MySQL, MariaDB, PostgreSQL, Oracle Database, and Microsoft SQL Server are the most popular relational database technologies. All these RDBMSs (Relational Database Management Systems) share one common aspect: they are all transactional databases. In other words, they all rely on database transactions.

In detail, database transactions play a key role in the functioning of a relational database. This is why it is so important to know what a database transaction is and how it works. Follow this guide and become an expert on database transactions!

What is a Database Transaction?

A database transaction is a logical unit that generally includes several low-level steps. If one step of the transaction fails, the whole transaction fails. A database transaction is used to create, update, or retrieve data.

Think of a database transaction as a series of operations performed within a DBMS. The transaction system ensures that the data in the database always remains in a reliable and consistent state.

- **If a transaction is successful**, the data in the database is updated as described in the instructions contained in the transaction. This is called a “commit.”

executed. This operation is called a “rollback.”

In other terms, a database transaction ends with a commit or rollback. This ensures that database transactions are Atomic, Consistent, Isolated, and Durable. These are commonly known as the ACID properties. Let’s learn more about them.

ACID Properties in DBMS

The ACID acronym defines the four properties every database transaction must have to ensure data integrity in case of errors or failures. In a DBMS, these properties are:

- Atomicity: This property ensures that all operations within a transaction complete successfully, or do not complete at all. This means that a transaction is indivisible. Atomicity prevents a transaction from stalling and prevents partial database updates.
- Consistency: Any transaction leaves the database in a consistent state, regardless of the outcome of the transaction. If the database was in a consistent state before the transaction, it must remain consistent after the transaction is executed.
- Isolation: Each transaction has access to an isolated version of the database. Data used in transactions that have not yet been completed cannot be modified by other transactions.

data to disk, the data will be updated when the system returns to service.

Keep also in mind that ensuring ACID properties comes at a performance cost for a DBMS. So, not all database management systems fully support ACID. Plus, some DBMSs, such as MySQL, allow you to disable ACID for speed by changing a parameter from 0 to 1.

Now that we know what ACID properties are and learned a little about database transactions, you will see some of them in action.

How To Define a Database Transaction

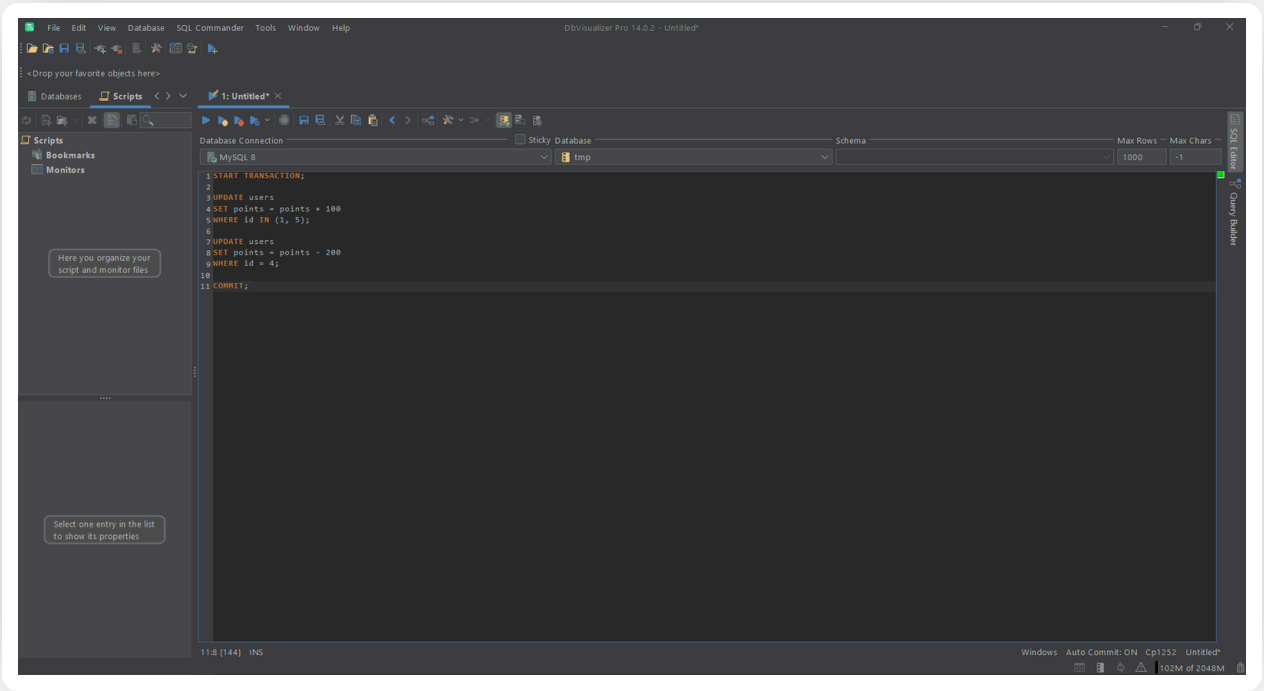
A database transaction is generally defined by a set of instructions wrapped by two keywords. This is especially true when it comes to transactions in SQL. These two keywords mark the beginning and end of the transaction, respectively. For example, in PostgreSQL and MySQL these are: `START TRANSACTION` and `COMMIT`. In SQL server these are: `BEGIN TRANSACTION` and `COMMIT TRANSACTION`.

