Relational Data Model **2nd sem (DBMS)**

What is Relational Model?

The relational model represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.

The table name and column names are helpful to interpret the meaning of values in each row. The data are represented as a set of relations. In the relational model, data are stored as tables. However, the physical storage of the data is independent of the way the data are logically organized.

Some popular Relational Database management systems are:

- DB2 and Informix Dynamic Server - IBM
- Oracle and RDB – Oracle
- SQL Server and Access - Microsoft

Relational Model Concepts

- Attribute: Each column in a Table. Attributes are the properties which define a relation. e.g., Student_Rollno, NAME,etc.
- Tables – In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.
- Tuple – It is nothing but a single row of a table, which contains a single record.
- Relation Schema: A relation schema represents the name of the relation with its attributes.
- Degree: The total number of attributes which in the relation is called the degree of the relation.
- Cardinality: Total number of rows present in the Table.
- Column: The column represents the set of values for a specific attribute.
- Relation instance – Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.
- Relation key - Every row has one, two or multiple attributes, which is called relation key.
- Attribute domain – Every attribute has some pre-defined value and scope which is known as attribute domain

Relational Data Model **2nd sem (DBMS)**

Relational Integrity constraints

Relational Integrity constraints is referred to conditions which must be present for a valid relation. These integrity constraints are derived from the rules in the mini-world that the database represents.

There are many types of integrity constraints. Constraints on the Relational database management system is mostly divided into three main categories are:

- Domain constraints
- Key constraints
- Referential integrity constraints

Domain Constraints

Domain constraints can be violated if an attribute value is not appearing in the corresponding domain or it is not of the appropriate data type.

Domain constraints specify that within each tuple, and the value of each attribute must be unique. This is specified as data types which include standard data type's integers, real numbers, characters, Booleans, variable length strings, etc.

Key constraints

An attribute that can uniquely identify a tuple in a relation is called the key of the table. The value of the attribute for different tuples in the relation has to be unique.

Example:

In the given table, CustomerID is a key attribute of Customer Table. It is most likely to have a single key for one customer, CustomerID =1 is only for the CustomerName =" Google".

Referential integrity constraints

Referential integrity constraints is based on the concept of Foreign Keys. A foreign key is an important attribute of a relation which should be referred to in other relationships. Referential integrity constraint state happens where relation refers to a key attribute of a different or same relation. However, that key element must exist in the table.

Operations in Relational Model

Four basic update operations performed on relational database model are

Insert, update, delete and select.

- Insert is used to insert data into the relation
- Delete is used to delete tuples from the table.
- Modify allows you to change the values of some attributes in existing tuples.
- Select allows you to choose a specific range of data.

Relational Data Model **2nd sem (DBMS)**

Best Practices for creating a Relational Model

- Data need to be represented as a collection of relations
- Each relation should be depicted clearly in the table
- Rows should contain data about instances of an entity
- Columns must contain data about attributes of the entity
- Cells of the table should hold a single value
- Each column should be given a unique name
- No two rows can be identical
- The values of an attribute should be from the same domain

Advantages of using Relational model

- Simplicity: A relational data model is simpler than the hierarchical and network model.
- Structural Independence: The relational database is only concerned with data and not with a structure. This can improve the performance of the model.
- Easy to use: The relational model is easy as tables consisting of rows and columns is quite natural and simple to understand
- Query capability: It makes possible for a high-level query language like SQL to avoid complex database navigation.
- Data independence: The structure of a database can be changed without having to change any application.
- Scalable: Regarding a number of records, or rows, and the number of fields, a database should be enlarged to enhance its usability.

Disadvantages of using Relational model

- Few relational databases have limits on field lengths which can't be exceeded.
- Relational databases can sometimes become complex as the amount of data grows, and the relations between pieces of data become more complicated.
- Complex relational database systems may lead to isolated databases where the information cannot be shared from one system to another.

Basic Relational Algebra Operations:

Relational Algebra divided in various groups

Unary Relational Operations

- SELECT (symbol: σ)
- PROJECT (symbol: π)
- RENAME (symbol:)

Relational Data Model **2nd sem (DBMS)**

Relational Algebra Operations From Set Theory

- UNION (υ)
- INTERSECTION ( ),
- DIFFERENCE (-)
- CARTESIAN PRODUCT (x)

Binary Relational Operations

- JOIN
- DIVISION

SELECT (σ)

The SELECT operation is used for selecting a subset of the tuples according to a given selection condition. Sigma(σ)Symbol denotes it. It is used as an expression to choose tuples which meet the selection condition. Select operation selects tuples that satisfy a given predicate.

