Introduction to SQL

# DDL = Data Definition Language:

Data Definition Language (DDL) is a subset of SQL (Structured Query Language) commands used to define and modify the database schema rather than to manipulate data. In essence, DDL focuses on the structure and the framework of the database itself. This includes creating, altering, and deleting databases and their objects such as tables, indexes, views, and stored procedures.

The key DDL commands include:

* **CREATE:** This command is used to create new databases, tables, views, indexes, etc. For example, CREATE TABLE is used to create a new table in the database.
* **ALTER:** This command is used to modify the structure of an existing database object. For example, you can use ALTER TABLE to add, delete, or modify columns in an existing table.
* **DROP:** This command is used to delete objects from the database. For example, DROP TABLE is used to delete a table and all of its data.
* **TRUNCATE:** This command is used to delete all records from a table, but it does not delete the table itself. It is a faster way to clear out records from a table because it does not log the deletion of each row and does not fire triggers.
* **COMMENT:** This command is used to add comments to the dictionary for the purpose of documentation.
* **RENAME:** This command is used to rename an existing database object, such as a table.

The execution of DDL commands can significantly alter the database structure, and in most database systems, these changes are automatically committed, meaning they cannot be rolled back. Therefore, it's crucial to be cautious when issuing DDL commands to avoid unintended loss of data or structure.

DDL is distinct from Data Manipulation Language (DML), which is used to query and modify data within the tables, and Data Control Language (DCL), which is used for permissions and access control.

# DML = Data Manipulation Language

Data Manipulation Language (DML) is a subset of SQL (Structured Query Language) used for managing data within database objects such as tables. Unlike Data Definition Language (DDL), which is used to define and modify database schema, DML focuses on performing operations on the data itself, such as inserting, updating, deleting, and retrieving data from tables.

The key DML commands include:

* **SELECT:** This command is used to retrieve data from one or more tables. It is the most widely used DML command and can be highly customized with conditions, grouping, and ordering of the retrieved data.
* **INSERT:** This command is used to add new rows (records) to a table. You can insert values into specific columns or into all columns of a table.
* **UPDATE:** This command is used to modify existing data in a table. It typically involves specifying a condition to identify the rows that need to be updated and the new values to be applied.
* **DELETE:** This command is used to remove rows from a table based on a condition. Without a condition, it can potentially remove all rows from a table, effectively emptying it.

DML operations are fundamentally about manipulating the data stored within the database structures defined by DDL. These operations can be transactional, meaning that changes can be committed (made permanent) or rolled back (undone) if not explicitly committed, depending on the database system's transaction management.

In addition to these basic commands, SQL also supports more advanced DML operations through constructs like JOINs (to combine rows from two or more tables based on a related column between them), subqueries, and set operations (such as UNION, INTERSECT, and EXCEPT), which allow for complex data retrieval and manipulation.

It's important to note that while DML commands modify the data within the database, they do not alter the structure or schema of the database tables themselves that is the role of DDL commands.

An important website for compile SQL code/schema: <https://sqlfiddle.com/>

## Is SQL case sensitive?

SQL's case sensitivity depends on two main aspects: the SQL dialect being used (e.g., MySQL, PostgreSQL, SQL Server, Oracle) and the context in which the SQL is being executed (identifiers vs. data vs. keywords).

* SQL Keywords: Generally, SQL keywords (like SELECT, INSERT, UPDATE, DELETE, FROM, WHERE, etc.) are not case-sensitive. This means you can write your SQL statements with keywords in uppercase, lowercase, or mixed case, and they will function the same. For readability and convention, SQL keywords are often written in uppercase.
* Identifiers (Table and Column Names): The case sensitivity of identifiers, such as table names and column names, depends on the database system's configuration and the operating system of the database server. For example:
* In MySQL, table names are case-sensitive on Unix-like systems but not case-sensitive on Windows. Column names are not case-sensitive in MySQL queries.
* SQL Server treats identifiers as case-insensitive, regardless of the case used when the identifier was created, by default. However, this behavior can be altered by changing the collation of the database or column.
* PostgreSQL treats identifiers (table names, column names, etc.) as case-insensitive by default, unless they are double-quoted. If double-quoted, PostgreSQL preserves the case, making the identifiers case-sensitive.
* Data: The case sensitivity of data in SQL queries, particularly when comparing string values, depends on the collation settings of the database or the specific column. Collation defines how data is sorted and compared in a database, including case sensitivity. Some collations are case-sensitive, while others are not. This means that a WHERE clause like WHERE name = 'john' could return different results based on the collation settings of the name column.

