Subject: Advance Database Management System

Subject Code: BCA(302)

UNIT-2

Present By: Smit Trivedi(Asst.Professor,Idar BCA/PGDCA)

Introduction to SQL

✓SQL stands for structure query language .

✓SQL is a language that provide and interface to relational database management System .

✓SQL developed by IBM in 1970.

✓SQL is the standard language used to manipulate and retrieve data from relational Database.

Feature of SQL

1. SQL is easy to learn.

2. SQL is used to access data from relational database management systems.

3. SQL can execute queries against the database.

4. SQL is used to describe the data.

5. SQL is used to define the data in the database and manipulate it when needed.

6. SQL is used to create and drop the database and table.

7. SQL is used to create a view, stored procedure, function in a database.

8. SQL allows users to set permissions on tables, procedures, and views.

Rules Of SQL

1. Only Use Lowercase Letters, Numbers, and Underscores. ..

2. Use Simple, Descriptive Column Names. ...

3. Use Simple, Descriptive Table Names. ...

4. Have an Integer Primary Key. ...

5. Be Consistent with Foreign Keys. ...

6. Store Date times as Date times. ...

Component of SQL

There are four types of component of SQL.

1. DDL (Data Definition Language)

2. DML (Data Manipulation Language)

3. DCL (Data Control Language)

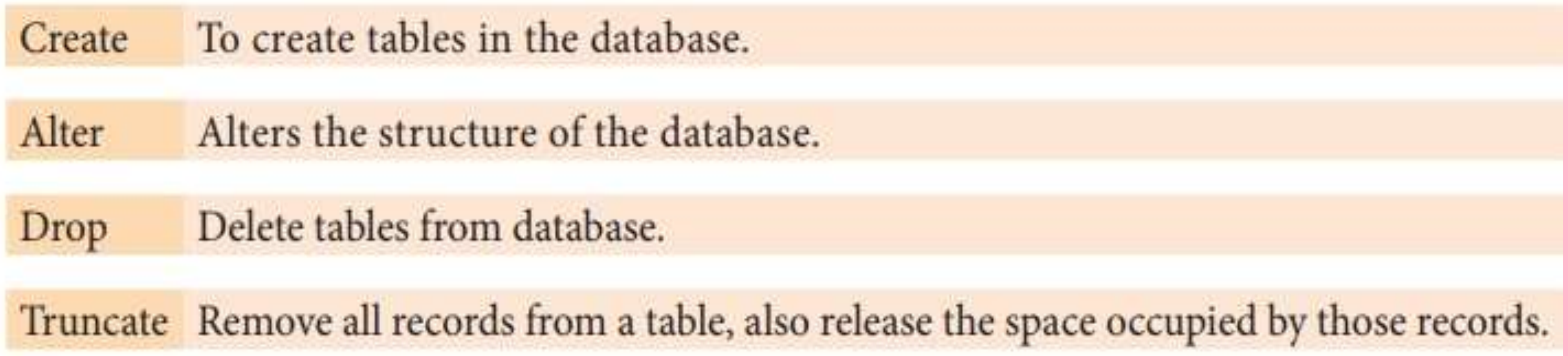
4. DQL(Data Query Language)

1. DDL (DATA DEFINITION LANGUAGE)

→The Data Definition Language (DDL) consist of SQL statements used to define the database structure or schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in databases.

→The DDL provides a set of definitions to specify the storage structure and access methods used by the database system.

SQL commands which comes under Data Definition Language are:



Create	To create tables in the database.
Alter	Alters the structure of the database.
Drop	Delete tables from database.
Truncate	Remove all records from a table, also release the space occupied by those records.

2. DML (DATA MANIPULATION LANGUAGE)

→**A Data Manipulation Language (DML)** is a computer programming language used for adding (inserting), removing (deleting), and modifying (updating) data in a database.

→In SQL, the data manipulation language comprises the SQL-data change statements, which modify stored data but not the schema of the database table.

SQL commands which comes under Data Manipulation Language are :

Insert : Inserts data into a table
Update : Updates the existing data within a table.
Delete : Deletes all records from a table, but not the space occupied by them.

3. DCL (DATA CONTROL LANGUAGE)

→A **Data Control Language (DCL)** is a programming language used to control the access of data stored in a database.

→It is used for controlling privileges in the database (Authorization). The privileges are required for performing all the database operations such as creating sequences, views of tables etc.

SQL commands which come under Data Control Language are:

Grant : Grants permission to one or more users to perform specific tasks.

Revoke : Withdraws the access permission given by the GRANT statement.

4. DQL (DATA QUERY LANGUAGE)

→The Data Query Language consist of commands used to query or retrieve data from a database.
→One such SQL command in Data Query Language is

Select : It displays the records from the table.

❖**Basic Datatypes:-**

1.Char
2.Varchar/Varchar2
3.Date
4.Number
5.Long
6.Row/Long Row

1.Char:-
→It is used to store char string value of fixed length.
→This data type can hold 255 Char.

2. Varchar/Varchar2:-
→It is used to store variable length alpha numeric data.
→This data type can hold up to 4000 Char.

3.Date:-
→It is used to store Date & Time.
→The standard format is DD/MON/YY.
→ Ex: DD/MON/YY➔27/JUN/15.

4.Number:-

→It is used to store number.

→The maximum number store up to 38 digits.

5.Long:

→it is used to store variable length char string containing up to 2 GB.

6.Row/Long Row:-

→it is used to store binary data such as digital picture or image.

Create Table Command

→The CREATE TABLE statement is used to create a new table in a database.

