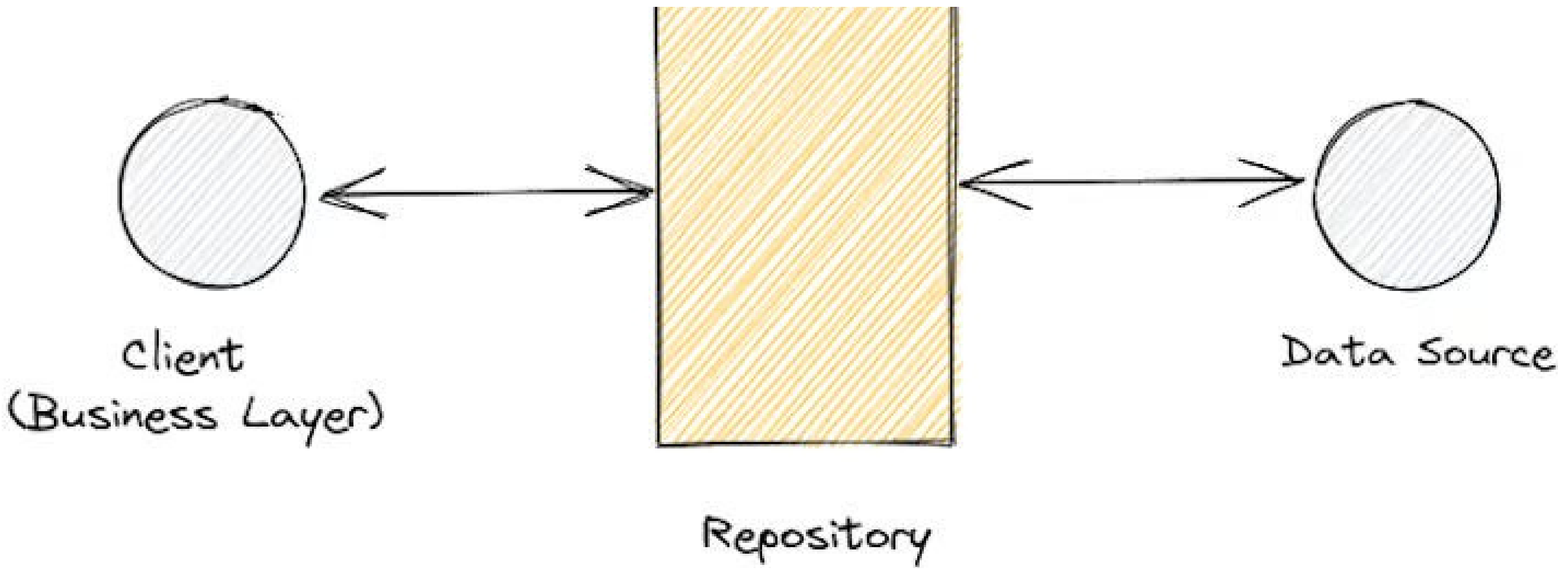
REPOSITORY AND TRANSACTION IN SQL

Repository Pattern

- The Repository Pattern is a design pattern that acts as an intermediary between the Business Logic Layer and the Data Access Layer.
- It hides the data access logic (SQL queries, ORM code, etc.) and provides a clean way for business logic to interact with the data.
- Seperates the data access layer from the bussiness logic.

