

UNIT - 3

UNIT-3

THE DATABASE LANGUAGE SQL

Introduction to SQL:

What is SQL?

1. SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in relational database.
2. SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, and Oracle, Sybase, Informix, postgres and SQL Server use SQL as standard database language.

Why SQL?

3. Allows users to access data in relational database management systems.
4. Allows users to describe the data.
5. Allows users to define the data in database and manipulate that data.
6. Allows embedding within other languages using SQL modules, libraries & pre-compilers.
7. Allows users to create and drop databases and tables.
8. Allows users to create view, stored procedure, functions in a database.
9. Allows users to set permissions on tables, procedures and views

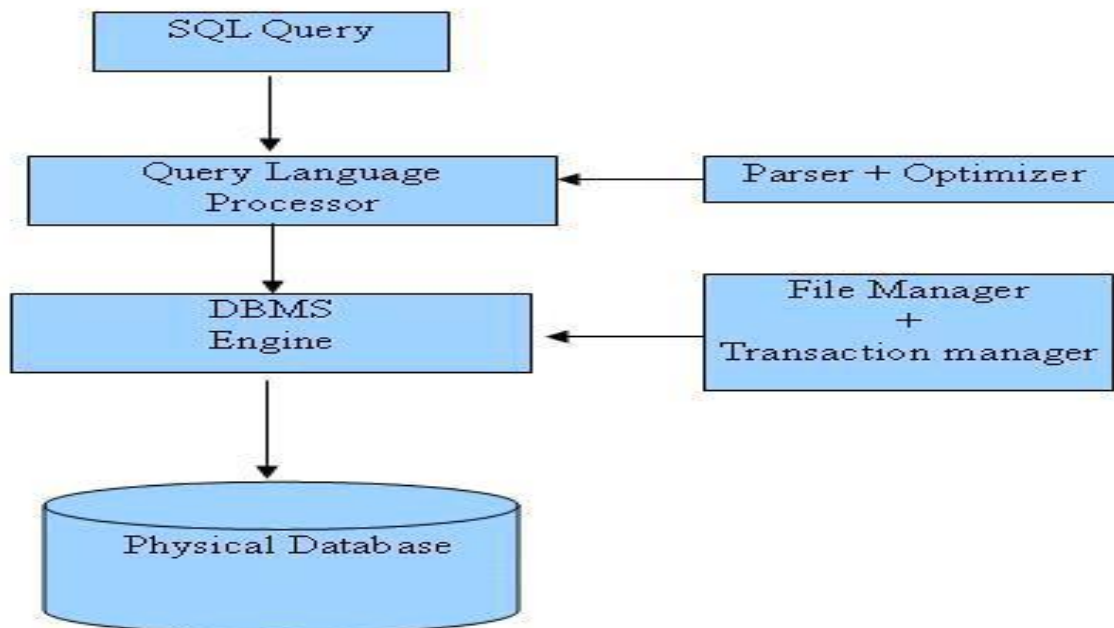
History:

10. **1970** -- Dr. E. F. "Ted" of IBM is known as the father of relational databases. He described a relational model for databases.
11. **1974** -- Structured Query Language appeared.
12. **1978** -- IBM worked to develop Codd's ideas and released a product named System/R.
13. **1986** -- IBM developed the first prototype of relational database and standardized by ANSI. The first relational database was released by Relational Software and its later becoming Oracle.

SQL Process:

14. When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.
15. There are various components included in the process. These components are Query Dispatcher, Optimization Engines, Classic Query Engine and SQL Query Engine, etc. Classic query engine handles all non-SQL queries, but SQL query engine won't handle logical files.

