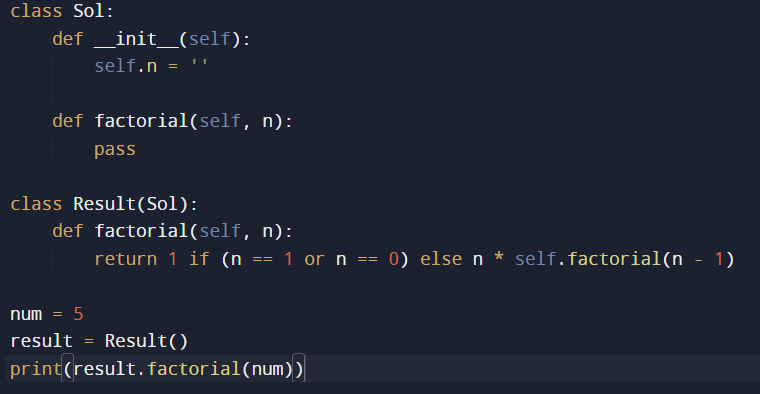
# ***Theory***

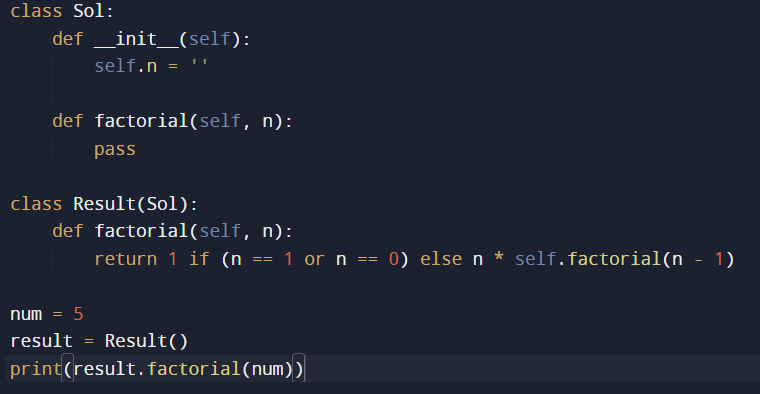
## *1. OOP - Abstraction.*

A fundamental concept in object-oriented programming (OOP) that involves hiding the complex implementation details of a class and exposing only the necessary and relevant parts. It allows a programmer to focus on what an object does instead of how it does it.



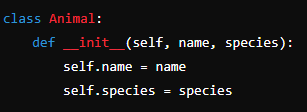
Interfaces:

An interface is like a promise made by a class. It says, "I will have these methods, but I won't tell you exactly how they work." This allows different classes to agree on a common set of behaviors without worrying about how those behaviors are implemented. Interfaces promote sharing code between different classes, making it easier to reuse and allowing objects to be treated interchangeably based on their common features.



Constructor:

* A special method in Python classes that is automatically called when an object is instantiated.
* It initializes the object's attributes and can take parameters to set the initial state of the object.
* The constructor method is defined using the \_\_init\_\_ keyword.

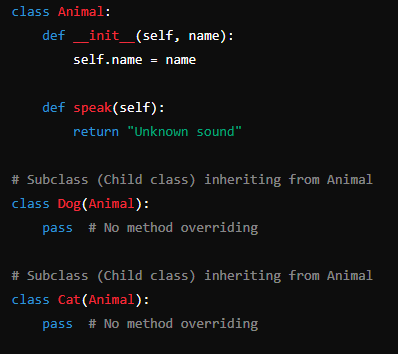


## *2. OOP - Inheritance.*

A fundamental concept in object-oriented programming (OOP) where a new class (subclass) is created by inheriting the properties and behaviors of an existing class (superclass).

Basic Terms:

* Parent Class (Superclass): This is indeed the superclass, which is the class whose attributes and methods are inherited by the subclass.
* Child Class (Subclass): This is the subclass, which inherits attributes and methods from the superclass and can also have its own attributes and methods.

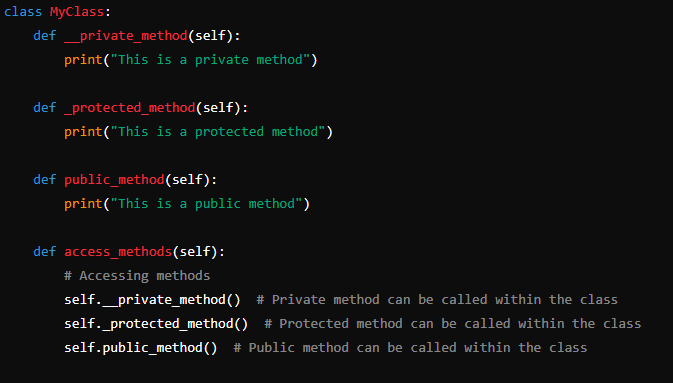


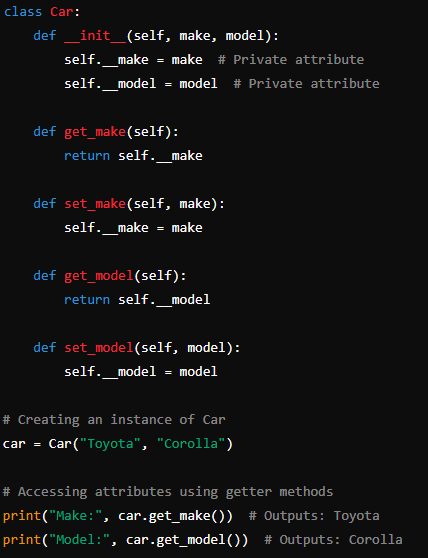
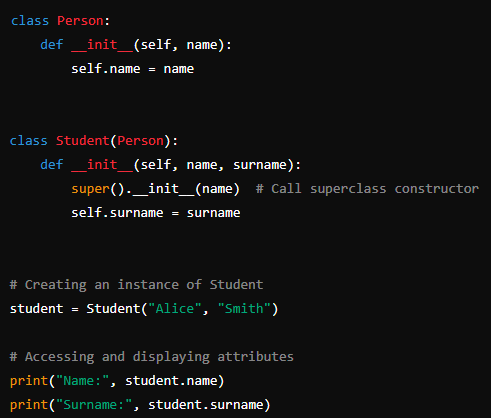
## *3. OOP - Encapsulation.*

A fundamental principle in object-oriented programming where the internal state of an object is hidden from the outside world, and access to it is restricted to methods of the class.

