SQL: The Query Language Part 1

R&G - Chapter 5

Announcement

- **(Tentative) date of midterm exam: Nov 1**
- **SQL lab sessions (Nov 8 & 10, potentially Nov 15 too)**
- **Final exam date:**
  - Survey result: 69 participants (81% participation rate)
    - The last week in class (Dec 13): 61 votes
    - Stevens exam date (Dec 21): 8 votes
    - Final voting result: Dec 13

Databases: the continuing saga

- When last we left databases
  - We knew how to conceptually model them in ER diagrams
  - We knew how to logically model them in the relational model
  - We could formally specify queries
- Now: how do most people write queries?

SQL!

Relational Query Languages

- **A major strength of the relational model: supports simple, powerful *querying*.**

- **Two sublanguages:**
  - DDL – Data Definition Language
    - define and modify schema
  - DML – Data Manipulation Language
    - Queries can be written intuitively.

DML – Querying Databases

- A simple SQL query has the form:

  SELECT $A_1, A_2, \ldots, A_n$
  FROM $r_1, r_2, \ldots, r_m$
  WHERE P

  - A_i represents an attribute
  - r_i represents a relation
  - P is a predicate

- This query is equivalent to the relational algebra expression:

Select

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$

Where

From

Roadmap of lecture

SELECT $A_1, A_2, \ldots, A_n$
FROM $r_1, r_2, \ldots, r_m$
WHERE P

- *SELECT Clause*
- *WHERE Clause*
- *FROM Clause*

SELECT clause

- Equivalent to projection (π) operator
  - *E.g.,* $\pi_{\text{sid, sname}}$ Students
  - *<u>In SQL:</u>*

```
SELECT   sid, sname
FROM     Students;
```

- Can use "*" to get all attributes

```
SELECT *
FROM     Students;
```

- By default, duplicates are preserved
- Can include arithmetic expressions in SELECT
- e.g., SELECT acct_no, balance*1.05
  FROM account

Selecting Distinct Values

- In SQL SELECT, the default is that duplicates are *not* eliminated! (Result is called a "multiset")
- Can write SELECT DISTINCT to eliminate duplicates

stud#	*name*	*address*
100	Fred	Aberdeen
200	Dave	Dundee
300	Bob	Aberdeen

```
SELECT DISTINCT address

FROM Student;
```

address
Aberdeen
Dundee

Roadmap of lecture

SELECT $A_1, A_2, \ldots, A_n$
FROM $r_1, r_2, \ldots, r_m$
WHERE P

- *SELECT Clause*
- *WHERE Clause*
- *FROM Clause*

WHERE clause

- Equivalent to selection (σ) operator
  - E.g., $\sigma_{\text{course = 'Computing'}}$ Students
  - *In SQL:*

```
SELECT *
FROM Students
WHERE course = 'Computing';
```

Predicates in WHERE clause

- WHERE predicate can be:
  - Simple format:
    - *Attribute* op *Constant*, or *Attribute1* op *Attribute2*
    - Op: *<, <=, =, <>, >=, >*
      - Note: distinguish from the operators for relational algebra
    - Attribute names should come from the relation(s) used in the FROM clause
  - Complex format:
    - Use AND, OR, NOT, BETWEEN, IN, LIKE
  - Allow arithmetic operations (e.g., WHERE rating*2>200)
  - Allow string operations (e.g., "||" for concatenation).

Connectives in WHERE Predicate

- **In Relational Algebra, we use:**
  - ∧ (and), ∨ (or), ¬ (not)

$$\pi_{bname}(\sigma_{color='red' \vee color='green'}(Boats))$$

- **In SQL, we use:**
  - AND, OR, NOT

```
SELECT Bname
FROM Boats
WHERE color='red' OR color='green';
```

Using Quotes in Selection Conditions

- SQL uses single quotes around text values (some database systems also accept double quotes).
- Numeric values should NOT be enclosed in quotes.

