Structured Query Language

A standard language for storing, manipulating and retrieving data in databases.

Dr. Tarkeshwar Barua

Introduction

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987
- There are different versions of the SQL language.
- All SQL support at least the major commands (such as SELECT, UPDATE, DELETE, INSERT, WHERE) in a similar manner.
- Most of the SQL database programs also have their own proprietary extensions in addition to the SQL standard!

What SQL can do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

What is RDBMS?

- RDBMS is the basis for SQL, and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
- The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows.
- Every table is broken up into smaller entities called fields. The fields in the Customers table consist of **CustomerID, CustomerName, etc**. A field is a column in a table that is designed to maintain specific information about every record in the table.
- A record, also called a row, is each individual entry that exists in a table. A record is a horizontal entity in a table.
- A column is a vertical entity in a table that contains all information associated with a specific field in a table.

What is Table?

- A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.
- sudo -u postgres psql
- sudo mysql -u root -p

SQL Statements

- Most of the actions you need to perform on a database are done with SQL statements.
- SQL keywords are NOT case sensitive: select is the same as SELECT
- Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.
- The SQL statement selects all the records in the "Customers" table:

```
SELECT * FROM Customers;
```

Sample Database

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						

Important SQL Commands

- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- INSERT INTO - inserts new data into a database
- CREATE DATABASE - creates a new database
- ALTER DATABASE - modifies a database
- CREATE TABLE - creates a new table
- ALTER TABLE - modifies a table
- DROP TABLE - deletes a table
- CREATE INDEX - creates an index (search key)
- DROP INDEX - deletes an index

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 column1, column2, … FROM table_name;

- Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

SELECT * FROM table_name;

SELECT * FROM Customers;

SELECT Column Example

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

SELECT CustomerName, City FROM Customers;

SELECT DISTINCT Statement

- The SELECT DISTINCT statement is used to return only distinct (different) values.
- Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

```
SELECT DISTINCT column1, column2, … FROM table_name;

SELECT Country FROM Customers;

SELECT DISTINCT Country FROM Customers;

SELECT COUNT(DISTINCT Country) FROM Customers;

/*Inner Query*/

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

The SQL WHERE Clause

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

