**24th April RDBMS Assignment**

1. What is RDBMS? Why do industries use RDBMS?

Ans- RDBMS stands for Relational Database Management System. It is a software system that manages relational databases, which are a type of database that stores and organizes data in tables consisting of rows and columns. In an RDBMS, data is stored in a structured manner and is accessible using a set of standard SQL (Structured Query Language) commands.

Industries use RDBMS for several reasons, including:

1. Data Integrity: RDBMS enforces the integrity of data by ensuring that the data is consistent and accurate. This is done through the use of data constraints, such as unique and foreign key constraints, which prevent data from being entered incorrectly.
2. Scalability: RDBMS can handle large amounts of data and can scale to accommodate growth.
3. Data Security: RDBMS provides data security features such as authentication, authorization, and encryption to protect sensitive data.
4. Data Retrieval: RDBMS provides efficient data retrieval through the use of indexing and querying mechanisms.
5. Data Consistency: RDBMS maintains data consistency through the use of transaction management, which ensures that data is updated or deleted in a consistent and reliable manner.

Overall, RDBMS is a reliable, scalable, and secure way to manage data for industries, which is why it is widely used in a variety of industries, including finance, healthcare, e-commerce, and more.

Top of Form

1. Explain the relationship data model in depth.

Ans- The relationship data model is a conceptual model that represents data as a set of interrelated entities and the relationships between them. It is a way of organizing data in a structured and efficient manner by defining entities, their attributes, and the relationships between them. The relationship data model is widely used in database design and is a fundamental concept in relational database management systems (RDBMS).

There are three main components of the relationship data model: entities, attributes, and relationships.

1. Entities: An entity is a real-world object, concept, or thing that has attributes and can be uniquely identified. For example, a customer, an employee, or a product are all examples of entities. Each entity has a set of attributes that describe it, such as a customer's name, address, and email.
2. Attributes: An attribute is a characteristic or property of an entity. For example, a customer entity may have attributes such as name, address, phone number, and email. Attributes can be of different types such as numeric, string, date, or Boolean.
3. Relationships: Relationships are the connections between entities. For example, a customer may place an order for a product. This relationship is represented in the database as a link between the customer and product entities. Relationships can be of different types such as one-to-one, one-to-many, or many-to-many.

To create a database using the relationship data model, the following steps are generally followed:

1. Identify the entities: Identify the real-world objects or concepts that need to be represented in the database.
2. Define the attributes: For each entity, define the attributes that describe it.
3. Define the relationships: Identify the relationships between the entities and specify their type and cardinality.
4. Normalize the data: Normalize the data to eliminate redundancy and ensure data consistency.
5. Implement the database: Implement the database using a database management system (DBMS) such as Oracle, SQL Server, or MySQL.

Overall, the relationship data model is a powerful tool for representing complex data structures in a structured and efficient manner. It provides a way to organize data and its relationships, making it easier to manage and query data.

Top of Form

Bottom of Form

1. What is the importance of Relationships in a Database management system? Explain the type of relationships.

Ans- Relationships are a critical component of a database management system (DBMS) because they define how data in different tables is related to one another. Relationships ensure that data is stored in a structured and efficient manner, eliminating duplication and ensuring data consistency. Here are some of the key reasons why relationships are important in a DBMS:

