# Database

## SQL

### ❖ What is SQL?

- 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
- SQL is a standard language for storing, manipulating and retrieving data in databases.

### ❖ RDBMS

- RDBMS stands for Relational Database Management System.
- 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.

### ❖ Database Tables

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.

### ❖ 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;
```

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;
```

## ❖ The SQL SELECT DISTINCT Statement

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

- SELECT DISTINCT Syntax

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

## ❖ 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.

- WHERE Syntax

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

```
e.g

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

**Note**: The WHERE clause is not only used in SELECT statements, it is also used in UPDATE, DELETE, etc.!

## ❖ 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.

- AND/OR Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND/OR condition2 AND condition3 ...;
```

```
e.g : WHERE Country='Germany' AND City='Berlin';

      WHERE City='Berlin' OR City='München';
```

- NOT Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

```
e.g WHERE NOT Country='Germany';

WHERE Country='Germany' AND (City='Berlin' OR City='München');

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.

- ORDER BY Syntax

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

```
e.g

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:

```
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:

e.g.

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

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

## ❖ 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:

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. Here, the INSERT INTO syntax would be as follows:

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

e.g.

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

## ❖ SQL NULL Values

- What is a NULL Value?

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.

- How to Test for NULL Values?

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.

- IS NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL/ IS NOT NULL;
```

```
e.g
```

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NULL/IS NOT NULL;
```

## ❖ 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;
```

```
e.g
```

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

## ❖ The SQL DELETE Statement

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

- DELETE Syntax

```
DELETE FROM table_name WHERE condition;
```

```
e.g
```

```
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; e.g. DELETE FROM Customers;
```

## ❖ The SQL SELECT TOP 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.

- SQL Server / MS Access Syntax:

```
SELECT TOP number|percent column_name(s)
FROM table_name
WHERE condition;
```

```
e.g.
```

```
SELECT TOP 3 * FROM Customers;
```

```
SELECT TOP 50 PERCENT * FROM Customers;
```

## ❖ 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.

- MIN()/ MAX() Syntax

```
SELECT MIN/MAX(column_name)
FROM table_name
WHERE condition;
```

```
e.g

SELECT MIN(Price) AS SmallestPrice
FROM Products;


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

## ❖ The SQL COUNT(), AVG() and SUM() Functions

The COUNT() function returns the number of rows that matches a specified criterion.

- COUNT() Syntax

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

```
e.g.

SELECT COUNT(ProductID)
FROM Products;

o/p:

COUNT(ProductID) : 77
```

The AVG() function returns the average value of a numeric column.

- AVG() Syntax

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

```
e.g.
```

```
SELECT AVG(Price)
FROM Products;
```

```
o/P
```

**AVG(Price) :** 28.866363636363637

The SUM() function returns the total sum of a numeric column.

- SUM() Syntax

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

```
e.g.
```

```
SELECT SUM(Quantity)

FROM OrderDetails;
```

O/P: **SUM(Quantity):** 12743

## ❖ 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 sign (_) represents one, single character

**Note:** 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!

- LIKE Syntax

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

e.g

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

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

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

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

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

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

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

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

## ❖ Wildcard Characters in SQL Server

| Symbol | Description | Example |
|---|---|---|
| % | 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 |

## ❖ The SQL 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.

- IN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

or:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

```
e.g.

SELECT * FROM Customers
WHERE Country IN/NOT IN ('Germany', 'France', 'UK');
```

```
SELECT * FROM Customers
WHERE Country IN (SELECT Country FROM Suppliers);
```

## ❖ The 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.

- BETWEEN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;


e.g.

SELECT * FROM Products
WHERE Price BETWEEN / 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 'Tigers' AND 'Mozzarella di Giovanni'
ORDER BY ProductName;
```

## ❖ 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.

- Alias Column Syntax

```
SELECT column_name AS alias_name
FROM table_name;
```

- Alias Table Syntax

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

```
e.g.
```

```
SELECT CustomerID AS ID, CustomerName AS Customer
FROM Customers;
```

```
SELECT CustomerName, 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;
```

