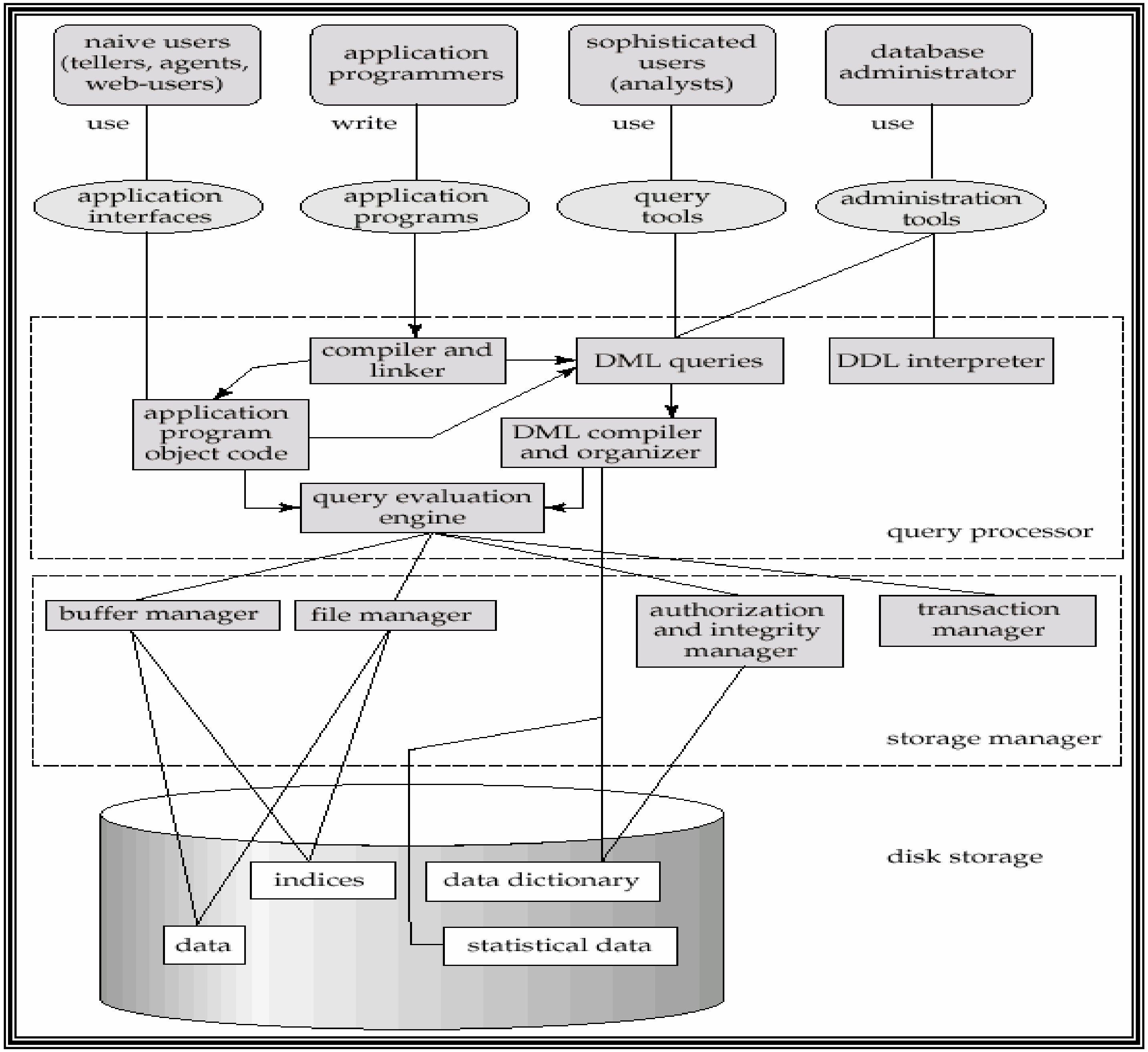
Part Ⅳ Transaction Management


naive users (tellers, agents, web-users)
application programmers
sophisticated users (analysts)
database administrator
use
write
use
use
application interfaces
application programs
query tools
administration tools
compiler and linker
DML queries
DDL interpreter
application program object code
DML compiler and organizer
query evaluation engine
query processor
buffer manager
file manager
authorization and integrity manager
transaction manager
storage manager
disk storage
indices
data dictionary
data
statistical data

content

- Transaction
- Recovery
- concurrent

Chapter 12 Transaction

Related to some contents in

text book chapter 14 (version 7)

text book chapter 15 (version 8)

