Transaction

This lesson defines and explains the concept of a transaction in the context of relational databases.

Transaction

A common online definition of a database transaction reads *"A transaction symbolizes a unit of work performed within a database management system against a database, and treated in a coherent and reliable way independent of other transactions."* In simpler terms, think about taking a two-legged flight from Mumbai to New York. The first airline takes you from Mumbai to Dubai and the second takes you from Dubai to New York. The two flight legs represent portions of a transaction. You must travel both legs to reach your destination, but if immigration forbids you from taking the second leg of the flight you'd want to return back to Mumbai and not remain stranded at Dubai. A database transaction is similar. You either want all the actions within the transaction to complete or none at all. You don't want a transaction to complete halfway through and then abort.

Need for Transactions

One may wonder why we need transactions. Well, whenever multiple users are interacting with a database it is possible for the actions of one user to interleave with another user and bring the data in an inconsistent state. The classic example is transferring funds from one bank account to another. Say the bank application wants to add 100 dollars to your bank account with a balance of 500 dollars. The application reads your balance, adds 100 dollars to it and then updates the new amount 600 dollars in the database against your username. It could happen that between the time the application reads and updates the new amount, you make an ATM withdrawal of 200 dollars. The application has already read your balance and doesn't know that 200 dollars has been withdrawn and mistakenly

writes your balance as 600 dollars to the database.

The application layer could avoid this mistake if it performed the two tasks of reading and then updating your balance atomically, or in other words as a transaction. Transactions allow you to batch together SQL statements as an indivisible set that either succeeds or has no effect on the database.

Syntax to Start & Commit a Transaction

> START TRANSACTION;
>
> ** – SQL statements
>
> COMMIT;

Syntax to Start & Rollback a Transaction

> START TRANSACTION;
>
> ** – SQL statements
>
> ROLLBACK;

> Connect to the terminal below by clicking in the widget. Once connected, the command line prompt will show up. Enter or copy and paste the command **./DataJek/Lessons/39lesson.sh** and wait for the MySQL prompt to start-up.

```
-- The lesson queries are reproduced below for convenient copy/paste into the terminal.

-- Query 1
START TRANSACTION;
UPDATE Actors
SET Id = 100
WHERE FirstName = "Brad";
COMMIT;

-- Query 2
START TRANSACTION;
```

```
UPDATE Actors
SET Id = 200
WHERE FirstName = "Tom";

ROLLBACK;

-- Query 3
SHOW ENGINES;
```

