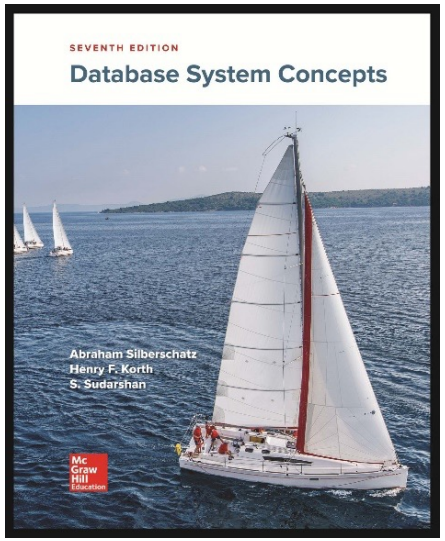
Introduction to Database
SWE3003-41
Lecture01 – Introduction to DBMS

Beomseok Nam (남범석)
bnam@skku.edu

Textbook & Piazza

- Database System Concepts (7th Ed.)
  - Avi Silberschatz
  - Henry F. Korth
  - S. Sudarshan
  - https://www.db-book.com/



- Q&A: Piazza
  - https://piazza.com/sungkyunkwan_university/spring2025/swe300341
  - Access Code:
    - **swe300341**

Exams and Projects

- Project 1 (20%)
  - Classic DB application using SQL
  - Java and PostgreSQL
- Midterm Exam (30%)
- Project 2 (20%)
  - More details will be available later
- Final Exam (30%)

Database vs. Database Management Systems

- What is Database?
  - Definition: A database is a **collection of data**

- What is DBMS (database management system)?
  - System software that manages database

Topics (SQL and DB Application Development)

- Ch 1. Introduction
- Ch 2. Relational Language
- Ch 3. Introduction to SQL
- Ch 4. Intermediate SQL
- Ch 5. Advanced SQL
- Ch 6. E-R Model
- Ch 7. Normal Forms and Functional Dependencies
- Ch 8. Data Types
- Midterm

Learn how to use DBMS

These knowledges may not be needed as AI will take over the role of application developers in the next few years.

Topics (How to Design DBMS)

- Ch 12. Storage Management
- Ch 13. Data Storage Structures
- Ch 14. Indexing
- Ch 15. Query Processing
- Ch 16. Query Optimization
- Ch 17. Transactions
- Ch 18. Concurrency Control
- Ch 19. Recovery Systems
- Final Exam

Learn how to build DBMS

Need DB Administrator when something goes wrong

Textbook Roadmap

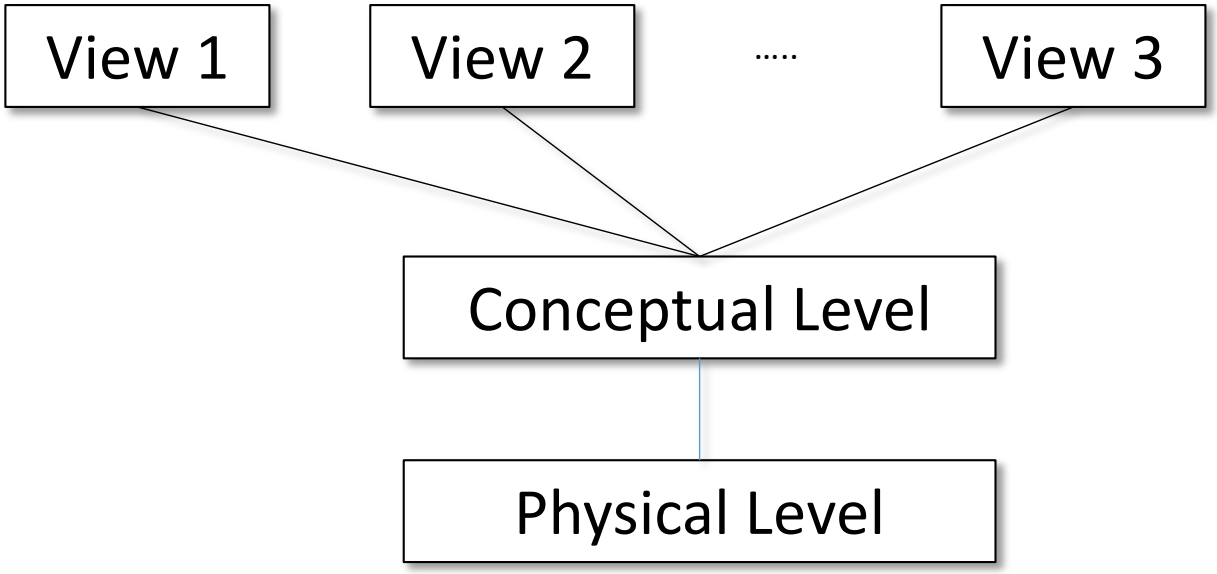
- **Part I: Relational Database**
- **Part II: Database Design**
- Part III: Data Storage
- Part IV: Query Optimization, Concurrency Control, etc

Database

- What do you need to develop a university management system?
  - Requirement analysis
    - Identify stakeholders
      - Students
      - Courses
      - Faculty
      - Departments
    - What actions will each stakeholder perform?
      - Students taking courses
      - Faculty teaching courses
      - Faculty advising students
  - Basically, what you are doing is ‘Data Abstraction’.