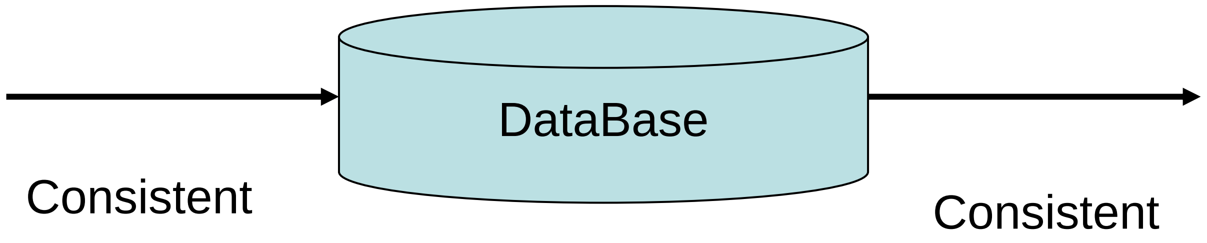
Contents

Transaction Concept

- A transaction is a unit of program execution that accesses and possibly updates various data items.
- A transaction must see a consistent database.
- During transaction execution the database may be inconsistent.

Transaction Concept–cont.

- When the transaction is committed, the database must be consistent.



Transaction Concept–cont.

- Two main issues to deal with:
  - – Failures of various kinds, such as hardware failures and system crashes
  - – Concurrent execution of multiple transactions

Transaction Property

- To preserve integrity of data, the database system must ensure ACID properties for transaction
  - **Atomicity.** Either all operations of the transaction are properly reflected in the database or none are.
  - **Consistency.** Execution of a transaction in isolation preserves the consistency of the database.
  - **Durability.** After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

Transaction Property–cont.

- **Isolation.** Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions.
  - That is, for every pair of transactions T_i and T_j, it appears to T_i that either T_j, finished execution before T_i started, or T_j started execution after T_i finished.

Example

- Transaction to transfer \$50 from account *A* to account *B*:
  1. **read**(*A*)
  2. *A* := *A* – 50
  3. **write**(*A*)
  4. **read**(*B*)
  5. *B* := *B* + 50
  6. **write**(*B*)

Example – cont.

- Consistency requirement – the sum of *A* and *B* is unchanged by the execution of the transaction.
- Atomicity requirement — if the transaction fails after step 3 and before step 6, the system should ensure that its updates are not reflected in the database, else an inconsistency will result.

Example – cont.

- Durability requirement — once the user has been notified that the transaction has completed (i.e., the transfer of the $50 has taken place), the updates to the database by the transaction must persist despite failures.

Example – cont.

- Isolation requirement — if between steps 3 and 6, another transaction is allowed to access the partially updated database, it will see an inconsistent database (the sum $A + B$ will be less than it should be).