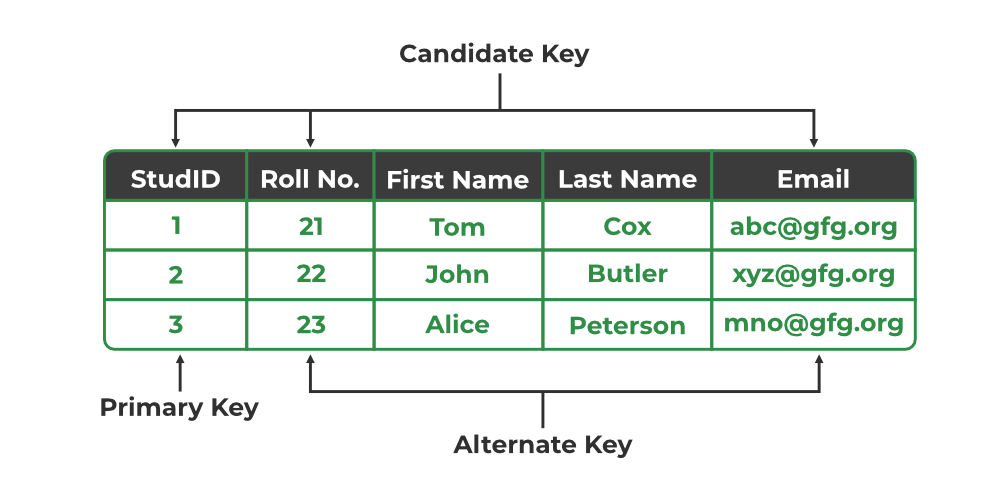
1. Data Integrity: Relationships help maintain the integrity of data by ensuring that data is entered and stored correctly. For example, a one-to-many relationship between a customer and order table ensures that each order is associated with a valid customer.
2. Efficiency: Relationships enable efficient data retrieval and processing. For example, when two tables are joined based on a common key, the DBMS can retrieve data from both tables in a single query.
3. Data Consistency: Relationships help ensure data consistency by enforcing referential integrity. For example, when a record is deleted from a parent table, any related records in child tables are also automatically deleted or updated.

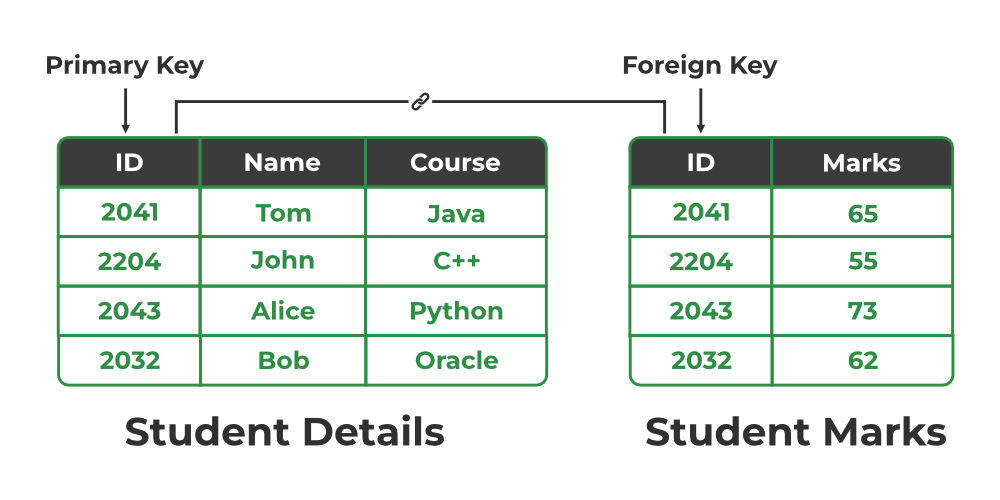
There are three types of relationships in a DBMS:

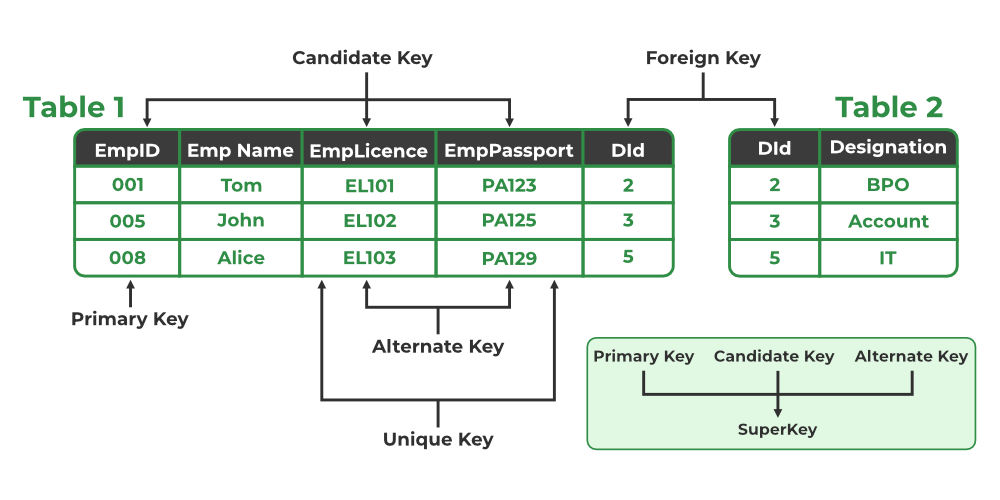
1. One-to-One Relationship: In a one-to-one relationship, one record in a table is related to one and only one record in another table. For example, a person may have one passport, and a passport may be associated with one person.
2. One-to-Many Relationship: In a one-to-many relationship, one record in a table is related to many records in another table. For example, a customer may have many orders, but each order is associated with only one customer.
3. Many-to-Many Relationship: In a many-to-many relationship, many records in one table are related to many records in another table. For example, a student may enroll in many courses, and each course may have many students. To represent a many-to-many relationship, a junction table is created that contains foreign keys from both tables.
4. Explain the different types of keys in RDBMS considering a real-life scenario.