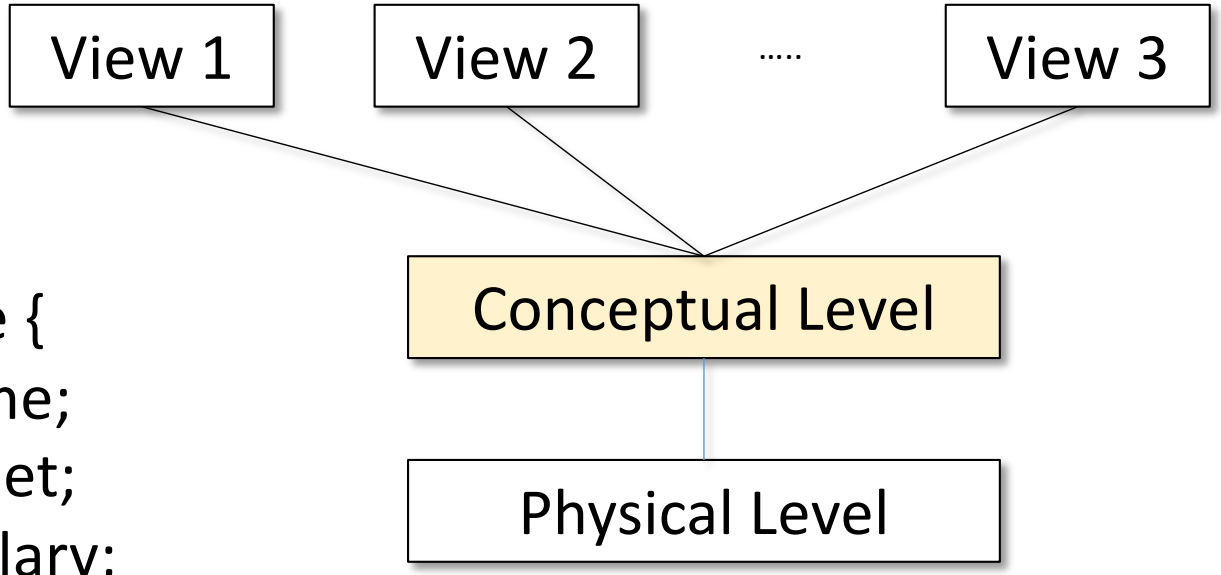
Data Abstraction

- 3 Levels of Data Abstraction



Data Abstraction

- 3 Levels of Data Abstraction



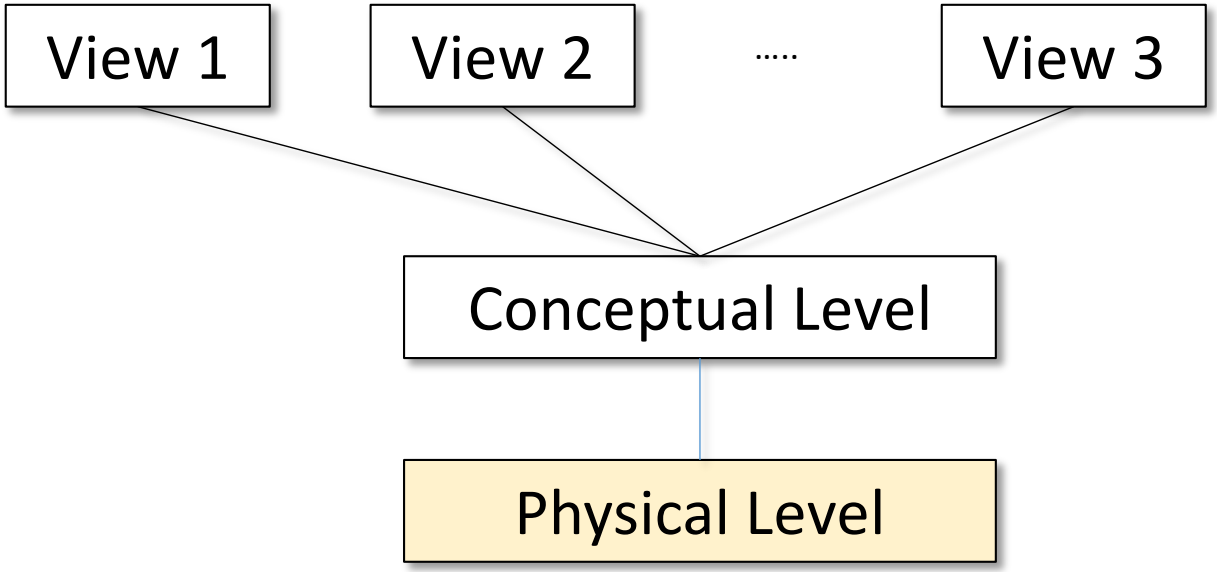
- **Conceptual level**
  - eg.

```
struct employee {
        string name;
        string street;
        integer salary;
};
```

  - This describes what an employee record looks like
    - it contains 3 fields of specified type
    - Does not specify how the record is stored