Basic Terms:

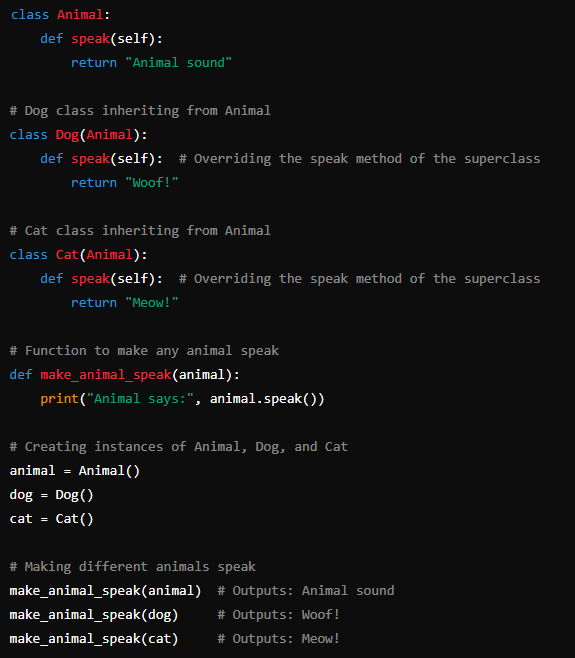
* **Data Hiding:** Encapsulation involves hiding the internal state of an object (data) and preventing direct access to it from outside the class.
* **Access Modifiers:** Keywords used in some programming languages to specify the accessibility of classes, methods, and attributes. In Python, we achieve encapsulation using naming conventions **(\_\_ prefix)** rather than access modifiers like **public**, **private**, or **protected**.
* **Private Methods:** Methods with names starting with double underscore \_\_ are considered private. These methods are *not directly inherited by subclasses*, but they can still be accessed indirectly using inheritance. ***But some exception***: The \_\_init\_\_ method is a special case in Python—it is inherited by subclasses even if it's declared as private. This behavior allows subclasses to properly initialize themselves even if the superclass constructor is private.
* **Protected Methods:** Methods with names starting with a single underscore \_ are considered protected. Although there's no strict enforcement of access control in Python, these methods are conventionally considered protected. *They are inherited by subclasses and can be accessed by them directly*.
* **Getter and Setter Methods:**
* Getter methods (get\_make, get\_model) allow access to the private attributes.
* Setter methods (set\_make, set\_model) allow modification of the private attributes.
* **Super () – Build - in Function:** Super ()is a built-in Python function used to call methods and access attributes from the parent class within a subclass. It's commonly used in inheritance to invoke the methods or constructors of the superclass.



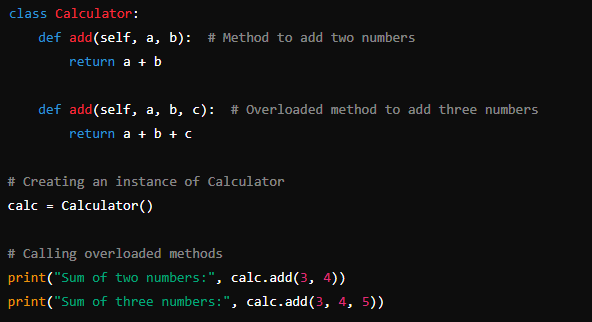
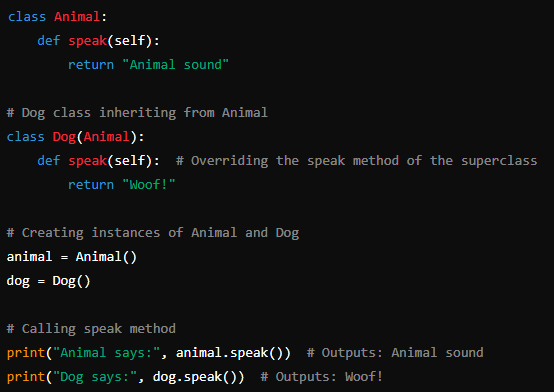
## *4. OOP - Polymorphism. Method overloading vs overriding.*

Polymorphism is a core principle in object-oriented programming that allows objects of different classes to be treated as objects of a common superclass. It enables a single interface to represent different types of objects and allows methods to behave differently based on the object they are invoked on.



Basic Terms:

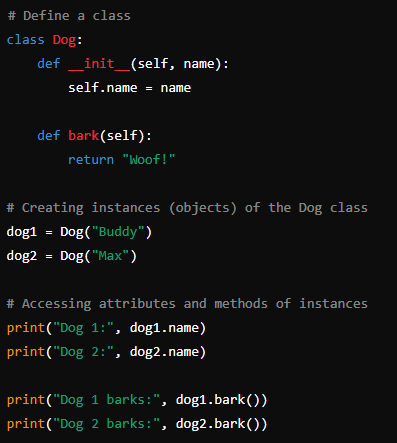
* **Overloading:** Overloading refers to the ability to define multiple methods in the same class with the same name but with different parameters or argument types. In Python, method overloading is not directly supported like in languages such as Java or C++, but we can achieve similar behavior through default parameter values or variable-length arguments.
* **Overriding:** Overriding occurs when a subclass provides a specific implementation of a method that is already defined in its superclass. The method in the subclass overrides the method in the superclass with the same name and signature.

## *5. OOP - Class vs Instance (or Object).*

**Class -** A class is a blueprint or template for creating objects. It defines the attributes and behaviors that objects of the class will have.It serves as a blueprint from which objects are created.Classes are defined using the class keyword in Python.

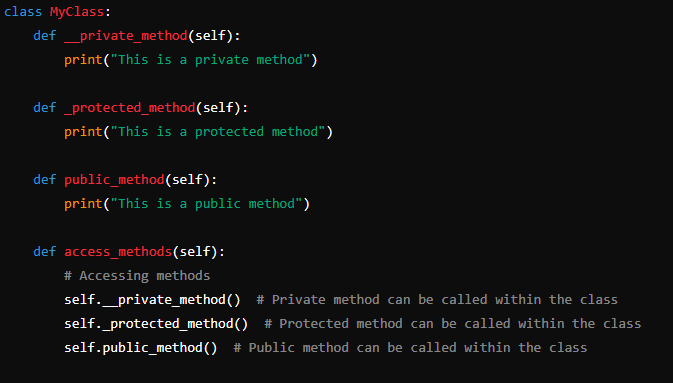
**Instance (or Object) -** An instance, also known as an object, is a concrete realization of a class.When you create an object from a class, you are creating an instance of that class.Each instance has its own set of attributes and can have its own state and behavior.Multiple instances of the same class can exist simultaneously, each with its own unique attributes and values.Instances are created using the class name followed by parentheses (ClassName()).

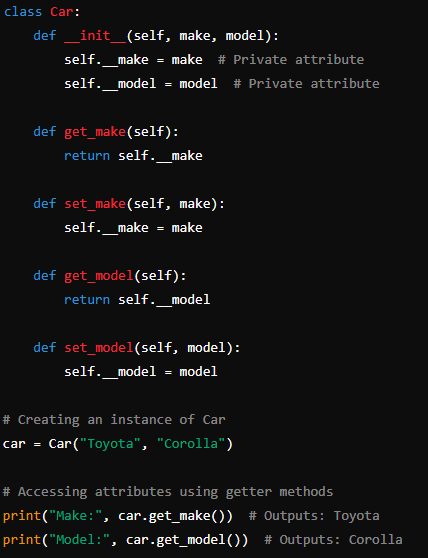
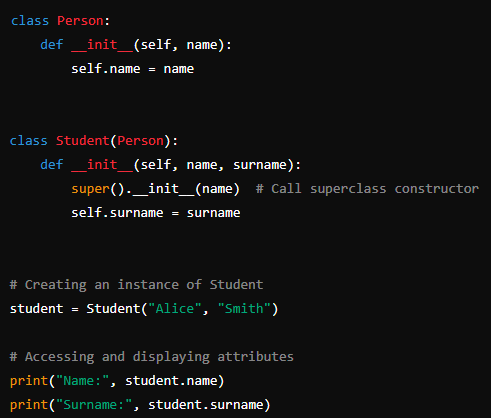
****

## *6. OOP - Access modifiers. Public, Private, Protected.*

**Access Modifiers:** Keywords used in some programming languages to specify the accessibility of classes, methods, and attributes. In Python, we achieve encapsulation using naming conventions **(\_\_ prefix)** rather than access modifiers like **public**, **private**, or **protected**.

