



Transaction Management

**Ch 6.6.1 – 6.6.3
and 18.1-18.4**

Abdu Alawini

University of Illinois at Urbana-Champaign
CS411: Database Systems

Learning Objectives

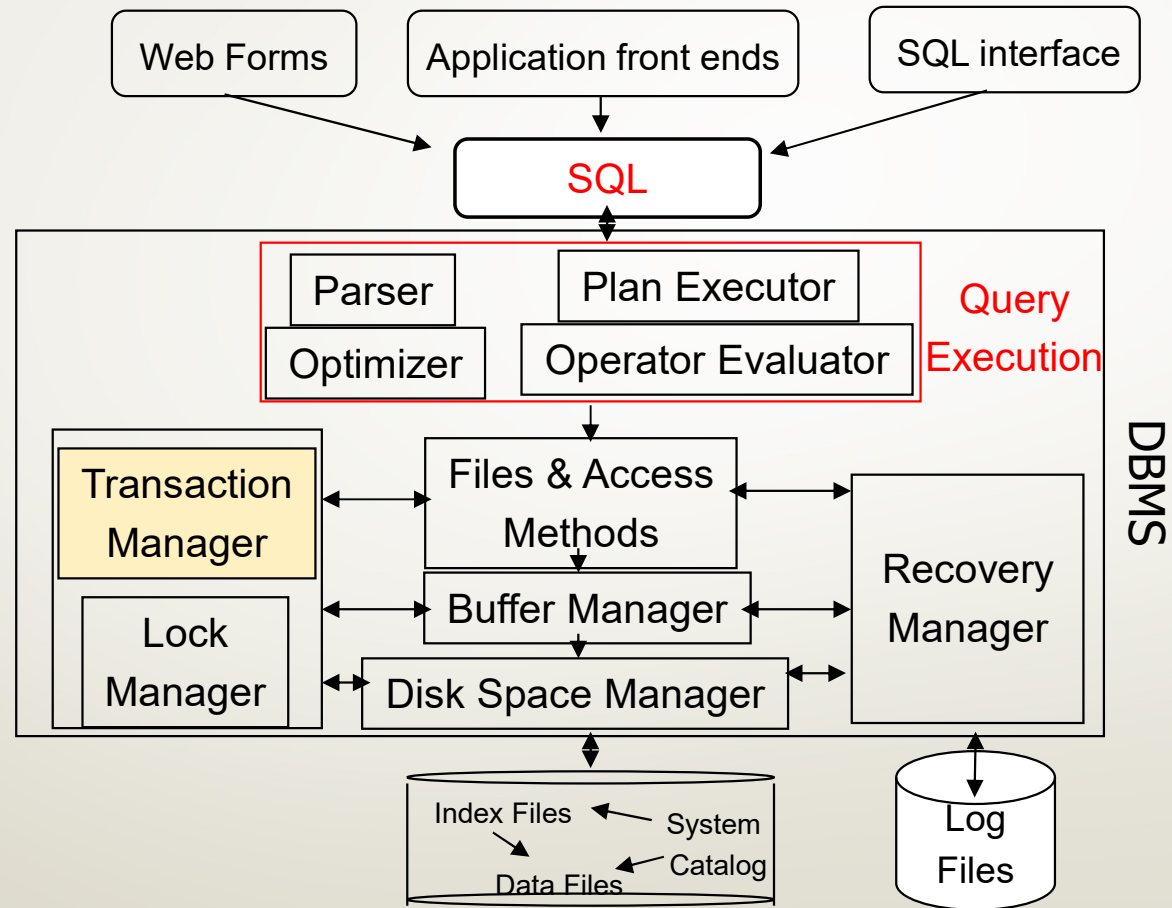
After this lecture, you should be able:

- Describe transactions, ACID properties and issues related to concurrent execution of transactions.
- Describe how database systems implement isolation levels

Outline

- Transactions and ACID properties
- Transactions in SQL
- Isolation levels
- Isolation Levels and Locking

DBMS Architecture



Motivating “Transactions”

- We’ve learned how to interact with DB using SQL.
- We assumed that:
 - each **operation** (e.g., UPDATE ... SET ... WHERE) is **executed one at a time**.
 - One operation executes, perhaps changes the DB state, then next operation executes. (**ISOLATION**)
 - each operation is executed in entirety or not executed at all. (**ATOMIC**)