→ **Syntax:-**

```
CREATE TABLE table name
  (  column1 data type(size),
    column2 data type(size),
    column3 data type(size),
   ....
);
```

→Example creates a table called “Student" that contains five columns: No, Name, Address, and City:

→**Example:-**

```
CREATE TABLE Student
   ( No number(25),
    Name varchar(25),
    Address varchar(25),
    City varchar(25)
);
```

The Table is Created.

Insert data into Table

→Syntax:-

INSERT INTO *table name* (*column1*, *column2*, *column3*, ...)
VALUES (*value1*, *value2*, *value3*, ...);

Example:-

INSERT INTO Student (No,Name,Address,City)
VALUES(101 , ‘Richa’ , ‘Modasa’ , ‘Modasa’);

One Row is Created.

OR

INSERT INTO *Student*
VALUES(101 , ‘Richa’ , ‘Modasa’ , ‘Modasa’);

One Row is Created.

Viewing Data

Example:-

INSERT INTO Student (No,Name,Address,City)
VALUES(101 , ‘Richa’ , ‘Modasa’ , ‘Modasa’);

One Row is Created.

VIEWING DATA :

Select * from Student;

Filtering Table Data

→The **Three** Way of Filtering Table data:

1)Selected column and All Rows
2) Selected Rows and All Columns
3) Selected Rows and Selected Columns

1.Selected Columns and All Rows:

→Syntax:

Select <column name1>,<column name 2> from <table name>;

→Example:

Select N0,Name from Student;

2.Selected Row and All Columns:

→If information of a particular line is to be retrieved data from a table its retrieve must be based on specific condition.

❖Where Clause:

→The WHERE clause is used to filter records.

→The WHERE clause is used to extract only those records that fulfill a specified condition.

→Syntax:

Select * from <table name> where <condition>;

→Example:

```
SELECT * FROM Customers
WHERE Country='Mexico';
```

3.Selected Rows and Selected Columns:

→Syntax:

Select <column name1><column name2>from <table name> where <condition>;

→Example:

Select No, Name from Student where name='Riva';

❖Sorting data in table

→The ORDER BY statement in sql is used to sort the fetched data in either ascending or descending according to one or more columns.

→By default ORDER BY sorts the data in **ascending order.**

→We can use the keyword DESC to sort the data in descending order and the keyword ASC to sort in ascending order.

→Syntax:

```
SELECT * FROM table_name ORDER BY column_name ASC|DESC
```

→Example:

```
SELECT * FROM Student ORDER BY ROLL_NO DESC;
```

❖Delete operation

The SQL DELETE Query is used to delete the existing records from a table.

You can use the WHERE clause with a DELETE query to delete the selected rows, otherwise all the records would be deleted.

Syntax

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

Example

```
SQL> DELETE FROM CUSTOMERS WHERE ID = 6;
```

❖Update operation

The SQL **UPDATE** Query is used to modify the existing records in a table.
You can use the WHERE clause with the UPDATE query to update the selected rows,
otherwise all the rows would be affected.
Syntax

```
UPDATE table_name SET column1 = value1, column2 = value2...., columnN =
valueN WHERE [condition];
```

example

```
SQL> UPDATE CUSTOMERS SET ADDRESS = 'Pune' WHERE ID = 6;
```

❖Modify structure of table

The SQL ALTER **TABLE** command is used to **change the structure** of an existing **table**.

1.Add new column
2.Drop column
3. Modify column

1.Add new column

Syntax:

ALTER TABLE <table name> **ADD <column name>** <Data type>(<size>);

Example:

ALTER TABLE vehicles **ADD model** VARCHAR(100) **NOT** NULL;

2.Drop column

Syntax:

ALTER TABLE <table name> **drop column <column name>** ;

Example

ALTER TABLE vehicles **drop model**;

2.Modify column

Syntax:

ALTER TABLE <table name> **modify (<column name> <new datatype>(<new size>)**);

Example

ALTER TABLE vehicles **modify (model number(10))**;

TRUNCATE TABLE command

The SQL **TRUNCATE TABLE** command is used to delete complete data from an existing table.

You can also use DROP TABLE command to delete complete table but it would remove complete table structure form the database and you would need to re-create this table once again if you wish you store some data.

Syntax

```
TRUNCATE TABLE table_name;
```

Example

```
SQL > TRUNCATE TABLE CUSTOMERS;
```

The SQL DROP TABLE Statement

The DROP TABLE statement is used to drop an existing table in a database.