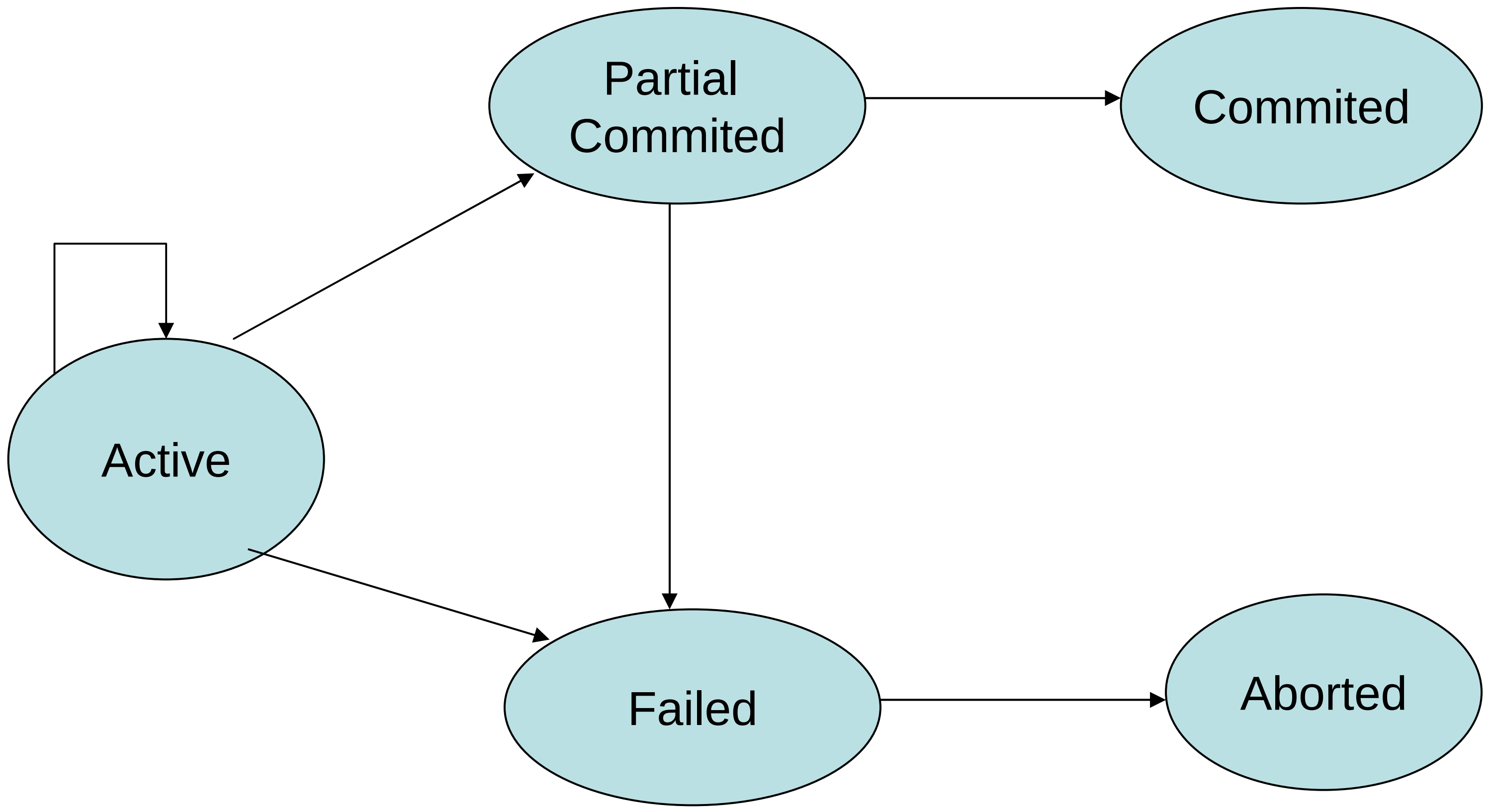
Transaction State

- Active, the initial state; the transaction stays in this state while it is executing
- Partially committed, after the final statement has been executed.
- Failed, after the discovery that normal execution can no longer proceed.

Transaction State - cont.

- Aborted after the transaction has been rolled back and the database restored to its state prior to the start of the transaction. Two options after it has been aborted:
  - restart the transaction – only if no internal logical error
  - kill the transaction
- Committed after *successful completion*.

Transaction State - Cont.



Transaction Manager

- A software product to create, execute and manage DB transactions
  - To make transaction execution efficient、 concurrent and reliable
- Target
  - Preserve transaction ACID properties
  - Minimal memory and CPU cost
- Primitive statement
  - Begin trans
  - Commit
  - Abort

Implementation of Atomicity and Durability

- The recovery-management component of a database system implements the support for atomicity and durability.

Concurrent Executions

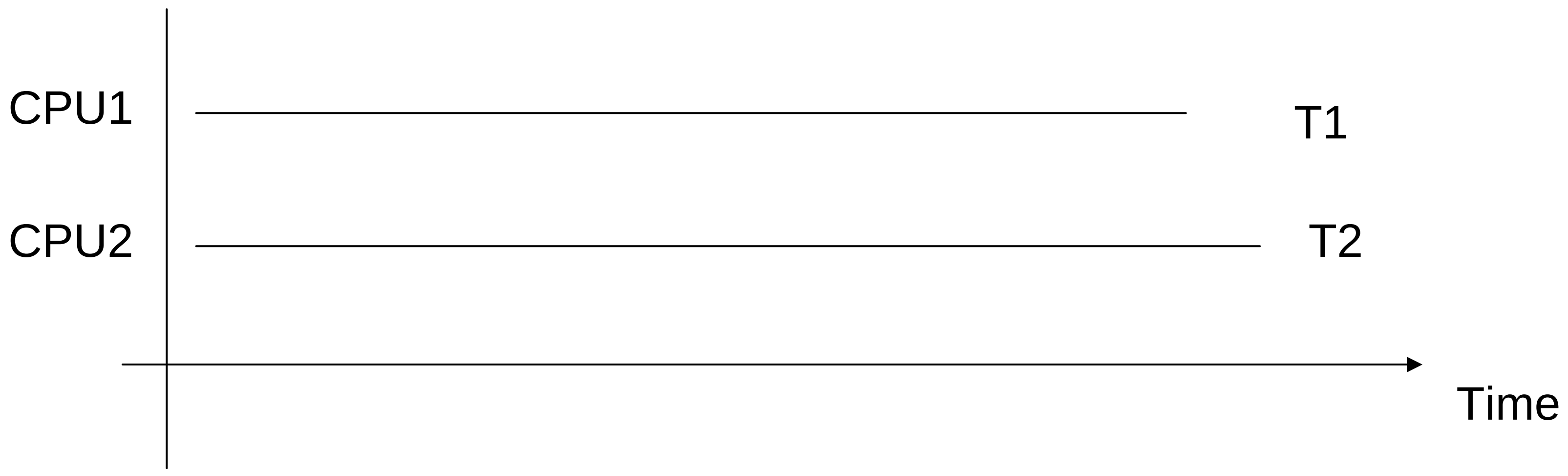
- Multiple transactions are allowed to run concurrently in the system. Advantages are:
  - increased processor and disk utilization, leading to better transaction *throughput:* one transaction can be using the CPU while another is reading from or writing to the disk
  - reduced average response time for transactions: short transactions need not wait behind long ones.