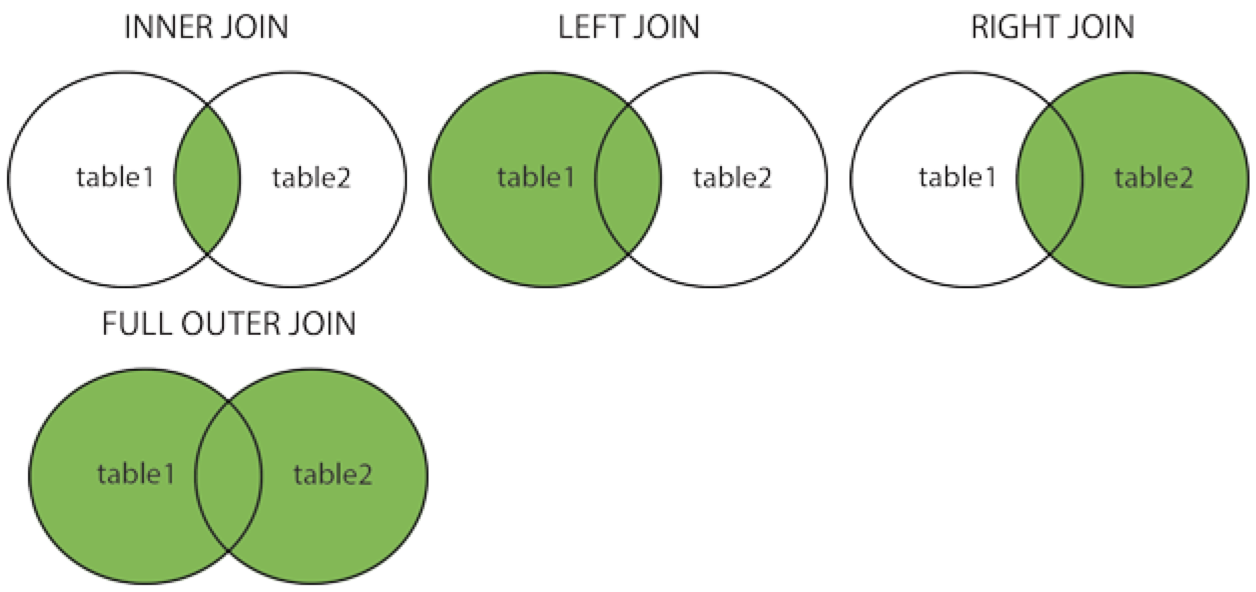
## ❖ SQL JOIN

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 (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

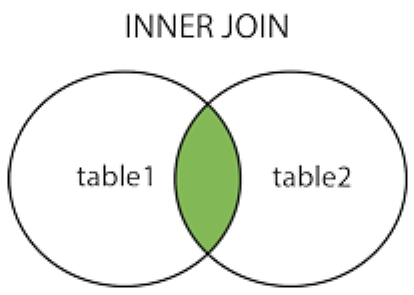


## ❖ SQL INNER JOIN Keyword

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;
```



e.g

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

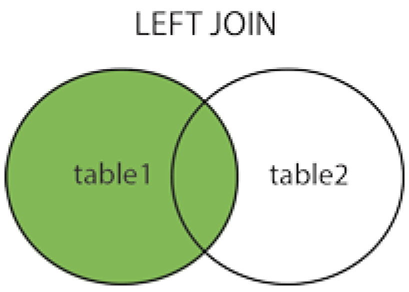
## ❖ SQL LEFT JOIN Keyword

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.

- LEFT JOIN Syntax

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

**Note:** In some databases LEFT JOIN is called LEFT OUTER JOIN.



e.g.

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

## ❖ SQL RIGHT JOIN Keyword

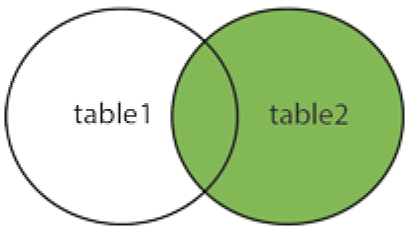
The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

- RIGHT JOIN Syntax

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

**Note:** In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

RIGHT JOIN



e.g

```
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

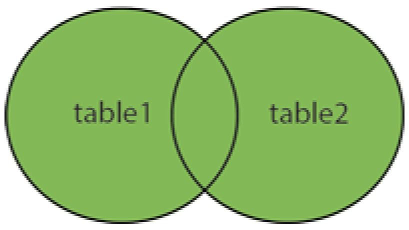
The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

**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;
```

FULL OUTER JOIN



e.g

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

## ❖ The SQL UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order
- The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL

  - UNION/ UNION_ALL Syntax

```
SELECT column_name(s) FROM table1
UNION / UNION_ALL
SELECT column_name(s) FROM table2;
```

e.g

The following SQL statement returns the cities (only distinct values) from both the "Customers" and the "Suppliers" table:

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```

The following SQL statement returns the cities (duplicate values also) from both the "Customers" and the "Suppliers" table:

```
SELECT City FROM Customers
UNION ALL
SELECT City FROM Suppliers
ORDER BY City;
```

## ❖ The SQL GROUP BY Statement

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

- GROUP BY Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

*e.g*

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

```
SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country

ORDER BY COUNT(CustomerID) DESC;
```

## ❖ The SQL HAVING Clause

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

- HAVING Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

*e.g*

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

## ❖ The SQL EXISTS Operator

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns TRUE if the subquery returns one or more records.

- EXISTS Syntax

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

e.g