SQL Process:



SQL Commands:

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into groups based on their nature. They are:

DDL Commands

DML Commands

DCL Commands

DRL/DQL Commands

TCL Commands

Data Definition Language (DDL) Commands:

Command	Description
CREATE	Creates a new table, a view of a table, or other object in database
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other object in the database.
TRUNCATE	Truncates the table values without delete table structure

Data Manipulation Language (DML) Commands:

Command	Description
INSERT	Creates a record
UPDATE	Modifies records
DELETE	Deletes records

Data Control Language (DCL) Commands:

Command	Description
GRANT	Gives a privilege to user
REVOKE	Takes back privileges granted from user

Data Query Language (DQL) Commands:

Command	Description
SELECT	Retrieves certain records from one or more tables

Transaction Control Language (TCL) Commands:

Command	Description
commit	Save work done
Save point	Identify a point in a transaction to which we can later roll back.
Roll backs	Restore database to original since the last Commit

What is Query?

- A query is a question.
- A query is formulated for a relation/table to retrieve some useful information from the table.
- Different query languages are used to frame queries.

Form of Basic SQL Query

- The basic form of an SQL query is as follows:
SELECT [DISTINCT] select-list (List of Attributes)
FROM from-list (Table (s) Name (s))

WHERE qualification (Condition)

- This SELECT command is used to retrieve the data from the database.

- For retrieving the data every query must have SELECT clause, which specifies what columns to be selected.
- And FROM clause, which specifies the table's names. The WHERE clause, specifies the selection condition.
- **SELECT:** The SELECT list is list of column names of tables named in the FROM list. Column names can be prefixed by a range variable.
- **FROM:** The FROM list in the FROM clause is a list of table names. A Table name can be followed by a range variable. A range variable is particularly useful when the same table name appears more than once in the from-list.
- **WHERE:** The qualification in the WHERE clause is a Boolean combination (i.e., an expression using the logical connectives AND, OR, and NOT) of conditions of the form expression op expression, where op is one of the comparison operators {<, <=, =, >, >=, >}.
- **An expression is a column name, a constant, or an (arithmetic or string) expression.**
- **DISTINCT:** The DISTINCT keyword is used to display the unique tuple or eliminated the duplicate tuple.
- This DISTINCT keyword is Optional.

DDL Commands:

- The following are the DDL commands. They are:
 - Create
 - Alter
 - Truncate
 - Drop

CREATE:

- The SQL CREATE TABLE statement is used to create a new table.
- Creating a basic table involves naming the table and defining its columns and each column's data type.

Syntax:

- Basic syntax of CREATE TABLE statement is as follows:

CREATE TABLE table name (column1 datatype (size), column2 datatype (size), column3 datatype (size) ... columnN datatype (size), PRIMARY KEY (one or more columns));

Example:

SQL> create table customers (id number (10) not null, name varchar2 (20) not null, age number (5) not null, address char (25), salary decimal (8, 2), primary key (id));

ALTER:

- SQL ALTER TABLE command is used to add, delete or modify columns in an existing table

Syntax:

- The basic syntax of ALTER TABLE to add a new column in an existing table is as follows:

ALTER TABLE table_name ADD column_name datatype;

EX: ALTER TABLE CUSTOMERS ADD phno number (12);

- ii) The basic syntax of ALTER TABLE to **DROP COLUMN** in an existing table is as follows:

ALTER TABLE table_name DROP COLUMN column_name;

EX: ALTER TABLE CUSTOMERS DROP column phno;

- The basic syntax of ALTER TABLE to change the **DATA TYPE** of a column in a table is as follows:

ALTER TABLE table_name MODIFY COLUMN column_name datatype;

Ex: ALTER TABLE customer MODIFY COLUMN phno number(12);

- The basic syntax of ALTER TABLE to add a **NOT NULL** constraint to a column in a table is as follows:

ALTER TABLE table_name MODIFY column_name datatype NOT NULL;

Ex:

ALTER TABLE customers MODIFY phno number (12); NOT NULL;

- The basic syntax of ALTER TABLE to **ADD PRIMARY KEY** constraint to a table is as follows:

ALTER TABLE table_name ADD PRIMARY KEY (column1, column2...);

Ex:

ALTER TABLE customer ADD PRIMARY KEY (id,phno);

TRUNCATE:

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

Syntax:

The basic syntax of **TRUNCATE TABLE** is as follows:

TRUNCATE TABLE table name;

EX:

TRUNCATE TABLE student;

SELECT * FROM student;

Empty set (0.00 sec).

