2. Transactions

3. Transaction Properties
2. Transactions

3. Transaction Properties

4. Problems in Multi-User Mode
2. Transactions

3. Transaction Properties

4. Problems in Multi-User Mode

5. Commands for Transaction Control
Transactions in Multi-User Mode

Run-time integrity

- Avoid errors resulting from simultaneous access of multiple users to the same data
- Multiple programs running simultaneously → concurrent and competing processes
- *Transaction* as a processing unit
Example scenarios

- Almost simultaneous seat reservations for flights from many travel agencies
  → Place could be sold several times when several travel agencies identify the space as available
- Overlapping operations of a bank account
- Statistical database operations
  → Results are falsified, if data is changed during the calculation
Transaction

A transaction is a sequence of operations (actions), which transfers a database from a consistent state into another eventually changed consistent state, applying the ACID properties.
ACID Properties

- **Atomicity:**
  Transactions are either completed, or not performed at all.

- **Consistency:**
  If the database is in a consistent state before a transaction starts, the database is also consistent after the transaction has ended.

- **Isolation:**
  A user who is working on the database should not notice any other user working on it.

- **Durability:**
  The result of a transaction must be permanently stored within the database, after the transaction is completed.
Commands of a transaction language

- **Beginning of a transaction:** Begin-of-Transaction-Command `BOT` (in SQL implicitly)
- **commit:** The transaction should be completed successfully
- **abort:** The transaction must be aborted
Problems in multi-user mode

- Non-repeatable read
- Read of inconsistent states
- Dirty read
- Phantom Problem
- Lost update
- Multi-user anomaly
- Problems with cursor references
Non-repeatable Read: Example I

- Constraint $X = A + B + C$ at the end of the transaction $T_1$
- $X$ and $Y$ are local variables
- $T_i$ is the transaction $i$
- Integrity constraint: $A + B + C = 0$
Non-repeatable Read: Example II

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X := A$;</td>
<td>$Y := A/2$; $A := Y$; $C := C + Y$; commit;</td>
</tr>
<tr>
<td>$X := X + B$; $X := X + C$; commit;</td>
<td></td>
</tr>
</tbody>
</table>
Read of Inconsistent States: Example

Integrity constraint $X + Y = 0$

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>read($X$);</td>
<td>read($X$);</td>
</tr>
<tr>
<td>read($Y$);</td>
<td>read($Y$);</td>
</tr>
<tr>
<td>$X := X - 1$;</td>
<td>$Y := Y + 1$;</td>
</tr>
<tr>
<td>write($X$);</td>
<td>write($Y$);</td>
</tr>
<tr>
<td>$Y := Y + 1$;</td>
<td></td>
</tr>
<tr>
<td>write($Y$);</td>
<td></td>
</tr>
<tr>
<td>commit;</td>
<td></td>
</tr>
</tbody>
</table>
Dirty Read: Example

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>read(X)</code>; $X := X + 100$; write(X); abort;</td>
<td><code>read(X)</code>; $Y := Y + X$; write(Y); commit;</td>
</tr>
</tbody>
</table>
# Lost Update: Example

<table>
<thead>
<tr>
<th></th>
<th>( T_1 )</th>
<th>( T_2 )</th>
<th>( X )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>read(( X ));</td>
<td>read(( X ));</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>( X := X + 1; )</td>
<td>( X := X + 1; )</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>write(( X ));</td>
<td>write(( X ));</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11</td>
</tr>
</tbody>
</table>
## The Phantom Problem: Example

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
</table>
| ```sql
select count(*)
into X
from employee
where dept='CS';
```
| ```sql
insert
into employee
values (6789, 'Lilo', 'Pause', 'CS');
```
| ```sql
update employee
set Bonus = Bonus+10000/X
where dept='CS';
```
| ```sql
commit;
```
| ```sql
commit;
```
Multi-user anomaly: Example I

- Integrity constraint $A = B$

  $T_1 : = \langle A := A + 10; B := B + 10 \rangle$
  $T_2 : = \langle A := A \cdot 1.1; B := B \cdot 1.1 \rangle$

- Each $T_1$ and $T_2$ keeps the IC isolated
## Multi-user anomaly: Example II

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$T_2$</th>
<th>$A$</th>
<th>$B$</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>read(A);</code></td>
<td><code>read(A);</code></td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td><code>$A := A + 10;$</code></td>
<td><code>$A := A \cdot 1.1;$</code></td>
<td>20</td>
<td></td>
</tr>
<tr>
<td><code>write(A);</code></td>
<td><code>write(A);</code></td>
<td></td>
<td>22</td>
</tr>
<tr>
<td><code>read(B);</code></td>
<td><code>read(B);</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>$B := B + 10;$</code></td>
<td><code>$B := B \cdot 1.1;$</code></td>
<td>11</td>
<td></td>
</tr>
<tr>
<td><code>write(B);</code></td>
<td><code>write(B);</code></td>
<td>22</td>
<td>21</td>
</tr>
</tbody>
</table>
### Problems with cursor references: Example

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position cursor $C_1$ on next tuple with property $P$ (Tuple $A$)</td>
<td>Change property $P \rightarrow P'$ of $A$</td>
</tr>
<tr>
<td>Read current tuple</td>
<td></td>
</tr>
</tbody>
</table>

**Examples:**
- Position cursor $C_1$ on next tuple with property $P$ (Tuple $A$)
- Change property $P \rightarrow P'$ of $A$
Problems with cursor: SQL-notation

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>declare C_1 cursor for select * from Product where Price &lt; 100;</code></td>
<td></td>
</tr>
<tr>
<td><code>...</code></td>
<td></td>
</tr>
<tr>
<td><code>fetch C_1 into ...</code></td>
<td></td>
</tr>
<tr>
<td><code>update Product</code></td>
<td></td>
</tr>
<tr>
<td><code>set Price = 100</code></td>
<td></td>
</tr>
<tr>
<td><code>where ProdNr = 42;</code></td>
<td></td>
</tr>
</tbody>
</table>
Operations: user commands

- User commands that may influence the aborting of a transaction:
  - **commit** attempts to perform a commit ↳ not always successful (integrity violation) ↳ commit is only guaranteed if DBMS confirms the commit
  - **abort** forces final abort of a transaction
  - Other database operations that may lead to an abort: division by zero, integrity violations, ...
Operations: Internal

- **commit** performs a final commit internally

- Two versions for **abort**:
  - **abort+restart** aborts the transaction but also attempts to successfully complete the transaction by performing a restart (by intervention of scheduler; if **abort+restart** fails repeatedly, final abort is possible as well)
  - **abort+stop** performs the final abort (due to explicit user request or due to irreparable integrity violations)
Operations: Relation

user level

commit

db operations

abort

internal

commit

abort + restart

abort + stop