**Intent :**

The Factory Design Pattern allows the system to decide which type of database connection to create based on dynamic conditions such as user input, configuration settings, or runtime conditions. The factory method abstracts the logic of creating database objects, allowing the system to switch between different types of databases (e.g., MySQL, PostgreSQL, MongoDB) seamlessly without requiring changes to the client code..

**Role of Participants:**

**1.DatabaseConnection:** :The DatabaseConnection interface defines the contract for all database connection classes. It enforces the presence of methods like connect(),executeQuery() and disconnect() that all concrete classes must implement. This provides a common interface for interacting with any type of database, ensuring uniformity and interchangeability.

**2.MySql :** :MySQL Database Connection: Implements the connect() method to establish a connection with a MySQL database, typically using MySQL JDBC drivers and connection strings specific to MySQL, Implements executeQuery() method to execute select query,Implements the disconnect() method to close the MySQL connection, ensuring resources are released properly..

**3.Sqlite :** The SQLite Connection Class contains the logic for establishing a connection to an SQLite database. It handles SQLite-specific configurations, such as the database file path or in-memory database setup.

4.**PostgreSql:** The PostgreSQL Connection Class contains the logic required to establish a connection to a PostgreSQL database. This includes setting up the JDBC driver, configuring the connection string, and managing the connection protocol specific to PostgreSQL.

**3.Factory :**The Factory Class is responsible for creating and returning instances of different database connection classes based on the provided input. It abstracts away the instantiation details, allowing the client to obtain database connections without knowing the specific implementation**.**

**4.Main** : The entry point of the application. This class simulates user input, interacts with the DatabaseFactory to create concrete class object, and invokes methods to connect, query, and disconnect from databases.

**Mapping participants with actual class :**

**1. Interface(DatabaseConnection)**.

**2. Concrete class (MySql, Sqlite,Postgresql)**

**3. Factory class (Factory)**

**4. Main class (connectionMain)**

**Consequence :**

**Positive Consequences:**

**1.Encapsulation of Object Creation:**

**The Factory Pattern encapsulates the logic of object creation, specifically database connections. The client code no longer needs to know the details about how database connections are instantiated. This is important for code maintainability as the object creation logic is centralized in the factory (DatabaseFactory).**

