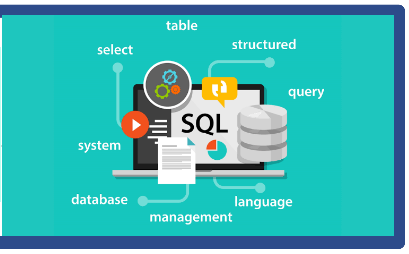
Database in SQL



CREATE BY - ATUL KUMAR (LINKEDIN)

SQL tutorial gives unique learning on **Structured Query Language** and it helps to make practice on SQL commands which provides immediate results. SQL is a language of database, it includes database creation, deletion, fetching rows and modifying rows etc.

SQL is an ANSI (American National Standards Institute) standard but there are many different versions of the SQL language.

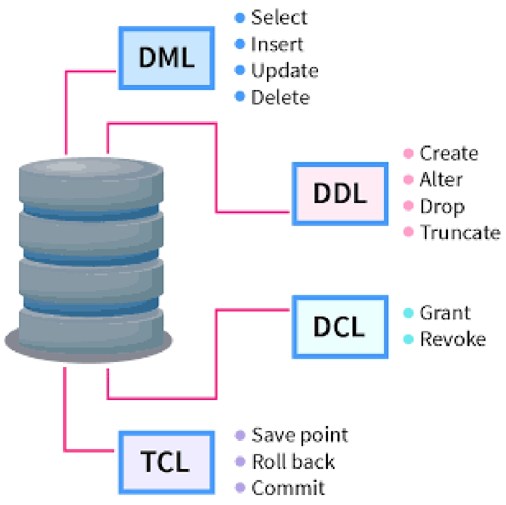
What is SQL?

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in relational database.

SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server use SQL as standard database language.

Also, they are using different dialects, such as:

- MS SQL Server using T-SQL,
- Oracle using PL/SQL,
- MS Access version of SQL is called JET SQL (native format) etc.



Why SQL?

- Allows users to access data in relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures, and views

History:

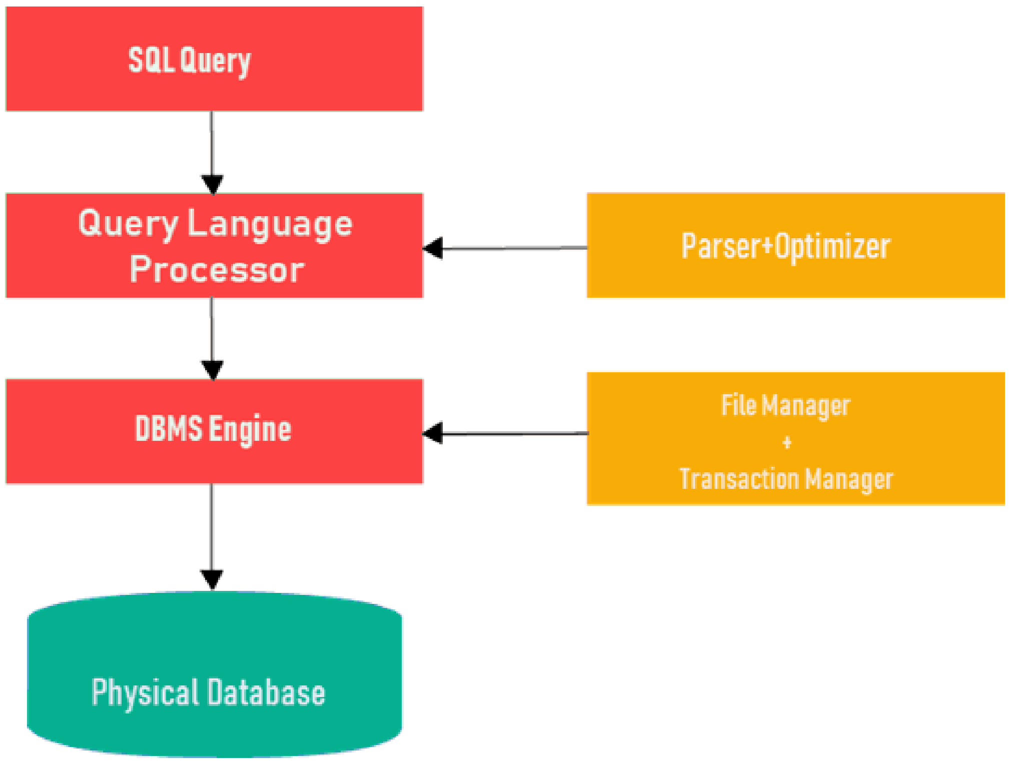
- **1970 --** Dr. Edgar F. "Ted" Codd of IBM is known as the father of relational databases. He described a relational model for databases.
- **1974 --** Structured Query Language appeared.
- **1978 --** IBM worked to develop Codd's ideas and released a product named System/R.
- **1986 --** IBM developed the first prototype of relational database and standardized by ANSI. The first relational database was released by Relational Software and its later becoming Oracle.

