ADVANCED TRANSACTION PROCESSING IN DATABASES

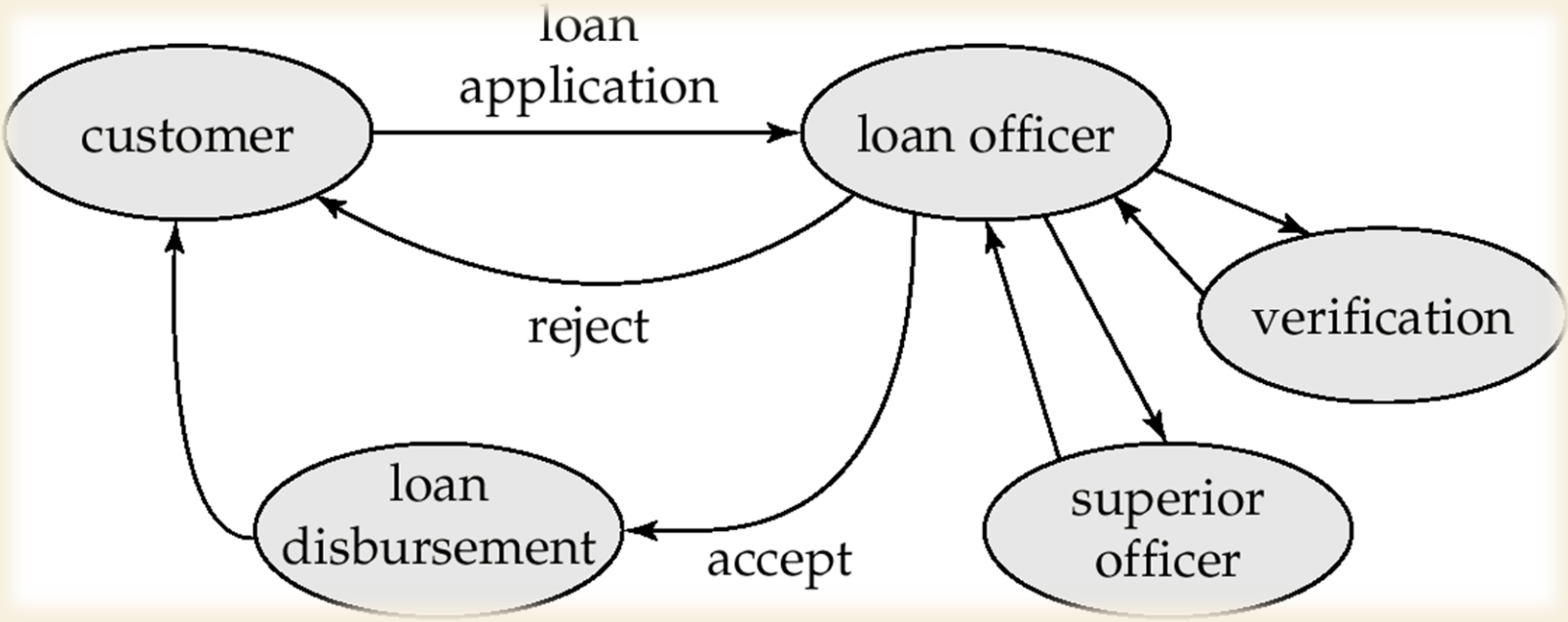
TRANSACTIONAL WORKFLOWS

COMPILED BY
M ABDUL JAWAD

TRANSACTIONAL WORKFLOWS

- **Workflows** are activities that involve the coordinated execution of multiple tasks performed by different processing entities.
- With the growth of networks, and the existence of multiple autonomous database systems, workflows provide a convenient way of carrying out tasks that involve multiple systems.
- Example of a workflow delivery of an email message, which goes through several mails systems to reach destination.
  - Each mailer performs a tasks: forwarding of the mail to the next mailer.
  - If a mailer cannot deliver mail, failure must be handled semantically (delivery failure message).
- Workflows usually involve humans: e.g. loan processing, or purchase order processing.

LOAN PROCESSING WORKFLOW



- In the past, workflows were handled by creating and forwarding paper forms
- Computerized workflows aim to automate many of the tasks. But the humans still play role e.g. in approving loans.

TRANSACTIONAL WORKFLOWS

- To automate the tasks involved in Loan Processing – We can store the Loan application and associated information in a database.
- Workflow itself then involves handling responsibility from one human to next.
- Workflows are very important in organizations and organizations today have multiple software systems that need to work together.
  - Employee joins an organization and information about the employee is provided to
    - Payroll System.
    - Library System.
    - Authentication Systems.