```
SELECT column1, column2, … FROM table_name WHERE condition;
```

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

- SQL requires single quotes around text values (most database systems will also allow double quotes).
- However, numeric fields should not be enclosed in quotes:

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

Operators in The WHERE Clause

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

The 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 AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.
- The NOT operator displays a record if the condition(s) is NOT TRUE.
- SELECT column1, column2, …FROM table_name WHERE condition1 AND condition2 AND condition3 ...;
- SELECT column1, column2, …FROM table_name WHERE condition1 OR condition2 OR condition3 ...;
- SELECT column1, column2, ...FROM table_name WHERE NOT condition;
- SELECT * FROM Customers WHERE Country='Germany' AND City='Berlin';
- SELECT * FROM Customers WHERE City='Berlin' OR City='München';
- SELECT * FROM Customers WHERE Country='Germany' OR Country='Spain';

The SQL AND, OR and NOT Operators

- SELECT * FROM Customers WHERE NOT Country='Germany';
- SELECT * FROM Customers WHERE Country='Germany' AND (City='Berlin' OR City='München');
- SELECT * FROM Customers WHERE NOT Country='Germany' AND NOT Country='USA';

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

INSERT INTO Statement

- The INSERT INTO statement is used to insert new records in a table.
- It is possible to write the INSERT INTO statement in two ways:
  - 1. Specify both the column names and the values to be inserted:
    - INSERT INTO table_name (column1, column2, column3, …) VALUES (value1, value2, value3, ...);
  - 2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table.
    - INSERT INTO table_name VALUES (value1, value2, value3, ...);

INSERT INTO Statement

- INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country) VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
- The CustomerID column is an auto-increment field and will be generated automatically when a new record is inserted into the table.
- 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):
- INSERT INTO Customers (CustomerName, City, Country) VALUES ('Cardinal', 'Stavanger', 'Norway');

NULL Values

- A field with a NULL value is a field with no value.
- If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.
- A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!
- It is not possible to test for NULL values with comparison operators, such as =, <, or <>.
- We will have to use the IS NULL and IS NOT NULL operators instead.

NULL Values

- SELECT column_names FROM table_name WHERE column_name IS NULL;
- SELECT column_names FROM table_name WHERE column_name IS NOT NULL;
- The IS NULL operator is used to test for empty values (NULL values).
- The following SQL lists all customers with a NULL value in the "Address" field:
- SELECT CustomerName, ContactName, Address FROM Customers WHERE Address IS NULL;

NULL Values

- The IS NOT NULL operator is used to test for non-empty values (NOT NULL values).
- SELECT CustomerName, ContactName, Address FROM Customers WHERE Address IS NOT NULL;

UPDATE Statement

- The UPDATE statement is used to modify the existing records in a table.
- UPDATE table_name SET column1 = value1, column2 = value2, ...WHERE condition;
- Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!
- UPDATE Customers SET ContactName = 'Alfred Schmidt', City= 'Frankfurt' WHERE CustomerID = 1;