* **Private Methods:** Methods with names starting with double underscore \_\_ are considered private. These methods are *not directly inherited by subclasses*, but they can still be accessed indirectly using inheritance. ***But some exception***: The \_\_init\_\_ method is a special case in Python—it is inherited by subclasses even if it's declared as private. This behavior allows subclasses to properly initialize themselves even if the superclass constructor is private.
* **Protected Methods:** Methods with names starting with a single underscore \_ are considered protected. Although there's no strict enforcement of access control in Python, these methods are conventionally considered protected. *They are inherited by subclasses and can be accessed by them directly*.
* **Getter and Setter Methods:**
* Getter methods (get\_make, get\_model) allow access to the private attributes.
* Setter methods (set\_make, set\_model) allow modification of the private attributes.
* **Super () – Build - in Function:** Super ()is a built-in Python function used to call methods and access attributes from the parent class within a subclass. It's commonly used in inheritance to invoke the methods or constructors of the superclass.

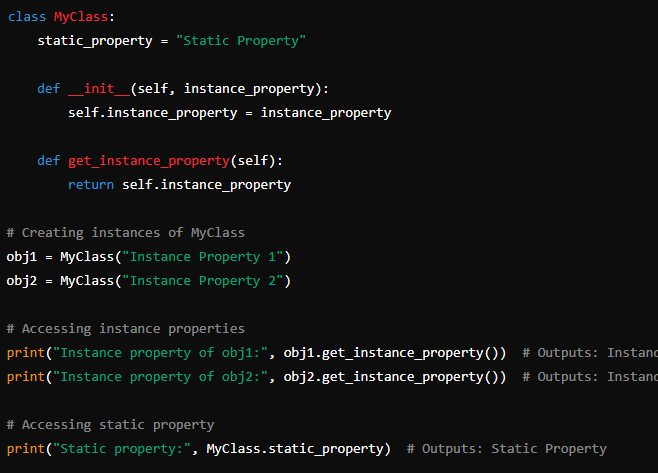


## *7. OOP - Static vs instance properties.*

**Instance Properties** - Instance properties are associated with individual instances (objects) of a class. Each instance has its own copy of instance properties, and their values can vary between different instances of the same class. Instance properties are defined inside the class but outside of any class methods, using self-keyword. They represent the state of individual objects and are accessed and modified using dot notation (object. Property).

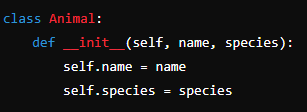
**Static (Class) Properties:** Static properties (or class properties) are associated with the class itself rather than with instances of the class. They are shared among all instances of the class. Changing the value of a static property for one instance affects all other instances. Static properties are defined at the class level, usually outside of any class methods, using the class name directly. They represent class-level data or configuration and are accessed using the class name (ClassName.property).



## *8. OOP - Constructors.*

Constructor:

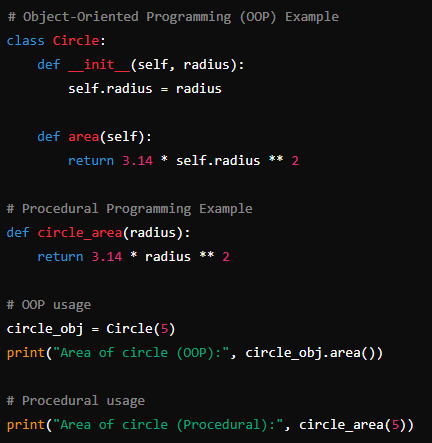
* A special method in Python classes that is automatically called when an object is instantiated.
* It initializes the object's attributes and can take parameters to set the initial state of the object.
* The constructor method is defined using the \_\_init\_\_ keyword.



## *9. OOP vs Procedural programming.*

**OOP:** OOP is a programming paradigm based on the concept of "objects", which can contain data (attributes) and code (methods). Focuses on modeling real-world entities using objects and classes. It emphasizes encapsulation, inheritance, and polymorphism to organize code into reusable and modular components.

**Procedural Programming:** Focuses on procedures or functions operating on data. It follows a top-down approach where the program is organized as a sequence of instructions or functions.

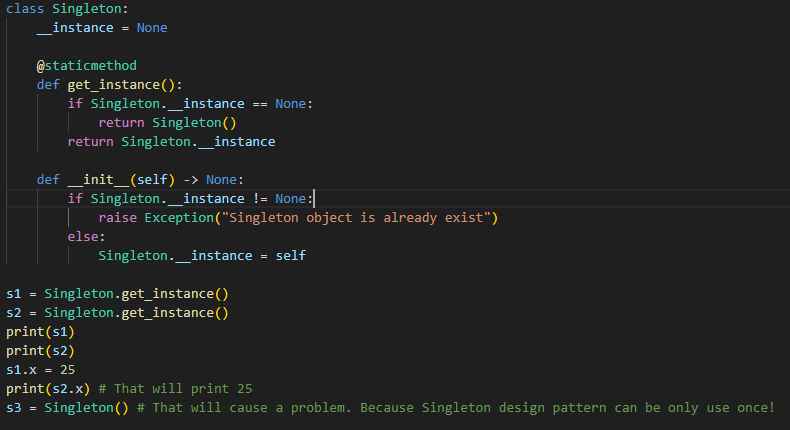


## *10. OOP Design patterns - Creation patterns.*

Creation patterns - Creation patterns are a category of design patterns in object-oriented programming (OOP) that deal with object creation mechanisms. These patterns help ensure that the creation process is more flexible and reusable.

***1. Singleton Pattern***  
 In this pattern, the user is not allowed to create an object by himself, that is, the user cannot create an object with `new`. A user can request an object from us, and in response to this request, we will give the user an existing object in memory. In this way, the user will only have to work with one object. Instead of creating 50 different objects for a task, we can use it in scenarios where we need to do the   
work with just 1 object.

The Singleton pattern is a creational design pattern that ensures a class has only one instance and provides a global point of access to that instance. Think of it as a way to make sure there is only one version of an object throughout the entire program.

The singleton pattern is a design pattern that restricts the instantiation of a class to one object.

Advantages of using the Singleton Method:

Initializations: An object created by the Singleton method is initialized only when it is requested for the first time.

Access to the object: We got global access to the instance of the object.

Count of instances: In singleton, method classes can’t have more than one instance

Disadvantages of using the Singleton Method:

Multithread Environment: It’s not easy to use the singleton method in a multithread environment, because we have to take care that the multithread wouldn’t create a singleton object several times.

Single responsibility principle: As the Singleton method is solving two problems at a single time, it doesn’t follow the single responsibility principle.

Unit testing process: As they introduce the global state to the application, it makes the unit testing very hard.

***2. Factory Method***

A Factory Pattern or Factory Method Pattern says that just define an interface or abstract class for creating an object but let the subclasses decide which class to instantiate. In other words, subclasses are responsible to create the instance of the class.