Data Abstraction

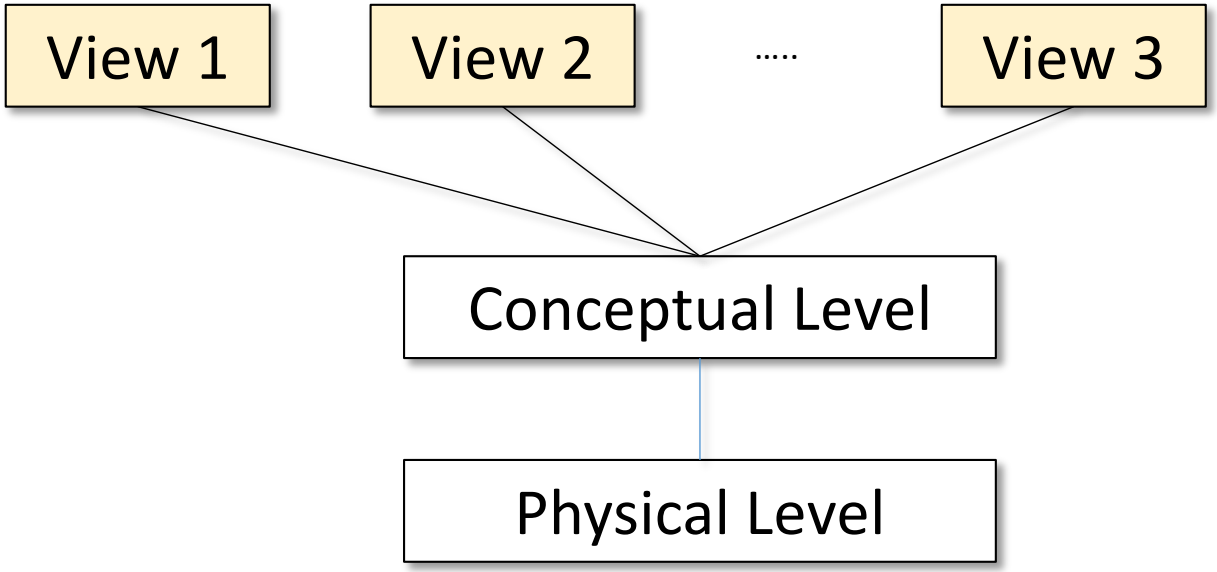
- 3 Levels of Data Abstraction



- **Physical level**
  - How an employee record is stored on storage
    - Eg. B-tree, Hash Table, etc

Data Abstraction

- 3 Levels of Data Abstraction

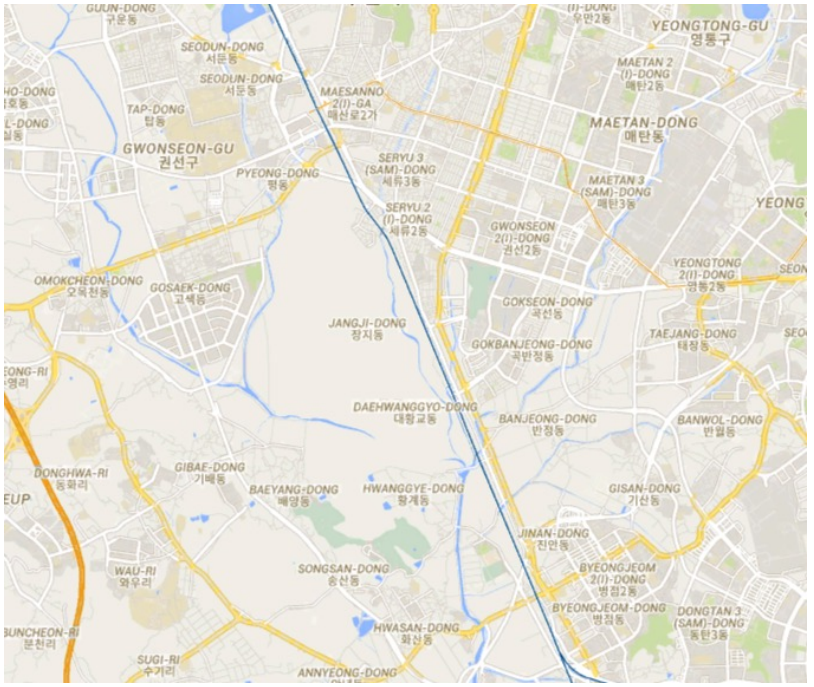


- **View level**
  - Only a portion of the database is visible to a user.
    - Eg. salary of employees is not exposed

Tool for DB Abstraction: Data Model

- Models are useful when examining or managing parts of the real world.

Real World (Satellite Image)



Model (Map)

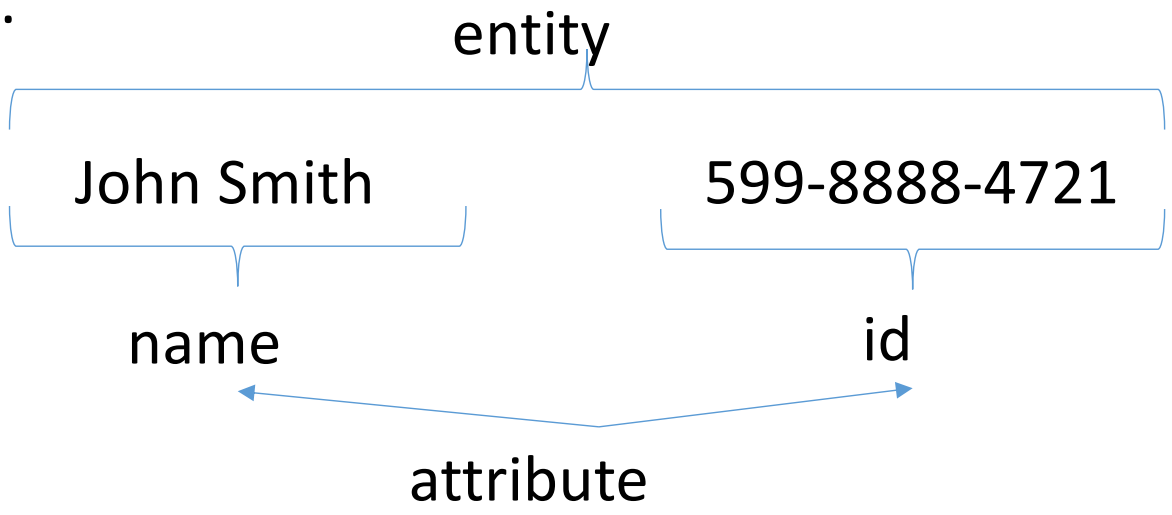
A model simplifies and hides unnecessary details

Data Models

- Data Model
  - high-level specification how a data is structured and used
- Collection of tools for describing
  - Data
  - Data relationships
  - Data semantics
  - Data constraints
- **Relational model** (Entity-Relationship data model)
- Object-based data models
- Semi-structured data model (XML)
- Less popular old models:
  - Network model
  - Hierarchical model

Quick view of Relational Model

- Relationship model
  - Real world consists of objects(entities)
  - Real world consists of relationships among the objects
  - Definition: **Entity**
    - Is an object that is distinguishable from other objects by a specific set of attributes
    - eg.