Let's assume you want to remove 200 points from user 4, and distribute them equally among user 1 and 5. This is how you can achieve this with a PostgreSQL database transaction:

COPY

```
1
    -- initializing the transaction
3   START TRANSACTION;
```

```
8   -- removing 200 points from user 4
9   UPDATE users
10  SET points = points - 200
11  WHERE id = 4;
12  -- commiting the change (or rolling it back later in case
    of failure)
13  COMMIT;
```



↑ Running the transaction query in DbVisualizer

the database crashes after the first update query, then the database data would be inconsistent. Users 1 and 5 would find themselves with 100 more points than they should have. Luckily, the transaction ensures that the entire logical operation is performed atomically. So, the database data will remain consistent even in case of failures.

Now you know how to write a transaction in SQL, but you still have to understand how transactional databases execute transactions. Let's learn more about this!

States of a Database Transaction

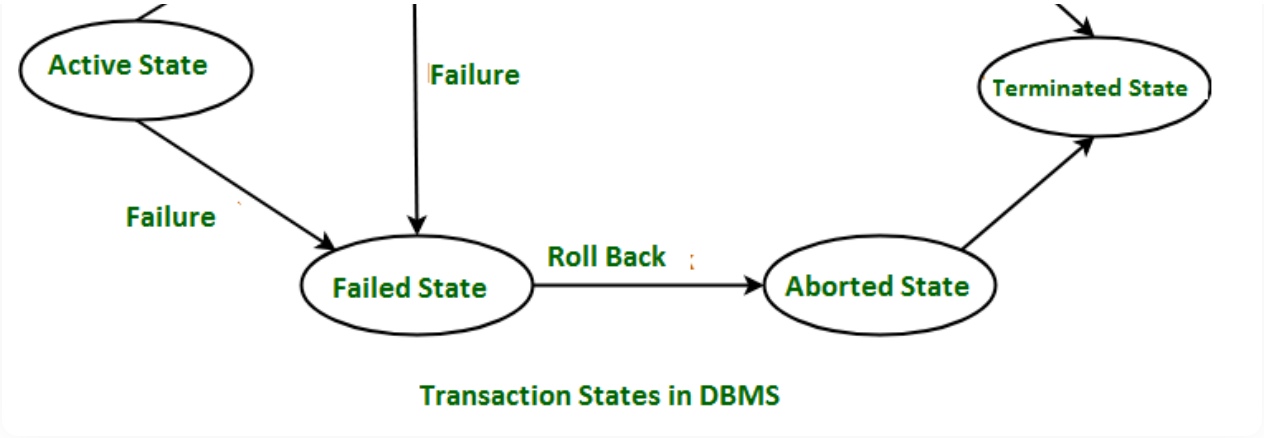
In the case of a transactional database, the life cycle of a transaction can be described by the following four steps:

1. **The transaction begins**: The transactional database prepares everything required to execute the transaction.
2. **The queries defined in the transaction are executed**: This is when data manipulation takes place.
3. **If no errors occur, the transaction is committed**: The transaction ends successfully.
4. **If an error occurs, it rolls back the transaction**: The transaction ends with failure and any query executed before it failed is reversed.

manager, then you've got to try DbVisualizer. It connects to nearly any database.

Specifically, a transaction in DBMS can have the following five states:

State	Transaction types
Active	This is the initial state of every transaction in a database. This state means that the transaction is being executed and can perform read and write operations.
Partially Committed	A transaction enters this state after performing its final operation/query.
Committed	A transaction is in this state if it has performed all its operations successfully. The effects of the transaction are now permanent in the database system.
Failed	A transaction enters this state if any transaction fails or if the transaction is aborted before completing. A failed transaction cannot proceed further.
Terminated	It is the final state of each transaction. It means that the transaction is finished, either successfully or not.



↑ State transition diagram for a database transaction

Conclusion

Here you have learned everything you need to know about database transactions. In detail, you have had the opportunity to see what a database transaction is, how to define one through an example, and what ACID properties are.

Database transactions are a powerful tool that allows you to build an atomic query composed of several sub-queries and sub-operations. This means that you should use transaction queries carefully. Also, you need to optimize them for good performance. In other words, such a powerful tool requires an advanced database client that supports you with query optimization features and much more, such as DbVisualizer. Try DbVisualizer for free!