The Transaction Concept

- Multiple online users:
  - Gives rise to the concurrency problem.
- Component unreliability:
  - Gives rise to the failure problem.
- Database designers confronted these problems in the context of managing persistent data:
  - Online transaction processing system
  - Design, implementation, and operation of large application system with hundreds of terminals, tens of computers, providing service with no downtime, guaranteeing application correctness and data consistency.

The Transaction Concept

- Transactions provide an integrative framework in the presence of many “moving parts”.
- Distributed transaction-oriented systems are the enabling technology:
  - Distributed and Networked applications
  - E-commerce and Workflow systems
  - Large-scale Information Infrastructures
- Without transactions, distributed systems/networked applications cannot be made to work.
The Transaction Concept

- Transactions were originally developed in the context of DBMS as a paradigm to deal with:
  - Concurrent access to shared data
  - Failures of different kinds/types.
- Typical and canonical application scenarios in the context of banking application: Debit/Credit operations, and fund Transfers.
- The key problem solved in an elegant manner:
  - Subtle and difficult issue of keeping data consistent in the presence of concurrency and failures while ensuring performance, reliability, and availability.

OLTP Example: Debit/Credit

```c
void main ( ) {
EXEC SQL BEGIN DECLARE SECTION
int BAL, AID, amount;
EXEC SQL END DECLARE SECTION;
scanf("%d %d", &AID, &amount); // USER INPUT
EXEC SQL Select Balance into :BAL From Account
Where Account_Id = :AID; // READ FROM DB
BAL = BAL + amount; // update BALANCE
EXEC SQL Update Account
Set Balance = :b
Where Account_Id = :AID; // WRITE TO DB
EXEC SQL Commit Work;
}
```

Concurrent Executions: Lost Update Anomaly

<table>
<thead>
<tr>
<th>DEBIT ($50)</th>
<th>Time</th>
<th>CREDIT ($100)</th>
</tr>
</thead>
</table>
| Select Balance into :b1 From Account
  Where Account_Id = :a |
| b1 = b1 - 50 |
| Update Account
  Set Balance = :b2
  Where Account_Id = :a |
| b2 = b2 + 100 |
| Select Balance into :b3 From Account |
| b3 = b3 + 100 |

Observation: concurrency or parallelism may cause inconsistencies, requires concurrency control for “isolation”
Funds Transfer: Inconsistent DATA

```c
void main () {
    /* read user input */
    scanf("%d %d %d", &srcid, &tgtid, &amount);
    /* subtract amount from source account */
    EXEC SQL Update Account
    Set Balance = Balance - :amount Where AccId = :srcid;
    /* add amount to target account */
    EXEC SQL Update Account
    Set Balance = Balance + :amount Where AccId = :tgtid;
    EXEC SQL Commit Work;
}
```

Observation: failures may cause inconsistencies, require recovery for "atomicity" and "durability"
Basic Ingredients
- Elementary Operations (read and write)
- Transactions (i.e., transaction program executions)
- Execution histories
- Characterization of correct executions
- Protocols (i.e., online algorithms to ensure correctness)

Transaction Page Model: Syntax

Page Model of Transaction:
A transaction $T$ is a partial order of steps (actions) of the form $r[x]$ or $w[x]$, where $x \in D$ and reads and writes as well as multiple writes applied to the same object are ordered.
We write $T = (op, <)$ for transaction $T$ with step set $op$ and partial order $<$.  

Example: $r[x] w[x] r[y] w[y]$

Transaction Page Model: Semantics

Interpretation of $j^{th}$ step, $p_j$, of $T$:
If $p_j = r[x]$, then interpretation is assignment $v_j := x$ to local variable $v_j$
If $p_j = w[x]$ then interpretation is assignment $x := f_j(v_{j_1}, ..., v_{j_k})$ with unknown function $f_j$ and $j_1, ..., j_k$ denoting $T$’s prior read steps.
Lost Update Problem

\[
\begin{array}{c|c|c}
\text{P1} & \text{Time} & \text{P2} \\
\hline
x & 100 \times & 1 \\
\hline
x := x+100 & 2 & x := x+200 \\
w(x) & 3 & w(x) \\
\hline
\hline
\text{update "lost"}
\end{array}
\]

Observation: problem is the interleaving \( r_1(x), r_2(x), w_1(x), w_2(x) \)

Dirty Read Problem

\[
\begin{array}{c|c|c}
\text{P1} & \text{Time} & \text{P2} \\
\hline
x & 100 \times & 1 \\
\hline
x := x+100 & 2 & x := x-100 \\
w(x) & 3 & w(x) \\
failure & rollback & 4 & 5
\end{array}
\]

Observation: transaction rollbacks could affect concurrent transactions

cannot rely on validity of previously read data

Correctness Requirements: ACID

- **Atomicity:**
  - All-or-none property of user programs

- **Consistency:**
  - User program is a consistent unit of execution

- **Isolation:**
  - User programs are isolated with the side-effects of other user programs

- **Durability:**
  - Effects of user programs are persistent forever
Transactions Executions: History

- History:
  - Contains all operations from all transactions
  - Distinct termination for every transaction
  - Preserves the order of operations of all transactions
  - Termination is the final step
  - Conflicting operations are ordered

Notion of Transaction Histories

- Goal:
  - A technique/algorithm/scheduler that prevents incorrect or bad execution.
  - Develop the notion of correctness – or characterize what does correct execution means.
  - This characterization will be based on the histories of transaction execution:

\[ \text{H} \quad \text{Good} \quad \text{Bad} \]

Transaction Executions: Histories

Let \( T=\{T_1, \ldots, T_n\} \) be a set of transactions, where each \( T_i \in T \) has the form \( T_i=(\text{op}_i, \prec_i) \).

A history for \( T \) is \( H=(\text{op}(H), \prec_H) \) such that:
1. \( \text{op}(s) \subseteq \bigcup_{i=1}^{n} \text{op}_i \cup \bigcup_{i=1}^{n} \{a, c\} \)
2. for all \( i \), if \( c \in \text{op}(s) \Rightarrow a \notin \text{op}(s) \)
3. \( \bigcup_{i=1}^{n} \prec_i \subseteq \prec_H \)
4. for all \( i \), is \( \text{isin}, \) and all \( p \in \text{op} : p \prec_H q \) or \( p \prec_H a_i \)
5. for all \( p, q \in \text{op}(s) \) s.t. at least one of them is a write and both access the same data item: \( p \prec_H q \) or \( q \prec_H p \)
History Example

Correctness

Serial History
**General Idea**

- Notion of equivalence of two histories $H_1$ and $H_2$.

- Use this notion of equivalence to accept all histories which are "equivalent" to some serial history as being correct.

- How to establish this equivalence notion?

**Semantics**

- Equivalence via a notion of semantics:
  - We do not know the semantics of transaction programs
  - We need a general notion that can capture all potential transaction semantics
  - Need a general enough and powerful notion that can capture all possible semantics of transactions.