**SQL**

* **Introduction to SQL**

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

* DBMS

What is Database ?

Database Management Systems (DBMS) are software systems used to store, retrieve, and run queries on data. A DBMS serves as an interface between an end-user and a database, allowing users to create, read, update, and delete data in the database.

DBMS manage the data, the database engine, and the database schema, allowing for data to be manipulated or extracted by users and other programs. This helps provide data security, data integrity, concurrency, and uniform data administration procedures.

* **RDBMS**
* **What is a Relational Database?**
* A *relational database* is a type of database. It uses a structure that allows us to identify and access data *in relation* to another piece of data in the database. Often, data in a relational database is organized into tables.
* **SQL Data type**

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

## **Data Type Categories:**

| Category | Data Types |
| --- | --- |
| Exact numerics | bit, tinyint, smallint, int, bigint, decimal, numeric, money, smallmoney |
| Approximate numerics | Real, Float |
| Date & Time | date, smalldatetime, datetime, datetime2, datetimeoffset, time |
| Character strings | char, varchar, text |
| Unicode Character strings | nchar, nvarchar, ntext |
| Other data types | cursor, hierarchyid, sql\_variant, spatial Geometry types, spatial Geography types, rowversion, uniqueidentifier, xml, table |

* **SQL Constraints**
* [NOT NULL](https://www.w3schools.com/sql/sql_notnull.asp) - Ensures that a column cannot have a NULL value
* [UNIQUE](https://www.w3schools.com/sql/sql_unique.asp) - Ensures that all values in a column are different
* [PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
* [FOREIGN KEY](https://www.w3schools.com/sql/sql_foreignkey.asp) - Prevents actions that would destroy links between tables
* [CHECK](https://www.w3schools.com/sql/sql_check.asp) - Ensures that the values in a column satisfies a specific condition
* [DEFAULT](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column if no value is specified
* [CREATE INDEX](https://www.w3schools.com/sql/sql_create_index.asp) - Used to create and retrieve data from the database very quickly
* **DDL**
* DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
* All the command of DDL are auto-committed that means it permanently save all the changes in the database.
* CREATE
* ALTER
* DROP
* TRUNCATE
* **DML**
* DML commands are used to modify the database. It is responsible for all form of changes in the database.
* The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.
* INSERT
* UPDATE
* DELETE

### **Transaction Control Language**

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

* COMMIT
* ROLLBACK
* SAVEPOINT

### **Data Query Language**

DQL is used to fetch the data from the database.

It uses only one command:

* SELECT

1. **SELECT:**  It is used to select the attribute based on the condition described by WHERE clause.

* **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.
* **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.
* **SQL Operator**

## **SQL Arithmetic Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + | Add | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_add) |
| - | Subtract | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_subtract) |
| \* | Multiply | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_multiply) |
| / | Divide | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_divide) |
| % | Modulo |  |

## **SQL Comparison Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Equal to | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_equal_to) |
| > | Greater than | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_greater_than) |
| < | Less than | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_less_than) |
| >= | Greater than or equal to | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_greater_than2) |
| <= | Less than or equal to | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_less_than2) |
| <> | Not equal to |  |

## **SQL Compound Operators**

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

## **SQL Logical Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| ALL | TRUE if all of the subquery values meet the condition | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_all&ss=-1) |
| AND | TRUE if all the conditions separated by AND is TRUE | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_and) |
| ANY | TRUE if any of the subquery values meet the condition | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_any&ss=-1) |
| BETWEEN | TRUE if the operand is within the range of comparisons | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_between) |
| EXISTS | TRUE if the subquery returns one or more records | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_exists) |
| IN | TRUE if the operand is equal to one of a list of expressions | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_in) |
| LIKE | TRUE if the operand matches a pattern | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_like) |
| NOT | Displays a record if the condition(s) is NOT TRUE | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_not) |
| OR | TRUE if any of the conditions separated by OR is TRUE | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_or) |
| SOME | TRUE if any of the subquery values meet the condition |  |

* **Aggregate Functions**

An aggregate function in [SQL](https://www.simplilearn.com/tutorials/sql-tutorial/what-is-sql) performs a calculation on multiple values and returns a single value. SQL provides many aggregate functions that include avg, count, sum, min, max, etc. An aggregate function ignores NULL values when it performs the calculation, except for the count function.

