BASICS OF SQL

Data:

Data is defined as facts or figures or information that's stored in or used by a computer.

Database:

A Database is a collection of information that is organized so that it can be easily accessed, managed and updated.

Creation of database:

```
CREATE DATABASE databasename;
```

Example :

```
Create database employeeDB;
```

To access/select the Database,

```
USE databasename;
```

To view all your databases,

```
SHOW databases;
```

To drop database,

```
DROP DATABASE Database_Name;
```

Table:

Table is a collection of data, organized in terms of rows and columns. Table is the simple form of data storage. A table is also considered as a convenient representation of relations.

To create a table,

```
CREATE TABLE table_name(
  column1 datatype,
  column2 datatype,
  column3 datatype,
  .....
  columnN datatype,
  );
```

Example:

```
  CREATE TABLE Employee (
  EmployeeID int,
  FirstName varchar(255),
  LastName varchar(255),
  Email varchar(255),
  AddressLine varchar(255),
  City varchar(255)
  );
```

To drop a table,

```
DROP TABLE table_name;
```

To truncate a table,

```
TRUNCATE TABLE table_name;
```

To describe the table,

```
DESC table_name;
```

Constraints:

Constraints are the rules enforced on data columns on a table. It aims to increase the accuracy and reliability of the data stored in the database.

Types of Constraints:

1) NOT NULL – This makes sure that a NULL value will not be stored in a column

```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
ADDRESS varchar(20)
);
```

2) UNIQUE – The values entered to the table will be unique if this constraint is applied

```
CREATE TABLE Student(
ID int(6) NOT NULL UNIQUE,
NAME varchar(10),
ADDRESS varchar(20)
);
```

3) PRIMARY KEY – It is the candidate key that is selected to uniquely identify a tuple in a relation.

```
CREATE TABLE Student(
ID int(6) NOT NULL UNIQUE,
NAME varchar(10),
ADDRESS varchar(20),
PRIMARY KEY(ID)
);
CREATE TABLE Student(
ID int(6) PRIMARY KEY,
NAME varchar(10),
ADDRESS varchar(20)
);
```

4) FOREIGN KEY – A foreign key is a set of one or more columns in the child table whose values are required to match with corresponding columns in the parent table

```
CREATE TABLE Orders(
O_ID int PRIMARY KEY,
ORDER_NO int NOT NULL,
C_ID int,
FOREIGN KEY (C_ID) REFERENCES Customers(C_ID)
);
```

5) CHECK – If we want to satisfy a specific condition in a column then we use CHECK constraint

```
CREATE TABLE Student(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
AGE int NOT NULL CHECK (AGE >= 18)
);
```

6) DEFAULT – When no value is specified then a set of default values for a column is added

```
CREATE TABLE Student(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
AGE int DEFAULT 18
);
```

7) INDEX – This is used to create and also retrieve data from the database

```
CREATE INDEX index_name
  ON table_name ( column1, column2.....);
Example:
CREATE INDEX idx_age
  ON CUSTOMERS ( AGE )
```

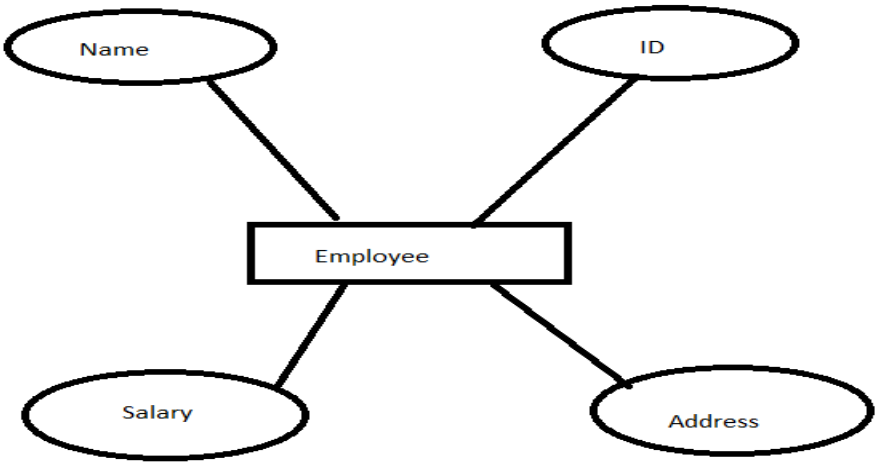
ER model:

ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.

It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.

In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram.

For example, Suppose we design a school database. In this database, the student will be an entity with attributes like address, name, id, age, etc. The address can be another entity with attributes like city, street name, pin code, etc and there will be a relationship between them.

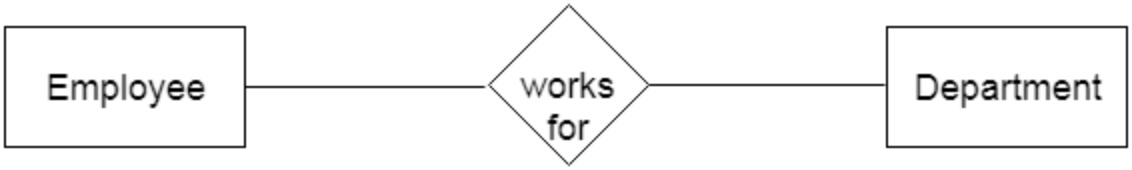


Component of ER Diagram:

Entity:

An entity may be any object, class, person or place. In the ER diagram, an entity can be represented as rectangles.

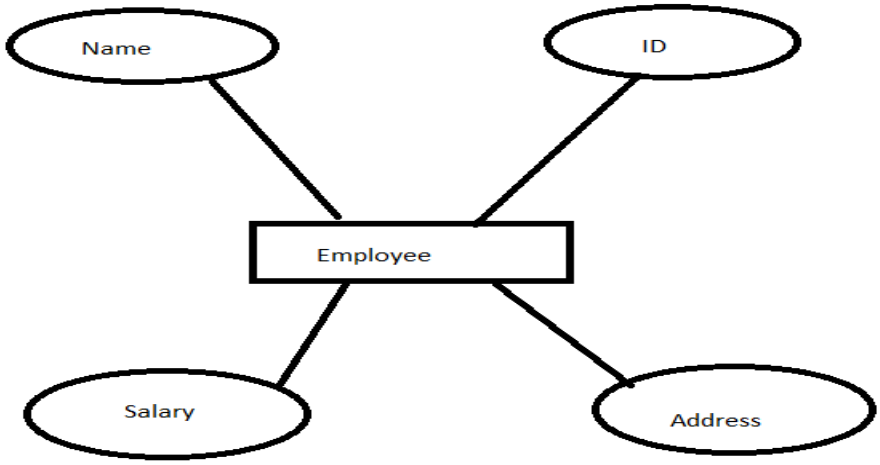
Consider an organization as an example- manager, product, employee, department etc. can be taken as an entity.