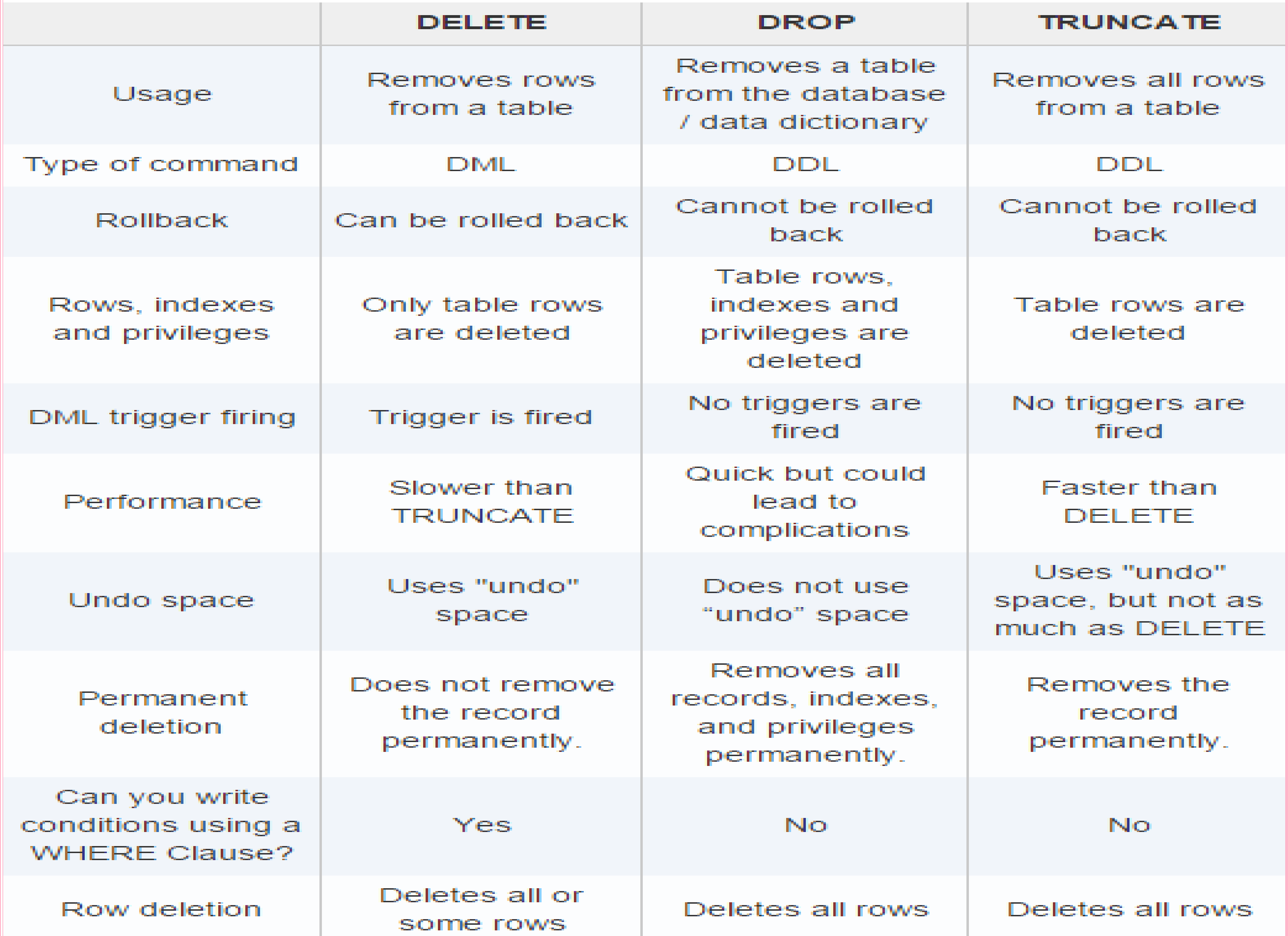
Syntax

DROP TABLE *table_name*;

Example:

DROP TABLE *Student*;

difference between truncate and delete and drop



	DELETE	DROP	TRUNCATE
Usage	Removes rows from a table	Removes a table from the database / data dictionary	Removes all rows from a table
Type of command	DML	DDL	DDL
Rollback	Can be rolled back	Cannot be rolled back	Cannot be rolled back
Rows, indexes and privileges	Only table rows are deleted	Table rows, indexes and privileges are deleted	Table rows are deleted
DML trigger firing	Trigger is fired	No triggers are fired	No triggers are fired
Performance	Slower than TRUNCATE	Quick but could lead to complications	Faster than DELETE
Undo space	Uses "undo" space	Does not use "undo" space	Uses "undo" space, but not as much as DELETE
Permanent deletion	Does not remove the record permanently.	Removes all records, indexes, and privileges permanently.	Removes the record permanently.
Can you write conditions using a WHERE Clause?	Yes	No	No
Row deletion	Deletes all or some rows	Deletes all rows	Deletes all rows

