



SQL Queries

Chapter 5



SQL Query Language

- Implements relational algebra...
 - Select, Project, Join, Set operators (we will see these in the next lecture)
- And so much more...
 - Correlated subqueries
 - Ordering of results
 - Aggregate queries (e.g., SUM, MAX, AVG)
 - Three-valued logic for NULL values
 - Etc.

Basic SQL Query

Optional

Attributes from
input relations

List of relations

SELECT [DISTINCT] attr-list
FROM relation-list
WHERE qualification

Attr1 **op** Attr2

OPS: <, >, =, <=, >=, <>
Combine using AND, OR, NOT

(Conceptual) Evaluation:

1. Take cross-product of relation-list
2. Select rows satisfying qualification
3. Project columns in attr-list
(eliminate duplicates only if DISTINCT)

**Optimizer
chooses
efficient
plan!!**



Example of Basic Query

- Schema:
 - Sailors (sid, sname, rating, age)
 - Boats (bid, bname, color)
 - Reserves (sid, bid, rday)
- Find the names of sailors who have reserved boat #103

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

Showing JOINS explicitly

- Find the names of sailors who have reserved boat #103. All the following equivalent.

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

Cross-product syntax:

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

Join syntax:

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

Natural join syntax (joins on common attributes):

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



INNER Joins

- The join we just saw is also called an INNER JOIN. (we will see outer joins shortly)

Join syntax:

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

Eqvt. Inner join syntax

```
SELECT S.sname  
FROM  
Sailors S INNER JOIN Reserves R  
ON S.sid = R.sid  
WHERE  
R.bid = 103;
```

Result from the query

Reserves

| sid | bid | rday |
|-----|-----|-------|
| 22 | 101 | 10/10 |
| 58 | 103 | 11/12 |

Sailors

| sid | sname | rating | age |
|-----|--------|--------|-----|
| 22 | Dustin | 7 | 45 |
| 58 | Rusty | 10 | 35 |
| 31 | Lubber | 8 | 55 |

Reserves x Sailors

| sid | bid | rday | sid | sname | rating | age |
|-----|-----|-------|-----|--------|--------|-----|
| 22 | 101 | 10/10 | 22 | Dustin | 7 | 45 |
| 22 | 101 | 10/10 | 58 | Rusty | 10 | 35 |
| 22 | 101 | 10/10 | 31 | Lubber | 8 | 55 |
| 58 | 103 | 11/12 | 22 | Dustin | 7 | 45 |
| 58 | 103 | 11/12 | 58 | Rusty | 10 | 35 |
| 58 | 103 | 11/12 | 31 | Lubber | 8 | 55 |



Eliminating duplicates

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




Another Example

- Schema:
 - Sailors (sid, sname, rating, age)
 - Boats (bid, bname, color)
 - Reserves (sid, bid, day)
- Find the colors of boats reserved by a sailor named rusty

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



Note on Range Variables

- Needed when same relation appears twice in FROM clause

```
SELECT S1.sname, S2.sname  
FROM Sailors S1, Sailors S2  
WHERE S1.age > S2.age;
```

**What does this
Query compute?**

Good style to always use range variables anyway...

Outer Joins

Sailors

| <u>sid</u> | sname | rating | age |
|------------|--------|--------|------|
| 22 | dustin | 7 | 45.0 |
| 58 | rusty | 10 | 35.0 |

Reserves

| <u>sid</u> | bid | day |
|------------|-----|----------|
| 22 | 101 | 10/10/99 |

Select S.sid, R.bid

From Sailors S NATURAL **LEFT** [OUTER] JOIN Reserves R

Result

| <u>sid</u> | bid |
|------------|------|
| 22 | 101 |
| 58 | null |

Similarly:

- **RIGHT OUTER JOIN**
- **FULL OUTER JOIN**

Note: OUTER is default, when using LEFT, RIGHT, or FULL

JOIN syntax with multiple tables

Sailors

| <u>sid</u> | sname | rating | age |
|------------|--------|--------|------|
| 22 | dustin | 7 | 45.0 |
| 58 | rusty | 10 | 35.0 |

Reserves

| <u>sid</u> | bid | day |
|------------|-----|----------|
| 22 | 101 | 10/10/99 |

```
Select S.sname, B.bname
From Sailors S JOIN Reserves R ON (S.sid = R.sid)
JOIN Boats B ON (R.bid = B.bid) WHERE S.name = 'dustin';
```

Similarly:

- RIGHT [OUTER] JOIN ON ...
- FULL [OUTER] JOIN ON ...
- LEFT [OUTER] JOIN ON ...
- NATURAL JOINS (outer and inner)



ORDER BY clause

- Helps sort the result for presentation

Attribute(s) in ORDER BY clause must be in SELECT list.

