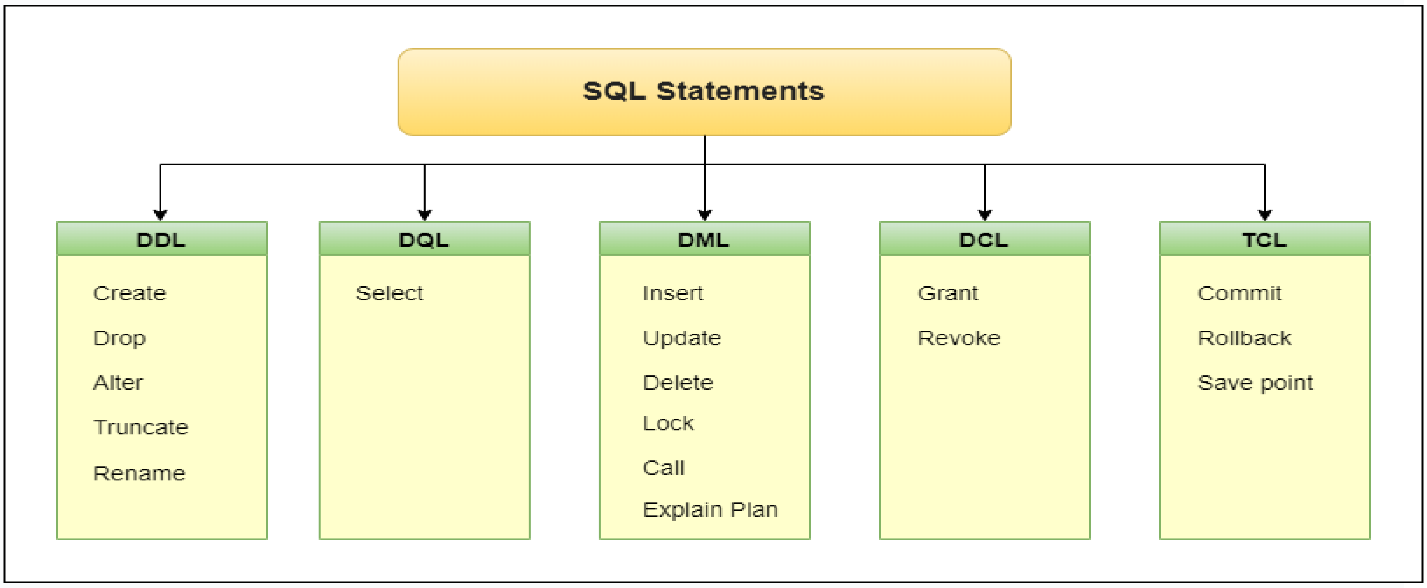
What is SQL?

Structured Query Language is a domain-specific language designed for managing data held in a relational database management system.

SQL was initially developed at IBM by Donald D. Chamberlin and Raymond F. Boyce after learning about the relational model from Edgar F. Codd in the early 1970s.

SQL commands are like instructions to a table. It is used to interact with the database with some operations. It is also used to perform specific tasks, functions, and queries of data. SQL can perform various tasks like creating a table, adding data to tables, dropping the table, modifying the table, set permission for users.

These sql commands are mainly categorized into five categories:
1. DDL – Data Definition Language
2. DQL – Data Query Language
3. DML – Data Manipulation Language
4. DCL – Data Control Language
5. TCL – Transaction Control Language



DCL (Data Control Language) Command in SQL

DCL or Data Control Language is to provide rights, permissions, and other controls of the database system

GRANT Command in SQL

GRANT command is helpful to provide privileges to the database.

Syntax
GRANT privileges ON object TO user;
Example: GRANT INSERT, SELECT on accounts TO Alex;

REVOKE Command in SQL

SQL Revoke command is to withdraw the user's access privileges given by using the GRANT command.

Syntax
REVOKE privileges ON object FROM user;
Example: REVOKE INSERT, SELECT on accounts FROM John;

TCL commands in SQL

Transaction control language or TCL commands deal with the transaction within the database.

COMMIT

This command is used to save all the transactions to the database.

Syntax:

```
COMMIT;
```

For example:

```
DELETE FROM Students
WHERE RollNo =25;
COMMIT;
```

ROLLBACK

Rollback command allows you to undo transactions that have not already been saved to the database.

Syntax:

```
ROLLBACK;
```

Example:

```
DELETE FROM Students
WHERE RollNo =25;
ROLLBACK;
```

The syntax for rollback to save point command is

```
ROLLBACK TO SavepointName;
```

SAVEPOINT

This command helps you to sets a savepoint within a transaction.