Terminal

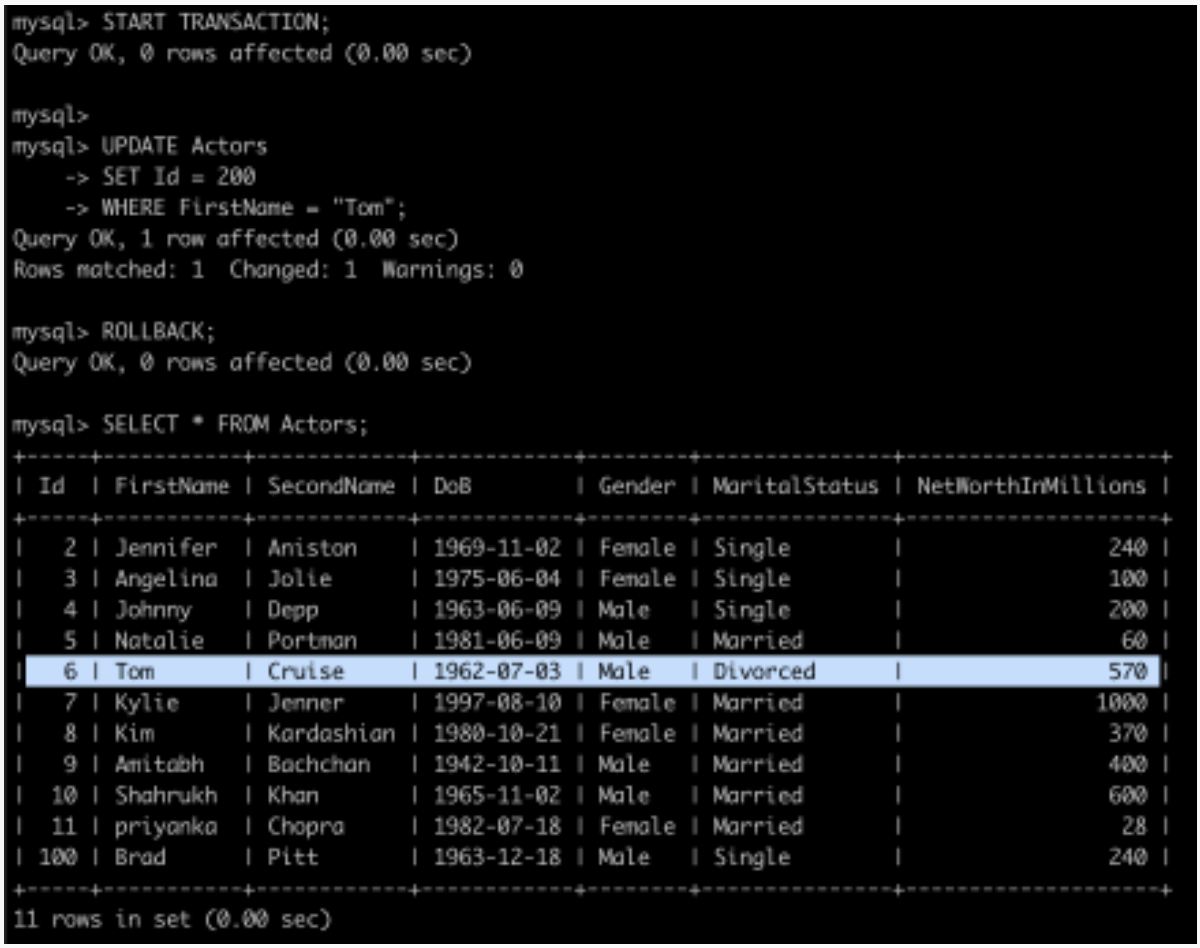
1. MySQL operates in AUTOCOMMIT mode so whatever commands we issue at the MySQL prompt are committed and treated as a transaction. However, we can explicitly start a transaction and then either proceed to commit it or roll it back. Consider the following sequence of commands:

```
START TRANSACTION;

UPDATE Actors
SET Id = 100
WHERE FirstName = "Brad";

COMMIT;
```

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> UPDATE Actors
    -> SET Id = 100
    -> WHERE FirstName = "Brad";
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql>
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> UPDATE Actors
    -> SET Id = 100
    -> WHERE FirstName = "Brad";
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM Actors;
+-----+-----------+------------+------------+--------+---------------+--------------------+
| Id  | FirstName | SecondName | DoB        | Gender | MaritalStatus | NetWorthInMillions |
+-----+-----------+------------+------------+--------+---------------+--------------------+
|   2 | Jennifer  | Aniston    | 1969-11-02 | Female | Single        |                240 |
|   3 | Angelina  | Jolie      | 1975-06-04 | Female | Single        |                100 |
|   4 | Johnny    | Depp       | 1963-06-09 | Male   | Single        |                200 |
|   5 | Natalie   | Portman    | 1981-06-09 | Male   | Married       |                 60 |
|   6 | Tom       | Cruise     | 1962-07-03 | Male   | Divorced      |                570 |
|   7 | Kylie     | Jenner     | 1997-08-10 | Female | Married       |               1000 |
|   8 | Kim       | Kardashian | 1980-10-21 | Female | Married       |                370 |
|   9 | Amitabh   | Bachchan   | 1942-10-11 | Male   | Married       |                400 |
|  10 | Shahrukh  | Khan       | 1965-11-02 | Male   | Married       |                600 |
|  11 | priyanka  | Chopra     | 1982-07-18 | Female | Married       |                 28 |
| 100 | Brad      | Pitt       | 1963-12-18 | Male   | Single        |                240 |
+-----+-----------+------------+------------+--------+---------------+--------------------+
11 rows in set (0.00 sec)
```

2. Now we'll start a transaction but roll it back midway and observe the changes that don't take place.

