Lecture 02
The Big Picture

Ilchul Yoon
Assistant Professor
State University of New York, Korea

CSE 532 – Theory of Database Systems

Databases
- Our interest - relational databases
- Data is stored in tables.

Table
- Set of rows (no duplicates)
- Each row - a different entity
- Each column - a particular fact about each entity
  - Each column has an associated domain

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Address</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>John</td>
<td>123 Main</td>
<td>fresh</td>
</tr>
<tr>
<td>2222</td>
<td>Mary</td>
<td>321 Oak</td>
<td>soph</td>
</tr>
<tr>
<td>1234</td>
<td>Bob</td>
<td>444 Pine</td>
<td>soph</td>
</tr>
<tr>
<td>9999</td>
<td>Joan</td>
<td>777 Grand</td>
<td>senior</td>
</tr>
</tbody>
</table>

- Domain of Status = \{fresh, soph, junior, senior\}

Relation
- Mathematical entity corresponding to a table
  - row ~ tuple
  - column ~ attribute

- Values in a tuple are related to each other
  - John is a freshman and lives at 123 Main

- Relation R as predicate R
  - $R(x,y,z)$ is true iff tuple $(x,y,z)$ is in $R$
Operations

- Operations on relations are precisely defined
  - Take relation(s) as argument, produce new relation as result
  - Unary (e.g., delete certain rows)
  - Binary (e.g., union, Cartesian product)
- Corresponding operations defined on tables as well
- Using mathematical properties, equivalence can be decided
  - Important for query optimization:
    $$\text{op}_1(\text{op}_2(\text{T}_1),\text{T}_2) = \text{op}_3(\text{op}_2(\text{T}_1),\text{T}_2)$$

Structured Query Language: SQL

- Language for manipulating tables
- Declarative – Statement specifies what needs to be obtained, not how it is to be achieved
  - e.g., how to access data, the order of operations
- DBMS determines evaluation strategies for query processing and optimization
  - Simplifies application programs
  - But DBMS is not infallible
    - Programmers must understand the mechanism behind SQL for better design and statements

Structured Query Language (SQL)

SELECT <attribute list>
FROM <table list>
WHERE <condition>

- Language for constructing a new table from argument table(s).
  - FROM - source table(s)
  - WHERE - which rows to retain (Filtering)
  - SELECT - which columns to keep from retained rows (Projection)
- The result is also a table.

Example

```
SELECT Name
FROM Student
WHERE Id > 4999
```

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Address</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>John</td>
<td>123 Main</td>
<td>fresh</td>
</tr>
<tr>
<td>5522</td>
<td>Mary</td>
<td>77 Pine</td>
<td>senior</td>
</tr>
<tr>
<td>9876</td>
<td>Bill</td>
<td>83 Oak</td>
<td>junior</td>
</tr>
</tbody>
</table>

Result

Student

<table>
<thead>
<tr>
<th>Name</th>
<th>Mary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bill</td>
<td></td>
</tr>
</tbody>
</table>
### Examples

- SELECT Id, Name FROM Student
- SELECT Id, Name FROM Student WHERE Status = 'senior'
- SELECT * FROM Student WHERE Status = 'senior'
- SELECT COUNT(*) FROM Student WHERE Status = 'senior'

**Result:**
- A table with one column and one row

### More Complex Example

- **Goal:** table in which each row names a senior and gives a course taken and grade
- **Combines information in two tables:**
  - Student: Id, Name, Address, Status
  - Transcript: StudId, CrsCode, Semester, Grade

```
SELECT Name, CrsCode, Grade
FROM Student, Transcript
WHERE StudId = Id AND Status = 'senior'
```

### Join

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>a2</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>17</td>
</tr>
</tbody>
</table>

- **SELECT a1, b1 FROM T1, T2 WHERE a2 = b2**
- **FROM T1, T2 yields:**
  - A 1 xxy 3.2 17
  - A 1 xxy 4.8 17
  - B 17 rst 3.2 17
  - B 17 rst 4.8 17
- **WHERE a2 = b2 yields:**
  - B 17 rst 3.2 17
  - B 17 rst 4.8 17

- **SELECT a1, b1 yields result:**
  - B 3.2
  - B 4.8

### Modifying Tables

- **UPDATE Student**
  - SET Status = 'soph'
  - WHERE Id = 111111111

- **INSERT INTO Student (Id, Name, Address, Status)**
  - VALUES (999999999, 'Bill', '432 Pine', 'senior')

- **DELETE FROM Student**
  - WHERE Id = 111111111
Creating Tables

```sql
CREATE TABLE Student
(
  Id INTEGER,
  Name CHAR(20),
  Address CHAR(50),
  Status CHAR(10),
  PRIMARY KEY (Id)
)
```

Integrity Constraints

- Rules (or limitations) enforced by the enterprise
- Generally, limit the occurrence of certain real-world events.
- Student cannot register for a course if current number of registrants = maximum allowed
- Allowable database states are restricted
  - `cur_reg <= max_reg`
- Expressed as integrity constraints
  - assertions that must be satisfied by the database state.