Syntax:

```
SAVEPOINT SAVEPOINT_NAME;
```

Example:

```
SAVEPOINT RollNo;
```

What is Database Security in DBMS?

Database security in DBMS is a technique for protecting and securing a database from intentional or accidental threats. As a result, **database security** encompasses hardware parts, software parts, human resources, and data.

We consider database security in the following scenarios:

- Theft and fraudulent.

- Loss of Data privacy.
- Loss of Data integrity.
- Loss of confidentiality or secrecy
- Loss of availability of data.

Why Database Security is Important?

- **Compromise of intellectual property:** Our intellectual property—trade secrets, inventions, or unique methods—could be essential for our ability to sustain an advantage in our industry. If our intellectual property is stolen or leaked, then we will lose our competitive advantage and it may be difficult to maintain or recover.
- **The reputational harm is done to our brand:** Customers or partners may refuse to buy goods or services from us (or do business with us) if they do not believe they can trust our company to protect their data or their own.
- **The concept of business continuity (or lack of it):** Some businesses are unable to operate until a breach has been resolved.
- **Penalties or fines to be paid for failure:** The cost of failing to comply with international regulations lead to fines exceeding many millions of dollars in the worst-case scenario.
- **Costs of correcting breaches and notifying consumers about them:** Along with notifying customers of a breach, the organization that was breached must fund the investigation and forensic services such as crisis management, triage repairs to the affected systems, and much more.

Database Security Threats

Many software vulnerabilities, **misconfigurations**, or practices of misuse or carelessness could lead to breaches. The following are some of the most well-known causes and types of database security cyber threats.

1) SQL INJECTION

It is a type of attack which occurs when a malicious code is injected into frontend (web) apps and then transmitted to the backend database. SQL injections provide hackers with unrestricted access to any data saved in a database.

Any database system is vulnerable to these attacks if developers do not follow secure coding practices and the organization does not conduct regular vulnerability testing.

2) Malware

Malware is software designed to corrupt data or harms a database. Malware could enter your system via any endpoint device connected to the database's network and exploit vulnerabilities in your system. Malware protection is important on any endpoint, but it is particularly necessary on database servers due to their high value and sensitivity. **Examples** of common malware include spyware, Trojan viruses, viruses, worms, adware, and ransomware.

3) Lack of Security Expertise and Education

Databases are breached and leaked due to insufficient level of IT security expertise and education of non-technical employees, who may violate basic database security standards and endanger databases. IT security employees may also lack the necessary expertise to create security controls, enforce rules, or execute incident response processes.

4) Denial of Service (DoS/DDoS) Attacks

In a denial of service (DoS) attack, the **cybercriminal** uses a huge number of fake requests to overwhelm the target service—in this case, the database server. As a result, the server cannot handle legitimate requests from actual users and frequently crashes or becomes unstable.

In a distributed denial of service **(DDoS)** attack, fake traffic is generated by a large number of computers that are part of an attacker-controlled botnet. This results in extremely high traffic volumes, which are difficult to stop without a highly scalable defensive architecture.

5) Exploitation of Database Software Vulnerabilities
Attackers are continuously attempting to isolate and target software vulnerabilities, and database management software is a particularly desirable target. New **vulnerabilities** are identified on daily basis,. However, if you do not apply these changes immediately, your database may be vulnerable to attack.

Control Measures for the Security of Data in Databases

The following are the key control measures used to ensure data security in databases:

Authentication

- Authentication is the process of confirming whether a user logs in only with the rights granted to him to undertake database operations. A certain user can only log in up to his privilege level, but he cannot access any other sensitive data.

Access Control

- Database access control is a means of restricting access to sensitive company data to only those people **(database users)** who are authorized to access such data and permitting access to unauthorized persons. It is a key security concept that reduces risk to the business or organization.

Encryption

- Data encryption protects data confidentiality by converting it to encoded information known as cipher text, which can only be decoded with a unique decryption key generated either during or before encryption.

Data Protection Tools and Platforms

Several companies now offer data protection platforms and tools.

1. Acronis 2. CybeReady 3. Barracuda 4. Cohesity 5. Eperi

NORMALIZATION (V.IMP)

Normalization is the process of organizing data in a database. It includes creating tables and establishing relationships between those tables according to rules designed both to protect the data and to make the database more flexible by eliminating data redundancy and inconsistent dependency.

