Transaction management

Transaction management & isolation levels

CODEUS_

Transaction

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Alice';
-- etc etc
COMMIT;
```

- A logical unit that bundles multiple steps into a single, **all-or-nothing operation**
- Basic commands:
  - `BEGIN` - open a transaction
  - `COMMIT` - commit the transaction
  - `ROLLBACK [TO savepoint]` - rollback the transaction. Optionally rollback to the savepoint

Postgres actually treats every SQL statement as being executed within a transaction. If you do not issue a `BEGIN` command, then each individual statement has an implicit `BEGIN` and (if successful) `COMMIT` wrapped around it.

ACID - key transaction properties

- A - atomicity. The transaction either happens completely or not at all
- C - consistency. Database is consistently viable before and after the transaction (e.g., table constraints)
- I - isolation. Multiple concurrent transactions should not affect each other
- D - durability. Once the transaction is completed and acknowledged by the database, it won't be lost even in case a crush occurs shortly thereafter

Savepoints

- Savepoints allow to selectively discard parts of the transaction, while committing the rest
- Can be defined with SAVEPOINT [name] statement
- If needed, you can roll back (even multiple times) to the savepoint with ROLLBACK TO [name] command
- In case you don’t need to roll back to a particular savepoint, it can be released with RELEASE [SAVEPOINT] [name], so the system can free some resources
- Releasing or rolling back to a savepoint will automatically release all savepoints defined after it

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Alice';
SAVEPOINT my_savepoint;
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Bob';
-- oops ... forget that and use Wally's account
ROLLBACK TO my_savepoint;
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Wally';
COMMIT;
```

Anomalies & Isolation levels

SQL standard defines 4 levels if transaction isolation:

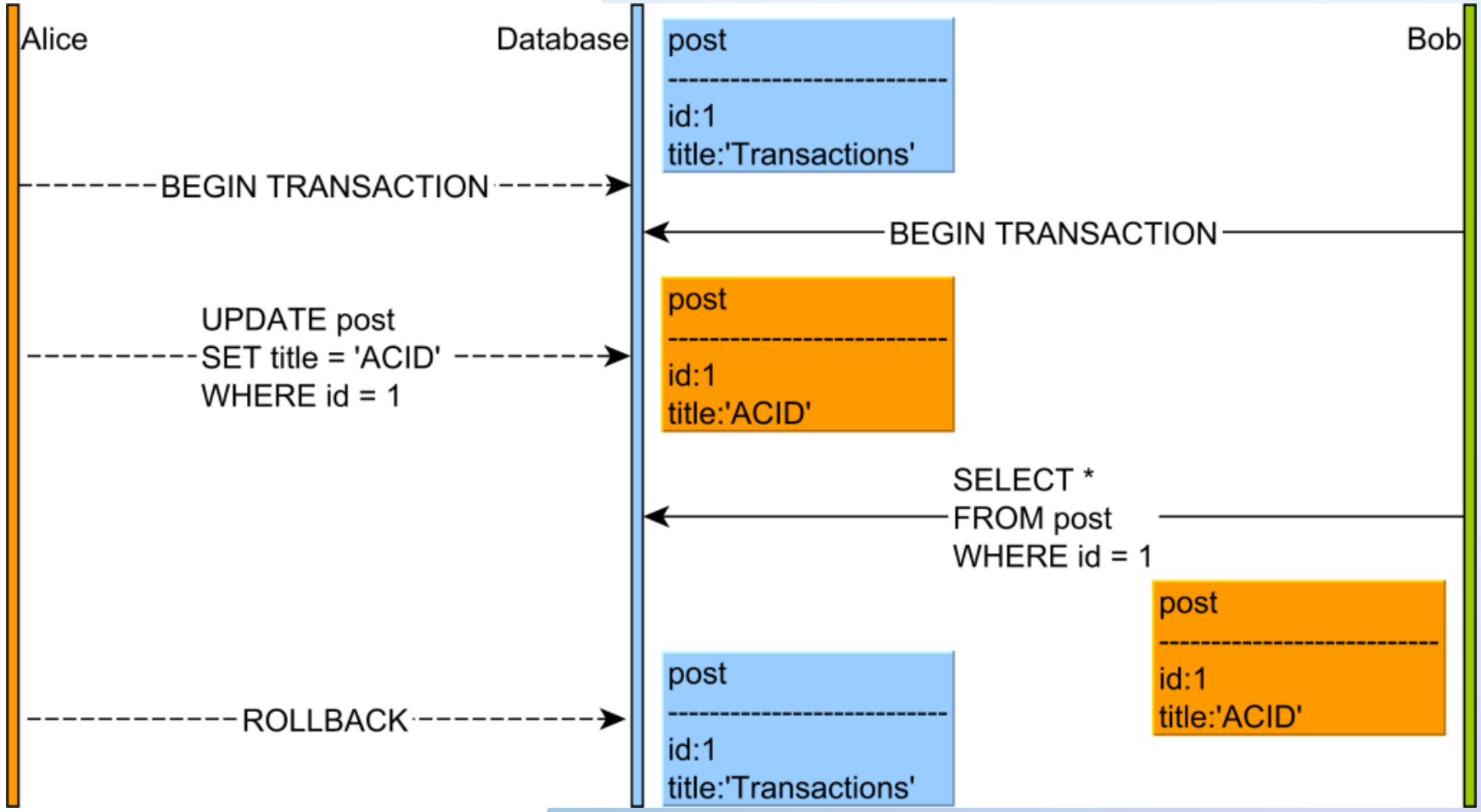
- Read uncommitted
- Read committed
- Repeatable read
- Serializable. The most strict one which says that concurrent transactions have effect as they were run one at a time at some order

```
BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SELECT pg_export_snapshot();
 pg_export_snapshot
--------------------
 00000003-0000001B-1
(1 row)
```