Concurrent Executions-cont.

- *Concurrency control* mechanisms to achieve isolation, i.e., to control the interaction among the concurrent transactions in order to prevent them from destroying the consistency of the database

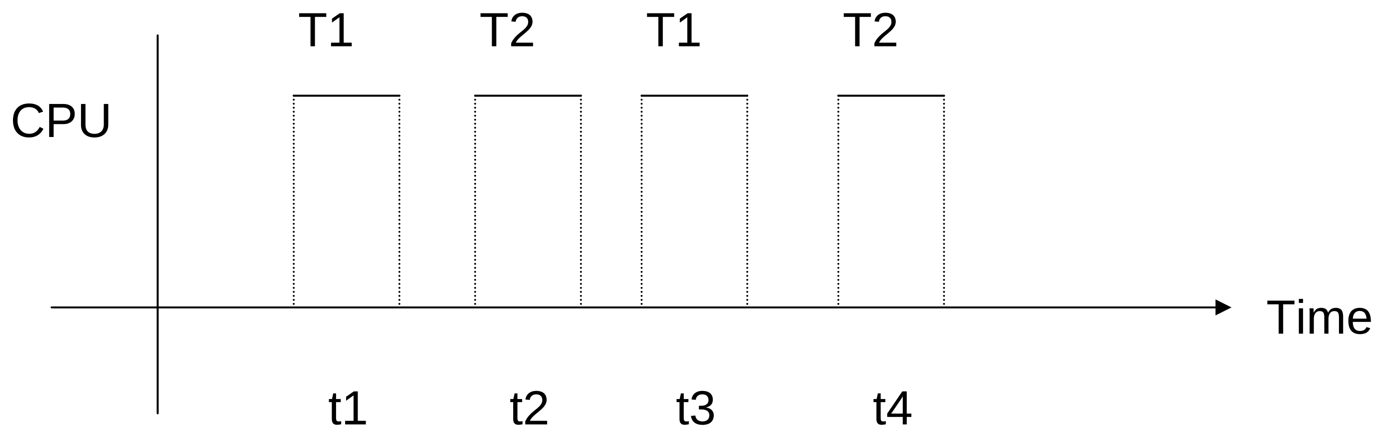
Concurrent Executions-cont.

- Multi-processor



Concurrent Executions-cont.

- Single- processor



Transaction Definition in SQL

- Data manipulation language must include a construct for specifying the set of actions that comprise a transaction.
- In SQL, a transaction begins implicitly.
- A transaction in SQL ends by:
  - **Commit work** commits current transaction and begins a new one.
  - **Rollback work** causes current transaction to abort.

Transaction Definition in SQL - cont.

- Levels of consistency specified by SQL-92:
  - **Serializable** — default
  - **Repeatable read**
  - **Read committed**
  - **Read uncommitted**

Levels of Consistency in SQL-92 - cont.

- **Serializable** — default
- **Repeatable read** — only committed records to be read, repeated reads of same record must return same value. However, a transaction may not be serializable – it may find some records inserted by a transaction but not find others.

Lower degrees of consistency useful for gathering approximate information about the database, e.g., statistics for query optimizer.

Levels of Consistency in SQL-92 - cont.

- **Read committed** — only committed records can be read, but successive reads of record may return different (but committed) values.
- **Read uncommitted** — even uncommitted records may be read.

Exercises

1. ACID property.
2. Transaction state transfer condition.
3. Give two transaction examples.

下节课

恢复