$\sigma_p(r)$

σ is the predicate

r stands for relation which is the name of the table

p is prepositional logic

Example 1

$\sigma_{\text{topic = "Database"}}$ (Tutorials)

Output - Selects tuples from Tutorials where topic = 'Database'.

Example 2

$\sigma_{\text{topic = "Database" and author = "class"}}$(Tutorials)

Output - Selects tuples from Tutorials where the topic is 'Database' and 'author' is class.

Example 3

$\sigma_{\text{sales > 50000}}$ (Customers)

Output - Selects tuples from Customers where sales is greater than 50000

Projection(π)

The projection eliminates all attributes of the input relation but those mentioned in the projection list. The projection method defines a relation that contains a vertical subset of Relation.

This helps to extract the values of specified attributes to eliminate duplicate values. (Pi) The symbol used to choose attributes from a relation. This operation helps you to keep specific columns from a relation and discards the other columns.

Example of Projection:

Consider the following table

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

Here, the projection of CustomerName and status will give

Π CustomerName, Status (Customers)

CustomerName	Status
Google	Active
Amazon	Active

Apple	Inactive
Alibaba	Active

Union operation (υ)

UNION is symbolized by ∪ symbol. It includes all tuples that are in tables A or in B. It also eliminates duplicate tuples. So, set A UNION set B would be expressed as:

The result <- A ∪ B

For a union operation to be valid, the following conditions must hold -

- R and S must be the same number of attributes.
- Attribute domains need to be compatible.
- Duplicate tuples should be automatically removed.

Example

Consider the following tables.

Table A		**Table B**	
column 1	**column 2**	**column 1**	**column 2**
1	1	1	1
1	2	1	3

A ∪ B gives

Table A U B	
column 1	**column 2**
1	1
1	2
1	3

Set Difference (-)

- Symbol denotes it. The result of A - B, is a relation which includes all tuples that are in A but not in B.

- The attribute name of A has to match with the attribute name in B.
- The two-operand relations A and B should be either compatible or Union compatible.
- It should be defined relation consisting of the tuples that are in relation A, but not in B.

Example

A-B

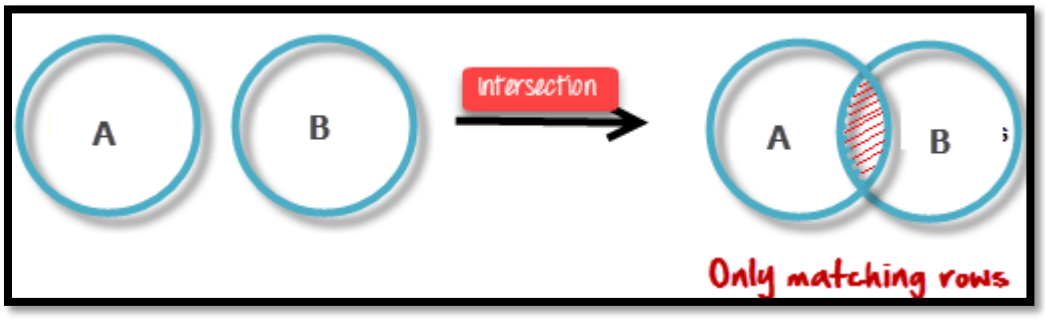
Table A - B	
column 1	**column 2**
1	2

Intersection

An intersection is defined by the symbol ∩

A ∩ B

Defines a relation consisting of a set of all tuple that are in both A and B. However, A and B must be union-compatible.



Example:

A ∩ B

Table A ∩ B	
column 1	**column 2**
1	1

Cartesian product(X)

This type of operation is helpful to merge columns from two relations. Generally, a Cartesian product is never a meaningful operation when it performs alone. However, it becomes meaningful when it is followed by other operations.

Example – Cartesian product

$$\sigma_{\text{column 2} = \text{'1'}} (A \times B)$$

Output – The above example shows all rows from relation A and B whose column 2 has value 1

σ column 2 = '1' (A X B)	
column 1	**column 2**
1	1
1	1

Join Operations

Join operation is essentially a Cartesian product followed by a selection criterion.

Join operation denoted by ⋈.