DROP:

SQL DROP TABLE statement is used to remove a table definition and all data, indexes, triggers, constraints, and permission specifications for that table.

Syntax:

Basic syntax of DROP TABLE statement is as follows:

DROP TABLE table_name;

EX: DROP TABLE student;

DML Commands:

The following are the DML commands. They are:

- Insert
- Update
- Delete

INSERT:

- SQL INSERT INTO Statement is used to add new rows of data to a table in the database.
- There are two basic syntaxes of INSERT INTO statement as follows:

Syntax1:

INSERT INTO TABLE_NAME (column1, column2, column3,...columnN) VALUES (value1, value2, value3,...valueN);

- Here, column1, column2...columnN are the names of the columns in the table into which you want to insert data.

EX:

```
insert into customers (id,name,age,address,salary) values (1, 'ramesh', 32, 'ahmedabad', 2000);
insert into customers (id,name,age,address,salary) values (2, 'khilan', 25, 'delhi', 1500.00 );
```

2 rows inserted.

Syntax2:

INSERT INTO TABLE_NAME VALUES (value1, value2, value3...valueN);

Ex:

```
insert into customers values (1, 'ramesh', 32, 'ahmedabad', 2000.00 );
```

UPDATE:

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

Syntax:

- The basic syntax of UPDATE query with WHERE clause is as follows:

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

EX:

- UPDATE CUSTOMERS SET ADDRESS = 'Pune' WHERE ID = 6;
- UPDATE CUSTOMERS SET ADDRESS = 'Pune', SALARY = 1000.00;

DELETE:

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

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

Syntax:

The basic syntax of DELETE query with WHERE clause is as follows:

DELETE FROM table_name WHERE [condition];

Ex: DELETE FROM CUSTOMERS WHERE ID = 6;

If you want to DELETE all the records from CUSTOMERS table, you do not need to use WHERE clause and DELETE query would be as follows:

DELETE FROM CUSTOMERS;

DRL/DQL Command:

The select command is comes under DRL/DQL.

SELECT:

SELECT Statement is used to fetch the data from a database table which returns data in the form of result table. These result tables are called result-sets.

Syntax1:

The Following Syntax is used to retrieve specific attributes from the table is as follows:

SELECT column1, column2, columnN FROM table_name;

Here, column1, column2...are the fields of a table whose values you want to fetch.

The Following Syntax is used to retrieve all the attributes from the table is as follows:

SELECT * FROM table_name;

Ex: Select * from student;

Distinct:

- SQL **DISTINCT** keyword is used in conjunction with SELECT statement to eliminate all the

duplicate records and fetching only unique records.

- There may be a situation when you have multiple duplicate records in a table. While fetching such records, it makes more sense to fetch only unique records instead of fetching duplicate records.

Syntax:

- The basic syntax of DISTINCT keyword to eliminate duplicate records is as follows:

SELECT DISTINCT column1, column2,.....columnN FROM table_name WHERE [condition];

Ex: SELECT DISTINCT SALARY FROM CUSTOMERS ORDER BY SALARY;

Queries involving more than one relation (or) Full Relation Operations :

- The following set operations are used to write a query to combine more than one relation. They are:

Union

Intersect

Except

UNION:

- SQL UNION clause/operator is used to combine the results of two or more SELECT statements without returning any duplicate rows.
- To use UNION, each SELECT 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 do not have to be the same length.

Syntax:

- The basic syntax of UNION is as follows:

**SELECT column1 [, column2] FROM table1 [, table2] [WHERE condition]
UNION**

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

EX:

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;

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 UNION apply to the UNION ALL operator.

Syntax:

- The basic syntax of UNION ALL is as follows:

SELECT column1 [, column2] FROM table1 [, table2] [WHERE condition]
UNION ALL

SELECT column1 [, column2] FROM table1 [, table2] [WHERE condition]
EX:

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;

INTERSECT:

- 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 **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];
```

Ex:

```
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;
```

EXCEPT:

- 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 second **SELECT** statement.
- Just as with the **UNION** operator, the same rules apply when using the **EXCEPT** operator.
- MySQL does not support **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];  
EX:
```

```
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;
```

SQL Operators

What is an Operator in SQL?


