SQL: The Query Language Part 1

R&G - Chapter 5

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!

Coming up in SQL…

- Data Definition Language (reminder)
- Basic Structure
- Set Operations
- Aggregate Functions
- Null Values
- Nested Subqueries
- Modification of the Database
- Views
- Integrity Constraints
- JDBC

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.

DDL – Create Table (Revisited)

- **SQL syntax**
  - `CREATE TABLE <name> ( <field> <domain>, … )`
- **Example: creates the Students relation.**

Schema

Students(*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: real)

```
CREATE TABLE Students
       (sid CHAR(20),
        name CHAR(20),
        login CHAR(10),
        age INTEGER,
        gpa FLOAT)
```

Note: the type (domain) of each field is specified, and enforced by the DBMS

Create Table (with Integrity Constrations) - Revisited

- A SQL relation is defined using the **create table** command:
  - create table *r* (*A*1 *D*1, *A*2 *D*2, ..., *An Dn*,
    (integrity-constraint$_1$),
    ...,
    (integrity-constraint$_k$))
- *Integrity constraints can be:*
  - *primary and candidate keys*
  - *foreign keys*

```
CREATE TABLE films (
    code            CHAR(5) PRIMARY KEY,
    title           VARCHAR(40),
    did             DECIMAL(3),
    date_prod       DATE,
    kind            VARCHAR(10),
    FOREIGN KEY did REFERENCES dist
    ON DELETE NO ACTION
);
```

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:**

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

Select

Where

From

The SELECT clause

- **Equivalent to projection (π) operator**
  - *E.g.,* π stud#, name Students
  - *In SQL:*

```
SELECT stud#, name
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 bname, 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

Student

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

SELECT DISTINCT address

FROM Student;

address
Aberdeen
Dundee

The WHERE clause

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

```
Select *
From Students
Where course = 'Computing';
```

The WHERE clause

- **WHERE predicate can be:**
  - Simple:
    - *attribute* op *constant*, or
    - *Attribute1* op *attribute2*
    - Op: *<, <=, =, ≠, >=, >*
    - attribute names should come from the relation(s) used in the FROM
  - Complex:
    - using 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

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

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

- **SQL:**
  - 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 (most database systems will 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 single character and
  - '%' stands for 0 or more arbitrary 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';
```

- AS and = are two ways to name fields in result.
- What does this query return?

The FROM clause

- **Cross-product (X) or Join ⋈ of tables T1, …Tn**
  - multiple tables T1, …. Tn, in the FROM clause

Cross Product 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	35.0

S1

Select *
FROM R1, S1;

R1 X S1 =

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

Natural 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

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

R1 ⋈ S1 =

Sid	Sname	Rating	Age	Bid	day
22	Dustin	7	45.0	101	10/10/96
58	Rusty	10	35.0	103	11/12/96

- Note
  - WITHOUT WHERE clause: cross-product;
  - WITH WHERE clause (on common attributes): natural-join

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.**

Conditional 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

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

S1.sid	Sname	Rating	Age	Bid	day
22	Dustin	7	45.0	103	11/12/96
31	Lubber	8	55.5	103	11/12/96

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

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?

Combinations of Operators

Schema:

- Students (sid, cid, sname, address)
- Courses (cid, cname)

R1= Students $\bowtie$ (σ cname='CS442' Courses)

R2= σ address="Hoboken" R1

R3= π Students.sname, Course.cname R2

Translate the relational algebra expressions into SQL statement