

Introduction

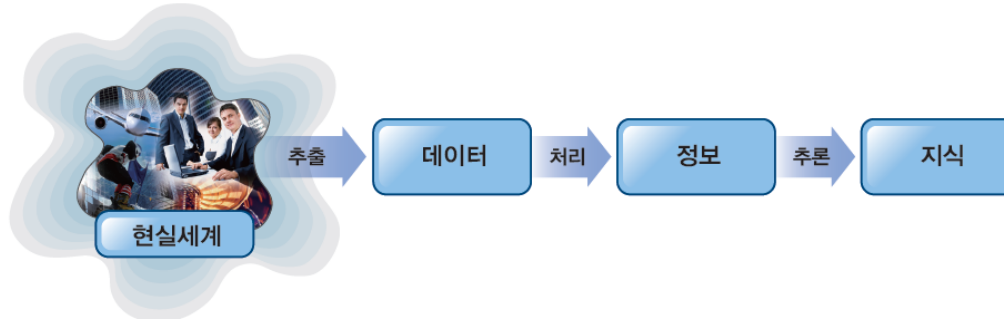
Outline



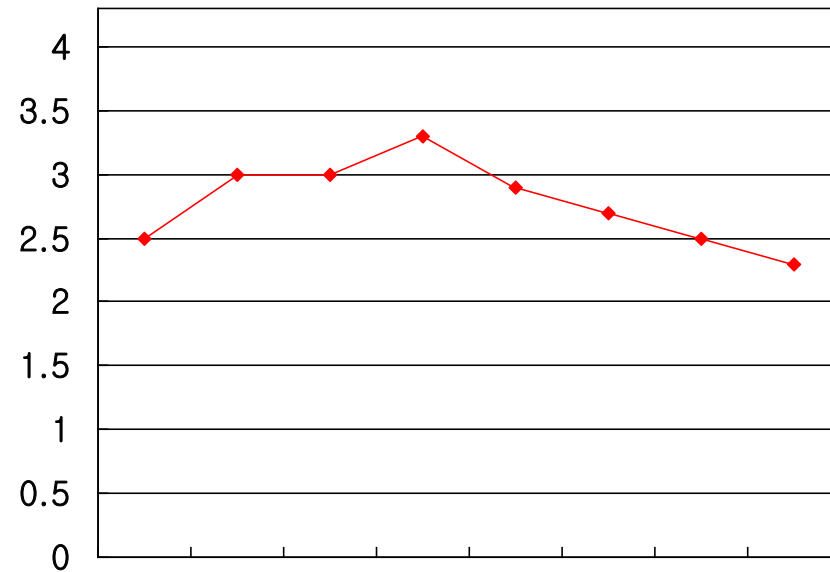
- ☐ The Need for Databases
- ☐ Data Models
- ☐ Relational Databases
- ☐ Database Design
- ☐ Storage Manager
- ☐ Query Processing
- ☐ Transaction Manager

Data

□ Data vs. Information



| | |
|-----|-----|
| 1학기 | 2.5 |
| 2학기 | 3.0 |
| 3학기 | 3.0 |
| 4학기 | 3.3 |
| 5학기 | 2.9 |
| 6학기 | 2.7 |
| 7학기 | 2.5 |
| 8학기 | 2.3 |



Data and Database

- ❑ Data is an entity, event, phenomena or idea that is worth recording.
- ❑ Database is an organized collection of data, generally stored and accessed electronically from a computer system.
 - Formally, a “database” refers to a set of related data and the way it is organized.
 - Conforming to the principle of a particular database model

Database

From Wikipedia, the free encyclopedia

For a topical guide to this subject, see [Outline of databases](#).

A **database** is an organized collection of **data**, generally stored and accessed electronically from a computer system. Where databases are more complex they are often developed using formal [design and modeling](#) techniques.

The [database management system](#) (DBMS) is the [software](#) that interacts with [end users](#), applications, and the database itself to capture and analyze the data. The DBMS software additionally encompasses the core facilities provided to administer the database. The sum total of the database, the DBMS and the associated applications can be referred to as a "database system". Often the term "database" is also used to loosely refer to any of the DBMS, the database system or an application associated with the database.

Computer scientists may classify database-management systems according to the [database models](#) that they support. [Relational databases](#) became dominant in the 1980s. These model data as [rows](#) and [columns](#) in a series of [tables](#), and the vast majority use [SQL](#) for writing and querying data. In the 2000s, non-relational databases became popular, referred to as [NoSQL](#) because they use different [query languages](#).

```
sqlite> select title, release_year, length, replacement_cost from film
sqlite> where length > 110 and replacement_cost > 20.00
sqlite> order by title desc
```

| title | release_year | length | replacement_cost |
|------------------------|--------------|--------|------------------|
| West Side Story | 2002 | 105 | 20.99 |
| Virgin Suicides | 2000 | 129 | 20.99 |
| House of M | 2006 | 112 | 20.99 |
| Tracy Flicker | 2000 | 142 | 20.99 |
| Temp Men | 2000 | 105 | 20.99 |
| Slender Lilies | 2000 | 129 | 20.99 |
| Rocky Horror | 2000 | 104 | 20.99 |
| Wine of the South | 2000 | 140 | 20.99 |
| Eight Women | 2000 | 112 | 20.99 |
| Guest House | 2000 | 117 | 20.99 |
| House of the Future | 2000 | 100 | 20.99 |
| Laughing Legally | 2000 | 140 | 20.99 |
| Laurel & Hardy | 2000 | 112 | 20.99 |
| Single Supper | 2000 | 124 | 20.99 |
| One of the | 2000 | 112 | 20.99 |
| Japanese War | 2000 | 105 | 20.99 |
| Officer | 2000 | 112 | 20.99 |
| Plants Garden | 2000 | 140 | 20.99 |
| Extraordinary Computer | 2000 | 122 | 20.99 |
| Expensive Craft | 2000 | 112 | 20.99 |
| Dirty Red | 2000 | 147 | 20.99 |
| Cyber Therapy | 2000 | 100 | 20.99 |
| Clockwork Paradise | 2000 | 141 | 20.99 |
| Baroque Wedding | 2000 | 112 | 20.99 |

