
CLARK	2450	RESEARCH
SCOTT	3000	RESEARCH
KING	5000	RESEARCH
TURNER	1500	RESEARCH
ADAMS	1100	RESEARCH
JAMES	950	RESEARCH
FORD	3000	RESEARCH
MILLER	1300	RESEARCH
SMITH	800	SALES
ALLEN	1600	SALES
WARD	1250	SALES
JONES	2975	SALES
MARTIN	1250	SALES
BLAKE	2850	SALES
CLARK	2450	SALES
SCOTT	3000	SALES
KING	5000	SALES
TURNER	1500	SALES
ADAMS	1100	SALES
JAMES	950	SALES
FORD	3000	SALES
MILLER	1300	SALES
SMITH	800	OPERATIONS
ALLEN	1600	OPERATIONS
WARD	1250	OPERATIONS
JONES	2975	OPERATIONS
MARTIN	1250	OPERATIONS
BLAKE	2850	OPERATIONS
CLARK	2450	OPERATIONS
SCOTT	3000	OPERATIONS
KING	5000	OPERATIONS
TURNER	1500	OPERATIONS
ADAMS	1100	OPERATIONS
JAMES	950	OPERATIONS
FORD	3000	OPERATIONS
MILLER	1300	OPERATIONS

56 rows selected.

- From above – we can see that the above query returns 56 records – but we are expecting 14 records.
- This is because each and every record of employee table will be combined with each & every record of department table.
- Thus, Cartesian join should not be used in real time scenarios.
- The Cartesian join contains both correct and incorrect sets of data. We have to retain the correct ones & eliminate the incorrect ones by using the **inner join**.

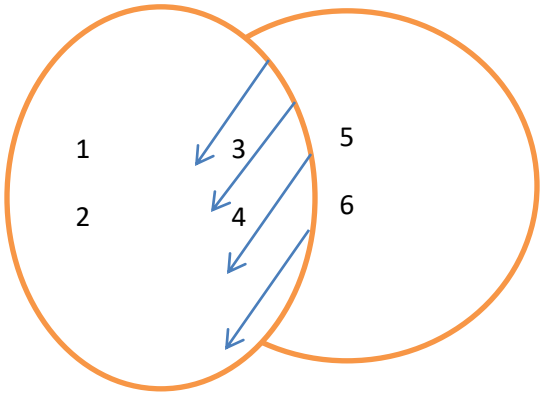
INNER JOIN

1. Inner join are also called as **equijoins.**
2. They return the **matching records between the tables by equating common column.**
3. In the real time scenarios, **this is the most frequently used Join.**

EX:

Table 1={1,2,3,4}

Table 2={3,4,5,6}



Note: JOIN condition is mandatory for removing the Cartesian output.

Ansi syntax:

Select <column_list>

From t1 join t2

On t1.common_column=t2.common_column;

Oracle syntax:

Select <column_list>

From t1,t2

Where t1.common_column=t2.common_column;

For ex, consider the query shown below,

```
Select A.ename, A.sal, B.dname
From emp A, dept B
Where A.deptno = B.deptno                    - JOIN condition
And A.sal > 2000                             - FILTER condition
Order by A.sal ;
```

Let us see the output shown below,

```
SQL> Select A.ename, A.sal, B.dname
  2  From emp A, dept B
  3  Where A.deptno = B.deptno
  4  And A.sal > 2000
  5  Order by A.sal ;

ENAME             SAL DNAME
---------- ---------- --------------
CLARK            2450 ACCOUNTING
BLAKE            2850 SALES
JONES            2975 RESEARCH
FORD             3000 RESEARCH
SCOTT            3000 RESEARCH
KING             5000 ACCOUNTING

6 rows selected.
```

Let us consider the following 2 scenarios shown below,

Scenario 1

A		
P	**Q**	**R**

B		
P	**S**	**T**

C		
P	**X**	**Y**

We want			
P	**Q**	**S**	**X**

The SQL query will be,

Select A.P, A.Q, B.S, C.X
From A, B, C
Where A.P = B.P **Number of joins = 2**
And A.P = C.P

Therefore, Number of JOINS = Number of tables - 1

Scenario 2

A		
P	**Q**	**R**

B			
P	**Q**	**S**	**T**

C		
P	**X**	**Y**

We want				
P	**Q**	**R**	**S**	**X**

The **SQL query is ,**

Select A.P, A.Q, A.R, B.S, C.X
From A, B, C
Where A.P = B.P
And A.Q = B.Q **Number of Joins = 3**
And A.P = C.P ;

Therefore, Number of JOINS = Number of common columns

If there are no common columns, then reject it saying that the two tables can be joined.

But there are some cases – where the 2 columns will be same but having different column names.

For ex – customerid & cid

Display employee name, his job, his dname and his location for all the managers living in New York or Chicago

```
SQL> select A.ename, A.job, B.dname, B.loc
  2  from emp A, dept B
  3  where A.deptno = B.deptno
  4  and A.job = 'MANAGER'
  5  and B.loc in ('NEW YORK', 'CHICAGO') ;

ENAME      JOB       DNAME          LOC
---------- --------- -------------- -------------
BLAKE      MANAGER   SALES          CHICAGO
CLARK      MANAGER   ACCOUNTING     NEW YORK
```

ANSI style JOINS:

1. This was introduced from **Oracle 9i.**
2. It is another way of writing inner joins with a few modifications.

```
SQL> select A.ename, A.job, B.dname, B.loc
  2  from emp A join dept B
  3  on A.deptno = B.deptno
  4  and A.job = 'MANAGER'
  5  and B.loc in ('NEW YORK', 'CHICAGO') ;

ENAME      JOB       DNAME          LOC
---------- --------- -------------- -------------
BLAKE      MANAGER   SALES          CHICAGO
CLARK      MANAGER   ACCOUNTING     NEW YORK
```

Thus we, can see the changes ,

- In the 2nd line - **,(comma)** has been **replaced by the word 'join'**
- In the 3rd line – **'where'** has been **replaced** with **'on'**

OUTTER JOIN

Outter join:

1. It returns **both matching and non-matching records**
2. Outer join = inner join + non-matching records
3. Non-matching records means data present in one table, but absent in another table. It will be read and write to use common columns.

For ex, 40 is there in deptno of dept table, but not there in deptno of emp table.

