SQL
MISCELLANEOUS

CLASS OUTLINE

- Database Objects
- Views
- Index
- Stored Procedure
- Triggers
- Cardinality
- Data Redundancy
- Normalization
- ACID Model
- BASE Model

DATABASE OBJECTS

- In SQL, a database object is a **container** (structure) that stores data or metadata related to the database.
- These objects can be **used to store, retrieve, manipulate, and organize data** within the database.

Some common database objects in SQL include:

1. **Tables:** Tables are the most common database objects in SQL. They store data in rows and columns, and can be used to organize and manage large amounts of data.
2. **Views:** Views are virtual tables that are created based on the data from one or more tables in the database. They allow users to query the data without having to access the underlying tables directly.
3. **Indexes:** Indexes are database objects that improve query performance by allowing the database to quickly locate the rows that match a specific value or set of values in one or more columns of a table.
4. **Constraints:** Constraints are rules that are defined on a table to enforce data integrity and consistency.

DATABASE OBJECTS...

5. **Stored Procedures:** Stored procedures are pre-compiled SQL statements that are stored in the database and can be executed by applications or other database objects.
6. **Triggers:** Triggers are database objects that automatically execute SQL statements in response to specific events or actions, such as an update or delete operation on a table.
7. **Functions:** Functions are database objects that return a value based on the input parameters provided. They can be used to perform calculations, manipulate data, or return specific values.

VIEWS

- In SQL, a view is a virtual table that is derived from one or more tables or other views in the database.
- A view **does not store data itself** but rather provides a way to present data from one or more tables in a specific format.
- Once created, the view can be queried like a regular table.

SYNTAX:

```
CREATE VIEW view_name AS
SELECT column_list (s)
FROM table_name
```

VIEWS…

Some benefits of using views in SQL include:

1. **Security:** Views can be used to restrict access to sensitive data by limiting the columns or rows that are visible to certain users or roles.
2. **Simplification:** Views can simplify complex queries by pre-defining joins and aggregations, making it easier for users to query the data without having to write complex SQL statements themselves.
3. **Performance:** Views can improve query performance by allowing the database to cache commonly used queries and avoid the overhead of joining tables on each query execution.

One potential drawback of using views is that they can impact database performance, especially if the underlying tables are large and complex.

INDEX…

Index is a way of organizing data in a table to make searching for specific values more efficient.

In **SQL**, an index is a **data structure** that enables the DBMS to locate data efficiently without scanning the entire table.

*Some **benefits** of using Indexes in SQL include:*

1. **Speed-up searches:** Indexes can be used to speed up searches **by providing quick access to specific rows** in a table. Without an index, the DBMS would need to scan the entire table to find the data that matches the search criteria.
2. **Improve query performance:** Indexes can also improve the performance of queries by faster query execution times.
3. **Enforce uniqueness:** Indexes can be used to enforce uniqueness constraints on a table. This ensures that no two rows have the same value for a particular column or set of columns.
4. **Optimize data access:** Indexes can be used to optimize data access by organizing the data in a way that reflects the most common query patterns. For example, if a table is frequently queried based on a particular column, creating an index on that column can improve performance.

INDEX…

It's important to note that while indexes can improve query performance, they can also have downsides.

- An index is created on one or more columns in a table, and **it contains a copy of the data** in those columns, along with a **pointer** to the corresponding row in the table.
  - Therefore, Indexes can consume a significant amount of disk space, and they can slow down data modifications (such as inserts, updates, and deletes) since the DBMS must update the index as well as the table.
- It is important to carefully consider the use of indexes and to **only** create indexes where they are truly necessary.

KEY VS INDEX...

In SQL, a key and an index are related concepts but they serve different purposes. Keys and an index both are used to enforce data integrity and improve query performance, but they have different roles and functions as well.

KEY

- A key is a **constraint** that **enforces uniqueness and data integrity** in a table.
- A **primary key** is a **unique identifier** for each row in a table and is used to enforce **data integrity** by ensuring that **each row is unique**.
- A **foreign key** is a field in one table that refers to the primary key in another table, and it is used to enforce **referential integrity**.