* **GROUP BY Clause**
* 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.
* **HAVING Clause**

The SQL HAVING clause is used in combination with the [GROUP BY clause](https://www.techonthenet.com/sql/group_by.php) to restrict the groups of returned rows to only those whose the condition is TRUE.

* **PRIMARY KEY & FOREIGN KEY**
* 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; and in the table, this primary key can consist of single or multiple columns (fields).
* The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.
* A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the [PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) in another table.
* The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.
* **IDENTITY**

The IDENTITY keyword is a property in SQL Server. **When a table column is defined with an identity property, its value will be auto-generated incremental value**. This value is created by the server automatically. Therefore, we can't manually enter a value into an identity column as a user. Hence, if we mark a column as identity, SQL Server will populate it in an auto-increment manner.

* **Seed:** It indicates the starting value of the row loaded into the table. By default, its value is 1.
* **Increment:** It indicates the incremental value, which is added to the identity value of the last loaded row. By default, its value is 1.
* **JOINS**
* **Ansi Format (new style)**
  + **Inner Join**

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

* + - **Outer Join**

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

* + - **Left Join**

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.

* + - **Right Join**
* 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.
  + - **Cross Join**
* **What is a CROSS JOIN in SQL?**
* In SQL, CROSS JOINs are used to combine each row of one table with each row of another table, and return the Cartesian product of the sets of rows from the tables that are joined.
  + - **Natural Join**

Natural join is an [SQL join](https://www.geeksforgeeks.org/sql-join-set-1-inner-left-right-and-full-joins/) operation that creates join on the base of the common columns in the tables. To perform natural join there must be one common attribute(Column) between two tables.

* + - **Non-Ansi Format (old style)**
* **Equi-Join**

EQUI JOIN creates a JOIN for equality or matching column(s) values of the relative tables. EQUI JOIN also create JOIN by using JOIN with ON and then providing the names of the columns with their relative tables to check equality using equal sign (=).

* **Non Equi-Join**

NON EQUI JOIN performs a JOIN using comparison operator other than equal(=) sign like >, <, >=, <= with conditions.

* **Self join**
* A self join is a regular join, but the table is joined with itself.
  + **Index**

Indexes are **used to retrieve data from the database more quickly than otherwise**. The users cannot see the indexes, they are just used to speed up searches/queries.

**Clustered**

* Clustered indexes sort and store the data rows in the table or view based on their key values. These are the columns included in the index definition. There can be only one clustered index per table, because the data rows themselves can be stored in only one order.

**Non Clustered**

A nonclustered index is **an index structure separate from the data stored in a table that reorders one or more selected columns**

* + **Set operators**

## **UNION:**

* UNION will be used to combine the result of two select statements.
* Duplicate rows will be eliminated from the results obtained after performing the UNION operation.

## **UNION ALL**

* This operator combines all the records from both the queries.
* Duplicate rows will be not be eliminated from the results obtained after performing the UNION ALL operation.

## **INTERSECT:**

* It is used to combine two SELECT statements, but it only returns the records which are common from both SELECT statements.

## **EXCEPT**

The SQL **EXCEPT** clause/operator is used to combine two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement. This means EXCEPT returns only rows, which are not available in the second SELECT statement.

* + **CASE Statement**
* The CASE expression goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.
* If there is no ELSE part and no conditions are true, it returns NULL.
  + **IIF Statement**
* IIF is **a shorthand way for writing a CASE expression**. It evaluates the Boolean expression passed as the first argument, and then returns either of the other two arguments based on the result of the evaluation.
  + **Sub –Queries**

In SQL a Subquery can be simply defined as a query within another query. In other words we can say that a Subquery is a query that is embedded in WHERE clause of another SQL query. Important rules for Subqueries:

**Correlated -** A correlated subquery is evaluated once for each row processed by the parent statement. The parent statement can be a **SELECT**, **UPDATE**, or **DELETE** statement.

* + **Non- corelated**

In a non-correlated subquery, **the inner query doesn't depend on the outer query and can run as a stand-alone query**. Subquery used along with IN or NOT IN SQL clause is a good example of Noncorrelated subquery in SQL.

