Introduction to Database Systems

2023-Fall

3. User Interfaces and SQL Language

QL 2

Basic SQL Query

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

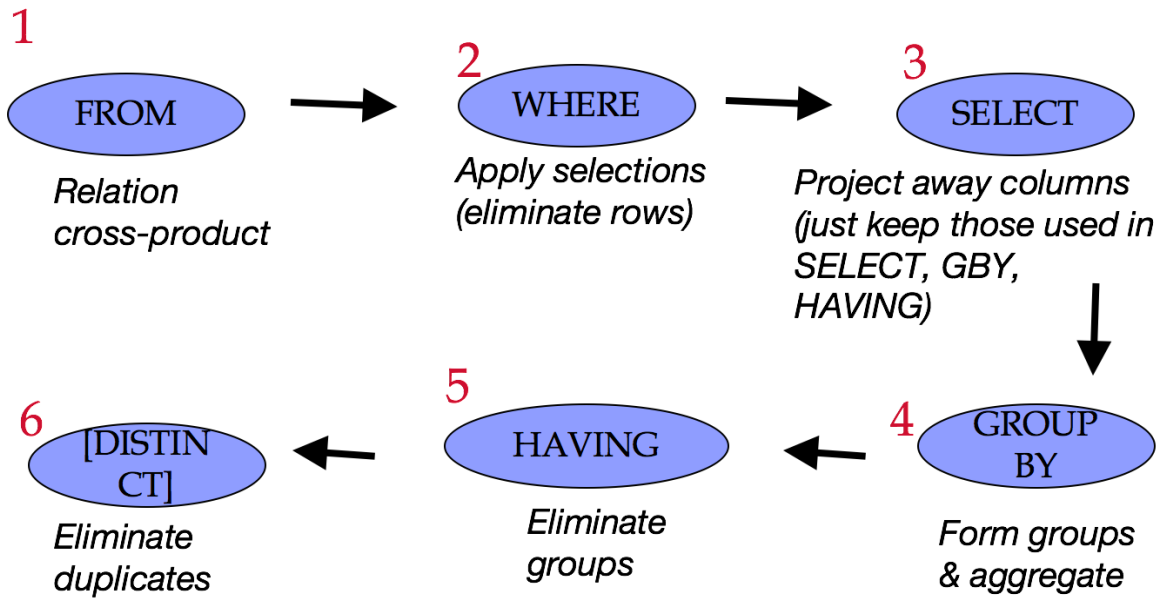
- *target-list* A list of attributes of relations in *relation-list*
- **DISTINCT** is an optional keyword indicating that the answer should not contain duplicates. Default is that duplicates are *not* eliminated!
- *relation-list* A list of relation/table names (possibly with a *range-variable* after each name).
- *qualification* Comparisons combined using AND, OR and NOT.

Conceptual Evaluation Strategy

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

- 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*.
  - If DISTINCT is specified, eliminate duplicate rows.
- This strategy is probably the least efficient way to compute a query! An optimizer will find more efficient strategies to compute *the same answers*.

Conceptual SQL Evaluation



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

Simple Example

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

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	**rusty**	10	35.0	58	103	11/12/96

Join Queries

SELECT [DISTINCT] <*column expression list*>
FROM <*table1 [AS t1], ... , tableN [AS tn]*>
[WHERE <*predicate*>]
[GROUP BY <*column list*>[HAVING <*predicate*>]]
[ORDER BY <*column list*>];

Cross (Cartesian) Product

- All pairs of tuples, concatenated

Sailors

sid	sname	rating	age
1	Popeye	10	22
2	OliveOyl	11	39
3	Garfield	1	27
4	Bob	5	19

Reserves

sid	bid	day
1	102	9/12
2	102	9/13
1	101	10/01

sid	sname	rating	age	sid	bid	day
1	Popeye	10	22	1	102	9/12
1	Popeye	10	22	2	102	9/13
1	Popeye	10	22	1	101	10/01
2	OliveOyl	11	39	1	102	9/12
…	…	…	…	…	…	…

