ScopeTreasure Courses

SQL COURSES · LESSON DB.10 · CODE SCOPE SERIES

Transactions — Ya Sab, Ya Kuch NAHI

All or nothing: the bank-transfer disaster, BEGIN / COMMIT / ROLLBACK, and how a transaction makes a half-done job impossible.

Introduction

Ali sends Sara **Rs 2,000** from his bank app. Behind the tap, two updates must happen: **take 2,000 out of Ali's account**, and **put 2,000 into Sara's**. Between those two updates, the server crashes. Ali's money is gone; Sara never received it. Two thousand rupees have evaporated — not stolen, just *half-processed*.

A **transaction** is SQL's answer: a bundle of commands that the database treats as **one indivisible job** — either *every* command lands, or *none* of them does. Ya sab, ya kuch NAHI. This lesson performs the disaster first, then the same transfer wearing its protective bundle.

(Roman Urdu: Transaction = bandha hua packet. Ya poora kaam hoga, ya bilkul nahi — aadha kaam database qubool hi nahi karti.)

Key points

- **A transaction groups commands into one job:** it starts with `START TRANSACTION;` (or `BEGIN;`), and everything after it is provisional — written in pencil, not yet ink.
- **COMMIT makes it permanent:** `COMMIT;` is the signature at the end — all the bundled changes become real, visible to everyone, at once.
- **ROLLBACK throws the bundle away:** `ROLLBACK;` undoes *every* change since the transaction began, as if none of it happened. Crash mid-transfer? On restart, the unfinished bundle is rolled back automatically — the money is back with Ali, exactly whole.

  (Roman Urdu: COMMIT = dastakhat, pakka. ROLLBACK = wapas, jaise kuch hua hi nahi.)
- **While a transaction is open, others see the old world:** your uncommitted changes are visible to you, but the rest of the system still sees the last committed state. No one ever sees a half-transfer.
- **Not everything needs a bundle:** a single INSERT or UPDATE is already atomic by itself — it is its own mini-transaction. Bundles earn their keep when a job is *several* commands that must agree: transfers, an order + its items, a seat booked + a payment recorded.
- **The habit:** any time you write two or more writes that belong to one real-world event, ask "what if it stops in the middle?" If the answer is "disaster", it is a transaction.

Code examples

Our bank's table for every example:

```
CREATE TABLE accounts (
  id INT PRIMARY KEY,
  name VARCHAR(50),
  balance INT NOT NULL
);

INSERT INTO accounts VALUES (1, 'Ali', 10000), (2, 'Sara', 3000);
```

Ali has Rs 10,000; Sara has Rs 3,000. Total in the bank: Rs 13,000 — a number that must never change during a transfer.

Example 1 — The disaster: a transfer with no bundle

```
UPDATE accounts SET balance = balance - 2000 WHERE id = 1;

-- *** server crashes here — the next line never runs ***

UPDATE accounts SET balance = balance + 2000 WHERE id = 2;

SELECT * FROM accounts;
```

What it does: The first UPDATE lands and is permanent the moment it runs (a lone command commits by itself). The crash kills the second. When the dust settles, the SELECT tells the ugly truth.

What you will see:

```
+----+------+---------+
| id | name | balance |
+----+------+---------+
|  1 | Ali  |    8000 |
|  2 | Sara |    3000 |
+----+------+---------+
2 rows in set
```

Ali is down 2,000, Sara received nothing, and the bank's total quietly dropped from 13,000 to 11,000. No error was raised; no rule was broken — each command was legal. The job was simply *two* commands pretending to be one event. That pretence is the bug.

Example 2 — The same transfer, inside a transaction

```
START TRANSACTION;

UPDATE accounts SET balance = balance - 2000 WHERE id = 1;
UPDATE accounts SET balance = balance + 2000 WHERE id = 2;

COMMIT;

SELECT * FROM accounts;
```

What it does: Both updates happen inside the bundle; COMMIT signs them together. If the crash had come between the updates this time, the whole bundle — including Ali's debit — would have been rolled back on restart, leaving 10,000 / 3,000 untouched. Half a transfer has become *impossible*, not just unlikely.