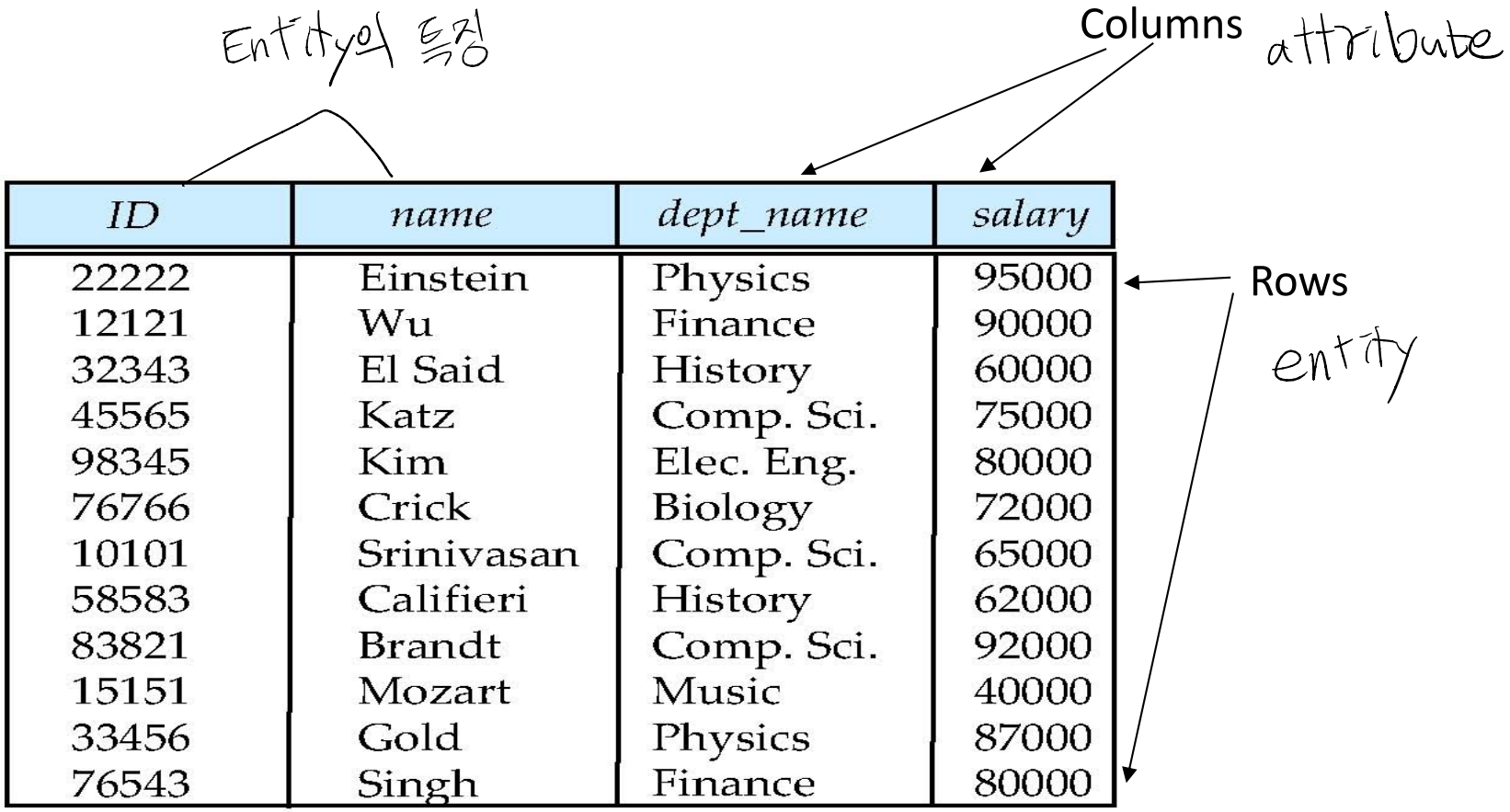


Quick view of Relational Model

- Entity-Relationship model (Cont'd)
  - Definition: **Relationship**
    - is an association among multiple entities
    - eg. CustAcct – associates a customer with each account they have
  - Def: **Entity Set** – set of all entities of the same type
  - Def: **Relationship Set** – set of all relationships of the same type
  - Def: **Mapping Cardinalities** – the number of entities to which an entity can be associated via a relationship set

Quick view of Relational Model

- Example of tabular data (Entity-Set) in the relational model



ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

A Sample Relational Database

entity set

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

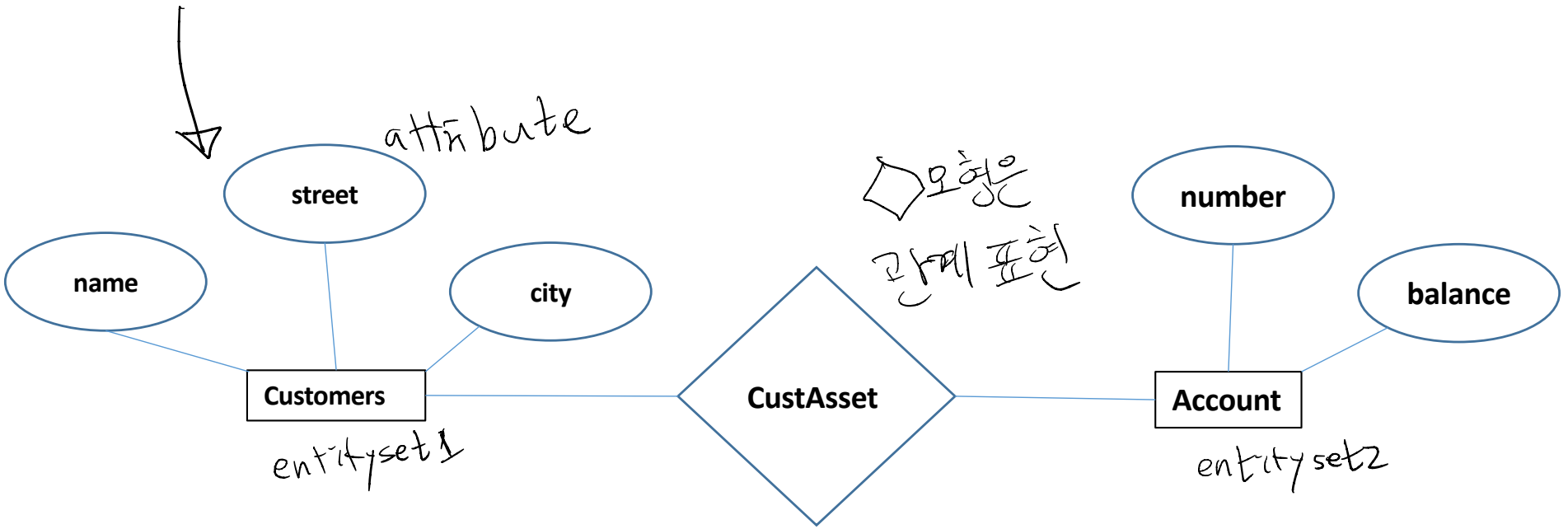
dept_name이라는 같은 열이 있기에 다른 set에서 연결해서 (Join) 생각할 수 있다.