Using right join

```
SQL> select A.ename, A.job, B.dname, B.loc
  2  from emp A right join dept B
  3  on A.deptno = B.deptno ;

ENAME      JOB       DNAME          LOC
---------- --------- -------------- -------------
CLARK      MANAGER   ACCOUNTING     NEW YORK
KING       PRESIDENT ACCOUNTING     NEW YORK
MILLER     CLERK     ACCOUNTING     NEW YORK
JONES      MANAGER   RESEARCH       DALLAS
FORD       ANALYST   RESEARCH       DALLAS
ADAMS      CLERK     RESEARCH       DALLAS
SMITH      CLERK     RESEARCH       DALLAS
SCOTT      ANALYST   RESEARCH       DALLAS
WARD       SALESMAN  SALES          CHICAGO
TURNER     SALESMAN  SALES          CHICAGO
ALLEN      SALESMAN  SALES          CHICAGO
JAMES      CLERK     SALES          CHICAGO
BLAKE      MANAGER   SALES          CHICAGO
MARTIN     SALESMAN  SALES          CHICAGO
                     OPERATIONS     BOSTON

15 rows selected.
```

Types of outter join:

1. Left outter join
2. Right outter join
3. Full outter join

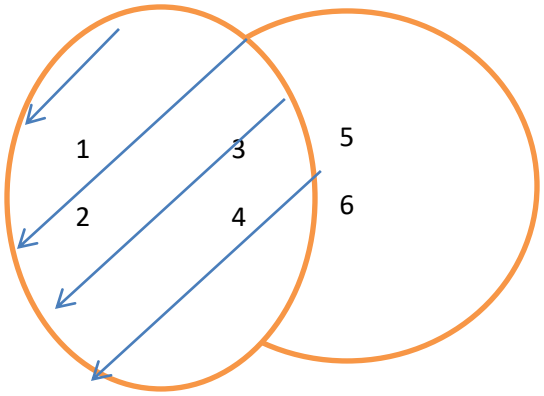
Left outter join:

1. It is used to **display the matched records from both the tables as well as the unmatched records from the left table.**

EX:

Table 1={1,2,3,4}

Table 2={3,4,5,6}



Ansi syntax:

Select <column_list>

From t1 left outter join t2

On t1.common_column=t2.common_column;

Oracle syntax:

Select <column_list>

From t1,t2

Where t1.common_column=t2.common_column(+);

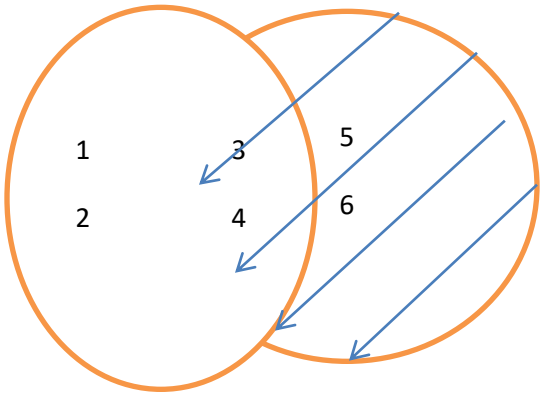
Right outter join:

1. It is used to **display the matched records from both the tables as well as the unmatched records from the right table.**

EX:

Table 1={1,2,3,4}

Table 2={3,4,5,6}



Ansi syntax:

Select <column_list>

From t1 right outter join t2

On t1.common_column=t2.common_column;

Oracle syntax:

Select <column_list>

From t1,t2

Where t1.common_column(+)=t2.common_column;

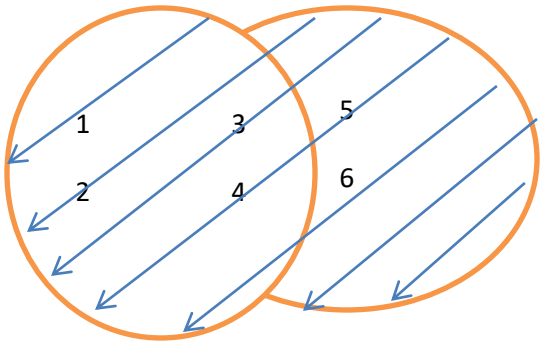
Full outter join:

1. It is used to **display both matched records from both the tables.**

EX:

Table 1={1,2,3,4}

Table 2={3,4,5,6}



Ansi syntax:

Select <column_list>

From t1 full outter join t2

On t1.common_column=t2.common_column;

Oracle syntax:

Select <column_list> From t1,t2

Where t1.common_column=t2.common_column(+);

Union

Select <column_list>

From t1,t2

Where t1.common_column(+)=t2.common_column;

SELF JOIN

Joining a table to itself is called self join

The **FROM** clause looks like this,

FROM emp A, emp B

Or

FROM emp A join emp B - *ANSI style*

For ex, - **Display employee name along with their manager name**

```
SQL> select A.ename "EMP",
  2          B.ename "MANAGER"
  3  from emp A, emp B
  4  where A.mgr = B.empno ;

EMP        MANAGER
---------- ----------
SMITH      FORD
ALLEN      BLAKE
WARD       BLAKE
JONES      KING
MARTIN     BLAKE
BLAKE      KING
CLARK      KING
SCOTT      JONES
TURNER     BLAKE
ADAMS      SCOTT
JAMES      BLAKE
FORD       JONES
MILLER     CLARK

13 rows selected.
```

Now, let us see how this i.e the logic (the above query) works,

Emp (A)		
EmpNo	**Ename**	**Mgr**
101	Scott	102
102	Blake	103
103	King	-
104	Smith	103
105	Jones	104

Emp (B)		
EmpNo	**Ename**	**Mgr**
101	Scott	102
102	Blake	103
103	King	-
104	Smith	103
105	Jones	104

Now, when we give the above query – in Oracle – it starts matching the '**mgr**' column of **emp A** with the '**empno**' of **emp b** – we get two tables because in **self join** – a duplicate of the table required is created.

Now let us consider the **first employee Scott** – it starts the **mgrid** of **Scott** with the **empno** of all the records in **emp B** – when two **ids** match, then the **empno** in **emp B** becomes the **mgr** of the **empno** in **emp A**. Thus, we can see that – **mgr id** 102 is matching with **empno** 102 **Blake** in **emp B**. Therefore, Blake is the manager of Scott.
Similarly we do the same for all the other records of **emp A** and thus find the employees and their respective managers.