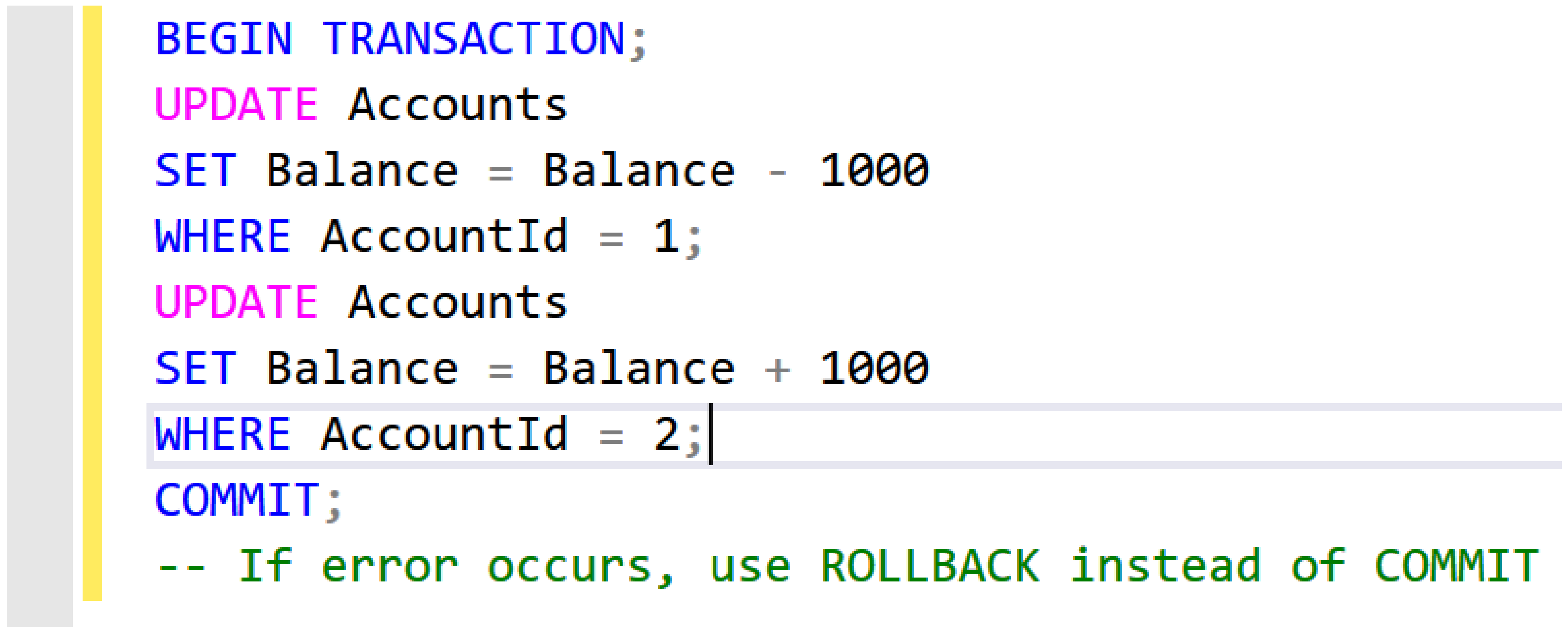


Benefits

- Seperation of bussiness logic and data access layer
- Resuability of data acess code
- Makes it easier to mock and test.
- Centralized data access
- Support for multiple data source

Transactions

- A transaction in SQL is a sequence of one or more database operations (INSERT, UPDATE, DELETE, etc.) that are executed as a single unit of work.
- Either all operations succeed (commit) or none of them take effect (rollback).
- Ensures data integrity and prevents partial updates.



```sql
BEGIN TRANSACTION;
UPDATE Accounts
SET Balance = Balance - 1000
WHERE AccountId = 1;
UPDATE Accounts
SET Balance = Balance + 1000
WHERE AccountId = 2;
COMMIT;
-- If error occurs, use ROLLBACK instead of COMMIT
```

Unit of Work pattern

- It is a design pattern that groups database related operations into a single transaction.
- Maintains a list of changes (insert, update, delete) made to objects during a business transaction and changes are committed as one transaction or rolled back if something fails.
- Used when there is need to work on or update multiple entity at one call.
- Minimizes database transactions.
- Maintains data integrity and consistency.

ACID Principles

- ACID is a set of properties of database transactions that ensure data reliability and consistency.
- Every transaction in a database should follow these four principles.

1) Automicity:

- All or nothing rule.
- A transaction is treated as a single unit.
- If one operation fails, the entire transaction fails, and all changes are rolled back.
- Example: In a bank transfer, if debit succeeds but credit fails → rollback.

2.Consistency:

- Ensures that a transaction brings the database from one valid state to another.
- Data must always follow defined rules, constraints, and relationships.
- Example: A transfer should not create or lose money — total balance before and after must remain consistent.

3. Isolation

- Ensures that transactions run independently of each other.
- One transaction should not see partial results of another transaction.
- Example: Two people booking the same seat → one should succeed, the other should fail, not both.

4. Durability

- Once a transaction is committed, the changes are permanent.
- Even in case of a system crash, power failure, or restart, committed data is safe .
- Example: After booking a ticket, confirmation remains even if the server crashes.

Thank you very much!