SQL Process:

When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.

There are various components included in the process. These components are Query Dispatcher, Optimization Engines, Classic Query Engine and SQL Query Engine, etc. Classic query engine handles all non-SQL queries but SQL query engine won't handle logical files.

Following is a simple diagram showing SQL Architecture:



SQL Commands:

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into groups based on their nature:

DDL - Data Definition Language:

Command	Description
CREATE	Creates a new table, a view of a table, or other object in database
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other object in the database.

DML - Data Manipulation Language:

Command	Description
SELECT	Retrieves certain records from one or more tables
INSERT	Creates a record
UPDATE	Modifies records
DELETE	Deletes records

DCL - Data Control Language:

Command	Description
GRANT	Gives a privilege to user
REVOKE	Takes back privileges granted from user

What is RDBMS?

RDBMS stands for **R**elational **D**atabase **M**anagement **S**ystem. RDBMS is the basis for SQL and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd.

What is table ?

The data in RDBMS is stored in database objects called **tables**. The table is a collection of related data entries and it consists of columns and rows.

Remember, a table is the most common and simplest form of data storage in a relational database.

What is field?

Every table is broken up into smaller entities called fields. The fields in the CUSTOMERS table consist of ID, NAME, AGE, ADDRESS and SALARY.

A field is a column in a table that is designed to maintain specific information about every record in the table.

What is record or row?

A record, also called a row of data, is each individual entry that exists in a table. For example, there are 7 records in the above CUSTOMERS table.

A record is a horizontal entity in a table.

What is column?

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

What is NULL value?

A NULL value in a table is a value in a field that appears to be blank which means A field with a NULL value is a field with no value.

It is very important to understand that a NULL value is different than 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.

SQL Constraints:

Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints could be column level or table level. Column level constraints are applied only to one column where as table level constraints are applied to the whole table.

SQL Syntax:

SQL is followed by unique set of rules and guidelines called Syntax. This tutorial gives you a quick start with SQL by listing all the basic SQL Syntax:

All the SQL statements start with any of the keywords like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW and all the statements end with a semicolon (;).

Important point to be noted is that SQL is **case insensitive** which means SELECT and select have same meaning in SQL statements but MySQL make difference in table names. So if you are working with MySQL then you need to give table names as they exist in the database.

SQL SELECT Statement:

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

SQL DISTINCT Clause:

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

SQL WHERE Clause:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  CONDITION;
```

SQL AND/OR Clause:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  CONDITION-1 {AND|OR} CONDITION-2;
```

SQL IN Clause:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  column_name IN (val-1, val-2,...val-N);
```

SQL BETWEEN Clause:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  column_name BETWEEN val-1 AND val-2;
```

SQL Like Clause:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  column_name LIKE { PATTERN };
```

SQL ORDER BY Clause:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  CONDITION
ORDER BY column_name {ASC|DESC};
```

SQL GROUP BY Clause:

```
SELECT SUM(column_name)
FROM   table_name
WHERE  CONDITION
GROUP BY column_name;
```

SQL COUNT Clause:

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

SQL HAVING Clause:

```
SELECT SUM(column_name)
FROM   table_name
WHERE  CONDITION
GROUP BY column_name
HAVING (arithematic function condition);
```