Motivating “Transactions”

- Complications arise if these assumptions are violated
 - **multiple operations** acting on the same table simultaneously?
 - **system crash** in the middle of an operation
 - e.g., half the tuples have been updated the others not

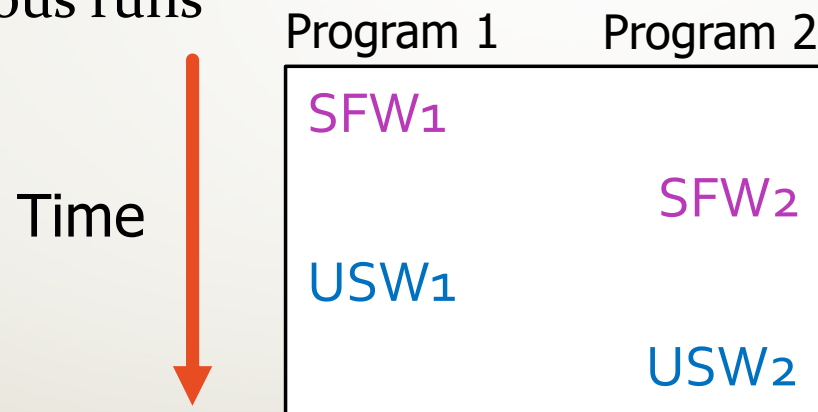
Example 1: flight seat selection

- Program:

A. Check if seat is available : SELECT FROM WHERE (SFW)

B. Book seat (change availability to “occupied”): UPDATE SET WHERE (USW)

Two simultaneous runs



Two executions of the same “UPDATE ... SET ... WHERE” leads to seat being double-booked

Example 1: lesson

- Group the SFW (which retrieved seat availability) and the USW (which reserved a seat) into one **TRANSACTION**.
- Transaction is a sequence of statements that are considered a “unit of operation” on a database
- Either user 1’s transaction executes first and then user 2’s transaction, or the other way, but not in parallel. *Serializability of transactions*.

Example 2: bank inter-account transfer

Problems can also occur if a crash occurs in the middle of executing a transaction:

Transfer

read(X.bal)

read(Y.bal)

{X.bal = X.bal - \$100}

write(X.bal)

CRASH 

Y.bal = Y.bal + \$100

write(Y.bal)

Need to guarantee that the write to X does not persist (**ABORT**)

- Default assumption if a transaction doesn't commit

Example 2: lesson

- Steps 1 and 2 must be done as one unit. *Atomically.*
- Either they both execute, or neither does.
- *Transactions must be atomic.*

Transactions

Definition: **Transaction** is a sequence of read and write operations on the database with the property that either all actions complete or none completes.

a **transaction** may

- Succeed: the effects of all write operations persist (**COMMIT**)
 - Or fail, no effects persist (**ABORT or ROLLBACK**)
-
- These guarantees are made despite concurrent activity in the system, and despite failures that may occur

Transaction Manager

The part DBMS that ensures a transaction is executed as expected.

- **Purpose 1: Ensure that transactions that execute in parallel don't *interfere* with each other.**
- Purpose 2: Talks to “Log Manager”, ensures that steps inside a transaction are being “logged”.
- Purpose 3: Performs recovery after crashes, using logs.

**We will not cover
recovery in this class.**

ACID Properties

Atomicity

- either all of the actions of a transaction are executed, or none are.

Consistency

- each transaction executed in isolation keeps the database in a consistent state

Isolation

- Transactions are isolated from the effects of other, concurrently executing, transactions.

Durability

- updates stay in the DBMS!!!

Outline

- ✓ Transactions and ACID properties
 - Transactions in SQL
 - Isolation levels
 - Isolation Levels and Locking

Transactions in SQL

- A transaction begins when any SQL statement that queries the db begins.
- To end a transaction, the user issues a **COMMIT** or **ROLLBACK** statement.

May be omitted if
autocommit is off

START TRANSACTION

[SQL statements]

COMMIT or ROLLBACK;

Read-Only Transactions

- When a transaction only reads information, we have more freedom to let the transaction execute concurrently with other transactions.
- We signal this to the system by stating:

```
SET TRANSACTION READ ONLY;  
SELECT * FROM Accounts  
WHERE account#= '1234' ;  
...
```


Read-Write Transactions

- If we state “read-only”, then the transaction cannot perform any updates.

ILLEGAL!

```
SET TRANSACTION READ ONLY;  
UPDATE Accounts  
SET balance = balance - $100  
WHERE account#= '1234' ;...
```

- Instead, we must specify that the transaction may update (the default):

```
SET TRANSACTION READ WRITE;  
update Accounts  
set balance = balance - $100  
where account#= '1234' ;...
```

Concurrent Execution Problems

- **Write-Read conflict (WR): dirty read, inconsistent read**

A transaction reads a value written by another transaction that has not yet committed

- **Read-Write conflict (RW): unrepeatable read**

A transaction reads the value of the same object twice.

Another transaction modifies that value in between the two reads

- **Write-Write conflict (WW): lost update**

Two transactions update the value of the same object. The second one to write the value overwrites the first change

Example of Dirty Reads

sid	name	zip
123	Qin	14001
321	Kristin	14104

Transaction 1

```
START TRANSACTION;
```

```
SELECT zip FROM student WHERE sid = 123;  
/* will read 14111*/
```

Transaction 2

```
START TRANSACTION;
```

```
UPDATE student  
SET zip = 14111 WHERE sid = 123;
```

```
ROLLBACK  
/* zip code will revert to 14001*/
```

Example of Non-repeatable Reads

sid	name	zip
123	Qin	14001
321	Kristin	14104

Transaction 1

START TRANSACTION;

SELECT zip FROM student WHERE sid = 123;
/* will read 14001*/

SELECT zip FROM student WHERE sid = 123;
/* will read 14111*/

Transaction 2

START TRANSACTION;

UPDATE student
SET zip = 14111 WHERE sid = 123;
COMMIT;

Another concurrency problem: **Phantom Reads**

A **phantom read** occurs when, in the course of a transaction, new rows are added by another transaction to the records being read.*

* [https://en.wikipedia.org/wiki/Isolation_\(database_systems\)](https://en.wikipedia.org/wiki/Isolation_(database_systems))

Example of Phantom Reads

sid	name	zip
123	Qin	14001
321	Kristin	14104

Transaction 1

```
START TRANSACTION;
```

```
SELECT COUNT(*) FROM students WHERE sid  
BETWEEN 100 AND 400; /* will return 2 */
```

```
SELECT COUNT(*) FROM students WHERE sid  
BETWEEN 100 AND 400; /* will return 3 */
```

Transaction 2

```
START TRANSACTION;
```

```
INSERT INTO students(sid,name,zip)  
VALUES ( 100, 'Abdu', 14444 );
```

```
COMMIT;
```

Outline

- ✓ Transactions and ACID properties
- ✓ Transactions in SQL
- Isolation levels
- Isolation Levels and Locking

Isolation

- The problems we've seen are all related to *isolation*
- General rules of thumb w.r.t. isolation:
 - Fully serializable isolation is more expensive than “no isolation”
 - We can't do as many things concurrently (or we have to undo them frequently)
 - For performance, we generally want to specify the most relaxed *isolation level* that's acceptable
 - Note that we're “slightly” violating a correctness constraint to get performance!

Specifying Acceptable Isolation Levels

- To signal to the system that a **dirty read** is acceptable,

```
SET TRANSACTION  
ISOLATION LEVEL READ UNCOMMITTED;
```

- In addition, there are

```
SET TRANSACTION  
ISOLATION LEVEL READ COMMITTED;  
SET TRANSACTION  
ISOLATION LEVEL REPEATABLE READ;
```

Specifying Acceptable Isolation Levels

- **READ COMMITTED:**

- Forbids the reading of dirty (uncommitted) data, but allows a transaction T to issue the same query several times and get different answers
- No value written by T can be modified until T completes

- **REPEATABLE READ**

- If a tuple is retrieved once it will be retrieved again if the query is repeated
- But it is **NOT** a guarantee that the same query will get the same answer!

Summary of Isolation Levels

Level	Dirty Read	Unrepeatable Read	Phantoms
READ UN-COMMITTED	Maybe	Maybe	Maybe
READ COMMITTED	No	Maybe	Maybe
REPEATABLE READ	No	No	Maybe
SERIALIZABLE	No	No	No

Outline

- ✓ Transactions and ACID properties
- ✓ Transactions in SQL
- ✓ Isolation levels
- Isolation Levels and Locking

Implementing Isolation Levels

- One approach – use locking at some level (tuple, page, table, etc.):
 - each data item is either locked (in some mode, e.g. **shared** or **exclusive**) or is available (**no lock**)
 - an action on a data item can be executed if the transaction holds an appropriate lock
 - consider *granularity* of locks – how big of an item to lock
 - Larger granularity = fewer locking operations but more contention!
- Appropriate locks:
 - Before a read, a **shared lock** must be acquired
 - Before a write, an **exclusive lock** must be acquired

Lock Compatibility Matrix

Locks on a data item are granted based on a **lock compatibility matrix**:

		Mode of Data Item		
		None	Shared	Exclusive
Request mode {	Shared	Y	Y	N
	Exclusive	Y	N	N

When a transaction requests a lock, it must wait (**block**) until the lock is granted

Isolation Levels and Locking

READ UNCOMMITTED allows queries in the transaction to **read** data *without acquiring any lock*

- Access mode READ ONLY, no updates are allowed

READ COMMITTED requires a read-lock to be obtained for all tuples touched by queries, but it **releases the locks immediately after the read**

- Exclusive locks must be obtained for updates and held to end of transaction

Isolation levels and locking, cont.

REPEATABLE READ places shared locks on tuples retrieved by queries, holds them until the end of the transaction

- Exclusive locks must be obtained for updates and held to end of transaction

SERIALIZABLE places shared locks on tuples retrieved by queries *as well as the index*, holds them until the end of the transaction

- Exclusive locks must be obtained for updates and held to end of transaction

Holding locks to the end of a txn is called “**strict**” locking

A Notation for Transactions and Schedules

- We only care about reads and writes
 - Transaction is a sequence of $R(A)$'s and $W(A)$'s

$T1: r_1(A); w_1(A); r_1(B); w_1(B)$

$T2: r_2(A); w_2(A); r_2(B); w_2(B)$

- E.g. of a schedule:

$r_1(A); w_1(A); r_2(A); w_2(A); r_1(B); w_1(B); r_2(B); w_2(B)$

Concurrent execution: Example 1

- Suppose we have two transactions:

T1: SET TRANSACTION READ WRITE

ISOLATION LEVEL READ COMMITTED;

SQL code that translates to: R1(A), R1(B) W1(B) W1(C)

T2: SET TRANSACTION READ WRITE

ISOLATION LEVEL READ COMMITTED;

SQL code that translates to: R2(C), R2(A) W2(A)

One possible interleaved execution of the transactions above:

R1(A) R2(C) R2(A) W2(A) R1(B) W1(B) W1(C)

S₁(A) R₁(A) REL₁(A) S₂(C) R₂(C) REL₂(C) S₂(A) R₂(A) X₂(A) W₂(A)
REL₂(A) S₁(B) R₁(B) X₁(B) W₁(B) X₁(C) W₁(C) REL₁(B,C)

Concurrent execution: Example 2

- Now suppose that T1 is REPEATABLE READ:

T1: SET TRANSACTION READ WRITE

ISOLATION LEVEL REPEATABLE READ;

SQL code that translates to: R1(A), R1(B) W1(B) W1(C)

T2: SET TRANSACTION READ WRITE

ISOLATION LEVEL READ COMMITTED;

SQL code that translates to: R2(C), R2(A) W2(A)

One possible interleaved execution of the transactions above:

R1(A) R2(C) R2(A) R1(B) W1(B) W1(C) W2(A)

S1(A) R1(A) S2(C) R2(C) REL2(C) S2(A) R2(A) REL2(A) S1(B) R1(B) X1(B) W1(B) X1(C)
W1(C) REL1(A, B, C) X2(A) W2(A) REL2(A)

Outline

- ✓ Transactions and ACID properties
- ✓ Transactions in SQL
- ✓ Isolation levels
- ✓ Isolation Levels and Locking