ECE 473/573
Cloud Computing and Cloud Native Systems
Lecture 11 Database Systems

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

September 22, 2025

Outline

Reading Assignment

- This lecture: Database systems
- Next two lectures: Distributed database systems
  - Cassandra - A Decentralized Structured Storage System
    https://www.cs.cornell.edu/projects/ladis2009/papers/lakshman-ladis2009.pdf
  - Spanner: Google's Globally-Distributed Database
    http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en//archive/spanner-osdi2012.pdf

Outline

Cloud Storage

- A fundamental component of cloud computing.
  - Persist state of microservices and applications.
  - Store intermediate data to facilitate communication and fault resilience.
- Metrics
  - Size
  - Performance: throughput and latency
  - Availability and reliability
  - Leverage scalability to improve all of them.
- Different types of cloud storage may have different trade-offs.
  - Block storage and file systems
  - Object storage
  - Database systems

Block Storage and File Systems

- Block storage provides byte blocks of fixed size that can be accessed randomly.
  - E.g. hard drives and solid-state drives.
  - Available locally or through a dedicated network (SAN) for high throughput and low latency.
- File systems built on top of block storage provide support to
  - Organize data as files and directories
  - Share files over network
  - Checksum, versioning, and redundancy
  - Security features like permission and encryption
- Not scalable
  - Strong tie to the underlying hardware for performance
  - Exclusive access is required to update a block.

Object Storage

- Manage data as objects that must be modified as a whole.
  - Accessed via networked services.
  - Use a key as identifier instead of a name.
  - Need other mechanisms to support hierarchy.
- Highly scalable
  - Able to utilize physical storage from many servers via networked services.
  - Many objects are not modified after creation – easy to maintain multiple copies of the same object.
- Optimize for different access patterns
  - Backups that are mostly write-once without read.
  - Intermediate data that require only sequential access.
  - Media files that are mostly read-only but need to be read frequently from all over the world.

Database Systems

- Provide rich accesses to highly structured data beyond read/write.
- Relational (SQL) database
  - Very strong guarantee on data consistency – a must to manage data that need to be consistent like payments.
  - Mature and well-understood.
  - Not scalable – need to maintain a lot of internal states.
- NoSQL databases
  - High scalability by giving up some part of the consistency guarantees of SQL databases.
  - Different NoSQL databases may explore different trade-offs to favorite different applications, making it tricky to pick up the right one to meet the requirement.

Outline

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 guarantee on database operations including data persistency and concurrent access.
- Usually run as a stand-alone service that clients can access locally or remotely.
  - Via management tools, or
  - 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 need to transfer \$100 to Bob, the transaction need 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 (Cont.)

- 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 initialy 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.
  - To the extent that the drive won't fail.
- It is quite challenge 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?

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.
  - Additionally to support a range query of keys.
- 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.

Outline

Cloud Storage

Relational Database

Relational Algebra and SQL

SQL Query

```sql
SELECT users.id, SUM(orders.total) 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 `JOIN`s: 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.
  - If `users` has N rows and `orders` has M rows, the new table will have NM rows, each consists 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 `JOIN`s.
  - For the above example, we remove the rows where `users.id` and `orders.buyer_id` are different.
  - The new table lists buyers and their orders together.
- Actual implementations may eliminate the need to calculate the Cartesian product depending if the criteria involves keys.

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.

Grouping and Aggregation

```
SELECT users.id, SUM(orders.total) 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) 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.

Other SQL Statements

- There are other SQL statements to create, update, and delete rows from tables and to manage tables as well.
- Check `https://www.w3schools.com/sql/default.asp` and run examples there to learn SQL.