Display the employees who are getting the same salary

```
SQL> select A.ename, A.sal
  2  from emp A join emp B
  3  on A.sal = B.sal
  4  and A.empno <> B.empno ;

ENAME             SAL
---------- ----------
MARTIN           1250
WARD             1250
FORD             3000
SCOTT            3000
```

Set operators

The SQL UNION clause/operator is used to combine the results of two or more SELECT statements without returning any duplicate rows.

To use this UNION clause, each SELECT statement must have

- The same number of columns selected
- The same number of column expressions
- The same data type and
- Have them in the same order

But they need not have to be in the same length.

Syntax

The basic syntax of a **UNION** clause is as follows −

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]

UNION

SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

Here, the given condition could be any given expression based on your requirement.

Example

Consider the following two tables.

Table 1 − CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+

| ID | NAME     | AGE | ADDRESS   | SALARY   |
```

```
+----+----------+-----+-----------+----------+
| 1 | Ramesh   | 32 | Ahmedabad |  2000.00 |
| 2 | Khilan   | 25 | Delhi     |  1500.00 |
| 3 | kaushik  | 23 | Kota      |  2000.00 |
| 4 | Chaitali |  25 | Mumbai    |  6500.00 |
| 5 | Hardik   | 27 | Bhopal    |  8500.00 |
| 6 | Komal    | 22 | MP        |  4500.00 |
| 7 | Muffy    | 24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Table 2 − ORDERS Table is as follows.

```
+-----+---------------------+------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+------------+--------+
| 102 | 2009-10-08 00:00:00 |          3 |   3000 |
| 100 | 2009-10-08 00:00:00 |          3 |   1500 |
| 101 | 2009-11-20 00:00:00 |          2 |   1560 |
| 103 | 2008-05-20 00:00:00 |          4 |   2060 |
+-----+---------------------+------------+--------+
```

Now, let us join these two tables in our SELECT statement as follows −

```
SQL> SELECT  ID, NAME, AMOUNT, DATE
  FROM CUSTOMERS
```

```
   LEFT JOIN ORDERS
   ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID
UNION
   SELECT  ID, NAME, AMOUNT, DATE
   FROM CUSTOMERS
   RIGHT JOIN ORDERS
   ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result −

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
|    1 | Ramesh   |   NULL | NULL                |
|    2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|    3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|    4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
|    5 | Hardik   |   NULL | NULL                |
|    6 | Komal    |   NULL | NULL                |
|    7 | Muffy    |   NULL | NULL                |
+------+----------+--------+---------------------+
```

The UNION ALL Clause

The UNION ALL operator is used to combine the results of two SELECT statements including duplicate rows.

The same rules that apply to the UNION clause will apply to the UNION ALL operator.

Syntax

The basic syntax of the **UNION ALL** is as follows.

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]

UNION ALL

SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

Here, the given condition could be any given expression based on your requirement.

Example

Consider the following two tables,

Table 1 − CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Table 2 − ORDERS table is as follows.

```
+-----+---------------------+------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+------------+--------+
| 102 | 2009-10-08 00:00:00 |          3 |   3000 |
| 100 | 2009-10-08 00:00:00 |          3 |   1500 |
| 101 | 2009-11-20 00:00:00 |          2 |   1560 |
| 103 | 2008-05-20 00:00:00 |          4 |   2060 |
+-----+---------------------+------------+--------+
```

Now, let us join these two tables in our SELECT statement as follows −

```
SQL> SELECT  ID, NAME, AMOUNT, DATE
   FROM CUSTOMERS
   LEFT JOIN ORDERS
   ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID
UNION ALL
   SELECT  ID, NAME, AMOUNT, DATE
   FROM CUSTOMERS
   RIGHT JOIN ORDERS
   ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result −

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
```

```
|    1 | Ramesh   |   NULL | NULL                |
|    2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|    3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|    4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
|    5 | Hardik   |   NULL | NULL                |
|    6 | Komal    |   NULL | NULL                |
|    7 | Muffy    |   NULL | NULL                |
|    3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|    2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|    4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```

There are two other clauses (i.e., operators), which are like the UNION clause.

INTERSECT:

SQL INTERSECT Clause − This is used to combine two SELECT statements, but returns rows only from the first SELECT statement that are identical to a row in the second SELECT statement.

The SQL **INTERSECT** clause/operator is used to combine two SELECT statements, but returns rows only from the first SELECT statement that are identical to a row in the second SELECT statement. This means INTERSECT returns only common rows returned by the two SELECT statements.

Just as with the UNION operator, the same rules apply when using the INTERSECT operator. MySQL does not support the INTERSECT operator.

Syntax

The basic syntax of **INTERSECT** is as follows.

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]

INTERSECT

SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

Here, the given condition could be any given expression based on your requirement.

Example

Consider the following two tables.

Table 1 − CUSTOMERS Table is as follows

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Table 2 − ORDERS Table is as follows.

```
+-----+---------------------+------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+------------+--------+
| 102 | 2009-10-08 00:00:00 |          3 |   3000 |
| 100 | 2009-10-08 00:00:00 |          3 |   1500 |
| 101 | 2009-11-20 00:00:00 |          2 |   1560 |
| 103 | 2008-05-20 00:00:00 |          4 |   2060 |
+-----+---------------------+------------+--------+
```

Now, let us join these two tables in our SELECT statement as follows.

```
SQL> SELECT  ID, NAME, AMOUNT, DATE
  FROM CUSTOMERS
  LEFT JOIN ORDERS
  ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID
INTERSECT
  SELECT  ID, NAME, AMOUNT, DATE
  FROM CUSTOMERS
  RIGHT JOIN ORDERS
  ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result.

```
+------+---------+--------+---------------------+
| ID   | NAME    | AMOUNT | DATE                |
+------+---------+--------+---------------------+
```

```
|    3 | kaushik |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik |   1500 | 2009-10-08 00:00:00 |
|    2 | Ramesh  |   1560 | 2009-11-20 00:00:00 |
|    4 | kaushik |   2060 | 2008-05-20 00:00:00 |
+------+---------+--------+---------------------+
```

EXCEPT:

SQL EXCEPT Clause − This combines two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement.