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

Entity-Relationship Model (ER-Diagram)

- Graphical Representation of database



- Rectangles – represent entity sets
- Ellipses – represent entity attributes
- Diamonds – represent relationships among entity set
- Lines – link attributes to entity sets & entity sets to relationships

Schema and Instance

- **Schema**
  - Similar to types and variables in programming languages
  - Logical Schema – the overall logical structure of the database
  - Physical schema – the overall physical structure of the database (E.g., B+tree table, Hash table, etc)

- Instance – the actual content of the database at a particular time
  - Analogous to the value of a variable

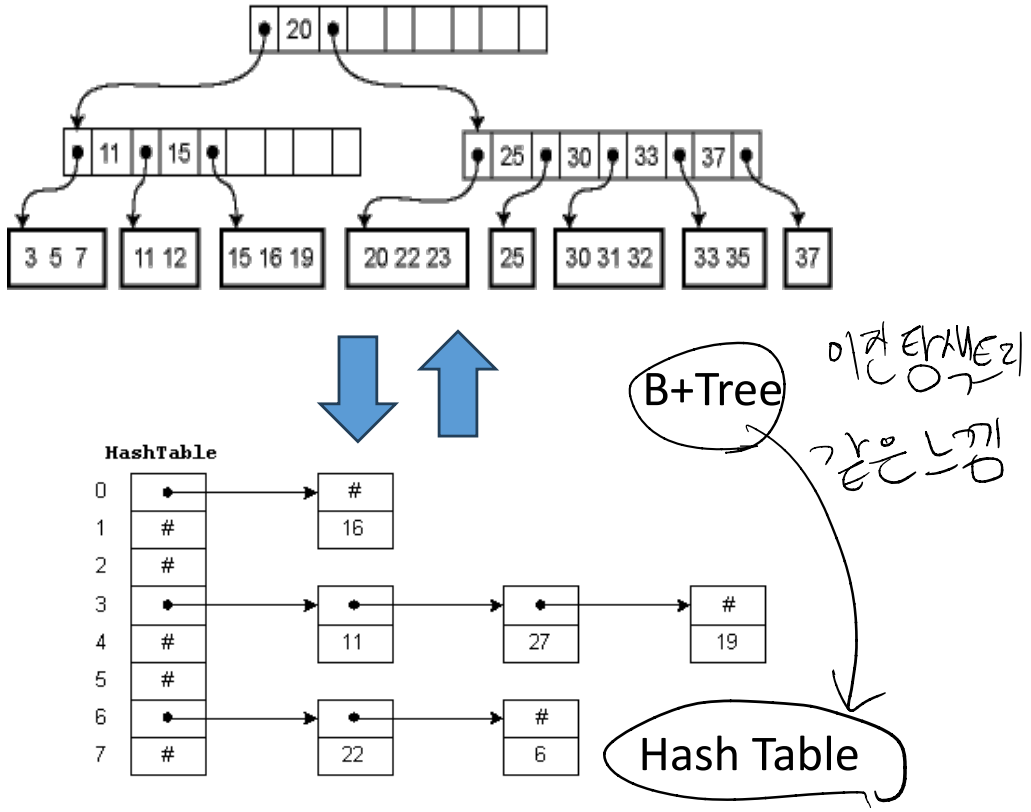
Physical Data Independence

- Physical Data Independence – the ability to modify the physical schema without changing the logical schema
  - Applications depend on the logical schema

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

Logical Schema



Physical Schema

SQL

- **SQL**: Structured Query Language (Chapter 3-5)
  - pronounced as /ˈsiːkwəl/ or /ˈɛskjuːˈɛl/
  - It's often pronounced as /ˈsiːkwəl/ due to IBM's SEQUEL (Structured English Query Language)
  - consists of
    - **DML** (Data Manipulation Language) and
    - **DDL** (Data Definition Language).

- Two types of programming languages
  - **Procedural** – users specify how to get data
    - e.g.) C, C++, Java, Python, …
  - **Declarative** – users do NOT specify how to get data
    - e.g.) **SQL**, Prologue, …

Data Definition Language (DDL)

- Used to define database schema
  - Example:

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

- Table definition are stored in ***data dictionary*** (also called ***metadata catalog***) in DBMS.

Data Manipulation Language (DML)

- Used to access and update the data
  - Example: find all instructors in Comp. Sci. department

search operation

```
select name
from instructor
where dept_name = 'Comp. Sci.'
```

← condition

  - DML also known as ***query language***
  - DML is usually embedded in some higher-level language

Java, Python

Normalization Theory

- Is there any problem with this design?