INDEX

- An index is a **data structure** that is **used to optimize query performance** by allowing the DBMS to find specific data quickly without scanning the entire table.
- When a query is executed that involves the indexed columns, the DBMS can use the index to quickly locate the relevant rows, rather than scanning the entire table.

KEY VS INDEX

"A key is used to enforce uniqueness and data integrity, while an index is used to optimize query performance."

"A key can be used as the basis for an index, but an index cannot be used as a key."

Reason:

- A key can be used as the basis for an index **because it uniquely identifies a record** (column values are unique) and can therefore be used to quickly locate that record.
- An index cannot be used as a key because **it does not uniquely identify a record** (column values can be the same). An index is simply a data structure that **points to the location of data on a disk.**

STORED PROCEDURE

- Stored procedures in SQL are a **set of pre-written SQL statements** that are **compiled** and **stored** on a database server.
- These pre-written SQL statements can be executed as a single unit, without having to write the SQL code again and again.
- The stored procedure can be called by other programs or SQL statements.
- Stored procedures are typically used for complex database operations that involve multiple SQL statements, as they provide a way to encapsulate complex database logic in a single unit.
- It can be used for a variety of purposes, including data manipulation, data validation, and business logic implementation.

STORED PROCEDURE…

SYNTAX: CREATING PROCEDURE	EXAMPLE:
DELIMITER // CREATE PROCEDURE procedure_name() BEGIN SELECT col_name FROM table_name ; END // DELIMITER ;	DELIMITER // CREATE PROCEDURE GetCustomers() BEGIN SELECT Customer_name FROM Customer_table ; END // DELIMITER ;
CALL procedure_name() ;	CALL GetCustomers();

TRIGGERS

- In a general sense, a trigger refers to something that causes or initiates a particular action or response.
- In SQL (Structured Query Language), a trigger is **a special type of stored procedure** that is **automatically executed** in response to certain database events or conditions, such as an **insert**, **update**, or **delete operation** on a table.
- Triggers can be used to enforce data integrity, implement business rules, or perform custom data processing tasks.
- For example, a trigger can be created to ensure that every new record inserted into a table meets certain criteria or to update a related table when a record is deleted.

Note: that triggers can be powerful tools, but they should be used with caution to avoid unintended consequences or performance issues.

TRIGGERS

SYNTAX:	EXAMPLE:
CREATE TRIGGER trigger_name AFTER INSERT ON table_1 FOR EACH ROW **UPDATE** table_2 **SET** column = column - 1 **WHERE** key_column= NEW.key_column;	CREATE TRIGGER inventory_update AFTER INSERT ON order_table FOR EACH ROW **UPDATE** inventory **SET** units = units - 1 **WHERE** Product_id= NEW. Product_id;

Note: Syntax of Trigger can vary depending on case by case.

CARDINALITY

In SQL, cardinality refers to the number of rows in a table that meets a certain condition or set of conditions. It is **used to describe the relationship between two tables** in a database.

There are three types of cardinality in SQL:

1. **One-to-One (1:1) Cardinality:** In this type of relationship, each row in one table is related to **only one** row in another table, and vice versa.

For example, in a library database, each book might have only one corresponding ISBN. In this case, each book is unique and can be identified by its unique ISBN number.

2. **One-to-Many (1:N) Cardinality:** In this type of relationship, each row in one table can be **related to multiple rows** in another table, but each row in the second table is related to only one row in the first table.

For example, a table of customers may be related to a table of orders, where each customer can have multiple orders, but each order is associated with only one customer.

CARDINALITY

1. **Many-to-Many (N:M) Cardinality:** In this type of relationship, each row in one table can be related to multiple rows in another table, and vice versa.

For example, a table of students may be related to a table of courses, where each student can be enrolled in multiple courses and each course can have multiple students.

Cardinality is an important concept in database design and helps to ensure data integrity and consistency. It is used to define the relationships between tables and to ensure that data is stored in a way that supports efficient querying and analysis.