TRANSACTIONAL WORKFLOWS

- We need to address two activities in general to automate a workflow:
  - **Workflow Specification**.
  - **Workflow Execution.**

- Both activities are complicated by the fact that many organizations use several independently managed information processing systems, that in most cases were developed separately to automate different functions.
- Workflow activities may require interactions among several such systems each performing a task as well as interactions with humans.

WORKFLOW SPECIFICATION

- A **task** may use parameters stored in its variables, may retrieve and update data in the local system, may store its results in its output variables.
- At any time during the execution, the **workflow state** consists of the collection of states of the workflows constituent tasks and the states of all variables in the workflow specification.
- The coordination of tasks can be specified either statically or dynamically.
  - A static specification defines the tasks and dependencies among them, before the execution of workflow begins.
  - A dynamic specification defines the dependencies and execution of tasks on demand and along the route of execution itself.

STATIC SPECIFICATION

- In static specification, the dependencies among the tasks may be simple and everything is known well in advance before any execution of any task starts – Each task may be completed before the next begins.
- The transactions whose preconditions are satisfied are executed. The preconditions can be defined through dependencies like:
  - **Execution states of other tasks - "task t_i cannot start until task t_j has ended".**
  - **Output values of other tasks. "task t_i can start if task t_j returns a value greater than 25"**
  - **External variables, that are modified by external events. "task t_i must be started within 24 hours of the completion of task t_j"**

FAILURE-ATOMICITY REQUIREMENTS

- Traditional notion of ***'failure atomicity'*** requires that failure of any task result in failure of workflow.
- *Designer of workflow is allowed to define failure atomicity requirements of workflow. Even if we may have a sub-transaction failing at any point of time, a workflow designer may execute a functionally equivalent task so that final transaction can commit.*
- System must guarantee that every transaction that every execution of workflow must terminate in a state that satisfies the failure atomicity requirement defined by the designer.
- **Acceptable Termination State**
  - ***Committed Acceptable Termination State****: Objectives achieved successfully.*
  - ***Aborted Acceptable Termination State****:Workflow fails to achieve objectives.*
- **Unacceptable Termination State**
  - All other states fit in unacceptable termination states.

WORKFLOW MANAGEMENT SYSTEM ARCHITECTURES

- Execution of workflow tasks may be controlled by ***Human Coordinator*** or by a ***Software system known as Workflow Management System.***

- Workflow Management System consists of:
  - ***Scheduler***: is a program that processes workflows by submitting various tasks for execution, monitoring various events and evaluating conditions related to inter-task dependencies.
  - ***Task Agents***: controls the execution of a task by a processing entity.
  - ***Query Mechanism*:** A query mechanism to query the state of the workflow.

ARCHITECTURE FOR DEVELOPMENT OF WORKFLOW MANAGEMENT SYSTEM

- We have three architectural approaches for development of a workflow management system:
  - ***Centralized Architecture:*** Possesses a single scheduler that schedules the tasks for all concurrently executing workflows.
    - used in workflow systems where the data is stored in a central database.
    - easier to keep track of the state of a workflow.
  - ***Partially Distributed Architecture:*** has one scheduler initiated for each workflow.
  - ***Fully Distributed Architecture:*** has no scheduler, but the task agents coordinate their execution by communicating with each other to satisfy task dependencies and other workflow requirements.
    - used in simplest workflow execution systems
    - based on electronic mail

LONG DURATION TRANSACTIONS

- Depending on the lifetime or duration, transactions can be classified as:
  - **Short Duration Transaction**: *Short duration transaction is also known as Online transaction requiring very short execution/response time and access small portion of the database.*
  - **Long Duration Transaction:** *A long duration transaction also known as Batch transaction requires a longer execution/response time and generally accesses larger portion of the database.*
    - **Alternately we can define long duration in context of database systems as the on that involve human intervention, while short duration transactions are more or less Non-Interactive.**
    - **Current normal database applications including Railway Reservation, Banking System require the short-duration category transactions.**

LONG DURATION TRANSACTIONS

- The long-duration transactions exhibit following properties:
  - **Long Duration**: *When interacting with Humans, it is quite natural the response will be very slow relative to computer speed.*
    - **In some applications, the human activity may involve hours, days, even longer periods of time.**
    - **So overall the transactions will be of longer duration.**
  - **Exposure to Uncommitted Data:** *In many cases of long duration transactions, the other transactions may be forced to read uncommitted data.*
    - **If several users are cooperating on a project, user transactions may need to exchange data prior to transaction commit.**

LONG DURATION TRANSACTIONS

- **Subtasks:** *Interactive transaction will consist of set of subtasks initiated by the user.*
  - **At some point of time, user may wish to abort a subtask, without necessarily causing the entire transaction to Abort.**
- **Recoverability:** *It is un-acceptable to abort a Long-duration interactive transaction given a system crash.*
  - **Even if we have a system crash, the active transaction must be recovered to a state that existed shortly before the crash, so that relatively human work is lost.**
- **Performance:**
  - **Good performance in Long Duration Transaction is defined as how fast the response has been generated for a particular transaction.**
  - **However, in case of non-interactive system, performance is measured as** *HIGH THROUGHPUT.*

