

# Transactions

by Sophia



## WHAT'S COVERED

This lesson explores the concept of transactions in a database to ensure multiple statements execute in an all-or-nothing scenario, in two parts. Specifically, this lesson will cover:

1. [Introduction](#)
2. [Examples](#)

## 1. Introduction

**Transactions** are a core feature of every database system. The purpose of a transaction is to combine multiple SQL statements into a scenario that would execute all of the statements or none of them. Each individual SQL statement within a transaction is not visible to other concurrent transactions in the database, and they are not saved to the database unless all of the statements in the transaction execute successfully. If any statement in the transaction fails to execute, the entire transaction is cancelled, and none of its statements execute.

A database management system's **ACID** properties guarantee the integrity and reliability of transactions by ensuring atomicity, consistency, isolation, and durability.

- Atomicity ensures that transactions will be handled as indivisible units, either completed or rolled back if they fail.
- The consistency of the database ensures that data integrity and business rules are maintained when transactions take the database from one consistent state to another.
- The goal of isolation is to ensure that each transaction takes place independently from the other, preventing interference and ensuring data accuracy.
- Data persistence is ensured by durability, which ensures that once a transaction is committed, its modifications will be permanently stored in the database as long as the system continues to operate.

Together, these properties ensure that data transactions are reliable and correct, which is crucial for maintaining database integrity. You will learn more about ACID properties in upcoming lessons.



## TERMS TO KNOW

## Transaction

A sequence of statements that are performed as a group; if one fails to execute, they all fail.

## ACID

An acronym for the four properties that ensure data validity: Atomicity, Consistency, Isolation, and Durability. Transactions are database operations that satisfy the ACID properties.

---

# 2. Examples

Let us take a look at a scenario in which a transaction would be necessary. James has gone to an online computer store and purchased a new computer for \$500. In this transaction, there are two things to track. The first is the \$500 being transferred from James to the store, and the second is the computer being deducted from the store inventory and transferred to James. The basic SQL statements would look something like the following:

```
UPDATE customer_account
SET balance = balance - 500
WHERE account_id = 1000;
```

```
UPDATE store_inventory
SET quantity = quantity - 1
WHERE store_id = 5 AND product_name = 'Computer';
```

```
INSERT INTO customer_order(account_id, product_name, store_id, quantity, cost)
VALUES (1000, 'Computer', 5, 1, 500);
```

```
UPDATE store_account
SET balance = balance + 500
WHERE store_id = 5;
```

As you can see, multiple SQL statements are needed to accomplish this operation. Both James and the store would want to be assured that either all of these statements occur or none of them do. It would not be acceptable if \$500 was deducted from James's account, the inventory for the store had the computer removed, and then there was a system failure. This would mean that James would not get the order, and the store would not get the \$500 that James had paid. We need to ensure that if anything goes wrong at any point within the entire operation, none of the statements that have been executed so far will take effect. This is where the use of a transaction is valuable.

To set up a transaction, we need to start with the **BEGIN** command, have our list of commands, and then end with **COMMIT**. Similar to our prior example, consider the following:

```
BEGIN;
```

```
UPDATE customer_account
SET balance = balance - 500
WHERE account_id = 1000;
```

```
UPDATE store_inventory
SET quantity = quantity - 1
WHERE store_id = 5 AND product_name = 'Computer';
INSERT INTO customer_order(account_id, product_name, store_id, quantity, cost)
VALUES (1000, 'Computer', 5, 1, 500);
```

```
UPDATE store_account
SET balance = balance + 500
WHERE store_id = 5;
COMMIT;
```

A transaction can contain a single statement or multiple statements; there is no limit. Note that each SQL statement needs to end with a semicolon to separate each out individually.

PostgreSQL and other databases treat each executed SQL statement as if it were an individual transaction. If we do not include a **BEGIN** command, then each statement is treated as a separate transaction; that is the way the statements we have run in this course so far have operated.



Your turn! Open the SQL tool by clicking on the **LAUNCH DATABASE** button below. Then, enter in one of the examples above and see how it works. Next, try your own choices for which columns you want the query to provide.




### **BEGIN**

A statement that marks the beginning of a transaction.

### **COMMIT**

A statement that marks the end of a transaction and makes the transaction's results permanent.

 **SUMMARY**

In this lesson, in the **introduction** you learned that database transactions are an essential part of ensuring data integrity and consistency within a system. Transactions are sequences of database operations that are treated as single, indivisible units of work. Database transactions are essential to maintain data accuracy and reliability, especially in multiuser environments where concurrent access can result in conflicts.

In the **examples**, you also learned that transactions are based on the ACID properties of atomicity, consistency, isolation, and durability. Transactions are atomic in nature—they are either fully completed, bringing the system to a consistent state, or fully aborted, leaving it unchanged. Consistency ensures that the database transitions from one valid state to another according to defined rules and constraints. By isolating concurrent transactions, anomalies such as the lost update problem or dirty reads are prevented, while durability ensures that a transaction's effects will remain permanent and persistent regardless of system failures once it is committed. During the execution of a transaction, it can be initiated, completed, and terminated, and its effects persist. This mechanism ensures data integrity in the event of hardware failures, software crashes, or other unexpected disruptions. A transactional system ensures database operations' reliability, consistency, and security by providing a structured framework for managing simultaneous data manipulation.

Source: THIS TUTORIAL WAS AUTHORED BY DR. VINCENT TRAN, PHD (2020) AND FAITHE WEMPEN (2024) FOR SOPHIA LEARNING. PLEASE SEE OUR [TERMS OF USE](#).



## TERMS TO KNOW

### ACID

An acronym for the four properties that ensure data validity: Atomicity, Consistency, Isolation, and Durability. Transactions are database operations that satisfy the ACID properties.

### BEGIN

A statement that marks the beginning of a transaction.

### COMMIT

A statement that marks the end of a transaction and makes the transaction's results permanent.

### Transaction

A sequence of statements that are performed as a group; if one fails to execute, they all fail.