SQL Basics

Icon Software Solutions

Sandarsh
QA Lead

Agenda

- About RDBMS
- Normalization
- SQL Data Types
- SQL Constraints
- SQL Operators
- DDL Commands
- DML Commands
- Important Select Commands
- Functions
- Joins

A **Relational Database management System**(RDBMS) is a database management system based on the relational model introduced by E.F Codd (12 Codd's rule). In relational model, data is stored in **relations**(tables) and is represented in form of **tuples**(rows).

RDBMS is used to manage Relational database. **Relational database** is a collection of organized set of tables related to each other, and from which data can be accessed easily. Relational Database is the most commonly used database these days.

The key **difference** is that **RDBMS**(**relational database management system**) applications store data **in a** tabular form, while **DBMS** applications store data as files

Normalization is a database design technique which organizes tables in a manner that reduces redundancy and dependency of data.

It divides larger tables to smaller tables and links them using relationships

The total normalization process includes 8 normal forms.

Or different types of normalization…

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- Boyce-Codd Normal Form (BCNF)
- Fourth Normal Form (4NF)
- Fifth Normal Form (5NF)
- Domain/Key Normal Form(DKNF)
- Sixth Normal Form(6NF)

1NF (First Normal Form) Rules:

- Each table cell should contain a single value.
- Each record needs to be unique.

2NF (Second Normal Form) Rules:

- Rule 1- Be in 1NF
- Rule 2- Single Column Primary Key

3NF (Third Normal Form) Rules

- Rule 1- Be in 2NF
- Rule 2- Has no transitive functional dependencies

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

  Even when a database is in 3rd Normal Form, still there would be anomalies resulted if it has more than one **Candidate** Key.

  Sometimes is BCNF is also referred as **3.5 Normal Form.**

- **4NF (Fourth Normal Form) Rules**

  If no database table instance contains two or more, independent and multivalued data describing the relevant entity, then it is in 4th Normal Form.

5NF (Fifth Normal Form) Rules

A table is in 5th Normal Form only if it is in 4NF and it cannot be decomposed into any number of smaller tables without loss of data.

6NF (Sixth Normal Form) Proposed

6th Normal Form is not standardized, yet however, it is being discussed by database experts for some time. Hopefully, we would have a clear & standardized definition for 6th Normal Form in the near future…

Summary

- Normalization helps produce database systems that are cost-effective and have better security models.
- Functional dependencies are a very important component of the normalize data process
- Most database systems are normalized database up to the third normal forms.
- A primary key uniquely identifies are record in a Table and cannot be null
- A foreign key helps connect table and references a primary key

SQL Data Types

The data type of a column defines what value the column can hold: integer, character, money, date and time, binary, and so on

Each column in a database table is required to have a name and a data type

- Number Datatypes
- Text Datatypes
- Date Datatypes
- String Datatypes
- Other Datatypes
- Microsoft Access Data types

In MySQL there are three main data types: text, number, string and date….

Number Data type	Description
TINYINT(size)	-128 to 127 normal. 0 to 255 UNSIGNED*. The maximum number of digits may be specified in parenthesis
SMALLINT(size)	-32768 to 32767 normal. 0 to 65535 UNSIGNED*. The maximum number of digits may be specified in parenthesis
MEDIUMINT(size)	-8388608 to 8388607 normal. 0 to 16777215 UNSIGNED*. The maximum number of digits may be specified in parenthesis
INT(size)	-2147483648 to 2147483647 normal. 0 to 4294967295 UNSIGNED*. The maximum number of digits may be specified in parenthesis
BIGINT(size)	-9223372036854775808 to 9223372036854775807 normal. 0 to 18446744073709551615 UNSIGNED*. The maximum number of digits may be specified in parenthesis
FLOAT(size,d)	A small number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DOUBLE(size,d)	A large number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DECIMAL(size,d)	A DOUBLE stored as a string , allowing for a fixed decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter

Text Data type	Description
CHAR(size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis. Can store up to 255 characters
VARCHAR(size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis. Can store up to 255 characters. Note: If you put a greater value than 255 it will be converted to a TEXT type
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT	Holds a string with a maximum length of 65,535 characters
BLOB	For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large Objects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large Objects). Holds up to 4,294,967,295 bytes of data
ENUM(x,y,z,etc.)	Let you enter a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. Note: The values are sorted in the order you enter them. You enter the possible values in this format: ENUM('X','Y','Z')
SET	Similar to ENUM except that SET may contain up to 64 list items and can store more than one choice

String Data type	Description	Max size	Storage
char(n)	Fixed width character string	8,000 characters	Defined width
varchar(n)	Variable width character string	8,000 characters	2 bytes + number of chars
varchar(max)	Variable width character string	1,073,741,824 characters	2 bytes + number of chars
text	Variable width character string	2GB of text data	4 bytes + number of chars
nchar	Fixed width Unicode string	4,000 characters	Defined width x 2
nvarchar	Variable width Unicode string	4,000 characters	
nvarchar(max)	Variable width Unicode string	536,870,912 characters	
ntext	Variable width Unicode string	2GB of text data	
binary(n)	Fixed width binary string	8,000 bytes	
varbinary	Variable width binary string	8,000 bytes	
varbinary(max)	Variable width binary string	2GB	
image	Variable width binary string	2GB	

Date Data type	Description
DATE()	A date. Format: YYYY-MM-DD Note: The supported range is from '1000-01-01' to '9999-12-31'
DATETIME()	*A date and time combination. Format: YYYY-MM-DD HH:MI:SS Note: The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'
TIMESTAMP()	*A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD HH:MI:SS Note: The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC
TIME()	A time. Format: HH:MI:SS Note: The supported range is from '-838:59:59' to '838:59:59'
YEAR()	A year in two-digit or four-digit format. Note: Values allowed in four-digit format: 1901 to 2155. Values allowed in two-digit format: 70 to 69, representing years from 1970 to 2069

SQL Operators

- Arithmetic Operators
- Comparison Operators
- Logical Operators
- Bitwise Operators
- Compound Operators

Arithmetic Operators:

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

Comparison Operators:

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

Logical Operators:

Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition

Bitwise Operators:

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR

Compound Operators:

Operator	Description
+=	Add equals
-=	Subtract equals
*=	Multiply equals
/=	Divide equals
%=	Modulo equals
&=	Bitwise AND equals
^-=	Bitwise exclusive equals
|*=	Bitwise OR equals

SQL Constraints

SQL constraints are used to specify rules for the data in a table.

SQL Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside table.

Constraints can be divided into the following two types,

1. **Column level constraints:** Limits only column data.
2. **Table level constraints:** Limits whole table data

The following constraints are commonly used in SQL:

- **NOT NULL** - Ensures that a column cannot have a NULL value
- **UNIQUE** - Ensures that all values in a column are different
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- **FOREIGN KEY** - Uniquely identifies a row/record in another table
- **CHECK** - Ensures that all values in a column satisfies a specific condition
- **DEFAULT** - Sets a default value for a column when no value is specified
- **INDEX** - Used to create and retrieve data from the database very quickly

SQL NOT NULL Constraint:

By default, a column can hold NULL values.
The NOT NULL constraint enforces a column to NOT accept NULL values.

SQL NOT NULL on CREATE TABLE

The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:

Example:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
);
```

SQL UNIQUE Constraint

The UNIQUE constraint ensures that all values in a column are different

A PRIMARY KEY constraint automatically has a UNIQUE constraint

However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

SQL UNIQUE Constraint on CREATE TABLE

The following SQL creates a UNIQUE constraint on the "ID" column when the "Persons" table is created:

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL UNIQUE,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

SQL PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only one primary key, which may consist of single or multiple fields.

SQL PRIMARY KEY on CREATE TABLE

The following SQL creates a PRIMARY KEY on the "ID" column when the "Persons" table is created:

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

SQL FOREIGN KEY Constraint

A FOREIGN KEY is a key used to link two tables together.

A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.

The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

SQL FOREIGN KEY on CREATE TABLE

SQL Server / Oracle / MS Access:

```
CREATE TABLE Orders (
    OrderID int NOT NULL PRIMARY KEY,
    OrderNumber int NOT NULL,
    PersonID int FOREIGN KEY REFERENCES Persons(PersonID)
);
```

SQL CHECK Constraint

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a single column it allows only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

SQL CHECK on CREATE TABLE

MySQL / SQL Server / Oracle / MS Access:

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255),
    CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes')
);
```

SQL DEFAULT Constraint

The DEFAULT constraint is used to provide a default value for a column.