Attribute:

The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute.

For example, id, age, contact number, name, etc. can be attributes of a student.



Relationship:

A relationship is used to describe the relation between entities. Diamond or rhombus is used to represent the relationship.

Types of relationship are as follows:

a. One-to-One Relationship

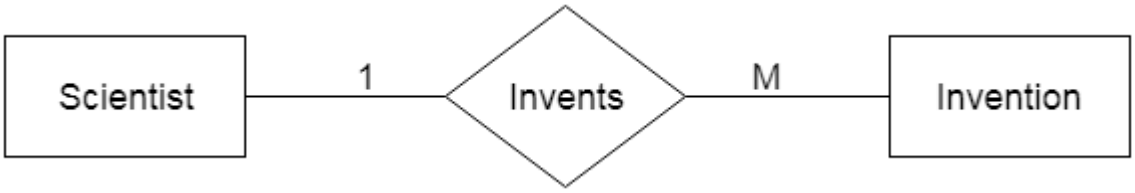
When only one instance of an entity is associated with the relationship, then it is known as one to one relationship.

For example, A female can marry to one male, and a male can marry to one female.

b. One-to-many relationship

When only one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then this is known as a one-to-many relationship.

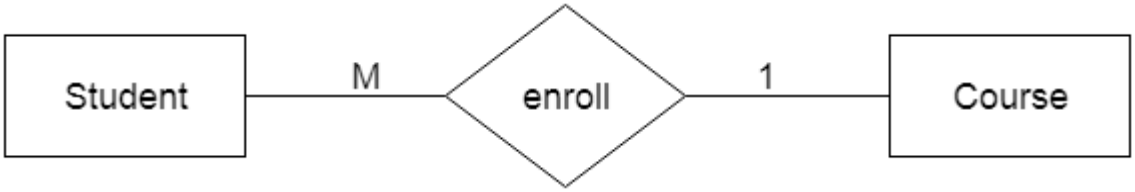
For example, Scientist can invent many inventions, but the invention is done by the only specific scientist.



c. Many-to-one relationship

When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a many-to-one relationship.

For example, Student enrolls for only one course, but a course can have many students.



d. Many-to-many relationship

When more than one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then it is known as a many-to-many relationship.

For example, Employee can assign by many projects and project can have many employees.

Normalization:

Normalization is the process of minimizing redundancy from a relation or set of relations. Redundancy in relation may cause insertion, deletion and updation anomalies. With the help of Normalization, we can organize this data and also reduce the redundant data. It usually divides a large table into smaller ones, so it is more efficient. Normalization in SQL will enhance the distribution of data.

Types of Normal forms:

1st Normal forms (1NF):

A relation is in first normal form if every attribute in that relation is singled valued attribute. In simple terms, a single cell cannot hold multiple values. If a table contains a composite or multi-valued attribute, it violates the First Normal Form.

Employee id	Employee name	Phone number
Id1	Alex	+91 7895466899 +91 4568621335
Id2	Barry	+91 3478989897
Id3	clair	+91 2576235685

In the above table, we can clearly see that the Phone Number column has two values. Thus, it violated the 1st NF. Now if we apply the 1st NF to the above table, we get the below table as the result.

Employee id	Employee name	Phone number
Id1	Alex	+91 7895466899
Id1	Alex	+91 4568621335
Id2	Barry	+91 3478989897
Id3	clair	+91 2576235685

By this, we have achieved atomicity and also each and every column have unique values.

2nd Normal Form:

To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency. A relation is in 2NF if it has No Partial Dependency, i.e., no non-

prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

Stud_no	Course_no	Course_fee
1	C1	1000
2	C2	1500
1	C4	2000
4	C3	1000
4	C1	1000
2	C5	2000

COURSE_FEE cannot alone decide the value of COURSE_NO or STUD_NO;
COURSE_FEE together with STUD_NO cannot decide the value of COURSE_NO;
COURSE_FEE together with COURSE_NO cannot decide the value of STUD_NO;
Hence,
COURSE_FEE would be a non-prime attribute, as it does not belong to the one only candidate key {STUD_NO, COURSE_NO} ;
But, COURSE_NO -> COURSE_FEE , i.e., COURSE_FEE is dependent on COURSE_NO, which is a proper subset of the candidate key. Non-prime attribute COURSE_FEE is dependent on a proper subset of the candidate key, which is a partial dependency and so this relation is not in 2NF.

To convert the above relation to 2NF,we need to split the table into two tables such as :
Table 1: STUD_NO, COURSE_NO
Table 2: COURSE_NO, COURSE_FEE
Table 1:

Stud_no	Course_no
1	C1
2	C2
1	C4
4	C3
4	C1
2	C5

Table 2:

Course_no	Course_fee
C1	1000
C2	1500
C3	1000
C4	2000

C5	2000

3rd Normal Form:

A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form. That means non-prime attributes (which doesn't form a candidate key) should not be dependent on other non-prime attributes in a given table

Transitive dependency – If A->B and B->C are two FDs then A->C is called transitive dependency.

Student_id	Student name	Subject id	Subject	address
Id1	Alex	S1	SQL	Mumbai
Id2	Barry	S2	Java	Goa
Id3	Clair	S3	C	Delhi
Id4	david	S2	Java	kochi

In the above table, Student ID determines Subject ID, and Subject ID determines Subject. Therefore, Student ID determines Subject via Subject ID. This implies that we have a transitive functional dependency, and this structure does not satisfy the third normal form.

Now in order to achieve third normal form, we need to divide the table as shown below:

Table 1:

Student_id	Student name	Subject id	address
Id1	Alex	S1	Mumbai
Id2	Barry	S2	Goa
Id3	Clair	S3	Delhi
Id4	david	S2	kochi

Table 2:

Subject id	Subject
S1	SQL
S2	Java
S3	C

As you can see from the above tables all the non-key attributes are now fully functional dependent only on the primary key. In the first table, columns Student Name, Subject ID and Address are only dependent on Student ID. In the second table, Subject is only dependent on Subject ID.

SQL Commands:

SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.

SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

Types of SQL Commands:

There are five types of SQL commands:

1) DDL - Data Definition language
2) DML – Data manipulation language
3) DCL – Data control language
4) TCL – Transaction control language
5) DQL -Data query language

DDL - Data Definition language:

DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc. All the command of DDL are auto-committed that means it permanently save all the changes in the database.