An SQL select statement and its results

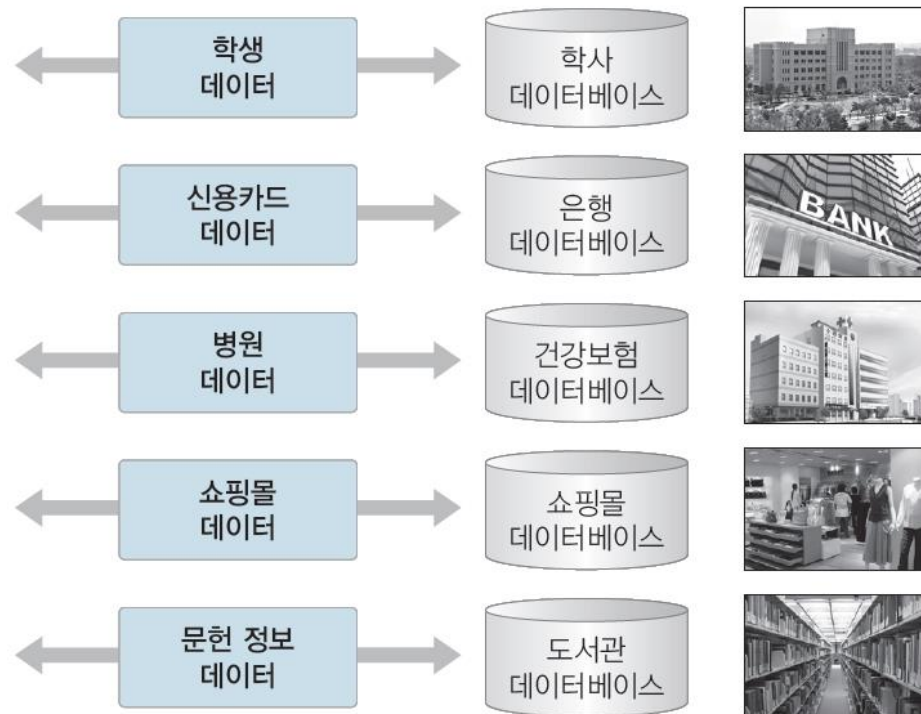
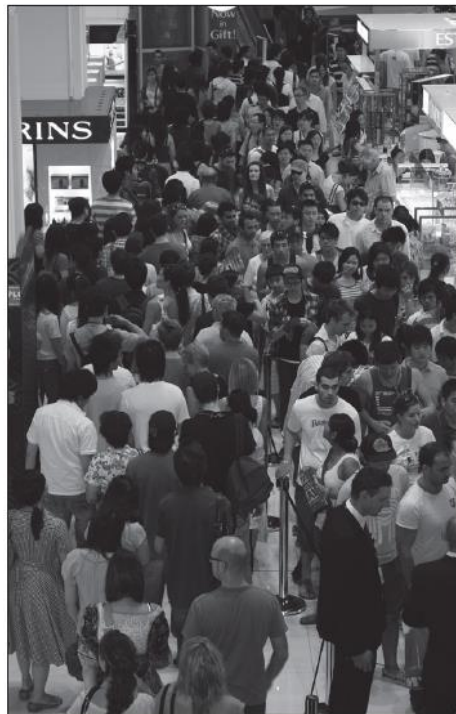
Database Management System (DBMS)



- ❑ DBMS contains information about a particular enterprise
 - Collection of interrelated data
 - Set of programs to access the data
 - An environment that is both *convenient* and *efficient* to use
- ❑ Database Applications:
 - Banking: transactions
 - Airlines: reservations, schedules
 - Universities: registration, grades
 - Sales: customers, products, purchases
 - Online retailers: order tracking, customized recommendations
 - Manufacturing: production, inventory, orders, supply chain
 - Human resources: employee records, salaries, tax deductions
- ❑ Databases can be very large.
- ❑ Databases touch all aspects of our lives

Database Examples in our lives

- ❑ Database is integrated, stored, operational, shared data
- ❑ Database can be accessed at real time, continuously and concurrently



University Database Example



- ❑ Application program examples
 - Add new students, instructors, and courses
 - Register students for courses, and generate class rosters
 - Assign grades to students, compute grade point averages (GPA) and generate transcripts
- ❑ In the early days, database applications were built directly on top of file systems

Drawbacks of using file systems to store data



- ❑ Data redundancy and inconsistency
 - Multiple file formats, duplication of information in different files
- ❑ Difficulty in accessing data
 - Need to write a new program to carry out each new task
- ❑ Data isolation
 - Multiple files and formats
- ❑ Integrity problems
 - Integrity constraints (e.g., $\text{account balance} > 0$) become “buried” in program code rather than being stated explicitly
 - Hard to add new constraints or change existing ones

Drawbacks of using file systems to store data (Cont.)



☐ Atomicity of updates

- Failures may leave database in an inconsistent state with partial updates carried out
- Example: Transfer of funds from one account to another should either complete or not happen at all

☐ Concurrent access by multiple users

- Concurrent access needed for performance
- Uncontrolled concurrent accesses can lead to inconsistencies
 - Example: Two people reading a balance (say 100) and updating it by withdrawing money (say 50 each) at the same time

☐ Security problems

- Hard to provide user access to some, but not all, data

Database systems offer solutions to all the above problems

File systems vs. DBMS

```
/* BOOK 데이터 구조 정의 */
typedef struct
{
    int bookid[5];
    char bookname[20];
    char publisher[20];
    int price;
} BOOK;

int main() {
    BOOK BOOKS[10];
    /* 구조체 배열 변수에 데이터 저장 */
    /* 첫 번째 도서 저장 */
    BOOKS[1].bookid=1;
    strcpy(BOOKS[1].bookname, "축구의 역사");
    strcpy(BOOKS[1].publisher, "굿스포츠");
    BOOKS[1].price=7000;

    /* 두 번째 도서 저장 */
    BOOKS[2].bookid=2;
    strcpy(BOOKS[2].bookname, "축구 아는 여자");
    strcpy(BOOKS[2].publisher, "나무수");
    BOOKS[2].price=13000;
    /* 나머지 다른 도서 저장(생략) */
    .....
    /* 모든 도서보기 프로그램 호출 */
    search_all();
    .....
}
```

```
/* BOOK 데이터 구조 정의 */
typedef struct
{
    int bookid[5];
    char bookname[20];
    char publisher[20];
    int price;
} BOOK;

int main( )
{
    BOOK BOOKS[10];
    int i=1;
    /* 도서 입력 함수 */
    insert( );
    /* 파일에 저장된 데이터를 배열 BOOKS[ ]에 저장 */
    fp=fopen("book.dat", "rb");
    bp=(BOOK *)calloc(1,sizeof(BOOK));
    /* 파일에서 책을 읽는다 */
    while(fread(bp,sizeof(BOOK),1,fp) != 0)
    {
        BOOKS[i].bookid =bp->bookid;
        strcpy(BOOKS[i].bookname, bp ->bookname);
        strcpy(BOOKS[i].publisher, bp ->publisher);
        BOOKS[i].price =bp ->price;
        i++;
    }
    /* 모든 도서보기 프로그램 호출 */
    search_all( );
    .....
}
```

