ECE 448/528
Application Software Design

Lecture 27. Database Integration I

Spring 2025

Won-Jae Yi, Ph.D.

Department of Electrical and Computer Engineering
Illinois Institute of Technology

Relational Database

Motivation

- In RESTful services, we build data models to store data that will be served to RESTful clients like web applications.
  - Data structure and basic CRUD operations.
  - Use intrinsic lock to protect accesses from multiple threads.
- Additional features.
  - Complex queries involving multiple data models.
  - Data persistency.
  - Concurrent access.
  - All of the above to be done efficiently for GB/TB/PB of data.

Relational Database

- A classical approach for data management.
  - Restrict functionality to what can be expressed in relational algebra, usually captured by the SQL language.
  - Provide ACID (atomicity, consistency, isolation, durability) guarantee on database operations, including data persistency and concurrent access.
- Usually run as a standalone service that clients can access locally or remotely.
  - Via management tools
  - Via APIs that are available from most programming languages.

ACID Guarantee

- Database updates are grouped into transactions to support application logic.
  - e.g. if Alice needs to transfer $100 to Bob, the transaction needs to deduct $100 from Alice’s account and add $100 to Bob’s account.
- **Atomicity**: either the transaction succeeds or fails as a whole.
  - It is not allowed to deduct $100 from Alice’s account while not changing Bob’s account.
- **Consistency**: database remains valid after transactions are executed.
  - Transactions are committed if succeed. Later transactions will see the changes.
  - Failed transactions should not change the database.
  - Transactions, if committed, should execute correctly, e.g., it is not allowed to deduct $100 from Alice’s account while adding $50 to Bob’s account, and not allowed for Alice to have a negative balance.

ACID Guarantee

- **Isolation**: transactions are executed as if sequentially.
  - Actual implementations may execute transactions concurrently to achieve better performance.
  - However, the outcome should be the same as if the transactions are executed one after another – note that the order is not specified.
  - e.g., if we assume Alice initially has \$0 in her account and that at the same time, Alice transfers \$100 to Bob, Carol transfers \$200 to Alice, then both are possible that the transaction from Alice to Bob succeeds or fails.
- **Durability**: committed transactions survive system failures.
  - Usually by storing data on a drive. (RAID?)
  - To the extent that the drive won't fail.
- It is quite challenging to achieve ACID at the same time.
  - e.g., what if there is a power outage when the database is about to commit one transaction by writing data to the disks?

Database

- An integrated, self-describing collection of data about related sets of things and the relationships among them
- Simple text files or office documents are one way to store data
  - Fine for small amounts of data
  - But impractical for large amounts of data
- Businesses must maintain huge amounts of data
  - A database management system (DBMS) is the typical solution to the data needs of business
  - Designed to store, retrieve and manipulate data
- Most programming languages can communicate with several DBMS
  - Tells DBMS what data to retrieve or manipulate
- Structured collection of data in **Tables, Fields, Query, Reports**

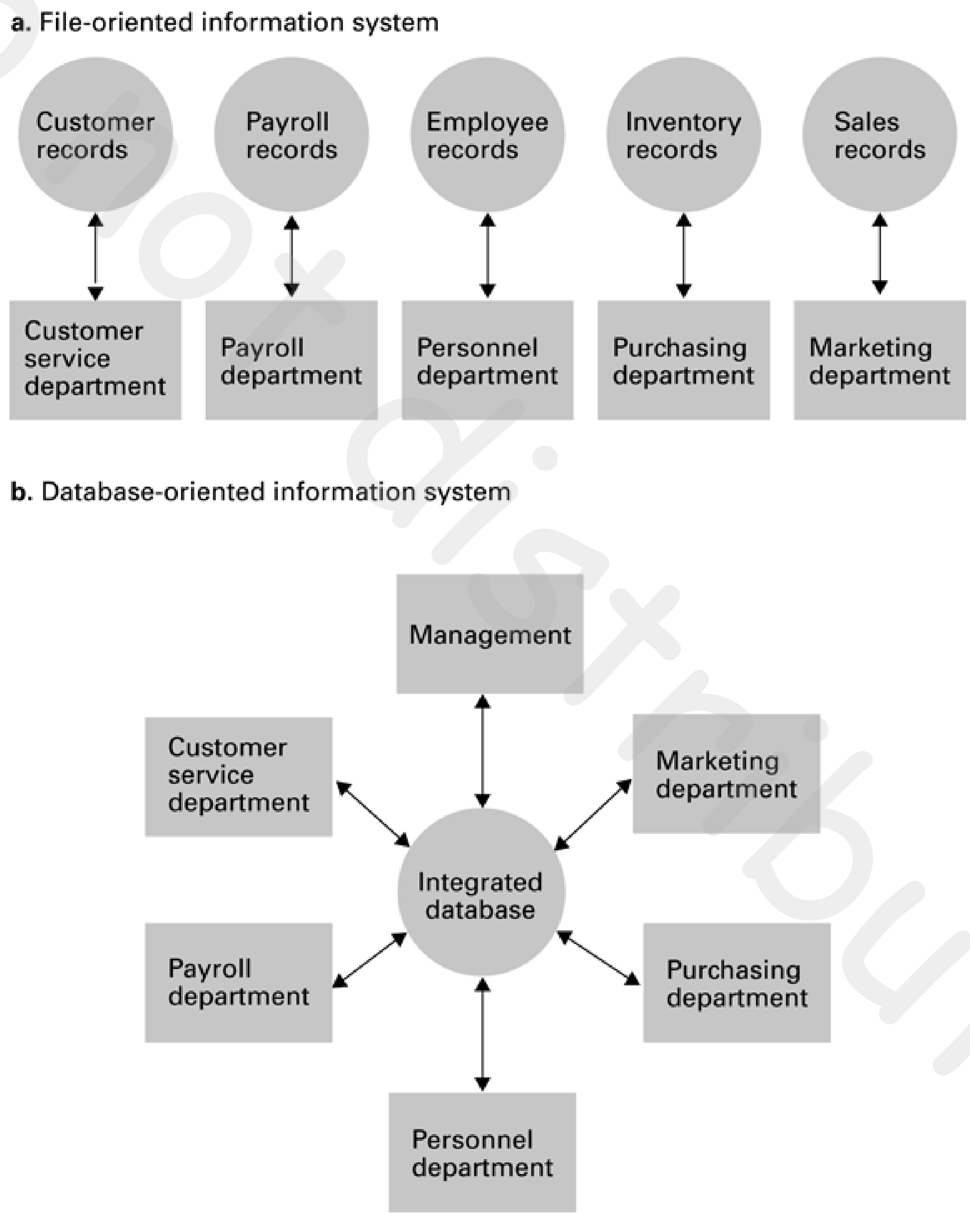
Database

- Optimizes data management and transforms data into information
  - Data: 273459368, Information: Serial Number: 273-45-9368
- Importance of Database design
  - Defines the database's expected use
    - Different approach needed for different types of databases
  - Avoid data redundancy and ensure data integrity
    - Data is accurate and verifiable
  - Poorly designed database generates errors
    - Leads to bad decisions, leading to failure of organization
- Functions of DBMS/Database System
  - Stores data and related data entry forms, report definitions, etc.
  - Hides the complexities of relational database model from the user
    - Facilitates the construction/definition of data elements and their relationship
    - Enables data transformation and presentation
  - Enforces data integrity
  - Implements data security management
    - Access, privacy, backup and restoration

Database

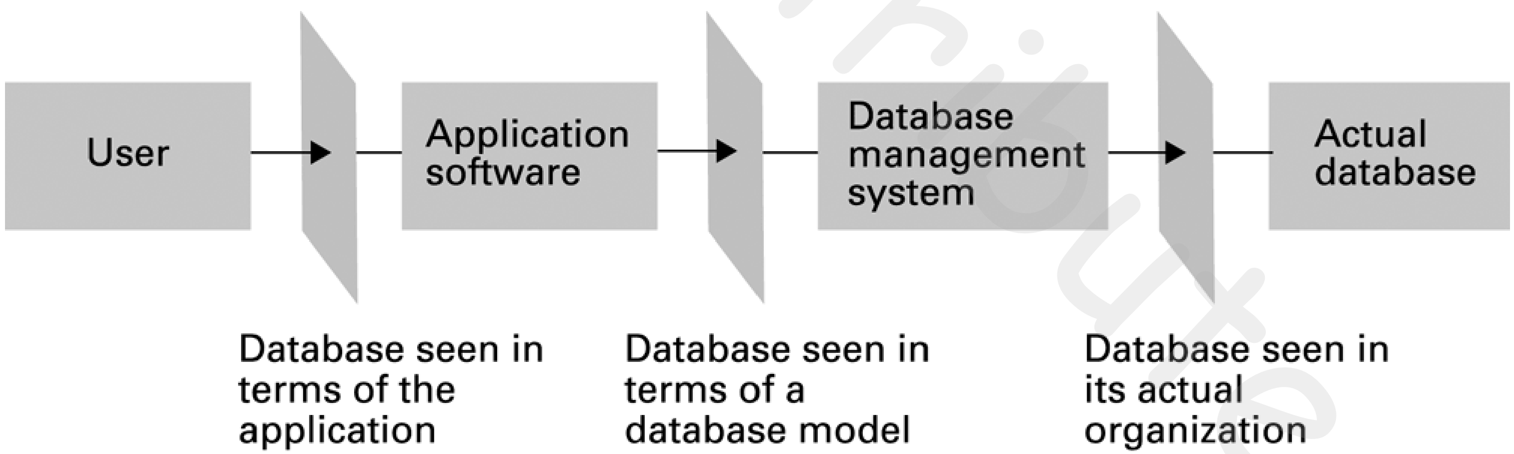
- **Tables**: a list of data organized into fields and records
- **Queries**: question structure to sort, filter and select specific information
- **Forms**: structures for screen views of data
- **Reports**: structures for written output of data
- **Program Modules & Macros**: program code to perform specific actions

Database – File vs. DB Organization



Layered Approach to using a DBMS

- Applications that work with a DBMS use a layered approach
  - Application is topmost layer
  - Application sends instructions to next layer, the DBMS
  - DBMS works directly with data
- Programmer need not understand the physical structure of the data
  - Just need to know how to interact with the database



Relational Database Model

- Problems with legacy database systems
  - **Required excessive effort to maintain**
    - Data manipulation (programs) too dependent on physical file structure
  - **Hard to manipulate by end-users**
    - No capacity for ad-hoc query (must rely on DB programmers)
- Evolution in Data Organization
  - **E.F. Codd's Relational Model proposal**
    - Separated the notion of physical representations (*machine-view*) from logical representation (*human-view*)
    - Considered ingenious but computationally impractical in 70s
  - **Relational Database Model**
    - Dominant database model of today
    - Eliminated pointers and used tables to represent data

Relational Database Model

- Introduced in the 60’s and 70’s, is the most common type of DBMS today
- Data elements stored in simple tables (related)
- General structure good for many problems
- Easy to understand, modify and maintain
- Views entities as two-dimensional tables
    - Records are rows
    - Attributes (fields) are columns
- Tables can be linked
- Supports one-to-many, many-to-many, one-to-one relationships

boats

bid	bname	color	price
101	Interlake	blue	30000
102	Interlake	red	29000
103	Clipper	green	42000
104	Marine	red	18000

Relational Database Model

- Provides a logical “human-level” view of the data and associations among groups of data (e.g., tables)

Customer_ID	Customer_Account	Agent_ID
1224	4556	23
1225	4558	25

Agent_ID	Last_Name	First_Name	Phone
23	Sturm	David	334-5678
25	Long	Kyle	556-3421

Customer_ID	Last_Name	First_Name	Phone	Account_Balance
1224	Vira	Dyne	678-9987	1223.95
1225	Davies	Tricia	556-3342	234.25

Relational Database Model

- Advantages
    - **Structural independence**
        - Separation of database design and physical data storage/access
        - Easier database design, implementation, management, and use
    - **Ad hoc query capability with Structure Query Language (SQL)**
        - SQL translates user queries
- Disadvantages
    - **Substantial hardware and system software overhead**
        - more complex system
    - **Poor design and implementation is made easy**
        - ease-of-use allows careless use of RDBMS

Data Models in Relational Database

- Data are organized into tables or relations.
- Each table consists of many rows or tuples of data.
- Each row consists of many columns or attributes or fields
  - Rows in the same table should have the same columns.
- Each row should have a special column called the key or the primary key that is unique among the rows in the same table.
  - Allow one to quickly locate the row given its key.
- Each column of a row is usually of an elementary data type.
  - That can be compared and operated on.
  - Opaque binary blobs are also supported by many database systems to store data like images.

Object Relational Mapping (ORM)

- Mapping objects with relational database automatically
  - *Relational database ← (Mapping) → Object fields*
- It is possible to recast data models from a relational database into object-based ones.
  - For developers that prefer OOP over relational algebra.
- Each table corresponds to one object, e.g., a class.
  - Rows are objects.
  - Columns are data members.
  - Tables are implemented as maps of key/value pairs, allowing to manage the objects and to search for a specific object using its primary key.
- Nevertheless, recent trends move toward utilizing such relationships to manage a large number of objects using relational algebra.
  - Developers tend to make fewer mistakes when they are restricted to relational algebra.
  - Relational algebra enables additional optimizations when processing a large amount of data.

ORM Advantages and Disadvantages

- **Advantages**
  - Allow to focus on business logics more
    - OOP is intuitive. ORM enables to control database with intuitive methods rather than complicated SQL queries.
  - Reusability and Maintenance
  - Less dependent on DBMS
- **Disadvantages**
  - Difficult to implement a system only with ORM
  - If a system is busy with other processes to deal with, the performance may be degraded.

Relational Algebra and SQL

SQL Query

```
SELECT users.id, SUM(orders.total) AS total_spending,
FROM users JOIN orders ON (users.id=orders.buyer_id)
WHERE orders.year=2023
GROUP BY users.id
ORDER BY total_spending DESC;
```

- SQL queries start with the SELECT clause.
- Each query will return rows of data.
  - Each row may contain data from multiple tables.
  - Columns are specified in the SELECT clause.
  - e.g.) two columns users.id and total_spending are generated here.

Data Source

```
SELECT …
FROM users JOIN orders ON (users.id=orders.buyer_id)
…
```

- The FROM clause specifies data to query from.
- You may query data from a single table, or
- From multiple tables by joining them together.
  - So that relevant data can be retrieved from multiple tables at the same time.

Join

```
SELECT …
FROM users JOIN orders ON (users.id=orders.buyer_id)
…
```

- There are many kinds of JOINs: one method to understand all of them is to consider JOIN as a two-step process.
- Step 1: form a new table by taking the Cartesian product of the tables.
  - e.g. if users have N rows and orders have M rows, the new table will have NxM rows, each consisting of a row from users and a row from orders.
- Step 2: remove rows from the new table following certain criteria as defined by different JOINs.
  - For the above example, we remove the rows where users.id and orders.buyer_id are different.
  - In other words, the new table lists buyers and their orders together.

Filtering

```
SELECT ...
FROM ...
WHERE orders.year=2023
...
```

- The WHERE clause filters rows by a given condition.
  - So that a portion of the whole table may be retrieved.
  - e.g., for this query, we only care about orders placed in 2023

Group By and Aggregation

```
SELECT users.id, SUM(orders.total) AS total_spending,
FROM ...
WHERE ...
GROUP BY users.id
...
```

- Rows in the joined new table may be further grouped via GROUP BY clause.
  - e.g., to group all rows belonging to the same buyer together.
- As SQL only operates on rows but not groups of rows, rows from each group must be aggregated into a new row.
  - via aggregate functions like SUM to calculate the total spending of each buyer.

Output Ordering

```
SELECT users.id, SUM(orders.total) AS total_spending,
FROM users JOIN orders ON (users.id=orders.buyer_id)
WHERE orders.year=2023
GROUP BY users.id
ORDER BY total_spending DESC;
```

- Finally, the output rows may be sorted via ORDER BY.
  - Either ascending (ASC) or descending (DESC).
  - So that we can find who spends the most for 2023.