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

What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases

What Can SQL 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

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

Keep in Mind That...

- SQL keywords are NOT case sensitive: select is the same as SELECT

Semicolon after SQL Statements?

Some database systems require a semicolon at the end of each SQL statement.

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.

SELECT Syntax

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

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

```sql
SELECT * FROM Customers;
```

SELECT DISTINCT Syntax

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

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

WHERE Syntax

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

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

AND Syntax

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

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

OR Syntax

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

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

NOT Syntax

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

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

Combining AND, OR and NOT

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

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

ORDER BY Syntax

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

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

ORDER BY Several Columns

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

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

INSERT INTO Syntax

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

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

What is a NULL Value?

A field with a NULL value is a field with no value.

IS NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

IS NOT NULL Syntax

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

UPDATE Syntax

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

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

DELETE Syntax

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

SELECT TOP Clause

MySQL Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
LIMIT number;
```

MIN() Syntax

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

MAX() Syntax

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

COUNT() Syntax

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

AVG() Syntax

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

SUM() Syntax

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

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 represents a single character

LIKE Syntax

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

LIKE Operator	Description
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"

SQL 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.

The following SQL statement selects all customers with a City starting with "b", "s", or "p":

```
SELECT * FROM Customers
WHERE City LIKE '[bsp]%';
```

The following SQL statement selects all customers with a City starting with "a", "b", or "c":

```
SELECT * FROM Customers
WHERE City LIKE '[a-c]%';
```

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);
```

```
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;
```

```
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 NOT BETWEEN 'Carnarvon 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 the query.

Alias Column Syntax

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

It requires double quotation marks or square brackets if the alias name contains spaces:

```
SELECT CustomerName AS Customer, ContactName AS [Contact Person]
FROM Customers;
```

```
SELECT CustomerName, Address + ', ' + PostalCode + ' ' + City + ', ' +
Country AS Address
FROM Customers;
```

Alias Table Syntax

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

```
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;
```

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

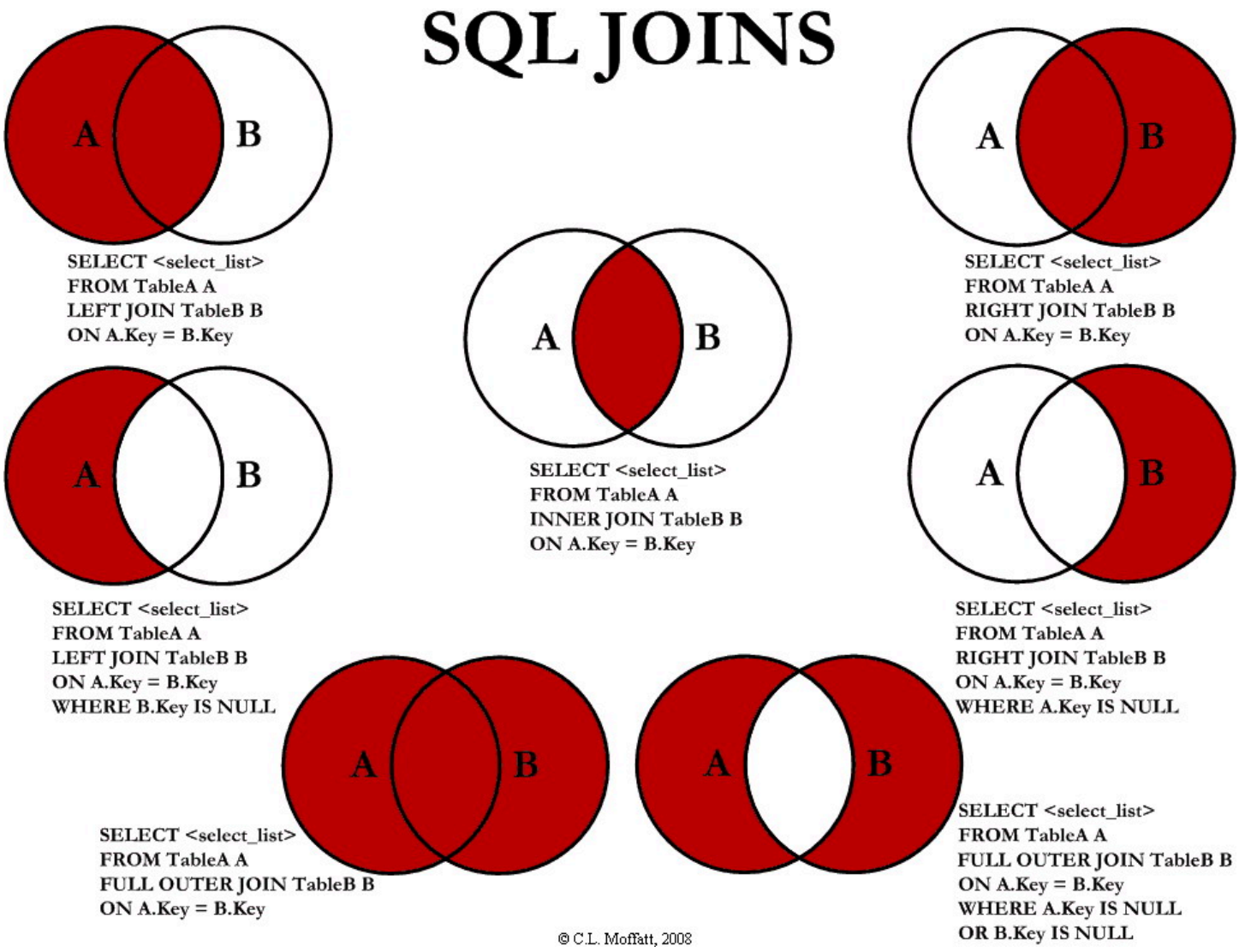
SQL JOIN

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

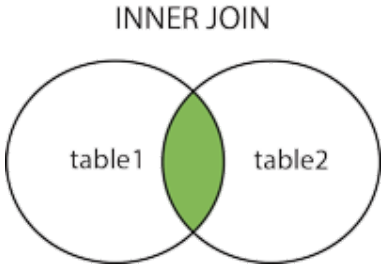


SQL INNER JOIN

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;
```




```
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.

JOIN Three Tables

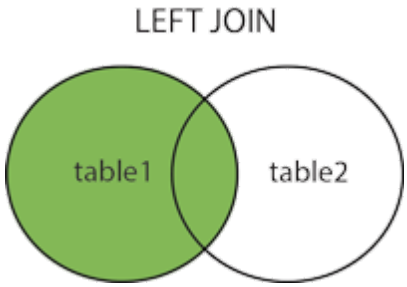
```
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

