CS 4092 Database Design and Development (DDD)

01: Introduction

Seokki Lee

Slides are adapted from:

Database System Concepts, 6th & 7th Ed. ©Silberschatz, Korth and Sudarshan

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 systems are used to manage collections of data that are:
  - Highly valuable
  - Relatively large
  - Accessed by multiple users and applications, often at the same time.
- A modern database system is a complex software system whose task is to manage a large, complex collection of data.
- Databases touch all aspects of our lives

Database Applications Examples

- Enterprise Information
  - Sales: customers, products, purchases
  - Accounting: payments, receipts, assets
  - Human Resources: Information about employees, salaries, payroll taxes.
- Manufacturing: management of production, inventory, orders, and supply chain.
- Banking and Finance
  - Customer information, accounts, loans, and banking transactions.
  - Credit card transactions
  - Finance: sales and purchases of financial instruments (e.g., stocks and bonds; storing real-time market data
- Universities: registration, grades

Database Applications Examples

- Airlines: reservations, schedules
- Telecommunication: records of calls, texts, and data usage, generating monthly bills, maintaining balances on prepaid calling cards
- Web-based services
  - Online retailers: order tracking, customized recommendations
  - Online advertisements
- Document databases
- Navigation systems: For maintaining the locations of varies places of interest along with the exact routes of roads, train systems, buses, etc.

University Database Example

- What is the data managed?
- What tasks should be done over the data?

University Database Example

- Data consists of information about:
  - Students
  - Instructors
  - Classes
- 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

- Data redundancy and inconsistency
  - Multiple file formats, duplication of information in different files
  - Example: if a student has a double major (say, music and mathematics)
- 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., 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

- 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

View of Data

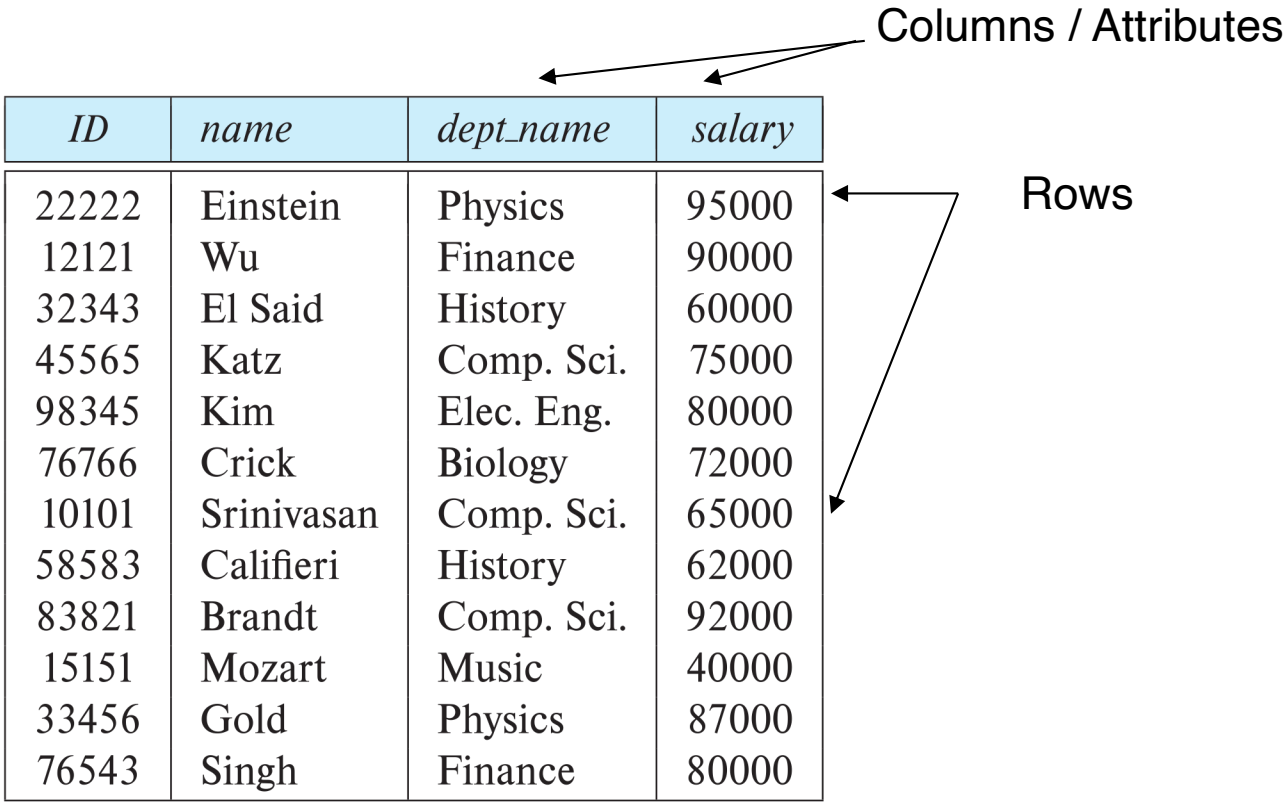
- A major purpose of a database system is to provide users with an abstract view of the data.
  - Data models
    - The underlying structure of the database
    - A collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.
  - Data abstraction
    - Hide the complexity of data structures to represent data in the database from users through several levels of data abstraction.
    - Simplify user's interaction with the system

