Outline

- Introduction
- Background
- Distributed DBMS Architecture
- Distributed Database Design
- Distributed Query Processing
- Distributed Transaction Management
  - ACID, Transaction Models
- Building Distributed Database Systems (RAID)
- Mobile Database Systems
- Privacy, Trust, and Authentication
- Peer to Peer Systems
Useful References

- Principles of Distributed Database Systems, Chapter 10.2-10.5
Properties of Transactions

**ATOMICITY**
- all or nothing

**CONSISTENCY**
- no violation of integrity constraints

**ISOLATION**
- concurrent changes invisible to other transactions

**DURABILITY**
- committed updates persist
Atomicity

- Either all or none of the transaction's operations are performed.
- Atomicity requires that if a transaction is interrupted by a failure, its partial results must be undone.
- The activity of preserving the transaction's atomicity in presence of transaction aborts due to input errors, system overloads, or deadlocks is called transaction recovery.
- The activity of ensuring atomicity in the presence of system crashes is called crash recovery.
Consistency

- Internal consistency
  - A transaction which executes alone against a consistent database leaves it in a consistent state.
  - Transactions do not violate database integrity constraints.
- Transactions are correct programs
Consistency Degrees

- **Degree 0**
  - Transaction $T$ does not overwrite dirty data of other transactions
  - Dirty data refers to data values that have been updated by a transaction prior to its commitment

- **Degree 1**
  - $T$ does not overwrite dirty data of other transactions
  - $T$ does not commit any writes before EOT
Consistency Degrees (cont’d)

- **Degree 2**
  - $T$ does not overwrite dirty data of other transactions
  - $T$ does not commit any writes before EOT
  - $T$ does not read dirty data from other transactions

- **Degree 3**
  - $T$ does not overwrite dirty data of other transactions
  - $T$ does not commit any writes before EOT
  - $T$ does not read dirty data from other transactions
  - Other transactions do not dirty any data read by $T$ before $T$ completes.
Isolation

- **Serializability**
  - If several transactions are executed concurrently, the results must be the same as if they were executed serially in some order.

- **Incomplete results**
  - An incomplete transaction cannot reveal its results to other transactions before its commitment.
  - Necessary to avoid cascading aborts.
Isolation Example

- Consider the following two transactions:
  
  \[ T_1: \text{Read}(x) \]
  
  \[ x \leftarrow x + 1 \]
  
  \[ \text{Write}(x) \]
  
  \[ \text{Commit} \]
  
  \[ T_2: \text{Read}(x) \]
  
  \[ x \leftarrow x + 1 \]
  
  \[ \text{Write}(x) \]
  
  \[ \text{Commit} \]

- Possible execution sequences:

  \[ T_1: \text{Read}(x) \]
  
  \[ T_1: x \leftarrow x + 1 \]
  
  \[ T_1: \text{Write}(x) \]
  
  \[ T_1: \text{Commit} \]
  
  \[ T_2: \text{Read}(x) \]
  
  \[ T_2: x \leftarrow x + 1 \]
  
  \[ T_2: \text{Write}(x) \]
  
  \[ T_2: \text{Commit} \]

  \[ T_1: \text{Read}(x) \]
  
  \[ T_1: x \leftarrow x + 1 \]
  
  \[ T_2: \text{Read}(x) \]
  
  \[ T_2: x \leftarrow x + 1 \]
  
  \[ T_1: \text{Write}(x) \]
  
  \[ T_1: \text{Commit} \]
  
  \[ T_2: \text{Write}(x) \]
  
  \[ T_2: \text{Commit} \]
SQL-92 Isolation Levels

Phenomena:

- Dirty read
  - $T_1$ modifies $x$ which is then read by $T_2$ before $T_1$ terminates; $T_1$ aborts $\Rightarrow T_2$ has read value which never exists in the database.

- Non-repeatable (fuzzy) read
  - $T_1$ reads $x$; $T_2$ then modifies or deletes $x$ and commits. $T_1$ tries to read $x$ again but reads a different value or can’t find it.

- Phantom
  - $T_1$ searches the database according to a predicate while $T_2$ inserts new tuples that satisfy the predicate.
SQL-92 Isolation Levels (cont’d)

- Read Uncommitted
  - For transactions operating at this level, all three phenomena are possible.

