SQL: Queries, Programming, Triggers

Chapter 5, Cow Book

or

See http://sqlzoo.net/

SQL Language

- DDL: Data definition language
- DML: Data manipulation language
- Embedded and Dynamic SQL
- Triggers
- Security
- Transaction Management
- Remote Database access

Basic SQL Query

Optional

Attributes from input relations

SELECT [DISTINCT] ***target-list***

List of relations

FROM ***relation-list***

WHERE ***qualification***

Attr1 **op** Attr2
OPS: <, >, =, <=, >=, <>
Combine using AND, OR, NOT

- Semantics/Conceptual evaluation strategy:
  - Compute the cross-product of *relation-list*.
  - Discard resulting tuples if they fail *qualifications*.
  - Delete attributes that are not in *target-list*.
  - If DISTINCT is specified, eliminate duplicate rows.
- Not an efficient evaluation plan! (Optimzier picks efficient plans)

Example of Conceptual Evaluation

SELECT S.name, A.hours FROM Senators S, Attendance A
WHERE S.ssn = A.ssn and A.date = '24-Sept-2016'

ssn	name	email	age	income
11-111	Bob	bob@ca.gov	51	100.1
22-222	Jane	jane@mi.gov	54	130.1
33-333	Jane	jane@wi.gov	51	99.8

ssn	date	hours
11-111	12-Aug-2016	1.1
33-333	24-Sept-2016	4.1

(ssn)	name	email	age	income	(ssn)	date	hours
11-111	Bob	bob@ca.gov	51	100.1	11-111	12-Aug-2016	1.1
22-222	Jane	jane@mi.gov	54	130.1	11-111	12-Aug-2016	1.1
33-333	Jane	jane@wi.gov	51	99.8	11-111	12-Aug-2016	1.1
11-111	Bob	bob@ca.gov	51	100.1	33-333	24-Sept-2016	4.1
22-222	Jane	jane@mi.gov	54	130.1	33-333	24-Sept-2016	4.1
33-333	**Jane**	jane@wi.gov	51	99.8	33-333	24-Sept-2016	**4.1**

Find senators who attended the '24-Sept-2016' session

```
SELECT S.name
FROM Senators S, Attendance A
WHERE S.ssn = A.ssn and date = '24-Sept-2016'
```

- Add DISTINCT to this query. Effect?
- Replace S.name by S.ssn.
  Effect of adding DISTINCT to this query

RA: $\pi_{name}\ (\sigma_{date='24\text{-}Sept\text{-}2016'}(Senators \triangleright\!\triangleleft Attendance))$

- Equivalent SQL?
- What is the schema of Senators $\bowtie$ Attendance?

A Note on Range Variables

- Needed only if the same relation appears twice in the FROM clause.

```
SELECT S.name, A.hours
FROM Senators S, Attendance A
WHERE S.ssn = A.ssn and    date = '24-Sept-2016'
```

OR

```
SELECT Senators.name, Attendance.hours
FROM Senators , Attendance
WHERE Senators.ssn = Attendance.ssn and
          Attendance.date = '24-Sept-2016'
```

It is good style, however, to always use range variables!

Expressions and Strings

```
SELECT S.name, S.age, age1=S.age+2, S.income/S.age AS iar
FROM  Senators S
WHERE  S.sname LIKE 'Ja_%Doe'
ORDER BY S.name
```

- Illustrates use of arithmetic expressions and string pattern matching
- AS and = are two ways to name fields in result.
- LIKE is used for string matching. `_' stands for any one character and `%' stands for 0 or more arbitrary characters.
- Collation: sort order for character sets

Find senators who attended either the '24-Sept-2016' or '25-Sept-2016' session

- UNION: Compute the union of two *union-compatible* sets of tuples
  - Same number/types of fields.
- Also available: INTERSECT and EXCEPT (What do we get if we replace UNION by EXCEPT?)
- SQL oddities: duplicates with union, except, intersect
  - Default: eliminate duplicates!
  - Use ALL to keep duplicates

```
SELECT S.ssn
FROM  Senators S, Attendance A
WHERE S.ssn = A.ssn
and  (A.date = '24-Sept-2016' or
        A.date = '25-Sept-2016')
```

```
SELECT S.ssn
FROM  Senators S, Attendance A
WHERE S.ssn = A.ssn
and  A.date = '24-Sept-2016'
UNION
SELECT S.ssn
FROM  Senators S, Attendance A
WHERE S.ssn = A.ssn
and  A.date = '25-Sept-2016'
```

Find senators who attended both the '24-Sept-2016' **and** '25-Sept-2016' session

- INTERSECT: Compute the intersection of any two *union-compatible* sets of tuples.
- In the SQL/92 standard, but some systems don't support it.

```
SELECT S.ssn
FROM   Senators S, Attendance A1,
       Attendance A2
WHERE S.ssn = A1.ssn and S.ssn = A2.ssn
and    A1.date = '24-Sept-2016'
and    A2.date = '25-Sept-2016'
```

```
SELECT S.ssn, S.name
FROM   Senators S, Attendance A
WHERE S.ssn = A.ssn
and    A.date = '24-Sept-2016'
INTERSECT
SELECT S.ssn, S.name
FROM   Senators S, Attendance A
WHERE S.ssn = A.ssn
and    A.date = '25-Sept-2016'
```

Key field!
What happens if S.name is used

Nested Queries

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

Find names of sailors who've reserved boat #103

Can you rewrite this to not use a nested query?