- An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations.
- Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.

1. Arithmetic operators
2. Comparison operators
3. Logical operators
4. Operators used to negate conditions

SQL Arithmetic Operators:

Operator	Description	Example
+	Addition - Adds values on either side of the operator	a + b will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	a - b will give -10
*	Multiplication - Multiplies values on either side of the operator	a * b will give 200
/	Division - Divides left hand operand by right hand operand	b / a will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	b % a will give 0

SQL Comparison Operators:

<u>Operator</u>	<u>Description</u>	<u>Example</u>
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	 (a = b) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes	(a >= b) is not true

true.

- <= Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. (a <= b) is true.
- !< Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true. (a !< b) is false.
- !> Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true. a !> b) is true.

- The following are example illustrate the relational operators usage on tables:

Ex:

- SELECT * FROM CUSTOMERS WHERE SALARY > 5000;
- SELECT * FROM CUSTOMERS WHERE SALARY = 2000;
- SELECT * FROM CUSTOMERS WHERE SALARY != 2000;
- SELECT * FROM CUSTOMERS WHERE SALARY >= 6500;

SQL Logical Operators:

Operator	Description
AND	The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause
OR	The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
NOT	The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. This is a negation operator

- SQL **AND** and **OR** operators are used to combine multiple conditions to narrow data in an SQL statement. These two operators are called conjunctive operators.
- These operators provide a means to make multiple comparisons with different operators in the same SQL statement.

AND Operator:

- The **AND** operator allows the existence of multiple conditions in an SQL statement's WHERE clause.

Syntax:

- The basic syntax of AND operator with WHERE clause is as follows:

SELECT column1, column2, columnN FROM table_name WHERE [condition1] AND [condition2]...AND [conditionN];

Ex:

SELECT * FROM CUSTOMERS WHERE AGE >= 25 AND SALARY >= 6500;

OR Operator:

- The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.

Syntax:

- The basic syntax of OR operator with WHERE clause is as follows:

~~**SELECT column1, column2, columnN FROM table_name WHERE [condition1] OR [condition2]...OR [conditionN];**~~

Ex:

SELECT * FROM CUSTOMERS WHERE AGE >= 25 OR SALARY >= 6500;

NOT Operator:

- The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.**

Syntax:

SELECT column1, column2, ... column FROM table_name WHERE NOT [condition];

EX:

SELECT * FROM CUSTOMERS WHERE AGE IS NOT NULL;

Special Operators in SQL:

Operator	Description
BETWEEN	The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
EXISTS	The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria.
IN	The IN operator is used to compare a value to a list of literal values that have been specified.
LIKE	
IS NULL	The LIKE operator is used to compare a value to similar values using wildcard operators.
UNIQUE	The NULL operator is used to compare a value with a NULL value.
	The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

LIKE Operator:

- SQL LIKE clause is used to compare a value to similar values using wildcard operators. There are two wildcards used in conjunction with the LIKE operator:
 1. The percent sign (%)
 2. The underscore (_)
- The percent sign represents zero, one, or multiple characters.
- The underscore represents a single number or character.

- The symbols can be used in combinations.

Syntax:

- The basic syntax of % and _ is as follows:

**SELECT FROM table_name
WHERE column LIKE 'XXXX%'**

or

**SELECT FROM table_name
WHERE column LIKE '%XXXX%'**

or

**SELECT FROM table_name
WHERE column LIKE 'XXXX_'**

or

SELECT FROM table_name WHERE column LIKE '_XXXX'

or

SELECT FROM table_name WHERE column LIKE '_XXXX_'

Ex:

Statement

WHERE SALARY LIKE 's%'

WHERE SALARY LIKE '%sad%'

WHERE SALARY LIKE '_00%'

WHERE SALARY LIKE '2_%_ %'

WHERE SALARY LIKE '%r'

WHERE SALARY LIKE '_2%3'

WHERE SALARY LIKE '2__3'

Description

Finds any values that start with s

Finds any values that have sad in any position

Finds any values that have 00 in the second and third positions

Finds any values that start with 2 and are at least 3 characters in length

Finds any values that end with r

Finds any values that have a 2 in the second position and end with a 3

Finds any values in a five-digit number that start with 2 and end with

BETWEEN Operator

The BETWEEN operator is used to select values within a range.

