CSE 305 / CSE532

Lecture 02
The Big Picture

Lecturer: Sael Lee

Slide adapted from the author’s slides and Dr. Ilchul Yoon’s slides.
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:

\[
op_1(T_1, \op_2(T_2)) = \op_3(\op_2(T_1), T_2)
\]
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

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

SELECT Id, Name FROM Student

SELECT Id, Name FROM Student
WHERE Status = 'senior'

SELECT * FROM Student
WHERE Status = 'senior'

result is a table with one column and one row

SELECT COUNT(*) FROM Student
WHERE Status = 'senior'
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

```
SELECT a1, b1
FROM T1, T2
WHERE a2 = b2
```

<table>
<thead>
<tr>
<th></th>
<th>a1</th>
<th>a2</th>
<th>a3</th>
<th></th>
<th>b1</th>
<th>b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>xxy</td>
<td></td>
<td></td>
<td>3.2</td>
<td>17</td>
</tr>
<tr>
<td>B</td>
<td>17</td>
<td>rst</td>
<td></td>
<td></td>
<td>4.8</td>
<td>17</td>
</tr>
</tbody>
</table>

FROM T1, T2

WHERE a2 = b2

yields:

```
SELECT a1, b1
FROM T1, T2
WHERE a2 = b2
```

<table>
<thead>
<tr>
<th></th>
<th>a1</th>
<th>a2</th>
<th>a3</th>
<th></th>
<th>b1</th>
<th>b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>xxy</td>
<td></td>
<td></td>
<td>3.2</td>
<td>17</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>xxy</td>
<td></td>
<td></td>
<td>4.8</td>
<td>17</td>
</tr>
<tr>
<td>B</td>
<td>17</td>
<td>rst</td>
<td></td>
<td></td>
<td>3.2</td>
<td>17</td>
</tr>
<tr>
<td>B</td>
<td>17</td>
<td>rst</td>
<td></td>
<td></td>
<td>4.8</td>
<td>17</td>
</tr>
</tbody>
</table>

WHERE a2 = b2

yields:

```
SELECT a1, b1
FROM T1, T2
WHERE a2 = b2
```

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>b1</th>
<th>b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>3.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>4.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

SELECT a1, b1

yields result:
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

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
    - \( \text{cur\_reg} \leq \text{max\_reg} \)

- 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

\[
\begin{align*}
\text{Atomicity} \\
\text{Consistency} \\
\text{Isolation} \\
\text{Durability}
\end{align*}
\]

\textit{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

sequence of database operations output by T_1

sequence of database operations input to dbms

local variables

computation
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

\[
T_1: \text{read}(cur\_reg: 29) \quad \text{write}(cur\_reg: 30)
\]
\[
T_2: \quad \text{read}(cur\_reg: 29) \quad \text{write}(cur\_reg: 30)
\]

\[\text{time} \rightarrow\]

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

DBMS

Conceptual view of data

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**
  - User 1: View 1
  - User 2: View 2
  - User n: View n

- **Conceptual level**
  - Conceptual schema

- **Internal level**
  - Internal schema

- **Physical data organization**
  - Database

- **Structural Data**
  ```c
  struct STAFF {
      int staffNo;
      int branchNo;
      char fName [15];
      char lName [15];
      struct date dateOfBirth;
      float salary;
      struct STAFF *next;
  }
  ```

  - index staffNo; index branchNo;
ANSI-SPARC 3-level Architecture

- **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
ANSI-SPARC 3-level Architecture

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

```c
struct STAFF {
    int staffNo;
    int branchNo;
    char fName [15];
    char lName [15];
    struct date dateOfBirth;
    float salary;
    struct STAFF *next;
};
index staffNo; index branchNo;
```
CSE 305 / CSE532

Lecture 03
The Big Picture

Ilchul Yoon
Assistant Professor
State University of New York, Korea

Adapted from book authors’ slides
Data Model

- **Schema**: description of data at some level
  - e.g., tables, attributes, constraints, domains

- **Model**: tools and language for describing:
  - Conceptual and external *schema*
    - Data definition language (DDL)
  - Integrity *constraints*, domains (DDL)
  - *Operations* on data
    - Data manipulation language (DML)
  - Directives that influence the physical schema (affects performance, not semantics)
    - Storage definition language (SDL)
Relational Model

- A particular way of structuring data (using relations)
- Simple
- Mathematically based
  - Expressions (\(\equiv\) queries) can be analyzed by DBMS
  - Queries are transformed to equivalent expressions automatically (query optimization)
    - Optimizers have limits
Relation Instance

- Relation is a set of tuples
  - Atomic values
  - Tuple ordering is unimportant
  - No duplicates
  - **Cardinality** of relation = number of tuples

- All tuples in a relation have the same structure; constructed from the same set of attributes
  - Attributes are named (ordering is immaterial)
  - Value of an attribute is drawn from the attribute’s **domain**
    - There is also a special value **null** (value unknown or undefined), which belongs to no domain
  - **Arity** (or degree) of relation = number of attributes
# Relation Instance (Example)

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Address</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111111</td>
<td>John</td>
<td>123 Main</td>
<td>freshman</td>
</tr>
<tr>
<td>2345678</td>
<td>Mary</td>
<td>456 Cedar</td>
<td>sophomore</td>
</tr>
<tr>
<td>4433322</td>
<td>Art</td>
<td>77 So. 3rd</td>
<td>senior</td>
</tr>
<tr>
<td>7654321</td>
<td>Pat</td>
<td>88 No. 4th</td>
<td>sophomore</td>
</tr>
</tbody>
</table>