IMPACT OF CONCURRENCY PROTOCOLS

- Given the properties of Long Duration Transactions, various Concurrency protocols are very difficult to implement:
  - **2 Phase Locks:** If a lock *isn't granted, the transaction requesting the lock is forced to wait for the data item to be unlocked.*
    - *If the transaction holding the lock on data item happens to be a Long Duration Transaction, Response Time increases leading to increased chances of Deadlock.*
  - **Time-Stamp-Based Protocols:** *Although timestamp based protocols never a transaction to wait; however they require transaction to abort under certain circumstances.*
    - *If a long duration transaction is aborted a substantial amount of work is lost.*

IMPACT OF CONCURRENCY PROTOCOLS

- **Validation Protocols**:
  - With the enforcement of Serializibility, the result is a **Long Wait** or **Abortion of Long Duration Transactions** or **Both.**
  - Further difficulties arise, with the enforcement of serializibility, when considering Recovery Issues.
  - A case of **CASCADING ROLLBACK** finally has undesirable effects on Long Duration Transaction.
- **Snapshot Isolation** is a solution for achieving Concurrency when dealing with Long Duration Transactions.

NESTED AND MULTILEVEL TRANSACTIONS

- A **long duration transaction**, can be viewed as a collection of related Subtasks or Sub-transactions.
- By structuring the Long Duration transaction as a set of Sub-transactions, we are able to run several sub-transactions in parallel, provided parallelism doesn't lead to conflicts.
- When Long duration transactions are modelled as a set of sub-transactions, a system crash or a failure doesn't necessarily mean roll back of entire Long Duration Transaction.
- Suppose some transaction ti in T may abort; it doesn't mean we will abort T, instead T will simply restart ti.
- Suppose transaction ti commits, this commit isn't permanent, ti commits to T. If T aborts ti will need to abort.

NESTED AND MULTILEVEL TRANSACTIONS

- A nested or multilevel Transaction **T** = $\{t_1, t_2, ..., t_n\}$ of sub-transactions and a partial order P on T.
- Execution of T mustn't violate the partial order.
- Multi-level Transaction is a **T** in which a sub-transaction is allowed to release locks on completion.
- **Nested Transaction is a T, in which a sub-transaction** t_i holding the locks, upon completion of t_i, Locks are automatically assigned to T.

COMPENSATING TRANSACTIONS

- Long Duration Transactions generally suffer from *long waiting time*. To reduce the waiting times, we expose the uncommitted updates to other concurrently executing transactions.
- This exposure of uncommitted updates to transactions have serious issues – Cascading Rollback.
- To overcome the issues, concept of Compensating Transactions were implemented.
  - We have a Long Duration Transaction 'T', divided into several Sub-Transactions t_1, t_2, ..., t_n.
  - If outer level sub-transaction of T commits, it releases its locks.
  - If the outer level sub-transaction ti of transaction T has aborted effect of its sub-transactions must be undone.

COMPENSATING TRANSACTIONS

- Suppose that sub-transactions t_1, t_2, ..., t_k *have committed and* t_{k+1} *was executing when decision to abort was made.*
- *Since* t_{k+1} *has aborted we simply undo the effects of that sub transaction.*
- Instead of undoing all changes made by the failed transaction, action is taken to "compensate" for the failure.
- Consider a long-duration transaction T_i representing a travel reservation, with sub-transactions $T_{i,1}$, which makes airline reservations, $T_{i,2}$ which reserves rental cars, and $T_{i,3}$ which reserves a hotel room.
  - Hotel cancels the reservation.
  - Instead of undoing all of T_i, the failure of T_i,3 is compensated for by deleting the old hotel reservation and making a new one.

REAL-TIME TRANSACTION SYSTEMS

- In systems with real-time constraints, correctness of execution involves both database consistency and the satisfaction of deadlines.
  - **Hard deadline** – Serious problems may occur if task is not completed within deadline
  - **Firm deadline** - The task has zero value if it completed after the deadline.
  - **Soft deadline** - The task has diminishing value if it is completed after the deadline.

REAL-TIME TRANSACTION SYSTEMS

- The wide variance of execution times for read and write operations on disks complicates the transaction management problem for time-constrained systems
  - main-memory databases are thus often used
  - Waits for locks, transaction aborts, contention for resources remain as problems even if data is in main memory
- Design of a real-time system involves ensuring that enough processing power exists to meet deadline without requiring excessive hardware resources