Ans- In an RDBMS, keys are used to uniquely identify records in a table. Keys play a critical role in maintaining data integrity, ensuring that each record in a table is unique and can be easily identified. Here are the different types of keys in RDBMS, along with a real-life scenario for each:

1. Primary Key: A primary key is a unique identifier for a record in a table. It ensures that each record in the table is unique and can be easily identified. For example, in a customer table, the customer ID could be the primary key, ensuring that each customer has a unique identifier.
2. Foreign Key: A foreign key is a field in a table that refers to the primary key of another table. It is used to create relationships between tables, ensuring that data is stored in a structured and efficient manner. For example, in an order table, the customer ID field could be a foreign key that refers to the customer table, ensuring that each order is associated with a valid customer.
3. Candidate Key: A candidate key is a field or combination of fields in a table that could serve as a primary key. It is an alternate key that is unique and can be used to identify each record in a table. For example, in a product table, both the product ID and the product name could be candidate keys.
4. Composite Key: A composite key is a combination of two or more fields in a table that together uniquely identify a record. It is used when a single field cannot uniquely identify a record. For example, in an order table, the combination of the order ID and the order date could be a composite key.
5. Super Key: A super key is a combination of fields in a table that uniquely identify a record. It is similar to a candidate key but may contain additional fields. For example, in a customer table, the combination of the customer ID and the customer email could be a super key.







1. Write a short note on Single Responsibility Principle.

Ans- The Single Responsibility Principle (SRP) is a design principle in software engineering that states that a class or module should have only one reason to change. It emphasizes that a class or module should have a single responsibility, and that responsibility should be encapsulated within that class or module. In other words, a class should have only one job or purpose.

The SRP is important because it promotes high cohesion, which means that the code in a module is related and works together to achieve a specific task. By keeping responsibilities separate, changes to one responsibility will not affect the other responsibilities. This makes the code easier to maintain, test, and modify, and reduces the risk of introducing bugs or errors.

For example, consider a class that handles both database operations and user interface interactions. This violates the SRP because it has two responsibilities that may change for different reasons. If there is a change in the database schema, it may impact the user interface, even though they are unrelated. To adhere to the SRP, the database operations and user interface interactions should be separated into two separate classes or modules.

Overall, the SRP is an important principle in software engineering that helps to improve the design and maintainability of code. By separating responsibilities into smaller, well-defined modules, we can create code that is easier to understand, test, and maintain.

1. Explain the different types of errors that could arise in a denormalized database.

Ans- A denormalized database is a database that has been optimized for performance by combining multiple tables into a single table or duplicating data across tables. While denormalization can improve performance, it can also introduce several types of errors. Here are some of the errors that could arise in a denormalized database:

1. Data Redundancy Errors: Data redundancy errors occur when data is duplicated across tables. In a denormalized database, this is a common occurrence since data is duplicated to optimize performance. However, if the duplicated data is not updated correctly, it can lead to inconsistencies in the database.
2. Update Anomalies: Update anomalies occur when a change to one record in a denormalized database requires updates to multiple records. For example, if a customer's name is changed in a denormalized database that combines customer and order data, all orders associated with that customer must also be updated.
3. Insertion Anomalies: Insertion anomalies occur when it is impossible to insert a record into a table without also adding data to another table. In a denormalized database, this can happen if a record in one table references a record in another table, but that referenced record does not exist yet.
4. Deletion Anomalies: Deletion anomalies occur when deleting a record from a table inadvertently deletes data from another table. For example, if a denormalized database combines customer and order data, deleting a customer record may also delete all orders associated with that customer.
5. Inconsistency Errors: Inconsistency errors occur when data in a denormalized database is inconsistent due to data duplication or update anomalies. This can lead to incorrect or unexpected results when querying the database.
6. What is normalization and what is the need for normalization?