File systems vs. DBMS

```
/* BOOK 데이터 구조 정의 */
typedef struct
{
    int bookid[5];
    char bookname[20];
    char publisher[20];
    int price;
} BOOK;

int main() {
    BOOK BOOKS[10];
    /* 구조체 배열 변수에 데이터 저장 */
    /* 첫 번째 도서 저장 */
    BOOKS[1].bookid=1;
    strcpy(BOOKS[1].bookname, "축구의 역사");
    strcpy(BOOKS[1].publisher, "굿스포츠");
    BOOKS[1].price=7000;

    /* 두 번째 도서 저장 */
    BOOKS[2].bookid=2;
    strcpy(BOOKS[2].bookname, "축구 아는 여자");
    strcpy(BOOKS[2].publisher, "나무수");
    BOOKS[2].price=13000;
    /* 나머지 다른 도서 저장(생략) */
    .....
    /* 모든 도서보기 프로그램 호출 */
    search_all();
    .....
}
```

```
int main( )
{
    /* 반환된 행의 수 */
    int num_ret;

    /* DBMS에 접속 */
    EXEC SQL CONNECT :username IDENTIFIED BY :password;

    /* SQL 문 실행 */
    EXEC SQL DECLARE c1 CURSOR FOR
        SELECT bookname, publisher, price FROM BOOK;
    EXEC SQL OPEN c1;

    /* 모든 도서보기 프로그램 호출 */
    search_all( );

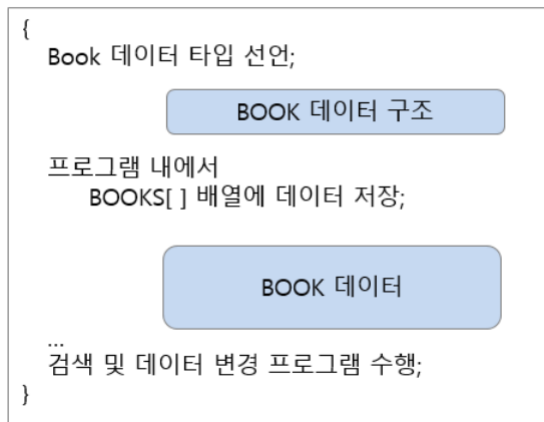
    /* SQL 문 실행 결과 출력 */
    for (;;) {
        EXEC SQL FETCH c1 INTO :BOOK_rec;
        print_rows(num_ret);
    }
    EXEC SQL CLOSE c1;

    /* 접속 해제 */
    EXEC SQL COMMIT WORK RELEASE;

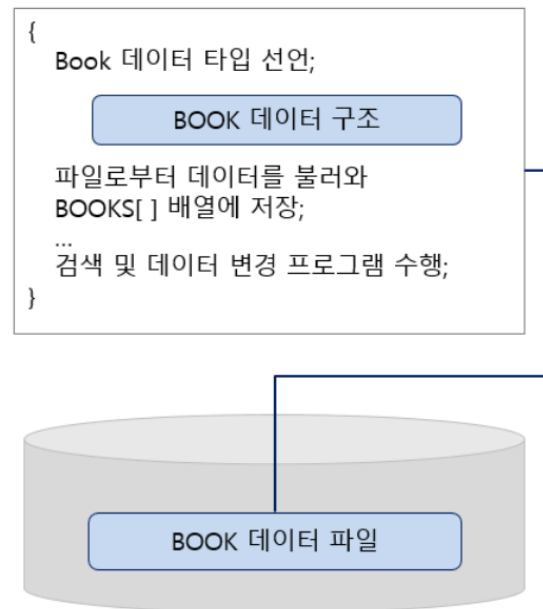
}
```

File systems vs. DBMS

[프로그램 1]



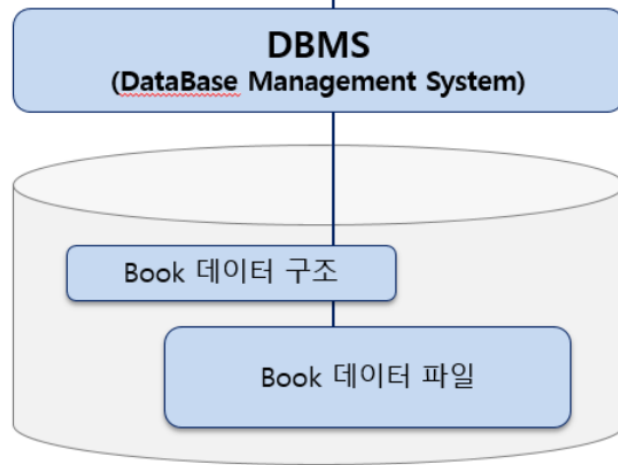
[프로그램 2]