What you will see:

```
+----+------+---------+
| id | name | balance |
+----+------+---------+
|  1 | Ali  |    8000 |
|  2 | Sara |    5000 |
+----+------+---------+
2 rows in set
```

Ali 8,000, Sara 5,000, total still exactly 13,000. Money moved; none was created or destroyed on the way.

Example 3 — ROLLBACK: the deliberate rescue

```
START TRANSACTION;

UPDATE accounts SET balance = balance - 2000 WHERE id = 1;

SELECT * FROM accounts;   -- inside the bundle, I can see my own change

ROLLBACK;

SELECT * FROM accounts;   -- after the rescue
```

What it does: Suppose after the debit you notice the amount was wrong — it should have been 200, not 2,000. Inside the open transaction, your own SELECT shows Ali at 8,000. Nothing is signed yet, so ROLLBACK throws the whole bundle away; the second SELECT shows the world exactly as it was.

What you will see:

```
-- inside the transaction:
|  1 | Ali  |    8000 |
|  2 | Sara |    3000 |

-- after ROLLBACK:
|  1 | Ali  |   10000 |
|  2 | Sara |    3000 |
```

This is also the honest answer to DB.3's warning: an UPDATE without WHERE *outside* a transaction is permanent damage; inside a transaction you get one look, and one rescue, before signing.

Quick recap

- A **transaction** bundles commands into one indivisible job: **all land, or none do**.
- `START TRANSACTION` opens the bundle; `COMMIT` signs it permanent; `ROLLBACK` erases it completely.
- Crashes roll unfinished bundles back automatically — a half-transfer cannot survive.
- Bundle whenever several writes form one real-world event.

Common mistakes

- **Two writes, no bundle, "it usually works":** transfers, order + items, booking + payment — the crash always picks the worst gap. If stopping in the middle is a disaster, it is a transaction, full stop.
- **Forgetting to COMMIT:** the session ends, the bundle silently rolls back, and your "saved" data was never saved. A transaction left open is work left in pencil.

  (Roman Urdu: COMMIT bhool gaye to kaam hawa mein — database ne use pakka mana hi nahi.)
- **Bundling reads and writes carelessly over minutes:** an open transaction holds its pencil over the data; long transactions (a user "thinking" mid-checkout) block others. Keep bundles short: gather first, then BEGIN, write, COMMIT.
- **Expecting ROLLBACK after COMMIT:** once signed, there is no unsign. ROLLBACK only rescues the still-open bundle — which is exactly why Example 3 checks *before* committing.

Practice

1. Write the full transaction for a Rs 500 transfer from Sara to Ali (accounts from this lesson), with COMMIT — then state the final balances (check the total is still 13,000).
2. A shop's checkout does three writes: insert the order, insert its items, reduce product stock. Write the skeleton (START TRANSACTION ... COMMIT) with a line where a stock-shortage check would ROLLBACK instead. Why does this job deserve a bundle?
3. Explain in your own words: your UPDATE ran, your SELECT showed the change, but a colleague's screen still showed the old value — and after your connection closed, your change vanished too. What happened, in transaction language?

Practice quiz

1. What three commands open, sign and erase a transaction bundle?

- BEGIN, SAVE, DELETE
- START TRANSACTION opens it, COMMIT makes it permanent, ROLLBACK throws the whole bundle away
- OPEN, CLOSE, UNDO
- START, COMMIT, COMMIT again

Answer: START TRANSACTION opens it, COMMIT makes it permanent, ROLLBACK throws the whole bundle away

2. What happens if the system crashes mid-transfer, inside an open transaction?

- Half the transfer survives
- On restart the unfinished bundle is rolled back automatically, the money is back whole, no half-transfer survives
- The transfer completes by itself
- The database deletes both accounts

Answer: On restart the unfinished bundle is rolled back automatically, the money is back whole, no half-transfer survives

3. After COMMIT, can ROLLBACK still rescue the bundle?

- Yes, any time
- No, once signed there is no unsign; ROLLBACK only rescues the still-open bundle
- Yes, but only within one minute
- Only for single-statement transactions

Answer: No, once signed there is no unsign; ROLLBACK only rescues the still-open bundle

---

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/sql-transactions · More free tools and notes at https://scopetreasure.com