Ans- **Normalization** is the process of structuring and handling the relationship between data to minimize redundancy in the relational table and avoid unnecessary anomalies properties from the database like insertion, update and delete. It helps to divide large database tables into smaller tables and make a relationship between them. It can remove the redundant data and ease to add, manipulate or delete table fields.

A normalization defines rules for the relational table as to whether it satisfies the normal form. **A normal form** is a process that evaluates each relation against defined criteria and removes the multivalued, joins, functional and trivial dependency from a relation. If any data is updated, deleted or inserted, it does not cause any problem for database tables and help to improve the relational table' integrity and efficiency.

Objective of Normalization

1. It is used to remove the duplicate data and database anomalies from the relational table.
2. Normalization helps to reduce redundancy and complexity by examining new data types used in the table.
3. It is helpful to divide the large database table into smaller tables and link them using relationship.
4. It avoids duplicate data or no repeating groups into a table.
5. It reduces the chances for anomalies to occur in a database.

without Normalization , we may face many issues such as

**Insertion anomaly**: It occurs when we cannot insert data to the table without the presence of another attribute

**Update anomaly:**  It is a data inconsistency that results from data redundancy and a partial update of data.

**Deletion Anomaly:** It occurs when certain attributes are lost because of the deletion of other attributes.

1. List out different levels of normalization and explain them in detail.

Ans- Following are the various types of Normal forms:

**First Normal Form (1NF)**

Tables are in 1NF if

* There are no repeated rows of data
* Columns only contain a single value
* The Table has a primary key

Example:

|  |  |  |
| --- | --- | --- |
| Student | Year | Class |
| Paul Dawson | 11 | Math |
| Peggy Mitchell | 10 | History |
| Paul Dawson | 11 | Math |
| Brian Cox | 8 | English, Chemistry |
| Linda Marsh | 7 | Math, History, Biology |

As you can see that Paul Dawson is repeated two times with same data in the 1st and the 3rd row .

And also the class column, contain multiple values .

So this cleary indicate this Table in not in 1NF.

|  |  |  |
| --- | --- | --- |
| Student | Year | Class |
| Paul Dawson | 11 | Math |
| Peggy Mitchell | 10 | History |
| Brian Cox | 8 | English |
| Brian Cox | 8 | Chemistry |
| Linda Marsh | 7 | Math |
| Linda Marsh | 7 | History |
| Linda Marsh | 7 | Biology |

Now this Table is in 1NF as it follows the above mentioned three rules.

**Second Normal Form (2NF)**

Tables are in 2NF if

* They confirm to 1NF
* Every Column that is not a primary key of the table is dependent on the whole of the primary key

|  |  |  |  |
| --- | --- | --- | --- |
| Student | Subject | Grade | Age |
| Natasha Williams | Maths | A | 15 |
| Natasha Williams | English | B | 15 |
| Daniel James | Maths | C | 16 |
| Simon Brown | Chemistry | A | 14 |
| Emma Thomas | Geography | B | 14 |

The primary key of the above table is (Student + Subject)

As we can see that the above Tables follows the rules of 1NF so it is in 1NF.

Now we will check for 2NF.

There are 2 non-primary keys Grade and Age.

We can see that the Grade is dependent on both Student and Subject column.

But the age is dependent only on the Student and it doesn’t depend on the Subject. So the Age is not dependent on the whole of the primary key. So it is not in 2NF.

To make it into 2NF lets divide this Table into two tables.