File systems vs. DBMS

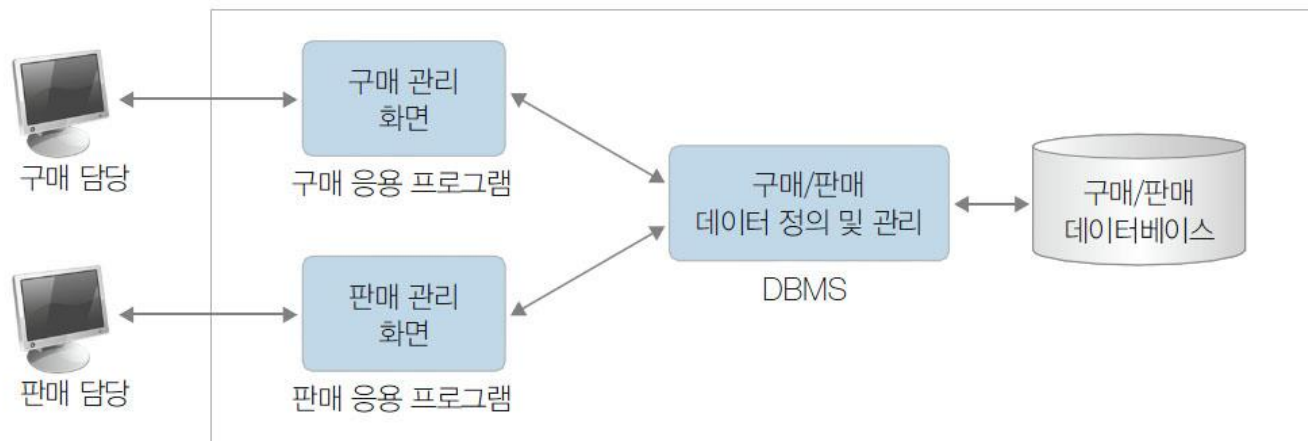
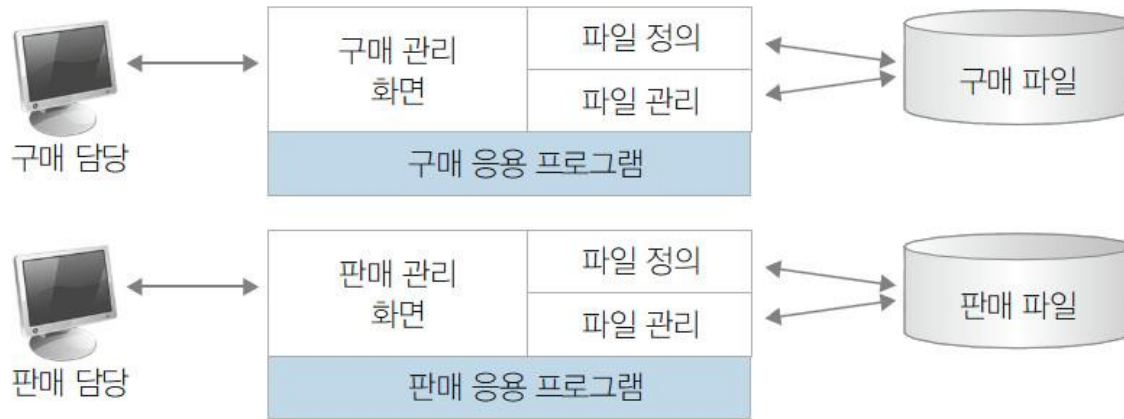
[프로그램 3]

```
{  
  /* BOOK 데이터 타입 선언 필요 없음 */  
  
  SQL 문을 실행하여 결과를 가져옴;  
  ...  
  SQL 문으로 데이터 변경 ;  
}
```



- DBMS가 데이터 정의와 데이터 값을 관리하는 방식
- BOOK 데이터 구조는 DBMS가 관리하고, 데이터 값은 데이터베이스에 저장됨
- 데이터 값이 바뀌거나 데이터 값이 바뀌어도 프로그램을 다시 컴파일할 필요 없음

File systems vs. DBMS



Levels of Abstraction



- ❑ **Physical level:** describes how a record (e.g., instructor) is stored.
- ❑ **Logical level:** describes data stored in database, and the relationships among the data.

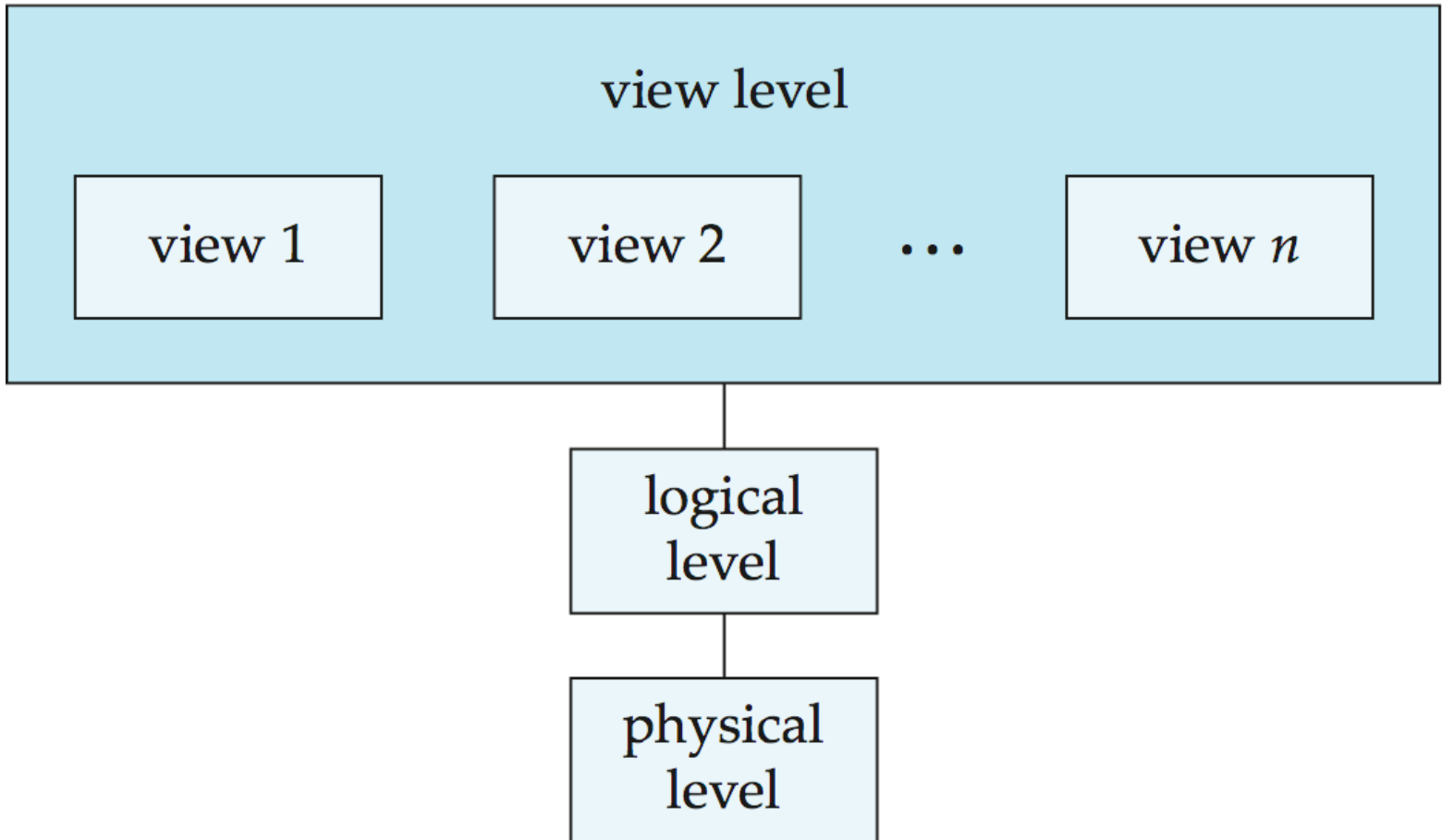
type *instructor* = **record**

```
ID : string;  
name : string;  
dept_name : string;  
salary : integer;  
end;
```

- ❑ **View level:** application programs hide details of data types. Views can also hide information (such as an employee's salary) for security purposes.