The SQL **EXCEPT** clause/operator is used to combine two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement. This means EXCEPT returns only rows, which are not available in the second SELECT statement.

Just as with the UNION operator, the same rules apply when using the EXCEPT operator. MySQL does not support the EXCEPT operator.

Syntax

The basic syntax of **EXCEPT** is as follows.

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]

EXCEPT

SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

Here, the given condition could be any given expression based on your requirement.

Example

Consider the following two tables.

Table 1 − CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Table 2 − ORDERS table is as follows.

```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
```

```
| 101 | 2009-11-20 00:00:00 |          2 |   1560 |
| 103 | 2008-05-20 00:00:00 |          4 |   2060 |
+-----+---------------------+------------+--------+
```

Now, let us join these two tables in our SELECT statement as shown below.

```
SQL> SELECT  ID, NAME, AMOUNT, DATE
  FROM CUSTOMERS
  LEFT JOIN ORDERS
  ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID
EXCEPT
  SELECT  ID, NAME, AMOUNT, DATE
  FROM CUSTOMERS
  RIGHT JOIN ORDERS
  ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result.

```
+----+---------+--------+---------------------+
| ID | NAME    | AMOUNT | DATE                |
+----+---------+--------+---------------------+
|  1 | Ramesh  |   NULL | NULL                |
|  5 | Hardik  |   NULL | NULL                |
|  6 | Komal   |   NULL | NULL                |
|  7 | Muffy   |   NULL | NULL                |
+----+---------+--------+---------------------+
```

SQL | MINUS Operator:

The Minus Operator in SQL is used with two SELECT statements. The MINUS operator is used to subtract the result set obtained by first SELECT query from the result set obtained by second SELECT query. In simple words, we can say that MINUS operator will return only those rows which are unique in only first SELECT query and not those rows which are common to both first and second SELECT queries.

Pictorial Representation:

sql-minus

As you can see is in the above diagram, the MINUS operator will return only those rows which are present in the result set from Table1 and not present in the result set of Table2.

Basic Syntax:

```
SELECT column1 , column2 , ... columnN
FROM table_name
WHERE condition
MINUS
SELECT column1 , column2 , ... columnN
FROM table_name
WHERE condition;
```

columnN: column1, column2.. are the name of columns of the table.

Important Points:

The WHERE clause is optional in the above query.

The number of columns in both SELECT statements must be same.

The data type of corresponding columns of both SELECT statement must be same.

Sample Tables:

Table1

table1

Queries:

```
SELECT NAME, AGE , GRADE
FROM Table1
MINUS
SELECT NAME, AGE, GRADE
FROM Table2
```

Output:

The above query will return only those rows which are unique in ‘Table1’. We can clearly see that values in the fields NAME, AGE and GRADE for the last row in both tables are same. Therefore, the output will be the first three rows from Table1. The obtained output is shown below:

output

Note: The MINUS operator is not supported with all databases. It is supported by Oracle database but not SQL server or PostgreSQL.

This article is contributed by Harsh Agarwal. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

NORMALIZATION

NORMALIZATION:

1. **Normalization** is the process of splitting the bigger table into many small tables without changing its functionality.
2. It is generally carried out during the design phase of SDLC.

Advantages:

1) it reduces the redundancy (unnecessary repeatation of data)
2) avoids problem due to delete anamoly (inconsistency)

Normalization is a step-by-step process and in each step, we have to perform some activities.

STEPS IN NORMALIZATION:

1) 1NF – 1st Normal form
2) 2NF – 2nd Normal form
3) 3NF – 3rd Normal form

1NF

1. We should collect all the required attributes into 1 or more bigger entities.
2. We have to assume no 2 records are same (i.e, records should not be duplicated)
3. Identify the probable primary key

At the end of 1NF, our data looks like this,

COLLEGE
RegNo - **PK**
Sname
Semester
DOB
MailID
Phone
BookNo - **PK**
Bname
Author
DOI
DOR
Fine

2NF:

To perform 2NF,

1. The tables have to be in 1NF
2. Here, we identify all the complete dependencies and move them separately into different tables.

At the end of 2NF, our data looks like this,

STUDENTS
RegNo - PK
Sname
Semester
DOB
MailID
Phone

BOOKS
BookNo - PK
RegNo - FK
Bname
Author
DOI
DOR
Fine

3NF:

1. The table will have to be in 2NF
2. Here, we identify all the partial dependencies and move such columns to a separate table.

STUDENTS
RegNo - PK
Sname
Semester
DOB
MailID
Phone

BOOKS
BookNo - PK
Bname
Author

LIBRARY
BookNo - FK
RegNo - FK
DOI
DOR
Fine

Disadvantage of Normalization:

1. The only minor disadvantage is we may have to write complex queries as we have more number of tables to be accessed.

Denormalization is the process of combining more than 1 smaller table to form 1 bigger table is called as denormalization.

CODD rules (**Differentiates between DBMS & RDBMS**):

1) should support NULL values
2) should support creation of relationship between tables
3) should support DDL, DML, TCL
4) should support constraints like PK, Unique, CHK
5) should support query techniques like sub – queries, joins, grouping etc.

Oracle 9i Features (i means internet):

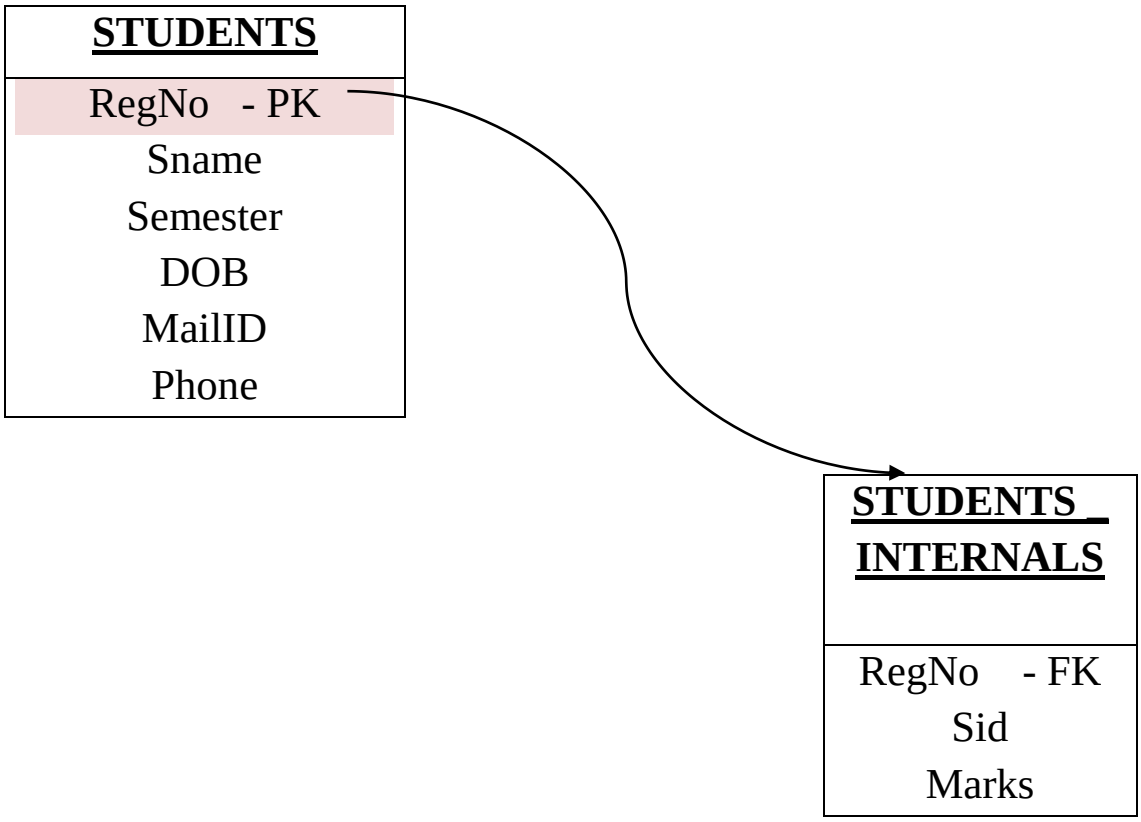
1. TIMESTAMP datatype
2. SYSTIMESTAMP function
3. ANSI style joins
4. Renaming a column

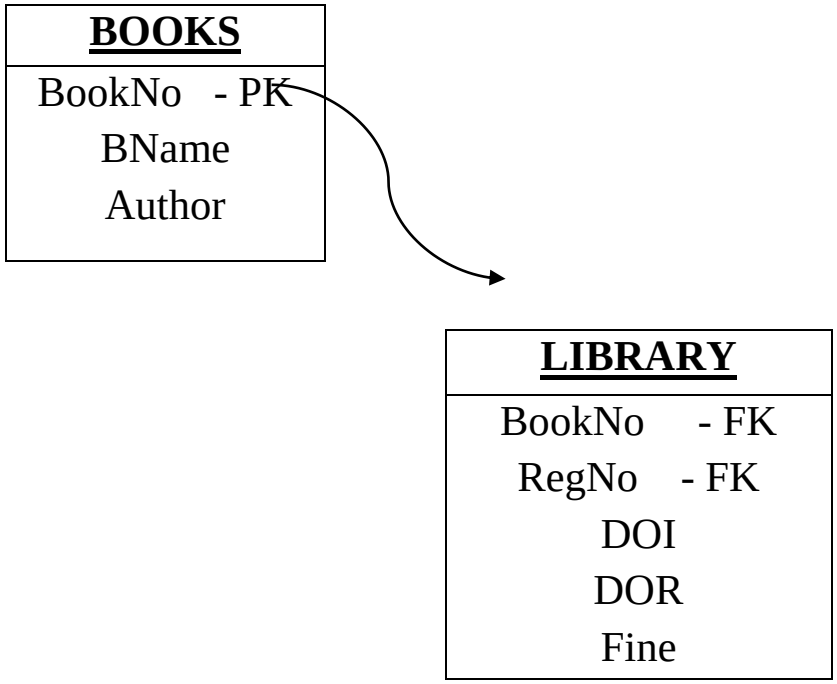
Oracle 10g features (g means grid)

1. Recycle Bin

ERD - Entity Relationship Diagram

1. It is the pictorial representation of all the entities and their relationships (tables).




BOOKS
BookNo - PK
BName
Author
LIBRARY
BookNo - FK
RegNo - FK
DOI
DOR
Fine

SQL indexes

In this we will see how to create, delete and uses of index in database.

An index is a schema object. It is used by the server to speed up the retrieval of rows by using a pointer. It can reduce disk I/O(input/output) by using a rapid path access method to locate data quickly. An index helps to speed up select queries and where clauses, but it slows down data input, with the update and the insert statements. Indexes can be created or dropped with no effect on the data.

For example, if you want to reference all pages in a book that discusses a certain topic, you first refer to the index, which lists all the topics alphabetically and are then referred to one or more specific page numbers.

Creating an Index – It's syntax is:

CREATE INDEX index

ON TABLE column;

where index is the name given to that index and TABLE is the name of the table on which that index is created and column is the name of that column for which it is applied.

For multiple columns –

CREATE INDEX index

ON TABLE (cloumn1, column2,.....);

Unique Indexes –

CREATE UNIQUE INDEX index

ON TABLE column;

Unique indexes are used for the maintenance of the integrity of the data present in the table as well as for the fast performance, it does not allow multiple values to enter into the table.

When should indexes be created –

A column contains a wide range of values

A column does not contain a large number of null values

One or more columns are frequently used together in a where clause or a join condition

When should indexes be avoided –

The table is small

The columns are not often used as a condition in the query

The column is updated frequently

Removing an Index – To remove an index from the data dictionary by using the DROP INDEX command.

DROP INDEX index;

To drop an index, you must be the owner of the index or have the DROP ANY INDEX privilege.

Confirming Indexes –

You can check the different indexes present on a particular table given by the user or the server itself and their uniqueness.

select *

from USER_INDEXES;

It will show you all the indexes present in the server, in which you can locate your own tables too.

Indexes are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index in the back of a book.

For example, if you want to reference all pages in a book that discusses a certain topic, you first refer to the index, which lists all the topics alphabetically and are then referred to one or more specific page numbers.

An index helps to speed up SELECT queries and WHERE clauses, but it slows down data input, with the UPDATE and the INSERT statements. Indexes can be created or dropped with no effect on the data.

Creating an index involves the CREATE INDEX statement, which allows you to name the index, to specify the table and which column or columns to index, and to indicate whether the index is in an ascending or descending order.

Indexes can also be unique, like the UNIQUE constraint, in that the index prevents duplicate entries in the column or combination of columns on which there is an index.

The CREATE INDEX Command

The basic syntax of a CREATE INDEX is as follows.

CREATE INDEX index_name ON table_name;

Single-Column Indexes

A single-column index is created based on only one table column. The basic syntax is as follows.