Data Models

- Relational model
- Entity-Relationship (ER) data model (mainly for database design)
- Object-based data models (Object-oriented and Object-relational)
- Semi-structured data model

Relational Model

- A collection of tables/relations to represent both data and the relationships among those data
- All the data is stored in various tables.
- The most widely used data model in a vast majority of database systems



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

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

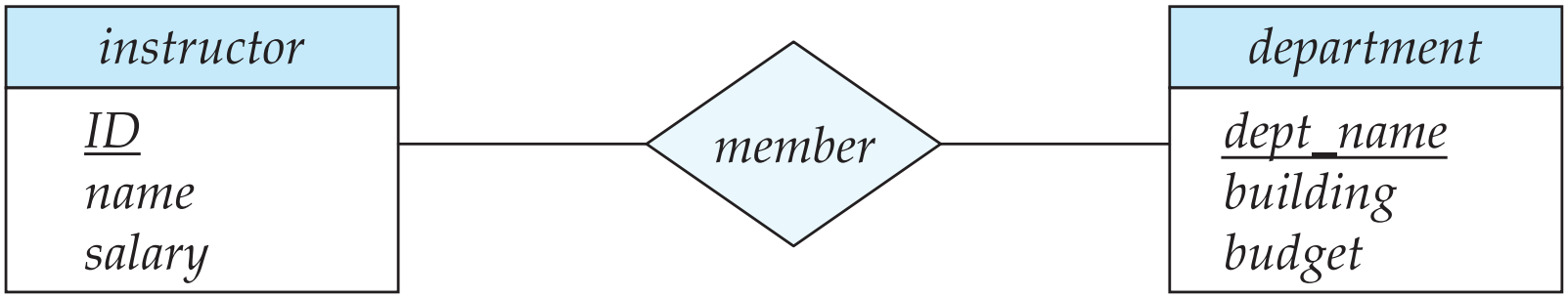
dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

The Entity-Relationship (ER) Model

- Models a domain as a collection of *entities* and *relationships*
  - Entity: a "thing" or "object" in the domain that is distinguishable from other objects
    - Described by a set of *attributes*
  - Relationship: an association among several entities

- Represented diagrammatically by an *entity-relationship diagram:*



Object-Relational Data Models

- Relational model: flat, “atomic” values
  - E.g., integer
- 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.