* + **Stored Procedure**
* A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.
* So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.
  + **Views**
* In SQL, a view is a virtual table based on the result-set of an SQL statement.
* A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
  + **System Functions**
  + **Date**

(Dateadd, datediff, datepart, datename, month, year, sysdatetime())

**Getdate();**

Return the current database system date and time:

SELECT GETDATE();

**Dateadd**

Add one year to a date, then return the date:

SELECT DATEADD(year, 1, '2017/08/25') AS DateAdd;

**Datediff**

Return the difference between two date values, in years:

SELECT DATEDIFF(year, '2017/08/25', '2011/08/25') AS DateDiff;

**Datepart**

Return a specified part of a date:

SELECT DATEPART(year, '2017/08/25') AS DatePartInt;

**Datename**

Return a specified part of a date:

SELECT DATENAME(year, '2017/08/25') AS DatePartString;

**Month**

Return the month part of a date:

SELECT MONTH('2017/08/25') AS Month;

**YEAR**

Return the year part of a date:

SELECT YEAR('2017/08/25') AS Year;

**sysdatetime**()

Return the date and time of the SQL Server:

SELECT SYSDATETIME() AS SysDateTime;

* **String**

(Replace,substring,left,right,charindex,Reverse,stuff)

**Replace -** Replace "T" with "M":

SELECT REPLACE('SQL Tutorial', 'T', 'M');

**LEN -** Return the length of a string:

SELECT LEN('W3Schools.com');

**Substring -** Extract 3 characters from a string, starting in position 1:

SELECT SUBSTRING('SQL Tutorial', 1, 3) AS ExtractString;

**Charindex -** Search for "t" in string "Customer", and return position:

SELECT CHARINDEX('t', 'Customer') AS MatchPosition;

**REPLICATE** - Repeat a string:

SELECT REPLICATE('SQL Tutorial', 5);

**Left -** Extract 3 characters from a string (starting from left):

SELECT LEFT('SQL Tutorial', 3) AS ExtractString;

**Reverse -** Reverse a string:

SELECT REVERSE('SQL Tutorial');

**Stuff -** Delete 3 characters from a string, starting in position 1, and then insert "HTML" in position 1:

SELECT STUFF('SQL Tutorial', 1, 3, 'HTML');

**Transaction**

** What is Transaction?**

A transaction is a group of commands that change the data stored in database. A transaction is treated as a single unit. A transaction ensures that either all of the commands succeed or none of them .If one of the command in the transaction fails, all of the commands fail and any data that was modified in database is rollback

**Transaction Processing follows these step**

* Begin Transaction
* Process Database Commands
* Check for errors

 If error occurs

Rollback transaction

Else

commit the transaction

**ACID Properties**

* Atomicity
* Consistency
* Isolation
* Durability

**Atomicity**

* All statements in the transaction either completed successfully or they were rolled back but in any case not left half done.

**Consistancy**

* All data touched by transaction is left in a logically consistent state

**Isolated**

* The transaction must affect data without interfering with other concurrent transaction or being interfered with by them . This parent transactions from making changes to data based on uncommitted information

**Durable**

* Once change is made It is Permanent .If a system error or power failure occurs before a set of commands is complete. Those commands are undone and data is restored to its original state once .the system being running again.

* + **Anomalies**

**Update** **Anomaly**

Employee David has two rows in the table given above since he works in two different departments. If we want to change David’s address, we must do so in two rows, else the data would become inconsistent.

If the proper address is updated in one of the departments but not in another, David will have two different addresses in the database, which is incorrect and leads to inconsistent data.

### **Insert Anomaly**

If a new worker joins the firm and is currently unassigned to any department, we will be unable to put the data into the table because the w\_dept field does not allow nulls.

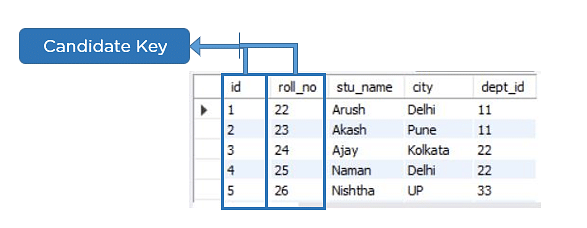
### **Delete Anomaly**

If the corporation closes the department F890 at some point in the future, deleting the rows with w\_dept as F890 will also erase the information of employee Mike, who is solely assigned to this department.