SQL LEFT JOIN

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.

LEFT JOIN Syntax

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



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

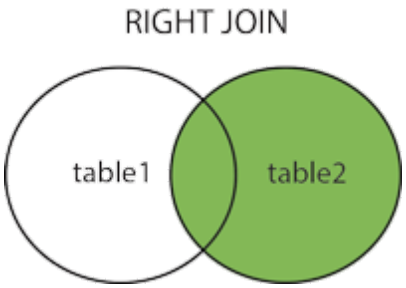
```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

SQL RIGHT JOIN

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.

RIGHT JOIN Syntax

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



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

```
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

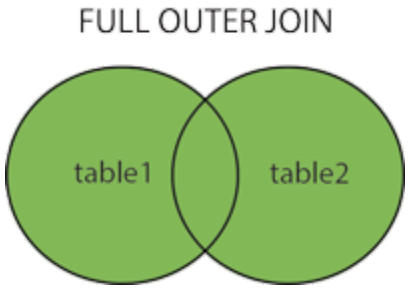
The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

Note: FULL OUTER JOIN can potentially return very large result-sets!

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_n
ame
WHERE condition;
```




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

SQL Self JOIN

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

Self JOIN Syntax

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

```
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;
```

SQL CREATE VIEW Statement

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.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName, Price
FROM Products
WHERE Price > (SELECT AVG(Price) FROM Products);
```

```
CREATE OR REPLACE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName, City
FROM Customers
WHERE Country = 'Brazil';
```

```
DROP VIEW [Brazil Customers];
```