Semi-structured Data: XML and JSON

- **XML**: 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 used to be the basis for many data interchange formats
  - A wide variety of tools is available for parsing, browsing, and querying XML documents/data

- **JSON**: Javascript Object Notation
  - Semi-structured data format similar to XML, but simpler
  - Well integrated with web technologies
  - Widely used today

Example of XML

```
<purchase_order>
    <identifier> P-101 </identifier>
    <purchaser>
        <name> Cray Z. Coyote </name>
        <address> Route 66, Mesa Flats, Arizona 86047, USA  </address>
    </purchaser>
    <supplier>
        <name> Acme Supplies  </name>
        <address> 1 Broadway, New York, NY, USA  </address>
    </supplier>
    <itemlist>
        <item>
            <identifier>  RS1  </identifier>
            <description> Atom powered rocket sled  </description>
            <quantity> 2 </quantity>
            <price> 199.95 </price>
        </item>
        <item>
            <identifier>  SG2  </identifier>
            <description> Superb glue  </description>
            <quantity> 1 </quantity>
            <unit-of-measure> liter </unit-of-measure>
            <price> 29.95 </price>
        </item>
    </itemlist>
    <total_cost> 429.85 </total_cost>
    <payment_terms> Cash-on-delivery </payment_terms>
    <shipping_mode> 1-second-delivery </shipping_mode>
</purchase_order>
```

Example of JSON

```
{
    "ID": "22222",
    "name": {
        "firstname: "Albert",
        "lastname: "Einstein"
    },
    "deptname": "Physics",
    "children": [
        {"firstname": "Hans", "lastname": "Einstein" },
        {"firstname": "Eduard", "lastname": "Einstein" }
    ]
}
```

Levels of Abstraction

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

  type *instructor* = **record**

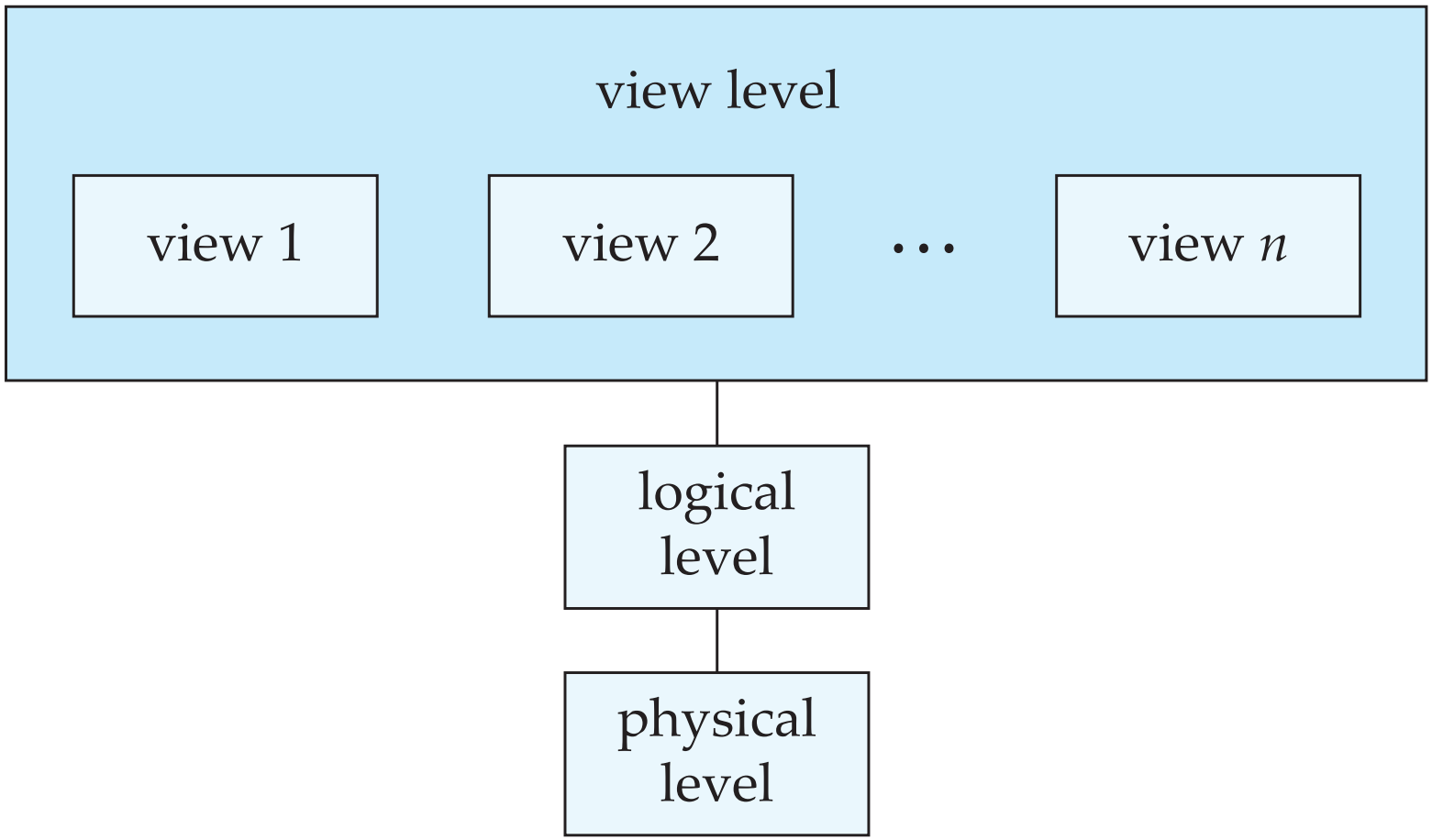
  ID : string;
  name : string;
  dept_name : string;
  salary : integer;