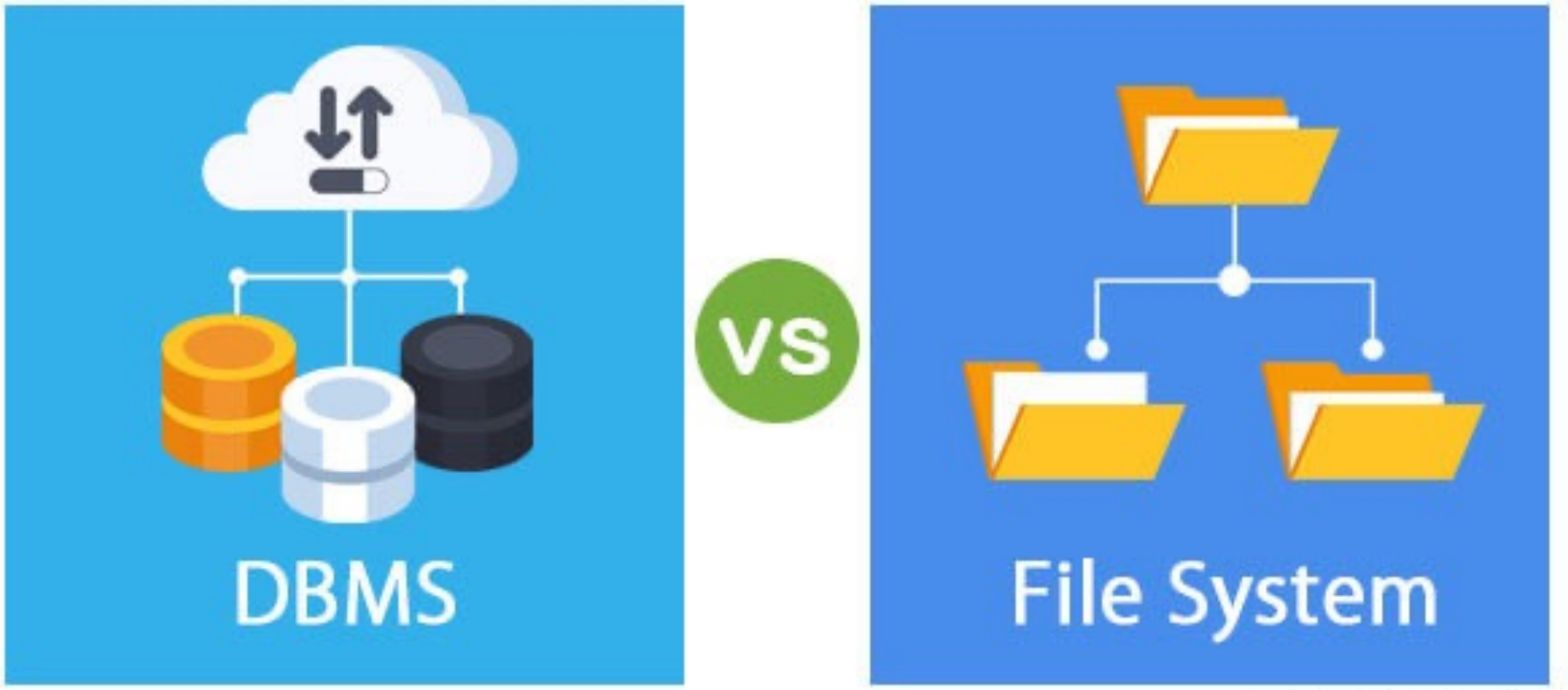
ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

- Normalization Theory
  - Formalize what designs are bad, and test for them

Part III: Storage Management

- System software designed to manage databases

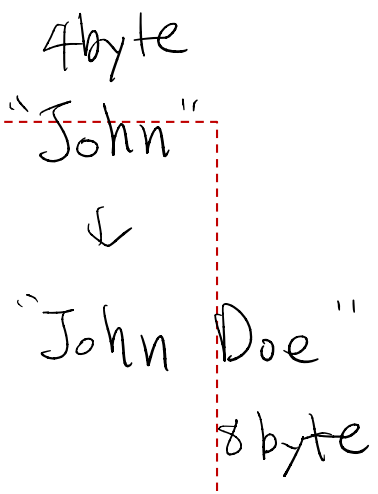
- Q: Why not use a simple file system to manage DB?



Why use DBMS rather than a file system?

- To add a student data to a file

```
def add_student(name, student_id, major):
    with open(FILE_PATH, "ab") as f:
        packed_data = struct.pack(
            STUDENT_STRUCT_FORMAT,
            name.encode('utf-8')[:20].ljust(20, b'\x00'),
            student_id.encode('utf-8')[:10].ljust(10, b'\x00'),
            major.encode('utf-8')[:20].ljust(20, b'\x00')
        )
        f.write(packed_data)
```



- Q: How can you modify the code to find and update a specific student data in the file?

Why use DBMS rather than a file system?

- Q: Suppose you have stored millions of records in a file, but then your boss asks you to manage phone numbers as well. How would you update the file and handle this request?

- Q: Your boss complains that your application runs too slow. How would you handle that?