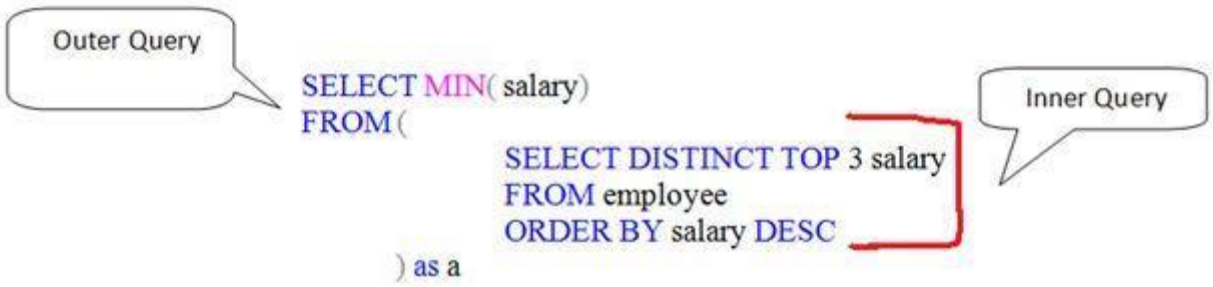
Trigger: A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

Syntax:

```
create trigger [trigger_name]
[before | after]
{insert | update | delete}
on [table_name]
[for each row]
[trigger_body]
```

Authentication and authorization

A fundamental step in securing a **database** system is validating the identity of the user who is accessing the **database** (**authentication**) and controlling what operations they can perform (**authorization**). A strong **authentication and authorization** strategy helps protect the users and their data from attackers.



Nth highest salary in MySQL using LIMIT clause:

```
SELECT salary FROM Employee

ORDER BY Salary DESC

LIMIT n-1,1
```

Top Keyword:

```
SELECT TOP 1 salary
FROM (
    SELECT DISTINCT TOP n salary
    FROM employee
    ORDER BY salary DESC
    ) a
ORDER BY salary
```

Normalization is the process of minimizing **redundancy** from a relation or set of relations. Redundancy in relation may cause insertion, deletion and updation anomalies. So, it helps to minimize the redundancy in relations. **Normal forms** are used to eliminate or reduce redundancy in database tables.

Database Normal Forms

Here is a list of Normal Forms

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

FULL NAMES	PHYSICAL ADDRESS	MOVIES RENTED	SALUTATION
Janet Jones	First Street Plot No 4	Pirates of the Caribbean, Clash of the Titans	Ms.
Robert Phil	3rd Street 34	Forgetting Sarah Marshal, Daddy's Little Girls	Mr.
Robert Phil	5th Avenue	Clash of the Titans	Mr.

1NF (First Normal Form) Rules

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

FULL NAMES	PHYSICAL ADDRESS	MOVIES RENTED	SALUTATION
Janet Jones	First Street Plot No 4	Pirates of the Caribbean	Ms.
Janet Jones	First Street Plot No 4	Clash of the Titans	Ms.
Robert Phil	3rd Street 34	Forgetting Sarah Marshal	Mr.
Robert Phil	3rd Street 34	Daddy's Little Girls	Mr.
Robert Phil	5th Avenue	Clash of the Titans	Mr.

What is a KEY?

A KEY is a value used to identify a record in a table uniquely. A KEY could be a single column or combination of multiple columns

Note: Columns in a table that are NOT used to identify a record uniquely are called non-key columns.

What is a Primary Key?

A primary is a single column value used to identify a database record uniquely.

It has following attributes

- A primary key cannot be NULL
- A primary key value must be unique
- The primary key values should rarely be changed
- The primary key must be given a value when a new record is inserted.

What is Composite Key?

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

2NF (Second Normal Form) Rules

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

It is clear that we can't move forward to make our simple database in 2nd Normalization form unless we partition the table above.

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3rd Street 34	Mr.
3	Robert Phil	5th Avenue	Mr.

Table 1

MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

Table 2

We have divided our 1NF table into two tables viz. Table 1 and Table2. Table 1 contains member information. Table 2 contains information on movies rented.