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read	Serialization Anomaly
Read uncommitted	Allowed, but not in PG	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible	Possible
Repeatable read	Not possible	Not possible	Allowed, but not in PG	Possible
Serializable	Not possible	Not possible	Not possible	Not possible

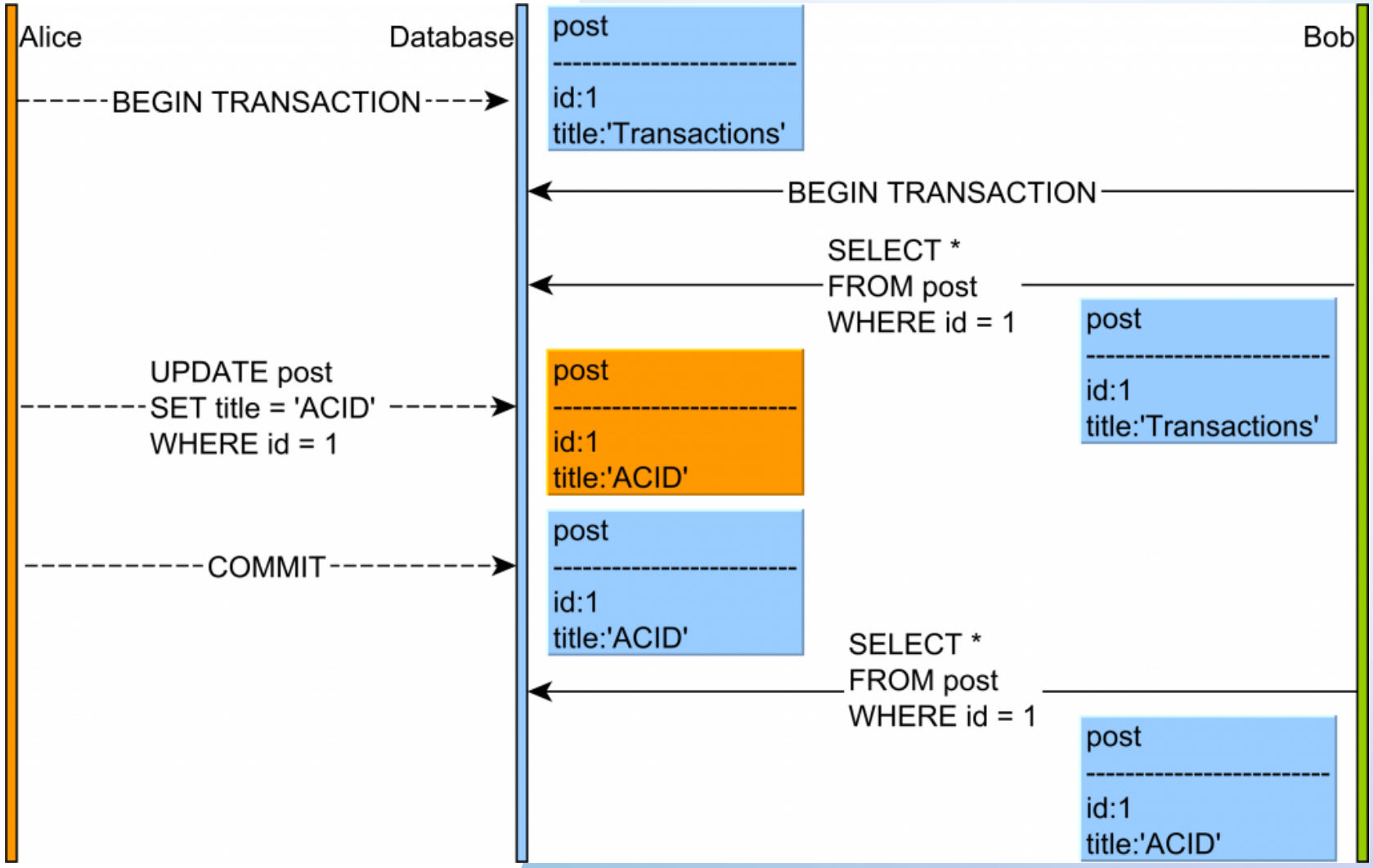
Anomalies & Isolation levels

Dirty read



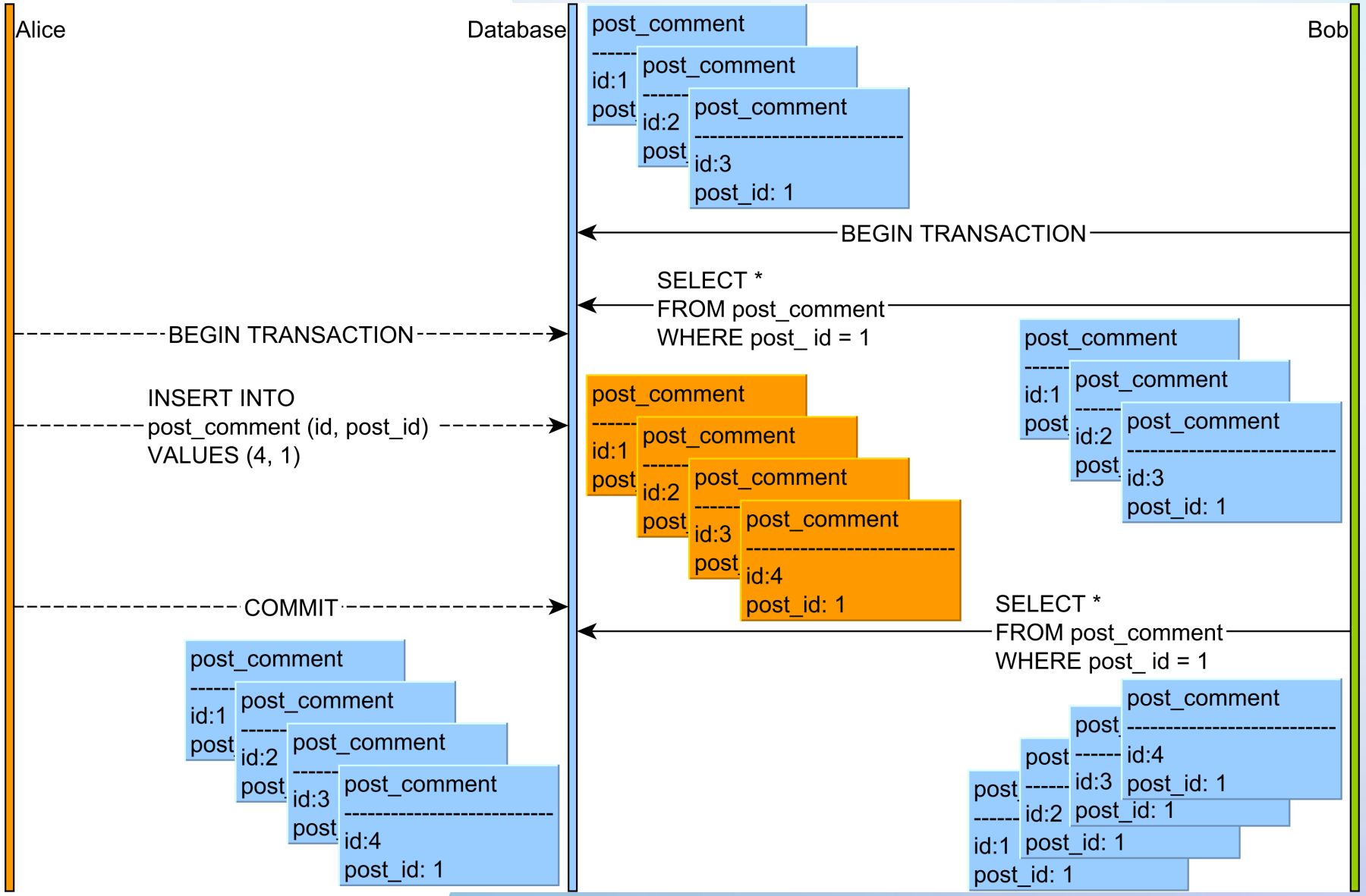
Anomalies & Isolation levels

Non-repeatable read



Anomalies & Isolation levels

Phantom read

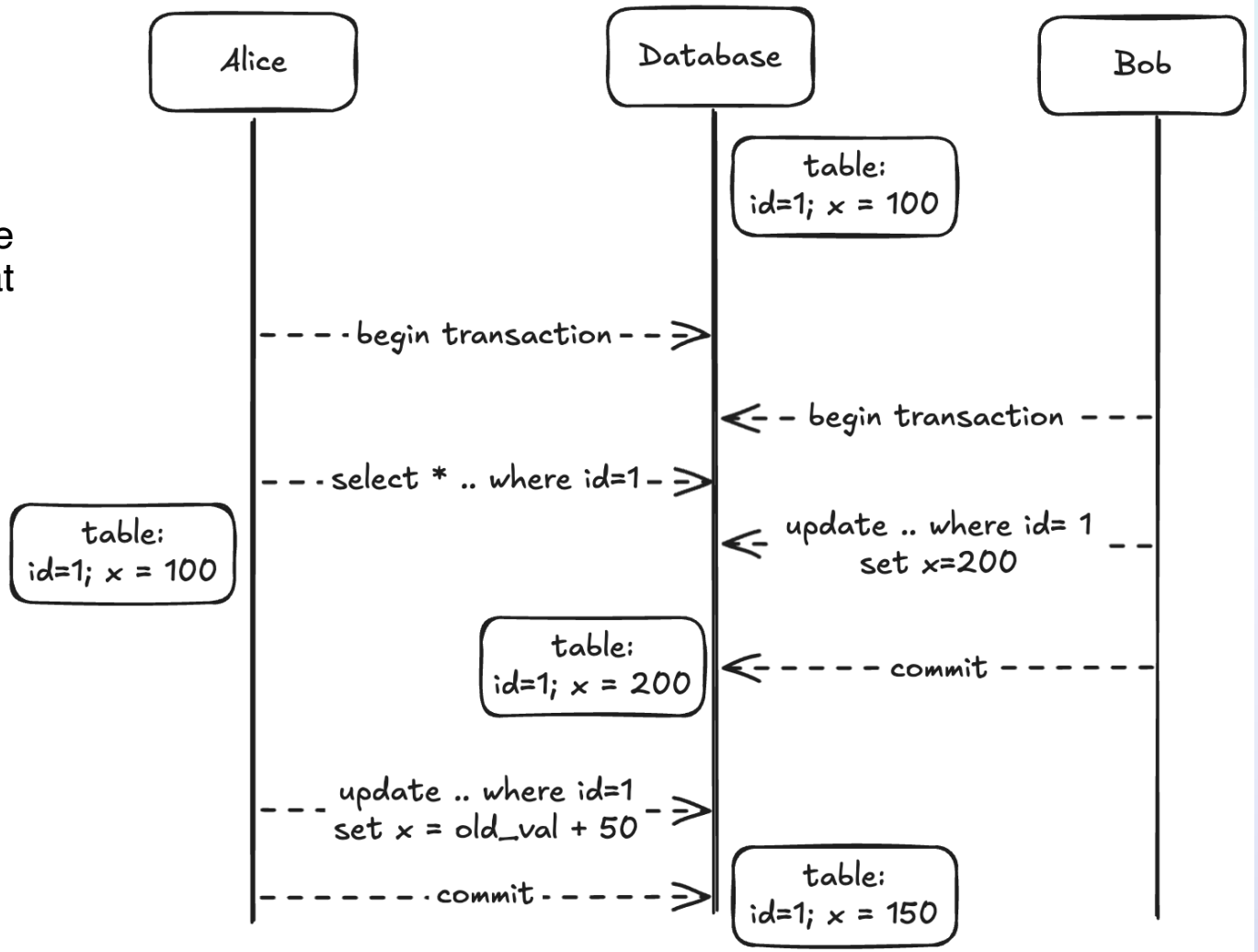


CODEUS

Anomalies & Isolation levels

Serialization anomaly

The result of successfully committing a group of transactions is inconsistent with all possible orderings of running those transactions one at a time.



Transaction management. JDBC

1. Disable auto-commit mode
2. If needed, change the isolation level (the default level depends on your database provider)
3. Execute desired statements. They all will be bundled into a single transaction
4. Either commit or rollback the transaction

