Overview of Transaction Processing

Outline of Lecture

- Introduction to Transaction Processing
  - Review
  - Problems and Solution
- Concepts
- Properties
- Example – Oracle DB

Databases - multi user applications

- Databases - multi user applications
- Concurrent access to the same data may induce different types of problems
- Errors may occur in the DBMS or its environment

Transactions, Recovery and Concurrency Control

- Examples of problems: hard disk failure, application/DBMS crash, division by 0, …
- **Recovery**: activity of ensuring that, whichever of the problems occurred, the database is returned to a consistent state without any data loss afterwards
- **Concurrency control:** coordination of transactions that execute simultaneously on the same data so that they do not cause inconsistencies in the data because of mutual

Types of Failures

- **Transaction failure** results from an error in the logic that drives the transaction’s operations and/or in the application logic
- **System failure** occurs if the operating system or the database system crashes
- **Media failure** occurs if the secondary storage is damaged or inaccessible

Transactions, Recovery and Concurrency Control

- Transaction: set of DML operations induced by a single user or application, that should be considered as one undividable unit of work
  - Transfer between two bank accounts
- Transaction - ‘succeeds’ or ‘fails’ in its entirety
- Transaction - database from one consistent state into another one

DML

- Main operations
  - INSERT
  - UPDATE
  - DELETE

SQL - DML

- **INSERT**

INSERT INTO R(A_1,A_2, …, A_n)

VALUES (v_1,v_2, …, v_n)

- **UPDATE**

UPDATE R SET <new value assignment>

WHERE <condition>

- **DELETE**

DELETE FROM *R* WHERE <condition>

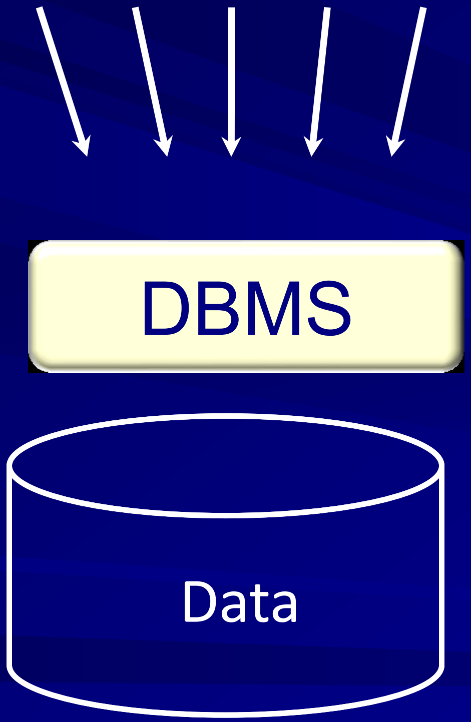
Real work

- It’s all about fast query response time and correctness
- DBMS is a multi-user systems
  - Many different requests
  - Some against same data items

Problems

- Failure occurs
- Concurrent database access
- Resilience to system failures

Concurrent DB access



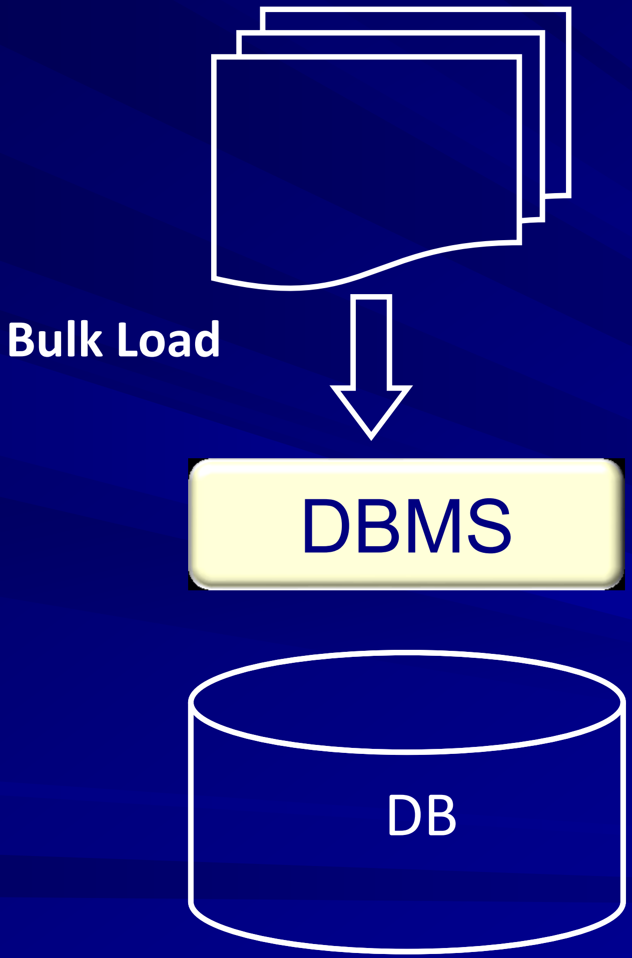
DBMS

Data

```
Update PIB_account
Set money = money - 500
Where cID = 1235817
```

```
Update PIB_account
Set money = money + 759
Where cID = 1235817
```

Bulk Inserts



```
INSERT INTO Studio(name)
   SELECT DISTINCT
   studioName
        FROM Movie
        WHERE studioName NOT
   IN
                (SELECT Name
                 FROM Studio);
```

Failure

- Example: *Transfer an amount of money* between two accounts
  - Checks that the first has at least that much money
  - If so, transfers money from first account to second
- What if failure occurs between two updates?
- Database will be left in inconsistent state
- See that certain operations needs to be done *atomically*
  - Either all operations complete or none goes through

Solution

Solution to previous problems is to group database operations into *transactions*

- **A transaction is a sequence of one or more (DML) SQL operations treated as a unit**

ACID Transactions

A DBMS is expected to support ”ACID transactions”, which are:

- **Atomic**: Either the whole transaction is run, or nothing
- **Consistent**: Database constraints are preserved.
- **Isolated**: Different transactions may not interact with each other.
- **Durable**: Effects of a transaction are not lost in case of a system crash.

System Log

- Remember, DBMS must assure that we don’t loose information due to system crashes
  - i.e., how do we recover from failure?
- Keep system log
  - Kept on disk, backed up periodically
  - Record every action

Controlling transactions

We can explicitly start transactions using the START TRANSACTION statement, and end them using COMMIT or ROLLBACK:

- COMMIT causes an SQL transaction to complete successfully.
  - Any modifications done by the transaction are now permanent in the database.
- ROLLBACK causes an SQL transaction to end by aborting it.
  - Any modifications to the database must be undone.
  - Rollbacks could be caused implicitly by errors.