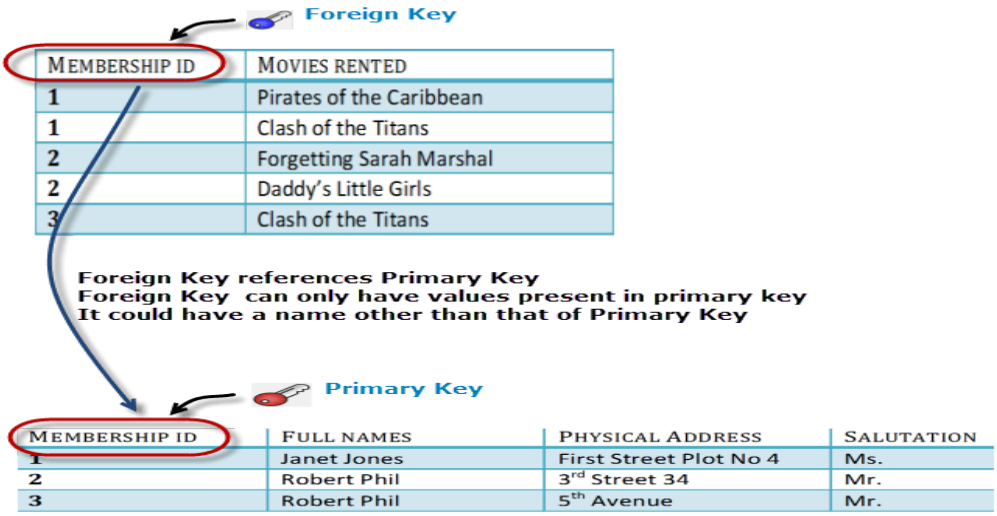
We have introduced a new column called Membership_id which is the primary key for table 1. Records can be uniquely identified in Table 1 using membership id

Database - Foreign Key

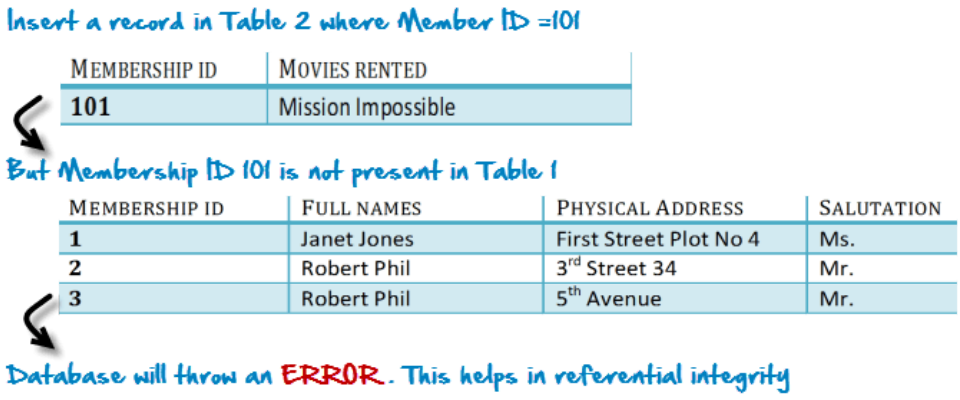
In Table 2, Membership_ID is the Foreign Key

Foreign Key references the primary key of another Table! It helps connect your Tables

- A foreign key can have a different name from its primary key
- It ensures rows in one table have corresponding rows in another
- Unlike the Primary key, they do not have to be unique. Most often they aren't
- Foreign keys can be null even though primary keys can not



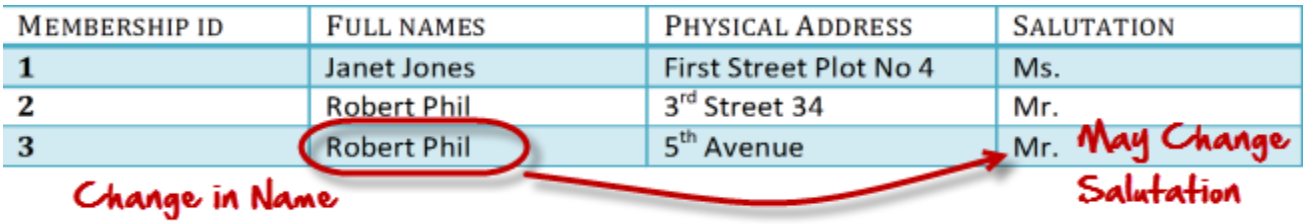
You will only be able to insert values into your foreign key that exist in the unique key in the parent table. This helps in referential integrity.



What are transitive functional dependencies?

A transitive functional dependency is when changing a non-key column, might cause any of the other non-key columns to change

Consider the table 1. Changing the non-key column Full Name may change Salutation.



3NF (Third Normal Form) Rules

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

3NF Example

Below is a 3NF example in SQL database:

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION ID
1	Janet Jones	First Street Plot No 4	2
2	Robert Phil	3rd Street 34	1
3	Robert Phil	5th Avenue	1

TABLE 1

MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

Table 2

SALUTATION ID	SALUTATION
1	Mr.
2	Ms.
3	Mrs.
4	Dr.

Table 3

We have again divided our tables and created a new table which stores Salutations.

There are no transitive functional dependencies, and hence our table is in 3NF

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.