  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

- An architecture for a database system



Instances and Schemas

- **Schema** – the overall design of the database
- **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

Instances and Schemas

- **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.

- **Logical Data Independence** – the ability to modify the logical schema without changing the applications
  - For example, add new information to each employee.

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 updating the data organized by the appropriate data model
  - DML also known as query language
- Two types of data manipulation languages
  - **Procedural DML** -- a user specifies what data are needed and how to get those data.
  - **Declarative DML** -- a user specifies what data are needed without specifying how to get those data.
- The portion of a DML that involves information retrieval is called a **query** language.

Structured Query Language (SQL)

- **SQL**: widely used **declarative** (non-procedural) language
  - Example: Find the name of the instructor with ID 22222

    select *name*
    from *instructor*
    where *instructor.ID* = '22222'
  - Example: Find the ID and building of instructors in the Physics dept.

    select *instructor.ID*, *department.building*
    from *instructor*, *department*
    where *instructor.dept_name = department.dept_name* **and**
    department.dept_name = 'Physics'

- Application programs generally access databases through one of
  - Language extensions to allow embedded SQL
  - Application program interface (e.g., ODBC/JDBC) which allows SQL queries to be sent to a database

Database Access from Application Program

- Non-procedural query languages such as SQL are not as powerful as a universal Turing machine.
- SQL does not support actions such as input from users, output to displays, or communication over the network.
- Such computations and actions must be written in a **host language**, such as C/C++, Java, or Python, with embedded SQL queries that access the data in the database.
- **Application programs** -- are programs that are used to interact with the database in this fashion.

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
  - File organization
  - Internal storage structure

Database Design

- 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

Database Design

- **Example:** Changing the budget of the 'Physics' department
  - Updates to many rows!
  - Easy to break **integrity**
    - If we forget to update a row, then we have multiple budget values for the physics department!