JOIN operation also allows joining variously related tuples from different relations.

Types of JOIN:

Various forms of join operation are:

Inner Joins:

- Theta join
- EQUI join
- Natural join

Outer join:

- Left Outer Join
- Right Outer Join
- Full Outer Join

Inner Join:

In an inner join, only those tuples that satisfy the matching criteria are included, while the rest are excluded. Let's study various types of Inner Joins:

Theta Join:

The general case of JOIN operation is called a Theta join. It is denoted by symbol **θ**

Example

$A \bowtie_\theta B$

Theta join can use any conditions in the selection criteria.

For example:

$A \bowtie_{A.column\ 2 > B.column\ 2} (B)$

A ⋈ A.column 2 > B.column 2 (B)	
column 1	**column 2**
1	2

EQUI join:

When a theta join uses only equivalence condition, it becomes a equi join.

For example:

$A \bowtie_{A.column\ 2 = B.column\ 2} (B)$

A ⋈ A.column 2 = B.column 2 (B)	
column 1	**column 2**
1	1

EQUI join is the most difficult operations to implement efficiently in an RDBMS and one reason why RDBMS have essential performance problems.

NATURAL JOIN (⋈)

Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same.

Example

Consider the following two tables

C	
Num	**Square**
2	4
3	9

D	
Num	**Cube**
2	8
3	27

C ⋈ D

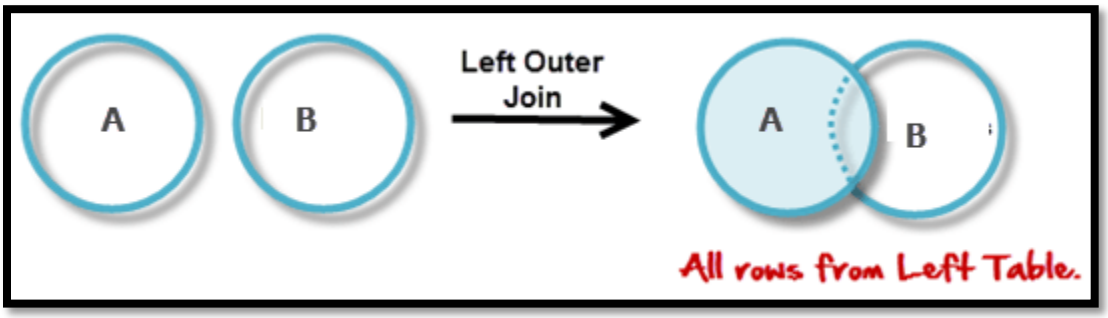
C ⋈ D		
Num	**Square**	**Cube**
2	4	4
3	9	27

OUTER JOIN

In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.

Left Outer Join(A ⟕ B)

In the left outer join, operation allows keeping all tuple in the left relation. However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values.



Relational Data Model **2nd sem (DBMS)**

Consider the following 2 Tables

A	
Num	**Square**
2	4
3	9
4	16

B	
Num	**Cube**
2	8
3	18
5	75

A ⟗ B

A ⋈ B		
Num	**Square**	**Cube**
2	4	4
3	9	9
4	16	-

Right Outer Join: (A ⟖ B)

In the right outer join, operation allows keeping all tuple in the right relation. However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.

A ⟗ B

A ⋈ B		
Num	**Cube**	**Square**
2	8	4
3	18	9
5	75	-

Full Outer Join: (A ⟗ B)

In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.

A ⟗ B

A ⋈ B		
Num	**Cube**	**Square**
2	4	8
3	9	18

4	16	-
5	-	75

Summary

Operation	**Purpose**
Select(σ)	The SELECT operation is used for selecting a subset of the tuples according to a given selection condition
Projection(π)	The projection eliminates all attributes of the input relation but those mentioned in the projection list.
Union Operation(∪)	UNION is symbolized by symbol. It includes all tuples that are in tables A or in B.
Set Difference(-)	- Symbol denotes it. The result of A - B, is a relation which includes all tuples that are in A but not in B.
Intersection(∩)	Intersection defines a relation consisting of a set of all tuple that are in both A and B.
Cartesian Product(X)	Cartesian operation is helpful to merge columns from two relations.
Inner Join	Inner join, includes only those tuples that satisfy the matching criteria.
Theta Join(θ)	The general case of JOIN operation is called a Theta join. It is denoted by symbol θ.
EQUI Join	When a theta join uses only equivalence condition, it becomes a equi join.
Natural Join(⋈)	Natural join can only be performed if there is a common attribute (column) between the relations.
Outer Join	In an outer join, along with tuples that satisfy the matching criteria.