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

What is a Database Transaction?

A **Database Transaction** is a logical unit of processing in a DBMS which entails one or more database access operation. In a nutshell, database transactions represent real-world events of any enterprise.

A transaction can be defined as a group of tasks. A single task is the minimum processing unit which cannot be divided further.

ACID Properties

A transaction is a very small unit of a program and it may contain several lowlevel tasks. A transaction in a database system must maintain **A**tomicity, **C**onsistency, **I**solation, and **D**urability − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity.

- **Atomicity** − This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where

a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.

- **Consistency** − The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
- **Durability** − The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.
- **Isolation** − In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

https://www.w3schools.com/sql/sql_injection.asp

Aggregate functions in SQL

In database management an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

Various Aggregate Functions

```
1) Count() 2) Sum() 3) Avg() 4) Min() 5) Max()
```

Difference between DELETE, DROP and TRUNCATE

1. DELETE :

Basically, it is a Data Manipulation Language Command (DML). It is use to delete the one or more tuples of a table. With the help of “DELETE” command we can either delete all the rows in one go or can delete row one by one. i.e., we can use it as per the requirement or the condition using Where clause. It is comparatively slower than TRUNCATE cmd.

- **SYNTAX –**
  If we want to delete all the rows of the table:
  ```
  DELETE from ;
  ```
- **SYNTAX –**
  If we want to delete the row of the table as per the condition then we use WHERE clause,
  ```
  DELETE from  WHERE  ;
  ```

Note –

Here we can use the “ROLLBACK” command to restore the tuple.

2. DROP :

It is a Data Definition Language Command (DDL). It is use to drop the whole table. With the help of “DROP” command we can drop (delete) the whole structure in one go i.e. it removes

the named elements of the schema. By using this command the existence of the whole table is finished or say lost.

- **SYNTAX –**
  If we want to drop the table:
  ```
  DROP table ;
  ```

Note –
Here we can't restore the table by using the "ROLLBACK" command.

3. TRUNCATE :
It is also a Data Definition Language Command (DDL). It is use to delete all the rows of a relation (table) in one go. With the help of "TRUNCATE" command we can't delete the single row as here WHERE clause is not used. By using this command the existence of all the rows of the table is lost. It is comparatively faster than delete command as it deletes all the rows fastly.

- **SYNTAX –**
  If we want to use truncate :
  ```
  TRUNCATE  ;
  ```

Note –
Here we can't restore the tuples of the table by using the "ROLLBACK" command
https://www.c-sharpcorner.com/blogs/difference-between-truncate-delete-and-drop-in-sql-server1

TRUNCATE is a DDL command, DELETE is a DML command. This is because TRUNCATE actually drops & re-creates the table, and resets the table's metadata (this is why TRUNCATE does not support a WHERE clause). If the table has an auto-increment primary key, it will be reset

A view is a virtual table. A view consists of rows and columns just like a table. The difference between a view and a table is that views are definitions built on top of other tables (or views), and do not hold data themselves. If data is changing in the underlying table, the same change is reflected in the view. A view can be built on top of a single table or multiple tables. It can also be built on top of another view. In the SQL Create View page, we will see how a view can be built.

Views can provide **advantages** over tables: **Views** can represent a subset of the data contained in a table. Consequently, a **view** can limit the degree of exposure of the underlying tables to the outer world: a given user may have permission to query the **view**, while denied access to the rest of the base table.

Views offer the following advantages:

1. Ease of use: A view hides the complexity of the database tables from end users. Essentially we can think of views as a layer of abstraction on top of the database tables.
2. Space savings: Views takes very little space to store, since they do not store actual data.
3. Additional data security: Views can include only certain columns in the table so that only the non-sensitive columns are included and exposed to the end user. In addition, some databases allow views to have different security settings, thus hiding sensitive data from prying eyes.

Advantages of views

Security

Each user can be given permission to access the database only through a small set of views that contain the specific data the user is authorized to see, thus restricting the user's access to stored data

Query Simplicity
A view can draw data from several different tables and present it as a single table, turning multi-table queries into single-table queries against the view.

Structural simplicity
Views can give a user a "personalized" view of the database structure, presenting the database as a set of virtual tables that make sense for that user.

Consistency
A view can present a consistent, unchanged image of the structure of the database, even if the underlying source tables are split, restructured, or renamed.