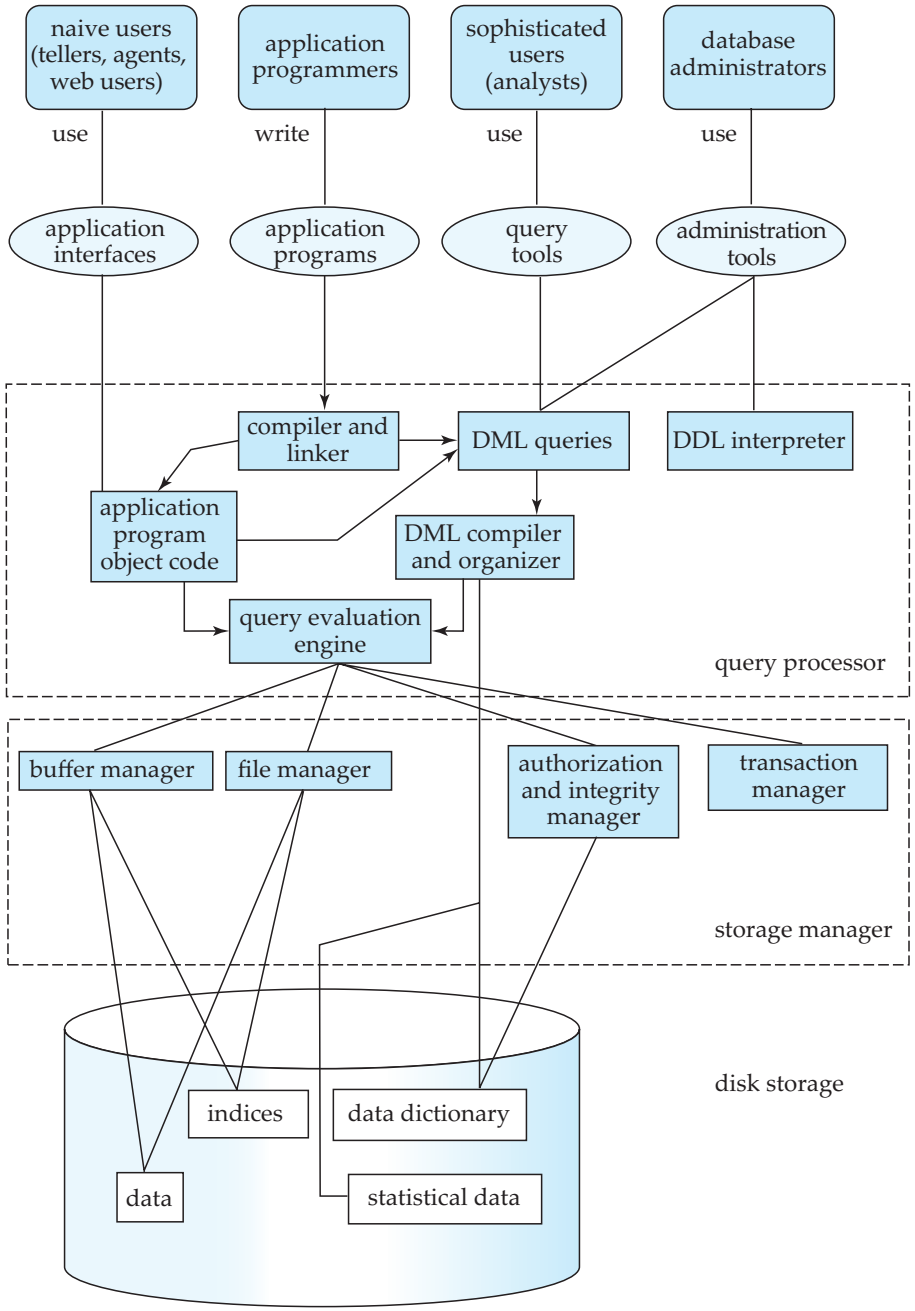
ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	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

Design Approaches

- Entity-Relationship (ER) Model (Chapter 6)
  - Models an domain as a collection of *entities* and *relationships*
    - Entity: a "thing" or "object" in the domain that is distinguishable from other objects
      - Described by a set of *attributes*
    - Relationship: an association among several entities
  - Represented diagrammatically by an *entity-relationship diagram*

- Normalization Theory (Chapter 7)
  - Formalize what designs are "good", and test for them
  - Translate a "bad" into a "good" design

Database System Internals



Storage Management

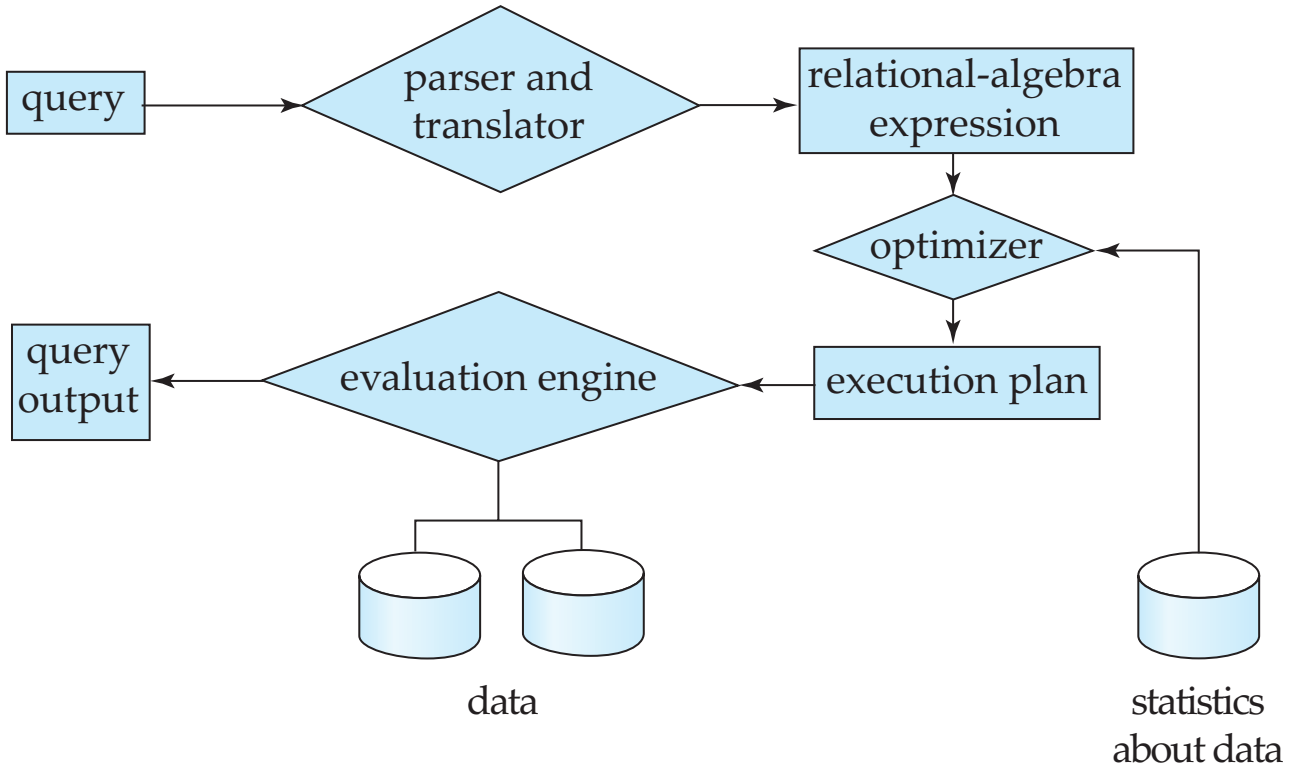
- A program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.

- The storage manager is responsible for 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 Processor

- The query processor components include:
  - DDL interpreter -- interprets DDL statements and records the definitions in the data dictionary.
  - DML compiler -- translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.
    - The DML compiler performs query optimization; that is, it picks the lowest cost evaluation plan from among the various alternatives.
  - Query evaluation engine -- executes low-level instructions generated by the DML compiler.

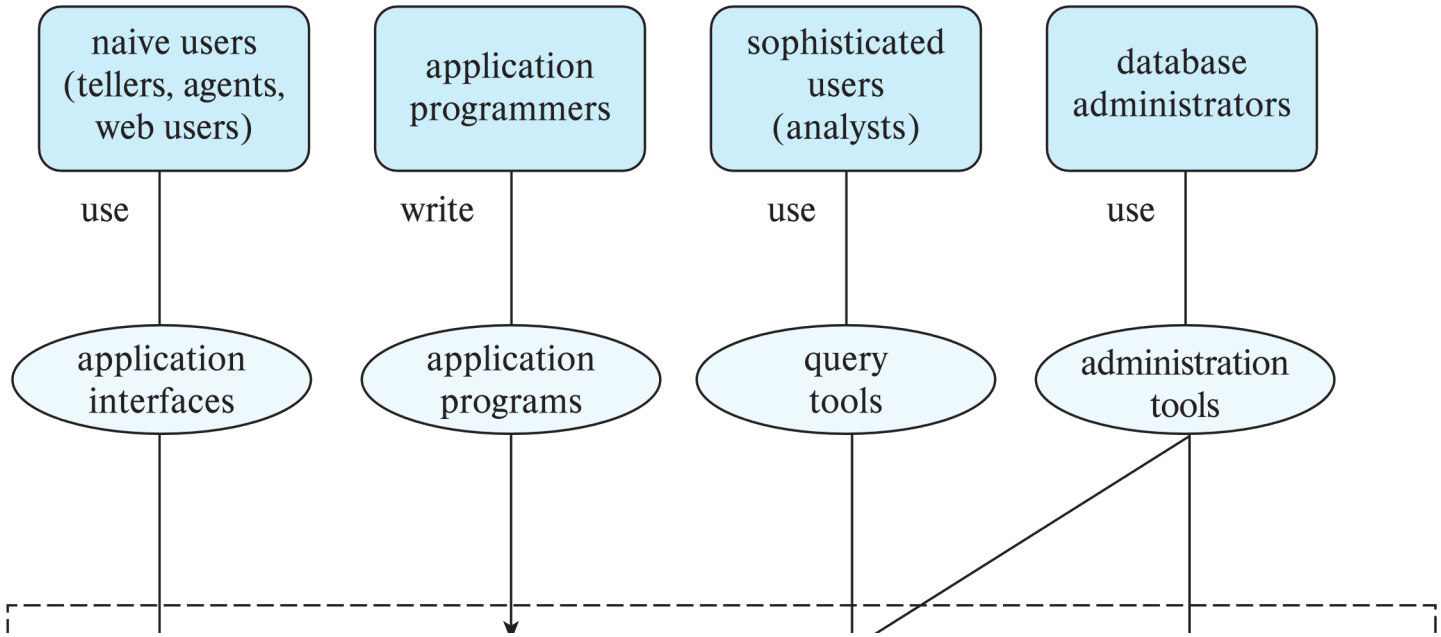
Transaction Management

- 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 Architecture

- The architecture of a database system is greatly influenced by the underlying computer system on which it runs
  - Centralized databases (embedded, e.g., SQLite)
  - Client-server (PostgreSQL, Oracle, etc.)
  - Parallel databases (Multi-processor)
  - Distributed databases (SparkSQL, Hive, etc.)

Database Users



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 (Michael Stonebraker) begins Ingres prototype
    - Oracle releases first commercial relational database
  - High-performance (for the era) transaction processing

History of Database systems

- 1980s:
  - Research relational prototypes evolve into commercial systems
    - SQL becomes the industrial standard.
  - Parallel and distributed database systems
    - Wisconsin, IBM, Teradata
  - Object-oriented database systems
- 1990s:
  - Large decision support and data-mining applications
  - Large multi-terabyte data warehouses
  - Emergence of Web commerce

Recap

- Why databases?
- What do databases do?
- Data independence
  - Physical and Logical
- Database design
- Data models
  - Relational, object, XML, network, hierarchical
- Query languages
  - DML
  - DDL
- Architecture and systems aspects of database systems
  - Recovery
  - Concurrency control
  - Query processing (optimization)
  - File organization and indexing
- History of databases

End of Chapter 1

- Database System Concepts 7th Edition
  - Chapter 1

Outline

- Introduction
- **Relational Data Model**
- Formal Relational Languages (relational algebra)
- SQL
- Database Design
- Transaction Processing, Recovery, and Concurrency Control
- Storage and File Structures
- Indexing and Hashing
- Query Processing and Optimization