View of Data (Three layers of schema architecture)

- ❑ An architecture for a database system



Instances and Schemas



- ❑ Similar to types and variables in programming languages
- ❑ **Logical Schema** – the overall logical structure of the database
 - Example: The database consists of information about a set of customers and accounts in a bank and the relationship between them
 - ▶ Analogous to type information of a variable in a program
- ❑ **Physical schema** – the overall physical structure of the database
- ❑ **Instance** – the actual content of the database at a particular point in time
 - Analogous to the value of a variable
- ❑ **Physical Data Independence** – the ability to modify the physical schema without changing the logical schema
 - Applications depend on the logical schema
 - In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others.

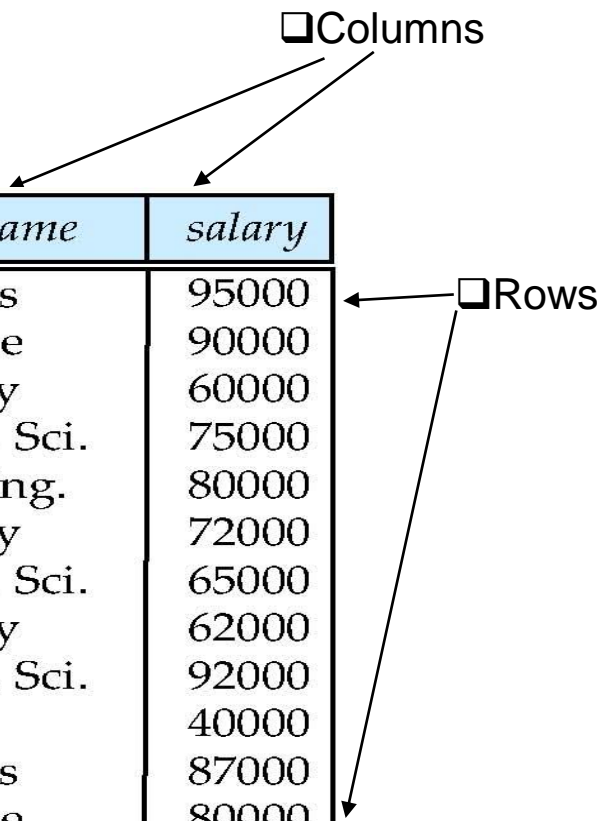
Data Models



- ☐ Providing data abstraction
- ☐ A collection of tools for describing
 - Data
 - Data relationships
 - Data semantics
 - Data constraints
- ☐ Relational model
- ☐ Entity-Relationship data model (mainly for database design)
- ☐ Object-based data models (Object-oriented and Object-relational)
- ☐ Semistructured data model (XML)
- ☐ Other older models:
 - Network model
 - Hierarchical model

Relational Model


- ❑ All the data is stored in various tables.
- ❑ Example of tabular data in the relational model



| <i>ID</i> | <i>name</i> | <i>dept_name</i> | <i>salary</i> |
|-----------|-------------|------------------|---------------|
| 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



| <i>ID</i> | <i>name</i> | <i>dept_name</i> | <i>salary</i> |
|-----------|-------------|------------------|---------------|
| 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

| <i>dept_name</i> | <i>building</i> | <i>budget</i> |
|------------------|-----------------|---------------|
| 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

Data Definition Language (DDL)

- ❑ Specification notation for defining the database schema

Example: **create table** *instructor* (
 ID **char**(5),
 name **varchar**(20),
 dept_name **varchar**(20),
 salary **numeric**(8,2))

- ❑ DDL compiler generates a set of table templates stored in a *data dictionary*
- ❑ Data dictionary contains metadata (i.e., data about data)
 - Database schema
 - Integrity constraints
 - Primary key (ID uniquely identifies instructors)
 - Authorization
 - Who can access what

Data Manipulation Language (DML)



- ❑ Language for accessing and manipulating the data organized by the appropriate data model
 - DML also known as query language
- ❑ Two classes of languages
 - **Pure** – used for proving properties about computational power and for optimization
 - Relational Algebra
 - Tuple relational calculus
 - Domain relational calculus
 - **Commercial** – used in commercial systems
 - SQL is the most widely used commercial language

SQL



- ❑ The most widely used commercial language
- ❑ SQL is NOT a Turing machine equivalent language
- ❑ To be able to compute complex functions SQL is usually embedded in some higher-level language
- ❑ Application programs generally access databases through one of
 - Language extensions to allow embedded SQL
 - Application program interface (e.g., ODBC/JDBC) which allow SQL queries to be sent to a database

Database Design



The process of designing the general structure of the database:

- ❑ Logical Design – Deciding on the database schema. Database design requires that we find a “good” collection of relation schemas.
 - Business decision – What attributes should we record in the database?
 - Computer Science decision – What relation schemas should we have and how should the attributes be distributed among the various relation schemas?
- ❑ Physical Design – Deciding on the physical layout of the database

Database Design (Cont.)

❑ Is there any problem with this relation?

| <i>ID</i> | <i>name</i> | <i>salary</i> | <i>dept_name</i> | <i>building</i> | <i>budget</i> |
|-----------|-------------|---------------|------------------|-----------------|---------------|
| 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 |