Transactions

- Many enterprises use databases to store information about their state
  - E.g., balances of all depositors

- Real world event \( \rightarrow \) corporate database update
  - requires the execution of a program that changes the database state in a corresponding way
  - E.g., balance must be updated when you deposit

- A transaction is a program that accesses the database in response to real-world events

Transactions

- Transactions are not just ordinary programs
- Additional requirements
  - Atomicity
  - Consistency
  - Isolation
  - Durability
  - **ACID properties**
Atomicity

- A real-world event either happens or does not happen.
  - Student either registers or does not register.

- Whether the transaction runs to completion (commits) or,
- If it does not complete, it has no effect at all (aborts).

Consistency

- Transaction designer must ensure
  - IF the database is in a state that satisfies all integrity constraints when execution of a transaction is started
  - THEN when the transaction completes:
    - All integrity constraints are once again satisfied (constraints can be violated in intermediate states)
    - New database state satisfies specifications of transaction

Isolation

- Deals with concurrent transaction execution
  - If the initial database state is consistent and accurately reflects the real-world state,
  - then the serial (one after another) execution of a set of consistent transactions will preserve consistency.
  - However…. Serial execution is inadequate from a performance perspective.

- Overall effect of the transaction schedule must be the same as if the transactions had executed serially in some order.
  - The execution is thus not serial, but serializable

Concurrent Transaction Execution
Isolation

- Concurrent (interleaved) transaction execution offers performance benefits, but might not be correct.

- Example: Two students execute the course registration transaction at about the same time
  - cur_reg is the number of current registrants

  \[
  \begin{align*}
  T_1: & \text{read}(\text{cur}_\text{reg} : 29) \quad \text{write}(\text{cur}_\text{reg} : 30) \\
  T_2: & \text{read}(\text{cur}_\text{reg} : 29) \quad \text{write}(\text{cur}_\text{reg} : 30)
  \end{align*}
  \]

  Result: Database state no longer corresponds to real-world state, integrity constraint violated.

Durability

- Once a transaction commits, its effect on the database state is not lost in spite of subsequently computer crashes.

ACID Properties

- The transaction monitor is responsible for ensuring atomicity, durability, and (the requested level of) isolation.
  - Hence it provides the abstraction of failure-free, non-concurrent environment, greatly simplifying the task of the transaction designer.

- The transaction designer is responsible for ensuring the consistency of each transaction, but doesn’t need to worry about concurrency and system failures.

Data and Its Structure

- Schema: Description of data at some abstraction level. Each level has its own schema.
  - We will be concerned with three schemas: physical, conceptual, and external.
Physical Data Level

- **Physical schema** describes details of how data is stored
  - tracks, cylinders, indices etc.
  - Early applications worked at this level – explicitly dealt with details.
- **Problem:**
  - Routines were hard-coded to deal with physical representation.
  - Changes to data structure difficult to make.
  - Application code becomes complex since it must deal with details.
  - Rapid implementation of new features impossible.

Conceptual Data Level

- Hides details.
  - In the relational model, the **conceptual schema** presents data as a set of tables (or relations).
  - DBMS maps from conceptual to physical schema automatically.
- **Physical schema can be changed without changing application:**
  - DBMS would change mapping from conceptual to physical transparently
  - This property is referred to as **physical data independence**

Conceptual Data Level (con’t)

Application

\[\text{Conceptual view of data}\]

DBMS

\[\text{Physical view of data}\]

External Data Level

- In the relational model, the **external schema** also presents data as a set of relations.
- An external schema specifies a **view** of the data in terms of the conceptual level. It is tailored to the needs of a particular category of users.
  - Portions of stored data should not be seen by some users.
    - Students should not see their files in full.
    - Faculty should not see billing data.
  - Information that can be derived from stored data might be viewed as if it were stored.
    - GPA not stored, but calculated when needed.
External Data Level (con’t)
- Application is written in terms of an external schema.
- A view is computed when accessed (not stored).
- Different external schemas can be provided to different categories of users.
- Translation from external to conceptual done automatically by DBMS at run time.
- Conceptual schema can be changed without changing application:
  - Mapping from external to conceptual must be changed.
  - Referred to as conceptual data independence.

ANSI-SPARC 3-level Architecture (1975)
- External Level
  - Multiple independent users or applications
  - Users' view of the database
  - Focus on each user or application

- Conceptual Level
  - Community view of the database
  - Describes what data is stored in database and relationships among the data
  - Focus on the organization

- Internal Level
  - Physical representation of the database on the computer
  - Describes how the data is stored in the database
  - Focus on the DBMS