For example, u have burgers and for creation the new ones u use builder burger, where u just give new name and input ingredients, but during calling or pre-building, u can’t see ingredients.

Advantages of using Factory method:

We can easily add new types of products without disturbing the existing client code.

Disadvantages of using Factory method:

You end up with a huge number of small files i.e, cluttering the files

3. Abstract Factory

The Abstract Factory Pattern is a creational design pattern that provides an interface for creating families of related or dependent objects without specifying their concrete classes, in simpler terms the Abstract Factory Pattern is a way of organizing how you create groups of things that are related to each other.

Advantages of using Abstract Factory method:

This pattern is particularly useful when the client doesn’t know exactly what type to create.

It is easy to introduce new variants of the products without breaking the existing client code.

Disadvantages of using Abstract Factory method:

Our simple code may become complicated due to the existence of a lot of classes.

We end up with a huge number of small files i.e, cluttering of files.

***4. Builder Method***

Builder Method is a Creation Design Pattern which aims to “Separate the construction of a complex object from its representation so that the same construction process can create different representations.” It allows you to construct complex objects step by step. Here using the same construction code, we can produce different types and representations of the object easily.

It is basically designed to provide flexibility to the solutions to various object creation problems in object-oriented programming.

Unlike in Factory method for creation burger u build it by ur self where u have methods for choosing ingredients by urself.

Advantages of using Builder Method:

Reusability: While making the various representations of the products, we can use the same construction code for other representations as well.

Single Responsibility Principle: We can separate out both the business logic as well as the complex construction code from each other.

Construction of the object: Here we construct our object step by step, defer construction steps or run steps recursively.

Disadvantages of using Builder method:

Code complexity increases: The complexity of our code increases, because the builder pattern requires creating multiple new classes.

Mutability: It requires the builder class to be mutable

Initialization: Data members of the class are not guaranteed to be initialized.

5.Prototype method:

The Prototype Design Pattern is a creational pattern that enables the creation of new objects by copying an existing object. Prototype allows us to hide the complexity of making new instances from the client. The concept is to copy an existing object rather than create a new instance from scratch, something that may include costly operations. The existing object acts as a prototype and contains the state of the object.

**Advantages**

1. **Less number of SubClasses :**All the other Creational Design Patterns provides a lot of new subClasses which are definitely not easy to handle when we are working on a large project. But using Prototype Design Pattern, we get rid of this.

**Disadvantages**

1. **Waste of resources at lower level:**It might be proved as the overkill of resources for a project that uses very few objects

Creation patterns are a category of design patterns that deal with object creation mechanisms, trying to create objects in a manner suitable to the situation. Here are some common creation patterns:

1. Singleton Pattern

Intent: Ensure that a class has only one instance and provide a global point of access to it.

Example: Used in logging classes, database connection classes, and thread pool implementations.

2. Factory Method Pattern

Intent: Define an interface for creating an object, but let subclasses decide which class to instantiate. It defines a method that must be implemented by subclasses to create objects.

Example: Used in frameworks where subclasses need to decide the type of object to create, such as UI frameworks.

3. Abstract Factory Pattern

Intent: Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

Example: Used in GUI toolkits to provide a way to create families of related UI components (e.g., buttons, checkboxes) without specifying their concrete types.

4. Builder Pattern

Intent: Separate the construction of a complex object from its representation so that the same construction process can create different representations.

Example: Used when creating objects with many optional parameters or configuration settings, such as creating HTML or XML documents.

5. Prototype Pattern

Intent: Specify the kinds of objects to create using a prototypical instance and create new objects by copying this prototype.

Example: Used when the instantiation process is expensive or complex, and many similar objects are needed.

Creation patterns provide solutions to object creation problems, addressing issues such as ensuring single instances, providing flexible object creation mechanisms, and managing complex instantiation processes. By understanding and applying these patterns appropriately, developers can improve code reusability, maintainability, and flexibility in their object-oriented designs.

## *11. OOP Design patterns - Structural patterns.*

Structural patterns are a category of design patterns in Object-Oriented Programming (OOP) that focus on the composition of classes and objects to form larger structures. They help to define relationships between classes or objects, making it easier to design flexible and efficient systems. Here are some common structural patterns:

**1. Adapter Pattern**

Intent: Convert the interface of a class into another interface that a client expects. The Adapter lets classes work together that couldn’t otherwise because of incompatible interfaces.

Advantages:

Allows classes with incompatible interfaces to work together.

Increases class reusability by making existing classes work with others without modifying their source code.

Disadvantages:

Can increase the complexity of the code as new interfaces need to be created.

Overuse can lead to a messy and hard-to-maintain system with many small adapter classes.

**2. Facade Pattern**

Intent: Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.

Advantages:

Reduces complexity by providing a simple interface to a complex subsystem.

Shields clients from the complexities of the subsystem components.

Disadvantages:

Can become a god object coupled to all classes in the subsystem.

Changes in the subsystem might require changes in the facade.

## *12. OOP Design patterns - Behavioral patterns.*

*Behavioral patterns are design patterns that deal with how objects work together and communicate in a program. They focus on the interactions and responsibilities between objects.*

Behavioral patterns are a category of design patterns in object-oriented programming (OOP) that focus on how objects interact and communicate with each other. These patterns are concerned with the assignment of responsibilities between objects, as well as the delegation of control. They help in defining the flow of control and data among the components in a system, thereby enhancing flexibility in carrying out communication between objects.

**1. Observer Pattern**

**Intent**: Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

* **Advantages**:
  + Promotes loose coupling between the subject and observers.
  + Provides a flexible mechanism for components to communicate.
* **Disadvantages**:
  + Can lead to memory leaks if observers are not properly managed.
  + Notification order is not guaranteed.

***2. Iterator Pattern***

Intent: Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

Advantages:

Simplifies the interface to the collection.

Supports variations in the traversal of a collection.

Disadvantages:

Can be overkill for simple collections.

Additional complexity if implementing custom iterators.

## *13. OOP - What is SOLID.*

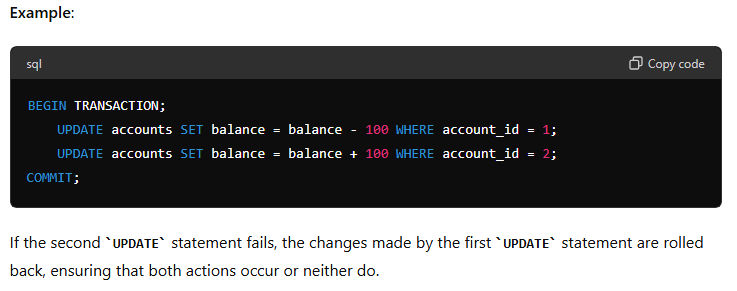
The SOLID Principles are five principles of Object-Oriented class design. They are a set of rules and best practices to follow while designing a class structure. These five principles help us understand the need for certain design patterns and software architecture in general. Following the SOLID acronym, they are:

* **Single Responsibility Principle (SRP)**: Each class should have only one reason to change, meaning it should only have one job or responsibility. This makes the code more modular, easier to understand, and maintain.
* **Open-Closed Principle (OCP)**: Software entities should be open for extension but closed for modification. This encourages adding new functionality through extensions rather than altering existing code, reducing the risk of introducing bugs.
* ·**Liskov Substitution Principle (LSP)**: Objects of a superclass should be replaceable with objects of a subclass without affecting the correctness of the program. This ensures that derived classes enhance functionality without altering the expected behavior of the base class.
* In simpler terms, if S is a subtype of T, then objects of type T can be replaced with objects of type S without altering any of the desirable properties of the program (correctness, task performed, etc.).
* **Interface Segregation Principle (ISP)**: Clients should not be forced to depend on interfaces they do not use. This leads to creating more specific and relevant interfaces, preventing the bloating of classes with unnecessary methods.
* **Dependency Inversion Principle (DIP)**: High-level modules should not depend on low-level modules; both should depend on abstractions. This decouples the code, making it more flexible and easier to manage changes.

