SQL

The plan for today

- SQL Basics
- UNION, INTERSECT, EXCEPT
- Nested Queries
- ANY, ALL operators
- Aggregate Operators
- Some SQL Examples

Querying Concepts

Querying should be:

- Reliable: provides correct results
- Efficient: Computation of results should be fast

Queries can be defined using:

- Relational Algebra
- Specialized Query Languages

Query Languages

- Specialized language for Querying the data in a database
- In this class, we will study the query language called SQL (Structured Query Language) used in most DBMSs.

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

Table Definitions

We will be using the following relations in our examples:

Sailors(sid:integer, sname:string, rating:integer, age:real)

Boats(bid:integer, bname:string, color:string)

Reserves(sid:integer, bid:integer, day:date)

Relation Instances…1

An Instance of *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

Relation Instances…2

An Instance of *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

Relation Instances…3

An Instance of *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

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

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
Horatio	35.0
Art	25.5
Bob	63.5

Appears only once

Example SQL Query…1

Find the names of sailors who have reserved boat 103

<u>Relational Algebra</u>:

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

<u>SQL</u>:

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

OR

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

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

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
```

(Would adding DISTINCT to this query make a difference?)

(What is the effect of replacing *S.sname* by *S.sid* in the SELECT clause? Would adding DISTINCT to this variant of the query make a difference?)

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.
- INTERSECT: Can be used to compute the intersection of any two *union-compatible* sets of tuples.

Illustration of UNION…1

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

Intuitively, we would write:

```
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')
```

Illustration of UNION…2

We can also do this using a UNION keyword:

```
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'
```

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  S.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??)

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.)

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'
```

Nested Queries

- A **nested** query is a query that has another query embedded within it; this embedded query is called the **subquery**.
- Subqueries generally occur within the WHERE clause (but can also appear within the FROM and HAVING clauses)
- Nested queries are a very powerful feature of SQL. They help us write short and efficient queries.

(Think of nested **for** loops in C++. Nested queries in SQL are similar)

Example of a Nested Query

Find names of sailors who have reserved boat 103

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

Another Example of a Nested Query

*Find names of sailors who **have 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 )
```

Correlated Nested Queries…1

- Thus far, we have seen nested queries where the inner subquery is independent of the outer query.
- We can make the inner subquery **depend** on the outer query. This is called <u>correlation</u>.

Correlated Nested Queries…2

Find names of sailors who have reserved boat 103

Tests whether the set is nonempty

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

(For finding sailors who have **not** reserved boat 103, we would use NOT EXISTS)

UNIQUE operator

- When we apply UNIQUE to a subquery, it returns **true** if no row is duplicated in the answer to the subquery.
- What would the following SQL query return?

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

(All sailors with at most one reservation for boat 103.)

ANY and ALL operators

Find sailors whose rating is better than some sailor named Horatio

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

(Can you find the probable bug in this SQL query??) Hint: what if there're several sailors named Horatio?

Using ALL operator

*Find sailors whose rating is better than **every** sailor named Horatio*

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

Aggregate operators

- What is aggregation?
  - Computing arithmetic expressions, such as **Minimum** or **Maximum**

- The aggregate operators supported by SQL are:
  COUNT, SUM, AVG, MIN, MAX

Aggregate Operators

- **COUNT**(A): The number of values in the column A
- **SUM**(A): The sum of all values in column A
- **AVG**(A): The average of all values in column A
- **MAX**(A): The maximum value in column A
- **MIN**(A): The minimum value in column A

(We can use DISTINCT with COUNT, SUM and AVG to compute only over non-duplicated columns)

Using the COUNT operator

Count the number of sailors

```
SELECT  COUNT (*)
FROM  Sailors S
```

Example of SUM operator

Find the sum of ages of all sailors with a rating of 10

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

Example of AVG operator

Find the average age of all sailors with rating 10

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

(Shouldn’t we use DISTINCT in this case to take care of duplicated sailor ages??)

Example of MAX operator

Find the name and age of the oldest sailor

```
SELECT S.sname, MAX(S.age)
FROM Sailors S
```

But this is illegal in SQL!!

Correct SQL Query for MAX

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

Another Aggregate Query

Count the number of different sailors

```
SELECT COUNT (DISTINCT S.sname)
FROM Sailors S
```

BETWEEN and AND operators

- The **BETWEEN** and **AND** operator selects a range of data between two values.
- These values can be numbers, text, or dates.

BETWEEN and **AND** Example

Find the names of sailors whose age is between 25 and 35

```
SELECT sname
FROM Sailors
WHERE age BETWEEN 25 AND 35
```

SQL Examples…1

What does the following SQL Query do:

```
SELECT sid, sname
FROM Sailors
WHERE age= 25.5
```

Finds the sids and names of all sailors whose age is 25.5 years

SQL Examples…2

```
SELECT COUNT(*)
FROM Sailors
WHERE age = 25.5
```

Counts the number of sailors whose age is 25.5

SQL Examples…3

```
SELECT *
FROM Sailors
WHERE sname NOT BETWEEN  'Hansen' AND 'Pettersen'
```

*Finds all sailors whose name is **not** (alphabetically) between Hansen and Pettersen*

SQL Examples…4

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

```
SELECT *
FROM Sailors
WHERE sname LIKE 'A%'
```

sid	*sname*	*rating*	*age*
32	Andy	8	25.5
85	Art	3	25.5

Finds all sailors whose name begins with 'A' and is at least 1-character long

SQL Examples…5

```
SELECT SUM(age)
FROM Sailors
WHERE age>20
```

Finds the sum of ages of all sailors whose age is greater than 20

SQL Examples…6

```
SELECT MIN(age)
FROM Sailors
WHERE age>20
```

Finds the minimum age from the ages of all sailors whose age is greater than 20

SQL Examples…7

What does the following SQL query do:

```
SELECT S.sname, MAX(S.age)
FROM Sailors S
```

(Isn't this illegal in SQL??) Nested query must be used here

More to come…

Advanced SQL concepts :

- GROUP BY
- ORDER BY
- HAVING