1st Table

|  |  |  |
| --- | --- | --- |
| Student | Subject | Grade |
| Natasha Williams | Maths | A |
| Natasha Williams | English | B |
| Daniel James | Maths | C |
| Simon Brown | Chemistry | A |
| Emma Thomas | Geography | B |

In the above Table the Grade column is dependent on the whole of the primary key (i.e. Student + Subject). So this Table is in 2NF

2nd Table

|  |  |
| --- | --- |
| Student | Age |
| Natasha Williams | 15 |
| Daniel James | 16 |
| Simon Brown | 14 |
| Emma Thomas | 14 |

In the above Table the Age column is dependent on the whole of the primary key (i.e. Student ). So this Table is also in 2NF

**Third Normal Form (3NF)**

Tables are in 3NF if

* They confirm to 2NF
* Every Column that is not a primary key is only dependent on the whole of primary key

|  |  |  |  |
| --- | --- | --- | --- |
| Subject | Year | Star Pupil | Star Pupil DOB |
| Math | 2015 | Matthew Taylor | 1999-03-21 |
| Physics | 2015 | William Edwards | 1999-09-15 |
| Chemistry | 2015 | Georgina Simon | 1998-11-04 |
| Math | 2016 | Benjamin Long | 2000-05-02 |
| Physics | 2016 | William Edwards | 1999-09-15 |

The Primary key is Subject + Year .

As we can see that there is no repeated rows of data and every cell has a single value so , the above Table is in 1NF.

Check for 2NF

We have two non-primary key Star Pupil column and Star Pupil DOB.

And we can clearly see that Star Pupil Column is dependent on both columns i.e. (Subject + Year).

And we can also say that Star Pupil DOB column is also dependent on both columns i.e. (Subject + Year).

So the above Table is in 2NF.

Check for 3NF

We can clearly see that Star Pupil Column is only dependent on the whole of the Primary Key (i.e. Subject + Year).

We can also say the Star Pupil DOB column is dependent on the whole of the Primary Key (i.e. Subject + Year). But it is also dependent on the Star Pupil Column, So we have a 2 step dependency.

So the Star Pupil DOB column depend on a column which is not a primary key.

To make it into 3NF we have split the above table into two tables.

1st Table

|  |  |  |
| --- | --- | --- |
| Subject | Year | Star Pupil |
| Math | 2015 | Matthew Taylor |
| Physics | 2015 | William Edwards |
| Chemistry | 2015 | Georgina Simon |
| Math | 2016 | Benjamin Long |
| Physics | 2016 | William Edwards |

Now we can clearly say that the Star Pupil Column is only dependent on the whole of the Primary key (i.e. Subject + Year) . So this Table is in 3NF.

2nd Table

|  |  |
| --- | --- |
| Star Pupil | Star Pupil DOB |
| Matthew Taylor | 1999-03-21 |
| William Edwards | 1999-09-15 |
| Georgina Simon | 1998-11-04 |
| Benjamin Long | 2000-05-02 |

Now we can clearly say that the Star Pupil DOB Column is only dependent on the whole of the Primary key (i.e. Star Pupil) . So this Table is also in 3NF.

**Boyce-Codd Normal Form (BCNF)**

The BCNF requires that for every functional dependency X → Y, where X is a set of attributes and Y is a single attribute, the determinant (X) must be a superkey. In other words, every non-trivial functional dependency in the table must be based on a candidate key.

Consider a table called "Students" that stores information about students in a school. The table has the following attributes:

* StudentID (Primary Key)
* StudentName
* Class
* Teacher

Let's assume that StudentID is the only candidate key for the Students table.

Students:

| **StudentID** | **StudentName** | **Class** | **Teacher** |
| --- | --- | --- | --- |
| 1 | John | A | Mr. Smith |
| 2 | Jane | B | Ms. Johnson |
| 3 | Mike | A | Mr. Smith |

In this example, the table satisfies BCNF because each attribute depends solely on the candidate key (StudentID). There are no non-trivial functional dependencies violating BCNF. Each student's name, class, and teacher are uniquely determined by their StudentID, and there are no partial dependencies or transitive dependencies present.

1. What are joins and why do we need them?

Ans- JOINS in SQL are commands which are used to combine rows from two or more tables, based on a related column between those tables.  There are predominantly used when a user is trying to extract data from tables which have one-to-many or many-to-many relationships between them.

Joins are used in relational databases to combine data from two or more tables into a single result set based on related columns between them.

In a database, data is often stored in multiple tables, and there may be relationships between those tables. For example, one table might contain information about customers, and another table might contain information about their orders. The orders table might have a column that contains the customer ID, which is a unique identifier for each customer.

To get a complete view of the data, we may need to combine data from both tables into a single result set. This is where joins come in. Joins allow us to specify how the tables are related to each other and return the data that satisfies that relationship.

