JOINS

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;
```

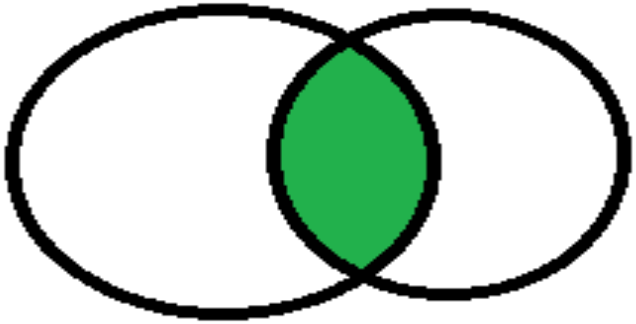
FULL JOIN

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

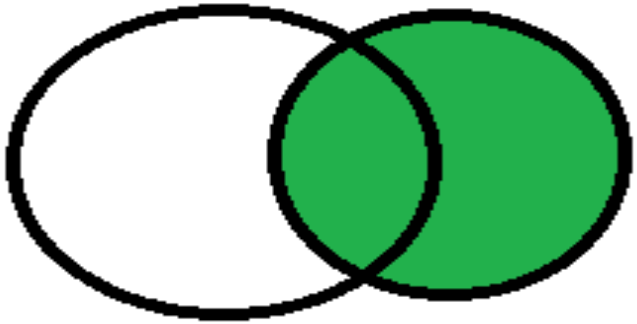
```
SELECT table1.column1, table2.column2...
FROM table1
FULL JOIN table2
ON table1.common_filed = table2.common_field;
```

JOINS

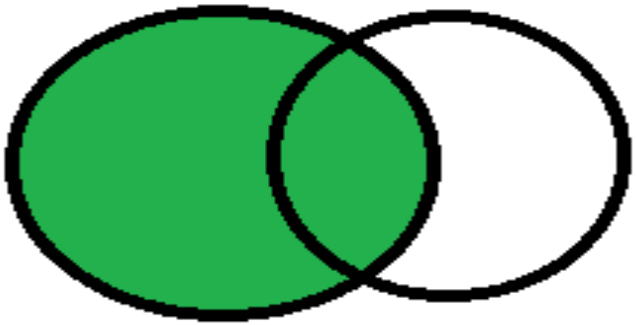
Inner Join



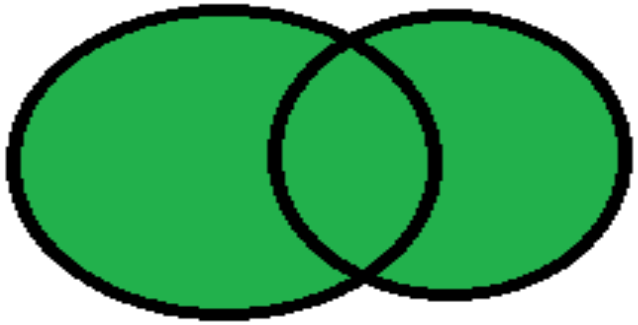
Right Join



Left Join



Full Join



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
```