Data Constraints

1. I/o Constraints
2. Business Rules constraint

1.I/o Constraints

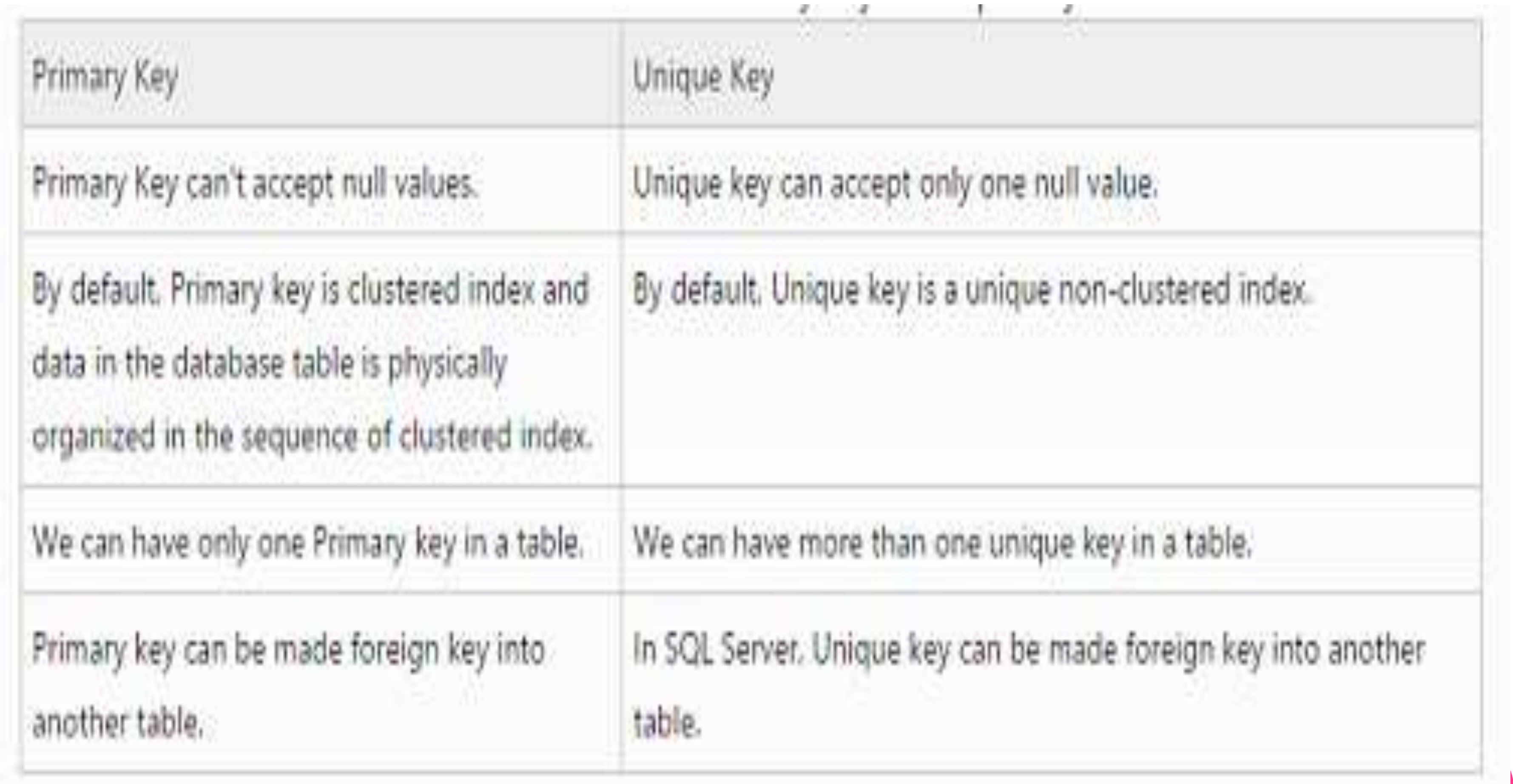
→Data constraints are used to specify rules for the data in a table. Constraints are used to limit the type of data that can go into a table.

→This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

→Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

→The following constraints are commonly used in SQL:
NOT NULL - Ensures that a column cannot have a NULL value
UNIQUE - Ensures that all values in a column are different
PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
FOREIGN KEY - Uniquely identifies a row/record in another table
CHECK - Ensures that all values in a column satisfies a specific condition
DEFAULT - Sets a default value for a column when no value is specified
INDEX - Used to create and retrieve data from the database very quickly

Difference between primary key and unique key



Primary Key	Unique Key
Primary Key can't accept null values.	Unique key can accept only one null value.
By default, Primary key is clustered index and data in the database table is physically organized in the sequence of clustered index.	By default, Unique key is a unique non-clustered index.
We can have only one Primary key in a table.	We can have more than one unique key in a table.
Primary key can be made foreign key into another table.	In SQL Server, Unique key can be made foreign key into another table.

2. Business Rules constraint

→A constraint business rule is used to manage multiple constraints on tables and columns in a database for the DBMS that support this feature.

→During generation, the variable %RULES% is evaluated in order to generate a single constraint in the SQL script.

1.check constraint

A check constraint in SQL Server (Transact-SQL) allows you to specify a condition on each row in a table.

Note

a. A check constraint can NOT be defined on a SQL View.
b. The check constraint defined on a table must refer to only columns in that table. It can not refer to columns in other tables.
c. A check constraint can NOT include a Subquery.
d. A check constraint can be defined in either a CREATE TABLE statement or a ALTER TABLE statement.

2. NOT NULL Constraint

→By default, a column can hold NULL values.

→The NOT NULL constraint enforces a column to NOT accept NULL values.
This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

Null Value Concept

→The **SQL NULL** is the term used to represent a missing **value**.

→A **NULL value** in a table is a **value** in a field that appears to be blank.

→ A field with a **NULL value** is a field with no **value**. It is very important to understand that a **NULL value** is different than a zero **value** or a field that contains spaces.

→**Null** (or NULL) is a special marker used in Structured Query Language to indicate that a data value does not exist in the database.

→A null value evaluate to null is any expression.

→Example:

Null * 10=null

Operators

1.Arithmatic Operator
2.Logical Operator
3.Relational Operator

1.Arithmetic Operator: Arithmetic Operator are used to perform Arithmetic operation like Addition, Substraction , Multiplication, Division etc.

Operator	Description
+	To Perform Addition of two No. (a+b)
-	To Perform Substraction of two No. (a-b)
*	To Perform Multiplication of two No. (a*b)
/	To Perform Division of two No. (a/b)
**	To Perform Exponentiation of two No. x^n

2.Logical Operator:-Logical Operator re used to Combine two or more relation Expression.

Operator	Description
AND	TRUE if all the conditions separated by AND is TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
NOT	Displays a record if the condition(s) is NOT TRUE

3.Relational Operator: it is used to perform comparison between two operands.

→you can divide **Relational Operator** in following category:

1.Single value comparison
2.Pattern Matching
3.Range Searching
4.List Searching

1. Single value comparison:

Operator	Description
>	Greater than.
<	Less than.
>=	Greater than equal to
<=	Less than equal to
=	Equal to.
<>	Not equal to

2.Pattern Matching:

❖Like Predicates:

→The like predicates allow comparison of one string value with another string value,which is not iditintical.

→This is achieve by using wild card characters.

→Two wild card Characters are available in oracle:

1)% (Percentage)
2)_ (Underscore)

1. **% (Percentage):**it allows to match any string of any length(including 0 length).

Example:

```
select * from Student where name like 'da%';
```

2)_ (Underscore):it allows to match on a single char.

