CS143: SQL Query (2)

Book Chapters

(4th) Chapter 4.1-6, 4.8-10, 3.3.4
(5th) Chapter 3.1-8, 3.10-11, 4.7
(6th) Chapter 3.1-9, 4.1, 4.3, 5.4-5
(7th) Chapter 3.6-7, 3.9, 4.1, 5.4-5

Things to Learn

- Subquery
- Aggregate

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 1:** 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 2:** 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 3:** Find the names of students who take no CS classes

  Q: Can we rewrite it without subqueries?

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

- **Query 4:** Find the ids of students whose GPA is greater than all students of age 18 or less

  - `ALL` is the universial quantifier $\forall$

- **Query 5:** 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 6:** 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

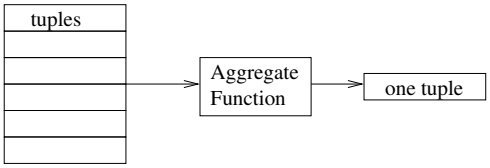
Common Table Expression

- 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?

Aggregates

- The operators so far check the condition "tuple-by-tuple"
- They never "summarize" multiple tuples into one.
  For example, 'SUM', 'AVG' of GPA is not possible.
- Aggregate function (aggregate diagram)



- **Query 7:** Find the average GPA

- Common aggregate functions: `SUM, AVG, COUNT, MIN, MAX` on single attribute or `COUNT(*)`.

Problems of Duplicates

- **Query 8:** The number of students taking CS classes

- **Query 9:** The average GPA of the students taking CS classes

GROUP BY clause

- Sometimes, we want to get separate statistics for each group of tuples

  Example:

Age	AVG(GPA)
17	3.7
19	2.1
20	3.1

  But `AVG()` takes average over *all* tuples.

- **Query 10:** Find the average GPA for each age group

  Q: Is the following query meaningful?

```
SELECT sid, age, AVG(GPA)
FROM Student
GROUP BY age
```

  – SELECT can have only attributes that have a single value in each group or *aggregates*

- **Query 11:** Find the number of classes each student is taking

Q: What about the students who take no classes?

Comments: We will learn about outer join that can address this issue later.

HAVING clause

- **Query 12:** Find students who take two or more classes

  – Conditions on aggregates should appear in the HAVING clause.

Q: Can we rewrite the query without HAVING clause?

  – In general, we can rewrite a query not to have a HAVING clause.