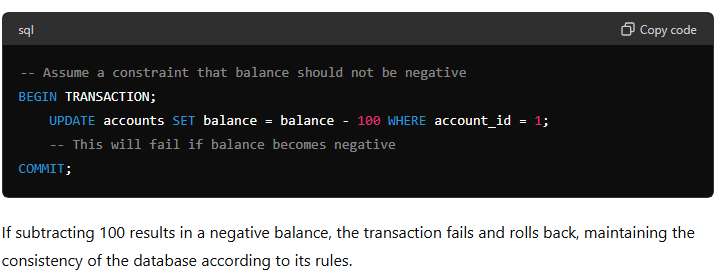
## *14. Relational Databases. What is ACID?*

ACID is an acronym that represents a set of properties that guarantee reliable processing of database transactions. These properties are crucial for maintaining the integrity and consistency of data in relational databases. Here’s a detailed explanation of each ACID property:

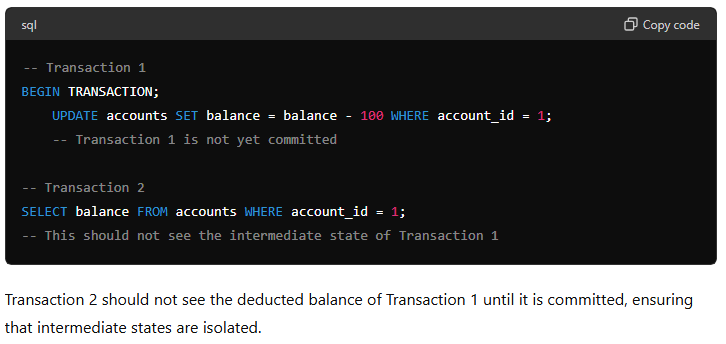
Definition: Atomicity ensures that a transaction is treated as a single unit, which either completely succeeds or completely fails. If any part of the transaction fails, the entire transaction is rolled back, leaving the database in its previous state.



Definition: Consistency ensures that a transaction brings the database from one valid state to another valid state, maintaining the predefined rules and constraints (such as primary keys, foreign keys, and unique constraints).



Definition: Isolation ensures that transactions are executed in isolation from one another. Concurrent transactions should not interfere with each other, and intermediate states of a transaction should not be visible to other transactions until the transaction is committed.



Definition: Durability ensures that once a transaction is committed, its changes are permanent and will survive any subsequent system failures. The data should be written to persistent storage such that it can be recovered after a crash.

## *15. Relational Databases. What is indexing and how it works.*

**Relational Databases**

Relational databases are a type of database that store and provide access to data points that are related to one another. They are structured to recognize relations among stored items of information. The standard user and application programming interface (API) of a relational database is the Structured Query Language (SQL). The most common relational databases include MySQL, PostgreSQL, Oracle Database, and Microsoft SQL Server.

Indexing is a technique used in databases and search engines to improve the speed of data retrieval operations. It works by creating a data structure, typically a B-tree or hash table, that allows for quick lookups, similar to the index at the end of a book. Here's a detailed explanation of indexing and its workings:

**What is Indexing?**

Indexing is a data structure technique that involves creating an auxiliary structure (an index) that stores a subset of the data or a transformed version of the data from the main table, along with pointers to the actual data. This index allows for faster searches, queries, and data retrieval.

**How Indexing Works**

Creation of Index:

*Indexing Key:* When an index is created, a specific column (or columns) is chosen as the index key. This key is used to create the index.

*Data Structure:* The index is typically stored in a data structure optimized for quick searches, such as a B-tree, B+ tree, or a hash table.

Insertion into Index:

When data is inserted into the main table, the same data (or a reference to it) is also inserted into the index structure. The index maintains the order of the keys, facilitating faster searches.

Searching with Index:

When a search query is performed, the system uses the index to quickly locate the keys matching the query.

The index provides pointers to the location of the corresponding data in the main table, allowing for fast retrieval.

Updating Index:

If the data in the indexed column is updated, the index also needs to be updated to reflect these changes. This involves finding the old index entry, removing it, and inserting the new entry.

**Types of Indexes**

Primary Index:

Created on the primary key of a table.

Ensures that each key in the index is unique.

Secondary Index:

Created on non-primary key columns.

Can be non-unique and used to improve the performance of queries involving those columns.

Composite Index:

An index on multiple columns.

Useful for queries that filter on multiple columns together.

Clustered Index:

The data rows are stored in the index order.

Only one clustered index is allowed per table because the data rows themselves can be sorted in only one order.

Non-clustered Index:

The index contains pointers to the data rows stored separately.

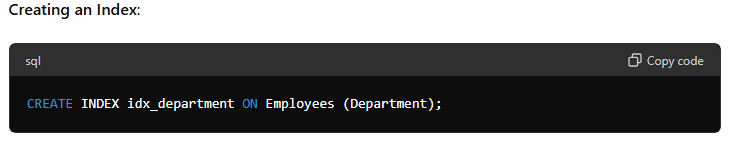
Multiple non-clustered indexes can exist on a table.

### Advantages of Indexing

1. **Faster Query Performance**: Indexes significantly reduce the time required to locate records, leading to faster query execution.
2. **Efficient Sorting**: Indexes can be used to retrieve sorted data without additional sorting.
3. **Improved Join Performance**: Indexes on columns used in joins can speed up the join operations.

### Disadvantages of Indexing

1. **Increased Storage**: Indexes require additional disk space.
2. **Slower Write Operations**: Insert, update, and delete operations can be slower because the index needs to be updated alongside the data.
3. **Complexity**: Maintaining and optimizing indexes can add complexity to database management.



## *16. Relational Databases. Database Denormalization.*

Denormalization is the process of optimizing the read performance of a database by adding redundant data. It involves merging tables and storing the results in a single table, which can make some queries faster and easier to write. Denormalization is typically used in situations where the performance of read-heavy operations needs to be improved, often at the cost of increased complexity in write operations and data consistency.

Why Denormalize?