CREATE – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).

CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);

Example:

CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);

CREATE DATABASE university;

CREATE TABLE students;

CREATE VIEW for_students;

DROP – is used to delete objects from the database.

DROP TABLE table_name;

Example:

DROP TABLE EMPLOYEE;

ALTER-is used to alter the structure of the database.

ALTER TABLE table_name ADD column_name COLUMN-definition;

Example:

Alter table student add column age int;

TRUNCATE–is used to remove all records from a table, including all spaces allocated for the records are removed.

TRUNCATE TABLE table_name;

Example:

TRUNCATE table students;

COMMENT –is used to add comments to the data dictionary.

RENAME –is used to rename an object existing in the database.

DML – Data manipulation language:

DML commands are used to modify the database. It is responsible for all form of changes in the database.

The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

INSERT – is used to insert data into a table.

```
INSERT INTO TABLE_NAME  (col1, col2, col3,.... col N)
VALUES (value1, value2, value3, .... valueN);
Or
INSERT INTO TABLE_NAME
VALUES (value1, value2, value3, .... valueN);
```

Example:

```
INSERT INTO students (RollNo, FIrstName, LastName) VALUES ('60', 'Tom', Erichsen');
```

UPDATE – is used to update existing data within a table.

```
UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [WHERE CONDITION]
```

Example:

```
UPDATE students
SET FirstName = 'John', LastName= 'Wick'
WHERE StudID = 3;
```

DELETE – is used to delete records from a database table.

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

Example:

```
DELETE FROM students
WHERE FirstName = 'John';
```

DQL -Data query language:

DQL is used to fetch the data from the database.

SELECT – is used to retrieve data from the database.

```
SELECT expressions
FROM TABLES
WHERE conditions;
```

Example:

```
SELECT emp_name
FROM employee
WHERE age > 20;
```

DCL – Data control language:

DCL commands are used to grant and take back authority from any database user. It mainly deals with the rights, permissions and other controls of the database system.

GRANT-gives users access privileges to the database.

GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;

REVOKE-withdraw user's access privileges given by using the GRANT command.

REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

TCL – Transaction control language:

TCL commands deal with the transaction within the database.

COMMIT– commits a Transaction.

ROLLBACK– rollbacks a transaction in case of any error occurs.

SAVEPOINT–sets a savepoint within a transaction.

SET TRANSACTION–specify characteristics for the transaction.

Logical order of operations (or) order of execution for an SQL query:

1. FROM, including JOINs
2. WHERE
3. GROUP BY
4. HAVING
5. WINDOW functions
6. SELECT
7. DISTINCT
8. UNION
9. ORDER BY
10. LIMIT and OFFSET

What is wrong with this query?

1. SELECT Id, YEAR(TrialDate) AS TrialYear
FROM Payments
WHERE TrialYear <= 2015;

2. SELECT Id, TrialDate
FROM Payments
GROUP BY Id;

3. SELECT UserId, AVG(Total) AS AvgOrderTotal
FROM Invoices
HAVING COUNT(OrderId) >= 1

Common interview questions:

- ✓ Are NULL values in a database the same as that of blank space or zero?
- ✓ Explain the difference between the DELETE, TRUNCATE, DROP command in a DBMS.
- ✓ What is meant by normalization and denormalization?
- ✓ Explain different types of keys in a database.
- ✓ What are Constraints in SQL?
- ✓ What are some common clauses used with SELECT query in SQL?
- ✓ List the different types of relationships in SQL.
- ✓ What is an Alias in SQL?
- ✓ What are the differences between Having and Where clause?
- ✓ What is primary key, foreign key and unique key?

Group by:

- ➢ The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".
- ➢ The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

Examples:

1. SQL statement lists the number of customers in each country:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

2. SQL statement lists the number of orders sent by each shipper

```
SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders
LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID
GROUP BY ShipperName;
```

Order by:

- ➢ The ORDER BY keyword is used to sort the result-set in ascending or descending order.
- ➢ The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

Examples:

1. SQL statement selects all customers from the "Customers" table, sorted by the "Country" column

```
SELECT * FROM Customers
ORDER BY Country;
```

2. SQL statement selects all customers from the "Customers" table, sorted DESCENDING by the "Country" column

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

3. SQL statement selects all customers from the "Customers" table, sorted ascending by the "Country" and descending by the "CustomerName" column