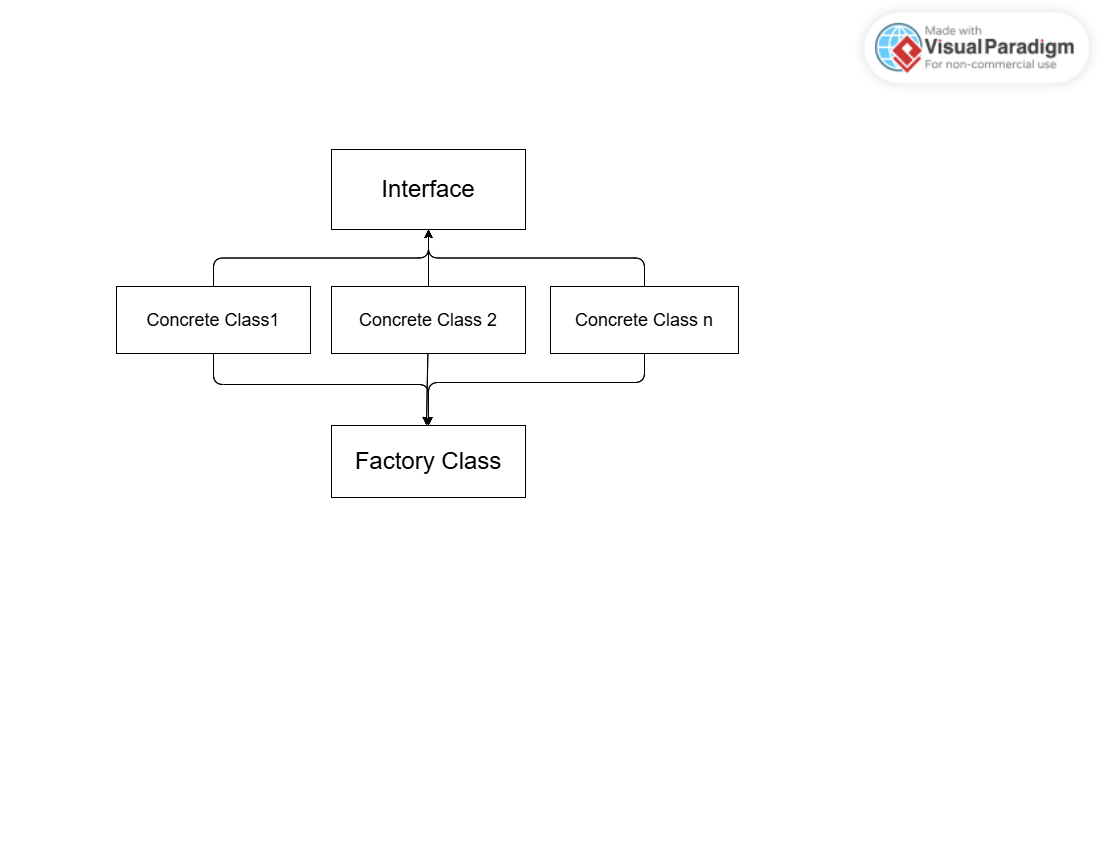
**2.Flexibility and Extensibility:**

**New database types can easily be added to the system by simply adding new concrete connection classes and updating the DatabaseFactory to handle the new types. This makes the system highly extensible.**

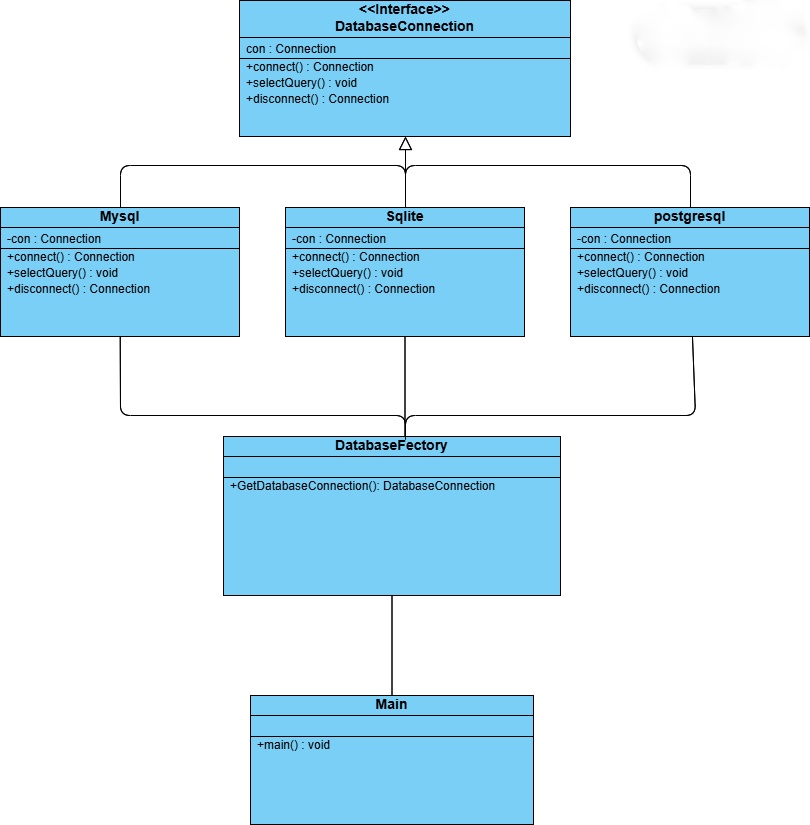
**3.Consistency and Uniform Interface:**

**All concrete database classes implement the same interface (DatabaseConnection), providing the client code with a consistent interface for interacting with different types of databases. The client doesn’t need to worry about the specifics of the database connection; it can use the same set of methods (connect(), disconnect(), performQuery()).**

**Structure Diagram:**

****

**Class Diagram:**

****