SQL: Queries, Constraints, Triggers

Linda Wu

(CMPT 354 • 2004-2)

Topics

- Basic SQL queries
- Set operations
- Aggregate operations
- Null values
- General constraints
- Triggers

Basic SQL Queries

- Syntax of a basic SQL query
  - SELECT clause: specify columns to be retained in the result (projection in RA)
  - FROM clause: specify a cross-product of tables
  - WHERE clause: optional; specify selection conditions on the tables in FROM clause (selections in RA)

```
SELECT    [DISTINCT] select-list
FROM      from-list
WHERE     qualification
```


Basic SQL Queries (Cont.)

- *from-list*: a list of tables
  - Possibly with a range variable after each table name
- *select-list*: a list of column names of tables in *from-list*
- *qualification*: comparisons combined using AND, OR and NOT
  - Comparison: attr *op* const, or, attr1 *op* attr2
  - *op* is one of $<$, $>$, $=$, $\leq$, $\geq$, $\neq$
- DISTINCT: an optional keyword indicating that the result should not contain duplicates
  - Default: duplicates are not eliminated!

Basic SQL Queries (Cont.)

- Semantics of an SQL query is defined in terms of the following conceptual evaluation strategy
  - Compute the cross-product of *from-list*
  - Discard resulting tuples if they fail *qualification*
  - Delete attributes that are not in *select-list*
  - If DISTINCT is specified, eliminate duplicate rows
- This strategy is typically the least efficient way to compute a query!
- An optimizer will find more efficient strategies to compute the same answers

Basic SQL Queries (Cont.)

- Examples

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

```
SELECT  DISTINCT S.sname, S.age
FROM    Sailors AS S
```

• Without *DISTINCT*:
The result might be a multiset (an unordered collection of elements, in which each element could appear for several times)

• *AS*: optional keyword to introduce a range variable

Basic SQL Queries (Cont.)

- Range variables
  - Needed only if the same relation appears more than once in the FROM clause
  - It is a good style to always use range variables!

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

||

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


Basic SQL Queries (Cont.)

- Find the *sid*'s of sailors who have reserved at least one boat
  - 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?

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

Basic SQL Queries (Cont.)

- More general version of *select-list*
  - A list of columns
  - *expression* AS *new_column_name*
  - *new_column_name* = *expression*
    - *expression*: any arithmetic or string expression over column names and constants
    - *new_column_name*: a new name for a field in result
  - sum, count

Basic SQL Queries (Cont.)

- More general version of *qualification*
  - Arithmetic expressions
  - String comparison (=, <, >, etc.)
    - The ordering of strings is determined alphabetically
    - String pattern matching using LIKE and wild card symbols
      - '_' stands for any one character
      - '%' stands for 0 or more arbitrary characters

Basic SQL Queries (Cont.)

- Examples

```
SELECT S.age, age1 = S.age - 5, 2 * S.age AS age2
FROM   Sailors S
WHERE  S.sname  LIKE 'B_%B'
```

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


Set Operations

- UNION: union
- INTERSECT: intersection
- EXCEPT: set difference
  - UNION/INTERSECT/EXCEPT ALL
- IN: to check if an element is in a set
- EXISTS: to check if a set is empty
  - NOT IN / NOT EXISTS
- UNIQUE / NOT UNIQUE: to check duplicates
- ANY / ALL: to compare a value with the elements in a set

Set Operations (Cont.)

- UNION
  - To compute the union of two union-compatible sets of tuples
  - Example: find sid's of sailors who have reserved a red or a green boat
  - What if we replace UNION by EXCEPT?

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

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


Set Operations (Cont.)

- INTERSECT
  - To compute the intersection of two union-compatible sets
  - In the SQL/92 standard; but some systems don't support it
  - Example: find sid's of sailors who have reserved a red and a green boat

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

```
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'
```


Set Operations (Cont.)

- Duplicate elimination in UNION, INTERSECT, and EXCEPT
  - Default: duplicates are eliminated
  - To retain duplicates
    - UNION/INTERSECT/EXCEPT ALL

* Compare with basic query form:
The default is NO duplicate elimination unless DISTINCT is specified

Set Operations (Cont.)

- Nested queries
  - A very powerful feature of SQL
  - A nested query is a query that has another query (subquery) embedded within it
  - Subquery typically appears in WHERE clause
  - Subquery sometimes appears in FROM or HAVING clauses
  - Set comparison operators
    - IN, EXISTS, UNIQUE, ANY, ALL (the first 3 may be used together with NOT)

Set Operations (Cont.)