- Read Committed
  - Fuzzy reads and phantoms are possible, but dirty reads are not.

- Repeatable Read
  - Only phantoms possible.

- Anomaly Serializable
  - None of the phenomena are possible.
Durability

- Once a transaction commits, the system must guarantee that the results of its operations will never be lost, in spite of subsequent failures.

- Database recovery
Characterization of Transactions

Based on

- Application areas
  - non-distributed vs. distributed
  - compensating transactions
  - heterogeneous transactions

- Timing
  - on-line (short-life) vs batch (long-life)

- Organization of read and write actions
  - two-step
  - restricted
  - action model

- Structure
  - flat (or simple) transactions
  - nested transactions
  - workflows
Transaction Structure

- **Flat transaction**
  - Consists of a sequence of *primitive* operations embraced between a `begin` and `end` markers.

```
Begin_transaction Reservation
  ...
end.
```

- **Nested transaction**
  - The operations of a transaction may themselves be transactions.

```
Begin_transaction Reservation
  ...
  Begin_transaction Airline
    ...
  end. {Airline}
  Begin_transaction Hotel
    ...
  end. {Hotel}
end. {Reservation}
```
Nested Transactions

- Have the same properties as their parents □ may themselves have other nested transactions.

- Introduces concurrency control and recovery concepts to within the transaction.

- Types
  - Closed nesting
    - Subtransactions begin after their parents and finish before them.
    - Commitment of a subtransaction is conditional upon the commitment of the parent (commitment through the root).
  - Open nesting
    - Subtransactions can execute and commit independently.
    - Compensation may be necessary.
Workflows

- “A collection of tasks organized to accomplish some business process.” [D. Georgakopoulos]

**Types**

- **Human-oriented workflows**
  - Involve humans in performing the tasks.
  - System support for collaboration and coordination; but no system-wide consistency definition

- **System-oriented workflows**
  - Computation-intensive & specialized tasks that can be executed by a computer
  - System support for concurrency control and recovery, automatic task execution, notification, etc.

- **Transactional workflows**
  - In between the previous two; may involve humans, require access to heterogeneous, autonomous and/or distributed systems, and support selective use of ACID properties
Workflow Example

\[ T_1: \text{Customer request obtained} \]
\[ T_2: \text{Airline reservation performed} \]
\[ T_3: \text{Hotel reservation performed} \]
\[ T_4: \text{Auto reservation performed} \]
\[ T_5: \text{Bill generated} \]
Transactions Provide...

- *Atomic* and *reliable* execution in the presence of failures
- *Correct* execution in the presence of multiple user accesses
- Correct management of *replicas* (if they support it)
Transaction Processing Issues

- Transaction structure (usually called transaction model)
  - Flat (simple), nested

- Internal database consistency
  - Semantic data control (integrity enforcement) algorithms

- Reliability protocols
  - Atomicity & Durability
  - Local recovery protocols
  - Global commit protocols
Transaction Processing Issues

- Concurrency control algorithms
  - How to synchronize concurrent transaction executions (correctness criterion)
  - Intra-transaction consistency, Isolation

- Replica control protocols
  - How to control the mutual consistency of replicated data
  - One copy equivalence and ROWA
Architecture Revisited

Begin_transaction, Read, Write, Commit, Abort

Scheduler (SC)

Scheduling/Descheduling Requests

Transaction Manager (TM)

Results

Distributed Execution Monitor

With other SCs

With other TMs

To data processor
Centralized Transaction Execution

- **User Application**
  - Begin_Transaction, Read, Write, Abort, EOT
- **Transaction Manager (TM)**
  - Read, Write, Abort, EOT
  - Results
- **Scheduler (SC)**
  - Scheduled Operations
  - Results
- **Recovery Manager (RM)**
  - Results & User Notifications
Distributed Transaction Execution

User application

Begin_transaction, Read, Write, EOT, Abort

Results & User notifications

TM

SC

RM

TM

SC

RM

Distributed Transaction Execution Model

Replica Control Protocol

Distributed Concurrency Control Protocol

Local Recovery Protocol