Left Outer Join(⟕)	In the left outer join, operation allows keeping all tuple in the left relation.
Right Outer join(⟖)	In the right outer join, operation allows keeping all tuple in the right relation.
Full Outer Join(⟗)	In a full outer join, all tuples from both relations are included in the result irrespective of the matching condition.

SQL Syntax

The SQL SELECT Statement

The SELECT statement is used to select data from a database.

The data returned is stored in a result table, called the result-set.

SELECT Syntax

SELECT column1, column2, ...

FROM table_name;

SELECT * FROM table_name;

SELECT Column Example

The following SQL statement selects the "CustomerName" and "City" columns from the "Customers" table:

Example

SELECT CustomerName, City FROM Customers;

SELECT * Example

The following SQL statement selects all the columns from the "Customers" table:

Example

SELECT * FROM Customers;

The SQL SELECT DISTINCT Statement

The SELECT DISTINCT statement is used to return only distinct (different) values.

SELECT DISTINCT Syntax

SELECT DISTINCT column1, column2, ...

FROM table_name;

SELECT Example Without DISTINCT

The following SQL statement selects ALL (including the duplicates) values from the "Country" column in the "Customers" table:

Example

SELECT Country FROM Customers;

SELECT DISTINCT Examples

The following SQL statement selects only the DISTINCT values from the "Country" column in the "Customers" table:

Example

SELECT DISTINCT Country FROM Customers;

The SQL WHERE Clause

The WHERE clause is used to filter records.

The WHERE clause is used to extract only those records that fulfill a specified condition.

WHERE Syntax

SELECT column1, column2, ...

FROM table_name

WHERE condition;

WHERE Clause Example

The following SQL statement selects all the customers from the country "Mexico", in the "Customers" table:

Example

```
SELECT * FROM Customers
WHERE Country='Mexico';
```

Text Fields vs. Numeric Fields

SQL requires single quotes around text values (most database systems will also allow double quotes).

However, numeric fields should not be enclosed in quotes:

Example

```
SELECT * FROM Customers
WHERE CustomerID=1;
```

The SQL ORDER BY Keyword

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.

ORDER BY Syntax

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

ORDER BY Example

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" column:

Example

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

ORDER BY DESC Example

The following SQL statement selects all customers from the "Customers" table, sorted DESCENDING by the "Country" column:

Example

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

ORDER BY Several Columns Example

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" and the "CustomerName" column. This means that it orders by Country, but if some rows have the same Country, it orders them by CustomerName:

Example

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

The SQL INSERT INTO Statement

The INSERT INTO statement is used to insert new records in a table.

INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two ways.

The first way specifies both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

INSERT INTO Example

The following SQL statement inserts a new record in the "Customers" table:

Example

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
```

Insert Data Only in Specified Columns

It is also possible to only insert data in specific columns.

The following SQL statement will insert a new record, but only insert data in the "CustomerName", "City", and "Country" columns (CustomerID will be updated automatically):

Example

```
INSERT INTO Customers (CustomerName, City, Country)
VALUES ('Cardinal', 'Stavanger', 'Norway');
```

The SQL UPDATE Statement

The UPDATE statement is used to modify the existing records in a table.

UPDATE Syntax

UPDATE table_name

SET column1 = value1, column2 = value2, ...

WHERE condition;

UPDATE Table

The following SQL statement updates the first customer (CustomerID = 1) with a new contact person and a new city.

Example

UPDATE Customers

SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'

WHERE CustomerID = 1;

UPDATE Multiple Records

It is the WHERE clause that determines how many records will be updated.

The following SQL statement will update the contactname to "Juan" for all records where country is "Mexico":

Example

UPDATE Customers

SET ContactName='Juan'

WHERE Country='Mexico';

The SQL DELETE Statement

The DELETE statement is used to delete existing records in a table.

DELETE Syntax

DELETE FROM table_name WHERE condition;

SQL DELETE Example

The following SQL statement deletes the customer "Alfreds Futterkiste" from the "Customers" table:

Example

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

Delete All Records

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

```
DELETE FROM table_name;
```

The following SQL statement deletes all rows in the "Customers" table, without deleting the table:

Example