### **Candidate Key**

A candidate key is a set of one or more columns that can identify a record uniquely in a table, and YOU can use each candidate key as a [Primary Key.](https://www.simplilearn.com/tutorials/sql-tutorial/primary-key-in-sql)

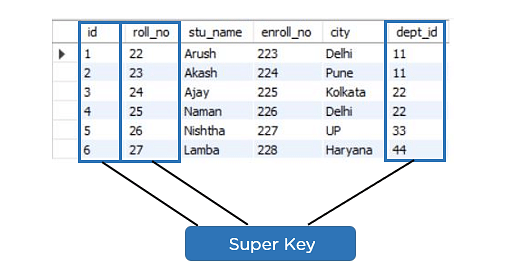
Now, let’s use an example to understand this better.



### **Super Key**

Super key is a set of over one key that can identify a record uniquely in a table, and the Primary Key is a subset of Super Key.

Let’s understand this with the help of an example.



* + **Normalization**
* Normalization is the process of organizing the data in the database.
* Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
* Normalization divides the larger table into smaller and links them using relationships.

### **What is Composite Key?**

A composite key is a primary key composed of multiple columns used to identify a record uniquely

In our database, we have two people with the same name Robert Phil, but they live in different places.

## **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 that does not functionally dependant on any subset of candidate key relation

## 3NF (Third Normal Form) Rules

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

## BCNF (Boyce-Codd Normal Form)

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

* **CTE**

What is a CTE in SQL?

A common table expression, or CTE, is **a temporary named result set created from a simple SELECT statement that can be used in a subsequent SELECT statement**. Each SQL CTE is like a named query, whose result is stored in a virtual table (a CTE) to be referenced later in the main query

* + **Windows Function**

Window functions applies aggregate and ranking functions over a particular window (set of rows). OVER clause is used with window functions to define that window. OVER clause does two things :

* Partitions rows into form set of rows. (PARTITION BY clause is used)
* Orders rows within those partitions into a particular order. (ORDER BY clause is used)

**Ranking Window Functions :**   
Ranking functions are, RANK(), DENSE\_RANK(), ROW\_NUMBER()

* **RANK() –**   
  As the name suggests, the rank function assigns rank to all the rows within every partition. Rank is assigned such that rank 1 given to the first row and rows having same value are assigned same rank. For the next rank after two same rank values, one rank value will be skipped.

* **DENSE\_RANK() –**   
  It assigns rank to each row within partition. Just like rank function first row is assigned rank 1 and rows having same value have same rank. The difference between RANK() and DENSE\_RANK() is that in DENSE\_RANK(), for the next rank after two same rank, consecutive integer is used, no rank is skipped.

* **ROW\_NUMBER() –**   
  It assigns consecutive integers to all the rows within partition. Within a partition, no two rows can have same row number.

**What is Relationship?**

A relationship can be defined as:

• a connection or set of associations, or

• a rule for communication among entities:

Example: In college the database, the association between student and course entity, i.e., “Student opts course” is an example of a relationship.

Degree

The degree of a relationship type is the number of participating entity types. The relationship between two entities is called binary relationship. A relationship among three entities is called ternary relationship. Similarly relationship among n entities is called n-ry relationship.

Relationship Cardinality

Cardinality specifies the number of instances of an entity associated with another entity participating in a relationship. Based on the cardinality binary relationship can be further classified into the following categories:

**One-to-one**:

1

1

Principal

Has

College

An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A. Example 17: Relationship between college and principal

One college can have at the most one principal and one principal can be assigned to only one college.

**One-to-many:**

An entity in A is associated with any number of entities in B. An entity in B is associated with at the most one entity in A.

Example 18: Relationship between department and faculty.

Department

Works -in

Faculty

1

N

One department can appoint any number of faculty members but a faculty member is assigned to only one department.

**Many-to-one:**

An entity in A is associated with at most one entity in B. An entity in B is associated with any number in A.

Example 19: Relationship between course and instructor. An instructor can teach various courses but a course can be taught only by one instructor

N

1

Instructor

Teaches

Course

M

1

**Many-to-many:**

Entities in A and B are associated with any number of entities from each other.

Taught\_by Relationship between course and faculty. One faculty member can be assigned to teach many courses and one course may be taught by many faculty members.

Course

Taught

Faculty

M

N

Relationship between book and author.

One author can write many books and one book can be written by more than one authors.

Book

writes

Author

M

N