**#9 Transactions**

In the Database, a transaction is a group of operations that are handled as one unit of work. In practice, this means that you may have many operations**, and if any of these operations fails, the entire group of operations is treated as fail, and the database is restored to its state before the group of operations was started.**

For example, imagine a financial application where several tables need to be updated for a given transaction, if any of these operations fails, corresponding data in the other tables would be out of sync and invalid. By combining all of these actions into a single transaction, the state of the database will be automatically rolled back to a valid state if any of the individual operations fails.

Transactions are also used to ensure that concurrent operations result in a state as if they were handled separately and sequentially. In other words, if your database is used by many clients at the same time, and they're all conducting similar complex operations grouped into transactions, those transactions will affect the database as if each transaction were completed separately.

**Transactions can also improve performance**, sometimes radically, for example, if you have a lot of rows to insert into a table, or a set of tables, each of these inserts takes time to write to the storage device, when making individual writes, the database system uses resources to ensure that each row has been successfully committed to storage before the next write begins. When you make a group of inserts to your table as a transaction, the database system can perform many write operations together, significantly reducing the overhead associated with writing to physical media.

Generally, transactional operations could improve reliability and performance for larger or more complex operations.

**Example:**



BEGIN TRANSACTION;

INSERT INTO widgetSales ( inv\_id, quan, price ) VALUES ( 1, 5, 500 );

UPDATE widgetInventory SET onhand = ( onhand - 5 ) WHERE id = 1;

END TRANSACTION;

BEGIN and END keywords are used in SQLite but for other applications it could be differenct

BEGIN TRANSACTION;

INSERT INTO widgetInventory ( description, onhand ) VALUES ( 'toy', 25 );

ROLLBACK;

SELECT \* FROM widgetInventory;

The ROLLBACK will simply discard the changes that were made since the beginning of the transaction.

As the name suggests, the table will be rollbacked to it previous version

**Transactions are also used for performance.**

Let’s check this out…

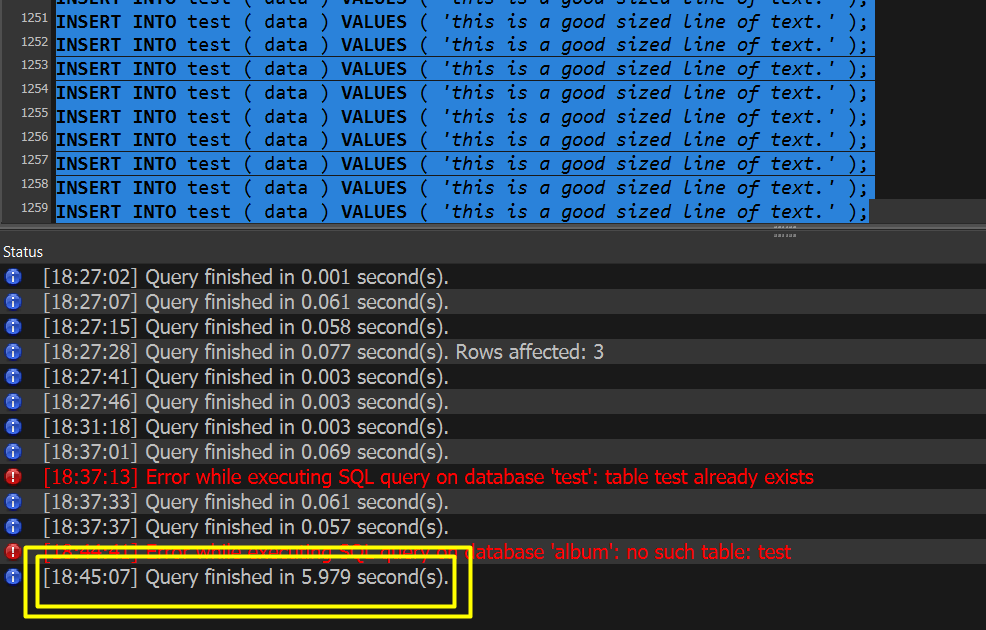
Creating a table called test:

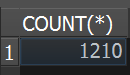
CREATE TABLE test ( id INTEGER PRIMARY KEY, data TEXT );

Now, to test the performance difference, lets add 1000(yes, thousand) records into the table ‘test’.

INSERT INTO test ( data ) VALUES ( 'this is a good sized line of text.' ); {X 1000}

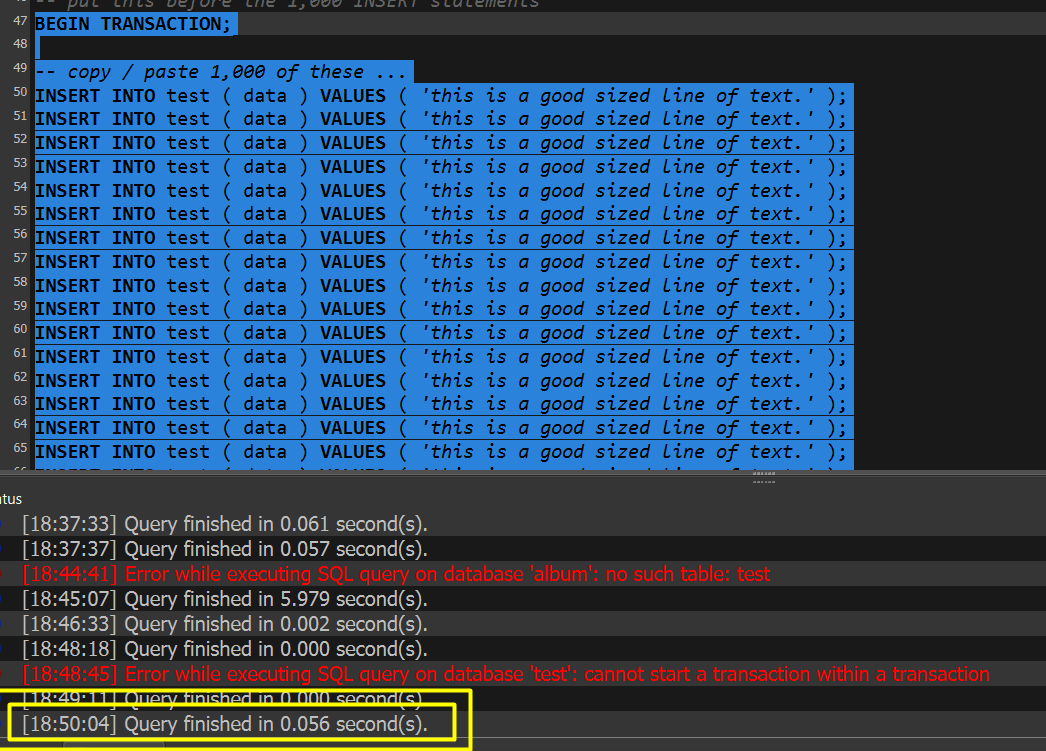
First, we are going to add records without using the Transactions, it will be just by doing simple INSERT statement.



It took almost 6 seconds!!  to insert 1210 records.

Next page, same thing using Tranactions.

Now, let’s try with Transactions!.



**0.056 seconds only! Awesome**

How do transactions improve performance?

* They speed up execution by making reading and writing more efficient.

You do not need to open and close data files as many times.

Why would you use transactions instead of individual instructions?

* to maintain data integrity and improve performance

Transactions allow more efficient processing, and prevent leaving the data in an inconsistent state.