```
DELETE FROM Customers;
```

The SQL LIKE Operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the LIKE operator:

% - The percent sign represents zero, one, or multiple characters

_ - The underscore represents a single character

LIKE Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

SQL LIKE Examples

The following SQL statement selects all customers with a CustomerName starting with "a":

Example

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

The following SQL statement selects all customers with a CustomerName ending with "a":

Example

```
SELECT * FROM Customers
```

WHERE CustomerName LIKE '%a';

The following SQL statement selects all customers with a CustomerName that have "or" in any position:

Example

SELECT * FROM Customers

WHERE CustomerName LIKE '%or%';

The following SQL statement selects all customers with a CustomerName that have "r" in the second position:

Example

SELECT * FROM Customers

WHERE CustomerName LIKE '_r%';

The following SQL statement selects all customers with a CustomerName that starts with "a" and are at least 3 characters in length:

Example

SELECT * FROM Customers

WHERE CustomerName LIKE 'a__%';

The following SQL statement selects all customers with a ContactName that starts with "a" and ends with "o":

Example

SELECT * FROM Customers

WHERE ContactName LIKE 'a%o';

The following SQL statement selects all customers with a CustomerName that does NOT start with "a":

Example

SELECT * FROM Customers

WHERE CustomerName NOT LIKE 'a%';

The SQL CREATE DATABASE Statement

The CREATE DATABASE statement is used to create a new SQL database.

Syntax

CREATE DATABASE databasename;

CREATE DATABASE Example

The following SQL statement creates a database called "testDB":

Example

```
CREATE DATABASE testDB;
```

The SQL DROP DATABASE Statement

The DROP DATABASE statement is used to drop an existing SQL database.

Syntax

```
DROP DATABASE databasename;
```

DROP DATABASE Example

The following SQL statement drops the existing database "testDB":

Example

```
DROP DATABASE testDB;
```

The SQL CREATE TABLE Statement

The CREATE TABLE statement is used to create a new table in a database.

Syntax

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

SQL CREATE TABLE Example

The following example creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City:

Example

```
CREATE TABLE Persons (
   PersonID int,
   LastName varchar(255),
   FirstName varchar(255),
```

```
    Address varchar(255),
    City varchar(255)
);
```

The SQL DROP TABLE Statement

The DROP TABLE statement is used to drop an existing table in a database.

Syntax

```
DROP TABLE table_name;
```

SQL DROP TABLE Example

The following SQL statement drops the existing table "Shippers":

Example

```
DROP TABLE Shippers;
```

SQL TRUNCATE TABLE

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

Syntax

```
TRUNCATE TABLE table_name;
```

SQL JOIN

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

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

SQL INNER JOIN Keyword

The INNER JOIN keyword selects records that have matching values in both tables.

INNER JOIN Syntax

SELECT column_name(s)

FROM table1

INNER JOIN table2

ON table1.column_name = table2.column_name;

SQL INNER JOIN Example

The following SQL statement selects all orders with customer information:

Example

SELECT Orders.OrderID, Customers.CustomerName

FROM Orders

INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

JOIN Three Tables

The following SQL statement selects all orders with customer and shipper information:

Example

SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName

FROM ((Orders

INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)

INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);

SQL LEFT JOIN Keyword

The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

LEFT JOIN Syntax

SELECT column_name(s)

```
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

SQL LEFT JOIN Example

The following SQL statement will select all customers, and any orders they might have:

Example

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.

RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

SQL RIGHT JOIN Example

The following SQL statement will return all employees, and any orders they might have placed:

Example

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

SQL FULL OUTER JOIN Keyword

The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

Note: FULL OUTER JOIN can potentially return very large result-sets!

Tip: FULL OUTER JOIN and FULL JOIN are the same.

FULL OUTER JOIN Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

SQL FULL OUTER JOIN Example

The following SQL statement selects all customers, and all orders:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

SQL Self JOIN

A self JOIN is a regular join, but the table is joined with itself.