SQL CREATE TABLE Statement:

```
CREATE TABLE table_name(
column1 datatype,
column2 datatype,
column3 datatype,
.....
columnN datatype,
PRIMARY KEY( one or more columns )
);
```

SQL DROP TABLE Statement:

```
DROP TABLE table_name;
```

SQL CREATE INDEX Statement :

```
CREATE UNIQUE INDEX index_name
ON table_name ( column1, column2,...columnN);
```

SQL DROP INDEX Statement :

```
ALTER TABLE table_name
DROP INDEX index_name;
```

SQL DESC Statement :

```
DESC table_name;
```

SQL TRUNCATE TABLE Statement:

```
TRUNCATE TABLE table_name;
```

SQL ALTER TABLE Statement:

```
ALTER TABLE table_name {ADD|DROP|MODIFY} column_name {data_ype};
```

SQL ALTER TABLE Statement (Rename) :

```
ALTER TABLE table_name RENAME TO new_table_name;
```

SQL INSERT INTO Statement:

```
INSERT INTO table_name( column1, column2....columnN)
VALUES ( value1, value2....valueN);
```

SQL UPDATE Statement:

```
UPDATE table_name
SET column1 = value1, column2 = value2....columnN=valueN
[ WHERE  CONDITION ];
```

SQL DELETE Statement:

```
DELETE FROM table_name
WHERE  {CONDITION};
```

SQL CREATE DATABASE Statement:

```
CREATE DATABASE database_name;
```

SQL DROP DATABASE Statement:

```
DROP DATABASE database_name;
```

SQL USE Statement:

```
USE DATABASE database_name;
```

SQL COMMIT Statement:

```
COMMIT;
```

SQL ROLLBACK Statement:

```
ROLLBACK;
```

SQL - Operators:

SQL Arithmetic Operators:

Assume variable a holds 10 and variable b holds 20, then:

Show Examples

Operator	Description	Example
+	Addition - Adds values on either side of the operator	a + b will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	a - b will give -10
*	Multiplication - Multiplies values on either side of the operator	a * b will give 200
/	Division - Divides left hand operand by right hand operand	b / a will

		give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	b % a will give 0

SQL Comparison Operators:

Assume variable a holds 10 and variable b holds 20, then:

Show Examples

Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a = b) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.
!<	Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.	(a !< b) is false.
!>	Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.	(a !> b) is true.

SQL Logical Operators:

Here is a list of all the logical operators available in SQL.

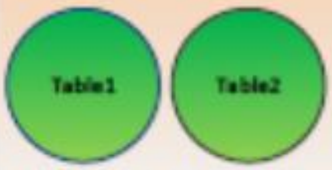
Show Examples

Operator	Description

ALL	The ALL operator is used to compare a value to all values in another value set.
AND	The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
ANY	The ANY operator is used to compare a value to any applicable value in the list according to the condition.
BETWEEN	The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
EXISTS	The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria.
IN	The IN operator is used to compare a value to a list of literal values that have been specified.
LIKE	The LIKE operator is used to compare a value to similar values using wildcard operators.
NOT	The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.**
OR	The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
IS NULL	The NULL operator is used to compare a value with a NULL value.
UNIQUE	The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

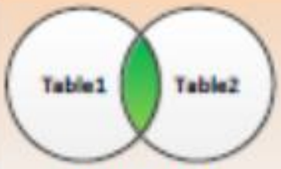
TSQL JOIN TYPES

Created by Steve Stedman



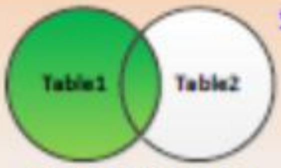
```
SELECT *
  FROM Table1;

SELECT *
  FROM Table2;
```

SELECT from two tables



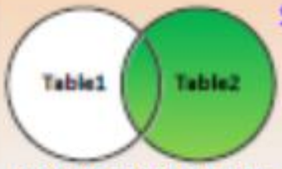
```
SELECT *
  FROM Table1 t1
 INNER JOIN Table2 t2
   ON t1.fk = t2.id;
```

