SQL Joins

The 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.

SQL Join Types:

- **INNER JOIN:** returns rows when there is a match in both tables.
- **LEFT JOIN:** returns all rows from the left table, even if there are no matches in the right table.
- **RIGHT JOIN:** returns all rows from the right table, even if there are no matches in the left table.
- **FULL JOIN:** returns rows when there is a match in one of the tables.
- **CARTESIAN JOIN:** returns the cartesian product of the sets of records from the two or more joined tables.
- **SELF JOIN:** is used to join a table to itself, as if the table were two tables, temporarily renaming at least one table in the SQL statement.

INNER JOIN

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

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

LEFT JOIN

The SQL Left Join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching.

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

RIGHT JOIN

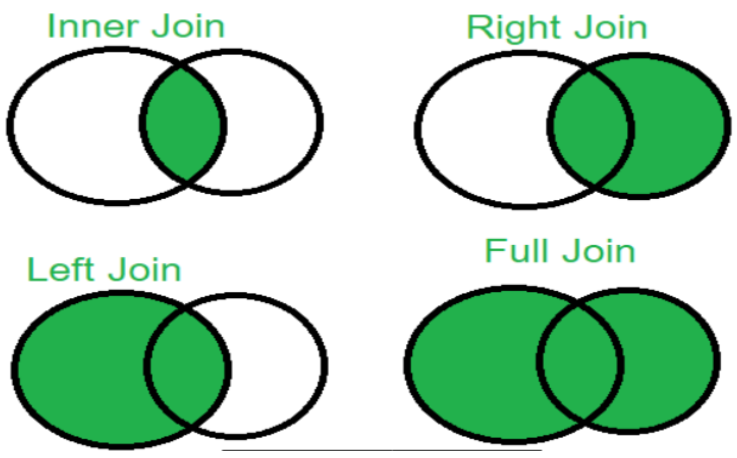
The SQL Right Join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching.

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

FULL JOIN

The SQL FULL JOIN combines the results of both left and right outer joins.

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



CARTESIAN JOIN

- The CARTESIAN JOIN or CROSS JOIN returns the cartesian product of the sets of records from the two or more joined tables.
- It produces a result set which is the number of rows in the first table multiplied by the number of rows in the second table if no WHERE clause is used along with CROSS JOIN.
- If WHERE clause is used with CROSS JOIN, it functions like an INNER JOIN.

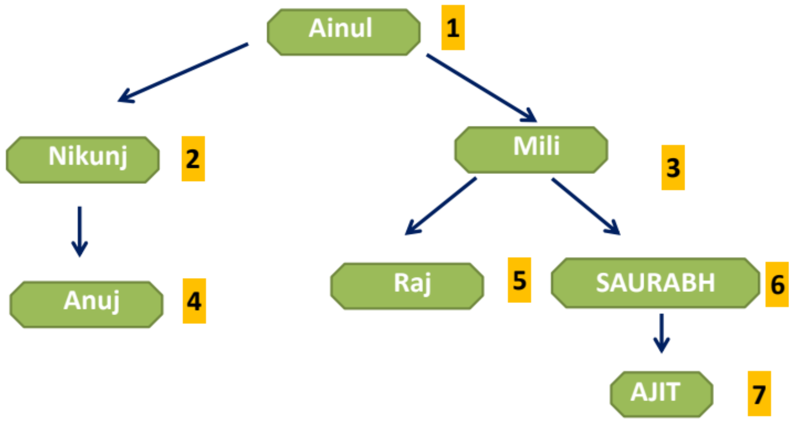
```
SELECT table1.column1, table2.column2...
FROM table1
CROSS JOIN table2

SELECT table1.column1, table2.column2...
FROM table1
CROSS JOIN table2
WHERE table1.common_filed = table2.common_field;
```

SELF JOIN

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

```
SELECT a.column_EID, b.column_EID...
FROM table1 a, table1 b
WHERE a.common_filed = b.common_field
```



UNION CLAUSE

The 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.

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

```
[WHERE condition]
```

UNION

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

UNION ALL CLAUSE

The SQL UNION ALL clause/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.

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

UNION ALL

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

INTERSECT CLAUSE

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.

The same rules that apply to UNION apply to the INTERSECT operator.

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

INTERSECT

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

EXCEPT CLAUSE

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.

