CSE462/562: Database Systems (Fall 24)

Lecture 2: Relational Model & Query Languages

8/29/2024

University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

Last Update: 8/28/24, 1:00 PM

Data abstraction

- A revisit of the personal spending DB
- What if we want to
  - record the payment method
  - track budgets/bills
  - link entries to itemized receipts
- Or what if
  - the program/spreadsheet is slow after a while
  - you are managing the spending DB for many people (e.g., a company)

Date	Amount	Description
2/1	$20.21	Grocery
2/2	$10.54	Fast food
2/3	$39.22	Cell phone bill
…		
2/27	$33.00	Clothes

- Constant changes in data management
  - for efficiency or for new application usages
  - impractical to break existing applications for every change

Data abstraction

- Data abstraction
  - View level: what and how to present data to different applications/users

    Logical Data Independence: ability to change logical schema without changing the external views and upper-level applications

  - Logical level: what data are stored

    Physical Data Independence: ability to change physical data storage without changing the logical schema

  - Physical level: how data are stored

Data models

- Data models are conceptual tools for
  - describing and defining the data abstractions
  - linking user's view to the bits stored in DBMS
- Many data models exist
  - Relational model (aka structured data model)
  - Entity-Relationship Model
  - Semi-structured data model
  - Graph data model
  - …

We'll focus on relational model and Relational DataBase Management Systems (RDBMS) in this course:

It's the foundation of many other data models (including semi-structured data model, graph data model and etc.).

- The survey below gives a historical view of why relational models are successful
  - Joseph M. Hellerstein and Michael Stonebraker. What Goes Around Comes Around. Readings in Database Systems, 4th Edition (2005).
  - Keep it simple and stupid!

Relational model

- Example: student records database

student

sid	name	login	major	adm_year
100	Alice	alicer34	CS	2021
101	Bob	bob5	CE	2020
102	Charlie	charlie7	CS	2021
103	David	davel	CS	2020

enrollment

sid	semester	cno	grade
100	s22	562	2.0
102	s22	562	2.3
100	f21	560	3.7
101	s21	560	3.3
102	f21	560	4.0
103	s22	460	2.7
101	f21	560	3.3
103	f21	250	4.0

Relational model

- Relational database: a collection of named relations (aka tables)
- Relation: a set of records (aka tuples) – no duplicates
  - In reality: multi-set semantics are more prevalent – allow duplicates
- Record: a sequence of values
  - represents relationships among values

- Two concepts
  - Database Schema: names of the relations + names and types of the columns + constraints
    - e.g., student(sid: integer, name: string, login: string, major: string, adm_year: date)
    - each named column is also called an attribute or a field
  - Database instance: a snapshot of the data at a time point
    - e.g., the specific data in our student record database example

Relational model

Database schema

student(sid: integer, name: string, login: string, major: string, adm_year: date)

enrollment(sid: integer, semester: string, cno: integer, grade: float)

Relation (schema)

Relation (instance)

student

sid	name	login	major	adm_year
100	Alice	alicer34	CS	2021
101	Bob	bob5	CE	2020
102	Charlie	charlie7	CS	2021
103	David	davel	CS	2020

Record

Column

enrollment

sid	semester	cno	grade
100	s22	562	2.0
102	s22	562	2.3
100	f21	560	3.7
101	s21	560	3.3
102	f21	560	4.0
103	s22	460	2.7
101	f21	560	3.3
103	f21	250	4.0

Database instance

Integrity constraints

- Key constraints
  - Superkey: a set of columns that uniquely identify a record
    - e.g., $\{sid\}$ is a superkey of student relation; $\{sid, name\}$ is too;
    - e.g., $\{sid, semester, cno\}$ is a superkey of enrollment relation; but $\{sid, cno\}$ is not
    - Has nothing to do with specific instances
      - $\{sid, cno\}$ is not a superkey even if no one's ever taken a course twice
      - but it will be if the university policy prohibits retaking the same course

  - Candidate key: a superkey $K\ s.t. \nexists K' \subset K: K'$ is a superkey
    - e.g., $\{sid\}$ and $\{login\}$ are both candidate keys of student; $\{sid, login\}$ is not

  - (Primary) key: a chosen candidate key by the database designer
    - e.g., student(<u>sid</u>: integer, name: string, login: string, major: string, adm_year: date)

Integrity constraints

- Foreign-key constraints
  - from attributes A of referencing relation R to primary key A' of referenced relation R':
  - such that for any DB instance, any value of A must appear in A' of some tuple in R'