The default value will be added to all new records IF no other value is specified.

SQL DEFAULT on CREATE TABLE

My SQL / SQL Server / Oracle / MS Access:

The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes’
);
```

SQL DEFAULT on ALTER TABLE

```
ALTER TABLE Persons
ADD CONSTRAINT df_City
DEFAULT 'Sandnes' FOR City;
```

DDL(Data Definition Language) : DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in database

Examples of DDL commands:

- CREATE – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
- DROP – is used to delete objects from the database.
- ALTER-is used to alter the structure of the database.
- TRUNCATE–is used to remove all records from a table, including all spaces allocated for the records are removed.

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,
   ....
);
```

Example:

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

SQL DROP TABLE Statement

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

Syntax

DROP TABLE *table_name*;

Example:

DROP TABLE Shippers;

SQL ALTER TABLE Statement

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

ALTER TABLE - ADD Column

ALTER TABLE *table_name*
ADD *column_name datatype*;

ALTER TABLE - DROP COLUMN

ALTER TABLE *table_name*
DROP COLUMN *column_name*;

ALTER TABLE – MODIFY COLUMN

ALTER TABLE *table_name*
MODIFY COLUMN *column_name datatype*;

SQL TRUNCATE TABLE

The Truncate Table commands deletes the data inside a table, but not the table itself

The following SQL truncates the table "Categories“

Example:

```
TRUNCATE TABLE Categories;
```

DML(Data Manipulation Language) :

The SQL commands that deals with the manipulation of data present in database belong to DML

Examples of DML:

- **SELECT** – is used to retrieve data from the a database.
- **INSERT** – is used to insert data into a table.
- **UPDATE** – is used to update existing data within a table.
- **DELETE** – is used to delete records from a database table.

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*;

Examples:

SELECT CustomerName, City FROM Customers;

SELECT * FROM Customers;

SQL INSERT INTO Statement

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

INSERT INTO Example

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

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


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

SQL UPDATE Statement

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

Examples:

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt’
WHERE CustomerID = 1;
```

```
UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';
```

SQL DELETE Statement

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

DELETE Syntax

DELTE FROM *table_name* WHERE *condition*;

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:

Example:

DELETE FROM Customers;

Important 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;
```

Examples:

```
SELECT * FROM Customers;
```

```
SELECT CustomerName, City FROM Customers;
```

SQL SELECT DISTINCT Statement

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

Examples:

```
SELECT DISTINCT Country FROM Customers;
```

```
SELECT COUNT(DISTINCT Country) FROM Customers;
```

```
SELECT Count(*) AS DistinctCountries
FROM (SELECT DISTINCT Country FROM Customers);
```

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.

Examples:

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

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

The following operators can be used in the WHERE clause:

Operator	Description
=	Equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

SQL AND, OR and NOT Operators

The WHERE clause can be combined with AND, OR, and NOT operators.
The AND and OR operators are used to filter records based on more than one condition:
The NOT operator displays a record if the condition(s) is NOT TRUE.