Find sailors who've reserved at least one boat

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

sid	sname	rating	age
1	Popeye	10	22
2	OliveOyl	11	39
3	Garfield	1	27
4	Bob	5	19

sid	bid	day
1	102	9/12
2	102	9/13
1	101	10/01

sid	sname	rating	age	sid	bid	day
1	Popeye	10	22	1	102	9/12
1	Popeye	10	22	2	102	9/13
1	Popeye	10	22	1	101	10/01
2	OliveOyl	11	39	1	102	9/12
…	…	…	…	…	…	…

Find sailors who've reserved at least a boat cont

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

sid	sname	rating	age
1	Popeye	10	22
2	OliveOyl	11	39
3	Garfield	1	27
4	Bob	5	19

sid	bid	day
1	102	9/12
2	102	9/13
1	101	10/01

sid	sname	bid
1	Popeye	102
1	Popeye	101
2	OliveOyl	102

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?

Column Names and Table Aliases

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

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

More Aliases

```
SELECT x.sname, x.age,
       y.sname AS sname2,
       y.age AS age2
FROM Sailors AS x, Sailors AS y
WHERE x.age > y.age
```

- Table aliases in the FROM clause
  - Needed when the same table used multiple times ("***self-join***")
- Column aliases in the SELECT clause

Combining Predicates

- Subtle connections between:
  - Boolean logic in WHERE (i.e., AND, OR)
  - Traditional Set operations (i.e. INTERSECT, UNION)
- Let's see some examples…

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

- UNION: Can be used to compute the union of any two *union-compatible* sets of tuples (which are themselves the result of SQL queries).
- If we replace OR by AND in the first version, what do we get?
- Also available: EXCEPT (What do we get 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'
```

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

- INTERSECT: Can be used to compute the intersection of any two *union-compatible* sets of tuples.
- Included in the SQL/92 standard, but some systems don't support it.
- Contrast symmetry of the UNION and INTERSECT queries with how much the other versions differ.

```
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'
```

Find sid's of sailors who've reserved a red or a green boat Pt 2

```
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'
```

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

```
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  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'
```

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

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

INTERSECT

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

Find 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
```

Set Semantics

- Set: a collection of distinct elements
- Standard ways of manipulating/combining sets
  - Union
  - Intersect
  - Except
- Treat tuples within a relation as elements of a set

Default: Set Semantics

R = {A, A, A, A, B, B, C, D}
S = {A, A, B, B, B, C, E}

- UNION
  {A, B, C, D, E}
- INTERSECT
  {A, B, C}
- EXCEPT
  {D}

Note: Think of each letter as being a **tuple** in **relation.**

ex:
A: (Jim, 18, English, 4.0)
B: (Marcela , 20, CS, 3.8)
C: (Gail, 19, Statistics, 3.74)
D: (Goddard, 20, Math, 3.8

“ALL”: Multiset Semantics

R = {A, A, A, A, B, B, C, D} = {A(4), B(2), C(1), D(1)}
S = {A, A, B, B, B, C, E} = {A(2), B(3), C(1), E(1)}

“UNION ALL”: Multiset Semantics

R = {A, A, A, A, B, B, C, D} = {A(4), B(2), C(1), D(1)}
S = {A, A, B, B, B, C, E} = {A(2), B(3), C(1), E(1)}

- UNION ALL: sum of cardinalities
  {A(4+2), B(2+3), C(1+1), D(1+0), E(0+1)}
  = {A, A, A, A, A, A, B, B, B, B, B, C, C, D, E}

“INTERSECT ALL”: Multiset Semantics

R = {A, A, A, A, B, B, C, D} = {A(4), B(2), C(1), D(1)}
S = {A, A, B, B, B, C, E} = {A(2), B(3), C(1), E(1)}

- INTERSECT ALL: min of cardinalities
  {A(min(4,2)), B(min(2,3)), C(min(1,1)),
  D(min(1,0)), E(min(0,1))}
  = {A, A, B, B, C}

“EXCEPT ALL”: Multiset Semantics

R = {A, A, A, A, B, B, C, D} = {A(4), B(2), C(1), D(1)}
S = {A, A, B, B, B, C, E} = {A(2), B(3), C(1), E(1)}

- EXCEPT ALL: difference of cardinalities
  {A(4-2), B(2-3), C(1-1), D(1-0), E(0-1)}
  = {A, A, D, }