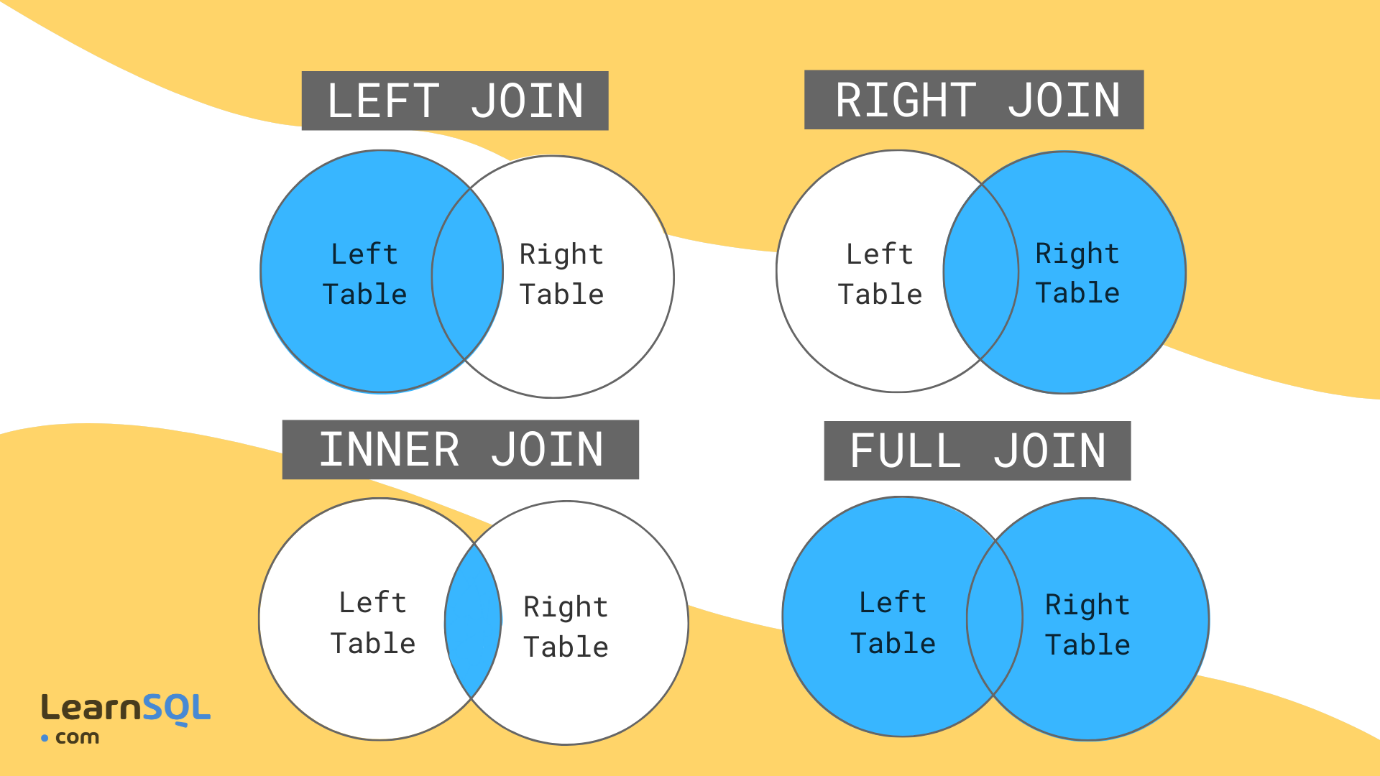
There are different types of joins such as inner join, left join, right join, and full outer join, and the type of join used depends on the specific requirements of the query. Joins are important because they enable us to query data from multiple tables and retrieve a more complete and meaningful result set than we could obtain by querying each table individually.