Data Integrity
If data is accessed and entered through a view, the DBMS can automatically check the data to ensure that it meets the specified integrity constraints.

Logical data independence.
View can make the application and database tables to a certain extent independent. If there is no view, the application must be based on a table. With the view, the program can be established in view of above, to view the program with a database table to be separated.

Disadvantages of views

Performance
Views create the appearance of a table, but the DBMS must still translate queries against the view into queries against the underlying source tables. If the view is defined by a complex, multi-table query then simple queries on the views may take considerable time.

Update restrictions
When a user tries to update rows of a view, the DBMS must translate the request into an update on rows of the underlying source tables. This is possible for simple views, but more complex views are often restricted to read-only.

What is Indexing?

Indexing is a data structure technique which allows you to quickly retrieve records from a database file. An Index is a small table having only two columns. The first column comprises a copy of the primary or candidate key of a table. Its second column contains a set of pointers for holding the address of the disk block where that specific key value stored.

An index -

- Takes a search key as input
- Efficiently returns a collection of matching records.

https://www.guru99.com/indexing-in-database.html

The **good** thing in **sub-queries** is that they are more readable than **JOIN** s: that's why most new SQL people prefer them; it is the easy way; but when it comes to **performance**, **JOINS** are **better** in most cases even though they are not hard to read too.

The advantage of a **join** includes that it executes **faster**. The retrieval time of the query using **joins** almost always will be **faster** than that of a **subquery**. By using **joins**, you can maximize the calculation burden on the database i.e., instead of multiple queries using one **join** query.

Like functions in programming languages, SQL Server user-defined functions are routines that accept parameters, perform an action, such as a complex calculation, and return the result of that action as a value. The return value can either be a single scalar value or a result set.

Is NoSQL faster than SQL?

In general, NoSQL is not faster than SQL just as SQL is not faster than NoSQL. For those that didn't get that statement, it means that speed as a factor for SQL and NoSQL databases depends on the context.

SQL databases are normalized databases where the data is broken down into various logical tables to avoid data redundancy and data duplication. In this scenario, SQL databases are faster than their NoSQL counterparts for joins, queries, updates, etc.

On the other hand, NoSQL databases are specifically designed for unstructured data which can be document-oriented, column-oriented, graph-based, etc. In this case, a particular data entity is stored together and not partitioned. So, performing read or write operations on a single data entity is faster for NoSQL databases as compared to SQL databases.

Key highlights on SQL vs NoSQL:

SQL	NoSQL
RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS)	Non-relational or distributed database system.
These databases have fixed or static or predefined schema	They have have dynamic schema
These databases are not suited for hierarchical data storage.	These databases are best suited for hierarchical data storage.
These databases are best suited for complex queries	These databases are not so good for complex queries
Vertically Scalable	Horizontally scalable
Follows ACID property	Follows CAP(consistency, availability, partition tolerance)

Keys in Database

Primary Key
A Primary key uniquely identifies each record in a table and must never be the same for the 2 records. Primary key is a set of one or more fields (columns) of a table that uniquely identify a record in database table. A table can have only one primary key and one candidate key can select as a primary key. The primary key should be chosen such that its attributes are never or rarely changed,

for example, we can't select Student_Id field as a primary key because in some case Student_Id of student may be changed.

Candidate Key
A Candidate key is an attribute or set of attributes that uniquely identifies a record. Among the set of candidate, one candidate key is chosen as Primary Key. So a table can have multiple candidate key but each table can have maximum one primary key.

Alternate Key
Alternate keys are candidate keys that are not selected as primary key. Alternate key can also work as a primary key. Alternate key is also called "**Secondary Key**".

Composite Key
Composite key is a combination of more than one attributes that can be used to uniquely identity each record. It is also known as "Compound" key. A composite key may be a candidate or primary key.

Foreign Keys
Foreign key is used to generate the relationship between the tables. Foreign Key is a field in database table that is Primary key in another table. A foreign key can accept null and duplicate value.

Super Key
Super key is a set of an attribute which can uniquely identify a tuple. Super key is a superset of a candidate key.
Super key is a set of one or more than one keys that can be used to uniquely identify the record in table. A Super key for an entity is a set of one or more attributes whose combined value uniquely identifies the entity in the entity set. A super key is a combine form of Primary Key, Alternate key and Unique key and Primary Key, Unique Key and Alternate Key are subset of super key. A Super Key is simply a non-minimal Candidate Key, that is to say one with additional columns not strictly required to ensure uniqueness of the row. A super key can have a single column.