And Example:

SELECT * FROM Customers WHERE Country='Germany' AND City='Berlin’;

OR Example:

SELECT * FROM Customers WHERE City='Berlin' OR City='München’;

NOT Example

SELECT * FROM Customers WHERE NOT Country='Germany';

Combining AND, OR and NOT

Examples:

```
SELECT * FROM Customers
WHERE Country='Germany' AND (City='Berlin' OR City='München’);
```

```
SELECT * FROM Customers
WHERE NOT Country='Germany' AND NOT Country='USA';
```

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

Examples:

```
SELECT * FROM Customers ORDER BY Country;

SELECT * FROM Customers ORDER BY Country DESC;

SELECT * FROM Customers ORDER BY Country, CustomerName;

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

SQL GROUP BY Statement

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:

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

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

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

SQL HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

Examples:

```
SELECT COUNT(CustomerID), Country FROM Customers
GROUP BY Country HAVING COUNT(CustomerID) > 5;
```

```
SELECT COUNT(CustomerID), Country FROM Customers
GROUP BY Country HAVING COUNT(CustomerID) > 5
ORDER BY COUNT(CustomerID) DESC;
```

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM (Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID)
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 10;
```

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
WHERE LastName = 'Davolio' OR LastName = 'Fuller'
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 25;
```

The SQL MIN() and MAX() Functions

The MIN() function returns the smallest value of the selected column.
The MAX() function returns the largest value of the selected column.

Examples:

```
SELECT MIN(Price) AS SmallestPrice FROM Products;
```

```
SELECT MAX(Price) AS LargestPrice FROM Products;
```

SQL COUNT(), AVG() and SUM() Functions

The COUNT() function returns the number of rows that matches a specified criteria.
The AVG() function returns the average value of a numeric column.
The SUM() function returns the total sum of a numeric column.

Examples:

```
SELECT COUNT(ProductID) FROM Products;

SELECT AVG(Price) FROM Products;

SELECT SUM(Quantity) FROM OrderDetails;
```

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

```
SELECT * FROM Customers WHERE CustomerName LIKE 'a%';
SELECT * FROM Customers WHERE CustomerName LIKE '%a’;
SELECT * FROM Customers WHERE CustomerName LIKE '%or%’;
SELECT * FROM Customers WHERE CustomerName LIKE '_r%';
SELECT * FROM Customers WHERE CustomerName LIKE 'a_%_%’
SELECT * FROM Customers WHERE ContactName LIKE 'a%o’;
SELECT * FROM Customers WHERE CustomerName NOT LIKE 'a%';
```

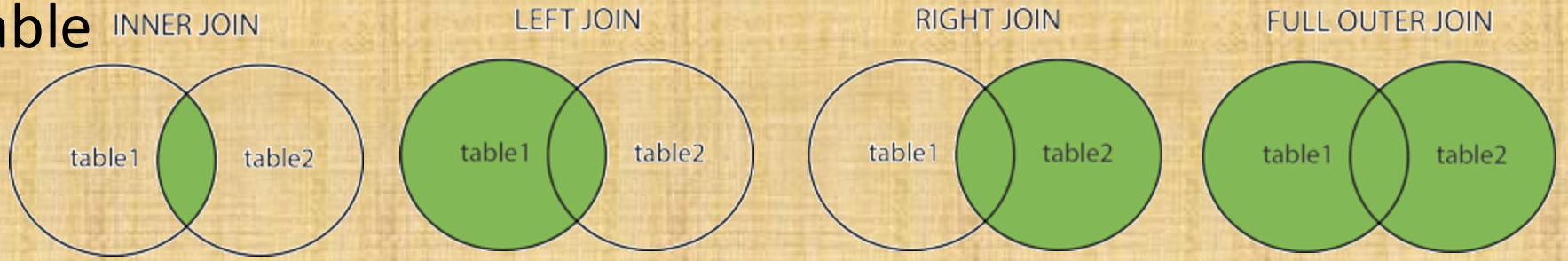
SQL JOIN

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

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 JOIN**: Return all records from the left table, and the matched records from the right table
- **RIGHT JOIN**: Return all records from the right table, and the matched records from the left table
- **FULL JOIN**: Return 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.

Examples:

```
SELECT Orders.OrderID, Customers.CustomerName FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

JOIN Three Tables

Examples:

```
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.

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.

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 return all records when there is a match in either left (table1) or right (table2) table records.

Example:

SELECT *column_name(s)*
FROM *table1*
FULL OUTER JOIN *table2*
ON *table1.column_name* = *table2.column_name*;

SQL Self JOIN

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

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;
```