Lecture #01: Relational Model & Algebra

15-445/645 Database Systems (Spring 2024)
https://15445.courses.cs.cmu.edu/spring2024/
Carnegie Mellon University
Jignesh Patel

1 Databases

A *database* is an organized collection of inter-related data that models some aspect of the real-world (e.g., modeling the students in a class or a digital music store). People often confuse "databases" with "database management systems" (e.g., MySQL, Oracle, MongoDB, Snowflake). A database management system (DBMS) is the software that manages a database.

Consider a database that models a digital music store (e.g., Spotify). Let the database hold information about the artists and which albums those artists have released.

2 Flat File Strawman

Database is stored as comma-separated value (CSV) files that the DBMS manages. Each entity will be stored in its own file. The application has to parse files each time it wants to read or update records.

Keeping along with the digital music store example, there would be two files: one for artist and the other for album.

Each entity has its own set of attributes, so in each file, different records are delimited by new lines, while each of the corresponding attributes within a record are delimited by a comma.

Example: An artist could have a name, year, and country attributes, while an album has name, artist and year attributes.

The following is an example CSV file for information about artists with the schema (name, year, country):

```
"Wu-Tang Clan", 1992, "USA"
"Notorious BIG", 1992, "USA"
"GZA", 1990, "USA"
```

Issues with Flat File (for the digital music store example)

- **Data Integrity** How do we ensure that the artisi is the same for each album entry?What if somebody overwrites the album year with an invalid string? What if there are multiple artists on an album? What happens if we delete an artist that has albums?
- **Implementation** How do you find a particular record? What if we now want to create a new application that uses the same database? What if that application is running on a different machine? What if two threads try to write to the same file at the same time?
- **Durability** What if the machine crashes while our program is updating a record? What if we want to replicate the database on multiple machines for high availability?

3 Database Management System

A *DBMS* is a software that allows applications to store and analyze information in a database.

A general-purpose DBMS is designed to allow the definition, creation, querying, update, and administration of databases in accordance with some data model.

A *data model* is a collection of concepts for describing the data in database.

Examples: relational (most common), NoSQL (key/value, document, graph), array/matrix/vectors (for machine learning)

A *schema* is a description of a particular collection of data based on a data model.

Common Data Models

- Relational (Most DBMSs)
- Key/Value (NoSQL)
- Graph (NoSQL)
- Document/XML/Object (NoSQL)
- Wide-Column/Column-family (NoSQL)
- Array/Matrix/Vectors (Machine Learning)
- Hierarchical (Obsolete/Legacy/Rare)
- Network (Obsolete/Legacy/Rare)
- Multi-Value (Obsolete/Legacy/Rare)

Early DBMSs

Early database applications were difficult to build and maintain because there was a tight coupling between logical and physical layers.

The logical layer describes which entities and attributes the database has while the physical layer is how those entities and attributes are being stored. Early on, the physical layer was defined in the application code, so if we wanted to change the physical layer the application was using, we would have to change all of the code to match the new physical layer.

4 Relational Model

Ted Codd noticed that people were rewriting DBMSs every time they wanted to change the physical layer, so in 1969 he proposed the relational model to avoid this.

The relational model defines a database abstraction based on relations to avoid maintenance overhead. It has three key points:

- Store database in simple data structures (relations).
- Access data through high-level language, DBMS figures out best execution strategy.
- Physical storage left up to the DBMS implementation.

The relational data model defines three concepts:

- **Structure:** The definition of relations and their contents. This is the attributes the relations have and the values that those attributes can hold.
- **Integrity:** Ensure the database's contents satisfy constraints. An example constraint would be that any value for the year attribute has to be a number.
- **Manipulation:** How to access and modify a database's contents.

A *relation* is an unordered set that contains the relationship of attributes that represent entities. Since the relationships are unordered, the DBMS can store them in any way it wants, allowing for optimization. It is possible to have repeated elements in a relation.

A *tuple* is a set of attribute values (also known as its *domain*) in the relation. Originally, values had to be atomic or scalar, but now values can also be lists or nested data structures. Every attribute can be a special value, NULL, which means for a given tuple the attribute is undefined.

A relation with n attributes is called an n*-ary relation*. We will interchangeably use *relation* and *table* in this course. An n-ary relation is equivalent to a table with n columns.

Keys

A relation's *primary key* uniquely identifies a single tuple. Some DBMSs automatically create an internal primary key if you do not define one. A lot of DBMSs have support for autogenerated keys so an application does not have to manually increment the keys, but a primary key is still required for some DBMSs.

A *foreign key* specifies that an attribute from one relation has to map to a tuple in another relation. For example, we can include artist id (foreign key referring to the artist table) in the album table.

Constraints

A *constraint* is a user-defined condition that must hold for any instance of the database.

5 Data Manipulation Languages (DMLs)

Methods to store and retrieve information from a database. There are two classes of languages for this:

- **Procedural:** The query specifies the (high-level) strategy the DBMS should use to find the desired result based on sets / bags. For example, use a for loop to scan all records and count how many records are there to retrieve the number of records in the table.
- **Non-Procedural (Declarative):** The query specifies only *what* data is wanted and not *how* to find it. For example, use SQL `select count(*) from artist` to count how many records are there in the table.

6 Relational Algebra

Relational Algebra is a set of fundamental operations to retrieve and manipulate tuples in a relation. Each operator takes in one or more relations as inputs, and outputs a new relation. To write queries we can "chain" these operators together to create more complex operations.

Select

Select takes in a relation and outputs a subset of the tuples from that relation that satisfy a selection predicate. The predicate acts like a filter, and we can combine multiple predicates using conjunctions and disjunctions.

Syntax: $\sigma_{\text{predicate}}(R)$.

Example: $\sigma_{\text{a\_id='a2'}}(R)$

SQL: `SELECT * FROM R WHERE a_id = 'a2'`

Projection

Projection takes in a relation and outputs a relation with tuples that contain only specified attributes. You can rearrange the ordering of the attributes in the input relation as well as manipulate the values.

Syntax: $\pi_{\text{A1,A2,...,An}}(R)$.

Example: $\pi_{\text{b\_id-100, a\_id}}(\sigma_{\text{a\_id='a2'}}(R))$

SQL: `SELECT b_id-100, a_id FROM R WHERE a_id = 'a2'`

Union

Union takes in two relations and outputs a relation that contains all tuples that appear in at least one of the input relations. Note: The two input relations have to have the exact same attributes.

Syntax: $(R \cup S)$.

SQL: `(SELECT * FROM R) UNION ALL (SELECT * FROM S)`

Intersection

Intersection takes in two relations and outputs a relation that contains all tuples that appear in both of the input relations. Note: The two input relations have to have the exact same attributes.

Syntax: $(R \cap S)$.

SQL: `(SELECT * FROM R) INTERSECT (SELECT * FROM S)`

Difference

Difference takes in two relations and outputs a relation that contains all tuples that appear in the first relation but not the second relation. Note: The two input relations have to have the exact same attributes.

Syntax: $(R - S)$.

SQL: `(SELECT * FROM R) EXCEPT (SELECT * FROM S)`

Product

Product takes in two relations and outputs a relation that contains all possible combinations for tuples from the input relations.

Syntax: $(R \times S)$.

SQL: `(SELECT * FROM R) CROSS JOIN (SELECT * FROM S)`, or simply `SELECT * FROM R, S`

Join

Join takes in two relations and outputs a relation that contains all the tuples that are a combination of two tuples where for each attribute that the two relations share, the values for that attribute of both tuples is the same.

Syntax: $(R \bowtie S)$.

SQL: `SELECT * FROM R JOIN S USING (ATTRIBUTE1, ATTRIBUTE2...)`

Observation

Relational algebra is a procedural language because it defines the high level-steps of how to compute a query. For example, $\sigma_{\text{b\_id=102}}(R \bowtie S)$ is saying to first do the join of R and S and then do the select, whereas $(R \bowtie (\sigma_{\text{b\_id=102}}(S)))$ will do the select on S first, and then do the join. These two statements will actually produce the same answer, but if there is only 1 tuple in S with b_id=102 out of a billion tuples, then $(R \bowtie (\sigma_{\text{b\_id=102}}(S)))$ will be significantly faster than $\sigma_{\text{b\_id=102}}(R \bowtie S)$.

A better approach is to say the result you want (retrieve the joined tuples from R and S where b_{id} equals 102), and let the DBMS decide the steps it wants to take to compute the query. SQL will do exactly this, and it is the de facto standard for writing queries on relational model databases.

7 Other Data Models

Document Data Model

A collection of record documents containing a hierarchy of named field/value pairs.

A field's value can be either a scalar type, an array of values, or another document.

Modern implementations use JSON. Older systems use XML or custom object representations.

Vector Data Model

One-dimensional arrays used for nearest-neighbor search (exact or approximate).

Used for semantic search on embeddings generated by ML-trained transformer models (think ChatGPT).

Native integration with modern ML tools and APIs (e.g., LangChain, OpenAI).

At their core, these systems use specialized indexes to perform NN searches quickly.

8 P0 Intro: CRDT

Conflict-free Replicated Data Type(CRDT) is a type of data structure that enables concurrent updates across multiple replicas without the need for coordination between them.

This is useful in scenarios we want a distributed data structure (many copies) that allows local updates to be made independently and the states eventually converge.

Take, for example, a "global" counter that can be incremented independently by each node. These nodes can communicate or "gossip" at any time, sharing their state. The objective is for every node to ultimately reflect the same accurate global value. The key to achieving this with a CRDT lies in its merge function. This method is crafted to be commutative, associative, and idempotent, ensuring reliable data convergence.