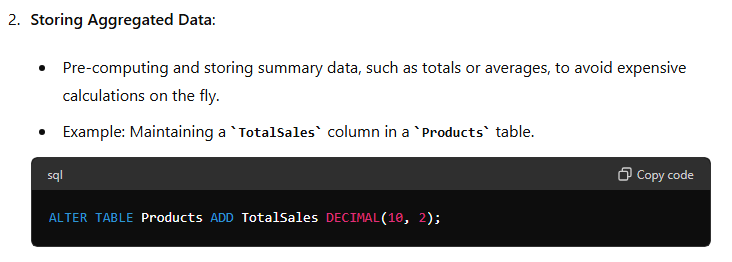
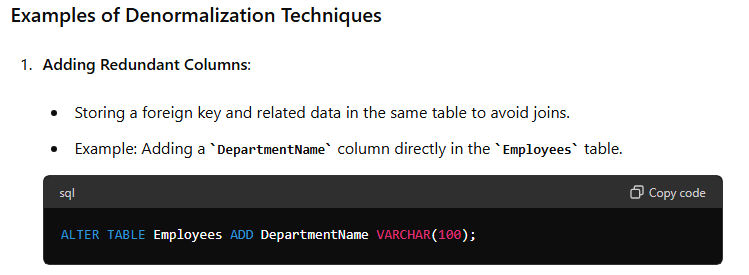
Performance Improvement: Denormalization can significantly speed up read operations because it reduces the need for complex joins and can take advantage of pre-computed data.

Simplified Queries: By reducing the number of tables and joins required to retrieve data, queries become simpler and more efficient.

Caching Optimization: It can be easier to cache denormalized data, further speeding up read operations.

How Denormalization Works

Denormalization involves introducing redundancy into a database by combining data from multiple normalized tables into fewer tables. This process is often guided by specific query patterns that require optimized performance.

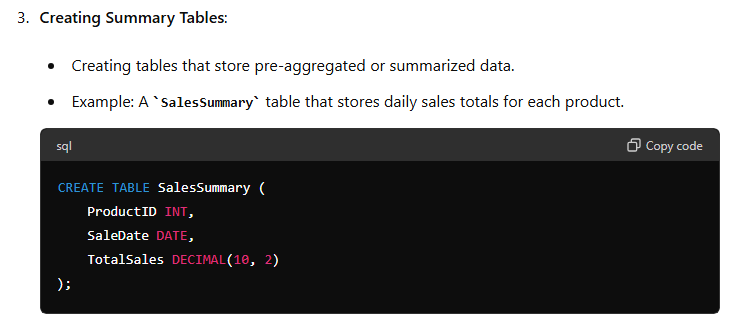


### Advantages of Denormalization

1. **Improved Read Performance**: Queries that would require multiple joins in a normalized database can be executed more quickly.
2. **Simpler Query Logic**: Denormalization can reduce the complexity of queries, making them easier to write and maintain.
3. **Better Reporting and Analytics**: Aggregated data and summary tables can speed up complex analytical queries and reports.

### Disadvantages of Denormalization

1. **Increased Storage Requirements**: Redundant data requires more storage space.
2. **Data Anomalies and Inconsistencies**: Maintaining redundant data can lead to anomalies if the data is not kept in sync.
3. **Complexity in Data Management**: Inserts, updates, and deletes become more complex and need to ensure consistency across redundant data.
4. **Potential for Data Integrity Issues**: Without careful management, denormalized data can lead to integrity issues.



## *17. Relational Databases. Foreign keys. Relationships types - one-to-one, one-to-many etc.*

**Relational Databases**

Relational databases are a type of database that store and provide access to data points that are related to one another. They are structured to recognize relations among stored items of information. The standard user and application programming interface (API) of a relational database is the Structured Query Language (SQL). The most common relational databases include MySQL, PostgreSQL, Oracle Database, and Microsoft SQL Server.

Foreign Keys

A foreign key is a column or a set of columns in one table that uniquely identifies a row of another table. It is used to establish and enforce a link between the data in two tables. The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

Purpose of Foreign Keys

Data Integrity: Ensures that the data remains accurate and consistent. A foreign key ensures that the value in a child table matches one of the values in the parent table.

Referential Integrity: Prevents actions that would leave orphan records in a child table. For example, it ensures that you cannot delete a record from a parent table if related records exist in the child table.

Relationship Types

Relationships in relational databases can be categorized based on the number of entitie involved and how they are associated. The most common types are:

One-to-One (1:1)

In a one-to-one relationship, each record in one table is linked to a single record in another table, and vice versa. This type of relationship is less common and is often used to split a table into two parts for efficiency or to store sensitive data separately.

Example:

A Person table and a Passport table where each person has one passport and each passport is assigned to one person.

One-to-Many (1

)

In a one-to-many relationship, a single record in one table (the parent) is related to multiple records in another table (the child). This is the most common relationship type.

Example:

• A Customer table and an Order table where each customer can place multiple orders.

Many-to-One (M:1)

This is simply the reverse of the one-to-many relationship. Multiple records in one table are related to a single record in another table.

Example:

• A Book table and a Publisher table where each book is published by one publisher, but each publisher can publish multiple books.

Many-to-Many (M

)

In a many-to-many relationship, multiple records in one table are related to multiple records in another table. This type of relationship requires a junction table (also known as a join table) to break down the many-to-many relationship into two one-to-many relationships.

Example:

• A Student table and a Course table where each student can enroll in multiple courses, and each course can have multiple students.

Summary

Relational databases are powerful tools for organizing and retrieving related data efficiently. Foreign keys are fundamental in maintaining relationships and ensuring data integrity. Understanding the types of relationships (one-to-one, one-to-many, many-to-many) is crucial for designing a database schema that accurately reflects the real-world entities and their interactions.

## *18. SQL basics. DDL commands.*

In SQL (Structured Query Language), DDL (Data Definition Language) commands are used to define, modify, and delete database objects such as tables, indexes, views, and schemas. DDL commands are critical for setting up the structure and schema of a database. Here are the primary DDL commands and their usage:

1-> CREATE

The CREATE command is used to create new database objects like tables, indexes, views, or schemas.

Syntax:

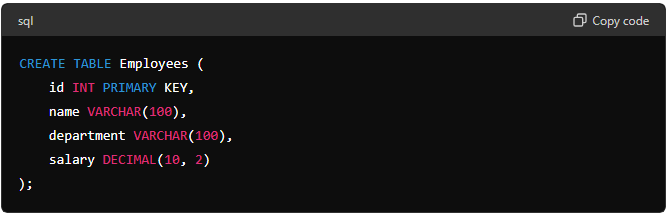
CREATE TABLE table\_name (

column1 datatype,

column2 datatype,

...

);



2 -> ALTER

The ALTER command is used to modify existing database objects.

Syntax for adding new column to table:

ALTER TABLE table\_name ADD column\_name datatype;

Syntax for modifying a column in a table:

ALTER TABLE table\_name MODIFY COLUMN column\_name datatype;



3 -> DROP

The DROP command is used to delete existing database objects like tables, indexes, or views.

Syntax:

DROP TABLE table\_name;



4 -> TRUNCATE

The TRUNCATE command is used to remove all rows from a table, but the table structure and its metadata remain intact.

Syntax:

TRUNCATE TABLE table\_name;

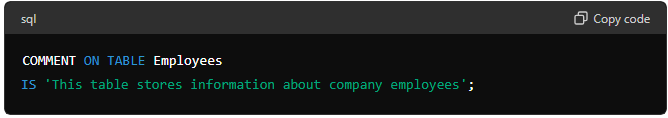


5 -> COMMENT

The COMMENT command is used to add comments or descriptions to database objects.