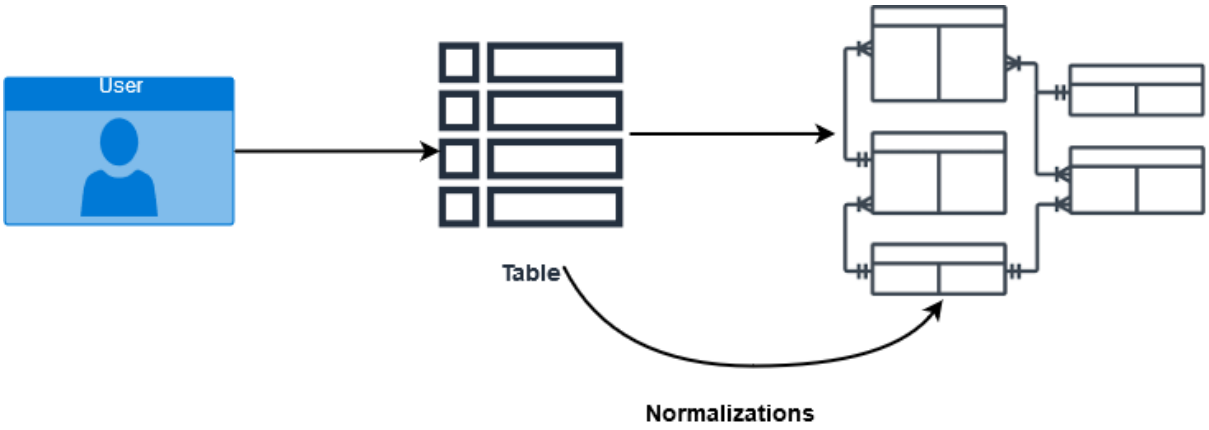
```
SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

difference between Order by and group by clause:

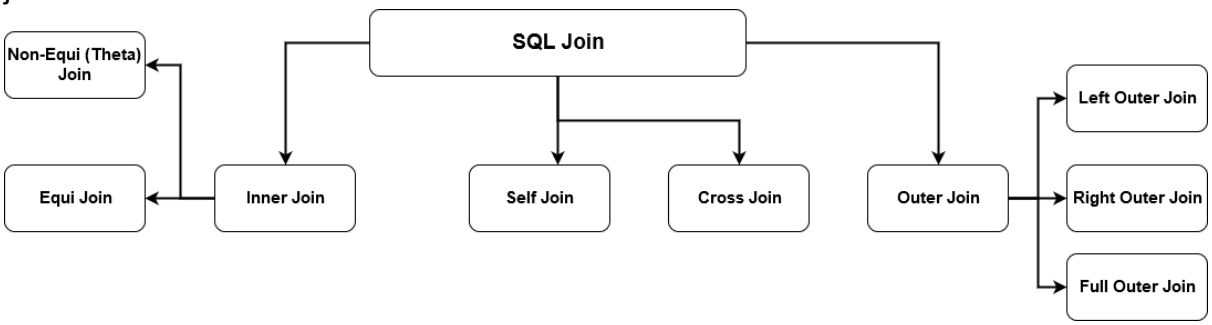
S.NO	GROUP BY	ORDER BY
1.	Group by statement is used to group the rows that have the same value.	Whereas Order by statement sort the result-set either in ascending or in descending order.
2.	It may be allowed in CREATE VIEW statement.	While it does not use in CREATE VIEW statement.
3.	In select statement, it is always used before the order by keyword.	While in select statement, it is always used after the group by keyword.
4.	Attribute cannot be in the group by statement under aggregate function.	Whereas in order by statement, attribute can be under aggregate function.
5.	In group by clause, the tuples are grouped based on the similarity between the attribute values of tuples.	Whereas in order by clause, the result-set is sorted based on ascending or descending order.
6.	Group by controls the presentation of tuples(rows).	While order by clause controls the presentation of columns.

JOINS

SQL JOIN is a clause that is used to combine multiple tables and retrieve data based on a common field in relational databases. Database professionals use normalizations for ensuring and improving data integrity. In the various normalization forms, data is distributed into multiple logical tables. These tables use referential constraints – primary key and foreign keys – to enforce data integrity in SQL Server tables. In the below image, we get a glimpse of the database normalization process.



SQL JOIN generates meaningful data by combining multiple relational tables. These tables are related using a key and have one-to-one or one-to-many relationships. To retrieve the correct data, you must know the data requirements and correct join mechanisms. SQL Server supports multiple joins and each method has a specific way to retrieve data from multiple tables. The below image specifies the supported SQL Server joins.



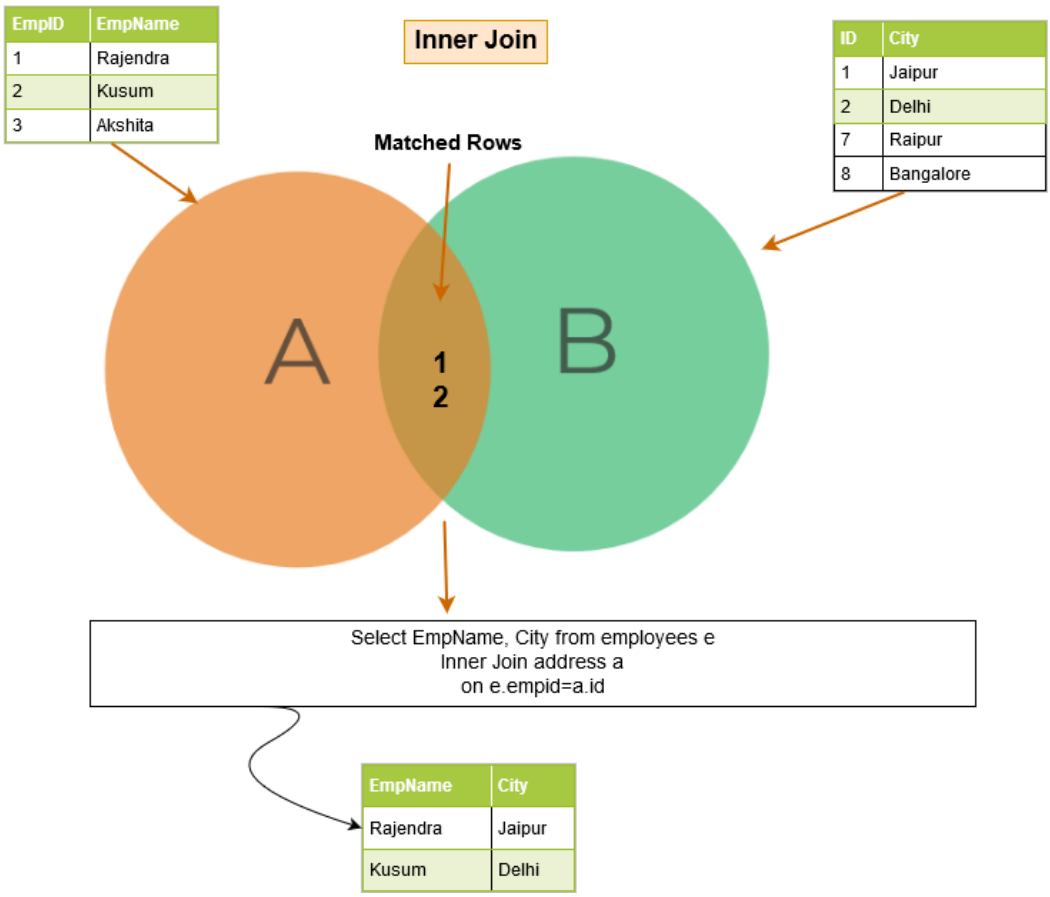
SQL inner join

The SQL inner join includes rows from the tables where the join conditions are satisfied. For example, in the below Venn diagram, inner join returns the matching rows from Table A and Table B.

In the below example, notice the following things:

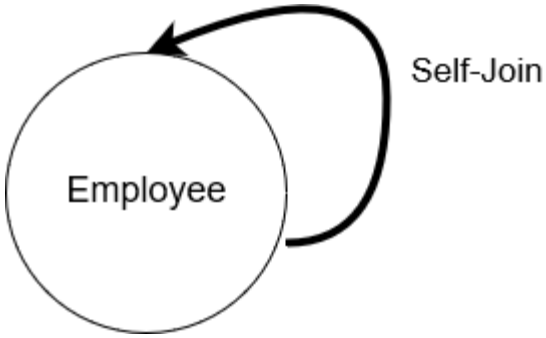
We have two tables – [Employees] and [Address].
The SQL query is joined on the [Employees].[EmpID] and [Address].[ID] column.

The query output returns the employee records for EmpID that exists in both tables.



The inner join returns matching rows from both tables; therefore, it is also known as Equi join. If we don't specify the inner keyword, SQL Server performs the inner join operation.
self-join

In a self-join, SQL Server joins the table with itself. This means the table name appears twice in the from clause.



Below, we have a table [Emp] that has employees as well as their managers' data. The self-join is useful for querying hierarchical data. For example, in the employee table, we can use self-join to learn each employee and their reporting manager's name.

EmpID	Name	EmpMgrid
1	Rajendra	1
2	Mohan	1
3	Amit	1
4	Manoj	1
5	Manish	2
6	Kapil	2

Self Join