```java
private final Connection connection;  8 usages

public void updateCoffeeSales(Map<String, Integer> salesForWeek) throws SQLException {  no usages
    try (PreparedStatement updateSales = connection.prepareStatement(
            sql: "update COFFEES set SALES = ? where COF_NAME = ?");
         PreparedStatement updateTotal = connection.prepareStatement(
                 sql: "update COFFEES set TOTAL = TOTAL + ? where COF_NAME = ?")) {

        connection.setAutoCommit(false);
        connection.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);
        for (Map.Entry<String, Integer> e : salesForWeek.entrySet()) {
            updateSales.setInt( parameterIndex: 1, e.getValue().intValue());
            updateSales.setString( parameterIndex: 2, e.getKey());
            updateSales.executeUpdate();

            updateTotal.setInt( parameterIndex: 1, e.getValue().intValue());
            updateTotal.setString( parameterIndex: 2, e.getKey());
            updateTotal.executeUpdate();
        }
        connection.commit();
    } catch (SQLException e) {
        log.warn(e.getMessage());
        if (connection != null) {
            try {
                log.warn("Transaction is being rolled back");
                connection.rollback();
            } catch (SQLException excep) {
                log.error(e.getMessage());
            }
        }
    }
```

CODEUS_

Savepoints. JDBC

`java.sql.Connection` API allows to control savepoints: set, rollback, release them.

```java
private final Connection connection;  15 usages

public void processLoanPayment(int loanId, BigDecimal paymentAmount, String description) {  no usages
    Savepoint loanUpdatedSavepoint = null;

    try {
        connection.setAutoCommit(false);
        connection.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);

        // ...

        loanUpdatedSavepoint = connection.setSavepoint("UPDATE_LOAN");

        // ...

        connection.commit();
    } catch (SQLException e) {
        try {
            if (loanUpdatedSavepoint != null) {
                connection.rollback(loanUpdatedSavepoint);

                // ...
            } else {
                log.error("Failed to process loan payment. No savepoint is available. Transaction is being rolled back", e);
                connection.rollback();
            }
        } catch (SQLException ex) {
            try {
                log.warn("Failed to process loan payment. Transaction is being rolled back", ex);
                connection.rollback();
            } catch (SQLException rollbackEx) {
                throw new DaoOperationException("Failed to process loan payment", rollbackEx);
            }
        }
    }
}
```

References

- Postgres documentation. Transaction Isolation
- Postgres documentation. SET TRANSACTION SQL command
- Vlad Mihalcea. A beginner’s guide to Dirty Read anomaly
- Vlad Mihalcea. A beginner’s guide to Non-Repeatable Read anomaly
- Vlad Mihalcea. A beginner’s guide to Phantom Read anomaly
- Oracle documentation. Using Transactions

Thank you

- Author: Yevhenii Savonenko
- My LinkedIn: https://www.linkedin.com/in/yevhenii-savonenko-624a32172/
- Date: April 2025
- Join Codeus community in Discord
- Join Codeus community in LinkedIn