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

- Powerful feature of SQL
  - WHERE clause can itself contain an SQL query!
  - Actually, so can FROM and HAVING clauses
- To find sailors who've *not* reserved #103, use NOT IN
- Conceptual Evaluation: *nested loops* *For each Sailors tuple, check the qualification by computing the subquery.*

Nested Queries with Correlation

~~---Find names of sailors who've reserved boat #103---~~

Find names of sailors with exactly one reservation for boat #103

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

Why R.bid?

- EXISTS tests if the set is not empty
- UNIQUE returns true if the row appears only once
- Illustrates why, in general, subquery must be re-computed for each Sailors tuple.

More on Set-Comparison Operators

- We've already seen IN, EXISTS and UNIQUE. Can also use NOT IN, NOT EXISTS and NOT UNIQUE.
- Also available: *op* ANY, *op* ALL, op is <, ≤, >, ≥, =, ≠
- 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')
```

Rewriting Except Queries Using NOT IN

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

Find sailors (sid) who've reserved some boat for '24-Sept-2016' but have no reservations for '09-Oct-2016'

```
SELECT S.sid
FROM   Sailors S, Reserves R
WHERE S.sid=R.sid
    AND R.day='24-Sept-2016'
EXCEPT
(SELECT S2.sid
 FROM   Sailors S2, Reserves R2
 WHERE S2.sid=R2.sid
      AND R2.day="09-Oct-2016')
```

```
SELECT S.sid
FROM Sailors S, Reserves R
WHERE S.sid=R.sid
   AND R.day='24-Sept-2016'
    AND S.sid NOT IN
      (SELECT S2.sid
        FROM   Sailors S2, Reserves R2
        WHERE S2.sid=R2.sid
        AND R2.day="09-Oct-2016')
```

- Similarly, INTERSECT queries re-written using IN.

Division in SQL

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

Find sailors who've reserved **all** boats.

(1)
```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS
      ((SELECT B.bid
        FROM Boats B)
       EXCEPT
       (SELECT R.bid
        FROM Reserves R
        WHERE R.sid=S.sid))
```

Without EXCEPT:

(2)
```
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

Aggregate Operators

COUNT (*)
COUNT ([DISTINCT] A)
SUM ([DISTINCT] A)
AVG ([DISTINCT] A)
MAX (A) *Can use Distinct*
MIN (A) *Can use Distinct*

single column

```
SELECT  COUNT (*) FROM  Sailors S
```

```
SELECT COUNT (DISTINCT S.name)
FROM Sailors S
```

```
SELECT  AVG (S.age)
FROM  Sailors S
WHERE  S.rating=10
```

```
SELECT  AVG ( DISTINCT S.age)
FROM  Sailors S
WHERE  S.rating=10
```

```
SELECT  S.sname FROM  Sailors S
WHERE  S.rating= (SELECT  MAX(S2.rating) FROM  Sailors S2)
```

Find name & age of the oldest sailor(s)

- The first query is illegal! (wait for GROUP BY.)
- Q3 is allowed in the SQL/92 standard, but not supported in some systems

How many tuples in the result?

```
SELECT  S.sname, MAX (S.age)
FROM  Sailors S
```

```
SELECT  S.sname, S.age
FROM  Sailors S
WHERE  S.age =
          (SELECT  MAX (S2.age)
          FROM  Sailors S2)
```

```
SELECT  S.sname, S.age
FROM  Sailors S
WHERE  (SELECT  MAX (S2.age)
           FROM  Sailors S2)
           = S.age
```

GROUP BY and HAVING

- Apply aggregate to each of several *groups* of tuples
- Find the age of the youngest sailor *for each rating level*
  - Don't know: # rating levels, and rating values
  - Suppose we did know that rating values go from 1 to 10
    - we can write 10 queries that look like this (!):

For i = 1, 2, ... , 10:

```
SELECT  MIN (S.age)
FROM  Sailors S
WHERE  S.rating = i
```

```
SELECT  MIN (S.age), S.rating
FROM  Sailors S
GROUP BY  S.rating
```

Queries With GROUP BY and HAVING

```
SELECT      [DISTINCT] target-list
FROM        relation-list
WHERE       qualification
GROUP BY    grouping-list
HAVING      group-qualification
```

How many tuples in the result?