```
CREATE INDEX index_name
ON table_name (column_name);
```

Unique Indexes

Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table. The basic syntax is as follows.

```
CREATE UNIQUE INDEX index_name
on table_name (column_name);
```

Composite Indexes

A composite index is an index on two or more columns of a table. Its basic syntax is as follows.

```
CREATE INDEX index_name
on table_name (column1, column2);
```

Whether to create a single-column index or a composite index, take into consideration the column(s) that you may use very frequently in a query's WHERE clause as filter conditions.

Should there be only one column used, a single-column index should be the choice. Should there be two or more columns that are frequently used in the WHERE clause as filters, the composite index would be the best choice.

Implicit Indexes

Implicit indexes are indexes that are automatically created by the database server when an object is created. Indexes are automatically created for primary key constraints and unique constraints.

The DROP INDEX Command

An index can be dropped using SQL DROP command. Care should be taken when dropping an index because the performance may either slow down or improve.

The basic syntax is as follows −

```
DROP INDEX index_name;
```

You can check the INDEX Constraint chapter to see some actual examples on Indexes.

When should indexes be avoided?

Although indexes are intended to enhance a database's performance, there are times when they should be avoided.

The following guidelines indicate when the use of an index should be reconsidered.

Indexes should not be used on small tables.

Tables that have frequent, large batch updates or insert operations.

Indexes should not be used on columns that contain a high number of NULL values.

Columns that are frequently manipulated should not be indexed.

VIEW

A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.

A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.

Views, which are a type of virtual tables allow users to do the following –

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.

Creating Views

Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables or another view.

To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic **CREATE VIEW** syntax is as follows −

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE [condition];
```

You can include multiple tables in your SELECT statement in a similar way as you use them in a normal SQL SELECT query.

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Following is an example to create a view from the CUSTOMERS table. This view would be used to have customer name and age from the CUSTOMERS table.

```
SQL > CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM  CUSTOMERS;
```

Now, you can query CUSTOMERS_VIEW in a similar way as you query an actual table. Following is an example for the same.

```
SQL > SELECT * FROM CUSTOMERS_VIEW;
```

This would produce the following result.

```
+----------+-----+
| name     | age |
+----------+-----+
| Ramesh   |  32 |
| Khilan   |  25 |
| kaushik  |  23 |
| Chaitali |  25 |
| Hardik   |  27 |
| Komal    |  22 |
| Muffy    |  24 |
+----------+-----+
```

The WITH CHECK OPTION

The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition.

If they do not satisfy the condition(s), the UPDATE or INSERT returns an error.

The following code block has an example of creating same view CUSTOMERS_VIEW with the WITH CHECK OPTION.

```
CREATE VIEW CUSTOMERS_VIEW AS

SELECT name, age

FROM  CUSTOMERS

WHERE age IS NOT NULL

WITH CHECK OPTION;
```

The WITH CHECK OPTION in this case should deny the entry of any NULL values in the view's AGE column, because the view is defined by data that does not have a NULL value in the AGE column.

Updating a View

A view can be updated under certain conditions which are given below –

- The SELECT clause may not contain the keyword DISTINCT.
- The SELECT clause may not contain summary functions.
- The SELECT clause may not contain set functions.
- The SELECT clause may not contain set operators.
- The SELECT clause may not contain an ORDER BY clause.
- The FROM clause may not contain multiple tables.
- The WHERE clause may not contain subqueries.
- The query may not contain GROUP BY or HAVING.
- Calculated columns may not be updated.
- All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

So, if a view satisfies all the above-mentioned rules then you can update that view. The following code block has an example to update the age of Ramesh.

```
SQL > UPDATE CUSTOMERS_VIEW
  SET AGE = 35
  WHERE name = 'Ramesh';
```

This would ultimately update the base table CUSTOMERS and the same would reflect in the view itself. Now, try to query the base table and the SELECT statement would produce the following result.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
```

```
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Inserting Rows into a View

Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the INSERT command.

Here, we cannot insert rows in the CUSTOMERS_VIEW because we have not included all the NOT NULL columns in this view, otherwise you can insert rows in a view in a similar way as you insert them in a table.

Deleting Rows into a View

Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.

Following is an example to delete a record having AGE = 22.

```
SQL > DELETE FROM CUSTOMERS_VIEW
  WHERE age = 22;
```

This would ultimately delete a row from the base table CUSTOMERS and the same would reflect in the view itself. Now, try to query the base table and the SELECT statement would produce the following result.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
```

```
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Dropping Views

Obviously, where you have a view, you need a way to drop the view if it is no longer needed. The syntax is very simple and is given below −

```
DROP VIEW view_name;
```

Following is an example to drop the CUSTOMERS_VIEW from the CUSTOMERS table.

```
DROP VIEW CUSTOMERS_VIEW;
```

SQL | Views

Views in SQL are kind of virtual tables. A view also has rows and columns as they are in a real table in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain condition.

In this article we will learn about creating , deleting and updating Views.

Sample Tables:

StudentDetails

Screenshot (57)

StudentMarks

Screenshot (58)

CREATING VIEWS

We can create View using CREATE VIEW statement. A View can be created from a single table or multiple tables.

Syntax:

CREATE VIEW view_name AS

SELECT column1, column2.....

FROM table_name

WHERE condition;

view_name: Name for the View

table_name: Name of the table

condition: Condition to select rows

Examples:

Creating View from a single table:

In this example we will create a View named DetailsView from the table StudentDetails.

Query:

```
CREATE VIEW DetailsView AS
SELECT NAME, ADDRESS
FROM StudentDetails
WHERE S_ID < 5;
```

To see the data in the View, we can query the view in the same manner as we query a table.

```
SELECT * FROM DetailsView;
```

Output:

Screenshot (57)

In this example, we will create a view named StudentNames from the table StudentDetails.

