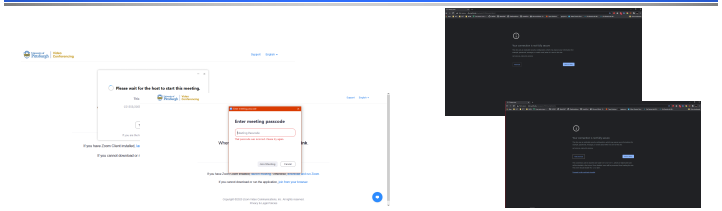
Announcements



- Zoom links issues on Friday
  - Zoom link for Recitations now available on Canvas and the website
  - No credit is lost ☺
- Website TLS issue
  - Please click the advance option and click Proceed to db.cs.pitt.edu (unsafe)

Overview of Database Management Systems

Part II

Database Vs. File Systems Approaches

- **Abstraction**
- Reliability
- Efficiency/Performance

Data Abstraction

- Data are structured in a way meaningful to applications
- **Data Model:**
  - A collection of high-level data description constructs that hide low-level storage details
- **The Relational / Object-Relational Model:**
  - Is the most widely used data model today
  - Main construct is a ***relation***: table of records
  - Every relation has a ***schema***:
    - Relation name
    - Names of fields
    - Types of fields

Example

Students

Attribute or Field

Schema

Record or Tuple

SID	*Name*	*Age*	*GPA*
546007	Peter	18	3.8
546100	Bob	19	3.65
546500	Bill	20	3.7

- Schema:
  - Students (*sid:* `string`, *name*: `string`, *age*: `integer`, *gpa*: `real`)
- State: Actual data at a given point in time

Database Languages

- **Data Definition Language (*DDL*):**
  - Define schemas
  - Define **Integrity Constraints**
    - Example: unique *SID*s
  - More…
- **Data Manipulation Language (*DML*):**
  - To ask questions = ***Query***
    - Example: Which students have GPA > 3.75?
  - To create and modify data
- ***SQL:*** Most widely used database language

Good DBMS ≠ Good Design!

Attribute or Field

Schema

Record or Tuple

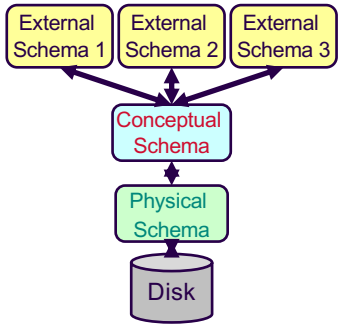
SID	*Name*	*Age*	*GPA*
546007	Peter	18	3.8
546100	Bob	19	3.65
546500	Bill	20	3.7

- Schema:
  - Students (*sid:* `string`, *name*: `string`, *age*: `integer`, *gpa*: `real`)
  - Alternative Schema:
    - Students (*sid*: `integer`, *fname*: `string`, *lname*: `string`, *dob*: `date`, *gpa*: `real`)

Now we know the color of the Database

Levels of Data Abstraction in a DBMS

- The data in a DBMS is described at three levels of abstraction:
  1. *Conceptual Schema*
     DDL: data definition language
  2. *Physical Schema*
     SDL: storage definition language
  3. *External Schema* (Views)
     VDL: view definition language



- Many external schemas
- plus one conceptual
- plus one physical

External Schema - Views

- Allows data access to be customized and authorized at the user-level
  - Defined in terms of data model
  - Consists of a collection of **views**
  - Example:
    - CourseEnroll (*cid*: `string`, *enrollment*: `int`)

CID	*Enroll*
CS 1555	37
CS 2550	17
CS 3551	15

- Guided by end-user requirements
- Views are computed as needed
- Multiple Views of data allows each user/application to get different perspective of the database

3-level Architecture

- view level
  - CSMajors
  - MathMajors
- logical level: entire database schema
  - Courses (CourseNo,CourseName,Credits,Dept)
  - Students (StudentID,Lname,Name,Class,Major)
  - GradeReport (StudentID,CourseNo,Grade,Term)
- physical level:
  - how these tables are stored
  - how many bytes and attribute, etc.