- The target-list contains
  - Attribute names: must be a subset of *grouping-list*.
  - Terms with aggregate operations (e.g., MIN (*S.age*)).
- The group-qualification
  - Must have a single value per group

Conceptual Evaluation

- Cross-product -> discard tuples -> apply projection
  -> partition into groups using the *grouping-list* attribute values
  -> eliminate groups that don't satisfy the *group-qualification*

- Expressions in *group-qualification* have a single value per group!
  - In effect, an attribute in *group-qualification* that is not an argument of an aggregate op also appears in *grouping-list*. (SQL does not exploit primary key semantics here!)
- One answer tuple is generated per qualifying group.

Find the age of the youngest sailor with age >= 18, for each rating with at least 2 such sailors

```
SELECT  S.rating,  MIN (S.age)
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1
```

- 2nd column of result is unnamed. (Use AS to name it)

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
71	zorba	10	16.0
64	horatio	7	35.0
29	brutus	1	33.0
58	rusty	10	35.0

rating	age
1	33.0
7	45.0
7	35.0
8	55.5
10	35.0

rating	
7	35.0

Answer relation

For each red boat, find the number of reservations for this boat

```
SELECT B.bid,  COUNT (*) AS scount
FROM   Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='red'
GROUP BY  B.bid
```

```
SELECT B.bid,  COUNT (*) AS scount
FROM   Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid
GROUP BY  B.bid
HAVING     B.color = 'red'
```

Would this work?

note: one color per bid

Find the age of the youngest sailor with age>18, for each rating with at least 2 sailors (of any age)

```
SELECT  S.rating,  MIN (S.age) AS MINAGE
FROM  Sailors S
WHERE  S.age > 18
GROUP BY  S.rating
HAVING  1 < (SELECT  COUNT (*) FROM  Sailors S2
                WHERE  S.rating=S2.rating)
```

- Subquery in the HAVING clause
- Compare this with the query where we considered only ratings with 2 sailors over 18!

Find ratings for which the average age is the minimum of the average age over all ratings

- Aggregate operations cannot be nested! WRONG:

```
SELECT  S.rating
FROM  Sailors S
WHERE  AVG(S.age) =
                (SELECT  MIN (AVG (S2.age))  FROM Sailors S2)
```

- Correct solution (in SQL/92):

```
SELECT  Temp.rating, Temp.avgage
FROM  (SELECT  S.rating, AVG (S.age) AS avgage
        FROM  Sailors S
        GROUP BY  S.rating) AS Temp
WHERE  Temp.avgage = (SELECT  MIN (Temp.avgage) FROM Temp)
```

Null Values

- Represent
  - *unknown* (e.g., rating not assigned) or
  - *inapplicable* (e.g., no spouse's name)
- Complications with nulls:
  - Operators to check if value is/is not *null*.
  - Is *rating > 8* true or false when *rating* is null?
    - Answer: Evaluate to unknown
  - What about AND, OR and NOT connectives?
    - Need 3-valued logic (true, false and *unknown*)
      - Not unknown = unknown
  - WHERE clause eliminates rows that don't evaluate to true
  - New operators (in particular, *outer joins*) possible/needed.

p	q	p AND q	p OR q
T	**T**	T	T
T	**F**	F	T
T	**U**	U	T
F	**T**	F	T
F	**F**	F	F
F	**U**	F	U
U	**T**	U	T
U	**F**	F	U
U	**U**	U	U

Outer Join

Sailors

sid	sname	rating	age
22	dustin	7	45.0
58	rusty	10	35.0

Reserves

sid	bid	day
22	101	10/10/99

```
Select S.sid, R.bid
From Sailors S NATURAL LEFT OUTER JOIN Reserves R
```

Result

sid	bid
22	101
58	null

Similarly:

- RIGHT OUTER JOIN
- FULL OUTER JOIN

Embedded SQL

- Call SQL commands from a host language (e.g., C) program.
  - SQL statements can refer to host variables (including special variables used to return status).
  - Must include a statement to *connect* to the right database.
- SQL relations are (multi-) sets of records, with no *a priori* bound on the number of records. No such data structure in C.
  - SQL supports a mechanism called a *cursor* to handle this.