Syntax:

COMMENT ON TABLE table\_name IS 'This is a comment.';



6 -> RENAME

The RENAME command is used to rename an existing database object.

Syntax:

RENAME TABLE old\_table\_name TO new\_table\_name;



## *19. SQL basics. DML commands.*

In SQL (Structured Query Language), DML (Data Manipulation Language) commands are used to manage data within database objects like tables. Unlike DDL (Data Definition Language) commands that focus on defining and modifying the structure of the database, DML commands are used to manipulate the data stored within those structures. Here are the primary DML commands in SQL:

1 -> SELECT

The SELECT statement is used to retrieve data from one or more tables.

Syntax:

SELECT column1, column2, ... FROM table\_name WHERE condition;

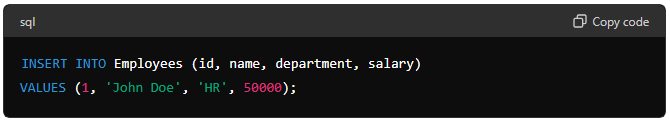


2 -> INSERT

The INSERT statement is used to insert new rows of data into a table.

Syntax:

INSERT INTO table\_name (column1, column2, column3, ...) VALUES (value1, value2, value3, ...);



3 -> UPDATE

The UPDATE statement is used to modify existing data in a table.

Syntax:

UPDATE table\_name SET column1 = value1, column2 = value2, ... WHERE condition;



4 -> DELETE

The DELETE statement is used to delete existing rows of data from a table.

Syntax:

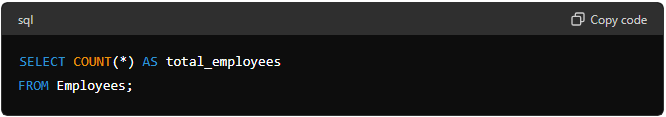
DELETE FROM table\_name WHERE condition;



## *20. SQL basics. Aggregation functions.*

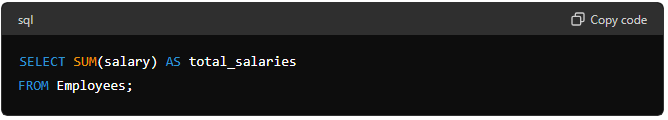
1 -> COUNT()

The COUNT() function returns the number of rows that match a specified criterion.



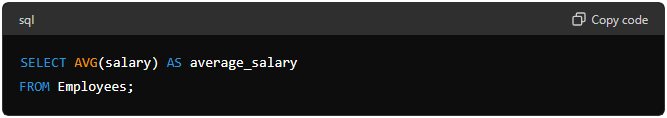
2 -> SUM()

The SUM() function returns the total sum of a numeric column.



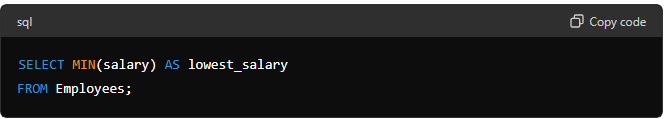
3 -> AVG ()

The AVG () function returns the average value of a numeric column.



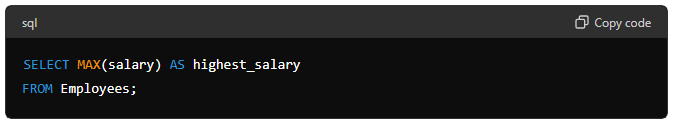
4 -> MIN ()

The MIN () function returns the smallest value in a column.



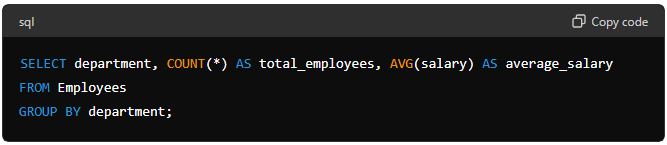
5 -> MAX ()

The MAX () function returns the largest value in a column.



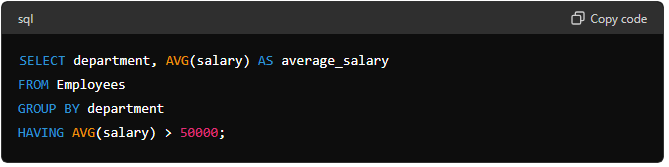
GROUP BY ->

Aggregation functions are often used with the GROUP BY clause to group the result set by one or more columns and perform the aggregation on each group.



HAVING ->

The HAVING clause is used to filter groups based on a condition. It is similar to the WHERE clause, but WHERE cannot be used with aggregate functions.



## *21. SQL basics. Transactions. Commit, Rollback.*

In SQL, transactions are a sequence of operations performed as a single logical unit of work. A transaction must be atomic, consistent, isolated, and durable (ACID properties). Transactions are essential for ensuring data integrity and consistency, especially in systems with concurrent users.

**Key Concepts of Transactions**

* **Atomicity:** Ensures that all operations within the transaction are completed successfully. If any operation fails, the entire transaction is rolled back.
* **Consistency:** Ensures that the database transitions from one valid state to another valid state.
* **Isolation:** Ensures that transactions are executed in isolation from each other, preventing concurrent transactions from affecting each other's outcome.
* **Durability:** Ensures that once a transaction is committed, its changes are permanent, even in the event of a system failure.

**Transaction Control Commands**

START TRANSACTION or BEGIN TRANSACTION

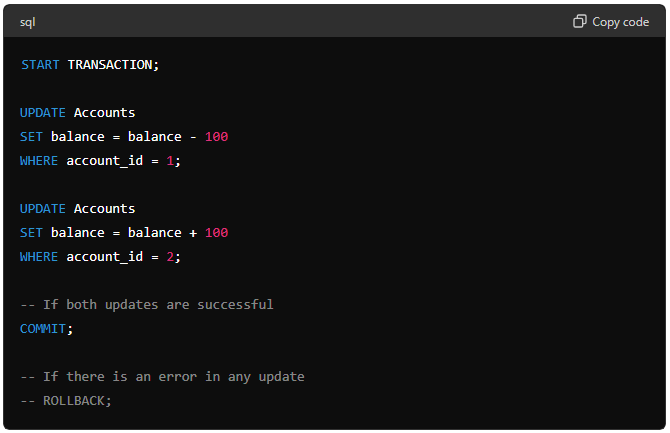
These commands are used to start a new transaction.



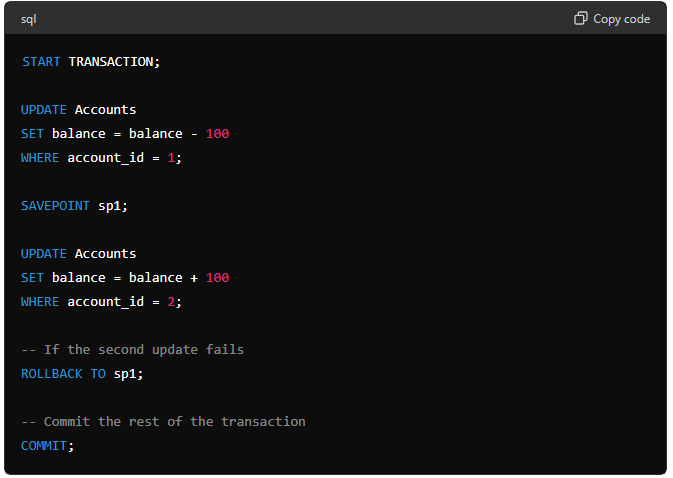
The COMMIT command is used to save all changes made during the transaction to the database. Once committed, the changes become permanent and visible to other users.

