Transactions
Why Have Concurrent Processing (from multiple users) against a Database?

• Higher throughput, lower response times

• Better utilization of resources: While one process is doing a disk read, another can be using the CPU or reading another disk.

• But… concurrent processing can lead to incorrect data in the database or incorrect data returned to users!
Transaction

• Both concurrency control and crash recovery are based on the concept of a transaction.

• A transaction is a set of SQL statements, chosen by the developer, that accomplish some meaningful unit of work.

• You don’t need concurrency control and you don’t need (sophisticated) recovery if your database is read-only. Concurrency control and recovery are intended for databases that are updated.
Transaction

A transaction is:

– one “complete” set of actions
– defined by the developer (meaningful to the application)
– establishes where certain integrity constraints are enforced.

For concurrency control purposes (inside DBMS):

– a transaction is one atomic unit of work.
  • Thus we must be able to undo it
– DBMS cares only about the reads/writes to the DB
– DBMS views a transaction as (only) a sequence of reads, writes plus commit & abort
  (ignoring the rest of the program)
Example Transaction

Transfer $100 from one account to another:

- **BEGIN** transaction
- **read** balance from first account
- **add** $100 to first account balance
- **write** balance to first account
- **read** balance in second account
- **verify** balance to see if it contains at least $100
  - if not, **ABORT** transaction
- **subtract** $100 from second account
- **write** balance to second account
- **COMMIT** transaction
Example Transaction

Transfer $100 from one account to another:

- BEGIN transaction
- read balance from first account
- add $100 to first account balance
- write balance to first account
- read balance in second account
- verify balance to see if it contains at least $100
  - if not, ABORT transaction
- subtract $100 from second account
- write balance to second account
- COMMIT transaction
User (application developer) must indicate:

- Begin transaction
- read/write/modify statements intermixed with other programming language statements

plus either

- commit - indicates successful completion or
- abort - indicates program wants to roll back (undo the transaction)

- In order to ensure the correctness of the database, the DBMS and the programmer must guarantee four properties of transactions, called the ACID properties
Transactions

• You can defer constraint checking until a transaction commits.
• If you need to insert several rows in different tables that are related by foreign keys, you can defer the enforcement of the foreign keys until the end of the transaction – when all of the rows have been inserted into the database.
• Constraints can be defined as “deferrable” or not.
• A deferrable constraint can be defined initially as “deferrable” or “immediate”. You can toggle back and forth between deferrable and immediate.
The **ACID** Properties of Transactions

- **Atomicity:** A transaction happens in its entirety or not at all.

- **Consistency:** If the DB starts in a consistent state, (this notion is defined by the user; some of it may be enforced by integrity constraints) and a transaction executes with no other queries active, then the DB ends up in a consistent state.

- **Isolation:** Each transaction is isolated from other transactions. The effect on the DB is (as if) the transaction executed by itself. Other transactions don’t see partial partial updates.

- **Durability:** If a transaction commits, its changes to the database state persist (changes are permanent).
Concurrency control system in a DBMS

- Provides isolation of transactions

- Isolation is a problem only when multiple users are accessing the same data, and their actions interleave.
Declaring Read-Only Transactions

- Recall – read-only DB’s don’t have to worry about concurrency control

BEGIN TRANSACTION
SET TRANSACTION TRANSACTION READ ONLY
...
COMMIT

- Must be set before any query or data modification statement
Isolation levels (DBMS’s may differ...)

Developers can choose how much isolation (protection) they want … There are four isolation levels defined in SQL:

- “READ UNCOMMITTED” allows dirty reads, unrepeatable reads, and “phantoms”
- “READ COMMITTED” allows unrepeatable reads and phantoms
- “REPEATABLE READ” allows phantoms
- “SERIALIZABLE” full isolation
We would like criteria that are less strict than serializability.

Consider the three problems with concurrent transactions

- **Write-Write conflicts**
  - any write over uncommitted data.

- **Read-Write conflicts (unrepeatable reads)**
  - You read a value
  - Then some other transaction modifies it
  - Then you read it again (IT’S DIFFERENT!)

- **Write-Read conflicts (dirty reads)**
  - Some transaction modifies data
  - Then you read it
  - Then the other transaction aborts – and their modifications to the database are UNDONE
  - You have read a value that never “officially” existed.
Unrepeatable Read

A starts with $150
T1: check if A has > $100 then give B $100
T2: A makes a $100 withdrawal

Possible Schedule:

\[
\begin{align*}
T1: & \quad A > 100, & A = A - 100, \quad B = B + 100 \\
T2: & \quad A = A - 100
\end{align*}
\]
Dirty Read

T1: moves $100 from bank account B to A
T2: add 6% interest two accounts A and B

Possible Schedule:

<table>
<thead>
<tr>
<th>T1:</th>
<th>A=A+100,</th>
<th>B=B-100</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2:</td>
<td>B=1.06*B</td>
<td>A=1.06*A</td>
</tr>
</tbody>
</table>
Phantoms

• Occurs when different reads of a collection see two different sets of objects
  – New rows show up, despite the current transaction not making any changes

• Can happen when one transactions reads all rows satisfying a predicate, and another transaction inserts a row that matches it.

```
SELECT *
FROM People
WHERE score > 3;
```