1. Explain the different types of joins?

Ans- **Types of Join statements**

The type of join statement you use depends on your use case. There are four different types of join operations:

* **(INNER) JOIN:** Returns dataset that have matching values in both tables
* **LEFT (OUTER) JOIN:** Returns all records from the left table and matched records from the right
* **RIGHT (OUTER) JOIN:** Returns all records from the right table and the matched records from the left
* **FULL (OUTER) JOIN:** Returns all records when there is a match in either the left table or right table



**INNER JOIN**

Table: Employees

| **EmployeeID** | **EmployeeName** | **DepartmentID** |
| --- | --- | --- |
| 1 | John Smith | 1 |
| 2 | Jane Doe | 2 |
| 3 | David Brown | 1 |
| 4 | Sarah Johnson | 3 |

Table: Departments

| **DepartmentID** | **DepartmentName** |
| --- | --- |
| 1 | Sales |
| 2 | Marketing |
| 3 | HR |

Example: Joining Employees and Departments tables

**sql code**

SELECT Employees.EmployeeName, Departments.DepartmentName FROM Employees INNER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;

The query above will perform an inner join on the "Employees" and "Departments" tables, based on the matching "DepartmentID" values. It will select the "EmployeeName" column from the "Employees" table and the "DepartmentName" column from the "Departments" table.

The result of this query will be:

| **EmployeeName** | **DepartmentName** |
| --- | --- |
| John Smith | Sales |
| Jane Doe | Marketing |
| David Brown | Sales |
| Sarah Johnson | HR |

**LEFT JOIN**

Table: Customers

| **CustomerID** | **CustomerName** | **Country** |
| --- | --- | --- |
| 1 | John Smith | USA |
| 2 | Jane Doe | Canada |
| 3 | David Brown | UK |
| 4 | Sarah Johnson | Australia |

Table: Orders

| **OrderID** | **OrderDate** | **CustomerID** | **TotalAmount** |
| --- | --- | --- | --- |
| 1 | 2023-05-01 | 1 | 100.00 |
| 2 | 2023-05-02 | 2 | 50.00 |
| 3 | 2023-05-03 | 1 | 200.00 |
| 4 | 2023-05-04 | 3 | 150.00 |

Example: LEFT JOIN between Customers and Orders tables

**sql code**

SELECT Customers.CustomerName, Orders.OrderID, Orders.OrderDate FROM Customers LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

The query above performs a LEFT JOIN between the "Customers" and "Orders" tables based on the matching "CustomerID" values. It selects the "CustomerName" column from the "Customers" table and the "OrderID" and "OrderDate" columns from the "Orders" table.

The result of this query will be:

| **CustomerName** | **OrderID** | **OrderDate** |
| --- | --- | --- |
| John Smith | 1 | 2023-05-01 |
| Jane Doe | 2 | 2023-05-02 |
| David Brown | 4 | 2023-05-04 |
| Sarah Johnson | NULL | NULL |

This result shows the customer names along with their respective order information (ID and date). Notice that the customer "Sarah Johnson" appears in the result even though she doesn't have any orders. In the result, the "NULL" values indicate the absence of matching records from the "Orders" table for the customer. This is because a LEFT JOIN returns all rows from the left table (Customers) and the matching rows from the right table (Orders).

**RIGHT JOIN**

Table: Employees

| **EmployeeID** | **EmployeeName** | **DepartmentID** |
| --- | --- | --- |
| 1 | John Smith | 1 |
| 2 | Jane Doe | 2 |
| 3 | David Brown | 1 |
| 4 | Sarah Johnson | 3 |

Table: Salaries

| **EmployeeID** | **Salary** |
| --- | --- |
| 1 | 5000 |
| 2 | 6000 |
| 5 | 4500 |
| 6 | 5500 |

Example: RIGHT JOIN between Employees and Salaries tables

**sql code**

SELECT Employees.EmployeeName, Salaries.Salary FROM Employees RIGHT JOIN Salaries ON Employees.EmployeeID = Salaries.EmployeeID;

The query above performs a RIGHT JOIN between the "Employees" and "Salaries" tables based on the matching "EmployeeID" values. It selects the "EmployeeName" column from the "Employees" table and the "Salary" column from the "Salaries" table.

The result of this query will be:

| **EmployeeName** | **Salary** |
| --- | --- |
| John Smith | 5000 |
| Jane Doe | 6000 |
| NULL | 4500 |
| NULL | 5500 |

This result shows the employee names along with their respective salaries. Notice that there are two NULL values in the "EmployeeName" column. This is because there are two salaries in the "Salaries" table (with EmployeeID 5 and 6) that do not have matching employees in the "Employees" table. In a RIGHT JOIN, all rows from the right table (Salaries) are returned, and the matching rows from the left table (Employees) are included.