Transactions

One or more SQL statements that form a logical unit of work.

- Any action that reads/writes (including deletes) data from DB.
- MySQL: Only DML statements: (select, update, insert, delete ...), affect "data."
- NOT DDL statements: (create, drop, alter ...), affect db structure.

The statements in a transaction will be executed as an atomic unit of work in the database. Either the results of **all** of the statements will be applied to the database, or **none** of the statements will have results posted to the database.

ANSI (American National Standards Institute) SQL database transactions support:
Two SQL statements: COMMIT and ROLLBACK

ANSI standards require: when a transaction sequence is initiated by a user or an application program, it must continue through all succeeding SQL statements until one of four events occurs:

1. COMMIT statement reached
2. ROLLBACK statement reached
3. End of program is successfully reached (equivalent to COMMIT)
4. Program abnormally terminated (equivalent to ROLLBACK)

General syntax, **MySQL** commands for transactions (require START TRANSACTION or BEGIN):

1. START TRANSACTION (or BEGIN) ("end" is either commit or rollback)
2. COMMIT: Makes changes permanent
3. ROLLBACK: Cancels changes since last COMMIT

Start Transaction vs. Begin Work - Syntax Note (From the MySQL manual):
http://dev.mysql.com/doc/refman/5.5/en/commit.html

BEGIN and BEGIN WORK are supported as aliases of START TRANSACTION for initiating a transaction. **START TRANSACTION is standard SQL syntax and is the recommended way to start an ad-hoc transaction.**

BEGIN (transaction) vs. BEGIN ... END compound statement:
The BEGIN statement differs from the use of the BEGIN keyword that starts a BEGIN ... END compound statement. The latter does not begin a transaction. See Section 13.6.1, "BEGIN ... END Compound-Statement Syntax".
http://dev.mysql.com/doc/refman/5.5/en/begin-end.html

Within all stored programs (stored procedures and functions, triggers, and events), the parser treats BEGIN [WORK] as the beginning of a BEGIN ... END block. Therefore, **Begin a transaction in this context with START TRANSACTION instead.**

ACIDS-compliant DBMS:

1) ATOMICITY:

- Each transaction treated as indivisible unit.
- All statements within transaction must be successful for transaction to be considered successful.
- If transaction failure, system returned to pre-transaction (consistent) state.

Example: following statements treated as one logical unit of work

```
START TRANSACTION;
  select ...
  update ...
  insert ...
  delete ...
COMMIT;
```

2) CONSISTENCY:

- Once transaction completed, system must be in consistent state.
- If any integrity constraints fail (e.g., domain constraints, entity/referential integrity constraints, etc.). Transaction aborted (rolled back).

Example (showing BEGIN)**:** if any statement fails, entire transaction aborted

```
BEGIN;
  select ... (uses incorrect syntax)
  update ...
  insert ... (uses incorrect data type)
  delete ...
COMMIT;
```

3) ISOLATION:

- Changes made by transaction invisible to other transactions (users) while in progress.
- Data used for one transaction cannot be used by another transaction until first transaction completed.

Example:

Connection 1:

```
START TRANSACTION;
insert into transaction (id) values (1);
select * from transaction; -- Connection 1 sees new data.
```

Connection 2:

```
SELECT * FROM transaction; -- Connection 2 does NOT (prior to COMMIT)
```

4) DURABILITY:

- Changes to the database persist.
- If transaction committed cannot be rolled back.

Example: Committed statements persist

```
START TRANSACTION;
  select ...
  update ...
  insert ...
  delete ...
COMMIT;
ROLLBACK;  (cannot rollback, even if power failure after COMMIT)
```

5) SERIALIZABILITY:
"Ques" all transactions to occur "serially" (sequentially), in order of access
DBMS Scheduler: manages concurrency control to ensure serializability of transactions (lock/unlock data)

Note:

- Isolation and serialization are nonissues in single-user database systems.
- Transaction log: records necessary information to process transaction, if interrupt/power failure.
- Engine will read logs on next startup and commit any remaining transactions.

End of Transaction:
1) COMMIT;
2) ROLLBACK;
3) Program ends successfully (equivalent to COMMIT)
4) Program abnormally terminates (equivalent to ROLLBACK)

MySQL

Example:

```
START TRANSACTION;
select * from user;
UPDATE user SET lname='Jones' WHERE uid=1;
select * from user;
COMMIT;
```

-- demo transactions do not work on DDL statements

```
drop table if exists transaction;
show tables;

START TRANSACTION;
create table if not exists transaction (id int) engine=innodb;
ROLLBACK;
show tables; -- transaction table still exists
```

Transaction Demo:

```
-- Transaction Example (test db):
use test;
show tables;

drop table if exists transaction;
create table transaction (id int) engine=innodb;
```

Connection 1:

```
START TRANSACTION;
insert into transaction(id) values (1);
select * from transaction;
rollback; -- w/o commit insert NOT permanent
select * from transaction;
-- commit;  (commit is commented out for now
```

Connection 2:

```
SELECT * FROM transaction; -- Connection 2 does NOT (prior to COMMIT)
```

MS SQL Server

Example:

```
-- *cannot* use "start" in SQL Server, must us BEGIN
BEGIN TRANSACTION;
select * from applicant;
UPDATE applicant SET app_lname='Jones' WHERE app_id=1;
select * from applicant;
COMMIT;
```

Example (maintain data integrity):

```
DECLARE @rtpID tinyint;
BEGIN TRY
    BEGIN TRANSACTION;

    -- demo with good and bad data
    INSERT INTO dbo.room_type
    (rtp_name, rtp_notes)
    VALUES
    ('Grand Room', NULL);

    -- @@IDENTITY: (global variable) last value inserted into IDENTITY column
    SET @rtpID = @@IDENTITY;

    -- rtp_id (fk) must maintain data integrity with parent table
    INSERT INTO dbo.room
    (prp_id, rtp_id, rom_size, rom_notes)
    VALUES
    (3,@rtpID, '40'' x 60''', 'Big Room!');

    COMMIT TRANSACTION;
    PRINT 'Successful!';
    select * from room;

END TRY
BEGIN CATCH
    ROLLBACK TRANSACTION;
    PRINT 'Rolled back!';
END CATCH;
```