Natural Keys

A natural key is a key composed of columns that actually have a logical relationship to other columns within a table. For example, if we use Student_Id, Student_Name and Father_Name columns to form a key then it would be "Natural Key" because there is definitely a relationship between these columns and other columns that exist in table. Natural keys are often called "Business Key" or "Domain Key".

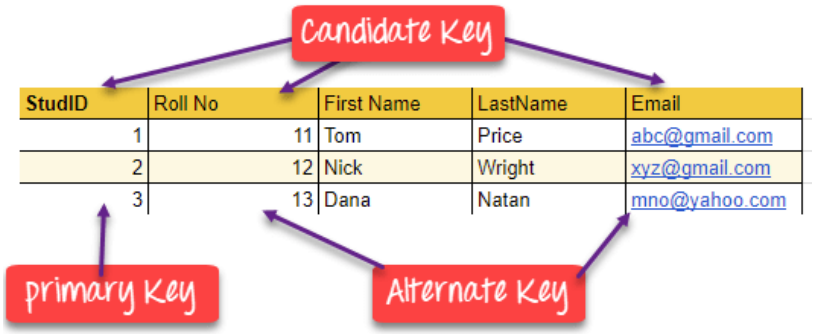
Surrogate Key

Surrogate key is an artificial key that is used to uniquely identify the record in table. For example, in SQL Server or Sybase database system contain an artificial key that is known as "**Identity**". Surrogate keys are just simple sequential number. Surrogate keys are only used to act as a primary key.

Minimal Super Key

A minimal super key is a minimum set of columns that can be used to uniquely identify a row. In other words the minimum number of columns that can be combined to give a unique value for every row in the table.

- A **super key** uniquely identifies a row. It could be made up of one column or many. A **composite key** is a **key** made of more than one column. If a **Super Key** is made of more than one column it is also a **composite**.



Anomalies are problems that can occur in poorly planned, un-normalised **databases** where all the data is stored in one table (a flat-file **database**). Insertion **Anomaly** - The nature of a **database** may be such that it is not possible to add a required piece of data unless another piece of unavailable data is also added.

Insertion Anomaly - The nature of a database may be such that it is not possible to add a required piece of data unless another piece of unavailable data is also added. E.g. A library database that cannot store the details of a new member until that member has taken out a book.

Deletion Anomaly - A record of data can legitimately be deleted from a database, and the deletion can result in the deletion of the only instance of other, required data, E.g. Deleting a book loan from a library member can remove all details of the particular book from the database such as the author, book title etc.

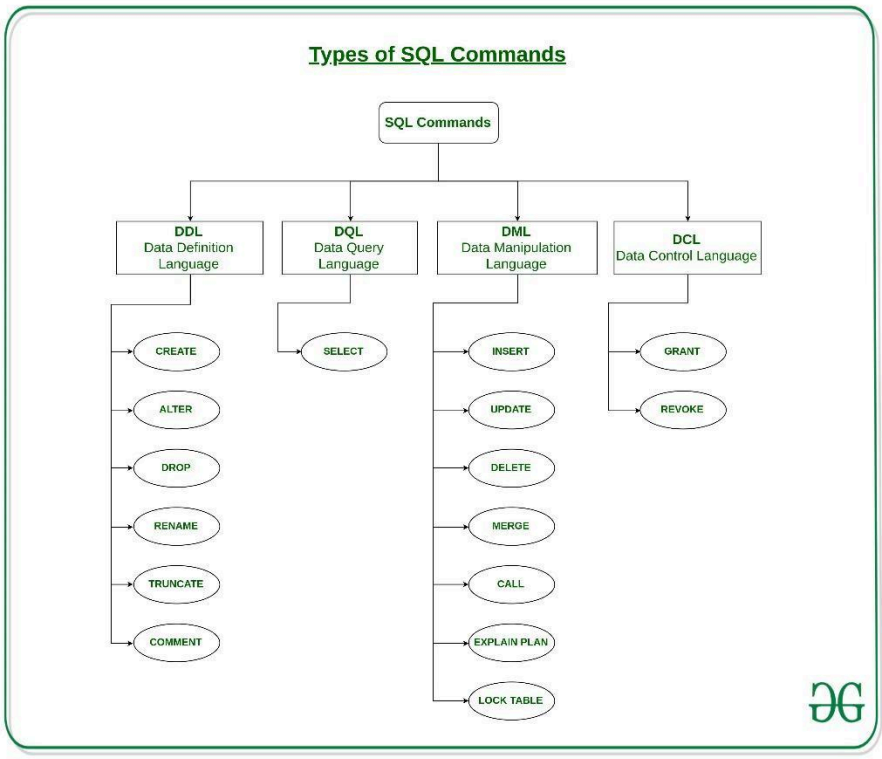
Modification Anomaly - Incorrect data may have to be changed, which could involve many records having to be changed, leading to the possibility of some changes being made incorrectly