Redundant data wastes disk space and creates maintenance problems. If data that exists in more than one place must be changed, the data must be changed in exactly the same way in all locations. A customer address change is easier to implement if that data is stored only in the Customers table and nowhere else in the database.

What is an "inconsistent dependency"? While it's intuitive for a user to look in the Customers table for the address of a particular customer, it may not make sense to look there for the salary of the employee who calls on that customer. The employee's salary is related to, or dependent on, the employee and thus should be moved to the Employees table. Inconsistent dependencies can make data difficult to access because the path to find the data may be missing or broken.

There are a few rules for database normalization. Each rule is called a "normal form."

Problems without Normalization in DBMS

If a table is not properly normalized and has data redundancy (repetition) then it will not only **eat up extra memory space** but will also make it difficult for you to handle and update the data in the database, without losing data.

Insertion, Updation, and Deletion Anomalies are very frequent if the database is not normalized.

To understand these anomalies let us take an example of a **Student** table.

RollNo	**name**	**branch**	**hod**	**office_tel**
401	Akon	CSE	Mr. X	53337
402	Bkon	CSE	Mr. X	53337
403	Ckon	CSE	Mr. X	53337
404	Dkon	CSE	Mr. X	53337

In the table above, we have *data for four Computer Sci. students*.

As we can see, data for the fields **branch**, **hod**(Head of Department), and **office_tel** are repeated for the students who are in the same branch in the college, this is **Data Redundancy**.

1. Insertion Anomaly in DBMS

- Suppose for a new admission, until and unless a student opts for a branch, data of the student cannot be inserted, or else we will have to set the branch information as **NULL**.

- Also, if we have to insert data for 100 students of the same branch, then the branch information will be repeated for all those 100 students.
- These scenarios are nothing but **Insertion anomalies**.
- If you have to repeat the same data in every row of data, it's better to **keep the data separately** and **reference that data** in each row.
- So in the above table, we can keep the branch information separately, and just use the **branch_id** in the student table, where **branch_id** can be used to get the branch information.

2. Updation Anomaly in DBMS

- What if Mr. X leaves the college? Or Mr. X is no longer the HOD of the computer science department? In that case, all the student records will have to be updated, and if by mistake we miss any record, it will lead to data inconsistency.
- This is an Updation anomaly because you need to update all the records in your table just because one piece of information got changed.

3. Deletion Anomaly in DBMS

- In our **Student** table, two different pieces of information are kept together, the **Student information** and the **Branch information**.
- So if only a single student is enrolled in a branch, and that student leaves the college, or for some reason, the entry for the student is deleted, we will lose the branch information too.
- So never in DBMS, we should keep two different entities together, which in the above example is Student and branch,

The solution for all the three anomalies described above is to keep the **student** information and the **branch** information in two different tables. And use the **branch_id** in the student table to reference the branch.

Advantages of Normal Form

- **Reduced data redundancy:** Normalization helps to eliminate duplicate data in tables, reducing the amount of storage space needed and improving database efficiency.
- **Improved data consistency:** Normalization ensures that data is stored in a consistent and organized manner, reducing the risk of data inconsistencies and errors.
- **Simplified database design:** Normalization provides guidelines for organizing tables and data relationships, making it easier to design and maintain a database.
- **Improved query performance:** Normalized tables are typically easier to search and retrieve data from, resulting in faster query performance.
- **Easier database maintenance:** Normalization reduces the complexity of a database by breaking it down into smaller, more manageable tables, making it easier to add, modify, and delete data.

1. First Normal Form (1NF)

For a table to be in the First Normal Form, it should follow the following 4 rules:

1. It should only have single (**atomic**) valued attributes/columns.

2. Values stored in a column should be of the same domain.
3. All the columns in a table should have unique names.
4. And the order in which data is stored should not matter.

   Let's see an example.
   If we have an **Employee** table in which we store the *employee information* along with the *employee skillset*, the table will look like this:

emp_id	emp_name	emp_mobile	emp_skills
1	John Tick	9999957773	Python, JavaScript
2	Darth Trader	8888853337	HTML, CSS, JavaScript
3	Rony Shark	7777720008	Java, Linux, C++

The above table has 4 columns:

- All the columns have different names.
- All the columns hold values of the same type like **emp_name** has all the names, **emp_mobile** has all the contact numbers, etc.
- The order in which we save data doesn't matter
- But the **emp_skills** column holds *multiple comma-separated values*, while as per the First Normal form, each column should have a single value.

So how do you fix the above table? There are two ways to do this:

1. Remove the **emp_skills** column from the **Employee** table and keep it in some other table.
2. Or add multiple rows for the employee and each row is linked with one skill.