- It is the WHERE clause that determines how many records will be updated.
- UPDATE Customers SET ContactName='Juan' WHERE Country='Mexico';
- UPDATE Customers SET ContactName='Juan';

DELETE Statement

- The DELETE statement is used to delete existing records in a table.
- DELETE FROM table_name WHERE condition;
- Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!
- DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
- 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;
- DELETE FROM Customers;

TOP, LIMIT, FETCH FIRST or ROWNUM Clause

- The SELECT TOP clause is used to specify the number of records to return.
- The SELECT TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.
- Not all database systems support the SELECT TOP clause. MySQL supports the LIMIT clause to select a limited number of records, while Oracle uses FETCH FIRST n ROWS ONLY and ROWNUM.

  SQL Server / MS Access Syntax:
- SELECT TOP number|percent column_name(s) FROM table_name WHERE condition;

TOP, LIMIT, FETCH FIRST or ROWNUM Clause

- MySQL Syntax:
- SELECT column_name(s) FROM table_name WHERE condition LIMIT number;
- Oracle 12 Syntax:
- SELECT column_name(s) FROM table_name ORDER BY column_name(s) FETCH FIRST number ROWS ONLY;
- Older Oracle Syntax:
- SELECT column_name(s) FROM table_name WHERE ROWNUM <= number;
- Older Oracle Syntax (with ORDER BY):
- SELECT * FROM (SELECT column_name(s) FROM table_name ORDER BY column_name(s)) WHERE ROWNUM <= number;
- SELECT 1+1;
- SELECT PI()*2;
- select version();

TOP, LIMIT, FETCH FIRST or ROWNUM Clause

- SELECT TOP 3 * FROM Customers;
- SELECT * FROM Customers LIMIT 3;
- SELECT * FROM Customers FETCH FIRST 3 ROWS ONLY;
- SELECT TOP 50 PERCENT * FROM Customers;
- SELECT * FROM Customers FETCH FIRST 50 PERCENT ROWS ONLY;
- SELECT TOP 3 * FROM Customers WHERE Country='Germany';
- SELECT * FROM Customers WHERE Country='Germany' LIMIT 3;
- SELECT * FROM Customers WHERE Country='Germany' FETCH FIRST 3 ROWS ONLY;

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.
- SELECT MIN(column_name) FROM table_name WHERE condition;
- SELECT MAX(column_name) FROM table_name WHERE condition;
- SELECT MIN(Price) AS SmallestPrice FROM Products;
- SELECT MAX(Price) AS LargestPrice FROM Products;

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 sign (_) represents one, single character
- MS Access uses an asterisk (*) instead of the percent sign (%), and a question mark (?) instead of the underscore (_).
- The percent sign and the underscore can also be used in combinations!
- SELECT column1, column2, … FROM table_name WHERE columnN LIKE pattern;

LIKE Operator

- You can also combine any number of conditions using AND or OR operators.
- Here are some examples showing different LIKE operators with '%' and '_' wildcards:

WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

LIKE Operator

- 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%';

Wildcard Characters

- A wildcard character is used to substitute one or more characters in a string.
- Wildcard characters are used with the LIKE operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

***   Represents zero or more characters   bl* finds bl, black, blue, and blob**

? Represents a single character h?t finds hot, hat, and hit

[] Represents any single character within the brackets   h[oa]t finds hot and hat, but not hit

!   Represents any character not in the brackets   h[!oa]t finds hit, but not hot and hat

-   Represents any single character within the specified range   c[a-b]t finds cat and cbt