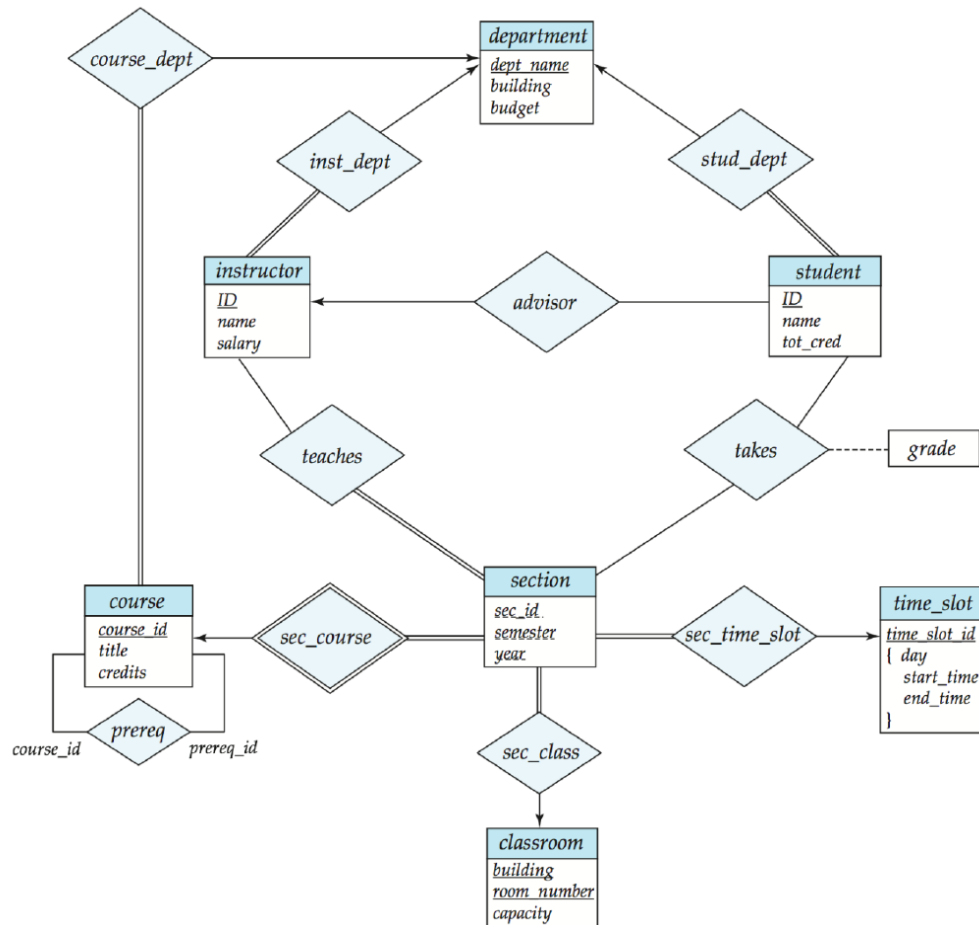
Design Approaches



- ❑ Need to come up with a methodology to ensure that each of the relations in the database is “good”
- ❑ Two ways of doing so:
 - Entity Relationship Model (Chapter 7)
 - Models an enterprise as a collection of *entities* and *relationships*
 - Represented diagrammatically by an *entity-relationship diagram*:
 - Normalization Theory (Chapter 8)
 - Formalize what designs are bad, and test for them

Entity-Relationship Model

- Database design in E-R model usually converted to design in the relational model which is used for storage and processing



Object-Relational Data Models



- ☐ Relational model: flat, “atomic” values
- ☐ Object Relational Data Models
 - Extend the relational data model by including object orientation and constructs to deal with added data types.
 - Allow attributes of tuples to have complex types, including non-atomic values such as nested relations.
 - Preserve relational foundations, in particular the declarative access to data, while extending modeling power.
 - Provide upward compatibility with existing relational languages.
- ☐ Not covered in this semester

XML: Extensible Markup Language



- ☐ Defined by the WWW Consortium (W3C)
 - ☐ Originally intended as a document markup language not a database language
 - ☐ The ability to specify new tags, and to create nested tag structures made XML a great way to exchange **data**, not just documents
 - ☐ XML has become the basis for all new generation data interchange formats.
 - ☐ A wide variety of tools is available for parsing, browsing and querying XML documents/data
-
- ☐ Not covered in this semester

Database Engine



- ☐ Storage manager
- ☐ Query processing
- ☐ Transaction manager

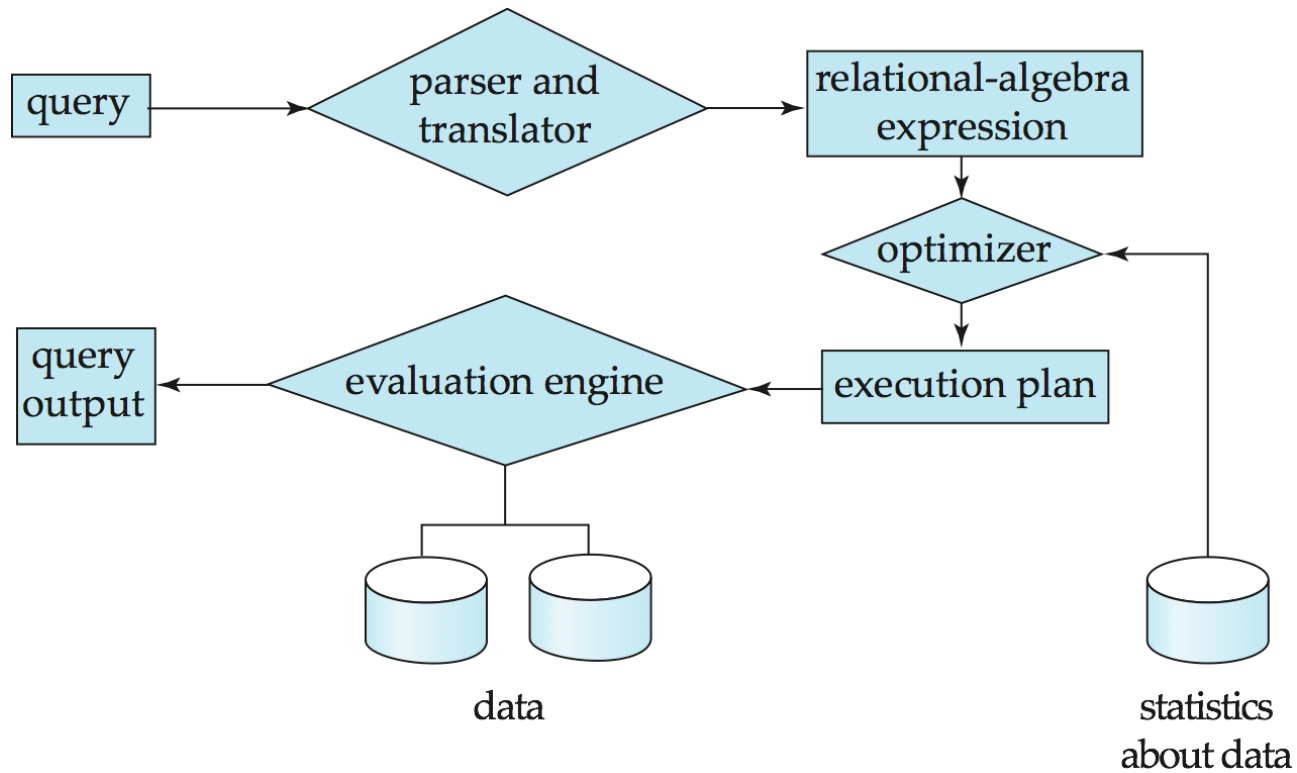
Storage Management



- ❑ **Storage manager** is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
- ❑ The storage manager is responsible to the following tasks:
 - Interaction with the OS file manager
 - Efficient storing, retrieving and updating of data
- ❑ Issues:
 - ~~Storage access~~
 - ~~File organization~~
 - Indexing and hashing

Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation



Query Processing (Cont.)