R' = student

sid	name	login	major	adm_year
100	Alice	alicer34	CS	2021
101	Bob	bob5	CE	2020
102	Charlie	charlie7	CS	2021
103	David	davel	CS	2020

R = enrollment

sid	semester	cno	grade
100	s22	562	2.0
102	s22	562	2.3
100	f21	560	3.7
101	s21	560	3.3
102	f21	560	4.0
103	s22	460	2.7
101	f21	560	3.3
103	f21	250	4.0

Integrity constraints

- Referential constraints
  - from attributes A of referencing relation R to *attributes* A' of referenced relation R'
  - such that for any DB instance, any value of A must appear in A' of some tuple in R'
  - Foreign-key constraints as a special case where A' is the primary key of R'

- Other general constraints

- These are less supported by DBMS due to efficiency reasons

Query Language

- Formal query languages
  - Relational algebra
    - Functional – describes how to query
  - Relational calculus
    - Declarative – describes what to query
  - No side effects! Does not include data definition, update, integrity checks, and etc.
  - Theoretical foundation of modern RDBMS; allows for query optimization

- Query language in practice: SQL (Structured Query Language)
  - Has its root in relational algebra and relational calculus
  - Includes many more beyond queries: imperative sublanguage, data definition, etc.

Structured Query Language (SQL)

- SQL stands for Structured Query Language
  - It's not only a "query language"
  - Consists of
    - Data Definition Language (DDL): define/modify schema, delete relations
      - Integrity checks: foreign-key constraints, general constraints, triggers
      - View definition, authorization specification, …
    - Data Manipulation Language (DML): query/insert/update/delete in a DB instance
    - Transaction control
    - Stored procedure, embedded SQL, SQL Procedural language, …

- The most widely used relational query language. Latest standard is SQL-2023
  - Each DBMS (e.g. MySQL/PostgreSQL) has some "unique" aspects
  - We'll only review the basics of SQL.

DDL - Create Table

- `CREATE TABLE` *`table_name`* `( {`
  `column_name data_type`
  `} [,…])`

- Data Types include:
  - `CHAR(n)` – fixed-length character string
  - `VARCHAR(n)` – variable-length character string with max length n
  - `SMALLINT, INTEGER, BIGINT` – signed 2/4/8-byte integers
    - No unsigned integer support in standard SQL, though they do exist in some SQL dialect
  - `NUMERIC[(p[,s])]` – exact numeric of selectable precision
  - `REAL, DOUBLE` – single/double floating point numbers
  - `DATE, TIME, TIMESTAMP, …`
  - `SERIAL` - unique ID for indexing and cross reference
  - …

DDL - Create Table w/ Column Constraints

- CREATE TABLE *table_name* ({

```
    column_name data_type
    [column_constraint [, ... ]]
} [,…])
```

- Column Constraints:

```
[CONSTRAINT constraint_name]  {
     DEFAULT default_expr |
     NOT NULL | NULL | UNIQUE | PRIMARY KEY |
     CHECK (boolean_expression) |
     REFERENCES reftable [(refcolumn)] [ON DELETE action]
       [ON UPDATE action ] }
```

can only reference the column's value

where *action* is one of:

```
NO ACTION, CASCADE, SET NULL, SET DEFAULT
```

DDL - Create Table w/ Table Constraints

- CREATE TABLE *table_name* ({

```
CREATE TABLE table_name ( {
    column_name data_type
    [column_constraint [, ... ]] |
     table_constraint
 } [,…])
```

- Table constraints:

```
[CONSTRAINT constraint_name]{
    UNIQUE (column_name [, ... ]) |
    PRIMARY KEY (column_name [, ... ]) |
    CHECK (boolean_expression) |
    FOREIGN KEY ( column_name [, ... ] )
        REFERENCES reftable [( refcolumn [, ... ])]
        [ON DELETE action] [ON UPDATE action]}
```

can only reference multiple table column's values

where *action* is one of:

```
    NO ACTION, CASCADE, SET NULL, SET DEFAULT
```

DDL -Create Table (Examples)

- ```
  CREATE TABLE student (
      sid       INTEGER PRIMARY KEY,
      name      VARCHAR(100) NOT NULL,
      login     VARCHAR(32) UNIQUE NOT NULL,
      major     VARCHAR(3),
      adm_year  DATE);
  ```

- ```
  CREATE TABLE enrollment(
      sid       INTEGER REFERENCES student ON DELETE
  SET NULL
      semester  VARCHAR(3),
      cno       INTEGER,
      grade     NUMERIC(2, 1)
      PRIMARY KEY (sid, semester, cno));
  ```