Example:

```
select * from Student where name like 'da_';
```

3.Range Searching:

→he **range searching** problem most generally consists of preprocessing a set *S* of objects, in order to determine which objects from *S* intersect with a query object, called a *range*.

→For example, if *S* is a set of points corresponding to the coordinates of several cities, a geometric variant of the problem is to find cities within a certain latitude and longitude range.

4.List Searching: The relational operator compare a single value to another single value.

Oracle function

1.Group function
2.Scaler function

<u>1.Group function: </u>function that act on set of value are called group function.

I) **<u>AVG(): </u>**The Oracle AVG() function accepts a list of values and returns the average.

Syntax:

AVG([DISTINCT | ALL] expression)

example:

Select AVG((balance)”average balance” from Account;

2.MIN(): return a minimum value of expression.

Syntax:

MIN([DISTINCT | ALL] expression)

example:

Select MIN((balance)"minimum balance" from Account;

3.MAX(): return a maximum value of expression.

Syntax:

MAX([DISTINCT | ALL] expression)

example:

Select MAX((balance)"maximum balance" from Account;

4.SUM(): return a sum of value n.

Syntax:

SUM([DISTINCT | ALL] n)

example:

Select SUM((balance)"totalbalance" from Account;

5.COUNT(exp): Counts the number of row where exp is not null.

Syntax:

COUNT([DISTINCT | ALL] exp)

example:

Select COUNT((balance)"Number of Account" from Account;

6.COUNT(*): return the number of row in the table including duplicate and null.

Syntax:

COUNT(*);

example:

Select COUNT((*)" Number of account" from Account;

2.Scalar function (single row function)

Function that act no only one value at a time are called Scalar function.

1. Numeric function
2. String function
3. Conversion function
4. Date function

1. Numeric function

A. ABS(): The ABS() function is used to calculate the absolute value of an expression.

Syntax:

```
ABS(N);
```

Example

```
SELECT ABS(5) FROM dual;
```

Output

```
ABS(5)
----------
5
```

B.Round(): returns n rounded to m. m places to the right of a decimal points.

Syntax:

Round(n,[m]);

Example

SELECT Round(15.19,1) “rounded” from dual;

Output

```
Rounded
----------
15,2
```

C.Power(): returns m raised to the n. the power n must be an integer.

Syntax:

power(m,n)

Example

SELECT power(3,2) “raised” from dual;

Output

```
Raised
----------
9
```

D.SQRT(): returns SQRT of n.

Syntax:

SQRT(N)

Example

SELECT SQRT(25) “square root” from dual;

Output

```
square
----------
5
```

E. Exp(): returns e raised to the n the power where e = 2.71828153

Syntax:

exp(n);

Example

SELECT exp(5) “exponent” from dual;

Output

```
exponent
----------
148.413159
```

F. GREATEST(): It returns the greatest value in a list of expressions.

Syntax: `SELECT GREATEST(30, 2, 36, 81, 125);`

Output: 125

G. LEAST(): It returns the smallest value in a list of expressions.

Syntax: `SELECT LEAST(30, 2, 36, 81, 125);`

Output: 2

H.MOD(): It returns the remainder of n divided by m.

Syntax: `SELECT MOD(18, 4);`

Output: 2

I.TRUNCATE(): This doesn't work for SQL Server.
It returns 7.53635 truncated to 2 places right of the decimal point.

Syntax: `SELECT TRUNCATE(7.53635, 2);`

Output: 7.53

J. FLOOR(): It returns the largest integer value that is less than or equal to a number.

Syntax: `SELECT FLOOR(25.75);`

Output: 25

K. CEIL(): It returns the smallest integer value that is greater than or equal to a number.

Syntax: `SELECT CEIL(25.75);`

Output: 26

2.String function

LOWER(): This function is used to convert the upper case string into lower case.

Syntax: `SELECT LOWER('GEEKSFORGEEKS.ORG');`
Output: `geeksforgeeks.org`

UCASE(): This function is used to make the string in upper case.

Syntax: `UCASE ("GeeksForGeeks");`

Output: `GEEKSFORGEEKS`

SUBSTR(): This function is used to find a sub string from the a string from the given position.

Syntax:`SUBSTR('geeksforgeeks', 1, 5);`

Output: ‘geeks’

ASCII(): This function is used to find the ASCII value of a character.

Syntax: `SELECT ascii('t');`

Output: 116

INSTR(): This function is used to find the occurrence of an alphabet.

Syntax: `INSTR('geeks for geeks', 'e');`

Output: 2 (the first occurrence of ‘e’)

LENGTH(): This function is used to find the length of a word.

Syntax: `LENGTH('GeeksForGeeks');`

Output: 13

LTRIM(): This function is used to cut the given sub string from the original string.**S**

yntax: `LTRIM('123123geeks', '123');`

Output: geeks

RTRIM(): This function is used to cut the given sub string from the original string.

Syntax: `RTRIM('geeksxyxzyyy', 'xyz');`

Output: ‘geeks’

TRIM(): This function is used to cut the given symbol from the string.

Syntax: `TRIM(LEADING '0' FROM '000123');`

Output: 123

LPAD(): This function is used to make the given string of the given size by adding the given symbol.

Syntax: `LPAD('geeks', 8, '0');`

Output: 000geeks

RPAD(): This function is used to make the given string as long as the given size by adding the given symbol on the right.

Syntax: `RPAD('geeks', 8, '0');`

Output: `‘geeks000’`

3.Conversion function

TO_NUMBER function

The TO_NUMBER function converts a character value to a numeric datatype.
 If the string being converted contains nonnumeric characters, the function returns an error.