INNER JOIN



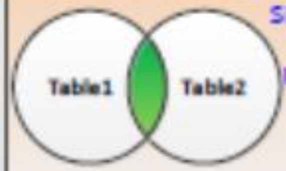
```
SELECT *
  FROM Table1 t1
  LEFT OUTER JOIN Table2 t2
    ON t1.fk = t2.id;
```

LEFT OUTER JOIN



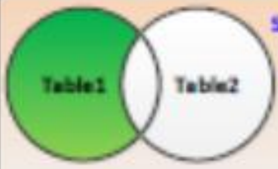
```
SELECT *
  FROM Table1 t1
 RIGHT OUTER JOIN Table2 t2
    ON t1.fk = t2.id;
```

RIGHT OUTER JOIN



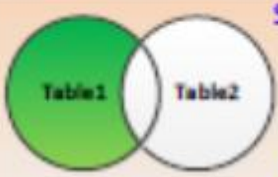
```
SELECT *
  FROM Table1 t1
WHERE EXISTS (SELECT 1
                FROM Table2 t2
               WHERE t1.fk = t2.id
             );
```

SEMI JOIN



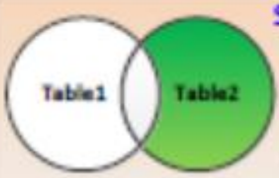
```
SELECT *
  FROM Table1 t1
WHERE NOT EXISTS (SELECT 1
                    FROM Table2 t2
                   WHERE t1.fk = t2.id
                 );
```

ANTI SEMI JOIN



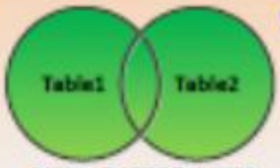
```
SELECT *
  FROM Table1 t1
  LEFT OUTER JOIN Table2 t2
    ON t1.fk = t2.id
 WHERE t2.id IS NULL;
```

LEFT OUTER JOIN with exclusion
– replacement for a NOT IN



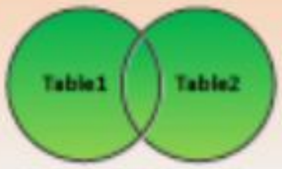
```
SELECT *
  FROM Table1 t1
  RIGHT OUTER JOIN Table2 t2
    ON t1.fk = t2.id
 WHERE t1.fk IS NULL;
```

RIGHT OUTER JOIN with exclusion
– replacement for a NOT IN



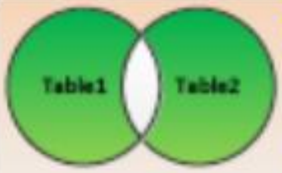
```
SELECT *
  FROM Table1 t1
  FULL OUTER JOIN Table2 t2
    ON t1.fk = t2.id;
```

FULL OUTER JOIN



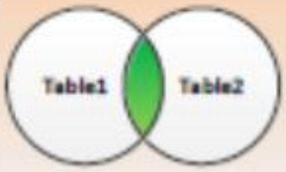
```
SELECT *
  FROM Table1 t1
 CROSS JOIN Table2 t2;
```

CROSS JOIN, the Cartesian product



```
SELECT *
  FROM Table1 t1
  FULL OUTER JOIN Table2 t2
    ON t1.fk = t2.id
 WHERE t1.fk IS NULL
    OR t2.id IS NULL;
```

FULL OUTER JOIN with exclusion

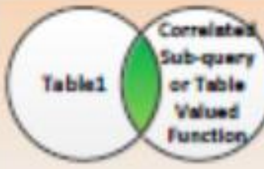


```
SELECT *
  FROM Table1 t1
 INNER JOIN Table2 t2
   ON t1.fk >= t2.id;
```

NON-EQUI INNER JOIN

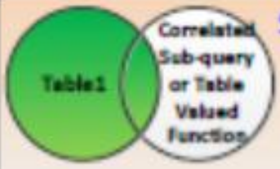
TSQL JOIN TYPES

Created by Steve Stedman



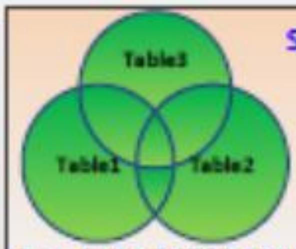
```
SELECT *
  FROM Table1 t1
 CROSS APPLY
      [dbo].[someTVF](t1.fk)
      AS t;
```

