SQL Overview

- Query capabilities
  - SELECT-FROM-WHERE blocks,
  - Basic features, ordering, duplicates
  - Set operations (union, intersect, except)
  - Aggregation & Grouping
  - Nested queries (correlation)
  - Null values

Set operations

- UNION
- INTERSECTION
- EXCEPT (sometimes called MINUS)

- Recall: schemas must match for these operations.

UNION example

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

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

UNION

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

UNION

- Duplicates **ARE NOT** eliminated by default in basic SELECT-FROM-WHERE queries
- Duplicate **ARE** eliminated by default for UNION queries.
- To preserve duplicates in UNION, you must use UNION ALL

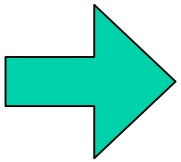
UNION example, alternative:

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

```
SELECT DISTINCT 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')
```

A small change in this query...

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



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

```
SELECT DISTINCT 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 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')
```

This doesn't work! What does this query return?

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

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

INTERSECT

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

SQL Overview

- Query capabilities
  - SELECT-FROM-WHERE blocks,
  - Basic features, ordering, duplicates
  - Set ops (union, intersect, except)
  - Aggregation & Grouping
  - Nested queries (correlation)
  - Null values

Aggregation

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

SQL supports several aggregation operations:

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

Aggregation: Count

```
SELECT  Count(*)
FROM    Sailors
WHERE   rating > 5
```

Except for COUNT, all aggregations apply to a single attribute

Aggregation: Count

COUNT applies to duplicates, unless otherwise stated:

```
SELECT  Count(category)
FROM    Product
WHERE   year > 1995
```

Better:

```
SELECT  Count(DISTINCT category)
FROM    Product
WHERE   year > 1995
```

Simple Aggregation

Purchase(product, date, price, quantity)

Example 1: find total sales for the entire database

```
SELECT  Sum(price * quantity)
FROM    Purchase
```

Example 1': find total sales of bagels

```
SELECT  Sum(price * quantity)
FROM    Purchase
WHERE   product = 'bagel'
```

GROUP BY and HAVING clauses

- We often want to apply aggregates to each of a number of groups of rows in a relation.

Find the age of the youngest sailor for each rating level.

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

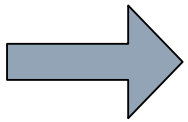
For i = 1, 2, ... 10

Grouping

Sailors

sid	sname	rating	age
29	brutus	1	33
85	art	3	25.5
95	bob	3	63.5
96	frodo	3	25.5
22	dustin	7	45
64	horatio	7	35
31	lubber	8	55.5
32	andy	8	25.5
74	horatio	9	35
58	rusty	10	35
71	zorba	10	16

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



New Table

rating	age?
1	
3	
7	
8	
9	
10	

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.

Conceptual Evaluation

- The cross-product of ***relation-list*** is computed, tuples that fail ***qualification*** are discarded, `*unnecessary'* fields are deleted, and the remaining tuples are partitioned into groups by the value of attributes in ***grouping-list***.
- The ***group-qualification*** is then applied to eliminate some groups. Expressions in *group-qualification* must have a ***single value per group*****!**
- One answer tuple is generated per qualifying group.

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

SQL Overview

- Query capabilities
  - SELECT-FROM-WHERE blocks,
  - Basic features, ordering, duplicates
  - Set ops (union, intersect, except)
  - Aggregation & Grouping
  - Nested queries (correlation)
  - Null values

Nested queries

- A **nested query** is a query with another query embedded within it.
- The embedded query is called the **subquery**.
- The subquery usually appears in the WHERE clause:

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

(Subqueries also possible in FROM or HAVING clause.)

Conceptual evaluation, extended

- For each row in cross product of outer query, evaluate the WHERE clause conditions, (re)computing the subquery.

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

equivalent to:

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

Correlated subquery

- If the inner subquery depends on tables mentioned in the outer query then it is a **correlated subquery.**
- In terms of conceptual evaluation, we must recompute subquery for each row of outer query.

Set-comparison operators

- Optional NOT may precede these:
  - EXISTS R -- true if R is non-empty
  - attr IN R -- true if R contains attr
  - UNIQUE R -- true if no duplicates in R
- For arithmetic operator op {<,<=,=,< >, >=,>}
  - attr op ALL R -- all elements of R satisfy condition
  - attr op ANY R -- some element of R satisfies condition

IN equivalent to = ANY
NOT IN equivalent to < > ALL

Example

- Find the sailors with the highest rating

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

Please write SQL

- Find sailors whose rating is higher than **some** sailor named Horatio.

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

- Find sailors whose rating is higher than **all** sailors named Horatio.

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

Find boats **not** reserved by sailor with sid = 100.

- B: all boats
- R: boats reserved by sailor with sid=100
- B – R is what we want.

```
SELECT B.bid
FROM Boats B
WHERE B.bid NOT IN (SELECT R.bid
                    FROM Reserves R
                    WHERE R.sid = 100 );
```

Existential conditions

- Find the names of sailors who have reserved **some** boat
- (i.e. *there exists* a boat they reserved)

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

- Existential conditions are natural and easy.

Universal conditions

- Find the names of sailors who have reserved **all** boats.
- (i.e. *for each* boat, they have reserved it.)
- Universal conditions are harder.

```
SELECT  S.sname
FROM    Sailors S
WHERE  NOT EXISTS  (
                        Set of boats not
                        reserved by S.sid
                   )
```

Universal conditions

- Find the names of sailors who have reserved **all** boats.

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

boats the sailor hasn't reserved
(we just wrote this query)

For each sailor, check that there is no boat s/he hasn't reserved.

Simulating INTERSECT

- Suppose we have tables R(a,b) and S(a,b)
- The following computes R ∩ S:

```
SELECT DISTINCT *
FROM R
WHERE (R.a, R.b) IN (SELECT *
                     FROM S );
```

This can be expressed without nesting:

- Given R(a,b), S(a,b), what is $R \bowtie S$?

  Intersection!

```
SELECT DISTINCT R.a, R.b
FROM R, S
WHERE R.a = S.a AND R.b = S.b;
```

*Find the names of sailors who reserved a red **and** a green boat.*

using INTERSECT

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

INTERSECT

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

without INTERSECT

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

"Find all sailors who have reserved a red boat and, further, have **sids** that are included in the set of **sids** of sailors who have reserved a green boat."

Simulating EXCEPT (set difference)

- What does this query compute?

```
SELECT DISTINCT *
FROM R
WHERE (R.a, R.b) NOT IN (SELECT *
                         FROM S );
```

Can this be expressed without a nested query? No.

(But this fact is not obvious)