CS 450

SQL - 1

Basic form of SQL Queries

```
SELECT      target-list
FROM        relation-list
WHERE       qualification
```

- *target-list* A list of attributes of output relations in *relation-list*
- *relation-list* A list of relation names (possibly with a *range-variable* after each name)

  e.g. Sailors S, Reserves R
- *qualification* Comparisons (Attr *op* const or Attr1 *op* Attr2, where *op* is one of <, >, ≤, ≥, =, ≠) combined using AND, OR and NOT.

What's contained in an SQL Query?

```
SELECT    target-list
FROM      relation-list
WHERE     qualification
```

Every SQL Query must have:

- *SELECT clause: specifies columns to be retained in result*
- *FROM clause: specifies a cross-product of tables*

The WHERE clause (optional) specifies selection conditions on the tables mentioned in the FROM clause

General SQL Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following 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*.
- This strategy is probably the least efficient way to compute a query! An optimizer will find more efficient strategies to compute *the same answers*.

Table Definitions

We will be using the following relations in our examples:

Sailors(sid, sname, rating, age)

Boats(bid, bname, color)

Reserves(sid, bid, day)

Sailors

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

Reserves

sid	*bid*	*day*
22	101	10/10/04
22	102	10/10/04
22	103	10/08/04
22	104	10/07/04
31	102	11/10/04
31	103	11/06/04
31	104	11/12/04
64	101	09/05/04
64	102	09/08/04
74	103	09/08/04

Boats

bid	*bname*	*Color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

A Simple SQL Query

Find the names and ages of all sailors

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

Result of Previous Query

sname	***age***
Dustin	45.0
Brutus	33.0
Lubber	55.5
Andy	25.5
Rusty	35.0
Horatio	35.0
Zorba	16.0
Horatio	35.0
Art	25.5
Bob	63.5

```
SELECT S.sname, S.age
FROM Sailors S;
```

Duplicate Results

Preventing Duplicate Tuples in the Result

- Use the DISTINCT keyword in the SELECT clause:

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

Results of Original Query without Duplicates

sname	***age***
Dustin	45.0
Brutus	33.0
Lubber	55.5
Andy	25.5
Rusty	35.0
Horatio	35.0
Zorba	16.0
Art	25.5
Bob	63.5

Appears only once

The from Clause

- The **from** clause lists the relations involved in the query
  - Corresponds to the Cartesian product operation of the relational algebra.
- Find the Cartesian product *sailors* x *reserves*

  select *
  from *sailors, reserves*

  - generates every possible sailors – reserves pair, with all attributes from both relations
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra)

Example SQL Query…1

Find the names of sailors who have reserved boat 103

Relational Algebra:

$\pi_{sname}((\sigma_{bid=103} Reserves) \bowtie Sailors)$

SQL:

Example SQL Query…1

Find the names of sailors who have reserved boat 103

<u>Relational Algebra</u>:

$\pi_{\text{sname}} ((\sigma_{\text{bid=103}} \mathit{Reserves}) \bowtie \mathit{Sailors})$

<u>SQL</u>:

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

Example SQL Query…1

Find the names of sailors who have reserved boat 103

Relational Algebra:

$\pi_{sname}\ ((\sigma_{bid=103} \mathit{Reserves}) \bowtie \mathit{Sailors})$

SQL – use NATURAL JOIN

```
SELECT  S.sname
FROM    Sailors S NATURAL JOIN Reserves R
WHERE   R.bid=103;
```

Example SQL Query…1

Find the names of sailors who have reserved boat 103

Relational Algebra:

$\pi_{\text{sname}} ((\sigma_{\text{bid=103}} \mathit{Reserves}) \bowtie \mathit{Sailors})$

SQL – use JOIN

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

Result of Previous Query

sid	*bid*	*day*
22	103	10/08/04
31	103	11/06/04
74	103	09/08/04

$\bowtie$

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

Result:

sname
Dustin
Lubber
Horatio

A Note on Range Variables

- Really needed only if the same relation appears twice in the FROM clause. The previous query can also be written as:

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

However, it is a good style to always use range variables!

