DSA5104 Principles of Data Management and Retrieval

AY2025/26 Sem1 By Zhao Peiduo

Lecture 1

Database Systems
- DBMS: interrelated data + programs; convenient and efficient environment.
- Manage data that are highly valuable, large, and concurrently accessed.
- Modern DBMSs manage large, complex collections of data; pervasive in daily life.

Purpose of Database Systems
- Redundancy & inconsistency (multiple file formats); difficulty accessing data (new program per task).
- Data isolation (multiple files/formats) → security challenges.
- Integrity constraints buried in code → hard to add/change.
- *Atomicity*: no partial updates; e.g., fund transfer must be all-or-nothing.
- *Concurrency*: needed for performance; uncontrolled access → inconsistencies (e.g., two withdrawals).
- *Security*: restrict access to subsets of data.
- DB systems address these issues; store & retrieve data safely.

View of Data
- DB system = interrelated data + programs to access/modify.
- Provide abstract view via *data models* (concepts, relationships, semantics, constraints).

Categories of Data Models (high level)
- Relational (tables)
- Entity–Relationship (design).
- Object-based (OO/OR features).
- Semi-structured (XML/JSON).

Instances and Schemas
- *Schema*: overall design. *Instance*: data at a moment.
- Analogy: schema - variable declaration; instance - current value (class/struct blueprint vs object).

Logical vs Physical Schema & Physical Data Independence
- Logical schema: what data/relationships. Physical schema: storage layout.
- Physical data independence: change physical without changing logical; well-defined interfaces.

Data Definition Language (DDL)
- Define schema; DDL compiler → templates in data dictionary (metadata: schema, constraints, auth).
- Example: `create table instructor (ID char(5), name varchar(20), dept_name varchar(20), salary numeric(8,2))`

Data Manipulation Language (DML)
- Access/update data; *procedural* (what + how) vs *declarative* (what).
- Declarative DMLs easier; query-language part handles retrieval.

SQL Query Language
- Nonprocedural; input tables → one output table.
- Typically embedded or called via APIs (ODBC/JDBC); app code handles I/O/network/UI.

Engine / Components (very high level)
- Storage manager (file/buffer/authorization/transaction). Query processor (DDL interpreter, DML compiler/optimizer, eval engine).
- Query Processing stages: Parsing & translation Optimization Evaluation

Transaction Management
- Transaction = logical unit of work (e.g., transfer \$50: read/update/write A,B).
- Ensure consistency under failures; concurrency control coordinates overlapping txns.

Architectures
- Centralized/shared-memory; Client–server; Parallel (shared-memory/disk/nothing); Distributed (geo, heterogeneity).
- App tiers: two-tier (client-DB) vs three-tier (client-app server-DB); 3-tier aids dev, scale, reliability, security.

Lecture 2

Relation Schema and Instance
- $A_1, A_2, \ldots, A_n$ are *attributes*.
- $R = (A_1, A_2, \ldots, A_n)$ is a *relation schema*.
- Example: `instructor = (ID, name, dept_name, salary)`.
- A relation instance r defined over schema R is denoted by $r(R)$.
- The current values of a relation are specified by a table.
- An element t of relation r is called a **tuple**, represented by a row in a table.

Attributes
- The set of allowed values for each attribute is its **domain**.
- Attribute values are required to be **atomic** (indivisible).
- Non-atomic example: concatenation of multiple attribute values, e.g. Silberschatz, Korth, Sudarshan for author. This should be broken into several atomic rows with one author each.
- Special value `null` indicates "unknown"; member of every domain, which could complicate operations.

Relations are Unordered
- Order of tuples is irrelevant; tuples may be stored arbitrarily.
- Example: `instructor` relation with unordered tuples.

Keys: Let $K \subseteq R$
- **Superkey:** K is a superkey if values for K uniquely identify a tuple (unique identifier).
  - Example: {`ID`}, {`ID, name`} are both superkeys of `instructor`.
  - **SQL:** Every declared `PRIMARY KEY` or `UNIQUE` constraint implicitly defines a superkey.
- **Candidate key:** Minimal superkey (only containing elements essential for unique identification).
  - Example: {`ID`} is a candidate key for `instructor`.
  - **SQL:** Use `UNIQUE` to enforce candidate keys.

```
create table instructor (
    ID varchar(5),
    name varchar(20),
    dept_name varchar(20),
    salary numeric(8,2),
    unique (name, dept_name)  -- candidate key
);
```

Keys: Let $K \subseteq R$
- **Primary key:** One candidate key chosen to uniquely identify tuples.
  - Example: `ID` is chosen as the primary key.
  - **SQL:**

```
create table instructor (
    ID varchar(5) primary key,
    name varchar(20),
    dept_name varchar(20),
    salary numeric(8,2)
);
```

- **Foreign key:** Attribute in one relation that refers to the primary key in another relation.
  - Example: `dept_name` in `instructor` refers to `department.dept_name`.
  - **SQL:**

```
create table department (
    dept_name varchar(20) primary key,
    building varchar(20),
    budget numeric(12,2)
);

create table instructor (
    ID varchar(5) primary key,
    name varchar(20),
    dept_name varchar(20),
    salary numeric(8,2),
    foreign key (dept_name) references department(dept_name)
);
```

Relational Query Languages
- SQL is mostly **non-procedural**: user specifies what, DB decides how.
- Three formal relational query languages:
  - Relational algebra (procedural).
  - Tuple relational calculus (non-procedural).
  - Domain relational calculus (non-procedural).