The same rules that apply to UNION apply to the EXCEPT operator.

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

EXCEPT

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

Indexes

Indexes are special lookup tables that the database search engine can use to speed up data retrieval. An index helps speed up SELECT queries and WHERE clauses, but it slows down data input, with UPDATE and INSERT statements. Indexes can be created or dropped with no effect on the data.

The CREATE INDEX Command:

```
CREATE INDEX index_EID ON table_EID (column_EID);
```

Composite Indexes:

```
CREATE INDEX index_EID on table_EID (column1, column2);
```

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.

DROP INDEX Command:

```
DROP INDEX index_EID ON table_EID;
```

VIEWS

A view is nothing more than a SQL statement that is stored in the database with an associated ID.

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.

VIEWS

```
CREATE VIEW view_EID AS
(SELECT column1, column2.....
FROM table_EID
WHERE [condition]
);
```

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.

```
CREATE VIEW view_EID AS
SELECT column1, column2.....
FROM table_EID
WHERE [condition]
WITH CHECK OPTION
;
```

Updating a Views

A view can be updated under certain conditions:

- The SELECT clause may not contain the keyword DISTINCT.

- The SELECT clause may not contain summary 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.

Dropping Views

DROP VIEW view_EID;

SQL HAVING CLAUSE

The HAVING clause enables we 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

The following is the position of the HAVING clause in a query

```
SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
```

The HAVING clause must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used.
The following is the syntax of the SELECT statement, including the HAVING clause:

```
SELECT column1, column2
FROM table1, table2
WHERE [ conditions ]
GROUP BY column1, column2
```

HAVING [conditions]
ORDER BY column1, column2

SQL FUNCTIONS

- **COUNT Function -** The SQL Server COUNT aggregate function is used to count the number of rows in a database table.
- **MAX Function -** The SQL Server MAX aggregate function allows the user to select the highest (maximum) value for a certain column.
- **MIN Function -** The SQL Server MIN aggregate function allows the user to select the lowest (minimum) value for a certain column.
- **AVG Function -** The SQL Server AVG aggregate function selects the average value for a certain table column.
- **SUM Function -** The SQL Server SUM aggregate function allows selecting the total for a numeric column.
- **SQRT Function -** This is used to generate a square root of a given number.
- **RAND Function -** This is used to generate a random number using SQL command.
- **CONCAT Function -** This is used to concatenate multiple parameters to a single parameter.

SQL STRING FUNCTIONS

- ASCII() Ascii code value will come as output for a character expression..
- CHAR() Character will come as output for given Ascii code or integer.
- CHARINDEX() Starting position for given search expression will come as output in a given string expression. EX: Select CHARINDEX('G', 'KING')
- LEFT() Left part of the given string till the specified number of characters
- RIGHT() Right part of the given string till the specified number of characters.
- LEN() Number of characters will come as output for a given string expression
- LOWER() Lowercase string will come as output for a given string data.
- UPPER() Uppercase string will come as output for a given string data.
- SUBSTRING() Part of a string based on the start position value and length value.
  Ex: Select SUBSTRING ('WORLD', 1,3)
- REPLACE() String expression will come as output for a given string data after replacing all occurrences of specified character with specified character.
  Ex: Select REPLACE('INDIA', 'I', 'K')
- REVERSE() Reverse string expression will come as output for a given string data
- STUFF() String expression will come as output for a given string data after replacing from starting character till the specified length with specified character.
  Ex: Select STUFF('ABCDEFGH', 2,4,'IJK')

SQL DATE FUNCTIONS

Below are the commonly used DATE Functions:

FUNCTION	SYNTAX
GETDATE	GETDATE()
DATEADD	DATEADD (datepart , number , date)
DATEDIFF	DATEDIFF (datepart , startdate , enddate)
DAY	DAY(DATE)
MONTH	MONTH(DATE)
YEAR	YEAR(DATE)
DATEPART	DATEPART(datepart, datecolumnEID)
CONVERT	CONVERT(datatype, expression, style)
FORMAT	FORMAT (getdate(), 'D') -> Wednesday, September 6, 2017

Select CONVERT (varchar(19),getdate()) -> Sep 6 2017 11:24PM
Select CONVERT (varchar(19),getdate(),10) -> 09-06-17
Select CONVERT (varchar(19),getdate(),110) -> 09-06-2017