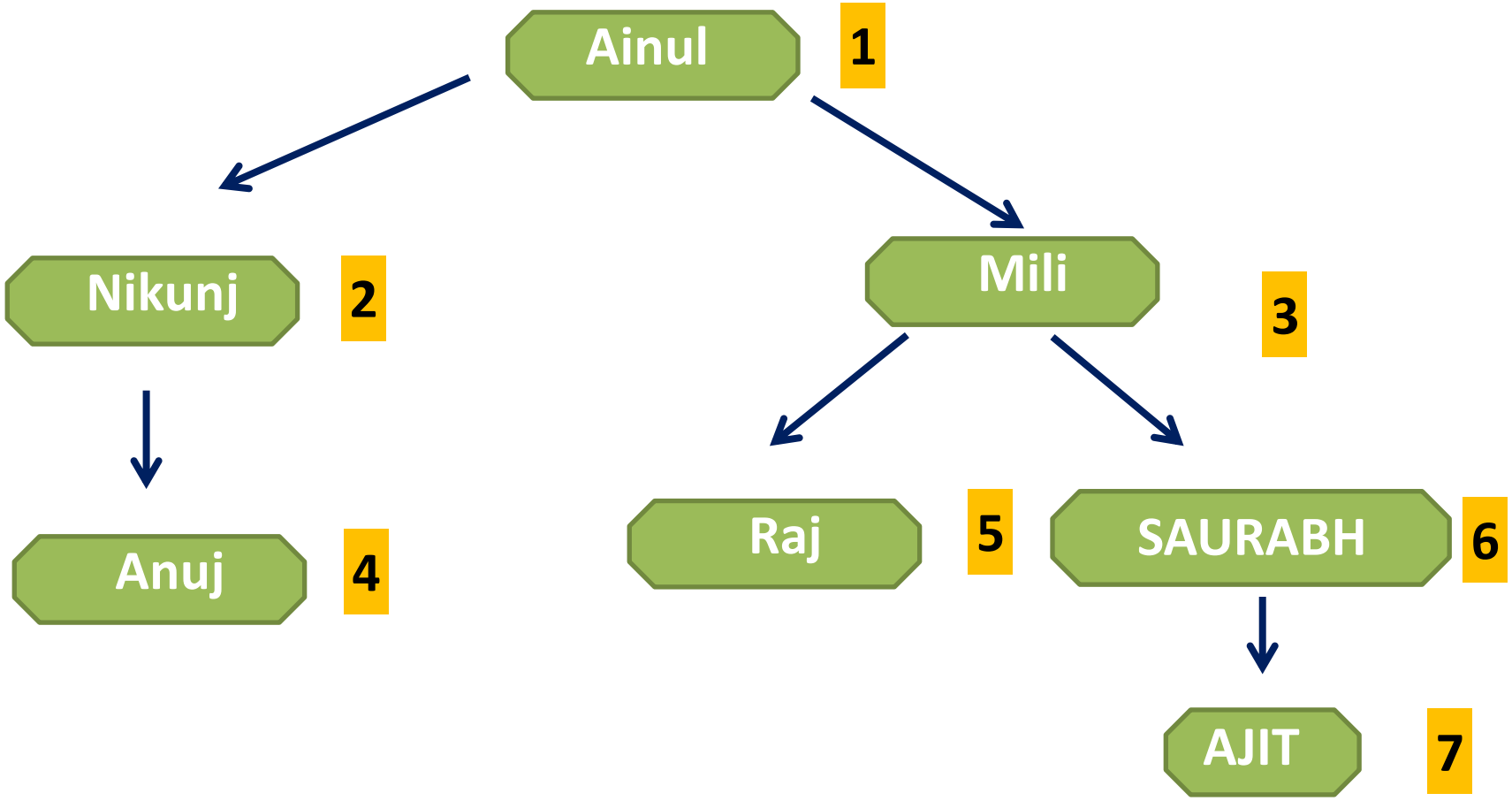
```
SELECT table1.column1, table2.column2...
FROM table1
CROSS JOIN table2
WHERE table1.common_filed = table2.common_field;
```

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;
```

SELF JOIN



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 second SELECT statement.

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

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

EXCEPT

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

ASSIGNMENT

ASSIGNMENT – 5

IN THE EMP TABLE DISPLAY :

1) EID NAME CITY DOJ DEPT DESI SALARY OF THE DELHI EMPLOYEES

2) DETAILS OF ALL THE EMPLOYEES WHOSE SALARY DETAILS ARE NOT AVAILABLE.

IN THE INVENTORY STRUCTURE DISPLAY :

1) PID, PDESC, CATEGORY, SNAME, SCITY

2) DISPLAY OID , ODATE , CNAME, CADDRESS, CPHONE, PDESC, PRICE,OQTY, AMT

INDEXES

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;
```