*Find the names and ages
of all sailors, in increasing
order of age*

```
SELECT S.sname, S.age  
FROM Sailors S  
ORDER BY S.age [ASC]
```

*Find the names and ages
of all sailors, in decreasing
order of age*

```
SELECT S.sname, S.age  
FROM Sailors S  
ORDER BY S.age DESC
```



ORDER BY clause

```
SELECT S.sname, S.age, S.rating  
FROM Sailors S  
ORDER BY S.age ASC, S.rating DESC
```

What does this query compute?

Find the names, ages, and rankings of all sailors.

Sort the result in increasing order of age.

If there is a tie, sort those tuples in decreasing order of rating.



Set Operators

- UNION (eliminates duplicates)
- UNION ALL (keeps duplicates)
- INTERSECT
- EXCEPT or MINUS (set difference)



Union Example

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

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

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




Intersect

- Find sids of sailors who have reserved a red and a green boat

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

What is wrong with the above query?



Set Difference Example

- Find tuples in A that are not in B.

```
SELECT * FROM A  
MINUS  
SELECT * FROM B;
```



Intersect Example

- Find sids of sailors who have reserved a red and a green boat

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



Aggregate Operators

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

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

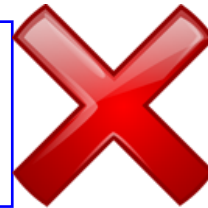
single column

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

Aggregate Query - Example

- Find name and age of oldest sailor(s)

```
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)
```

How many tuples
in the result?

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



Nested Queries

- Query with another query embedded inside

**What does this
Compute?**

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

Conceptual evaluation: *For each row of Sailors,
evaluate the subquery over reserves*

To find sailors who have not reserved 103, use NOT IN



Over-Use of Nesting

- Common error by novice SQL programmers.
- Query optimizers not as good at optimizing queries across nesting boundaries.
- Try hard first to write non-nested.

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



More Set Comparison Operators

- Set comparisons:
 - *attr* IN R : true if R contains *attr*
 - EXISTS R : true if R is not an empty relation
 - UNIQUE R : true if no duplicates in R
 - You can use NOT with these, e.g., NOT EXISTS
- Also available ANY or ALL: (op is $<$, \leq , $>$, \geq , $=$, \neq)
 - *attr* $>$ ANY R : some element of R satisfies the condition that *attr* $>$ that element.
 - *attr* $<$ ALL R : all elements of R satisfy the condition that *attr* $<$ element.



Example

- Find sailors whose rating is greater than that of all sailors called Horatio:

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

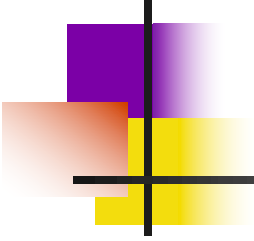


Example

- Q1: What does this query compute?

```
SELECT S.sid FROM Sailors S
WHERE S.rating > ANY (SELECT S2.rating
                      FROM Sailors S2
                      WHERE S2.name =
                        'John');
```

- Q2: Rewrite the query without using a nested query

- 
-
- `SELECT DISTINCT S.sid FROM
Sailors S, Sailors S2 WHERE
S.rating > S2.rating AND
S2.name = 'John';`



GROUP BY

- Conceptual evaluation
 - Partition data into groups according to some criterion
 - Evaluate the aggregate for each group

Example: *For each rating level, find the age of the youngest sailor*

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

How many tuples
in the result?