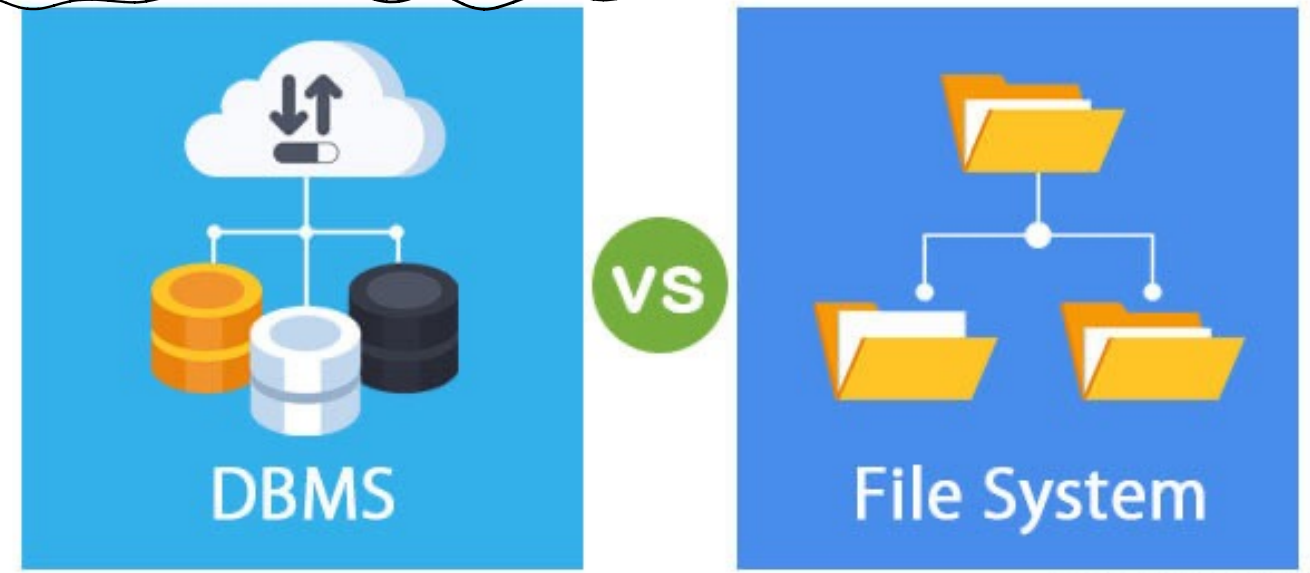
- Q: Suppose your application supports multiple concurrent queries. How would you prevent conflicts when accessing the file simultaneously?

Why use DBMS rather than a file system?

- Reason for using a DBMS rather than a file system
  - Uniform Access & Control of the data
    - No proprietary file formats needed
  - Control of redundancy
    - E.g. Don't need to store student names many times
  - Integrity
    - E.g. No salary can be below $5000.
  - Data Independence
    - Storage structure has no effect on the functionality
  - Concurrency control
  - Recovery guaranteed
  - Security
  - Consistency
  - Many more!!

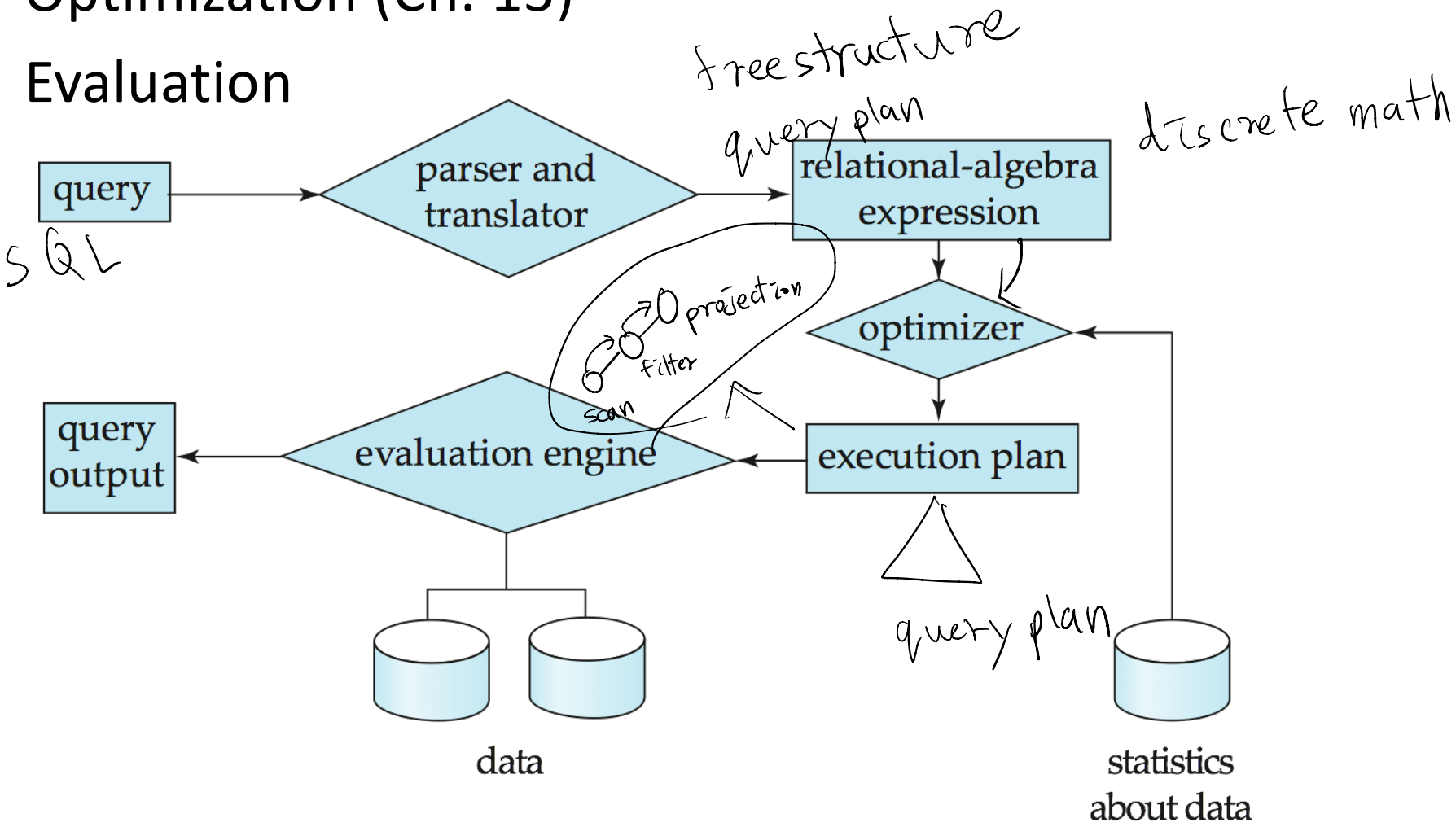
Why use DBMS rather than a file system?