Nested Queries

Find names of sailors who've reserved boat #103:

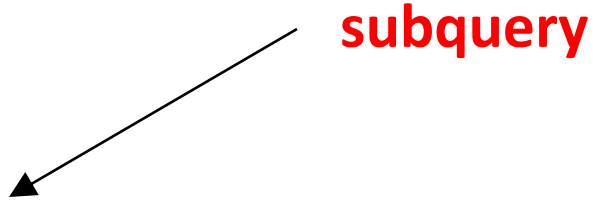
```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN (SELECT R.sid
                FROM Reserves R
                WHERE R.bid=103)
```

- A very powerful feature of SQL: a **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.
- To understand semantics of nested queries, think of a *nested loops* evaluation: *For each Sailors tuple, check the qualification by computing the subquery.*

Nested Queries: IN

- *Names of sailors who've reserved boat #102:*

```
SELECT S.sname
FROM   Sailors S
WHERE  S.sid IN
    (SELECT  R.sid
    FROM    Reserves R
    WHERE   R.bid=102)
```



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: EXISTS

- *This is a bit odd, but it is legal:*

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

Nested Queries with Correlation

Find 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)
```

- EXISTS is another set comparison operator, like IN.
- Illustrates why, in general, subquery must be ***re-computed*** for each Sailors tuple.
- How to *find names of sailors who've reserved boat #103 and reserved only one time*?

Nested Queries with Correlation

- ***Find IDs of boats which are reserved by only one sailor.***

```
SELECT bid
  FROM Reserves R1
  WHERE bid NOT IN (
            SELECT bid
            FROM Reserves R2
            WHERE R2.sid ¬= R1.sid)
```

More on Set-Comparison Operators

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

- ***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 INTERSECT Queries Using IN

Find sid's of sailors who've reserved both 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'
          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')
```

- Similarly, EXCEPT queries re-written using NOT IN.
- To find *names* (not *sid*'s) of Sailors who've reserved both red and green boats, just replace *S.sid* by *S.sname* in SELECT clause. (What about INTERSECT query?)

Division in SQL

Find sailors who've reserved all boats.

Solution 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))
```

All boats

Boat reserved by a sailor

Division in SQL

Solution 2:

Let's do it the hard way, without EXCEPT:

```
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

Queries With GROUP BY and HAVING

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

- The ***target-list*** contains
  - (i) attribute names
  - (ii) terms with aggregate operations (e.g., MIN (*S.age*)).
- The attribute list (i) must be a subset of ***grouping-list***. Intuitively, each answer tuple corresponds to a ***group***, and *these attributes must have a single value per group.* (A *group* is a set of tuples that have the same value for all attributes in *grouping-list*.)

Find age of the youngest sailor with age ≥ 18, for each rating 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
```

Answer relation:

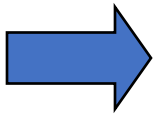
rating	minage
3	25.5
7	35.0
8	25.5

Sailors instance:

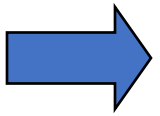
sid	sname	rating	age
22	dustin	7	45.0
29	brutus	1	33.0
31	lubber	8	55.5
32	andy	8	25.5
58	rusty	10	35.0
64	horatio	7	35.0
71	zorba	10	16.0
74	horatio	9	35.0
85	art	3	25.5
95	bob	3	63.5
96	frodo	3	25.5

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

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5



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

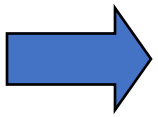


rating	minage
3	25.5
7	35.0
8	25.5

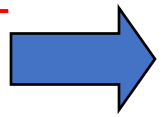
Find age of the youngest sailor with age ≥ 18, for each rating with at least 2 such sailors and with every sailor under 60.

HAVING COUNT (*) > 1 AND EVERY (S.age <=60)

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5



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



rating	minage
7	35.0
8	25.5

What is the result of changing EVERY to ANY?

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
```

- Grouping over a join of two relations.
- What do we get if we remove *B.color='red'* from the WHERE clause and add a HAVING clause with this condition?

