SQL: The Query Language Part 1

R &G - Chapter 5

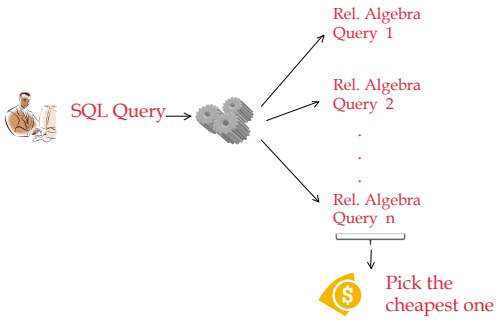
The important thing is not to stop questioning.

Albert Einstein

Review

- Relational Algebra (Operational Semantics)
  - Given a query, how to mix and match the relational algebra operators to answer it
  - Used for query optimization
- Relational Calculus (Declarative Semantics)
  - Given a query, what do I want my answer set to include?
- Algebra and safe calculus are simple and powerful models for query languages for relational model
  - Have same expressive power
- SQL can express every query that is expressible in relational algebra/calculus. (and more)

Query Optimization



Relational Query Languages

- **Two sublanguages:**
  - DDL – Data Definition Language
    - Define and modify schema (at all 3 levels)
  - DML – Data Manipulation Language
    - Queries can be written intuitively.
- **DBMS is responsible for efficient evaluation.**
  - The key: precise semantics for relational queries.
  - Optimizer can re-order operations, without affecting query answer.
  - Choices driven by cost model: how many disk accesses; how much CPU?

The SQL Query Language

- The most widely used relational query language.
- Standardized

  (although most systems add their own "special sauce" -- including PostgreSQL)
- We will study **SQL92** -- a basic subset

Example Database

Sailors

sid	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27

Boats

bid	bname	color
101	Nina	red
102	Pinta	blue
103	Santa Maria	red

Reserves

sid	bid	day
1	102	9/12
2	102	9/13

Review: SQL DDL

```
CREATE TABLE Sailors (
  sid INTEGER,
  sname CHAR(20),
  rating INTEGER,
  age REAL,
  PRIMARY KEY sid);

CREATE TABLE Boats (
  bid INTEGER,
  bname CHAR (20),
  color CHAR(10)
  PRIMARY KEY bid);

CREATE TABLE Reserves (
  sid INTEGER,
  bid INTEGER,
  day DATE,
  PRIMARY KEY (sid, bid, date),
  FOREIGN KEY sid REFERENCES Sailors,
  FOREIGN KEY bid REFERENCES Boats);
```

sid	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27

bid	bname	color
101	Nina	red
102	Pinta	blue
103	Santa Maria	red

sid	bid	day
1	102	9/12
2	102	9/13

The SQL DML

Sailors

sid	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27

- Find all 18-year-old sailors:

```
SELECT *
FROM  Sailors S
WHERE S.age=18
```

```
SELECT *
FROM  Sailors
WHERE age=18
```

- To find just names and ratings, replace the first line:

```
SELECT S.sname, S.rating
```

Querying Multiple Relations

```
SELECT S.sname
FROM   Sailors S, Reserves R
WHERE  S.sid=R.sid AND R.bid=102
```

Sailors

sid	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27

Reserves

sid	bid	day
1	102	9/12
2	102	9/13

Basic SQL Query

DISTINCT: optional keyword indicating answer should not contain duplicates. In SQL, default is that duplicates are *not* eliminated! (Result is called a "multiset")

target-list : A list of attributes of tables in *relation-list*

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

qualification : Comparisons combined using AND, OR and NOT. Comparisons are Attr *op* const or Attr1 *op* Attr2, where *op* is one of =,<,>,≠, etc.

relation-list : A list of relation names, possibly with a *range-variable* after each name

Query Semantics

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

1. FROM : compute *cross product* of tables.
2. WHERE : Check conditions, discard tuples that fail.
3. SELECT : Delete unwanted fields.
4. DISTINCT *(optional)* : eliminate duplicate rows.

Note: Probably the least efficient way to compute a query!

- *Query optimizer* will find more efficient ways to get the *same answer*.

Find sailors who've reserved at least one boat

```
SELECT S.sid
FROM   Sailors S, Reserves R
WHERE  S.sid=R.sid
```

- Would adding DISTINCT to this query make a difference?
- What is the effect of replacing *S.sid* by *S.sname* in the SELECT clause?
  - Would adding DISTINCT to this variant of the query make a difference?