Syntax

```
TO_NUMBER (string1, [format], [nls_parameter])
```

Output:

```
TO_NUMBER('1210.73','9999.99')
 ------------------------------
1210.73
```

TO_CHAR function (Numeric conversion)

TO_CHAR function is used to typecast a numeric or date input to character type with a format model (optional).

Syntax

```
TO_CHAR(number1, [format], [nls_parameter])
```

TO_CHAR (salary, '$99999.99') Salary

Output:
99999.99

TO_DATE function

The function takes character values as input and returns formatted date equivalent of the same.

The TO_DATE function allows users to enter a date in any format, and then it converts the entry into the default format used by Oracle 11g.

Syntax:

```
TO_DATE( string1, [ format_mask ], [ nls_language ] )
```

```
Example:
TO_DATE ('06-aug-2020','DD-MON-YY' from dual;

Output:

06-aug-20
```

4. Date Function:

1) Add_Months (d,n):

→Adds *n* months to date *d*.

Example:

Add_Months (‘06-Aug-2020’,4);

Output:

06-Dec-2020

2) **Last_Day** (d):

→Returns date of the last day of the month containing date *d*.

Example:

Last_Day (‘06-Aug-2020’);

Output:

31-Aug-2020

3) Next_Day :

→Returns the date of the first weekday *s* after date *d*. If s is omitted, add one day to *d*.

Example:

```
Next_Day (‘16/12/99',"Monday")
```

Output:

```
21/12/99
```

4) **Months_Between** (d1,d2):

→Returns the number of months between dates *d1* and *d2* as a real number (fractional value).

Example:

```
Months_Between (’06-Aug-2020’,’06-July-2020’);
```

Output:

```
1
```

GROUP BY clause

→The **GROUP BY clause** is a **SQL** command that is used to **group** rows that have the same values.

→The **GROUP BY clause** is used in the SELECT statement .

→Optionally it is used in conjunction with aggregate functions to produce summary reports from the database. That's what it does, summarizing data from the database.

→ The queries that contain the GROUP BY clause are called grouped queries and only return a single row for every grouped item.

→**GROUP BY Syntax:**

SELECT statements... GROUP BY column_name1[,column_name2,...] [HAVING condition];

Grouping using a Single Column:

→In order to help understand the effect of Group By clause, let's execute a simple query that retur
all the gender entries from the members table.

```
SELECT `gender` FROM `members` ;
```

```
gender
Female
Female
Male
Female
Male
Male
Male
Male
Male
```

Suppose we want to get the unique values for genders. We can use a following query

```
SELECT `gender` FROM `members` GROUP BY `gender`;
```

```
gender
Female
Male
```

HAVING clause

→It's not always that we will want to perform groupings on all the data in a given table.

→There will be times when we will want to restrict our results to a certain given criteria.

→In such cases , we can use the HAVING clause

Suppose we want to know all the release years for movie category id 8.

```
SELECT * FROM `movies` GROUP BY `category_id`,`year_released` HAVING `category_id` = 8;
```

movie_id	title	director	year_released	category_id
9	Honey mooners	John Schultz	2005	8
5	Daddy's Little Girls	NULL	2007	8

Note only movies with category id 8 have been affected by our GROUP BY clause.

❖ GROUP BY clause with ROLLUP operator:

→GROUP BY ROLLUP is an extension of the GROUP BY clause that produces sub-total rows (in addition to the grouped rows). Sub-total rows are rows that further aggregate whose values are derived by computing the same aggregate functions that were used to produce the grouped rows.

→You can think of rollup as generating multiple result sets, each of which (after the first) is the aggregate of the previous result set. So, for example, if you own a chain of retail stores, you might want to see the profit for:

→Each store.
→Each city (large cities might have multiple stores).
→Each state.
→Everything (all stores in all states).

→You could create separate reports to get that information, but it is more efficient to scan the data once.

❖ GROUP BY clause with CUBE operator:

→he **CUBE operator** is also used in combination with the **GROUP BY clause**, however the **CUBE operator** produces results by generating all combinations of columns specified in the **GROUP** BY **CUBE clause**.

→Similar to the ROLLUP, CUBE is an extension of the GROUP BY clause. CUBE allows you to generate subtotals like the ROLLUP extension.

→In addition, the CUBE extension will generate subtotals for all combinations of grouping columns specified in the GROUP BY clause.

→**Syntax:**

SELECT c1, c2, AGGREGATE_FUNCTION(c3) **FROM** table_name **GROUP BY CUBE**(c1 , c2);

→**Example:**

SELECT warehouse, **SUM**(quantity) **FROM** inventory **GROUP BY** warehouse;

<u>Sub Query</u>

→**”A Sub Query is a query within a Query.”**

→A Sub query or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

→A sub query is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

→A sub query cannot be immediately enclosed in a set function.

→Types of **Sub Query:**

1.Single Row Sub Query
2. Multi Row Sub Query
3. Multi Column Sub Query

1.Single Row Sub Query:

→You can place a sub query in the WHERE clause of another query.

→Let's take an example of a query that contains a sub query placed in it's WHERE clause.

→**Example:**

```
SELECT agent_code FROM agents WHERE agent_name = 'Alex';
```

2. Multi Row Sub Query:

→Multiple row subquery returns one or more **rows** to the outer **SQL** statement.

→You may use the IN, ANY, or ALL operator in outer query to handle a **subquery** returns **multiple rows**.

→Suppose you want to identify items that are low in stock, while also identifying orders for those items.

→You could execute a SELECT statement containing a subquery in the WHERE clause, similar to the following:

```
SELECT * FROM SalesOrderItems WHERE ProductID IN ( SELECT ID FROM Products
 WHERE Quantity < 20 ) ORDER BY ShipDate DESC;
```

3) Multiple-Column Sub query:

→If you want compare two or more columns. you must write a compound WHERE clause using logical operators Multiple-column subqueries enable you to combine duplicate WHERE conditions into a single WHERE clause.