```
SELECT e.EmpID, e.Name, m.Name as Manager  FROM Emp e Inner Join Emp m
On e.EmpMgrid=m.EmpID;
```

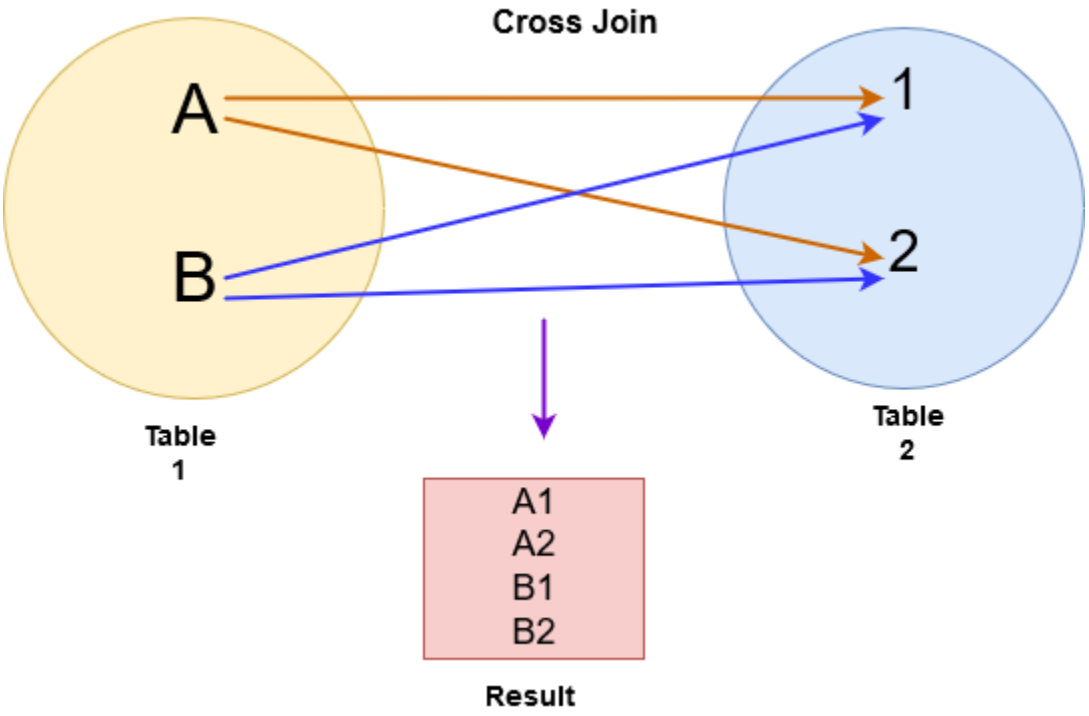
```
SELECT e.EmpID, e.Name,
m.Name as Manager
FROM Emp e Inner Join Emp m
On e.EmpMgrid=m.EmpID;
```

Results Messages

EmpID	Name	Manager
1	Rajendra	Rajendra
2	Mohan	Rajendra
3	Amit	Rajendra
4	Manoj	Rajendra
5	Manish	Mohan
6	Kapil	Mohan

The above query puts a self-join on [Emp] table. It joins the EmpMgrID column with the EmpID column and returns the matching rows.
cross join

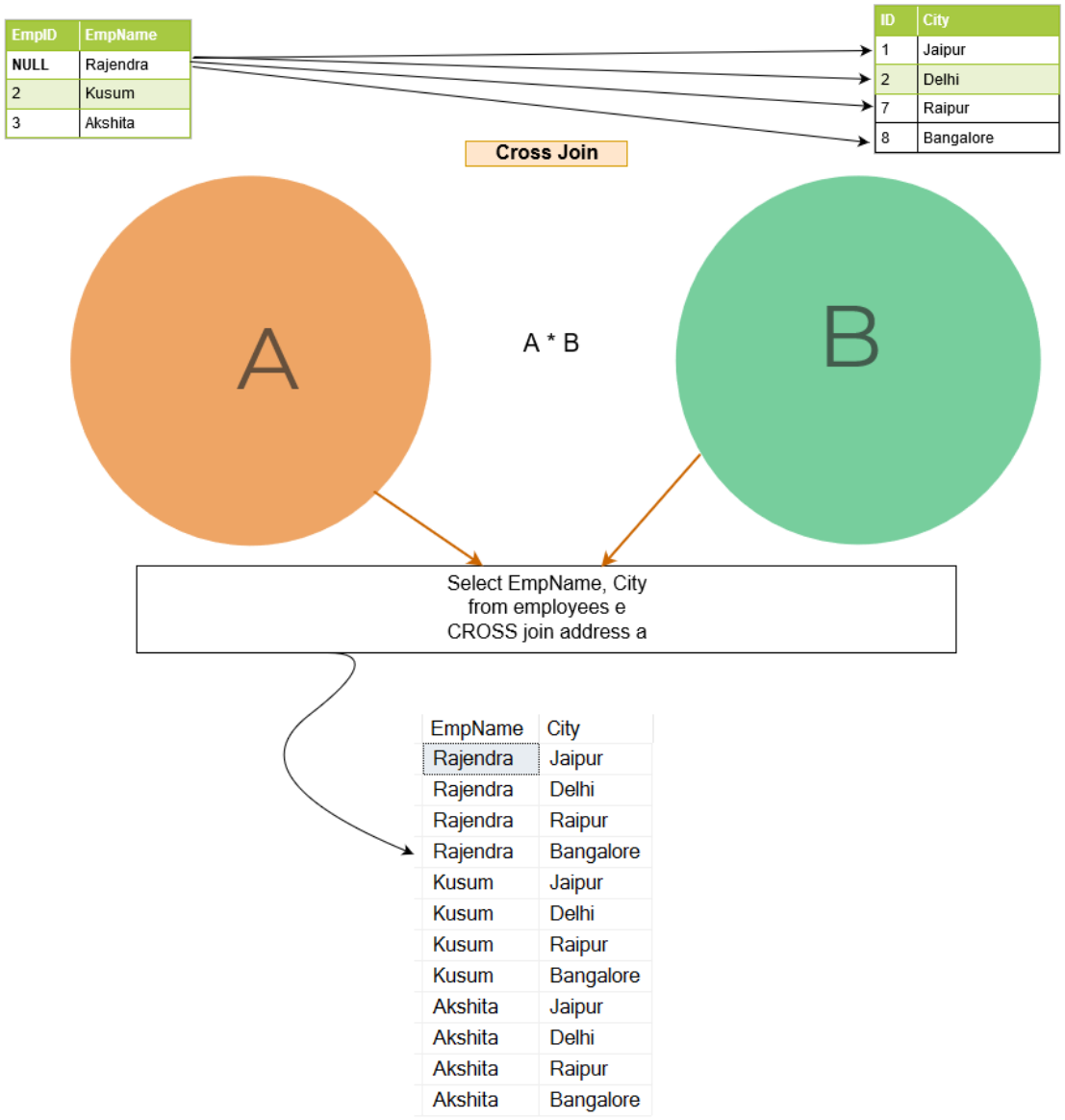
In the cross join, SQL Server returns a Cartesian product from both tables. For example, in the below image, we performed a cross-join for table A and B.



The cross join joins each row from table A to every row available in table B. Therefore, the output is also known as a Cartesian product of both tables. In the below image, note the following:

Table [Employee] has three rows for Emp ID 1,2 and 3.