- Nested query example

  Find the names of sailors who have reserved boat #103

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

  - To understand semantics of nested queries, think of a nested loop evaluation: for each Sailors tuple, check the qualification by (re)computing the subquery
  - What if IN is replaced by NOT IN?

Set Operations (Cont.)

- Multiply nested query
  - Find the names of sailors who have not reserved a red boat

```
SELECT S.sname
FROM   Sailors S
WHERE  S.sid NOT IN (
           SELECT R.sid
           FROM   Reserves R
           WHERE  R.bid IN ( SELECT B.bid
                             FROM   Boat B
                             WHERE  B.color = 'red' ) )
```


Set Operations (Cont.)

- Nested queries with correlation
  - Inner subquery depends on the row currently being examined in the outer query, and must be re-evaluated for each row in the outer query

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

Find names of sailors who have reserved boat #103

Set Operations (Cont.)

- UNIQUE / NOT UNIQUE
  - Checks for duplicate tuples
  - When apply UNIQUE to a subquery, the resulting condition returns true if:
    - there are no duplicate tuples in the answer to the subquery, or,
    - the answer to the subquery is empty

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

Finds sailors with at most one reservation for boat #103

Set Operations (Cont.)

- *op* ANY (subquery)
  - *op* is one of >, <, =, ≥, ≤, ≠
  - Subquery must return a row that makes the comparison "*op*" true in order for "*op* ANY" to return true
  - If subquery returns empty result, "*op* ANY" returns false
- *op* ALL (subquery)
  - Every row returned by subquery must makes the comparison "*op*" true in order for "*op* ALL" to return true
  - If subquery returns empty result, "*op* ALL" returns true

Set Operations (Cont.)

- Find sailors whose rating is greater than some sailor called Horatio

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

- Find sailors with the highest rating

```
SELECT *
FROM   Sailors S
WHERE  S.rating >= ALL ( SELECT S2.rating
                         FROM   Sailors S2 )
```


Set Operations (Cont.)

- INTERSECT query can be rewritten using IN
- EXCEPT query can be rewritten using NOT IN

```
SELECT  S.sid
FROM    Sailors S, Boats B, Reserves R
WHERE   S.sid=R.sid AND R.bid=B.bid AND B.color='red'
        AND S.sid IN ( SELECT  S2.sid
                       FROM    Sailors S2, Boats B2, Reserves R2
                       WHERE   S2.sid=R2.sid AND R2.bid=B2.bid
                               AND  B2.color='green')
```

or, **NOT IN**

Set Operations (Cont.)

- Division in SQL: find the names of sailors who have reserved all boats

Solution 1: for each sailor S, check the set of boats reserved by S includes every boat

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

Set Operations (Cont.)

Solution 2: for each sailor S, check there is no boat that has not been reserved by S

```
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 ) )
```


Aggregate Operations

- Aggregate operations
  - A significant extension of relational algebra
  - Five aggregate operators
    1. COUNT (*)
       COUNT ([DISTINCT] A)
    2. SUM ([DISTINCT] A)
    3. AVG ([DISTINCT] A)
    4. MAX (A)
    5. MIN (A)

       A: a single column of a relation

Aggregate Operations (Cont.)

```
SELECT  COUNT (*)
FROM    Sailors S
(1)
```

```
SELECT  COUNT (DISTINCT S.rating)
FROM    Sailors S
WHERE   S.sname = 'Bob'          (2)
```

```
SELECT  AVG (S.age)
FROM    Sailors S
WHERE   S.rating=10
(3)
```

```
SELECT  AVG (DISTINCT S.age)
FROM    Sailors S
WHERE   S.rating = 10
(4)
```

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


Aggregate Operations (Cont.)

- Example: find the name and age of the oldest sailor(s)

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

illegal!

```
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
```

may not be supported!

Aggregate Operations (Cont.)

- Motivation for grouping

  Consider: *find the age of the youngest sailor for each rating level*

  - In general, we don't know how many rating levels exist, and what the rating values for these levels are!
  - Suppose we 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
```


Aggregate Operations (Cont.)

- GROUP BY and HAVING clauses
  - To apply aggregate operations to each of a number of groups of rows in a relation
  - Each row in the result of the query corresponds to one group

* Group: a collection of rows sharing some common features (*grouping-list*)