About Range Variables

- Needed when ambiguity could arise.
  - e.g., same table used multiple times in FROM ("self-join")

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

Sailors

sid	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27

Arithmetic Expressions

```
SELECT S.age, S.age-5 AS age1, 2*S.age AS age2
FROM   Sailors S
WHERE  S.sname = 'dustin'
```

```
SELECT S1.sname AS name1, S2.sname AS name2
FROM   Sailors S1, Sailors S2
WHERE  2*S1.rating = S2.rating - 1
```

String Comparisons

```
SELECT S.sname
FROM   Sailors S
WHERE  S.sname LIKE 'B_%B'
```

`_' stands for any one character and `%' stands for 0 or more arbitrary characters.

Yes, every other language in the world uses Perl-like regular expressions. In fact, PostgreSQL supports this with substring(), but this is not standard or portable.

Intermission

Why are Databases useful?

Here's why

Find sid's of sailors who've reserved a red **or** a green boat

```
SELECT R.sid
FROM   Boats B, Reserves R
WHERE  R.bid=B.bid AND
       (B.color='red' OR
        B.color='green')
```

... or:

```
SELECT R.sid
FROM   Boats B, Reserves R
WHERE  R.bid=B.bid AND
       B.color='red'
UNION
SELECT R.sid
FROM   Boats B, Reserves R
WHERE  R.bid=B.bid AND B.color='green'
```

Find sid's of sailors who've reserved a red **and** a green boat

~~SELECT R.sid~~
~~FROM Boats B,Reserves R~~
~~WHERE R.bid=B.bid AND~~
~~(B.color='red' AND B.color='green')~~

Find sid's of sailors who've reserved a red **and** a green boat

```
SELECT S.sid
FROM   Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid
       AND R.bid=B.bid
       AND B.color='red'
INTERSECT
SELECT S.sid
FROM   Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid
       AND R.bid=B.bid
       AND B.color='green'
```

Find sid's of sailors who've reserved a red **and** a green boat

- Could use a self-join:

```
SELECT R1.sid
FROM   Boats B1, Reserves R1,
       Boats B2, Reserves R2
WHERE R1.sid=R2.sid
      AND R1.bid=B1.bid
      AND R2.bid=B2.bid
      AND (B1.color='red' AND B2.color='green')
```

Find sid's of sailors who have not reserved a boat

```
SELECT S.sid
FROM   Sailors S

EXCEPT

SELECT S.sid
FROM   Sailors S, Reserves R
WHERE  S.sid=R.sid
```

Nested Queries: IN

Names of sailors who've reserved boat #103:

```
SELECT S.sname
FROM   Sailors S
WHERE  S.sid IN
  (SELECT  R.sid
   FROM    Reserves R
   WHERE  R.bid=103)
```

Nested Queries: NOT IN

*Names of sailors who've **not** reserved boat #103:*

```
SELECT  S.sname
FROM    Sailors S
WHERE   S.sid NOT IN
  (SELECT  R.sid
   FROM    Reserves R
      WHERE  R.bid=103)
```

Nested Queries with Correlation

Names of sailors who've reserved boat #103:

```
SELECT  S.sname
FROM    Sailors S
WHERE EXISTS
    (SELECT  *
     FROM  Reserves R
     WHERE R.bid=103 AND S.sid=R.sid)
```

- **Subquery may need to be recomputed for each Sailors tuple.**
  - Think of subquery as a function call that runs a query!
- **Also: NOT EXISTS.**

More on Set-Comparison Operators

- we've seen: **IN, EXISTS**
- can also have: **NOT IN, NOT EXISTS**
- other forms: ***op* ANY, *op* ALL**

- **Find sailors whose rating is greater than that of some sailor called Horatio:**

```
SELECT *
FROM   Sailors S
WHERE  S.rating > ANY
   (SELECT  S2.rating
    FROM  Sailors S2
    WHERE S2.sname='Horatio')
```

A Tough One

Find sailors who've reserved all boats.

```
SELECT  S.sname
FROM  Sailors S
WHERE  NOT EXISTS (SELECT  B.bid
                   FROM  Boats B
                   WHERE  NOT EXISTS (SELECT  R.bid
                                      FROM  Reserves R
                                      WHERE  R.bid=B.bid
                                        AND R.sid=S.sid))
```

Sailors S such that ...

there is no boat B without ...

a Reserves tuple showing S reserved B

Summary

- Relational model has well-defined query semantics
- SQL provides functionality close to basic relational model
  (some differences in duplicate handling, null values, set operators, …)
- Typically, many ways to write a query
  - DBMS figures out a fast way to execute a query, regardless of how it is written.