Other DDL statements

- `DROP TABLE` *table_name*;
- `ALTER TABLE` *table_name action* [,…];
  where *action* is one of
  `ADD` *column_name data_type* [*column_constraints* [,…]]
  `DROP` *column_name data_type*
  `ALTER` *coumn_name* …
  `ADD` *table_constraint*
  `DROP CONSTRAINT` *constraint_name*
  …

SQL DML

- `SELECT` statement
- `INSERT` statement
- `DELETE` statement
- `UPDATE` statement

SQL DML Semantics

- SQL uses multi-set semantics (aka bag semantics) by default
  - meaning multiple tuples in the same table can have exactly the same values

  - SQL also supports operators that can't be expressed in the standard relational algebra
    - sorting
    - aggregation

  - Standard Relational Algebra uses set semantics
    - review in Lectures 5 & 6

Single-Table Query

- Single-table queries are straight-forward.

- To find all students admitted in 2021, we can write

```
SELECT *
FROM students S
WHERE S.adm_year = 2021;
```

student

sid	name	login	major	adm_year
100	Alice	alicer34	CS	2021
101	Bob	bob5	CE	2020
102	Charlie	charlie7	CS	2021
103	David	davel	CS	2020

→

result

sid	name	login	major	adm_year
100	Alice	alicer34	CS	2021
102	Charlie	charlie7	CS	2021

Multi-Table Query

- We can express a join as follows

```
SELECT S.name, E.grade
FROM student S, enrollment E
WHERE S.sid=E.sid AND E.cno=562;
```

or

```
SELECT S.name, E.grade
FROM student S JOIN enrollment E
  ON S.sid = E.sid
WHERE E.cno = 562;
```

student

sid	name	login	major	adm_year
100	Alice	alicer34	CS	2021
101	Bob	bob5	CE	2020
102	Charlie	charlie7	CS	2021
103	David	davel	CS	2020

enrollment

sid	semester	cno	grade
100	s22	562	2.0
102	s22	562	2.3
100	f21	560	3.7
101	s21	560	3.3
102	f21	560	4.0
103	s22	460	2.7
101	f21	560	3.3
103	f21	250	4.0

Result

name	grade
Alice	2.0
Charlie	2.3

SQL Query Syntax

```
SELECT  [DISTINCT] target-list
FROM     relation-list
[WHERE predicate]
```

- `SELECT` and `FROM` clauses are mandatory
- `WHERE` clause is optional
- *`relation-list`*: a list of relation
  - each possibly with a table alias (aka correlation name)
- *`target-list`*: a list of expressions that may reference columns in the relation list
  - "`*`" to denote all the columns in the relation list
  - each may be renamed with `AS` clause (e.g., `S.name as student_name`)
  - `DISTINCT`: an optional keyword to deduplicate the result
- *`predicate`*: boolean expressions over the columns in the relation list, may contain
  - comparisons such as <, >, <=, >=, =, <>, LIKE
  - AND/OR/NOT
  - nested query
  - …

SQL supports string matching operator LIKE:
`_' stands for any one character and `%' stands for 0 or more arbitrary characters.
e.g., dname LIKE '%Engineering' will match all departments that ends with "Engineering" in its name

ORDER BY Clause

- Optional ORDER BY clause sorts the final results before presenting them to the end user
  - `expr` is some expression of the columns in the relation list
  - Sort lexicographically
  - May also use positional notation (1, 2, 3, …)
    - denotes expr in target list
  - Default is ascending order `ASC`
    - Specify `DESC` for descending order

```
SELECT   [DISTINCT] target-list
FROM      relation-list
[WHERE predicate]
[ORDER BY] expr [ASC|DESC] [,…]
```

- Examples
  - `ORDER BY E.grade DESC` -- sort by descending order in grade
  - `ORDER BY 2 DESC` -- same as above
  - `ORDER BY E.grade DESC, S.name`
    - sort by descending grade first; then for equal values of grade, sort by name in ascending order
  - `ORDER BY 2 DESC, 1 ASC` -- same as above

Other DML Statements

```
INSERT [INTO] table_name [(column_list)] VALUES ( value_list);
INSERT [INTO] table_name [(column_list)] <select statement>;

DELETE [FROM] table_name [WHERE qualification];
UPDATE SET column_name = expr [,…] [WHERE qualification];
```