Execution Abstraction

- *A **transaction** is a **logical unit of work** in DBMSs*
  - *It is the execution of a **program segment** that performs some function or task by accessing shared data (e.g., a db)*
  - logical grouping of query and update requests needed to perform a task
- Examples:
  - deposit, withdraw, transfer money (banking transaction)
  - reserve a seat on a flight (airline reservation)
  - print monthly payment checks (business transaction)
  - update inventory (inventory transaction)

ACID Properties

- **Atomicity** (alias failure atomicity)
  Either all the operations associated with a transaction happen or none of them happens
- **Consistency Preservation**
  A transaction is a correct program segment. It satisfies the integrity constraints on the database at the transaction's boundaries
- **Isolation** (alias concurrency atomicity / serializability)
  Transactions are independent, the result of the execution of concurrent transactions is the same as if transactions were executed serially, one after the other
- **Durability** (alias persistence / permanence)
  The effects of completed transactions become permanent surviving any subsequent failures

Transfer Money Example

- Two accounts:
  - A: $100
  - B: $200
- Client 1: transfer $6 from A to B

Client 1
Read A: $100
Write A: $94
Read B: $200
Write B: $206

ACID Properties

- **Atomicity** (alias failure atomicity)
  Either all the operations associated with a transaction happen or none of them happens
- **Consistency Preservation**
  A transaction is a correct program segment. It satisfies the integrity constraints on the database at the transaction's boundaries
- **Isolation** (alias concurrency atomicity / serializability)
  Transactions are independent, the result of the execution of concurrent transactions is the same as if transactions were executed serially, one after the other
- **Durability** (alias persistence / permanence)
  The effects of completed transactions become permanent surviving any subsequent failures

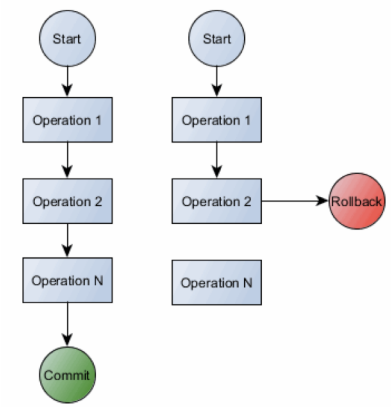
Transfer Money Example [Poll]

- Three accounts:
  - A: $100
  - B: $200
  - C: $300
- Client 1: transfer $6 from A to B
- Client 2: transfer $4 from C to B

Client 1	Client 2
Read A: $100	
Write A: $94	
	Read C: $300
	Write C: $296
Read B: $200	
	Read B: $200
	Write B: $204
Write B: $206	

SQL TRANSACTIONS

- Begin:
  - Each SQL statement should *implicitly* start a transaction, unless one is active.
- COMMIT [WORK]; or END [WORK];
- ROLLBACK [WORK] ;
  - ROLLBACK default action



Sample Transfer Money SQL code

```
-- Client 1

BEGIN;

UPDATE ACCOUNT
SET BALANCE = BALANCE - 6
WHERE NAME = 'A';

UPDATE ACCOUNT
SET BALANCE = BALANCE + 6
WHERE NAME = 'B';

COMMIT;
```

```
-- Client 2

BEGIN;

UPDATE ACCOUNT
SET BALANCE = BALANCE - 4
WHERE NAME = 'C';

UPDATE ACCOUNT
SET BALANCE = BALANCE + 4
WHERE NAME = 'B';

COMMIT;
```


ACID in NoSQL Databases

- **ACID** to **BaSE**: No immediate consistency, data freshness & accuracy
  - ***Basic Availability:*** The database appears to work most of the time.
  - ***Soft-state:*** Stores don't have to be write-consistent, nor do different replicas have to be mutually consistent all the time.
  - ***Eventual consistency:*** Stores exhibit consistency at some later point (e.g., lazily at read time).