



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

200
1822 - 2022

Advanced MySQL Concepts -Understanding Database Transactions

*Man Geni Addysg Uwch yng Nghymru
Where Higher Education Began in Wales*

www.uwtsd.ac.uk

Key Points Covered:

- What is a Transaction?
- The Need for Transactions
- ACID Properties
- Real-World Examples

What is a Transaction?

- **A Transaction is a Logical Unit of Work.**
 - It's a sequence of operations performed on a database.
 - bundle of tasks.
- **Key Principle:**
 - The entire sequence must complete successfully.
 - If any single operation fails, the entire transaction fails.

BEGIN TRANSACTION

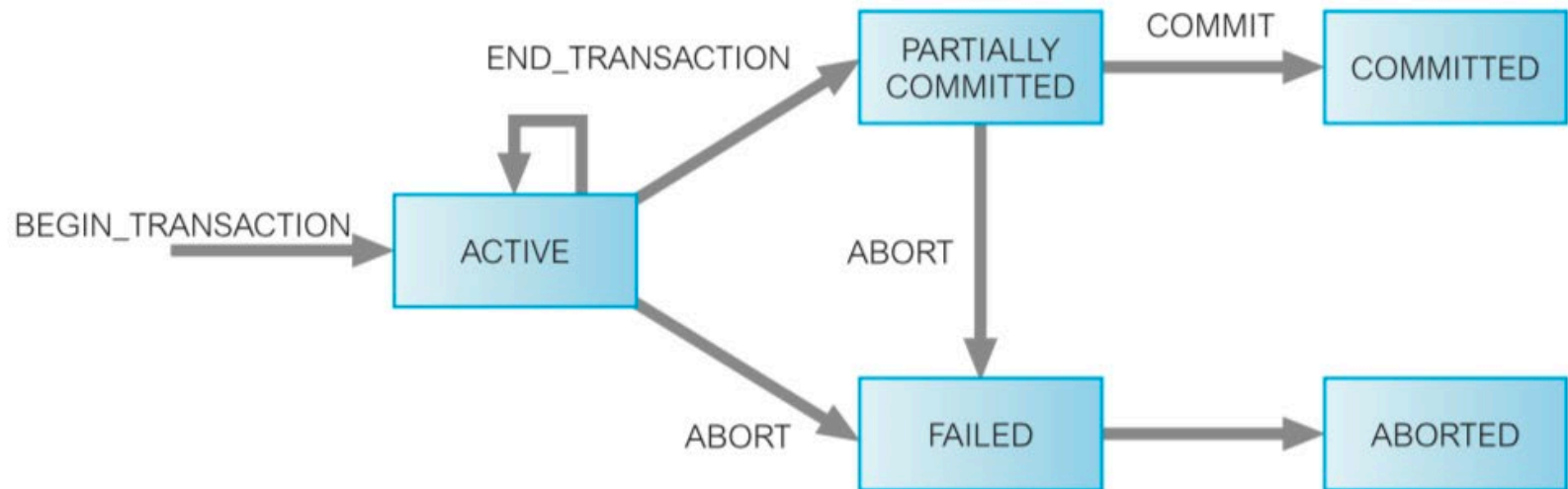
Operation 1: Deduct £100 from Account A

Operation 2: Add £100 to Account B

COMMIT TRANSACTION

Why Do We Need Transactions?

- **To Maintain Data Integrity in a Shared World.**
- **Concurrent Access:**
 - What if two users try to update the same bank account at the same time?
 - Without control, data can become corrupted (e.g., a lost update).
- **System Failures:**
 - What if the power fails *after* deducting from Account A but *before* adding to Account B?
- **Transactions** ensures our data remains accurate and consistent



Introducing ACID Properties

ACID is the standard for reliable transaction processing.

- It's an acronym that defines four crucial properties:
 - **A**tomicity
 - **C**onsistency
 - **I**solation
 - **D**urability

A is for Atomicity

The "All or Nothing" Property

- A transaction is treated as a single, indivisible unit.
- **It must execute completely or not at all.**
- If a failure occurs mid-transaction, the database rolls back (undoes) any partial changes.

Before:- X:500	Y:200
Transaction T	
T1	T2
Read(X) X:=X-100 Write(X)	Read(Y) Y:=Y+100 Write(Y)
After:- X:400	Y:300

C is for Consistency

The "Rules of the Game" Property.