1. Create Separate tables for Employee and Employee Skills
So the **Employee** table will look like this,

emp_id	emp_name	emp_mobile
1	John Tick	9999957773
2	Darth Trader	8888853337
3	Rony Shark	7777720008

And the new **Employee_Skill** table:

emp_id	emp_skill
1	Python
1	JavaScript
2	HTML
2	CSS
2	JavaScript
3	Java

emp_id	emp_skill
3	Linux
3	C++

Add multiple rows for multiple skills
You can also *simply add multiple rows* to add multiple skills. This will lead to repetition of the data, but that can be handled as you further Normalize your data using the Second Normal form and the Third Normal form.

emp_id	emp_name	emp_mobile	emp_skill
1	John Tick	9999957773	Python
1	John Tick	9999957773	JavaScript
2	Darth Trader	8888853337	HTML
2	Darth Trader	8888853337	CSS
2	Darth Trader	8888853337	JavaScript
3	Rony Shark	7777720008	Java

emp_id	emp_name	emp_mobile	emp_skill
3	Rony Shark	7777720008	Linux
3	Rony Shark	7777720008	C++

2. Second Normal Form (2NF)

For a table to be in the Second Normal Form,

1. It should be in the First Normal form.
2. And, it should not have **Partial Dependency**.

What is Partial Dependency?
Partial Dependency occurs when a nonprime attribute is functionally dependent on part of a candidate key.

Let us take an example of the following <EmployeeProjectDetail> table to understand what partial dependency is and how to normalize the table to the second normal form:

<EmployeeProjectDetail>

Employee_ID	Project_ID	Employee_Name	Project_Name
101	P03	John	Project103
101	P01	John	Project101
102	P04	Ryan	Project104
103	P02	Stephanie	Project102

In the above table, the prime attributes of the table are **Employee_ID** and **Project ID**. We have partial dependencies in this table because **Employee_Name** can be determined by **Employee_ID** and **Project_Name** can be determined by **Project_ID**. Thus, the above relational table violates the rule of 2NF.

The prime attributes in DBMS are those which are part of one or more candidate keys.To remove partial dependencies from this table and normalize it into second normal form, we can decompose the <EmployeeProjectDetail> table into the following three tables:

<EmployeeDetail>

Employee_ID	Employee _Name
101	John
102	Ryan

Employee_ID	Employee _Name
103	Stephanie

<EmployeeProject>

Employee_ID	Project_ID
101	P03
101	P01
102	P04
103	P02

<ProjectDetail>

Project_ID	Project_Name
P01	Project101
P02	Project102
P03	Project103
P04	Project104

Thus, we've converted the <EmployeeProjectDetail> table into 2NF by decomposing it into <EmployeeDetail>, <ProjectDetail> and <EmployeeProject> tables. As you can see, the above tables satisfy the following two rules of 2NF as they are in 1NF and every non-prime attribute is fully dependent on the primary key.

The relations in 2NF are clearly less redundant than relations in 1NF.

Third Normal Form

A table is said to be in the Third Normal Form when,

1. It satisfies the Second Normal form.
2. And, it doesn't have Transitive Dependency.

What is Transitive Dependency?

A transitive dependency exists when a non-primary column depends on some other non-primary column. There exists an indirect relationship between columns of the same table.

We assume the **Score** table is already in the Second Normal Form. If we have to store some extra information in it, like,

1. **exam_type**
2. **total_marks**

The **Score** table will look like this,

student_id	**subject_id**	**marks**	**exam_type**	**total_marks**
1	1	70	Theory	100
1	2	82	Theory	100
2	1	42	Practical	50

- In the table above, the column **exam_type** depends on both **student_id** and **subject_id**,

  But the column **total_marks** just depends on the **exam_type** column. And the **exam_type** column is not a part of the primary key. Because the primary key is **student_id + subject_id**, hence we have a Transitive dependency here.

How to Transitive Dependency?
You can create a separate table for **ExamType** and use it in the **Score** table.

New **ExamType** table,

exam_type_id	**exam_type**	**total_marks**	**Duration**
1	Practical	50	45
2	Theory	100	180
3	Workshop	150	300

We have created a new table **Exam Type** and we have added more related information in it like **duration** (duration of exam in mins.), and now we can use the **exam_type_id** in the **Score** table.

Example :

EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Candidate key: {EMP_ID}

Non-prime attributes: In the given table, all attributes except EMP_ID are non-prime.

That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago

06389	UK	Norwich
462007	MP	Bhopal