```
SELECT    [DISTINCT] select-list
FROM      from-list
WHERE     qualification
GROUP BY  grouping-list
HAVING    group-qualification
```


Aggregate Operations (Cont.)

- All rows in a group have the same value for all attributes in *grouping-list*
- An answer row is to be generated for a given group if it satisfies *group-qualification*
- *select-list* contains:
  1) A list of column names: must be a subset of *grouping-list*
  2) A list of terms with aggregate operations, e.g., MIN (*S.age*)

Aggregate Operations (Cont.)

- Conceptual evaluation strategy for GROUPING BY
  1) Construct the cross-product of tables in from-list
  2) Discard the tuples that fail qualification in WHERE clause
  3) Eliminate unnecessary fields (not in SELECT, GROUP BY or HAVING clause)
     - SQL does not eliminate duplicates unless DISTINCT is specified
  4) Partition the remaining tuples into groups by the values of attributes in grouping-list

Aggregate Operations (Cont.)

5) Apply the group-qualification to eliminate some groups. Expressions in group-qualification must have a single value per group!
   - In effect, an attribute in group-qualification that is not an argument of an aggregate operation also appears in grouping-list (SQL does not exploit primary key semantics here!)
6) Generate one answer tuple per qualifying group
7) Eliminate duplicates if DISTINCT is contained in SELECT clause

Aggregate Operations (Cont.)

- Find the age of the youngest sailor with age ≥ 18 for each rating level with at least 2 such sailors

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

sid	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5
96	Frodo	3	25.5

Aggregate Operations (Cont.)

Step 1,2,3

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
9	35.0
3	25.5
3	63.5
3	25.5

Step 4,5

rating	age
~~1~~	~~33.0~~
3	25.5
3	25.5
3	63.5
7	45.0
7	35.0
8	55.5
8	25.5
~~9~~	~~35.0~~
~~10~~	~~35.0~~

Step 6

rating	minage
3	25.5
7	35.0
8	25.5

Conceptual evaluation strategy

Aggregate Operations (Cont.)

- Find the age of the youngest sailor with age ≥ 18 for each rating level with at least 2 such sailors and with every sailor under 60

```
SELECT   S.rating, MIN (S.age)  AS minage
FROM     Sailors S
WHERE    S.age >= 18
GROUP BY S.rating
HAVING   COUNT (*) > 1 AND EVERY (S.age ≤ 60)
```

EVERY and ANY are supported by SQL 99

- What if EVERY is changed to ANY?

Aggregate Operations (Cont.)

Step 1,2,3

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
9	35.0
3	25.5
3	63.5
3	25.5

Step 4,5

rating	age
~~1~~	~~33.0~~
~~3~~	~~25.5~~
~~3~~	~~25.5~~
~~3~~	~~63.5~~
7	45.0
7	35.0
8	55.5
8	25.5
~~9~~	~~35.0~~
~~10~~	~~35.0~~

Step 6

rating	minage
7	35.0
8	25.5

Aggregate Operations (Cont.)

- Find the age of the youngest sailor with age between 18 and 60 for each rating level with at least 2 such sailors

```
SELECT   S.rating, MIN (S.age) AS minage
FROM     Sailors S
WHERE    S.age >= 18 AND S.age ≤ 60
GROUP BY S.rating
HAVING   COUNT (*) > 1
```

rating	minage
3	25.5
7	35.0
8	25.5

Answer relation

Aggregate Operations (Cont.)

- For each red boat, find the number of reservations for this boat
  - Grouping over a join of two relations

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

  - To move *B.color='red'* from the WHERE clause to HAVING clause: illegal!
    - Only columns in the GROUP BY clause can appear in the HAVING clause, unless they are arguments to an aggregate operator in the HAVING clause

Aggregate Operations (Cont.)

- Find the age of the youngest sailor with age > 18, for each rating level with at least 2 sailors (of any age)
  - HAVING clause can also contain a subquery
  - What if HAVING clause is replaced by:

    HAVING COUNT(*) >1 ?

```
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 )
```

Aggregate Operations (Cont.)

Step 1,2,3

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
9	35.0
3	25.5
3	63.5
3	25.5

Step 4,5

rating	age	# of sailors
~~1~~	~~33.0~~	~~(1)~~
3	25.5	
3	25.5	(3)
3	63.5	
7	45.0	
7	35.0	(2)
8	55.5	
8	25.5	(2)
~~9~~	~~35.0~~	~~(1)~~
10	35.0	(2)

Step 6

rating	minage
3	25.5
7	35.0
8	25.5
10	35.0

Aggregate Operations (Cont.)

- Find those ratings for which the average age is the minimum over all ratings
  - Aggregate operations, for example AVG(S2.age), cannot be nested!

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

Wrong solution !

Aggregate Operations (Cont.)

Correct solution:

```
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 )
```

- SQL allows nested subquery in FROM clause, but not all DBMSs support it
- HAVING clause is not necessary
- When a subquery appears in FROM, give it a name using keyword *AS*

Null Values

- Null
  - A special value that denote *unknown* (e.g., a rating has not been assigned) or *inapplicable* (e.g., no spouse's name)
- The presence of *null* complicates many issues
  - Special operators are needed to check if value is/is not *null*
  - Is (*rating>8*) true or false when *rating* is *null*? What about AND, OR and NOT connectives?
  - We need a 3-valued logic (true, false and *unknown*)
  - Meaning of SQL constructs must be defined carefully (e.g., WHERE clause eliminates rows that don't evaluate to true)
  - New operators (in particular, *outer joins*) are possible/needed

Null Values (Cont.)

- Outer joins
  - When *S* outer joins *R*, *S* rows without a matching *R* rows according to the join condition appear exactly once in the result, with the result columns inherited from *R* assigned *null* values

sid	sname	rate	age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

Sailors

sid	bid	day
22	101	10/10/96
58	103	11/12/96

Reserves

```
SELECT  S.sid,  R.bid
FROM    Sailors S NATURAL LEFT OUTER JOIN
        Reserves R
```

→

sid	bid
22	101
31	null
58	103

General Constraints

- Useful when more general ICs other than keys are involved
- Can use queries to express constraints
- Constraints can be named
- Constraints may be specified over a single table, or, multiple tables

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


General Constraints (Cont.)

- Table constraints: over a single table
  CHECK *conditional-expression*
  - *conditional-expression* may refer to other tables
  - *conditional-expression* is required to hold only if the associated table is not empty

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


General Constraints (Cont.)

(# of boats) + (# of sailors) < 100

```
CREATE  TABLE  Sailors                                  (?)
    ( sid  INTEGER,
      ......
      CHECK ( (SELECT COUNT (S.sid) FROM Sailors S)
              + (SELECT COUNT (B.bid) FROM Boats B)
              < 100 ) )
```

- If Sailors is empty, this constraint always holds (the number of Boats tuples can be anything!)
- ASSERTION is the right solution; not associated with either table

General Constraints (Cont.)

- Assertion
  - Constraint over multiple tables, but NOT associated with any table

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


Triggers

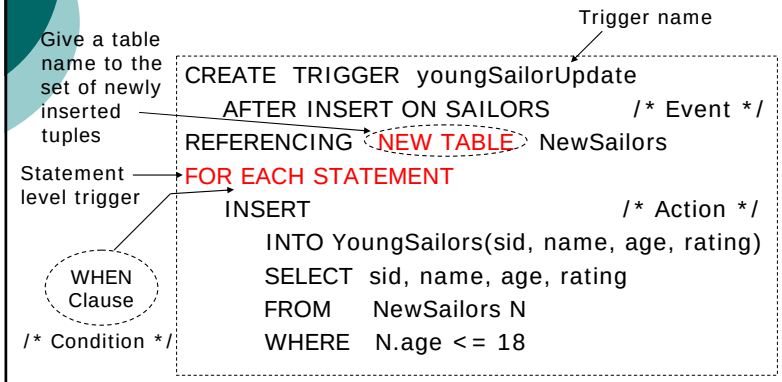
- Trigger
  - A procedure that is automatically invoked when specified changes occur to the DBMS
  - Can be though of as a 'daemon' in DBMS
- Trigger description has 3 parts
  - Event (activates the trigger)
  - Condition (tests whether the triggers should run)
  - Action (what happens if the trigger runs)
- A database with a set of associated triggers is called an active database

Triggers (Cont.)

- Trigger event
  - An insert, delete or update statement
- Trigger condition
  - A true/false statement
  - A query (true if its answer is nonempty)
- Trigger action
  - May examine the answer to the query in trigger condition, execute new queries, make change to the database, ...
  - May execute before / after / instead of activating statement
  - May execute once per modified record, or, per activating statement

Triggers (Cont.)

- Example (SQL 99 syntax)



Triggers (Cont.)

- If a statement activates multiple triggers, the DBMS processes all of them, in some arbitrary order
- Executing the action part of a trigger could in turn activates another trigger: recursive triggers
- Usage
  - Maintain database consistency in more flexible way than constraints
  - Gather statistics on table access and modification
  - Alert users to unusual events
  - .....

Summary

- SQL was an important factor in the early acceptance of the relational model
- SQL is relationally complete; in fact, it is more expressive than relational algebra
- Even queries that can be expressed in RA can often be expressed more naturally in SQL
- There are many alternative ways to write a query; optimizer should look for the most efficient evaluation plan
- NULL value brings many complications
- SQL allows rich integrity constraints
- Triggers respond to changes in the database