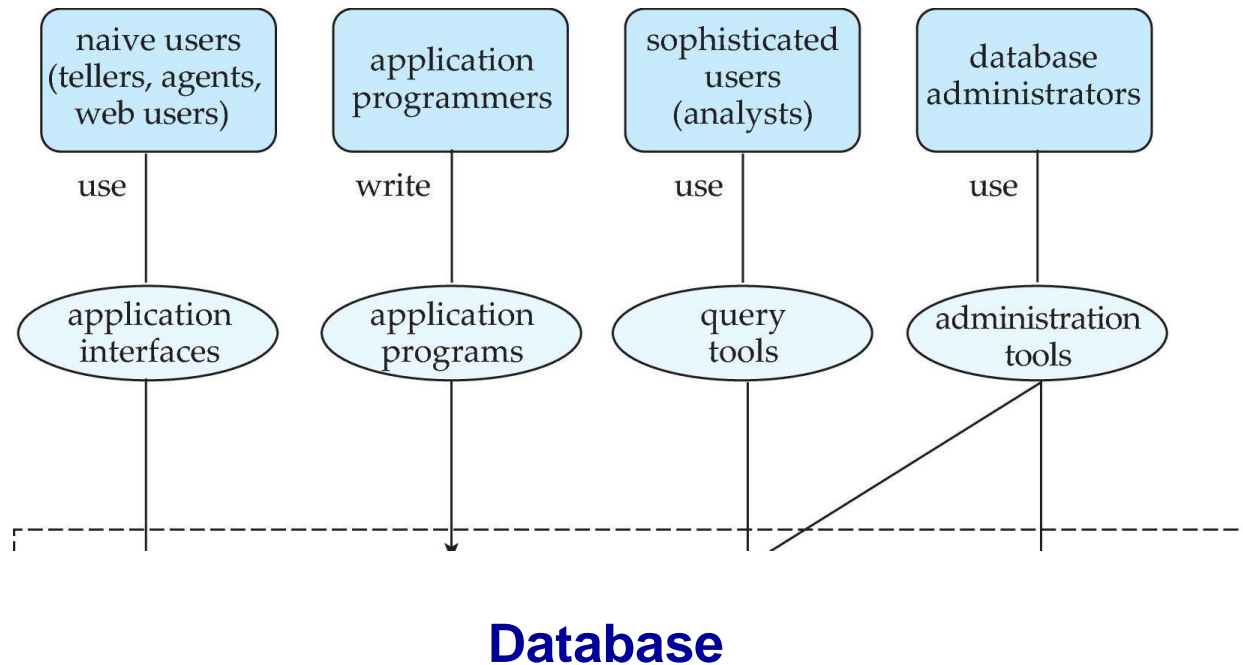
- ❑ Alternative ways of evaluating a given query
 - Equivalent expressions
 - Different algorithms for each operation
- ❑ Cost difference between a good and a bad way of evaluating a query can be enormous
- ❑ Need to estimate the cost of operations
 - Depends critically on statistical information about relations which the database must maintain
 - Need to estimate statistics for intermediate results to compute cost of complex expressions