Query:

```
CREATE VIEW StudentNames AS
SELECT S_ID, NAME
FROM StudentDetails
ORDER BY NAME;
```

If we now query the view as,

```
SELECT * FROM StudentNames;
```

Output:

Screenshot (64)

Creating View from multiple tables: In this example we will create a View named MarksView from two tables StudentDetails and StudentMarks. To create a View from multiple tables we can simply include multiple tables in the SELECT statement. Query:

```
CREATE VIEW MarksView AS
SELECT StudentDetails.NAME, StudentDetails.ADDRESS,
StudentMarks.MARKS
FROM StudentDetails, StudentMarks
WHERE StudentDetails.NAME = StudentMarks.NAME;
```

To display data of View MarksView:

```
SELECT * FROM MarksView;
```

Output:

Screenshot (59)

DELETING VIEWS

We have learned about creating a View, but what if a created View is not needed any more? Obviously we will want to delete it. SQL allows us to delete an existing View. We can delete or drop a View using the DROP statement.

Syntax:

DROP VIEW view_name;

view_name: Name of the View which we want to delete.

For example, if we want to delete the View MarksView, we can do this as:

DROP VIEW MarksView;

UPDATING VIEWS

There are certain conditions needed to be satisfied to update a view. If any one of these conditions is not met, then we will not be allowed to update the view.

The SELECT statement which is used to create the view should not include GROUP BY clause or ORDER BY clause.

The SELECT statement should not have the DISTINCT keyword.

The View should have all NOT NULL values.

The view should not be created using nested queries or complex queries.

The view should be created from a single table. If the view is created using multiple tables then we will not be allowed to update the view.

We can use the CREATE OR REPLACE VIEW statement to add or remove fields from a view.

Syntax:

CREATE OR REPLACE VIEW view_name AS

SELECT column1,coulmn2,..

FROM table_name

WHERE condition;

For example, if we want to update the view MarksView and add the field AGE to this View from StudentMarks Table, we can do this as:

CREATE OR REPLACE VIEW MarksView AS

```
SELECT StudentDetails.NAME, StudentDetails.ADDRESS,
StudentMarks.MARKS, StudentMarks.AGE

FROM StudentDetails, StudentMarks

WHERE StudentDetails.NAME = StudentMarks.NAME;
```

If we fetch all the data from MarksView now as:

```
SELECT * FROM MarksView;
```

Output:

Screenshot (60)

Inserting a row in a view:

We can insert a row in a View in a same way as we do in a table. We can use the INSERT INTO statement of SQL to insert a row in a View.Syntax:

```
INSERT view_name(column1, column2 , column3,..)

VALUES(value1, value2, value3..);
```

view_name: Name of the View

Example:

In the below example we will insert a new row in the View DetailsView which we have created above in the example of “creating views from a single table”.

INSERT INTO DetailsView(NAME, ADDRESS)

VALUES("Suresh","Gurgaon");

If we fetch all the data from DetailsView now as,

SELECT * FROM DetailsView;

Output:

Screenshot (62)

Deleting a row from a View:

Deleting rows from a view is also as simple as deleting rows from a table. We can use the DELETE statement of SQL to delete rows from a view. Also deleting a row from a view first delete the row from the actual table and the change is then reflected in the view.Syntax:

DELETE FROM view_name

WHERE condition;

view_name:Name of view from where we want to delete rows

condition: Condition to select rows

Example:

In this example we will delete the last row from the view DetailsView which we just added in the above example of inserting rows.

DELETE FROM DetailsView

WHERE NAME="Suresh";

If we fetch all the data from DetailsView now as,

SELECT * FROM DetailsView;

Output:

Screenshot (57)

WITH CHECK OPTION

The WITH CHECK OPTION clause in SQL is a very useful clause for views. It is applicable to a updatable view. If the view is not updatable, then there is no meaning of including this clause in the CREATE VIEW statement.

The WITH CHECK OPTION clause is used to prevent the insertion of rows in the view where the condition in the WHERE clause in CREATE VIEW statement is not satisfied.

If we have used the WITH CHECK OPTION clause in the CREATE VIEW statement, and if the UPDATE or INSERT clause does not satisfy the conditions then they will return an error.

Example:

In the below example we are creating a View SampleView from StudentDetails Table with WITH CHECK OPTION clause.

```
CREATE VIEW SampleView AS

SELECT S_ID, NAME

FROM  StudentDetails

WHERE NAME IS NOT NULL

WITH CHECK OPTION;
```

In this View if we now try to insert a new row with null value in the NAME column then it will give an error because the view is created with the condition for NAME column as NOT NULL.

For example,though the View is updatable but then also the below query for this View is not valid:

```
INSERT INTO SampleView(S_ID)

VALUES(6);
```

NOTE: The default value of NAME column is null.

Case statement

SQL | Case Statement

Control statements form the heart of most languages since they control the execution of other sets of statements. These are found in SQL too, and should be exploited for uses such as query filtering and query optimization through careful selection of tuples that match our requirement. In this post, we explore the Case-Switch statement in SQL. The CASE statement is SQL's way of handling if/then logic.
Syntax:
There can be two valid ways of going about the case-switch statements.

1. The first takes a variable called case_value and matches it with some statement_list.

```
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

7. The second considers a search_condition instead of variable equality and executes the statement_list accordingly.

```
CASE
    WHEN search_condition THEN statement_list
        [WHEN search_condition THEN statement_list] ...
        [ELSE statement_list]
 END CASE
```

Examples:

Say we have a relation, Faculty.

Faculty Table:

FACULTYID	NAME	DEPARTMENT	GENDER
001	Aakash	CS	M
002	Sahil	EC	M
003	John	HSS	M
004	Shelley	CS	F
005	Anannya	CS	F
006	Sia	HSS	F

Let's say we would like to modify this table such that if the department name is 'CS', it gets modified to 'Computer Science', if it is 'EC' it gets modified to 'Electronics and Communication', and if it is 'HSS' it gets modified to 'Humanities and Social Sciences'. This can be achieved using case statement.

Sample Query:

Consider a variable, department_name which is entered in the SQL code.