Relational Algebra
- Procedural language: operations on relations produce new relations.
- Six basic operators:
  - **Select** (σ) – filter rows.
  - **Project** (Π) – choose attributes.
  - **Union** ($\cup$).
  - **Set difference** ($-$).
  - **Cartesian product** ($\times$).
  - **Rename** (ρ).

Select Operation
- Selects tuples that satisfy a given **predicate**.
- Notation: $\sigma_p(r)$ where p is the **selection predicate**.
- Comparisons: $=, \neq, >, \geq, <, \leq$.
- Predicates can be combined: $\wedge$ (AND), $\vee$ (OR), $\neg$ (NOT).
- Example: $\sigma_{\text{dept\_name} = \text{"Physics"} \wedge \text{salary} > 90000}(\text{instructor})$

```
SELECT *
FROM instructor
WHERE dept_name = 'Physics' AND salary > 90000;
```

- Example: $\sigma_{\text{dept\_name} = \text{building}}(\text{department})$

```
SELECT *
FROM department
WHERE dept_name = building;
```

Project Operation (Subsetting)
- RA Notation: $\Pi_{A_1, A_2, \ldots, A_k}(r)$.
- Example: $\Pi_{ID, name, salary}(\text{instructor})$

```
SELECT DISTINCT ID, name, salary
FROM instructor;
```

Composition of Operations
- Example: $\Pi_{\text{name}}(\sigma_{\text{dept\_name} = \text{"Physics"}}(\text{instructor}))$

```
SELECT DISTINCT name
FROM instructor
WHERE dept_name = 'Physics';
```

Cartesian-Product Operation
- Example: instructor $\times$ teaches

```
SELECT *
FROM instructor
CROSS JOIN teaches;
```

Join Operation

- Example: $\sigma_{\text{instructor.ID} = \text{teaches.ID}}(\text{instructor} \times \text{teaches})$
- Equivalent: instructor $\bowtie_{\text{instructor.ID} = \text{teaches.ID}}$ teaches

```
SELECT *
FROM instructor
JOIN teaches
  ON instructor.ID = teaches.ID;
```

Union Operation

- Example: $\Pi_{\text{course\_id}}(\sigma_{\text{semester}=\text{“Fall”} \wedge \text{year}=2017}(\text{section})) \cup \Pi_{\text{course\_id}}(\sigma_{\text{semester}=\text{“Spring”} \wedge \text{year}=2018}(\text{section}))$
- ```
  SELECT DISTINCT course_id
  FROM section
  WHERE semester = 'Fall' AND year = 2017
  UNION
  SELECT DISTINCT course_id
  FROM section
  WHERE semester = 'Spring' AND year = 2018;
  ```

Set-Intersection Operation

- Example: $\Pi_{\text{course\_id}}(\sigma_{\text{semester}=\text{“Fall”} \wedge \text{year}=2017}(\text{section})) \cap \Pi_{\text{course\_id}}(\sigma_{\text{semester}=\text{“Spring”} \wedge \text{year}=2018}(\text{section}))$
- ```
  SELECT DISTINCT course_id
  FROM section
  WHERE semester = 'Fall' AND year = 2017
  INTERSECT
  SELECT DISTINCT course_id
  FROM section
  WHERE semester = 'Spring' AND year = 2018;
  ```

Set-Difference Operation

- $\Pi_{\text{course\_id}}(\sigma_{\text{semester}=\text{“Fall”} \wedge \text{year}=2017}(\text{section})) - \Pi_{\text{course\_id}}(\sigma_{\text{semester}=\text{“Spring”} \wedge \text{year}=2018}(\text{section}))$
- ```
  SELECT DISTINCT course_id
  FROM section
  WHERE semester = 'Fall' AND year = 2017
  EXCEPT
  SELECT DISTINCT course_id
  FROM section
  WHERE semester = 'Spring' AND year = 2018;
  ```

Assignment Operation

- Assigns the result of an expression to a temporary relation.
- Example: $C \leftarrow \sigma_{\text{dept\_name}=\text{'Physics'}}(\text{instructor})$
- ```
  CREATE TABLE C AS
  SELECT *
  FROM instructor
  WHERE dept_name = 'Physics';
  ```

Rename Operation

- Used to rename relations or attributes.
- Example: $\rho_X(E)$ renames the result of E to relation X.
- Example: $\rho_{Y(A1,A2,A3)}(E)$ renames E to relation Y with attributes $A1$, $A2$, $A3$.
- ```
  -- Rename relation
  SELECT *
  FROM instructor AS X;

  -- Rename relation + attributes
  SELECT ID AS A1, name AS A2, salary AS A3
  FROM instructor AS Y;
  ```

Equivalent Queries

- Different relational algebra expressions (or SQL queries) can produce the same result.
- Equivalence: $\sigma_{\text{dept\_name}=\text{'Physics'}}(\sigma_{\text{salary}>90000}(\text{instructor})) \equiv \sigma_{\text{dept\_name}=\text{'Physics'} \wedge \text{salary}>90000}(\text{instructor})$
- ```
  -- Two equivalent queries
  SELECT *
  FROM instructor
  WHERE dept_name = 'Physics' AND salary > 90000;

  SELECT *
  FROM (
    SELECT *
    FROM instructor
    WHERE salary > 90000
  ) AS temp
  WHERE dept_name = 'Physics';
  ```

Design of Database

- Relational algebra is the theoretical foundation of SQL.
- Operators (selection, projection, joins, set operations, rename, etc.) form the core building blocks.
- Database design should enable:
  - Expressive query formulation.
  - Efficient query execution.
  - Logical independence (queries written without depending on physical storage).