Structured Query Language(SQL) as we all know is the database language by the use of which we can perform certain operations on the existing database and also we can use this language to create a database. SQL uses certain commands like Create, Drop, Insert etc. to carry out the required tasks.

These SQL commands are mainly categorized into four categories as:

1. DDL – Data Definition Language
2. DQl – Data Query Language
3. DML – Data Manipulation Language
4. DCL – Data Control Language

Though many resources claim there to be another category of SQL clauses **TCL – Transaction Control Language**. So we will see in detail about TCL as well.



1. **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 the 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.
   - COMMENT –is used to add comments to the data dictionary.
   - RENAME –is used to rename an object existing in the database.
2. **DQL (Data Query Language) :**
   DML statements are used for performing queries on the data within schema objects. The purpose of DQL Command is to get some schema relation based on the query passed to it.

   Example of DQL:
   - SELECT – is used to retrieve data from the a database.
3. **DML(Data Manipulation Language) :** The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements.
   Examples of DML:
   - 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.

4. **DCL(Data Control Language) :** DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.
   Examples of DCL commands:
   - **GRANT**-gives user's access privileges to database.
   - **REVOKE**-withdraw user's access privileges given by using the GRANT command.
5. **TCL(transaction Control Language) :** TCL commands deals with the transaction within the database.
   Examples of TCL commands:
   - **COMMIT**– commits a Transaction.
   - ROLLBACK– rollbacks a transaction in case of any error occurs.
   - **SAVEPOINT**–sets a savepoint within a transaction.
   - **SET TRANSACTION**–specify characteristics for the transaction.

COMMIT :
COMMIT in SQL is a transaction control language which is used to permanently save the changes done in the transaction in tables/databases. The database cannot regain its previous state after the execution of it.

ROLLBACK :
ROLLBACK in SQL is a transactional control language which is used to undo the transactions that have not been saved in database. The command is only be used to undo changes since the last COMMIT.

Difference between COMMIT and ROLLBACK :

COMMIT	ROLLBACK
COMMIT permanently saves the changes made by current transaction.	ROLLBACK undo the changes made by current transaction.
Transaction can not undo changes after COMMIT execution.	Transaction reaches its previous state after ROLLBACK.
When transaction is successful, COMMIT is applied.	When transaction is aborted, ROLLBACK occurs.

COMMIT Command:
A COMMIT is a database command used in transaction management to save all changes made to the transaction as final. Once Commit is applied the transaction can never be reverted. The syntax for commit is COMMIT;
Example:

```
SQL> DELETE FROM STUDENTS
WHERE Student_Id=25;
SQL> COMMIT;
```

ROLLBACK Command:
A ROLLBACK is a database command used in transaction management to revert the previous changes on the transaction. This can be used to revert the changes on the transaction that are made only after the last COMMIT or ROLLBACK command. The syntax for rollback is ROLLBACK;
Example:

```
SQL> DELETE FROM STUDENTS
WHERE Student_Id=25;
```

```
SQL> ROLLBACK;
```

In the database, **View is a virtual table that combines the result set of a stored query**. It is very important when we want to restrict a certain user from accessing the entire database. **View** is dynamic and can be computed from the data in the database. Changing the data in a table alters the data shown in the **view** as well.

When the Professor applies this technique, the student got to see their marks only and thus create a positive impact on the students as they are now competing with the one person only, themselves.

In a relational database, a **view** is not the part of a relational schema.

```
create
view view10
select marks from student
where rollno = 10;
```

Advantages of a view in DBMS

1. Views can subset the data in a table.
2. Views can join and simplify the tables in a virtual table.
3. Views do not require additional storage.
4. Views can hide the complexity of the database and the data the user must hide that.
5. Views can act as aggregated tables where aggregated data (sum, average, etc.) are calculated and presented as part of data.
6. Views can provide additional security from unauthorized users and unauthorized access.