In summary, while SQL keywords and syntax are generally not case-sensitive, the case sensitivity of identifiers and data can vary based on the database system, its configuration, and the underlying operating system. It's important to consult the documentation of the specific SQL database you are working with to understand how it handles case sensitivity.

# What is a Database?

A database is an organized collection of data that can be easily accessed, managed, and updated. Databases are crucial for storing and retrieving data efficiently, and they support a wide range of applications, from websites and mobile apps to banking systems and healthcare records. The primary purpose of a database is to store data in a structured way, providing means for data manipulation and retrieval through queries.

Databases can be classified into several types based on their structure and the model they use to manage data:

* Relational Databases (RDBMS): These use a table-based structure, where data is stored in rows and columns. Each table, which represents a different entity or object, can be linked to others through relationships, facilitating complex queries and data integrity. Examples include MySQL, PostgreSQL, Oracle, and SQL Server.
* NoSQL Databases: A broad category that encompasses a variety of models that do not fit the traditional relational database schema. NoSQL databases are designed for specific data models and have flexible schemas for building modern applications. They include document databases (e.g., MongoDB), key-value stores (e.g., Redis), wide-column stores (e.g., Cassandra), and graph databases (e.g., Neo4j).
* In-Memory Databases: These databases reside entirely in the main memory (RAM) rather than on disk, offering faster data retrieval times compared to disk-based databases. They are used for applications requiring rapid access to data, such as caching layers or real-time analytics (e.g., Redis, Memcached).
* Distributed Databases: These are spread across multiple physical locations, either spread across different networks or regions. They are designed to provide high availability, scalability, and reliability by distributing data and load across multiple servers (e.g., Cassandra, CockroachDB).
* Object-oriented Databases: These store data in the form of objects, as used in object-oriented programming. Data is stored and retrieved through the same structures used in programming, making them a good fit for applications written in object-oriented programming languages (e.g., db4o, ObjectDB).
* Graph Databases: These store data in the form of nodes and edges, representing entities and the relationships between them, respectively. They are optimized for handling complex queries that involve traversing relationships (e.g., Neo4j, Amazon Neptune).

Databases are managed through Database Management Systems (DBMS), which provide the tools and interfaces necessary for users and applications to interact with the database. The choice of database and DBMS depends on the specific requirements of an application, including the nature of the data, the expected scale, the need for speed and efficiency, and the complexity of the queries.

# The relational database principles

Relational databases are based on the relational model, an approach to managing data that emphasizes the logical representation of data separate from its storage. This model, introduced by E.F. Codd in 1970, organizes data into tables (relations) consisting of rows and columns. The principles of relational databases are foundational to understanding how they store, organize, and manipulate data efficiently. Here are the key principles:

* **Structure:** Data is organized into tables (also called relations), which represent entities or objects. Each table is made up of rows (records) and columns (attributes or fields). Each column in a table is designed to hold a specific type of data, such as numbers, dates, or strings.
* **Data Integrity:** Relational databases enforce data integrity through rules that ensure the accuracy and consistency of the data. These include:
  + **Primary keys:** A unique identifier for each row in a table, ensuring that no two rows have the same value in this column.
  + **Foreign keys:** A field (or collection of fields) in one table that uniquely identifies a row of another table, creating a relationship between the two tables.
  + **Constraints:** Rules applied to columns to ensure that only valid data is added. For example, constraints can enforce data uniqueness, check conditions, or establish relationships between tables.
* **Data Manipulation:** SQL (Structured Query Language) is used for data manipulation, allowing users to perform tasks such as querying, inserting, updating, and deleting data. SQL commands are divided into Data Manipulation Language (DML) for handling data and Data Definition Language (DDL) for defining database schemas.
* **Normalization:** A process of organizing data in a database to reduce redundancy and improve data integrity. Normalization involves dividing large tables into smaller, interrelated tables and defining relationships between them. This process helps to minimize duplication of information and to ensure that data dependencies are logical.
* **Atomic Transactions:** Relational databases support transactions, which are sequences of operations performed as a single logical unit of work. A transaction must be entirely completed or entirely failed, ensuring the database remains in a consistent state. This principle is supported by the ACID properties (Atomicity, Consistency, Isolation, Durability), which guarantee that transactions are processed reliably.
* **Relational Operations:** The model supports a set of operations that can be applied to tables, enabling the manipulation and retrieval of data. These include basic operations like SELECT (to query data), INSERT (to add data), UPDATE (to modify data), and DELETE (to remove data), as well as set operations like UNION, INTERSECT, and MINUS, and JOIN operations to combine rows from two or more tables.
* **Independence:** Data abstraction allows changes in the schema (the logical view of the database) without affecting the data access layer. This principle is known as data independence, which can be physical (changes in how the data is stored do not affect the schema) or logical (changes in the schema do not affect the application layer).

These principles form the foundation of relational database design and operation, enabling them to support a wide range of applications with complex data models and the need for high integrity and availability.

# Entity Relationship Diagram

An Entity-Relationship (ER) diagram is a graphical representation of entities and their relationships to each other, typically used in database design. An ER diagram helps in structuring and organizing data requirements before creating a database or making modifications to an existing database. It's a fundamental part of the database modeling process, providing a clear and concise way to capture data interactions and relationships within a system.

Key components of ER diagrams include:

* **Entities:** These are objects or concepts that can have data stored about them. An entity represents a table in a database. Entities are depicted as rectangles in ER diagrams.
* **Attributes:** These are the data we want to collect for an entity. Attributes represent the columns or fields of a table. They are shown as ovals connected to their respective entities.
* **Relationships:** These illustrate how entities share information in the database. Relationships are depicted as diamonds, connected by lines to the entities they relate. Relationships can have cardinality that defines the nature of the relationship in terms of numbers (one-to-one, one-to-many, many-to-many).
* **Primary Keys and Foreign Keys:** In ER diagrams, primary keys are underlined to denote a unique identifier for an entity's instances. Foreign keys, when present, indicate how entities are related and ensure referential integrity between relationships.
* **Cardinality and Modality:** Cardinality refers to the number of instances of one entity connected to one instance of another entity. Modality (or optionality) indicates the necessity of the relationship, showing whether or not an entity instance must participate in the relationship.

ER diagrams can range from simple designs illustrating a few entities and their relationships, to complex diagrams depicting an entire system's data model with numerous entities, attributes, and intricate relationships. They serve as a blueprint for developing relational databases and are essential for understanding the system's structure, ensuring efficient data organization, and facilitating communication among stakeholders during the database design process.

# What is a Relational Database Management System (RDBMS)?

A Relational Database Management System (RDBMS) is a type of database management system (DBMS) that stores data in a structured format, using rows and columns in tables to make data management and querying efficient. The relational model, upon which these systems are based, uses a strong, formal structure and standard querying languages like SQL (Structured Query Language) for database access and manipulation. This model was introduced by E.F. Codd in 1970.