	DBMS (Abstraction & Automation)	File System (Flexibility & Control)
Purpose	Structured data management	General file storage
Data Access	SQL queries, indexing, transactions	Direct file operations (read/write)
Flexibility	Fixed schema, query optimization	Any file format, user-defined structure
Concurrency & Integrity	ACID compliance, automatic handling	Manual concurrency control needed
Performance	Optimized for complex queries	Faster for simple I/O operations



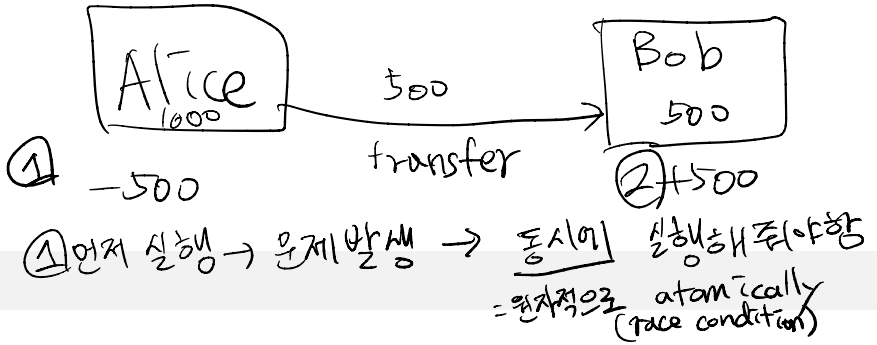
Part IV: Query Processing

1. Parsing and translation (Ch. 12)
2. Optimization (Ch. 13)
3. Evaluation

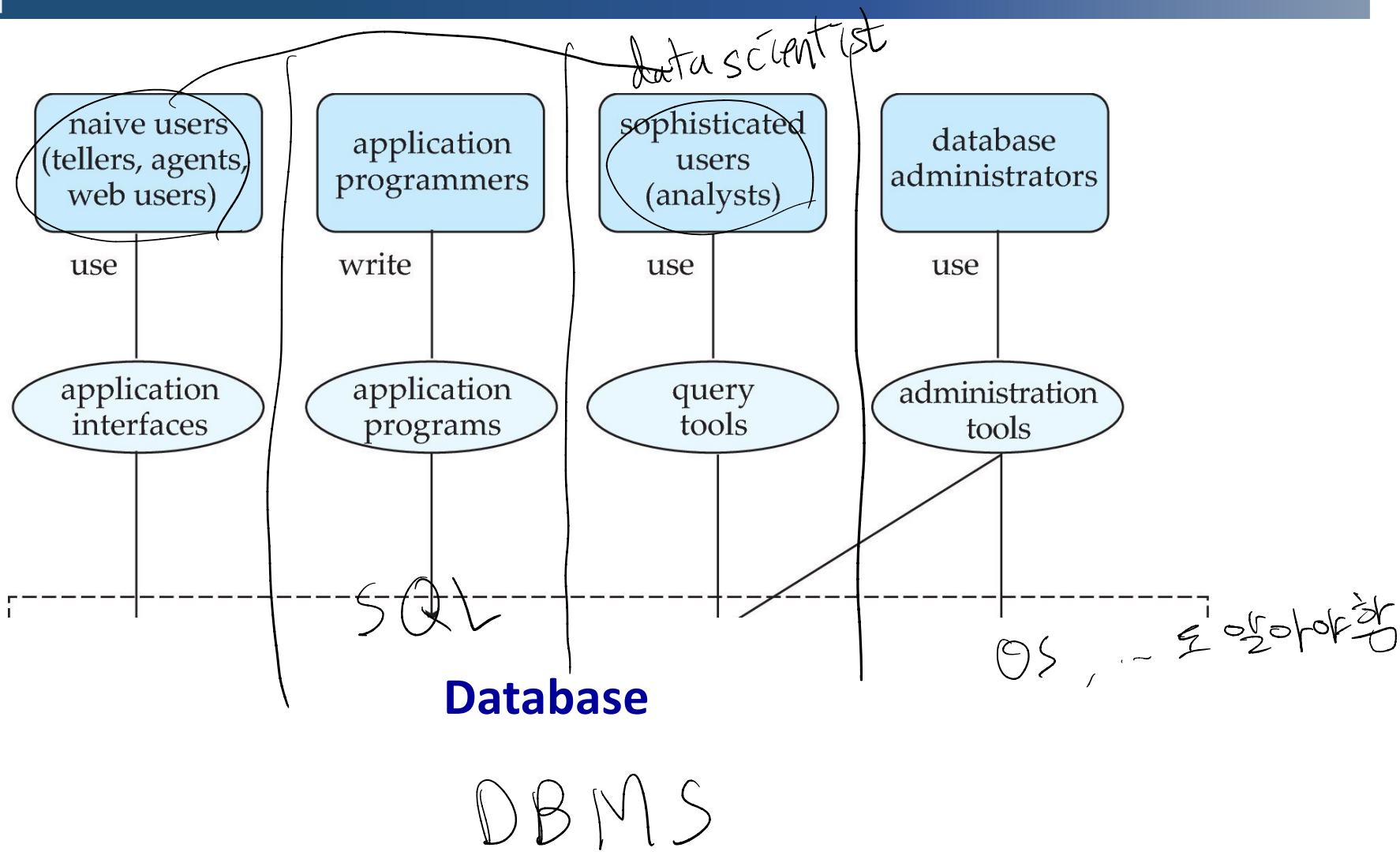


Transaction Management

- What if the system fails?
- What if more than one user is concurrently updating the same data?
- A **transaction** is a collection of operations that performs a single logical function in a database application
- **Transaction manager** ensures that the database remains in a consistent state
  - Recovery scheme (Ch. 16)
- **Concurrency-control manager** controls the interaction among the concurrent transactions
  - Lock-based protocols (Ch. 15)
    - version protocol
    - Timestamp protocol



Appendix: DB Users and Administrators



Appendix: Database System Internals