- A transaction must bring the database from one **valid state** to another.
- It must not violate any defined database rules, constraints, or triggers.
 - A consistent transaction ensures this rule holds true before and after the transfer.
- If a transaction would break a rule, it is aborted.
- The total money in the system (Account A + Account B) must remain the same

Student table- student id, name, age

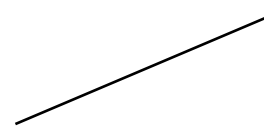
A check constraint: Age BETWEEN 16 AND 60

START TRANSACTION;

UPDATE Student SET Age = 10 WHERE StudentID = 1;

COMMIT;

Inconsistent
state



I is for Isolation

The "Mind Your Own Business" Property

- Concurrent transactions must not interfere with each other.
- Each transaction must execute as if it is the only one running on the system.

Example: people editing the same Google Doc.

D is for Durability

The "Permanent Record" Property.

- Once a transaction is successfully committed, its changes are permanent.
- The changes must survive any subsequent system failures (power loss, crash).

Committed = Saved forever.

Example: email-Once you click **Send** (COMMIT), it's gone — even if your laptop shuts down, that email is already in the recipient's inbox.

Example : Bank Transfer

Sending £500 from Alice to Bob.

[Vevox - Polls](#)

Example : Bank Transfer

Sending £500 from Alice to Bob.

- **The Transaction Steps:**

1. Check if Alice's balance $\geq$ £500.
2. Deduct £500 from Alice's account.
3. Add £500 to Bob's account.

- **How ACID Protects:**

- **Atomicity:** If the system crashes after Step 2, the entire transfer is rolled back. Alice doesn't lose her money.
- **Consistency:** The total money (Alice + Bob) remains the same. No account goes negative.
- **Isolation:** If Bob checks his balance mid-transfer, he won't see the £500 until it's fully committed.
- **Durability:** Once you get the "Transfer Successful" message, the change is permanent, even if the bank's server restarts.

Example 2: Order Processing

Scenario: A customer places an online order for 1 item.

- **The Transaction Steps:**

- Create a new order record.
- Reduce the product's inventory count by 1.
- Process the payment.

- **How ACID Protects the Business:**

- **Atomicity:** If payment fails (Step 3), the order isn't created and inventory isn't reduced. No "ghost" orders.
- **Consistency:** Ensures inventory never goes below zero. You can't sell what you don't have.
- **Isolation:** Prevents two customers from buying the last item simultaneously (a "race condition").
- **Durability:** The confirmed order and updated inventory are permanently recorded.

ACID Properties in SQL

Key SQL Commands Used:

BEGIN or START TRANSACTION

COMMIT

ROLLBACK

SAVEPOINT

Basic Bank Transfer Transaction

```
CREATE TABLE bank_accounts (  
    id SERIAL PRIMARY KEY,  
    account_holder VARCHAR(50),  
    balance DECIMAL(10,2)  
);
```

```
INSERT INTO bank_accounts (account_holder, balance) VALUES  
('Alice', 1000.00),  
('Bob', 500.00),  
('Charlie', 250.00);
```

Implement a secure money transfer

```
-- Start transaction
BEGIN;
-- Deduct from Alice
UPDATE bank_accounts SET balance = balance - 200
WHERE account_holder = 'Alice';

-- Add to Bob
UPDATE bank_accounts SET balance = balance + 200
WHERE account_holder = 'Bob';

-- Make it permanent
COMMIT;
```

Testing Atomicity with ROLLBACK

```
BEGIN;  
UPDATE bank_accounts SET balance = balance - 150  
WHERE account_holder = 'Charlie';  
-- Imagine a system crash or error happens here...  
ROLLBACK;
```

```
SELECT * FROM bank_accounts WHERE account_holder = 'Charlie';
```

Using Savepoints for Partial Rollbacks

Using Savepoints for Partial Rollbacks

Using Savepoints for Partial Rollbacks

Now, Alice wants to **transfer £300 to Bob** and **£200 to Charlie**. But you want to handle errors in stages—maybe the first transfer succeeds but the second fails, and you don't want to lose the first transfer.

- UPDATE users SET last_login = NOW() WHERE id = 1;
- SAVEPOINT before_payment;
-
- -- Step 2: Process payment (might fail)
- -- ROLLBACK TO SAVEPOINT before_payment;
- /

- Hands-On Recap & Next Steps
- What We Practiced:
 -  Basic transaction syntax (BEGIN, COMMIT, ROLLBACK)
 -  Implementing ACID properties in real scenarios
 -  Handling errors and concurrent access
 -  Using savepoints for complex workflows

Design your own transaction for a library system (check out book, update availability, create loan record).

Further Reading :

Chapter 8 Advanced SQL

Database Systems: A Practical Approach to Design, Implementation, and Management Sixth edition by • Thomas Connolly • Carolyn Begg