# Represents any single numeric character 2#5 finds 205, 215, 225, 235, 245, 255, 265, 275, 285, and 295

Wildcard Characters

% Represents zero or more characters bl% finds bl, black, blue, and blob

_ Represents a single character h_t finds hot, hat, and hit

[] Represents any single character within the brackets h[oa]t finds hot and hat, but not hit

^ Represents any character not in the brackets h[^oa]t finds hit, but not hot and hat

- Represents any single character within the specified range c[a-b]t finds cat and cbt

Wildcard Characters

WHERE CustomerName LIKE 'a%'   Finds any values that starts with "a"

WHERE CustomerName LIKE '%a'   Finds any values that ends with "a"

WHERE CustomerName LIKE '%or%'   Finds any values that have "or" in any position

WHERE CustomerName LIKE '_r%'   Finds any values that have "r" in the second position

WHERE CustomerName LIKE 'a__%'   Finds any values that starts with "a" and are at least 3 characters in length

WHERE ContactName LIKE 'a%o' Finds any values that starts with "a" and ends with "o"

Wildcard Characters

- SELECT * FROM Customers WHERE City LIKE 'ber%';
- SELECT * FROM Customers WHERE City LIKE '%es%';
- SELECT * FROM Customers WHERE City LIKE '_ondon';
- SELECT * FROM Customers WHERE City LIKE 'L_n_on';
- SELECT * FROM Customers WHERE City LIKE '[bsp]%';
- SELECT * FROM Customers WHERE City LIKE '[a-c]%';
- SELECT * FROM Customers WHERE City LIKE '[!bsp]%';
- SELECT * FROM Customers WHERE City NOT LIKE '[bsp]%';

IN Operator

- The IN operator allows you to specify multiple values in a WHERE clause.
- The IN operator is a shorthand for multiple OR conditions.
- SELECT column_name(s) FROM table_name WHERE column_name IN (value1, value2, ...);
- SELECT column_name(s) FROM table_name WHERE column_name IN (SELECT STATEMENT);
- SELECT * FROM Customers WHERE Country IN ('Germany', 'France', 'UK');
- SELECT * FROM Customers WHERE Country NOT IN ('Germany', 'France', 'UK');
- SELECT * FROM Customers WHERE Country IN (SELECT Country FROM Suppliers);

SQL BETWEEN Operator

- The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.
- The BETWEEN operator is inclusive: begin and end values are included.
- SELECT column_name(s) FROM table_name WHERE column_name BETWEEN value1 AND value2;
- SELECT * FROM Products WHERE Price BETWEEN 10 AND 20;
- SELECT * FROM Products WHERE Price NOT BETWEEN 10 AND 20;
- SELECT * FROM Products WHERE Price BETWEEN 10 AND 20 AND CategoryID NOT IN (1,2,3);
- SELECT * FROM Products WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni' ORDER BY ProductName;

SQL BETWEEN Operator

- SELECT * FROM Products WHERE ProductName BETWEEN "Carnarvon Tigers" AND "Chef Anton's Cajun Seasoning" ORDER BY ProductName;
- SELECT * FROM Products WHERE ProductName NOT BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni' ORDER BY ProductName;
- SELECT * FROM Order WHERE OrderDate BETWEEN #07/01/1996# AND #07/31/1996#;
- SELECT * FROM Orders WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31';

SQL Aliases

- SQL aliases are used to give a table, or a column in a table, a temporary name.
- Aliases are often used to make column names more readable.
- An alias only exists for the duration of that query.
- An alias is created with the AS keyword.
- SELECT column_name AS alias_name FROM table_name;
- SELECT column_name(s) FROM table_name AS alias_name;

SQL Aliases

- SELECT CustomerID AS ID, CustomerName AS Customer FROM Customers;
- SELECT CustomerName AS Customer, ContactName AS [Contact Person] FROM Customers;