The ROLLBACK command is used to undo all changes made during the transaction. This command reverts the database to its previous state before the transaction began.

Consider a scenario where we are transferring money between two accounts. This involves two operations: debiting one account and crediting another. Both operations must succeed or fail together to maintain data integrity.



Savepoints allow you to set a point within a transaction to which you can later roll back. This is useful for partial rollbacks within a larger transaction.

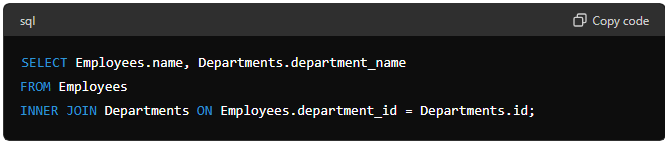


* SAVEPOINT sp1 sets a savepoint after the first update.
* If the second update fails, ROLLBACK TO sp1 reverts the transaction to the savepoint, undoing only the changes made after the savepoint.
* The transaction can then be committed or further modified before committing.

## *22. SQL basics. Types of joins.*

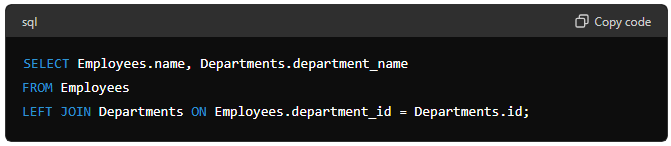
1 -> INNER JOIN

The INNER JOIN returns only the rows where there is a match in both tables.



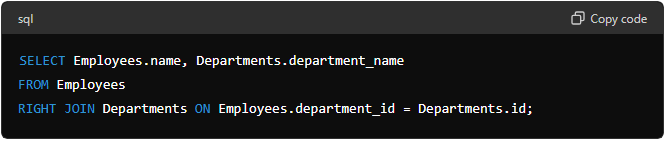
2 -> LEFT JOIN

The LEFT JOIN returns all rows from the left table, and the matched rows from the right table. If there is no match, the result is NULL on the side of the right table.



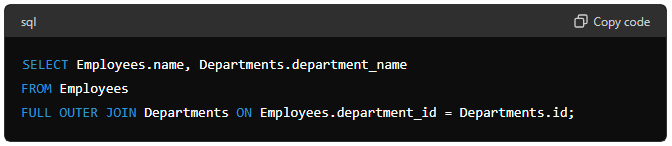
3 -> RIGHT JOIN

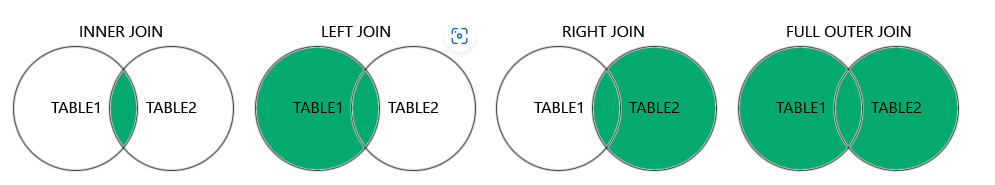
The RIGHT JOIN returns all rows from the right table, and the matched rows from the left table. If there is no match, the result is NULL on the side of the left table.



4 -> FULL JOIN

The FULL JOIN returns all rows when there is a match in either left or right table. Rows without a match in one of the tables will have NULL in the columns of that table.



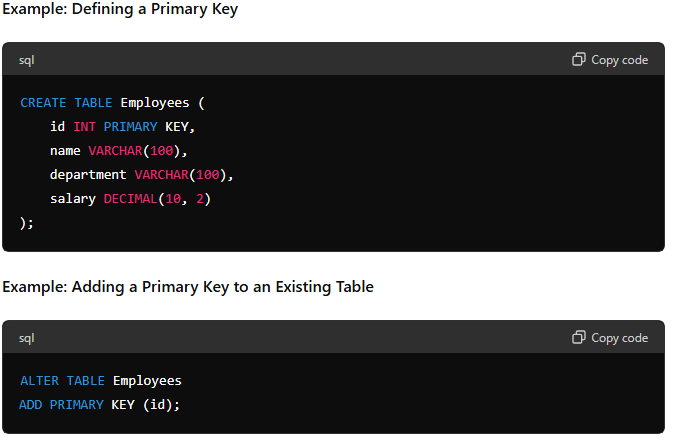


## *23. SQL basics. Primary key. Unique key. Null values.*

A primary key is a field (or a combination of fields) in a table that uniquely identifies each row in that table. Primary keys must contain unique values, and they cannot contain null values. Each table can have only one primary key.

Characteristics of Primary Keys:

* Uniquely identifies each record.
* Cannot contain null values.
* Ensures data integrity by preventing duplicate entries.



A unique key is a constraint that ensures all values in a column or a set of columns are unique. Unlike primary keys, a table can have multiple unique keys. Unique keys allow null values, but only one null value per column (if the column allows nulls).

Characteristics of Unique Keys:

* Ensures that all values in the column are different from each other.
* Allows null values (one null per unique column).
* A table can have multiple unique keys.



A null value represents missing or unknown data in a column. Null is not the same as zero or an empty string. Null means that no value has been assigned to the field. In SQL, you can check for null values using the IS NULL or IS NOT NULL conditions.



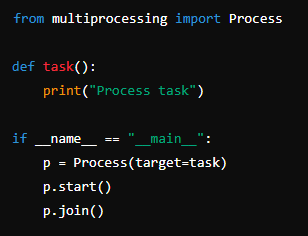
## *24. Python basics. Lists, tuples, sets, dictionaries.*



## *25. Python basics. Threads vs Processes.*

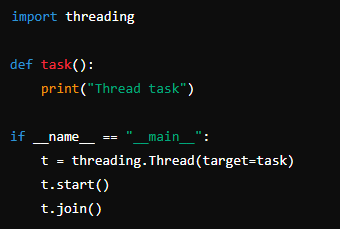
**Process:**

* **Definition**: Independent execution units with their own memory space.
* **Isolation**: Separate memory space; no shared data.
* **Best For**: CPU-bound tasks (e.g., calculations).
* **Overhead**: Higher; more resource-intensive to create and manage.
* **Library**: multiprocessing.



**Threads:**

* **Definition**: Smaller units of a process that share the same memory space.
* **Sharing Data**: Shared memory space; easy data sharing but needs synchronization.
* **Best For**: I/O-bound tasks (e.g., file operations, network requests).
* **Overhead**: Lower; lightweight to create and manage.
* **Library**: threading.



**Key Differences**

1. **Memory**:
   * Processes: Separate.
   * Threads: Shared.
2. **Execution**:
   * Processes: True parallelism on multi-core CPUs.
   * Threads: Concurrent but limited by Python's GIL for CPU-bound tasks.
3. **Overhead**:
   * Processes: Higher.
   * Threads: Lower.

**Conclusion**

* Use **processes** for tasks that require heavy computation.
* Use **threads** for tasks that involve waiting for I/O operations.