Example:

```
SELECT  ordid, prodid, qty
FROM        item
WHERE       (prodid,   qty)   IN
                         (SELECT prodid,   qty
                          FROM       item
                          WHERE      ordid =   365)
AND  ordid   =   365 ;
```

SET Operator

→"Set operator combine the result of multiple query into single query."

→Both query should have same number of column , same data type and same order.

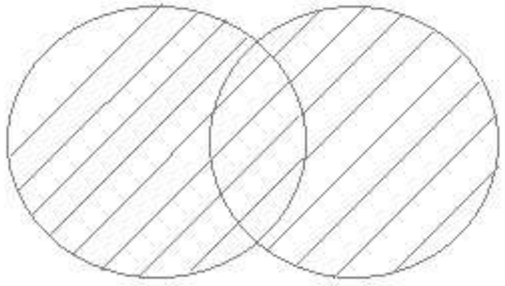
→Types of **Operator**:

1) UNION
2) INTERSECT
3) MINUS

1) UNION:

→UNION is used to combine the results of two or more SELECT statements.

→However it will eliminate duplicate rows from its resultset. In case of union, number of columns and datatype must be same in both the tables, on which UNION operation
is being applied.



→**Example of UNION**

The **First** table,

ID	Name
1	abhi
2	adam

The **Second** table,

ID	Name
2	adam
3	Chester

Union SQL query will be,

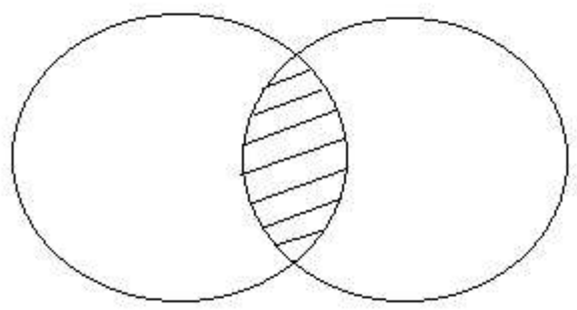
```
SELECT * FROM First UNION SELECT * FROM Second;
```

The **resultset table** will look like

ID	NAME
1	abhi
2	adam
3	Chester

2) INTERSECT:

→ Intersect operation is used to combine two SELECT statements, but it only retuns the records which are common from both SELECT statements. In case of **Intersect** the number of columns and datatype must be same.



Example of Intersect

The **First** table

ID	NAME
1	abhi
2	adam

The **Second** table,

ID	NAME
2	adam
3	Chester

Intersect query will be

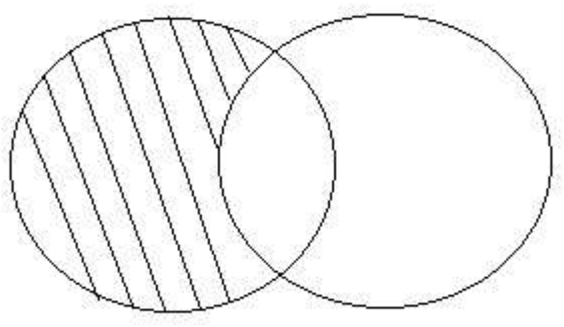
```
SELECT * FROM First INTERSECT SELECT * FROM Second;
```

The **resultset** table will look like

ID	NAME
2	adam

3) MINUS:

→The Minus operation combines results of two SELECT statements and return only those in the final result, which belongs to the first set of the result.



Example of Minus

The **First** table,

ID	NAME
1	abhi
2	adam

The **Second** table,

ID	NAME
2	adam
3	Chester

Minus query will be,

```
SELECT * FROM First MINUS SELECT * FROM Second;
```

The **resultset** table will look like,

ID	NAME
1	abhi

Joins

1)Join Multiple Table
2)Join Table to it Self

1)Join Multiple Table:

→A **JOIN** clause is used to combine rows from **two** or more **tables**, based on a related column between them. Notice that the "CustomerID" column in the "Orders" **table** refers to the "CustomerID" in the "Customers" **table**. The relationship between the **two tables** above is the "CustomerID" column.

❖**Types Of Joins:**

1)Inner Join
2)Outer Join
    i) Left Join
    ii)Right Join
    iii)Full Join
3)Cross Join

1) **Inner Join**: Returns records that have matching values in both tables.

→The most important and frequently used of the joins is the **INNER JOIN**. They are also referred to as an **EQUIJOIN**.

→The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate.

→**Syntax**

```
SELECT table1.column1, table2.column2...
FROM table1
INNER JOIN table2
ON table1.common_field = table2.common_field;
```

Example:

Table 1 – Person Table is as follows

P_Id	Fname	Lname	City
1	Bhut	Malhotra	Ahmedabad
2	Kali	Dharmatma	Ahmedabad
3	Lili	Patel	Modasa
4	Pili	Patel	Modasa

Table 2 – Order Table is as follows

O_Id	Order _No.	P_Id
1	01001	3
2	01002	3
3	01003	2
4	01004	1

Output:

Fname	Lname	Order_no
Bhut	Malhotra	01004
Kali	Dharmatma	01003
Lili	Patel	01001
Pili	Patel	01002

2) Outer Join:

→ In the SQL outer JOIN all the content of the both tables are integrated together either they are matched or not.

i) Left Join: Returns all records from the **left** table, and the matched records from the **right** table.