□ **Syntax:**

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

EX: SELECT * FROM Products WHERE Price BETWEEN 10 AND 20;

NOT BETWEEN Operator:

SELECT * FROM Products WHERE Price NOT BETWEEN 10 AND 20;

IN Operator:

- The IN operator allows you to specify multiple values in a WHERE clause.

Syntax

```
SELECT column_name(s)
FROM table_name

WHERE column_name IN (value1,value2,...);
```

Ex: SELECT * FROM Customers WHERE salary IN (5000, 10000);

SQL Joins:

- SQL Joins clause is used to combine records from two or more tables in a database.
- A JOIN is a means for combining fields from two tables by using values common to each.
- Consider the following two tables, CUSTOMERS and ORDERS tables are as follows:

CUSTOMERS TABLE

| 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 |

ORDERS TABLE

| 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 |

Ex:

```
SELECT ID, NAME, AGE, AMOUNT FROM CUSTOMERS,  
ORDERS WHERE CUSTOMERS.ID =  
ORDERS.CUSTOMER_ID;
```

This would produce the following
result: | ID | NAME | AGE |
AMOUNT |

| 3 | kaushik | 23 |
3000 | | 3 | kaushik
| 23 | 1500 | | 2 |
Khilan | 25 | 1560

NOTE:

- Join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal symbol.

SQL Join Types:

- There are different types of joins available in SQL: They are:
 - INNER JOIN
 - OUTER JOIN
 - SELF JOIN
 - CARTESIAN JOIN

INNER JOIN:

- The most frequently used and important 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.
- The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate.
- When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.

Syntax:

- The basic syntax of INNER JOIN is as follows:

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

```
Ex: SELECT ID, NAME, AMOUNT, DATE FROM  
CUSTOMERS INNER JOIN
```

```
_____ORDERS CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

OUTER JOIN:

- The Outer join can be classified into 3 types. They are:
 - Left Outer Join\
 - Right Outer Join
 - Full Outer Join

Left Outer Join:

- The SQL **LEFT JOIN** returns all rows from the left table, even if there are no matches in the right table.
- This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.

Syntax:

- The basic syntax of **LEFT JOIN** is as follows:

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

EX: SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS

LEFT JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;

RIGHT JOIN:

- The SQL **RIGHT JOIN** returns all rows from the right table, even if there are no matches in the left table.
- This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.

Syntax:

- The basic syntax of **RIGHT JOIN** is as follows:

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

Ex: SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS
RIGHT JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;

FULL JOIN:

- The SQL **FULL JOIN** combines the results of both left and right outer joins.
- The joined table will contain all records from both tables, and fill in NULLs for missing matches on either side.

Syntax:

- The basic syntax of **FULL JOIN** is as follows:

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

Ex: SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS FULL JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;

SELF JOIN:

- The SQL **SELF JOIN** is used to join a table to it as if the table were two tables, temporarily renaming at least one table in the SQL statement.

Syntax:

- The basic syntax of **SELF JOIN** is as follows:

SELECT a.column_name, b.column_name...FROM table1 a, table1 b WHERE a.common_field = b.common_field;

Ex:

SELECT a.ID, b.NAME, a.SALARY FROM CUSTOMERS a, CUSTOMERS b WHERE a.SALARY < b.SALARY;

CARTESIAN JOIN:

- The **CARTESIAN JOIN** or **CROSS JOIN** returns the cartesian product of the sets of records from the two or more joined tables.
- Thus, it equates to an inner join where the join-condition always evaluates to True or where the join-condition is absent from the statement.

Syntax:

- The basic syntax of **CROSS JOIN** is as follows:

SELECT table1.column1, table2.column2... FROM table1, table2 [, table3]; **Ex:** SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS, ORDERS;

VIEWS IN SQL:

- 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 kind 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 such 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.

Advantages of views:

- Views provide data security
- Different users can view same data from different perspective in different ways at the same time.
- Views can also be used to include extra/additional information

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];

Ex: CREATE VIEW CUSTOMERS_VIEW AS SELECT name, age FROM CUSTOMERS;

You can query CUSTOMERS_VIEW in similar way as you query an actual table. Following is the example:

SELECT * FROM CUSTOMERS_VIEW;

Updating a View:

A view can be updated under certain conditions: TUTORIALS POINT Simply Easy Learning

- 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 sub queries.
- The query may not contain GROUP BY or HAVING.

NOTE:

So if a view satisfies all the above mentioned rules then you can update a view. Following is an example to update the age of Ramesh:

Ex: UPDATE CUSTOMERS_VIEW SET AGE = 35 WHERE name='Ramesh';

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.

delete from customers_view where age = 22;

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 as given below:

DROP VIEW view_name;

- Following is an example to drop CUSTOMERS_VIEW from CUSTOMERS table:

```
DROP VIEW CUSTOMERS_VIEW;
```

GROUP BY:

- SQL **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups.
- The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

Syntax:

- The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

```
SELECT    column1,  
column2   FROM  
table_name WHERE  
[ conditions ]
```

```
GROUP BY column1,  
column2 ORDER BY  
column1, column2; Ex:
```

```
select name, sum(salary) from customers group by name;
```

ORDER BY:

- SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns.
- Some database sorts query results in ascending order by default.

Syntax:

- The basic syntax of ORDER BY clause is as follows:

```
SELECT  
column-list  
FROM  
table_name
```

**[WHERE
condition]**

**[ORDER BY column1, column2, .. columnN] [ASC |
DESC]; Ex:**

1. select * from customers order by name, salary;
2. select * from customers order by name desc;

HAVING Clause:

- The HAVING clause enables you to specify conditions that filter which group results appear in the final results.
- The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

Syntax:

**SELECT column1,
column2 FROM
table1, table2 WHERE
[conditions]**

**GROUP BY column1,
column2 HAVING
[conditions] ORDER BY
column1, column2; Ex:**

select id, name, age, address, salary from customers group by age having count(age) >= 2;

Aggregate Functions:

- SQL aggregate functions return a single value, calculated from values in a column.
- Useful aggregate functions:
 - AVG()** - Returns the average value
 - COUNT()** - Returns the number of rows
 - MAX()** - Returns the largest value
 - MIN()** - Returns the smallest value
 - SUM()** - Returns the sum

AVG () Function

The AVG () function returns the average value of a numeric column.

AVG () Syntax

**SELECT AVG (column_name) FROM
table_name; Ex:**

SELECT AVG (Price) FROM Products;

COUNT () Function

COUNT aggregate function is used to count the number of rows in a database table.

COUNT () Syntax:

**SELECT COUNT (column_name) FROM
table_name; Ex:**

SELECT COUNT (Price) FROM Products;

MAX () Function

The SQL MAX aggregate function allows us to select the highest (maximum) value for a certain column.

MAX () Syntax:

**SELECT MAX (column_name) FROM
table_name; EX:**

SELECT MAX (SALARY) FROM EMP;

SQL MIN Function:

SQL MIN function is used to find out the record with minimum value among a record set.

MIN () Syntax:

**SELECT MIN (column_name) FROM
table_name; EX:**

SELECT MIN (SALARY) FROM EMP;

SQL SUM Function SQL:

SUM function is used to find out the sum of a field in various records.

SUM () Syntax:

```
SELECT COUNT (column_name) FROM  
table_name; EX:
```

```
SELECT COUNT (EID) FROM EMP;
```

PRIMARY Key:

- A primary key is a field in a table which uniquely identifies each row/record in a database table.

Properties Primary key:

•A primary keys must contain:

- 1) Unique values
 - 2) NOT NULL values.
- A table can have only one primary key, which may consist of single or multiple fields.
 - If a table has a primary key defined on any field(s), then you cannot have two records having the same value of that field(s).

FOREIGN Key:

- A foreign key is a key used to link two tables together.
- This is sometimes called a referencing key.
- Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.
- The relationship between 2 tables matches the Primary Key in one of the tables with a Foreign Key in the second table.

Sub-Queries/Nested Queries in SQL: Introduction to Nested Queries :

- One of the most powerful features of SQL is nested queries.
- A nested query is a query that has another query embedded within it; the embedded query is called a sub query.
- When writing a query, we sometimes need to express a condition that refers to a table that must itself be computed.
- A subquery typically appears within the WHERE clause of a query. Subqueries can sometimes appear in the FROM clause or the HAVING clause.
- Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN etc.
- There are a few rules that subqueries must follow:
 1. Subqueries must be enclosed within parentheses.
 2. A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
 3. A subquery cannot be immediately enclosed in a set function.

Subqueries with the SELECT Statement:

Subqueries are most frequently used with the SELECT statement. The basic syntax is as follows:

```
SELECT column_name  
[, column_name ] FROM table1  
[, table2 ]  
  
WHERE column_name OPERATOR  
  (SELECT column_name  
  [, column_name ] FROM table1 [,  
  table2 ]  
  [WHERE])
```

Ex: select *from customers where id in (select id from customers where salary >4500);

Subqueries with the INSERT Statement:

- Sub queries also can be used with INSERT statements.
- The INSERT statement uses the data returned from the subquery to insert into another table.
- The selected data in the subquery can be modified with any of the character, date or number functions.

Syntax

```
INSERT INTO table_name [ (column1
    [, column2 ] ) ] SELECT
    [ *|column1 [, column2 ]

    FROM table1 [, table2]
    [ WHERE VALUE OPERATOR ]
```

Ex:

insert into customers_bkp select * from customers where id in (select id from customers) ;

Subqueries with the UPDATE Statement:

- The subquery can be used in conjunction with the UPDATE statement.
- Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.

Syntax:

```
UPDATE table SET column_name = new_value [ WHERE OPERATOR
[ VALUE ] (SELECTCOLUMN_NAME FROM TABLE_NAME) [ WHERE) ];
```

EX:

```
UPDATE CUSTOMERS SET SALARY = SALARY * 0.25 WHERE AGE IN (SELECT AGE
FROM CUSTOMERS_BKP WHERE AGE >= 27 );
```

Transactions:

A transaction is a unit of program execution that accesses and possibly updates various data items.

(or)

A transaction is an execution of a user program and is seen by the DBMS as a series or list of actions i.e., the actions that can be executed by a transaction includes the reading and writing of database.

Transaction Operations:

Access to the database is accomplished in a transaction by the following two operations,

read(X) : Performs the reading operation of data item X from the database.

write(X) : Performs the writing operation of data item X to the database.

Example:

Let T1 be a transaction that transfers \$50 from account A to account B. This transaction can be illustrated as follows,

```
T1    : read(A);  
      A := A - 50;  
      write(A);  
      read(B);  
      B := B + 50;  
      write(B);
```

Transaction Concept:

The concept of transaction is the foundation for concurrent execution of transaction in a DBMS and recovery from system failure in a DBMS.

A user writes data access/updates programs in terms of the high-level query language supported by the DBMS.

~~To understand how the DBMS handles such requests, with respect to concurrency control and recovery, it is convenient to regard an execution of a user program or transaction, as a series of reads and writes of database objects.~~

To read a database object, it is first brought in to main memory from disk and then its value is copied into a program. This is done by read operation.

To write a database object, in-memory, copy of the object is first modified and then written to disk. This is done by the write operation.

Properties of Transaction (ACID):

There are four important properties of transaction that a DBMS must ensure to maintain data in concurrent access of database and recovery from system failure in DBMS.

The four properties of transactions are,