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

i-clicker

What is the result of these two queries on the Sailors relation?

A

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

B

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

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

i-clicker

What is the result of these two queries on the Sailors relation?

A1

sid
31
32
74
58
71

A2

sid
31
32
74
58
71

B1

sid
22
64
31
32
74
58
71

B2

sid
74
58
71

C1

sid
31
32
74
58
71

C2

sid
58
71

D1

sid
74
58
71

D2

sid
74
58
71

E Honestly, I have no idea.

Answer on next slide

i-clicker

What is the result of these two queries on the Sailors relation?

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

A

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

B

- 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')
```

i-clicker

What is the result of these two queries on the Sailors relation?

A1

sid
31
32
74
58
71

A2

sid
31
32
74
58
71

B1

sid
22
64
31
32
74
58
71

B2

sid
74
58
71

C1

sid
31
32
74
58
71

C2

sid
58
71

D1

sid
74
58
71

D2

sid
74
58
71

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 in SQL.

Universal conditions

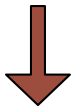
- Find the names of sailors who have reserved **all** boats.
- (i.e. *for each* boat, they have reserved it.)
- Universal conditions are harder...

```
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.

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



```
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 )
                   )
```

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

*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)

- Suppose we have tables R(a,b) and S(a,b)
  - 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?

Query monotonicity

- A query Q is monotonic if, adding tuples to an instance I can only grow the result of Q(I) but never shrink it.
  - Let I be any database instance.
  - Consider the instance that adds tuple t: I U {t}
  - Compare: Q(I) with Q(I U {t})
  - If Q is monotonic, then Q(I) $\subseteq$ Q(I U {t})

Monotonicity

- QUESTION: Is this query monotonic?

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

- FACT: All simple Select-From-Where queries (without subqueries) are monotonic.
- QUESTION: Can the query above be expressed without a nested query?

SQL Overview

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

NULLS in SQL

- Whenever we don't have a value for a cell, we can put a special value "**NULL**"
- Null can mean many things:
  - *Value does not exists*
  - *Value exists but is unknown*
  - *Value not applicable*
- The schema specifies for each attribute whether it is allowed to be null (*nullable* attribute)

```
CREATE TABLE Sailor (
  sid NOT NULL
  ... )
```

Outer Joins

- In a typical join, tuples of one relation that don't match any tuple from the other relation are omitted from result.
- In an outer join, **tuples without a match can be preserved** in the output.
- Missing values are filled with NULL.

Outer Joins

- LEFT OUTER JOIN: rows of **left** relation without matching row in right relation appear in result.
- RIGHT OUTER JOIN: rows of **right** relation without matching row in left relation appear in result.
- FULL OUTER JOIN: rows of both relations

Standard joins we have seen so far are called **inner joins** to distinguish them from outer joins.

Outer Joins

Sailors

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55.5
58	rusty	10	35

Reserves

sid	bid	day
22	101	10/10
58	103	10/10
58	105	12/1

Sailors LEFT OUTER JOIN Reserves:

sid	sname	rating	age	bid	day
22	dustin	7	45	101	10/10
31	lubber	8	55.5	NULL	NULL
58	rusty	10	35	103	10/10
58	rusty	10	35	105	12/1

Full Outer Join

(Unless otherwise specified, we assume the **natural** join)

R

A	B
22	foo
31	bar
58	wee

S

B	C
bar	X
wee	Y
yah	Z

Full Outer Join of R and S:

A	B	B	C
22	foo	NULL	NULL
31	bar	bar	X
58	wee	wee	Y
NULL	NULL	yah	Z

i-clicker

Compute quantities for each of the following joins on R and S:

```
(i)   Number of tuples in the crossproduct
(ii)  Number of tuples in the natural join
(iii) Number of tuples in the left outer join
(iv)  Number of tuples in the right outer join
(v)   Number of tuples in the full outer join
```

R

A	B
alfalfa	3
barley	5
chard	2
dandelion	1

S

B	C
5	X
3	Y
4	Z

What is the sum: (i) + (ii) + (iii) + (iv) + (v)

A. 20
B. 24
C. 26
D. 27
E. 28

Answer on next slide

i-clicker

Compute quantities for each of the following joins on R and S:

```
(i)   Number of tuples in the crossproduct        = 12
(ii)  Number of tuples in the natural join        = 2
(iii) Number of tuples in the left outer join     = 4
(iv)  Number of tuples in the right outer join    = 3
(v)   Number of tuples in the full outer join     = 5
```

R

A	B
alfalfa	3
barley	5
chard	2
dandelion	1

S

B	C
5	X
3	Y
4	Z

What is the sum: (i) + (ii) + (iii) + (iv) + (v)

A. 20
B. 24
C. 26
D. 27
E. 28

Alternative Join Specification in SQL

FROM Sailors S NATURAL JOIN Reserves R
FROM Sailors S JOIN Reserves R ON (S.sid = R.sid)
FROM Sailors S LEFT OUTER JOIN Reserves R ON (S.sid = R.sid)
FROM (Sailors S NATURAL JOIN Reserves R) NATURAL JOIN Boats B

Database Views

Views

- A **view** is a relation defined by a query.