CROSS APPLY



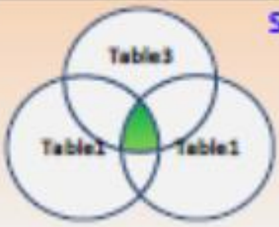
```
SELECT *
  FROM Table1 t1
 OUTER APPLY
      [dbo].[someTVF](t1.fk)
       AS t;
```

OUTER APPLY



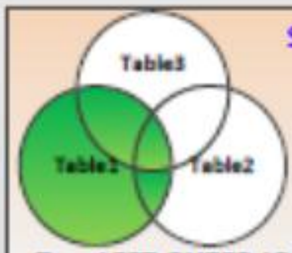
```
SELECT *
  FROM Table1 t1
  FULL OUTER JOIN Table2 t2
    ON t1.fk = t2.id
  FULL OUTER JOIN Table3 t3
    ON t1.fk_table3 = t3.id;
```

Two FULL OUTER JOINS



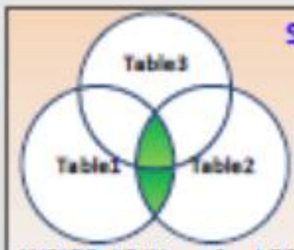
```
SELECT *
  FROM Table1 t1
 INNER JOIN Table2 t2
    ON t1.fk = t2.id
 INNER JOIN Table3 t3
    ON t1.fk_table3 = t3.id;
```

Two INNER JOINs



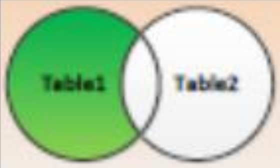
```
SELECT *
  FROM Table1 t1
  LEFT OUTER JOIN Table2 t2
    ON t1.fk = t2.id
  LEFT OUTER JOIN Table3 t3
    ON t1.fk_table3 = t3.id;
```

Two LEFT OUTER JOINS



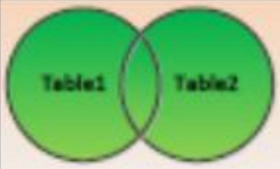
```
SELECT *
  FROM Table1 t1
 INNER JOIN Table2 t2
    ON t1.fk = t2.id
  LEFT OUTER JOIN Table3 t3
    ON t1.fk_table3 = t3.id;
```

INNER JOIN and a LEFT OUTER JOIN



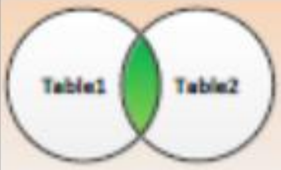
```
SELECT fk as id
  FROM Table1
EXCEPT
SELECT ID
  FROM Table2;
```

EXCEPT



```
SELECT fk as id
  FROM Table1
UNION
SELECT ID
  FROM Table2;
```

UNION



```
SELECT fk as id
  FROM Table1
INTERSECT
SELECT ID
  FROM Table2;
```

INTERSECT

Sample Schema

Table 1 (People)

	id	Name	fk	fk_table3
1	1	Steve	1	NULL
2	2	Aaron	3	NULL
3	3	Mary	2	NULL
4	4	Fred	1	NULL
5	5	Anne	5	NULL
6	6	Beth	5	1
7	7	Johnny	NULL	1
8	8	Karen	NULL	2

Table 2 (Favorite Colors)

	id	FavoriteColor
1	1	red
2	2	green
3	3	blue
4	4	pink
5	5	purple
6	6	mauve
7	7	orange
8	8	yellow
9	1	indigo

Table 3 (Favorite Foods)

	id	dataValue
1	1	Pizza
2	2	Burger
3	3	Sushi

Note: Column names are very generic to simplify the sample queries.
Foreign keys are
Table1.fk -> Table2.id
Table2.fk_table3 -> Table3.id