Table [Address] has records for Emp ID 1,2,7 and 8.

In the cross-join output, row 1 of [Employee] table joins with all rows of [Address] table and follows the same pattern for the remaining rows.

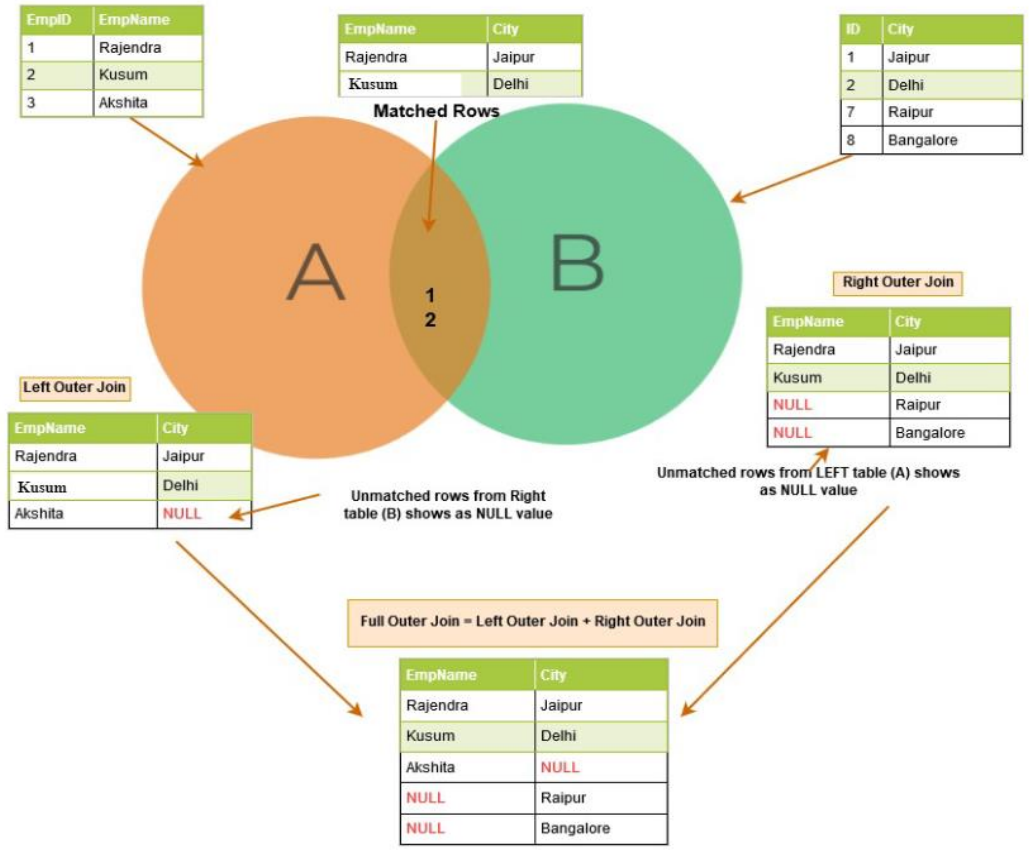


If the first table has x number of rows and the second table has n number of rows, cross join gives x*n number of rows in the output. You should avoid cross join on larger tables because it might return a vast number of records and SQL Server requires a lot of computing power (CPU, memory and IO) for handling such extensive data.

outer join

As we explained earlier, the inner join returns the matching rows from both of the tables. When using a SQL outer join, it not only lists the matching rows, but it also returns the unmatched rows from the other tables. The unmatched row depends on the left, right or full keywords.

The below image describes at a high-level the left, right and full outer join.

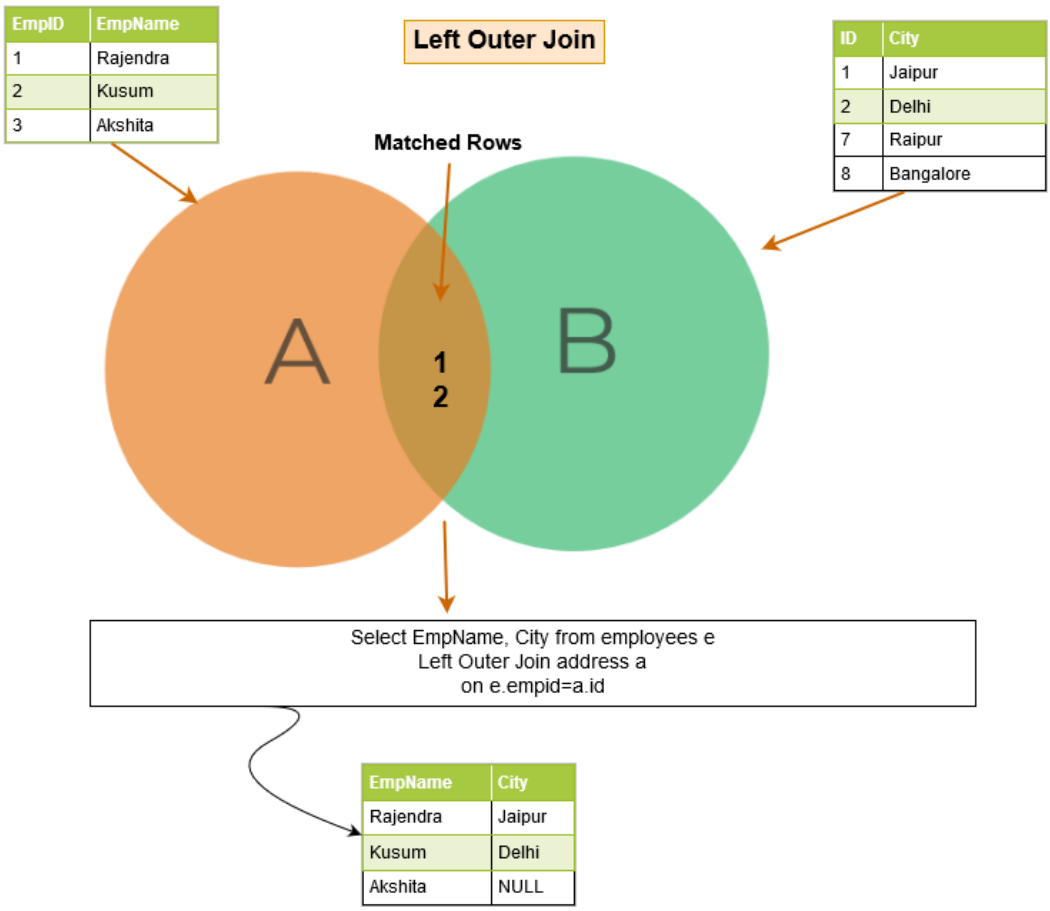