Key Features of an RDBMS:

* **Tables (Relations):** Data is organized into tables, which represent entities. Each table consists of rows (records) and columns (attributes or fields).
* **Data Integrity:** RDBMSs enforce data integrity rules, ensuring the accuracy and consistency of the data. This includes mechanisms like primary keys (unique identifier for each record in a table), foreign keys (identifiers that establish a relationship between tables), and various constraints (e.g., NOT NULL, UNIQUE, CHECK).
* **SQL Support:** RDBMSs use SQL for data manipulation and querying, allowing users to perform tasks such as inserting, querying, updating, and deleting data (CRUD operations).
* **Transactions**: Support for transactions allows multiple database operations to be executed as a single unit of work, ensuring data integrity. Transactions follow the ACID properties (Atomicity, Consistency, Isolation, Durability) to ensure that all operations within a transaction are completed successfully before committing to the database.
* **Access Control:** RDBMSs provide comprehensive access control mechanisms, enabling database administrators to define permissions and access levels for different users and groups, ensuring data security.
* **Normalization:** RDBMSs support the normalization process, which organizes data to reduce redundancy and dependency by dividing large tables into smaller tables and defining relationships between them.
* **Backup and Recovery:** RDBMSs typically include robust tools for data backup and recovery, ensuring data can be restored in case of corruption or loss.
* **Scalability and Performance:** Advanced RDBMSs are designed to efficiently handle large volumes of data and high transaction rates, providing mechanisms for optimization, indexing, and partitioning to enhance performance and scalability.

Popular RDBMS Examples:

* Oracle Database: Known for its feature-rich environment, scalability, and enterprise-level capabilities.
* MySQL: Widely used for web applications, known for its ease of use and open-source licensing.
* Microsoft SQL Server: Popular in enterprise environments, offering a broad range of tools and features for data management.
* PostgreSQL: An open-source RDBMS known for its advanced features, extensibility, and compliance with SQL standards.
* IBM Db2: Used for high-volume transactions and analytics, offering robust data management and optimization features.

RDBMSs are foundational to modern data management and are used across industries for a wide variety of applications, from simple web applications to complex transaction processing systems. Their standardized use of SQL, adherence to the ACID properties for transactions, and support for complex queries and data integrity rules make them a critical component of data infrastructure in organizations around the world.

# MySQL – ACID Compliance

ACID compliance refers to a set of properties—Atomicity, Consistency, Isolation, Durability—that ensure reliability in the processing of database transactions. These properties are crucial for ensuring data integrity and error handling in database management systems, particularly in systems where the correctness of transactions is critical, such as financial systems, e-commerce platforms, and more. Let's delve into each of the ACID properties in detail:

1. Atomicity

Definition: Atomicity guarantees that each transaction is treated as a single, indivisible unit, which means it either completes in its entirety or does not execute at all. If any part of the transaction fails, the entire transaction fails, and the database state is left unchanged.

Example: Consider a bank transfer from Account A to Account B. The transaction involves debiting an amount from Account A and crediting the same amount to Account B. Atomicity ensures that both these operations must succeed for the transaction to be considered successful. If either operation fails, any changes made are rolled back, leaving the accounts as they were before the transaction began.

2. Consistency

Definition: Consistency ensures that a transaction can only bring the database from one valid state to another, maintaining all the predefined rules, including integrity constraints and triggers. This means that any data written to the database must be valid according to all defined rules; otherwise, the transaction is rolled back.

Example: If a database has a rule that the balance of a bank account cannot be negative, a withdrawal transaction must check that the withdrawal amount does not exceed the account's current balance. If the rule is violated, the transaction is aborted, thus maintaining consistency.

3. Isolation

Definition: Isolation ensures that concurrent transactions occur separately from one another, preventing them from interfering with each other. It ensures that the intermediate state of a transaction is invisible to others and that transactions are not affected by each other.

Example: If two bank customers are simultaneously transferring money from their accounts to a third account, isolation ensures that each transaction sees a consistent view of the third account’s balance, preventing them from being aware of each other's interim operations, which might otherwise lead to data inconsistencies.

4. Durability

Definition: Durability guarantees that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors. This property ensures that the completed transactions are permanently recorded in the database's storage system.

Example: After completing a sales transaction, durability ensures that the record of that sale is saved to a permanent storage device, making the transaction recoverable in the event of a system crash or restart.

ACID compliance is a cornerstone of relational database systems, ensuring that transactions are processed reliably and that the database integrity is maintained, even in the face of system failures or attempts to concurrently access and modify the same data. This reliability is critical for applications requiring a high degree of data integrity and consistency.

# Installation

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente