Transactions

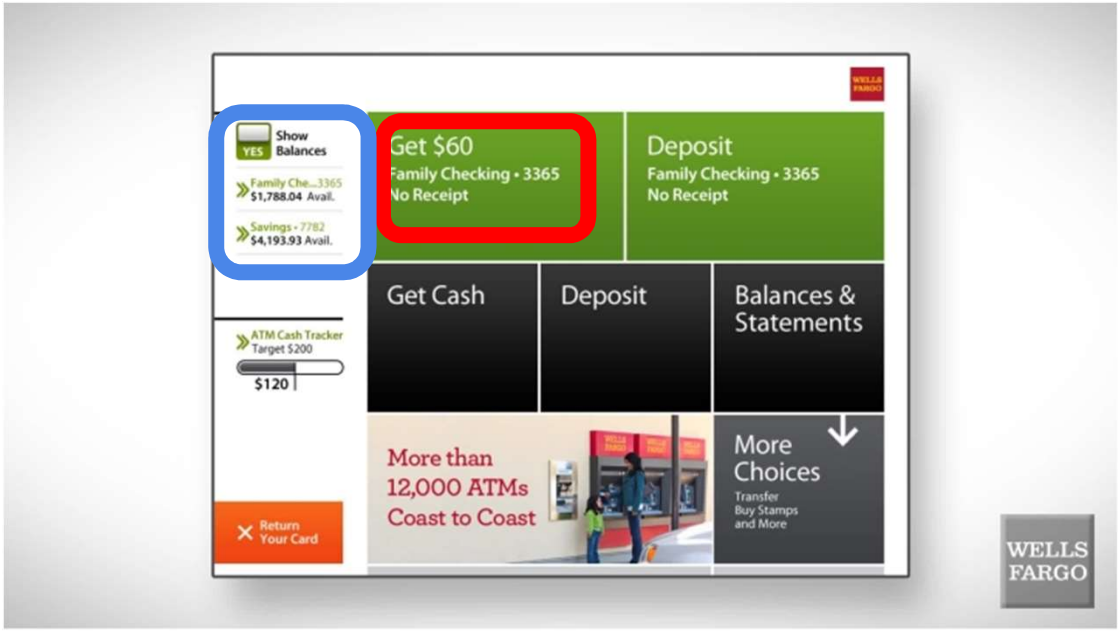
SQL Writes

```
UPDATE Product
SET Price = Price – 1.99
WHERE pname = ‘Gizmo’
```

```
INSERT INTO SmallProduct(name, price)
        SELECT pname, price
        FROM Product
        WHERE price <= 0.99
```

```
DELETE Product
        WHERE price <=0.99
```

Example: ATM Transaction



Read Balance
Give money
Update Balance

vs

Read Balance
Update Balance
Give money

Transactions: Basic Definition

A transaction ("TXN") is a sequence of one or more ***operations*** (reads or writes) which reflects ***a single real-world transition***.

In the real world, a TXN either happened completely or not at all

```
START TRANSACTION
                UPDATE Product
                SET Price = Price – 1.99
                WHERE pname = 'Gizmo'
COMMIT
```

Transactions

- Major component of database systems
- Critical for most applications; arguably more so than SQL

- Turing awards to database researchers:
  - Charles Bachman 1973
  - Edgar Codd 1981 for inventing relational dbs
  - Jim Gray 1998 for inventing transactions

Transactions in SQL

- In “ad-hoc” SQL, each statement = one transaction

- In a program, multiple statements can be grouped together as a transaction

```
START TRANSACTION
                UPDATE Bank SET amount = amount – 100
                WHERE name = ‘Bob’
                UPDATE Bank SET amount = amount + 100
                WHERE name = ‘Joe’
COMMIT
```

Motivation for Transactions

Grouping user actions (reads & writes) into ***transactions*** helps with two goals:

1. Recovery & Durability: Keeping the DBMS data consistent and durable in the face of crashes, aborts, system shutdowns, etc.

1. Concurrency: Achieving better performance by parallelizing TXNs ***without*** creating anomalies

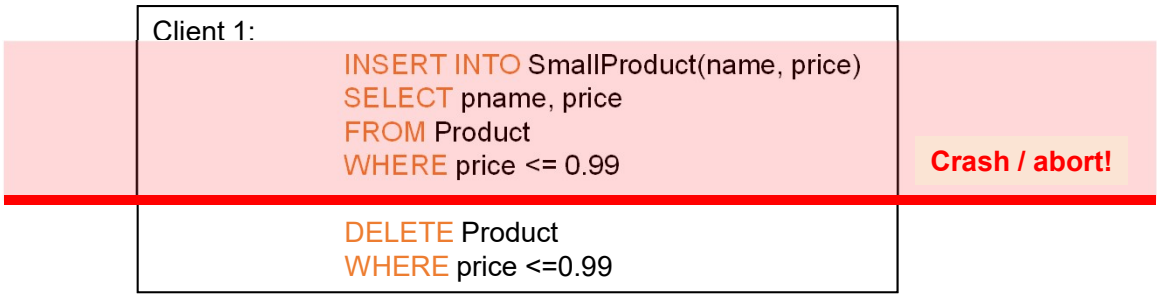
Motivation -- Recovery & Durability

1. Recovery & Durability of user data is essential for reliable DBMS usage

- The DBMS may experience crashes (e.g. power outages, etc.)
- Individual TXNs may be aborted (e.g. by the user)

Idea: Make sure that TXNs are either **durably stored in full**, **or not at all**; keep log to be able to “roll-back” TXNs

Protection against crashes / aborts



What goes wrong?

Protection against crashes / aborts

Client 1:

```
START TRANSACTION
INSERT INTO SmallProduct(name, price)
        SELECT pname, price
        FROM Product
        WHERE price <= 0.99

DELETE Product
        WHERE price <=0.99
COMMIT OR ROLLBACK
```

Now we'd be fine! We'll see how / why this lecture

Motivation -- Concurrent execution

2. Concurrent execution of user programs is essential for good DBMS performance.

- Disk accesses may be frequent and slow- optimize for throughput (# of TXNs), trade for latency (time for any one TXN)
- Users should still be able to execute TXNs as if in isolation and such that consistency is maintained

Idea: Have the DBMS handle running several user TXNs concurrently, in order to keep CPUs humming…

Multiple users: single statements

```
Client 1:   UPDATE Product
                    SET Price = Price – 1.99
                    WHERE pname = 'Gizmo'

Client 2:       UPDATE Product
                    SET Price = Price*0.5
                    WHERE pname='Gizmo'
```

Two managers attempt to discount products *concurrently*-
What could go wrong?

Multiple users: single statements

Client 1: START TRANSACTION

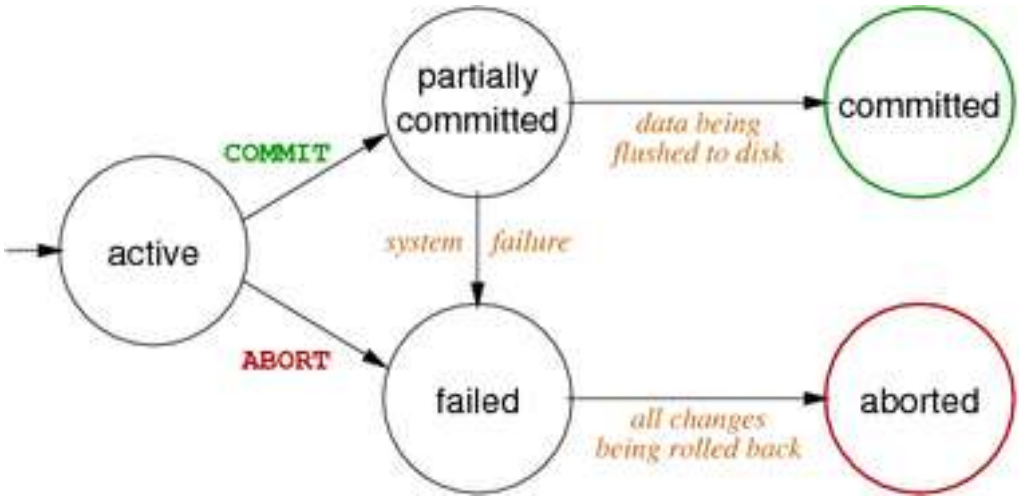
```
                                    UPDATE Product
                                    SET Price = Price – 1.99
                                    WHERE pname = ‘Gizmo’
                    COMMIT
```

Client 2: START TRANSACTION

```
                                    UPDATE Product
                                    SET Price = Price*0.5
                                    WHERE pname=‘Gizmo’
          COMMIT
```

Now works like a charm- we’ll see how / why next lecture…

Transaction States



Transaction Properties: ACID

In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called **ACID** properties.

- Atomic
  - State shows either all the effects of txn, or none of them
- Consistent
  - Txn moves from a state where integrity holds, to another where integrity holds
- Isolated
  - Effect of txns is the same as txns running one after another (ie looks like batch mode)
- Durable
  - Once a txn has committed, its effects remain in the database

ACID continues to be a source of great debate!

ACID: Atomicity

- TXN's activities are atomic: all or nothing
  - Intuitively: in the real world, a transaction is something that would either occur ***completely*** or ***not at all***
- Two possible outcomes for a TXN
  - It ***commits***: all the changes are made
  - It ***aborts***: no changes are made

Atomicity

- Either the **entire** transaction takes place at once or doesn't happen at all. There is no midway i.e. transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves the following two operations:

  Commit: If a transaction commits, changes made are visible.

  Abort: If a transaction aborts, changes made to database are not visible.

- Atomicity is also known as the 'All or nothing rule'.

Consider the following transaction **T** consisting of **T1** and **T2**:
Transfer of 100 from account **X** to account **Y**.

Before: X : 500	Y: 200
Transaction T	
T1	**T2**
Read (X) X: = X – 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

If the transaction fails after completion of **T1** but before completion of **T2**.(say, after **write(X)** but before **write(Y)**), then amount has been deducted from **X** but not added to **Y**.
This results in an inconsistent database state. Therefore, the transaction must be executed in entirety in order to ensure correctness of database state.

ACID: Consistency

- The tables must always satisfy user-specified ***integrity constraints***
  - *Examples:*
    - Account number is unique
    - Stock amount can't be negative
    - Sum of ***debits*** and of ***credits*** is 0

- How consistency is achieved:
  - Programmer writes a TXN to go from one consistent state to a consistent state
  - ***System*** makes sure that the TXN is atomic

Consistency

Integrity constraints must be maintained so that the database is consistent **before** and **after** the transaction. It refers to the correctness of a database.

Before: X : 500	Y: 200
Transaction T	
T1	**T2**
Read (X) X: = X − 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

The total amount before and after the transaction must be maintained.

Total **before T** occurs = **500 + 200 = 700**.

Total **after T occurs** = **400 + 300 = 700**.

Therefore, database is **consistent**. Inconsistency occurs in case **T1** completes but **T2** fails. As a result T is incomplete.

ACID: Isolation

- A transaction executes concurrently with other transactions
- **Isolation**: the effect is as if each transaction executes in *isolation* of the others.
  - E.g. Should not be able to observe changes from other transactions during the run

ACID: Durability

- The effect of a TXN must continue to exist (***"persist"***) after the TXN
  - And after the whole program has terminated
  - And even if there are power failures, crashes, etc.
  - And etc…
- Means: Write data to disk

What Could Go Wrong?

Why is it hard to provide ACID properties?

- Concurrent operations
  - Isolation problems
  - We saw one example earlier
- Failures can occur at any time
  - Atomicity and durability problems
- Transaction may need to abort

Challenges for ACID properties

- In spite of Power failures (not media failures)
- Users may abort the program: need to "**rollback** changes"
  - Need to ***log*** what happened (support **Atomicity** and **Durability**)
- Many users executing concurrently
  - Can be solved via **locking** (support **Cuncurrency**)

A Note: ACID is contentious!

- Many debates over ACID, both **historically** and **currently**
- Some “NoSQL” DBMSs relax ACID
- In turn, now “NewSQL” reintroduces ACID compliance to NoSQL-style DBMSs…

ACID is an extremely important & successful paradigm, but still debated!

Acknowledgement

Some of these slides are taken from cs145 course offered by Stanford University.