Left outer join

SQL left outer join returns the matching rows of both tables along with the unmatched rows from the left table. If a record from the left table doesn't have any matched rows in the right table, it displays the record with NULL values.

In the below example, the left outer join returns the following rows:

Matched rows: Emp ID 1 and 2 exists in both the left and right tables.
Unmatched row: Emp ID 3 doesn't exist on the right table. Therefore, we have a NULL value in the query output.



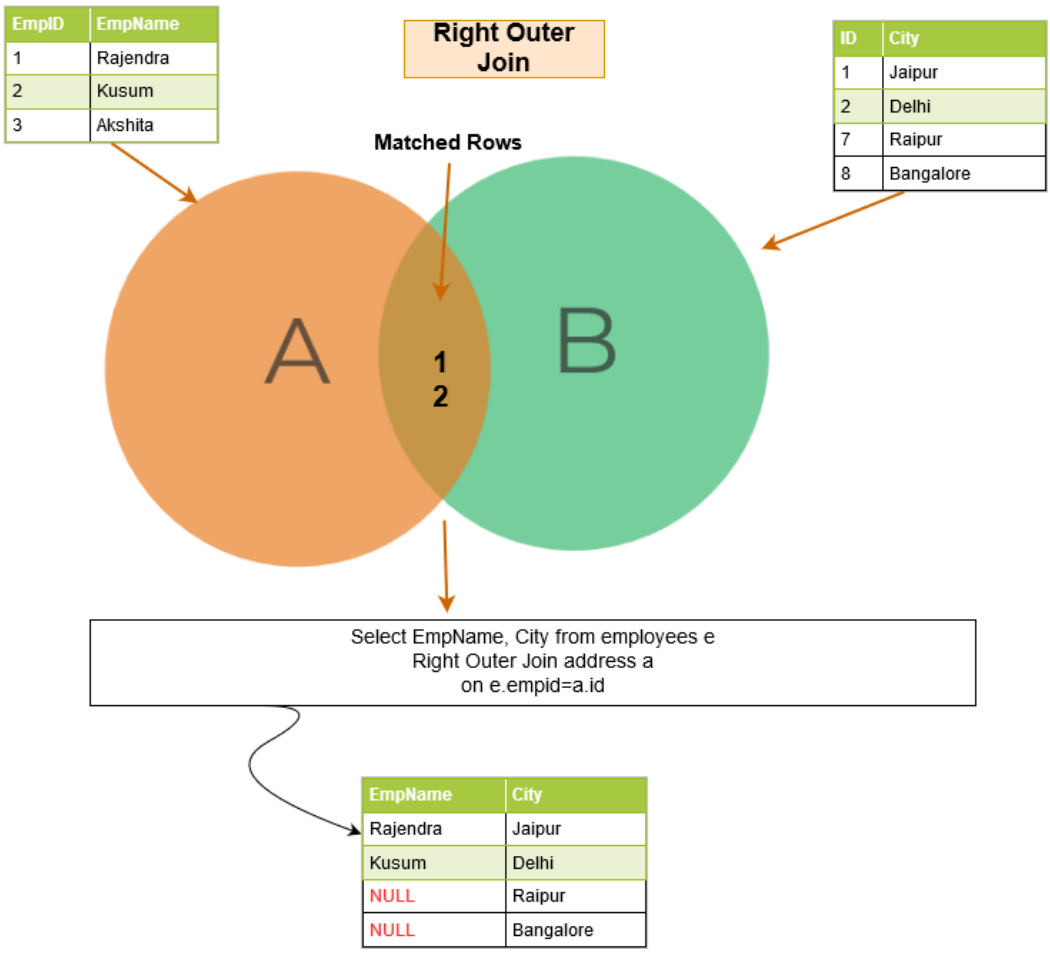
Right outer join

SQL right outer join returns the matching rows of both tables along with the unmatched rows from the right table. If a record from the right table does not have any matched rows in the left table, it displays the record with NULL values.

In the below example, we have the following output rows:

Matching rows: Emp ID 1 and 2 exists in both tables; therefore, these rows are matched rows.