→ **SQL left outer join** is also known as **SQL left join**. Suppose, we want to **join** two tables: A and B. **SQL left outer join** returns all rows in the **left** table (A) and all the matching rows found in the right table (B). It means the result of the **SQL left join** always contains the rows in the **left** table

→**Syntax:**

```
SELECT column1, column2...
FROM table_A
LEFT JOIN table_B ON join_condition
WHERE row_condition
```

Example:

```
select person.fname,person.lname,orders.order_no from
person left join orders on person.p_id=order.p_id;
```

Output:

Fname	Lname	Order_no
Bhut	Malhotra	01004
Kali	Dharmatma	01003
Lili	Patel	01001
Lili	Patel	01001
Pili	Patel	-

ii)Right Join:

→Returns all records from the **right** table, and the matched records from the **left** table.

Syntax:

SELECT *column_name(s)*
FROM *table1*
RIGHT JOIN *table2*
ON *table1.column_name* = *table2.column_name*;

Example:

Select person.fname,person.lname,order.order_no
from person Right join order on person.p_id=order.p_id;

Output:

Fname	Lname	Order_no
Bhut	Malhotra	01004
Kali	Dharmatma	01003
Lili	Patel	01001
Lili	Patel	01001
-	-	01005

iii) Full Join(Both Table):

→The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

Syntax:

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

Example:

```
Select person.fname,person.lname,order.order_no
from person full join order on person.p_id=order.p_id;
```

Output:

Fname	Lname	Order_no
Bhut	Malhotra	01004
Kali	Dharmatma	01003
Lili	Patel	01001
Lili	Patel	01001
Pili	Patel	-
-	-	01005

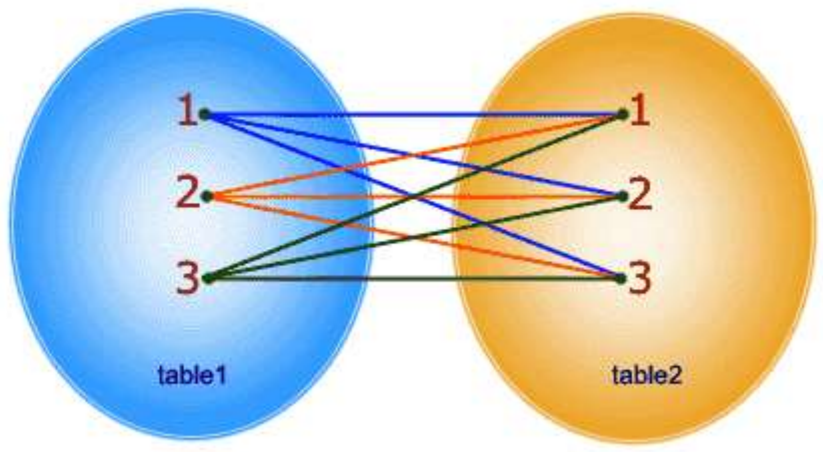
3) Cross Join:

→In **SQL**, the **CROSS JOIN** is used to combine each row of the first table with each row of the second table. It is also known as the Cartesian **join** since it returns the Cartesian product of the sets of rows from the joined tables.

Syntax:

```
SELECT * FROM table1 CROSS JOIN table2;
```

SELECT * FROM table1 CROSS JOIN table2;



In CROSS JOIN, each row from 1st table joins with all the rows of another table.
If 1st table contain x rows and y rows in 2nd one the result set will be x * y rows.

Example:

```
Select * from person cross join order;
```

Output:(Page Behind)

P_Id	Fname	Lname	O_Id	Order _No.	P_Id
1	Bhut	Malhotra	1	01001	3
2	Kali	Dharmatma	1	01001	3
3	Lili	Patel	1	01001	3
4	Pili	Patel	1	01001	3
1	Bhut	Malhotra	2	01002	3
2	Kali	Dharmatma	2	01002	3
3	Lili	Patel	2	01002	3
4	Pili	Patel	2	01002	3
1	Bhut	Malhotra	3	01003	2
2	Kali	Dharmatma	3	01003	2
3	Lili	Patel	3	01003	2
4	Pili	Patel	3	01003	2
1	Bhut	Malhotra	4	01004	1
2	Kali	Dharmatma	4	01004	1
3	Lili	Patel	4	01004	1
4	Pili	Patel	4	01004	1

2) Join Table to it Self(Self Join) :

→A **self join** uses the inner **join** or left **join** clause. Because the query that uses **self join** references the same **table**, the **table** alias is used to assign different names to the same **table** within the query. Note that referencing the same **table** more than one in a query without using **table** aliases will result in an error.

→A self JOIN is a regular join, but the table is joined with itself.

→**Syntax:**

SELECT *column_name(s)*
FROM *table1 T1, table1 T2*
WHERE *condition*;

→**Example:**

select a.fname,b.lname from person a person b where a.p_id=b.p_id;

Output:

P_Id	Fname	Lname
1	Bhut	Malhotra
2	Bhut	Malhotra
3	Bhut	Malhotra
4	Bhut	Malhotra
1	Kali	Dharmatma
2	Kali	Dharmatma
3	Kali	Dharmatma
4	Kali	Dharmatma
1	Lili	Patel
2	Lili	Patel
3	Lili	Patel
4	Lili	Patel
1	Pili	Patel
2	Pili	Patel
3	Pili	Patel
4	Pili	Patel

Important Que Unit-2

1.Introduction to SQL.

2.Features of SQL.

3.Components of SQL.

4.Explain Data Constraint in Detail.

5.Define:Null value Concept.

6.Explain Operators in Detail.

7.Explain Oracle function in Detail.

8.Expalin Group By Clause.

9.Define:Sub Query

10.Define:Set Operator

11.Explain Joins in Detail.