Cursors

- Can declare a cursor on a relation or query statement (which generates a relation).
- Can *open* a cursor, and repeatedly *fetch* a tuple then *move* the cursor, until all tuples have been retrieved.
  - Special clause, called ORDER BY, in cursor queries to control the order in which tuples are returned.
  - Fields in ORDER BY must also appear in SELECT clause.
- Can also modify/delete tuple pointed to by a cursor

Cursor that gets names of sailors who've reserved a red boat, in alphabetical order

```
EXEC SQL DECLARE sinfo CURSOR FOR
  SELECT  S.sname
  FROM  Sailors S, Boats B, Reserves R
  WHERE  S.sid=R.sid AND R.bid=B.bid AND B.color='red'
  ORDER BY  S.sname
```

- Can we replace *S.sname* by *S.sid* in the ORDER BY clause!
  - Every column in the ORDER BY clause must appear in the SELECT clause

Integrity Constraints

- An IC describes conditions that every *legal instance* of a relation must satisfy.
  - Inserts/deletes/updates that violate IC's are disallowed.
- *Types of IC's*: Domain constraints, primary key constraints, foreign key constraints, general constraints.
- Can create new domains, Domain Constraints:
  - CREATE DOMAIN LegalRatings INTEGER DEFAULT 0
    CHECK (VALUE >= 1 and VALUE <=10)
  - Create Table Sailor (…, rating LegalRatings, …)
  - Underlying domain is still Integers for comparison
- Can create new types: CREATE TYPE AllRatings as INTEGER
  - Underlying domain is now a new type. Can't compare with INTEGER without a cast. None of the aggregates on INTEGER work on AllRatings

Table Constraints

- More general ICs than key constraints
- Can use queries to express constraint
- Constraints can be named.

```
CREATE TABLE Sailors
        ( sid INTEGER,
        sname CHAR(10),
        rating INTEGER,
        age REAL,
        PRIMARY KEY (sid),
        CHECK ( rating >= 1
                AND rating <= 10 )
```

```
CREATE TABLE Reserves
        ( sname CHAR(10),
        bid INTEGER,
        day DATE,
        PRIMARY KEY (bid,day),
        CONSTRAINT noInterlakeRes
        CHECK (`Interlake' <>
                ( SELECT B.bname
                FROM Boats B
                WHERE B.bid=bid)))
```

Constraints Over Multiple Relations

- Awkward & Wrong!
- If Sailors is empty, the number of Boats tuples can be anything!
- ASSERTION is the right solution; not associated with either table

```
CREATE TABLE Sailors
( sid INTEGER,
sname CHAR(10),
rating INTEGER,
age REAL,
PRIMARY KEY (sid),
CHECK
((SELECT COUNT (S.sid) FROM Sailors S) +
(SELECT COUNT (B.bid) FROM Boats B) < 100)
```

Number of boats plus number of sailors is < 100

```
CREATE ASSERTION smallClub
CHECK
((SELECT COUNT (S.sid) FROM Sailors S) +
 (SELECT COUNT (B.bid) FROM Boats B) < 100)
```

Triggers

- Trigger: procedure that starts automatically if specified changes occur to the DBMS
- Three parts:
  - Event (activates the trigger)
  - Condition (tests whether the triggers should run)
  - Action (what happens if the trigger runs)
    - Before and After Triggers
- Trigger Execution
  - Row-level Triggers: Once per row
  - Statement-level Triggers: Once per SQL statement

Triggers: Example

```
CREATE TRIGGER init_count BEFORE INSERT ON Students   /* Event */
    DECLARE
     count INTEGER                                    /* Action */
    BEGIN
     count := 0
    END


CREATE TRIGGER incr_count AFTER INSERT ON Student     /* Event */
    WHEN (new.age < 18)                    /* Condition */
    FOR EACH ROW
    BEGIN                                 /* Action */
     count := count + 1;
    END
```

Triggers

- First trigger executed *before* the activating statement, second executes *after* the activating statement.
- Options:
  - "BEFORE"
  - "AFTER"
  - "INSTEAD OF" (only valid on views)
- In combination with:
  - "FOR EACH ROW" - execute once per modified record
  - (default) - execute once per activating statement.
    Can also specify using "FOR EACH STATEMENT"
- In combination with:
  - "INSERT"
  - "DELETE"
  - "UPDATE"

Triggers

- Referring to values
  - Old
  - New
  - Set of changed record

Triggers: Example

```
CREATE TRIGGER youngSailorUpdate
    AFTER INSERT ON SAILORS
REFERENCING NEW TABLE NewSailors
FOR EACH STATEMENT
    INSERT
     INTO YoungSailors(sid, name, age, rating)
     SELECT sid, name, age, rating
     FROM NewSailors N
     WHERE N.age <= 18
```

Triggers v/s Constraints

- Often used to maintain consistency
  - Can you use a foreign key?
  - Foreign keys are not defined operationally
- Constraints are easier to understand than triggers
- Triggers are more powerful.
  - Often used to fill out fields in a form
  - Check complex actions (such as credit limit in a shopping application)
  - Check preferred customer status
  - Generate logs for auditing and security checks.
  - Internally can be used by the DBMS for replication management.