OR

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

Example SQL Query…2

*Find the **sids** of sailors who have reserved a red boat*

```
SELECT R.sid
FROM Boats B, Reserves R
WHERE B.bid=R.bid AND B.color= ‘red’;
```

Example SQL Query…3

*Find the **names** of sailors who have reserved a red boat*

```
SELECT S.sname
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND B.bid=R.bid AND
            B.color= ‘red’ ;
```

Example SQL Query…4

*Find the **colors** of boats reserved by 'Lubber'*

```
SELECT B.color
FROM Sailors S, Reserves R, Boats B
WHERE S.sid=R.sid AND R.bid=B.bid AND
          S.sname= 'Lubber' ;
```

Example SQL Query…5

Find the ***names*** *of sailors who have reserved* ***at least*** *one boat*

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

Expressions and Strings

- AS and = are two ways to name fields in result.
- LIKE is used for string matching. ‘_’ stands for exactly one arbitrary character and ‘%’ stands for 0 or more arbitrary characters.

Expressions and Strings Example

Find triples (of ages of sailors and two fields defined by expressions, i.e. current age-1 and twice the current age) for sailors whose names begin and end with B and contain at least three characters.

```
SELECT  S.age, age1=S.age-1, 2*S.age AS age2
FROM  Sailors S
WHERE  S.sname LIKE ‘B_%B’;
```

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

Result:

age	*age1*	*age2*
63.5	62.5	127.0

UNION, INTERSECT, EXCEPT

- UNION: Can be used to compute the union of any two *union-compatible* sets of tuples (which are themselves the result of SQL queries).
- EXCEPT: Can be used to compute the set-difference operation on two *union-compatible* sets of tuples (Note: In ORACLE, the command for set-difference is *MINUS*).
- INTERSECT: Can be used to compute the intersection of any two *union-compatible* sets of tuples.

Illustration of UNION…1

Find the names of sailors who have reserved a red ***or*** *a green boat*

Intuitively, we would write:

```
SELECT  S.sname
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’ );
```

Illustration of UNION…2

We can also do this using a UNION keyword:

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

Unlike other operations, UNION eliminates duplicates! Same as INTERSECT, EXCEPT. To retain duplicates, use “**UNION ALL**”

Illustration of INTERSECT…1

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

Intuitively, we would write the SQL query as:

```
SELECT  S.sname
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');
```

Illustration of INTERSECT…2

We can also do this using a INTERSECT keyword:

```
SELECT  S.sname
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid AND B.color= ‘red’
INTERSECT
SELECT  S2.sname
FROM  Sailors S2, Boats B2, Reserves R2
WHERE  S2.sid=R2.sid AND R2.bid=B2.bid AND B2.color= ‘green’ ;
```

(Is this correct??)

(Semi-)Correct SQL Query for the Previous Example

```
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  S2.sid
FROM  Sailors S2, Boats B2, Reserves R2
WHERE  S2.sid=R2.sid AND R2.bid=B2.bid
             AND B2.color= ‘green’ ;
```

(This time we have actually extracted the ***sids*** of sailors, and not their names.)
(But the query asks for the names of the sailors.)

Illustration of EXCEPT

*Find the sids of all sailors who have reserved red boats **but not** green boats:*

```
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid AND B.color= ‘red’
EXCEPT
SELECT  S2.sid
FROM  Sailors S2, Boats B2, Reserves R2
WHERE  S2.sid=R2.sid AND R2.bid=B2.bid AND B2.color= ‘green’ ;
```

Use MINUS instead of EXCEPT in Oracle

Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null*
  - Example: 5 + *null* returns null
- The predicate **is null** can be used to check for null values.
  - Example: Find all sailors whose ratings are null.

SELECT *S.sid*
FROM *Sailors S*
where *S.rating* **is null**