- The query defining the view is called the **view definition**

Two Kinds of Views

Virtual View	Materialized View
the view relation is defined, but not computed or stored.	the view relation is computed and stored in system.
Computed only on-demand (slower at runtime).	Pre-computed offline (faster at runtime).
Always up to date	May have stale data
CREATE VIEW Developers AS SELECT .... FROM ... WHERE	CREATE TABLE Developers AS SELECT .... FROM ... WHERE

Virtual view example

Person(name, city)
Purchase(buyer, seller, product, store)
Product(name, maker, category)

```
CREATE VIEW  Seattle-view  AS

        SELECT  buyer, seller, product, store
        FROM    Person, Purchase
        WHERE   Person.city = "Seattle"    AND
                Person.name = Purchase.buyer
```

We have a new **virtual** table:
Seattle-view(buyer, seller, product, store)

View Example

We can use the view in a query as we would any other relation:

```
SELECT   name, store
FROM     Seattle-view, Product
WHERE    Seattle-view.product = Product.name  AND
         Product.category = “shoes”
```

Querying a virtual view

```
SELECT   name, Seattle-view.store
FROM     Seattle-view, Product
WHERE    Seattle-view.product = Product.name  AND
         Product.category = “shoes”
```

“View expansion”

```
SELECT  name, Purchase.store
FROM    Person, Purchase, Product
WHERE   Person.city = “Seattle”    AND
        Person.name = Purchase.buyer   AND
        Purchase.product = Product.name  AND
        Product.category = “shoes”
```

i-clicker

Given this view definition:

```
CREATE VIEW recent_reservations AS
SELECT S.sid, S.sname, R.sid, R.bid, R.day
FROM Sailors S, Reserves R
WHERE S.sid = R.sid AND R.day > 5
```

Compute the result of this query:

```
SELECT *
FROM recent_reservations V, Boats B
WHERE B.bid = V.bid AND B.color = red
```

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

Reserves

sid	bid	day
22	101	1
22	102	9
22	103	2
22	104	10
31	102	4
31	103	6
31	104	3
64	101	7
64	102	6
74	103	7

Boats

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

i-clicker

Given this view definition:

```
CREATE VIEW recent_reservations AS
SELECT S.sid, S.sname, R.sid, R.bid, R.day
FROM Sailors S, Reserves R
WHERE S.sid = R.sid AND R.day > 5
```

Compute the result of this query:

```
SELECT *
FROM recent_reservations V, Boats B
WHERE B.bid = V.bid AND B.color = red
```

How many tuples are there in the result?

A. 0
B. 1
C. 2
D. 3
E. 5

Answer on next slide

i-clicker

Given this view definition:

```
CREATE VIEW recent_reservations AS
SELECT S.sid, S.sname, R.sid, R.bid, R.day
FROM Sailors S, Reserves R
WHERE S.sid = R.sid AND R.day > 5
```

Compute the result of this query:

```
SELECT *
FROM recent_reservations V, Boats B
WHERE B.bid = V.bid AND B.color = red
```

How many tuples are there in the result?

A. 0
B. 1
C. 2
D. **3**
E. 5

The great utility of views

- Data independence (virtual)
- Efficient query processing (materialized)
  - materializing certain results can improve query execution
- Controlling access (virtual)
  - Grant access to views only to filter data
- Data integration (virtual)
  - Combine data sources using views

View-related issues

1. View selection
   - which views to materialize, given workload
2. View maintenance
   - when base relations change, (materialized) views need to be refreshed.
3. Updating virtual views
   - can users update relations that don’t exist?
4. Answering queries using views
   - when only views are available, what queries over base relations are answerable?

Updating Virtual Views

How can I insert a tuple into a table that doesn't exist?

Employee(ssn, name, department, project, salary)

```
CREATE VIEW  Developers AS
  SELECT name, project
  FROM  Employee
  WHERE department = "Development"
```

If we make the following insertion:

```
INSERT INTO  Developers  VALUES("Joe", "Optimizer")
```

It becomes:

```
INSERT INTO  Employee(ssn, name, department, project, salary)
VALUES(NULL, "Joe", "Development", "Optimizer", NULL)
```

Non-Updatable Views

Person(name, city)
Purchase(buyer, seller, product, store)

```
CREATE VIEW  City-Store  AS

        SELECT Person.city, Purchase.store
        FROM     Person, Purchase
        WHERE  Person.name = Purchase.buyer
```

How can we add the following tuple to the view?

("Seattle", "Nine West")

We don't know the name of the person who made the purchase; cannot set to NULL.

Another troublesome example

```
CREATE VIEW  OldEmployees AS
   SELECT name, age
   FROM  Employee
   WHERE age > 30
```

```
INSERT INTO  OldEmployees  VALUES("Joe", 28)
```

If this tuple is inserted into the view, it won't appear. Allowed by default in SQL!

Ambiguous updates

Name	group
Alice	fac
Bob	fac
Bob	cvs

group	file
fac	foo.txt
fac	bar.txt
cvs	foo.txt

Join →

view

Alice	foo.txt
Alice	bar.txt
Bob	foo.txt
Bob	bar.txt

Delete (“Alice”, “foo.txt”)

Updating views in practice

- Updates on views highly constrained:
  - SQL-92: updates only allowed on single-table views with projection, selection, no aggregates.
  - SQL-99: takes into account primary keys; updates on multiple table views may be allowed.
  - SQL-99: distinguishes between updatable and insertable views