CS143: Basic SQL Query

Book Chapters

(5th) Chapter 3.1, 3.3-4, 3.7
(6th) Chapter 3.1, 3.3-5, 3.8
(7th) Chapter 3.1, 3.3-5, 3.8

Things to Learn

- Basic SELECT query
- SQL set operator
- Subqueries

SQL

- Structured Query Language
- The standard language for all commericial RDBMS
- SQL has many aspects
  - DDL: schema definition, constraints, index, ...
  - DML: query, update, ...
  - triggers, transaction, authorization, ...
- In this lecture, we cover the DML aspect of SQL
  - How to query and modify exsiting databases
- SQL and DBMS
  - SQL is high-level description of user's query
    - No concrete procedure for query execution is given
  - The beauty and success of DBMS
    - The system understands the query and find the best way possible to execute it *automatically*

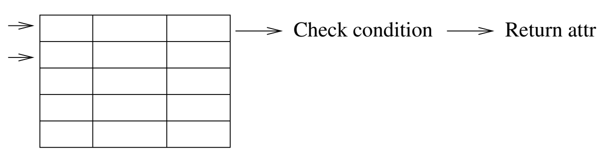
Example to Use in the Class

- School information
  - Student(sid, name, age, GPA, address, ...)
  - Class(dept, cnum, sec, unit, title, instructor, ...)
  - Enroll(sid, dept, cnum, sec)

Basic SELECT statement

- **Query 1:** Find the titles and instructors of all CS courses

- **Semantics**
  - Interpret and write FROM → WHERE → SELECT
    - FROM: the list of tables to look up
    - WHERE: conditions to meet
    - SELECT: the attributes to return
  - *Conceptual* execution (table cursor diagram)



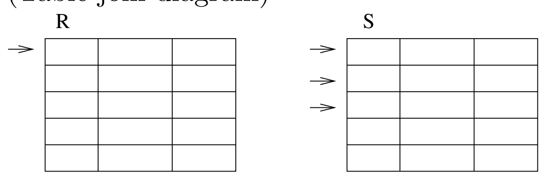
General SQL statement

- `SELECT A1, ..., An`
  `FROM R1, ..., Rm`
  `WHERE C`
  $\equiv \pi_{A_1,\ldots,A_n}(\sigma_C(R_1 \times \cdots \times R_m))$
- `SELECT *`: all attributes
- `SELECT` is "projection" not "selection": can be confusing
- SQL does not remove duplicates: Major difference between SQL and relational algebra
  - More examples will follow

SQL join

- **Query 2:** Find the names and GPAs of all students taking CS classes

  - Conceptually `WHERE R, S`
    (Table join diagram)



  - For every pair of tuples from R and S, we check condition and produce output

  Notes:

  - S, E: tutple variable
    - renaming operator
    - We can consider that S and E are variables that bind to every pair of tuples
  - Attribues can also be renamed
    - `GPA (AS) grade`
  - `DISTINCT`: remove duplicates in the results

WHERE conditions

- **Query 3:** All student names and GPAs who live on Wilshire

  - `%`: any length (0–∞) string
    `_`: one character
    `'%Wilshire%'`: Any string containing Wilshire

  Q: What does `'___%'` mean?

- Other useful string functions: `UPPER(), LOWER(), CONCAT(), ...`

Set operators

- $\cap$: `INTERSECT`, $\cup$: `UNION`, $-$: `EXCEPT`
- Can be applied to the result of SELECT statements or to relations
- **Query 4:** All names of students and instructors

- **Important points to note**
  - Set operators should have the same schema for operands
    * In practice, it is okay to have just compatible types
  - Set operators follow *set* semantics and remove duplicates
    * Set semantics is well understood for set operations. Not many people know bag semantics.
    * Efficiency
  - To keep duplicates, use `UNION ALL, INSERSECT ALL, EXCEPT ALL`
- **Query 5:** Find ids of all students who are not taking any CS courses.

- MySQL support:
  - Standard MySQL does not support `INTERSECT` or `EXCEPT`.
  - MariaDB v10.3 introduced supports for `INTERSECT` and `EXCEPT`.

Subqueries

- `SELECT` statement may appear in `WHERE` clause
  - Treated the same as regular relations
  - If the result is one-attribute one-tuple relation, the result can be used like a 'value'

Scalar-value subqueries

- **Query 6:** Find the student ids who live at the same addr as the student with id 301

- **Q:** Can we rewrite it without subquery?

- **Notes:**
  - There is a whole theory about whether/how to rewite a subquery to non-subquery SQL
  - The basic result is we can rewrite subqueries as long as we do not have negation.
  - With negation, we need `EXCEPT`
  - One of the reasons why relational model has been so successful
    * Because it is easy to understand and model, we can design and prove elegant theorems.
    * Many efficient and provable algorithms.

Set membership (IN, NOT IN)

- **Query 7:** Find all student names who take CS classes.

  Idea: Find the set of sids that take CS classes first. Then check whether any student's id belong to that set or not.

  - `IN` is a set membership operator
    * `(a IN R)` is `TRUE` if `a` appears in `R`

Q: Can we write the same query without subqueries?

Q: Are the above two queries equivalent?

Q: Why we care about duplicates so much?

- **Query 8:** Find the names of students who take no CS classes

  Q: Can we rewrite it without subqueries?

Set comparison operator (> ALL, < SOME, . . .)

- **Query 9:** Find the ids of students whose GPA is greater than all students of age 18 or less

  – `ALL` is the universal quantifier ∀

- **Query 10:** Find the IDs of students whose GPA is better than at least one other student of age ≤ 18

  - `SOME` is the existential quantifier $\exists$

  Other Set comparison operators: `> ALL, <= SOME, = SOME, ...`, etc.

  - `(<> ALL)` $\equiv$ `(NOT IN)`, `(= SOME)` $\equiv$ `IN`

Correlated subqueries

- **Query 11:** Find the names of the students who take any class

  - **EXISTS:** `WHERE EXISTS(SELECT ... FROM ... WHERE)`
    * True if `SELECT .. FROM .. WHERE` returns at least one tuple
  - **Correlated subquery interpretation:**
    * Outer query looks at one tuple at a time and binds the tuple to `S`
    * For each `S`, we execute the inner query and check the condition
    * This is just interpretation. ***DBMS executes it more efficiently but get the same result*** (but not necessarily MySQL).

Subqueries in FROM clause

- Can be used like a regular relation

- **Example:**
```
SELECT name
FROM (SELECT name, age FROM Student) S
WHERE age > 17
```
  - A subquery inside FROM **MUST** be renamed
  - Student names with age > 17

Common Table Expression

- Introduced in SQL1999

- Similar to subqueries in FROM, but makes it easier to reuse query results

- Syntax:
```
WITH alias AS (query)
SELECT ...
```

- **Example:**
```
WITH S AS (SELECT name, age FROM Student)
SELECT name FROM S WHERE age > 17
```

- **Q:** Do subqueries make SQL more expressive than relational algebra?