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
```

- What do we get if we remove *B.color='red'* from the WHERE clause and add a HAVING clause with this condition?

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

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

rating	minage
3	25.5
7	35.0
8	25.5
10	35.5

- Shows HAVING clause can also contain a sub-query.
- Compare this with the query where ***we considered only ratings with 2 sailors over 18!***
- ***What if HAVING clause is replaced by:***
  - HAVING COUNT(*) >1

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

- Aggregate operations cannot be nested! WRONG:

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

  - Correct solution:

```
SELECT Temp.rating
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)
```

ARGMAX? Pt 1

- ***The sailor with the highest rating***
- Correct or Incorrect?

```
SELECT MAX(S.rating)
FROM Sailors S;
```

VS

```
SELECT S.*, MAX(S.rating)
FROM Sailors S;
```

ARGMAX? Pt 2

- ***The sailor with the highest rating***

- Correct or Incorrect? Same or different?

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

VS

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

ARGMAX? Pt 3

- ***The sailor with the highest rating***

- Correct or Incorrect? Same or different?

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

VS

```
SELECT *
FROM   Sailors S
ORDER BY rating DESC
LIMIT 1;
```

Null Values

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

NULL in the WHERE clause

- Consider a tuple where rating IS NULL.

```
 INSERT INTO sailors VALUES
  (11, 'Jack Sparrow', NULL, 35);

SELECT * FROM sailors
WHERE rating > 8;
```

Is Jack Sparrow in the output?

NULL in comparators

- ***Rule: (x op NULL) evaluates to … NULL!***

```
SELECT 100 = NULL;
SELECT 100 < NULL;
SELECT 100 >= NULL;
```

Explicit NULL Checks

```
SELECT * FROM sailors WHERE rating IS NULL;

SELECT * FROM sailors WHERE rating IS NOT NULL;
```

NULL at top of WHERE

- ***Rule: Do not output a tuple WHERE NULL***

```
SELECT * FROM sailors;
SELECT * FROM sailors WHERE rating > 8;
SELECT * FROM sailors WHERE rating <= 8;
```

NULL in Boolean Logic

Three-valued logic:

NOT	T	F	N
	F	T	

AND	T	F	N
T	T	F	
F	F	F	
N			

OR	T	F	N
T	T	T	
F	T	F	
N			

```
SELECT * FROM sailors WHERE rating > 8 AND TRUE;
SELECT * FROM sailors WHERE rating > 8 OR TRUE;

SELECT * FROM sailors WHERE NOT (rating > 8);
```

NULL in Boolean Logic

Three-valued logic:

NOT	T	F	N
	F	T	N

AND	T	F	N
T	T	F	N
F	F	F	F
N	N	F	N

OR	T	F	N
T	T	T	T
F	T	F	N
N	T	N	N

```
SELECT * FROM sailors WHERE rating > 8 AND TRUE;

SELECT * FROM sailors WHERE rating > 8 OR TRUE;

SELECT * FROM sailors WHERE NOT (rating > 8);
```

NULL and Aggregation

```
SELECT count(*) FROM sailors;

SELECT count(rating) FROM sailors;

SELECT sum(rating) FROM sailors;

SELECT avg(rating) FROM sailors;
```

General rule: NULL **column values are ignored by aggregate functions**

Aggregates with NULL

- What do you get for
- SELECT count(*) from R1?
- SELECT count(rating) from R1?

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

R1

Aggregates with NULL

- What do you get for
- SELECT count(*) from R1?
- SELECT count(rating) from R1?

- What do you get for
- SELECT count(*) from R2?
- SELECT count(rating) from R2?

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

R1

sid	sname	rating	age
22	dustin	7	45
31	lubber	null	55
58	rusty	10	35

R2

- COUNT, SUM, AVG, MIN, MAX (with or without DISTINCT) – Discards null values first
  – Then applies the aggregate
  – Except count(*)
- If only applied to null values, the result is null

sid	sname	rating	age
22	dustin	7	45
31	lubber	null	55
58	rusty	10	35

R2

SELECT sum(rating) from R2?

sid	sname	rating	age
22	dustin	null	45
31	lubber	null	55
58	rusty	null	35

R3

SELECT sum(rating) from R3?

NULLs: Summary

- NULL op NULL is NULL
- WHERE NULL: do not send to output
- Boolean connectives: 3-valued logic
- Aggregates ignore NULL-valued inputs