- SELECT CustomerName, Address + ', ' + PostalCode + ' ' + City + ', ' + Country AS Address FROM Customers;
- SELECT CustomerName, CONCAT(Address,', ',PostalCode,', ',City,', ',Country) AS Address FROM Customers;
- SELECT o.OrderID, o.OrderDate, c.CustomerName FROM Customers AS c, Orders AS o WHERE c.CustomerName='Around the Horn' AND c.CustomerID=o.CustomerID;
- SELECT Orders.OrderID, Orders.OrderDate, Customers.CustomerName FROM Customers, Orders WHERE Customers.CustomerName='Around the Horn' AND Customers.CustomerID=Orders.CustomerID;

Aliases can be useful when

- There are more than one table involved in a query
- Functions are used in the query
- Column names are big or not very readable
- Two or more columns are combined together
- It requires double quotation marks or square brackets if the alias name contains spaces

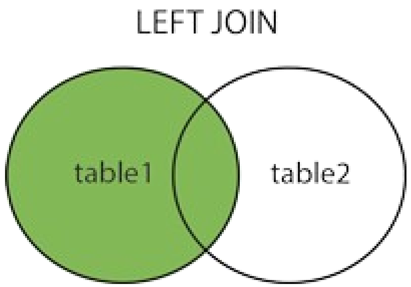
SQL JOIN

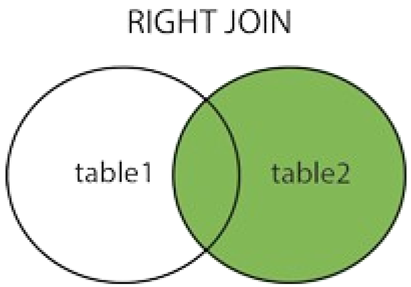
- A JOIN clause is used to combine rows from two or more tables, based on a related column between them.
- Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.
- Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

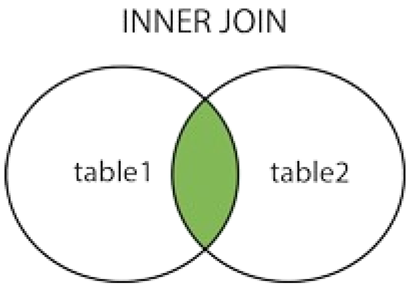
SQL JOIN

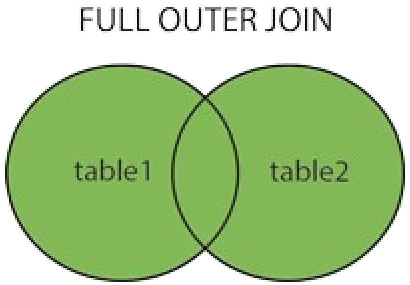
- SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate FROM Orders INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
- Here are the different types of the JOINs in SQL:
- (INNER) JOIN: Returns records that have matching values in both tables
- LEFT (OUTER) JOIN: Returns all records 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 JOIN


LEFT JOIN
table1
table2


RIGHT JOIN
table1
table2


INNER JOIN
table1
table2


FULL OUTER JOIN
table1
table2

INNER JOIN Keyword

- The INNER JOIN keyword selects records that have matching values in both tables.
- SELECT column_name(s) FROM table1 INNER JOIN table2 ON table1.column_name = table2.column_name;
- SELECT Orders.OrderID, Customers.CustomerName FROM Orders INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
- The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns. If there are records in the "Orders" table that do not have matches in "Customers", these orders will not be shown!

INNER JOIN Keyword

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);

LEFT JOIN Keyword

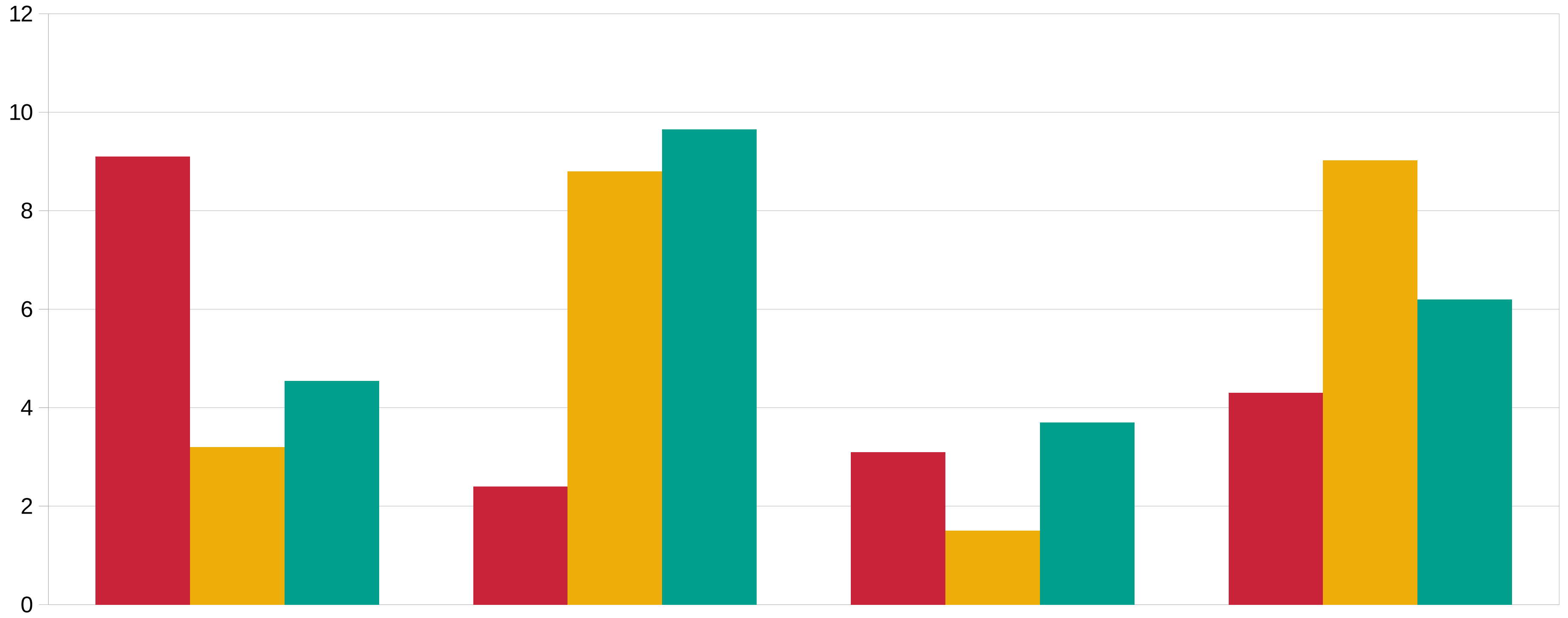
- The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.
- SELECT column_name(s) FROM table1 LEFT JOIN table2 ON table1.column_name = table2.column_name;
- In some databases LEFT JOIN is called LEFT OUTER JOIN.
- SELECT Customers.CustomerName, Orders.OrderID FROM Customers LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID ORDER BY Customers.CustomerName;

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.
- FULL OUTER JOIN and FULL JOIN are the same.
- SELECT column_name(s) FROM table1 FULL OUTER JOIN table2 ON table1.column_name = table2.column_name WHERE condition;
- FULL OUTER JOIN can potentially return very large result-sets!
- SELECT Customers.CustomerName, Orders.OrderID FROM Customers FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID ORDER BY Customers.CustomerName;
- The FULL OUTER JOIN keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

SQL Self Join

- A self join is a regular join, but the table is joined with itself.
- SELECT column_name(s) FROM table1 T1, table1 T2 WHERE condition;
- T1 and T2 are different table aliases for the same table.
- 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;


12
10
8
6
4
2
0