Self JOIN Syntax

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

SQL Self JOIN Example

The following SQL statement matches customers that are from the same city:

Example

```
SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
```

```
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```


SQL Keywords

Keyword	Description
ADD	Adds a column in an existing table
ADD CONSTRAINT	Adds a constraint after a table is already created
ALTER	Adds, deletes, or modifies columns in a table, or changes the data type of a column in a table
ALTER COLUMN	Changes the data type of a column in a table

ALTER TABLE	Adds, deletes, or modifies columns in a table
ALL	Returns true if all of the subquery values meet the condition
AND	Only includes rows where both conditions is true
ANY	Returns true if any of the subquery values meet the condition
AS	Renames a column or table with an alias
ASC	Sorts the result set in ascending order
BACKUP DATABASE	Creates a back up of an existing database
BETWEEN	Selects values within a given range
CASE	Creates different outputs based on conditions
CHECK	A constraint that limits the value that can be placed in a column
COLUMN	Changes the data type of a column or deletes a column in a table

Relational Data Model **2nd sem (DBMS)**

CONSTRAINT	Adds or deletes a constraint
CREATE	Creates a database, index, view, table, or procedure
CREATE DATABASE	Creates a new SQL database
CREATE INDEX	Creates an index on a table (allows duplicate values)
CREATE OR REPLACE VIEW	Updates a view
CREATE TABLE	Creates a new table in the database
CREATE PROCEDURE	Creates a stored procedure
CREATE UNIQUE INDEX	Creates a unique index on a table (no duplicate values)
CREATE VIEW	Creates a view based on the result set of a SELECT statement
DATABASE	Creates or deletes an SQL database
DEFAULT	A constraint that provides a default value for a column

DELETE	Deletes rows from a table
DESC	Sorts the result set in descending order
DISTINCT	Selects only distinct (different) values
DROP	Deletes a column, constraint, database, index, table, or view
DROP COLUMN	Deletes a column in a table
DROP CONSTRAINT	Deletes a UNIQUE, PRIMARY KEY, FOREIGN KEY, or CHECK constraint
DROP DATABASE	Deletes an existing SQL database
DROP DEFAULT	Deletes a DEFAULT constraint
DROP INDEX	Deletes an index in a table
DROP TABLE	Deletes an existing table in the database
DROP VIEW	Deletes a view

EXEC	Executes a stored procedure
EXISTS	Tests for the existence of any record in a subquery
FOREIGN KEY	A constraint that is a key used to link two tables together
FROM	Specifies which table to select or delete data from
FULL OUTER JOIN	Returns all rows when there is a match in either left table or right table
GROUP BY	Groups the result set (used with aggregate functions: COUNT, MAX, MIN, SUM, AVG)
HAVING	Used instead of WHERE with aggregate functions
IN	Allows you to specify multiple values in a WHERE clause
INDEX	Creates or deletes an index in a table
INNER JOIN	Returns rows that have matching values in both tables
INSERT INTO	Inserts new rows in a table

INSERT INTO SELECT	Copies data from one table into another table
IS NULL	Tests for empty values
IS NOT NULL	Tests for non-empty values
JOIN	Joins tables
LEFT JOIN	Returns all rows from the left table, and the matching rows from the right table
LIKE	Searches for a specified pattern in a column
LIMIT	Specifies the number of records to return in the result set
NOT	Only includes rows where a condition is not true
NOT NULL	A constraint that enforces a column to not accept NULL values
OR	Includes rows where either condition is true
ORDER BY	Sorts the result set in ascending or descending order

OUTER JOIN	Returns all rows when there is a match in either left table or right table
PRIMARY KEY	A constraint that uniquely identifies each record in a database table
PROCEDURE	A stored procedure
RIGHT JOIN	Returns all rows from the right table, and the matching rows from the left table
ROWNUM	Specifies the number of records to return in the result set
SELECT	Selects data from a database
SELECT DISTINCT	Selects only distinct (different) values
SELECT INTO	Copies data from one table into a new table
SELECT TOP	Specifies the number of records to return in the result set
SET	Specifies which columns and values that should be updated in a table

Relational Data Model **2nd sem (DBMS)**

TABLE	Creates a table, or adds, deletes, or modifies columns in a table, or deletes a table or data inside a table
TOP	Specifies the number of records to return in the result set
TRUNCATE TABLE	Deletes the data inside a table, but not the table itself
UNION	Combines the result set of two or more SELECT statements (only distinct values)
UNION ALL	Combines the result set of two or more SELECT statements (allows duplicate values)
UNIQUE	A constraint that ensures that all values in a column are unique
UPDATE	Updates existing rows in a table
VALUES	Specifies the values of an INSERT INTO statement
VIEW	Creates, updates, or deletes a view
WHERE	Filters a result set to include only records that fulfill a specified condition