Unmatched rows: In the right table, we have additional rows for Emp ID 7 and 8, but these rows are not available in the left table. Therefore, we get NULL value in the right outer join for these rows.



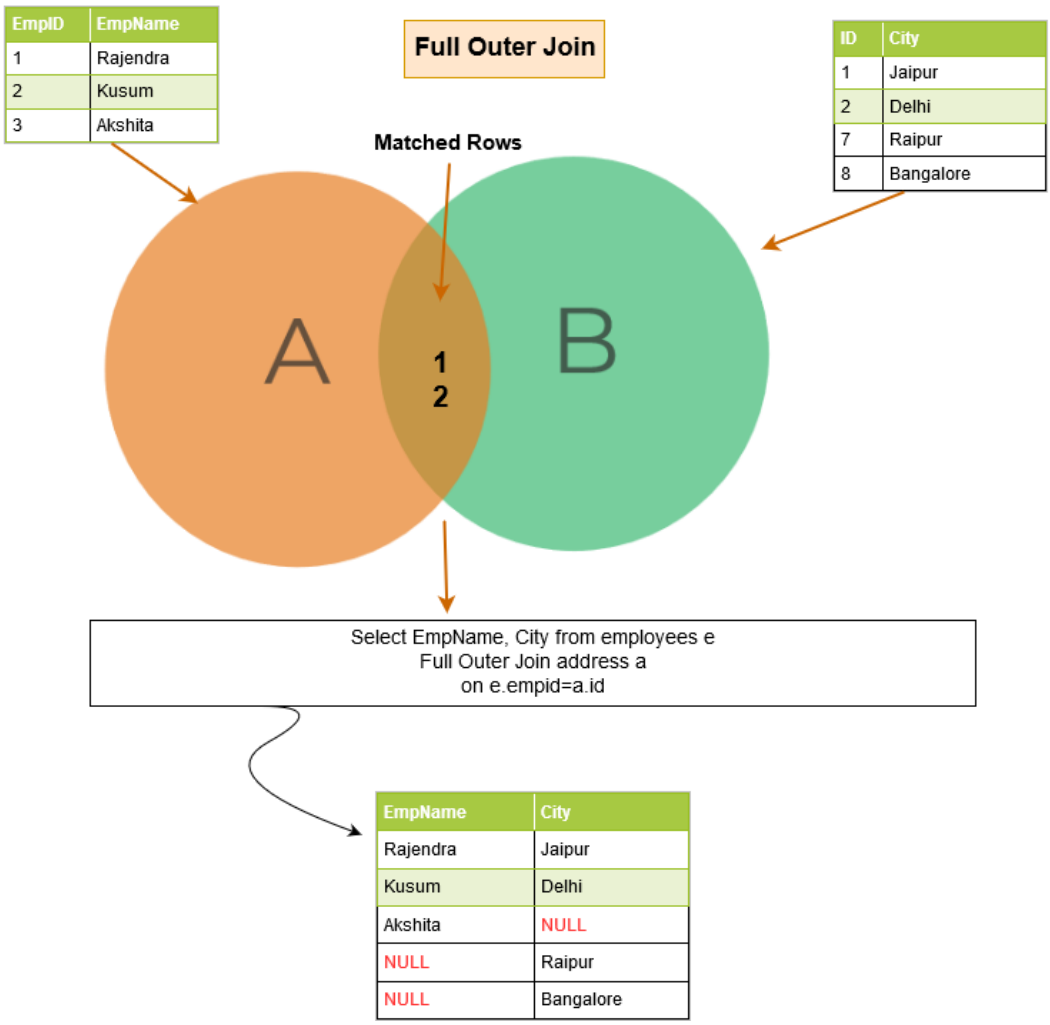
Full outer join

A full outer join returns the following rows in the output:

Matching rows between two tables.

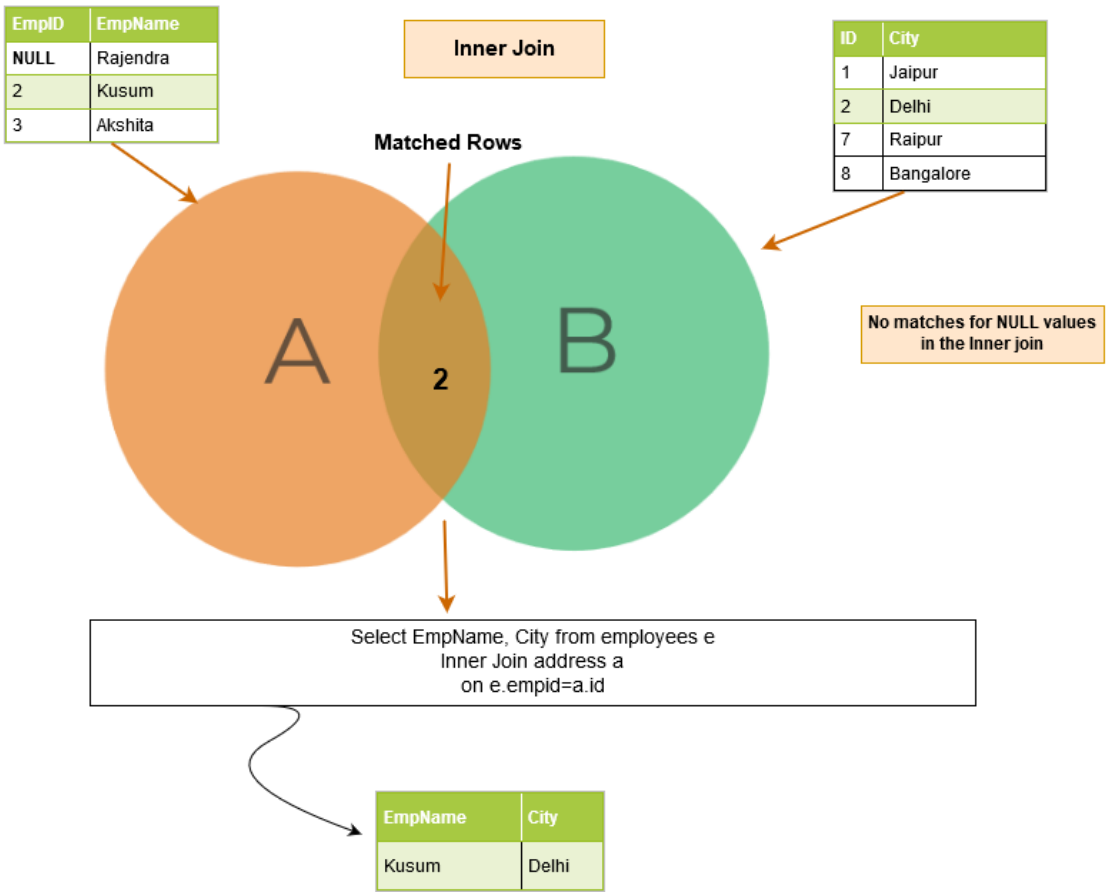
Unmatched rows similar to left outer join: NULL values for unmatched rows from the right table.

Unmatched rows similar to right outer join: Null values for unmatched rows from the left table.



NULL values and SQL joins

The NULL values do not match one another. Therefore, SQL Server could not return the matching row. In the below example, we have NULL in the EmpID column of the [Employees] table. Therefore, in the output, it returns the matching row for [EmpID] 2 only.



We can get this NULL row in the output in the event of a SQL outer join because it returns the unmatched rows as well.