Student
Relation Schema

- Relation name
- Attribute names & domains
- Integrity constraints like
  - The values of a particular attribute in all tuples are unique
  - The values of a particular attribute in all tuples are greater than 0
- Default values
Relational Database

- Finite set of relations
- Each relation consists of a schema and an instance

- **Database schema** = set of relation schemas constraints among relations (*inter-relational constraints*)
- **Database instance** = set of (corresponding) relation instances
Database Schema (Example)

- Student (Id: INT, Name: STRING, Address: STRING, Status: STRING)
- Professor (Id: INT, Name: STRING, DeptId: DEPTS)
- Course (DeptId: DEPTS, CrsName: STRING, CrsCode: COURSES)
- Transcript (CrsCode: COURSES, StudId: INT, Grade: GRADES, Semester: SEMESTERS)
- Department(DeptId: DEPTS, Name: STRING)
Integrity Constraints

- Part of schema
- Restriction on state (or of sequence of states) of database
- Enforced by DBMS
- **Intra-relational** - involve only one relation
  - Part of relation schema
  - e.g., all IDs are unique
- **Inter-relational** - involve several relations
  - Part of relation schema or database schema
Constraint Checking

- Automatically checked by DBMS
- Protects database from errors
- Enforces enterprise rules
Kinds of Integrity Constraints

- Static – restricts legal states of database
  - Syntactic (structural)
    - e.g., all values in a column must be unique (atomic values)
  - Semantic (involve meaning of attributes)
    - e.g., cannot register for more than 18 credits

- Dynamic – limitation on sequences of database states
  - e.g., cannot raise salary by more than 5%
Key Constraint

- A **key constraint** is a sequence of attributes $A_1, \ldots, A_n$ of a relation schema, $S$, with the following property:
  - A relation instance $s$ of $S$ satisfies the key constraint *iff* at most one row in $s$ can contain a particular (or unique) set of values, $a_1, \ldots, a_n$, for the attributes $A_1, \ldots, A_n$

- **Minimality**: no subset of $A_1, \ldots, A_n$ satisfies the key constraint

- **Key**
  - Set of attributes mentioned in a key constraint
    - e.g., Id in **Student**
    - e.g., (StudId, CrsCode, Semester) in **Transcript**
  - It is minimal: no subset of a key is a key
    - (Id, Name) is not a key of **Student**
Key Constraint (cont’d)

- **Superkey** - set of attributes containing key
  - (Id, Name) is a superkey of Student
- Every relation has a key
- Relation can have several keys:
  - **Primary key**: Id in Student (can’t be null)
  - **Candidate key**: (Name, Address) in Student
Foreign Key Constraint

- **Referential integrity**: Item named in one relation must refer to tuples that describe that item in another
  - Transcript (CrsCode) references Course (CrsCode)
  - Professor(DeptId) references Department (DeptId)

- Attribute $A_1$ is a **foreign key** of $R1$ referring to attribute $A_2$ in $R2$, if whenever there is a value $v$ of $A_1$, there is a tuple of $R2$ in which $A_2$ has value $v$, and $A_2$ is a key of $R2$
  - This is a special case of referential integrity: $A_2$ must be a candidate key of $R2$ (e.g., CrsCode is a key of Course in the above)
  - If no row exists in $R2$ => violation of referential integrity
  - Not all rows of $R2$ need to be referenced: relationship is not symmetric (e.g., some course might not be taught)
  - Value of a foreign key might not be specified (DeptId column of some professor might be null)
Foreign Key Constraint (Example)

A_1

v1
v2
v3
v4
null
v3

A_2

v3
v5
v1
v6
v2
v7
v4

R1

Foreign key

R2

Candidate key
Foreign Key (cont’d)

- Names of the attributes $A_1$ and $A_2$ can be different.
  - With tables:
    
    $\text{Teaching}(\text{CrsCode}: \text{COURSES}, \text{Sem}: \text{SEMESTERS}, \text{ProfId}: \text{INT})$
    
    $\text{Professor}(\text{Id}: \text{INT}, \text{Name}: \text{STRING}, \text{DeptId}: \text{DEPTS})$
  
  - $\text{ProfId}$ attribute of $\text{Teaching}$ references $\text{Id}$ attribute of $\text{Professor}$

- $R_1$ and $R_2$ need not be distinct.
  
  - $\text{Employee}(\text{Id}: \text{INT}, \text{MgrId}: \text{INT}, \text{....})$
    
    - $\text{Employee}(\text{MgrId})$ references $\text{Employee}(\text{Id})$
  
  - Every manager is also an employee and hence has a unique row in $\text{Employee}$
Foreign Key (cont’d)

- Foreign key might consist of several columns 
  (CrsCode, Semester) of Transcript references 
  (CrsCode, Semester) of Teaching

- \( R_1(A_1, \ldots A_n) \) references \( R_2(B_1, \ldots B_n) \)
  - \( A_i \) and \( B_i \) must have same domains (although not necessarily the same names)
  - \( B_1, \ldots, B_n \) must be a candidate key of \( R_2 \)
Inclusion Dependency

- Referential integrity constraint that is not a foreign key constraint
  
  \((\text{CrsCode}, \text{Semester})\) of Teaching references
  
  \((\text{CrsCode}, \text{Semester})\) of Transcript

- Target attributes is not a CK in Transcript

- No simple enforcement mechanism for inclusion dependencies in SQL (requires assertions)