```
SELECT SupplierName
FROM Suppliers
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID =
Suppliers.supplierID AND Price < 20);
```

## ❖ The SQL ANY Operator

The ANY operator:

- returns a boolean value as a result
- returns TRUE if ANY of the subquery values meet the condition

ANY means that the condition will be true if the operation is true for any of the values in the range.

- ANY Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

## ❖ The SQL ALL Operator

The ALL operator:

- returns a boolean value as a result
- returns TRUE if ALL of the subquery values meet the condition
- is used with SELECT, WHERE and HAVING statements

ALL means that the condition will be true only if the operation is true for all values in the range.

- ALL Syntax With SELECT

```
SELECT ALL column_name(s)
FROM table_name
WHERE condition;
```

e.g

The following SQL statement lists the ProductName if it finds ANY records in the OrderDetails table has Quantity equal to 10 (this will return TRUE because the Quantity column has some values of 10):

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity = 10);
```

The following SQL statement lists the ProductName if ALL the records in the OrderDetails table has Quantity equal to 10. This will of course return FALSE because the Quantity column has many different values (not only the value of 10):

```
SELECT ProductName
FROM Products
WHERE ProductID = ALL
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity = 10);
```

## ❖ The SQL SELECT INTO Statement

The SELECT INTO statement copies data from one table into a new table.

- SELECT INTO Syntax

Copy all columns into a new table:

```
SELECT *
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

e.g.

The following SQL statement copies only the German customers into a new table:

```
SELECT * INTO CustomersGermany
FROM Customers
WHERE Country = 'Germany';
```

## ❖ The SQL INSERT INTO SELECT Statement

The INSERT INTO SELECT statement copies data from one table and inserts it into another table.

The INSERT INTO SELECT statement requires that the data types in source and target tables matches.

**Note:** The existing records in the target table are unaffected.

- INSERT INTO SELECT Syntax

Copy all columns from one table to another table:

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

Copy only some columns from one table into another table:

```
INSERT INTO table2 (column1, column2, column3, ...)
SELECT column1, column2, column3, ...
FROM table1
WHERE condition;
```

```
e.g
```

The following SQL statement copies "Suppliers" into "Customers" (the columns that are not filled with data, will contain NULL):

❖ Example

```
INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country FROM Suppliers;
```

## SQL Interview Questions

**1. What is Database?**

A database is an organized collection of data, stored and retrieved digitally from a remote or local computer system. Databases can be vast and complex, and such databases are developed using fixed design and modeling approaches.

**2. What is DBMS?**

DBMS stands for Database Management System. DBMS is a system software responsible for the creation, retrieval, updation, and management of the database. It ensures that our data is consistent, organized, and is easily accessible by serving as an interface between the database and its end-users or application software.

**3. What is RDBMS? How is it different from DBMS?**

RDBMS stands for Relational Database Management System. The key difference here, compared to DBMS, is that RDBMS stores data in the form of a collection of tables, and relations can be defined between the common fields of these tables. Most modern database management systems like MySQL, Microsoft SQL Server, Oracle, IBM DB2, and Amazon Redshift are based on RDBMS.

**4. What is SQL?**

SQL stands for Structured Query Language. It is the standard language for relational database management systems. It is especially useful in handling organized data comprised of entities (variables) and relations between different entities of the data.

**5. What is the difference between SQL and MySQL?**

SQL is a standard language for retrieving and manipulating structured databases. On the contrary, MySQL is a relational database management system, like SQL Server, Oracle or IBM DB2, that is used to manage SQL databases.

**6. What are Tables and Fields?**

A table is an organized collection of data stored in the form of rows and columns. Columns can be categorized as vertical and rows as horizontal. The columns in a table are called fields while the rows can be referred to as records.

**7. What are Constraints in SQL?**

Constraints are used to specify the rules concerning data in the table. It can be applied for single or multiple fields in an SQL table during the creation of the table or after creating using the ALTER TABLE command. The constraints are:

- NOT NULL - Restricts NULL value from being inserted into a column.
- CHECK - Verifies that all values in a field satisfy a condition.
- DEFAULT - Automatically assigns a default value if no value has been specified for the field.
- UNIQUE - Ensures unique values to be inserted into the field.
- INDEX - Indexes a field providing faster retrieval of records.
- PRIMARY KEY - Uniquely identifies each record in a table.
- FOREIGN KEY - Ensures referential integrity for a record in another table.

**8. What is a Primary Key?**

The PRIMARY KEY constraint uniquely identifies each row in a table. It must contain UNIQUE values and has an implicit NOT NULL constraint.
A table in SQL is strictly restricted to have one and only one primary key, which is comprised of single or multiple fields (columns).

SQL Interview Questions CHEAT SHEET (2021) - InterviewBit