```
CASE department_name
 WHEN 'CS'
```

```
 THEN UPDATE Faculty SET
 department='Computer Science';
WHEN 'EC'
 THEN UPDATE Faculty SET
 department='Electronics and Communication';
ELSE UPDATE Faculty SET
 department='Humanities and Social Sciences';
END CASE
```

Output:

The department name corresponding to the given input gets renamed.

Consider another query which selects all the fields corresponding to the Faculty table. Since the values written in the Gender field are single character values (M/F), we would like to present them in a more readable format.

```
SELECT FacultyID, Name, Department,
CASE Gender
 WHEN'M' THEN 'Male'
 WHEN'F' THEN 'Female'
END
FROM Faculty
```

Output:

FACULTYID	NAME	DEPARTMENT	GENDER

001	Aakash	CS	Male
002	Sahil	EC	Male
003	John	HSS	Male
004	Shelley	CS	Female
005	Anannya	CS	Female
006	Sia	HSS	Female

Consider yet another application of case-switch in SQL- custom sorting.

```
CREATE PROCEDURE GetFaculty(@ColToSort varchar(150)) AS
SELECT FacultyID, Name, Gender, Department
FROM Customers
ORDER BY
  CASE WHEN @ColToSort='Department' THEN Department
       WHEN @ColToSort='Name' THEN Name
       WHEN @ColToSort='Gender' THEN Gender
       ElSE FacultyID
  END
```

Output:

The output gets sorted according to the provided field.

The above procedure (function) takes a variable of varchar data type as its argument, and on the basis of that, sorts the tuples in the Faculty table.

This article is contributed by **Anannya Uberoi**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

DECODE:

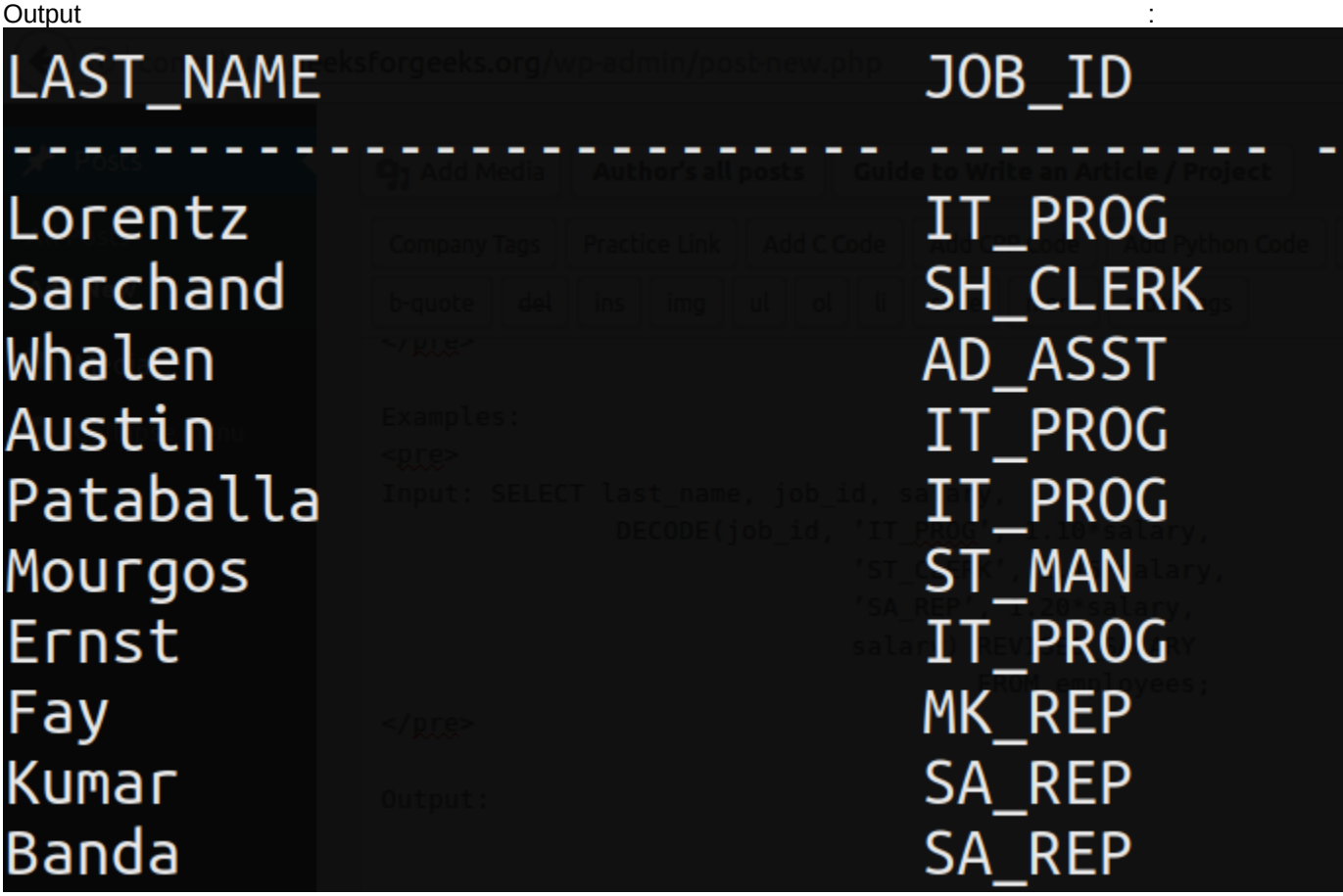
- **DECODE()** : Facilitates conditional inquiries by doing the work of a CASE or IF-THEN-ELSE statement.
  The DECODE function decodes an expression in a way similar to the IF-THEN-ELSE logic used in various languages. The DECODE function decodes expression after comparing it to each search value. If the expression is the same as search, result is returned.
  If the default value is omitted, a null value is returned where a search value does not match any of the result values.
  Syntax –

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

Example –

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

Output :



```
LAST_NAME                 JOB_ID
------------------------- ---------- -
Lorentz                   IT_PROG
Sarchand                  SH_CLERK
Whalen                    AD_ASST
Austin                    IT_PROG
Pataballa                 IT_PROG
Mourgos                   ST_MAN
Ernst                     IT_PROG
Fay                       MK_REP
Kumar                     SA_REP
Banda                     SA_REP
```