GROUP BY and HAVING

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

Target-list contains:

- Attribute names (subset of grouping-list)
- Aggregate operations (e.g., min(age))

Conceptual Evaluation:

1. Eliminate tuples that don't satisfy qualification
2. Partition remaining data into groups
3. Eliminate groups according to group-qualification
4. Evaluate aggregate operation(s) for each group

Find the age of the youngest sailor with age ≥ 18 for each rating, such that there are at least 2 such sailors for that rating.

```
SELECT S.rating, MIN (S.age)
FROM Sailors S
WHERE S.age  $\geq$  18
GROUP BY S.rating
HAVING COUNT (*)  $\geq$  2
```

| <u>sid</u> | 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 |

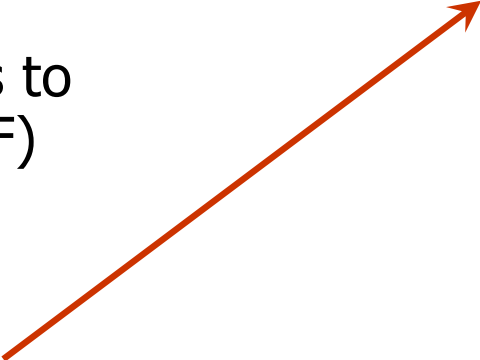
| rating | age |
|--------|------|
| 1 | 33.0 |
| 7 | 45.0 |
| 7 | 35.0 |
| 8 | 55.5 |
| 10 | 35.0 |

| rating | |
|--------|------|
| 7 | 35.0 |

Answer relation

NULL Values in SQL

- NULL represents ‘unknown’ or ‘inapplicable’
- Query evaluation complications
 - Q: Is (rating > 10) true when rating is NULL?
 - A: Condition evaluates to ‘unknown’ (not T or F)
- What about AND, OR connectives?
 - Need 3-valued logic
- WHERE clause eliminates rows that don't evaluate to true



| 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 |



NULL Values Example

What does this query return?

```
SELECT sname
FROM sailors
WHERE age > 45
      OR age <= 45
```

sailors

| sid | sname | rating | age |
|-----|--------|--------|------|
| 22 | dustin | 7 | 45 |
| 58 | rusty | 10 | NULL |
| 31 | lubber | 8 | 55 |



NULL Values in Aggregates

- NULL values are generally ignored when computing aggregates

```
SELECT AVG(age)  
FROM sailors
```

Returns 50!

sailors

| sid | sname | rating | age |
|-----|--------|--------|------|
| 22 | dustin | 7 | 45 |
| 58 | rusty | 10 | NULL |
| 31 | lubber | 8 | 55 |



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

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

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

-- Would this work?

--note: one color per bid



Subtle Errors

- Find the sid of sailors who have reserved exactly one boat.

```
(SELECT S1.sid FROM Sailors S1)
MINUS
(SELECT R1.sid
FROM
    Reserves R1, Reserves R2 WHERE
    R1.sid=R2.sid AND R1.bid <> R2.bid);
```

There is a subtle error in the above.



Error fixed

- Find the sid of sailors who have reserved exactly one boat.

```
SELECT R3.sid
FROM Reserves R3
MINUS
SELECT R1.sid
FROM
    Reserves R1, Reserves R2
WHERE
    R1.sid=R2.sid AND R1.bid <> R2.bid;
```



Error fixed: Another Solution

- Find the sid of sailors who have reserved exactly one boat.

```
SELECT R.sid FROM Reserves R  
GROUP BY R.sid  
HAVING COUNT(*) = 1;
```



Intersect on Non-Key

- Find the names of sailors who have reserved a red and a green boat

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

What is wrong with the above query?



Error fixed

- Find the names of sailors who have reserved a red and a green boat

```
CREATE VIEW RedGreenSailors AS
(SELECT S.sid
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red')
INTERSECT
(SELECT S.sid
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid and R.bid = B.bid AND B.color = 'green');

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

DROP VIEW RedGreenSailors;
```



Error fixed: another solution

- Find the names of sailors who have reserved a red and a green boat. Get rid of the VIEW.

```
SELECT S.sname FROM
Sailors S,
(SELECT S.sid
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
INTERSECT
SELECT S.sid
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid and R.bid = B.bid AND B.color = 'green')
RedGreenSailors WHERE
S.sid = RedGreenSailors.sid;
```

Another sol:
See Q8 in
Ch. 5, p. 150



**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 S2.rating=S.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 – may fail in practice):

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

If above does not work, one solution is to define T as a view .



Solution Using Views

```
CREATE VIEW AVG_AGE_BY_RATING AS  
SELECT S.rating, AVG(S.age) AS avgage  
FROM Sailors S GROUP BY S.rating;
```

```
SELECT T.rating, T.avgage FROM  
AVG_AGE_BY_RATING T  
WHERE T.avgage= (SELECT MIN(A.avgage) FROM  
AVG_AGE_BY_RATING A);
```



Suggested Review

- Exercises 5.1, 5.3, 5.7