Transactions

Commit and Rollback
Transfer $100 from account 1001 to 1007.

update account
set balance = balance + 100
where acc_id = 1001;
update account
set balance = balance - 100
where acc_id = 1007;

What if 1007 has less than $100?

• We can undo uncommitted work: ROLLBACK
• What does this mean for multiple users?
• We can commit work: COMMIT

Atomicity

update account
set balance = balance + 100
where acc_id = 1001;
update account
set balance = balance - 100
where acc_id = 1007;

should be a single unit: either both or neither succeeds

SQL uses Transactions to guarantee Atomicity
**Transaction (using PL/SQL)**

**Not transactional**

update account  
set balance = balance + 100  
where acc_id = 1001;
update account  
set balance = balance - 100  
where acc_id = 1007;

**Transactional**

begin  
update account  
set balance = balance + 100  
where acc_id = 1001;  
update account  
set balance = balance - 100  
where acc_id = 1007;  
end;

- will fail if 1007 has less than $100
- what if there is no account 1001?

**Consistency**

Constraint enforcement can be deferred to end of transaction (if constraint is deferrable).

STUDENT(sid, lastname, mentorid)

insert into student values (1, 'Brennigan', 3);
insert into student values (3, 'Patel', null);

set constraint fk_super deferred;

begin  
insert into student values (1, 'Brennigan', 3);  
insert into student values (3, 'Patel', null);  
end;

Run as script

**ACID Properties**

- Atomicity: Transaction succeeds as a whole or fails as a whole  
  Example: Money Transfer
- Consistency: Database is in consistent state at end of transaction  
  Example: Adding employees with supervisors
- Isolation: Transactions appear to serialize  
  Example: airline seat booking
- Durability: Committed changes are permanent  
  Example: system failure
Concurrent Processing

Let’s try to withdraw money from 1003 at two different ATMs.
What happens?

\[
\begin{array}{ll}
T1: & \text{read(balance)} \\
 & \text{balance := balance} - 100 \\
 & \text{if balance >= 0} \\
 & \text{write(balance)} \\
 & \text{commit}
\end{array}
\quad
\begin{array}{ll}
T2: & \text{read(balance)} \\
 & \text{balance := balance} - 50 \\
 & \text{if balance >= 0} \\
 & \text{write(balance)} \\
 & \text{commit}
\end{array}
\]

Potential problems

- P0 (Dirty Writes): T2 overwrites a T1 write before T1 commits
- P1 (Dirty Read): T2 reads T1 written cell before T1 commits
- P2 (Nonrepeatable Read): T2 modifies data that T1 has read.
- P3 (Phantom): T2 adds records that belong to a T1 query
- P4 (Lost Update): T2 writes over an item T1 has read, T1 then writes and commits.

Isolation Levels (SQL 92)

<table>
<thead>
<tr>
<th>Isolation Level</th>
<th>P1 Dirty Read</th>
<th>P2 Nonrepeatable Read</th>
<th>P3 Phantom</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Uncommitted</td>
<td>Allowed</td>
<td>Allowed</td>
<td>Allowed</td>
</tr>
<tr>
<td>Read Committed</td>
<td>x</td>
<td>Allowed</td>
<td>Allowed</td>
</tr>
<tr>
<td>Repeatable Read</td>
<td>x</td>
<td>x</td>
<td>Allowed</td>
</tr>
<tr>
<td>Serializable</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
**Isolation Levels in Oracle**

- set transaction isolation level read committed; (default, minimum level)
- set transaction isolation level serializable;
- set transaction read only;

Read Committed: no P1, but P2, P3 is possible
Serializable: no P1, P2, P3 possible
Read Only: no P1, P2, P3 possible

P1 (Dirty Read): T2 reads T1 written cell before T1 commits
P2 (Nonrepeatable Read): T2 modifies data that T1 has read.
P3 (Phantom): T2 adds records that belong to a T1 query

more at [http://docs.oracle.com/cd/B12037_01/server.101/b10743/consist.htm](http://docs.oracle.com/cd/B12037_01/server.101/b10743/consist.htm)

**Implementing Transactions**

- pessimistic
  - Locking (cell, row, table)
- optimistic
  - MVCC (Multiversion concurrency control)