Transaction Management

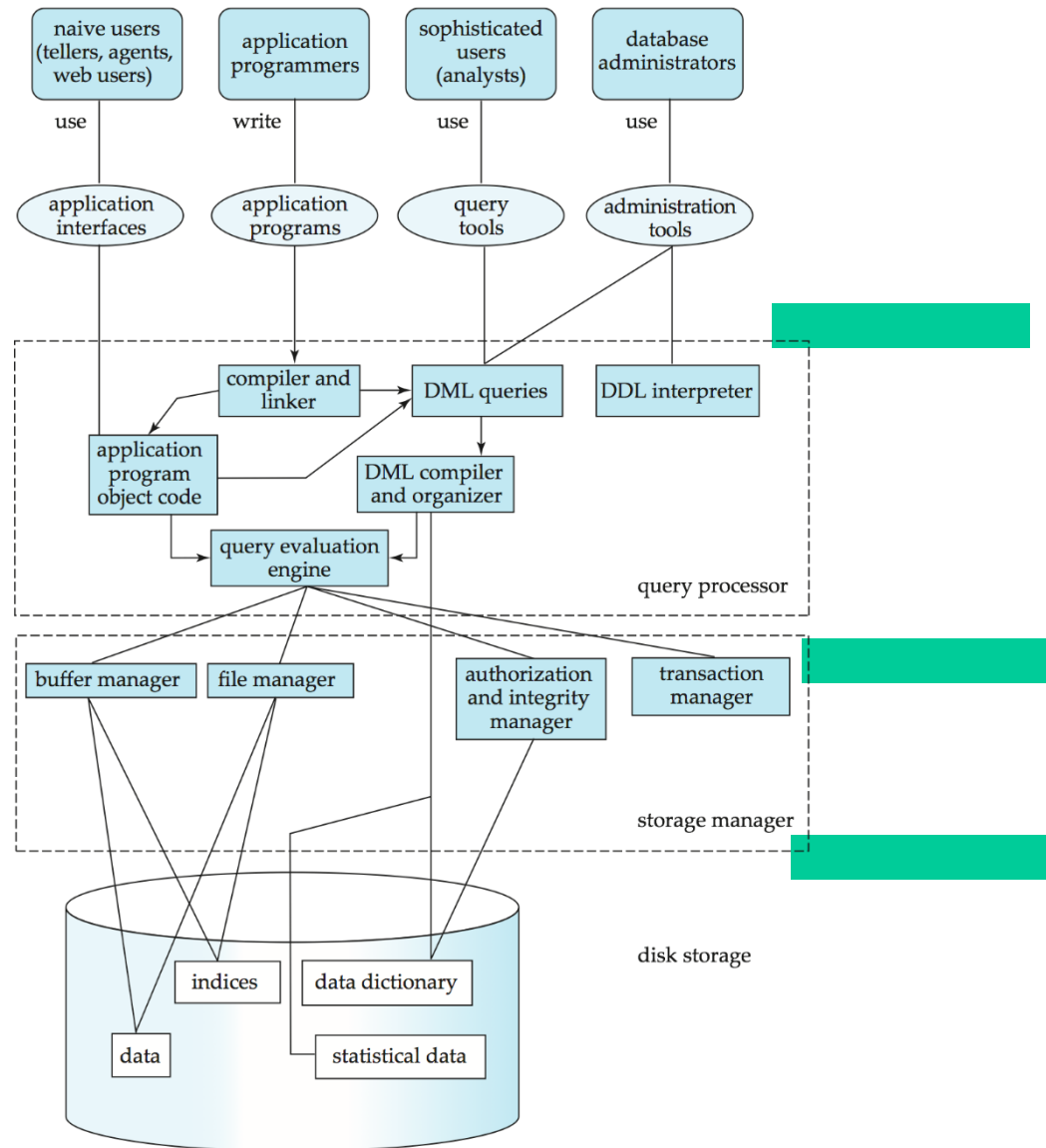


- ❑ What if the system fails?
- ❑ What if more than one user is concurrently updating the same data?
- ❑ A **transaction** is a collection of operations that performs a single logical function in a database application
- ❑ **Transaction-management component** ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.
- ❑ **Concurrency-control manager** controls the interaction among the concurrent transactions, to ensure the consistency of the database.

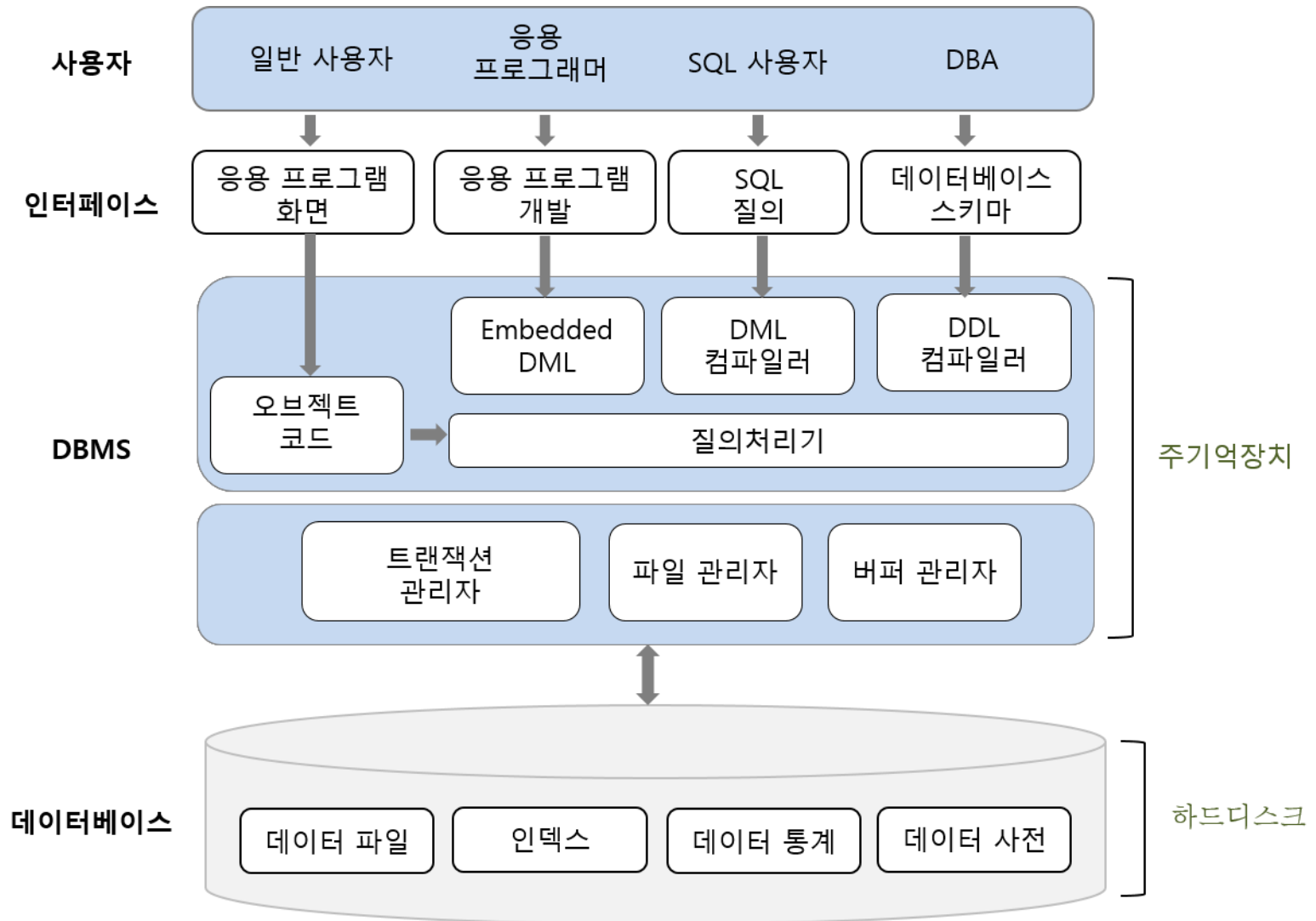
Database Users and Administrators



Database System Internals



Database System Internals



Database Architecture



The architecture of a database systems is greatly influenced by the underlying computer system on which the database is running:

- ☐ Centralized
- ☐ Client-server
- ☐ Parallel (multi-processor)
- ☐ Distributed

History of Database Systems



❑ 1950s and early 1960s:

- Data processing using magnetic tapes for storage
 - Tapes provided only sequential access
- Punched cards for input

❑ Late 1960s and 1970s:

- Hard disks allowed direct access to data
- Network and hierarchical data models in widespread use
- Ted Codd defines the relational data model
 - Would win the ACM Turing Award for this work
 - IBM Research begins System R prototype
 - UC Berkeley begins Ingres prototype
- High-performance (for the era) transaction processing

History (cont.)



❑ 1980s:

- Research relational prototypes evolve into commercial systems
 - SQL becomes industrial standard
- Parallel and distributed database systems
- Object-oriented database systems

❑ 1990s:

- Large decision support and data-mining applications
- Large multi-terabyte data warehouses
- Emergence of Web commerce

❑ Early 2000s:

- XML and XQuery standards
- Automated database administration

❑ Later 2000s:

- Giant data storage systems
 - Google BigTable, Yahoo PNuts, Amazon, ..

Summary



- ❑ Many slides from Database System Concepts, 6th Ed.
 - See www.db-book.com for conditions on re-use
- ❑ Some slides (figures in Korean) from 한빛아카데미 데이터베이스 개론

Reference



- ❑ Many slides from Database System Concepts, 6th Ed.
 - See www.db-book.com for conditions on re-use
- ❑ Some slides (figures in Korean) from 한빛아카데미 데이터베이스 개론