CS143: Database Systems

Professor Cho

Thilan Tran

Winter 2021

Contents

CS143: Database Systems

- **database management system (DBMS)** is a way to manage and store data:
    - how does a database differ from a spreadsheet software like Excel?
        - expected to *efficiently* scale to a *massive* amount of data, without suffering
        - expected to persist the data
        - expected to provide secured and safe access to data
        - expected to be conveniently used by a large number of clients at a time
    - program data is not assumed to entirely reside in main memory RAM, but instead in disk
        - this leads to utilization of different data structures
- database architecture:
    1. disk for data (sometimes stored in main memory, if data can fit)
    2. OS
    3. DBMS engine
        - database system may access disk through OS, or directly through raw IO
    4. API
        - eg. standard APIs like JDBC (Java), ODBC (Microsoft)
    5. app, or CLI

    - downloading a DBMS software like MySQL installs parts 3, 4, and CLI
- popular DBMS software:
    - relational:
        - open source: MySQL, PostgreSQL
        - closed source: Oracle, Microsoft SQL, IBM DB2
    - non-relational (NoSQL):
        - MongoDB, Spark
- five steps in database construction:
    1. domain analysis
        - captured in an entity-relationship (ER) model, or with unified modeling language (UML)
    2. database design
        - normalization theory
    3. table creation
        - uses a data definition language (DDL)
    4. load
        - using SQL, or bulk load
    5. query and update
        - using data manipulation language (DML)

Data Models

- what is a data model? why do we need it?
  - eg. the human brain acts as a data model
    - remembers information, easily accessible and understandable data
  - how do we translate the memory of a human brain in computers?
    - computers speak in binary, only stores data in a very specific, concrete format
  - a **data model** is a very specific way to model or represent data in comptuers:
- types of data models include graph, tree, relational models:
  - a **graph model** (or network model) has nodes, edges, and labels, eg. airline flights:
    - initially most popular data model
    - more flexible and semistructured, often used for JSON data eg. MongoDB
  - a **tree model** (or hierarchical model) is a graph arranged as a tree, eg. company hierarchies
  - in a **relational model**, all data is represented as a set of tables:
    - introduced in 1970 by Codd, and completely revolutionized the database field
    - in graph and tree models, not as efficient to access and modify data:
      - intead, represent all data through relationships and store them in tables, like a spreadsheet
      - a downside to the relational model is the rigidity of its schemas
    - in mathematics, tables are expressed as **relations** eg. tuples, triplets, etc.
    - now, the most popular data model used in DBs

Relational Models

- relational model terminology:
  - **relations** ie. tables contain many **tuples** ie. rows, each with various **attributes** ie. columns
  - each attribute has a **domain** ie. type
  - a **schema** is the structure of relations in a database:
    - includes properties like the relation name, attribute names, domains
    - eg. `Student(sid: int, name: str, addr: Addr, age: int, GPA: int)`

    - the **instance** is the actual data populating a relation that conforms to some schema
        - ie. schema is the variable type, and the instance is the variable value
    - **keys** are a set of attributes that *uniquely* identify a tuple in a relation:
        - multiple keys are possible
        - eg. in `Course(dept, cnum, sec, unit, instructor, title)`, the keys may be the department, course number, and section number
            - or alternatively department, section, and title if course numbers are repeated
        - generally, assuming the set of attributes for keys is the minimum set
            - in the worst case, with no duplicates, a relation's keys are all its attributes
    - more specifically, different types of keys:
        1. a **super key** is any key
        2. a **candidate key** is a key with the minimum number of attributes
        3. a **primary key** is a candidate key chosen to be used as the main key for a relation
            - shouldn't have null values
    - **null values** are used to indicate not applicable, uninitialized values, etc.:
        - are compatible with every type, unless otherwise explicitly defined
        - leads to complications, eg. comparisons in conditional queries
            - thus DBMS may return unexpected answers
        - requires **3-valued logic** where every condition can be true, false, or unknown:
            - need concrete rules to deal with null and unknown values
            - adds increased implementation and execution complexity
- name scopes of relational models:
    - the name of a relation is unique across relations
    - the names of attributes are uniue in a table
        - thought there cane be the same attribute names in different tables
- set semantics in relational models:
    - in a set, there are no duplicate elements, and order is unimportant
    - thus in the relational model:
        1. no duplicate tuples are allowed
            - but SQL allows duplicate tuples for practical reasons
        2. attribute order does not matter

Relational Algebra

- relational query langauges include:
  - formal languages eg. relational algebra, relational calculus, datalog
  - practical languages eg. SQL, Quel, QBE
- in relational algebra:
  - inputs and outputs are both relations, so **piping** is possible
  - set semantics are followed, so duplicates are automatically eliminated

Table 1: Students

sid	name	addr	age	GPA
301	John	183 Westwood	19	2.1
303	Elaine	301 Wilshire	17	3.9
401	James	183 Westwood	17	3.5
208	Esther	421 Wilshire	20	3.1

Table 2: Classes

dept	cnum	sec	unit	title	instructor
CS	112	01	03	Modeling	Dick Muntz
CS	143	01	04	DB Systems	John Cho
EE	143	01	03	Signals	Dick Muntz
ME	183	02	05	Mechanics	Susan Tracey

Table 3: Enrollments

sid	dept	cnum	sec
301	CS	112	01
301	CS	143	01
303	EE	143	01
303	CS	112	01
401	CS	112	01

- queries can be done using the formal relational algebra language
  - consider the following example queries using Table 1, Table 2, and Table 3

1. get all the students:

$$Student$$

  - to get all the tuples from a relation, just write the name of the relation
2. get all students with age > 18:

$$\sigma_{age<18}(Student)$$

  - the select operator $\sigma_C(R)$ filters based on boolean expression C
    - R can be either a relation or a result from another operator
3. get all students with GPA > 3.7 and age < 18:

$$\sigma_{gpa>3.7 \wedge age<18}(Student)$$

4. get student ID and GPA of all students:

$$\pi_{sid,\text{GPA}}(Student)$$

  - need a different operator
  - the project operator $\pi_A(R)$ filters column-wide based on attributes A
    - returns a new set of *columns*
5. get all departments offering a class:

$$\pi_{dept}(Class)$$

  - due to set semantics, does not return two elements for “CS”
6. get student ID and GPA of students with age < 18:

$$\pi_{sid,\text{GPA}}(\sigma_{age<18}(Student))$$

  - composing operators next to each other
  - however, for projection, it is not useful to compose projections

- the **cross product** ie. Cartesian product operator in relational algebra:
  - $R \times S = \{t | t = (r, s) \ \forall \ r \in R, s \in S\}$ concatenates every tuple of each relation together:
    * if $|R| = r$ and $|S| = s$, $|R \times S| = rs$
    * eg. $R \times S$ contains $a_1 b_2, a_1 b_2, \ldots, a_n b_m$
  - ie. creates one output per every pair of input tuples
  - useful when combining tuples from multiple relations

7. get the names of students who take CS classes:

$$\pi_{name}(\sigma_{dept=\text{'CS'}}(\sigma_{Student.sid=Enroll.sid}(Student \times Enroll)))$$

$$\pi_{name}(\sigma_{dept=\text{'CS'} \wedge Student.sid=Enroll.sid}(Student \times Enroll))$$

  - but this cross product is quite expensive, so we can change the ordering to improve efficiency

$$\pi_{name}(\sigma_{Student.sid=Enroll.sid}(Student \times \sigma_{dept=\text{'CS'}}(Enroll)))$$

  - equivalently, using the natural join operator:

$$\pi_{name}(\sigma_{dept=\text{'CS'}}(Student \bowtie Enroll))$$

- the **natural join** operator $\bowtie$ is used to join two tables *naturally*:
  - filters the Cartesian product by the tuples based on the equality conditions of *all* shared attributes
  - equivalent to:

$$R \bowtie S = \sigma_{\langle equal \quad shared \quad attributes \rangle}(R \times S)$$

Table 4: $Class \bowtie Enroll$

dept	cnum	sec	unit	title	ins	sid	dept	cnum	sec
CS	112	01	03	Modeling	Dick Muntz	301	CS	112	01
CS	112	01	03	Modeling	Dick Muntz	303	CS	112	01
CS	112	01	03	Modeling	Dick Muntz	401	CS	112	01
CS	143	01	03	DB Systems	John Cho	301	CS	143	01
EE	143	01	03	Signals	Dick Muntz	303	EE	143	01

8. get the names of students who take classes offered by `'Dick Muntz'`:

$$\pi_{name}(Student \bowtie (Enroll \bowtie \sigma_{inst=\text{'Dick Muntz'}}(Class)))$$

9. get the names of student pairs who live at the same address:

$$\pi_{Student.name,S.name}(\sigma_{C_1 \wedge C_2}(Student \times \rho_S(Student)))$$

  - where $C_1 = (Student.addr = S.addr)$ and $C_2 = Student.sid > S.sid$
  - crossing a relation with itself is also known as **self-join**:
    - need to use the rename operator $\rho_{S(A)}$
    - renames table to S and optionally, specific attribute to A
  - the comparison check on `sid` prevents duplicates such as
    - `('John', 'John'), ('John', 'James'), ('James', 'John')`

10. get all students and instructor names:

$$\pi_{name}(Student) \cup \rho_{Person(name)}(\pi_{inst}(Class))$$

  - when using union, the schema must be equal between the relations
  - ie. the attribute names have to match up
  - no duplicate tuples in the result

11. get all the courses (department, number, section) that no one takes:

$$\pi_{dept,cnum,sec}(Class) - \pi_{dept,cnum,sec(Enroll)}$$

- easier to express this query using its complement
- using the set difference operator $R - S$

12. get instructor names who teach both CS and EE courses:

$$\pi_{inst}(\sigma_{dept=\text{'CS'}}(Class)) \cap \pi_{isnt}(\sigma_{dept=\text{'EE'}}(Class))$$

- when using intersection, the schema should again be the same
- $R \cap S = R - (R - S)$

13. get IDs of students who did not take any CS class:

$$\pi_{sid}(Student) - \pi_{sid}\sigma_{dept=\text{'CS'}}(Enroll)$$

- the core relational operators are:
  - $\sigma, \pi, \times, \cup, \rho, -$
    - set difference is the only non-monotonic operator, so it is core
  - while the other operators $\bowtie, \cap$ can be expressed with other operators

SQL

- **structured query language (SQL)** is the standard langauge for interacting with relational DBMS (RDBMS):
  - many versions of the SQL standard exists, SQL92 or SQL2 is the main standard
  - has multiple components:
    * the **data definition language (DDL)** includes schema definitions, constraints, etc.
      - eg. `CREATE, ALTER, DROP`
    * the **data manipulation language (DML)** includes queries, modifications, etc.
      - eg. `SELECT, INSERT, UPDATE`
    * other components for transactions and authorization

Data Definition Language (DDL)

- the DDL component of SQL allows for expressing schema definitions
- basic common SQL data types:
  - string:
    * `Char(n)` with padded fixed length n
      - will pad shorter values
    * `Varchar(n)` with variable length and max length n
  - number:
    * `Integer` 32-bit
    * `Decimal(d, f)` with d total digits and f precision
      - eg. the max value of `Decimal(5, 2)` is `999.99` , useful for financial values
    * `Real` 32-bit, `Double` 64-bit
  - datetime:
    * `Date` eg. `2010-01-15`
      - no timezone in SQL standard!
    * `Time` eg. `13:50:00`
    * `Timestamp` eg. `2010-01-15 13:50:00`
      - on MySQL, `Datetime` is preferred instead
- SQL table creation:
  - `CREATE TABLE` statement

  - one `PRIMARY KEY` per table:
    - `UNIQUE` is used for other keys (attributes may be null by SQL92)
    - by SQL standard, primary keys cannot be null
      - system will automatically mark those keys as not null, though the standard requires it to be explicitly written
  - `DEFAULT` sets the default value for an attribute
  - `DROP TABLE` statement for deleting a table

SQL table creation example:

```
CREATE TABLE Course(
  dept CHAR(2) NOT NULL DEFAULT 'CS',
  cnum INTEGER NOT NULL,
  sec  INTEGER NOT NULL,
  unit INT,
  instructor VARCHAR(50),
  title      VARCHAR(100),
  PRIMARY KEY(dept, cnum, sec),
  UNIQUE(dept, sec, title)
);
```

Loading Data

- there is no SQL standard for bulk data loading:
  - in Oracle and MySQL, `LOAD DATA INFILE <file> INTO TABLE <table>`
- options:
  - comma vs. tab separation for columns
    - `FIELDS TERMINATED BY ','`
  - columns enclosed with quotes
    - `OPTIONALLY ENCLOSED BY '"'`

Database Manipulation Language (DML)

- SQL gives a high-level description of what a user wants:
  - given a SQL query, DBMS figures out how best to execute it *automatically*
  - the core query operators are selection, projection, and join (SPJ)
- consider the following example queries using Table 1, Table 2, and Table 3

1. Get the titles and instructors of all CS classes:

```sql
SELECT title, instructor -- project in SELECT clause
FROM Class               -- specify relations in FROM clause
WHERE dept='CS';         -- filter through condition in WHERE clause
```

- thus, the SQL statement `SELECT A1...An FROM R1...Rm WHERE C` is roughly equivalent to:
  - $\pi_{A_1 \dots A_n}(\sigma_C(R_1 \times \cdots \times R_m))$
  - differences:
    - `SELECT` is projection rather than selection
    - SQL *does not* remove duplicates, uses *multiset* semantics instead

2. Get the names and GPAs of all students who take CS classes:

```sql
SELECT name, GPA AS grade -- renaming attributes, AS optional
FROM Student S, Enroll E  -- renaming operator for tuples, ie. tuple variables
WHERE dept='CS' AND S.sid=E.sid;

-- this returns duplicates if a student takes multiple classes!

SELECT DISTINCT name, GPA -- DISTINCT removes duplicates on final projected output
...
```

3. Get all student names and GPAs who live on Wilshire:

```sql
SELECT name, gpa
FROM Student
WHERE address = '%Wilshire%';
```

- string variables:
  - `%` for any length string, `_` for a single character
  - eg. `%Wilshire%` matches any string containing `Wilshire`
  - eg. `___%` matches any string with length ≥ 3
- other string functions include `UPPER(), LOWER(), CONCAT()`

4. Get all student and instructor names:

```sql
(SELECT name
FROM Student)
UNION -- set operator automatically takes care of duplicate instructors
(SELECT instructor -- can optionally rename
FROM Class);
```

- set operators:
  - eg. `UNION, INTERSECT, EXCEPT`
  - these operators *do* follow set semantics and remove duplicates
    - to keep duplicates, use `UNION ALL` etc.

  - schemas of input relations should be the same
    * in practice, compatible types are fine

5. Get all sids of students who do not take any cs class

```sql
(SELECT sid FROM Student)
EXCEPT
(SELECT sid FROM Enroll WHERE dept='CS');
```