DATA REDUNDANCY

- Data redundancy refers to the **duplication of data** within a database or a system.
- This occurs when the same information is stored multiple times in different locations or tables within a database.
- Redundancy can lead to a number of issues such as **data inconsistencies**, **increased storage requirements, slower query times**, and **difficulty in maintaining data integrity**.
- Data redundancy can occur for a variety of reasons, such as **poor database design, lack of normalization**, or inefficient data management practices.
- By reducing redundancy, databases can operate more efficiently, ensure data integrity, and improve overall system performance.

NORMALIZATION

- The process typically involves **breaking down a larger table into smaller, more manageable tables,** each with a unique purpose.
- The **primary goal is to reduce data duplication**, which can cause inconsistencies and make data maintenance more difficult.
- In normalization, the tables are arranged in a specific hierarchy based on their level of dependency.
- Normalization in SQL involves following a set of rules, known as **normal forms**, to organize data in a database and reduce data redundancy.
- The higher the normal form, the more normalized the database is, which reduces redundancy and improves data integrity. However, higher normal forms can also result in more complex queries and slower database performance, so it is important to find the appropriate level of normalization for each specific application.

NORMALIZATION

The most commonly used normal forms are:

1. **First Normal Form (1NF):** This rule states that each table should have a primary key and each column should contain atomic values. Atomic values are indivisible and cannot be further decomposed.
2. **Second Normal Form (2NF):** This rule states that each non-key attribute of a table should be dependent on the table's primary key. This means that each column in a table should contain information about a single entity.
3. **Third Normal Form (3NF):** This rule states that each non-key attribute of a table should be independent of the other non-key attributes. This means that the data in each column of a table should not be dependent on other columns.

Some other higher-up normal forms are **Boyce-Codd Normal Form (BCNF), Fourth Normal Form (4NF), Fifth Normal Form (5NF)** etc.

ACID MODEL

The ACID Model is a **set of database design principles**. By using the **ACID properties**, MySQL ensures that **database transactions are processed reliably and consistently**, even in the presence of errors or failures.

The acronym ACID stands for:

- **Atomicity:** This property ensures that transactions are indivisible and are processed as a single unit of work. Either all operations within the transaction are completed successfully or none of them are.
- **Consistency:** This property ensures that after a transaction is completed, the database is in a valid state. The transaction should not violate any of the constraints defined on the database schema.
- **Isolation:** This property ensures that concurrent transactions do not interfere with each other. Each transaction should be executed independently as if it is the only transaction running on the database.
- **Durability:** This property ensures that once a transaction is committed, its effects will persist even in the event of a system failure, such as a power outage or a system crash.

ACID MODEL

Advantages of ACID Properties in DBMS:

1. **Data Consistency:** These design principles ensure that the data remains consistent and accurate after any transaction execution.
2. **Data Integrity:** ACID properties maintain the integrity of the data by ensuring that any changes to the database are permanent and cannot be lost.
3. **Concurrency Control:** These properties help to manage multiple transactions occurring concurrently by preventing interference between them.
4. **Recovery:** These set of rules ensure that in case of any failure or crash, the system can recover the data up to the point of failure or crash.

ACID MODEL

Disadvantages of ACID Properties in DBMS:

1. **Performance:** The ACID properties can cause a performance overhead in the system, as they require additional processing to ensure data consistency and integrity.
2. **Scalability:** These properties may cause scalability issues in large distributed systems where multiple transactions occur concurrently.
3. **Complexity:** Implementing the ACID properties can increase the complexity of the system and require significant expertise and resources.

Overall, the advantages of ACID properties in DBMS outweigh the disadvantages. They provide a reliable and consistent approach to data

BASE MODEL

The BASE properties are a more relaxed version of ACID that **trade off some consistency guarantees for greater scalability and availability.**

The acronym BASE stands for:

1. **Basically Available:** In a BASE system, availability is prioritized over strict consistency. This means that the system is designed to always be available, even if it means returning stale or partially updated data.
2. **Soft state:** In a BASE system, the state of the database may be soft or indeterminate at times. This means that the data may be changing rapidly, and updates may not always be immediately visible to all users.
3. **Eventually Consistent:** In a BASE system, the consistency of the data is eventually achieved, but not immediately. This means that updates may take some time to propagate through the system and become visible to all users.

THANK YOU