For text values:

This is correct:

```
SELECT * FROM Persons WHERE FirstName='Tove'
```

This is wrong:

```
SELECT * FROM Persons WHERE FirstName=Tove
```

BETWEEN Operator in WHERE Clause

```
SELECT *
FROM Book
WHERE catno BETWEEN 200 AND 400;

SELECT *
FROM Product
WHERE prod_desc BETWEEN 'C' AND 'S';

SELECT *
FROM Book
WHERE catno NOT BETWEEN 200 AND 400;
```

IN Operator in WHERE Clause

```
SELECT Name
FROM Member
WHERE memno IN (100, 200, 300, 400);

SELECT Name
FROM Member
WHERE memno NOT IN (100, 200, 300, 400);
```

IS/NOT Operators in WHERE Clause

- Most useful for missing values in database

```
SELECT Catno
FROM Loan
WHERE Date-Returned IS NULL;

SELECT Catno
FROM Loan
WHERE Date-Returned IS NOT NULL;
```

LIKE Operator in WHERE Clause

- "LIKE" is used for string approximate matching.
  - '_' stands for any character;
  - '%' stands for any string (0 or more characters)

```
SELECT Name
FROM Member
WHERE address LIKE 'T%';
```

```
SELECT Name
FROM Member
WHERE Name NOT LIKE '_ES%';
```

An Example of String Operations

```
SELECT 2*S.Salary AS DoubleSalary,
TripleSalary =3*S.Salary
FROM MovieStar S
WHERE S.Name LIKE 'B_%T';
```

DoubleSalary	TripleSalary
200,000	300,000
400,000	600,000

Output

- AS and = are two ways to name new attributes defined in the output.
- Question:
  - How to interpret the WHERE condition in this query?

Exercise of String Operations

- **Schema**
  - Boats (bid, bname, color)

- Question: write the WHERE clause of SQL query that finds the boats whose name starts with “M” and has at least 3 characters.

Roadmap of lecture

SELECT $A_1, A_2, \ldots, A_n$
FROM $r_1, r_2, \ldots, r_m$
WHERE P

- *SELECT Clause*
- *WHERE Clause*
- *FROM Clause*

The FROM clause

- Cross-product (X) or Join (⋈) of tables T1, …Tn
  - Has multiple tables T1, …. Tn in the FROM clause
- Distinguish attributes of the same name by "<relation>.<attribute>"
  - E.g., Sailor.name, boat.name

Cross Product VS Natural Join

Sid	Bid	day
22	101	10/10/96
58	103	11/12/96

R1

Sid	Sname	Rating	Age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.5

S1

```
SELECT  R1.sid, sname, ratings, age, bid, day
FROM R1, S1
WHERE R1.sid = S1.sid;
```

- WITHOUT WHERE clause: cross-product (X);
- WITH WHERE clause that checks equivalence on ALL common attributes: natural-join ($\bowtie$)
  - The order of tables does NOT matter (the non-joinable tables still can be put side-by-side in FROM class)

Another Way to Specify Natural Join

- Syntax

```
SELECT  A1, …An
FROM R NATURAL JOIN S;
```

- More variants of NATURAL JOIN operator will be discussed in later lectures.
- The order of tables matters for NATUAL JOIN
  - Only join-able tables are put at both sides of NATURAL JOIN
- Can have join of >2 tables
  - For example: `R NATURAL JOIN S NATURAL JOIN T`

Condition Join Example

Sid	Bid	day
22	101	10/10/96
58	103	11/12/96

R1

Sid	Sname	Rating	Age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.5

S1

$$\pi_{s1.sid,sname,ratings,age,bid,day}(S1 \bowtie_{S1.sid < R1.sid} R1)$$

```
SELECT  S1.sid, sname, ratings,age, bid, day
FROM R1, S1
WHERE S1.sid < R1.sid;
```

Range Variables

- Can associate "range variables" with the tables in the FROM clause.
  - saves writing, makes queries easier to understand

```
SELECT sname
FROM Sailors, Reserves
WHERE Sailors.sid=Reserves.sid AND bid=103
```

Can be rewritten using range variables as:

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid=R.sid AND bid=103
```

- Needed when ambiguity could arise.
  - For example, if same table used multiple times in same FROM (called a "self-join")

Example of Range Variables in Self-Join

```
SELECT  x.sname
FROM Sailors x, Sailors y
WHERE  x.age > y.age
```

Question: what does this query return?

Exercise: Translate Relational Algebra Expression to SQL

Schema:

- Students (<u>sid</u>, <u>cid</u>, sname, address)
- Courses (<u>cid</u>, cname)

R1= Students $\bowtie$ ($\sigma_{\text{cname='CS442'}}$ Courses)

R2= $\sigma_{\text{address='Hoboken'}}$ R1

R3= $\pi_{\text{Students.sname, Course.cname}}$ R2

Translate the relational algebra expressions into SQL statement.