```sql
START TRANSACTION;

UPDATE Actors
SET Id = 200
WHERE FirstName = "Tom";

ROLLBACK;
```

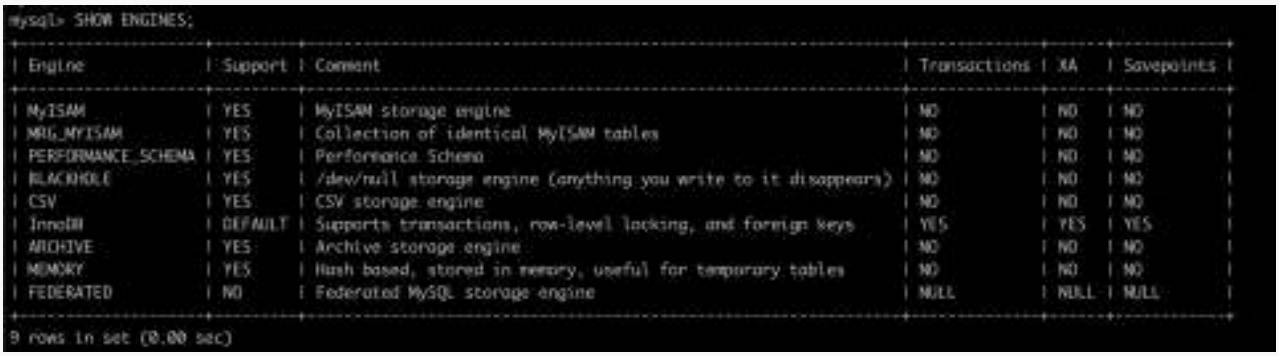


You can observe from the screenshot that once we roll back the transaction, the changes we intended don't take place.

Locking

Transactions are important so that users don't step on each other's feet when interacting with the database. The database system has to block other transactions from executing while another is in progress and targets the same tables. Also, there are some storage engines that don't support transactions. Irrespective of whether transactions are supported or not, the database system has to implement some sort of a locking mechanism to protect tables from being modified by multiple users at the same time. There are various levels of sophistication built into database engines on how to handle concurrent users. For instance, in the case of MyISAM, the entire table gets locked, while InnoDB provides granular locking at the row level. You can view all the available types of storage engines on your version of MySQL as follows:

```
SHOW ENGINES;
```



```
mysql> SHOW ENGINES;
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
| Engine             | Support | Comment                                                        | Transactions | XA   | Savepoints |
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
| MyISAM             | YES     | MyISAM storage engine                                          | NO           | NO   | NO         |
| MRG_MYISAM         | YES     | Collection of identical MyISAM tables                          | NO           | NO   | NO         |
| PERFORMANCE_SCHEMA | YES     | Performance Schema                                             | NO           | NO   | NO         |
| BLACKHOLE          | YES     | /dev/null storage engine (anything you write to it disappears) | NO           | NO   | NO         |
| CSV                | YES     | CSV storage engine                                             | NO           | NO   | NO         |
| InnoDB             | DEFAULT | Supports transactions, row-level locking, and foreign keys     | YES          | YES  | YES        |
| ARCHIVE            | YES     | Archive storage engine                                         | NO           | NO   | NO         |
| MEMORY             | YES     | Hash based, stored in memory, useful for temporary tables      | NO           | NO   | NO         |
| FEDERATED          | NO      | Federated MySQL storage engine                                 | NULL         | NULL | NULL       |
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
9 rows in set (0.00 sec)
```

As you can observe from the output there are nine types of database engines available and by default, tables get created as InnoDB type. You can also see from the output which engines support transactions and which don't. MyISAM doesn't support transactions that InnoDB does. There are pros and cons of using each type. InnoDB supports fine-grained locking at the cost of slower performance but allows multiple users to make modifications to a table at the same time. On the other hand, MyISAM places whole-table locks to handle multiple users which is simpler and faster but reduces concurrency. We'll end our general discussion on transactions and locking here but there are many other nuances to this topic that are out of scope for an introductory course on MySQL. The takeaway is to be cognizant that the type of storage engine you choose for your tables has consequences on the performance of your overall application. As the number of users of your application increase you'll need to use more sophisticated storage engines that are used to handle the high concurrent load.