

DSAA 5012

Advanced Data Management for Data Science

LECTURE 7

STRUCTURED QUERY LANGUAGE (SQL)



STRUCTURED QUERY LANGUAGE (SQL): **OUTLINE**

- ✓ SQL Basic Structure and Operations
- ✓ Additional Basic Operations
- ➔ **Aggregate Functions**
 - **Group By Clause**
 - **Having Clause**

Nested Subqueries and Set Operations

Database Definition

Database Modification

Using SQL in Applications

EXAMPLE BANK RELATIONAL SCHEMA

Branch(branchName, district, assets)

Client(clientId, name, address, district)

Loan(loanNo, amount, *branchName*)

Account(accountNo, balance, *branchName*)

Borrower(clientId, *loanNo*)

Depositor(clientId, *accountNo*)

Attribute names in italics are foreign key attributes.

AGGREGATE FUNCTIONS

- An aggregate function **operates on an attribute** of a relation and **returns a single value** (i.e., a table with one row and one column).

count number of tuples / values

avg average value

stdev standard deviation of values

max maximum value

sum sum of values (total)

min minimum value

- For **avg**, **stdev** and **sum** the **input must be numbers**.
- For **other functions**, the **input can be non-numeric** (e.g., strings).
- All aggregate functions, except **count(*)**, **ignore null values** in the input collection and **return a value of null** for an empty collection.

 **The count of an empty collection is defined to be 0.**

AGGREGATE FUNCTIONS: COMPUTATION

Query: Find the average account balance at the Pacific Place branch.

```
select avg(balance) as avgBalance
from Account
where branchName='Pacific Place';
```

Select the tuples where branch name equals Pacific Place.

Project on the balance attribute retaining duplicates.

Calculate the average.

avgBalance
83333.33

Account

accountNo	balance	branchName
A-102	40000	Star House
A-222	75000	Pacific Place
A-201	90000	Star House
A-215	75000	Pacific Place
A-217	7500	Langham Place
A-224	100000	Pacific Place

accountNo	balance	branchName
A-222	75000	Pacific Place
A-215	75000	Pacific Place
A-224	100000	Pacific Place

balance
75000
75000
100000

avg(balance)



AGGREGATE FUNCTIONS: EXAMPLES

Query: Find the number of accounts.

```
select count(*)  
from Account;
```

 **Remember * stands for all attributes.**

accountNo	balance	branchName
A-102	40000	Star House
A-222	75000	Pacific Place
A-201	90000	Star House
A-215	75000	Pacific Place
A-217	7500	Langham Place
A-224	100000	Pacific Place

Same as:

```
select count(branchName)  
from Account;
```

Why?

Different from:

```
select count(distinct branchName)  
from Account;
```

Why?

```
Cannot say:  
select count(distinct *)  
from Account;
```

SQL does not allow the use of distinct with count(*).



GROUP BY CLAUSE

A **group by** clause permits **aggregate results** to be displayed (e.g., max, min, sum, etc.) **for groups**. For example, **group by x** will get a result for every **different** value of **x**.

👉 **Aggregate queries without group by return a single number.**

Query: Find the number of accounts for *each* branch.

```
select branchName, count(*)  
from Account  
group by branchName;
```



GROUP BY CLAUSE (cont'd)

Query: Find the number of accounts for *each* branch.

```
select branchName, count(*)  
from Account  
group by branchName;
```

Group the tuples
by branch name.

For each group, count the
number of tuples in the group.

Account

accountNo	balance	branchName
A-102	40000	Star House
A-222	75000	Pacific Place
A-201	90000	Star House
A-215	75000	Pacific Place
A-217	7500	Langham Place
A-224	100000	Pacific Place

accountNo	balance	branchName
A-102	40000	Star House
A-201	90000	Star House
A-222	75000	Pacific Place
A-215	75000	Pacific Place
A-224	100000	Pacific Place
A-217	7500	Langham Place

branchName	count(*)
Star House	2
Pacific Place	3
Langham Place	1



GROUP BY CLAUSE: ATTRIBUTES

Query: Find the balance and the number of accounts for *each* branch.

```
select branchName, balance, count(*)  
from Account  
group by branchName;
```

accountNo	balance	branchName
A-102	40000	Star House
A-201	90000	Star House
A-222	75000	Pacific Place
A-215	75000	Pacific Place
A-224	100000	Pacific Place
A-217	7500	Langham Place

Illegal! Why?

An attribute in the **select** clause **must** also appear in the **group by** clause.

The opposite is not true!

Attributes in the **group by** clause **do not** need to appear in the **select** clause.



GROUP BY CLAUSE: ATTRIBUTES (cont'd)

Query: Find the balance and the number of accounts for *each* branch.

```
select branchName, balance, count(*)  
from Account  
group by branchName, balance;
```

branchName	balance	count
Star House	40000	1
Star House	90000	1
Pacific Place	75000	2
Pacific Place	100000	1
Langham Place	7500	1

Either is
legal SQL.

```
select branchName, sum(balance), count(*)  
from Account  
group by branchName;
```

branchName	sum (balance)	count
Star House	130000	2
Pacific Place	25000	3
Langham Place	7500	1



GROUP BY CLAUSE: WITH JOIN

Query: Find the number of depositors for each branch.

```
select branchName, count(distinct clientId)
from Depositor natural join Account
group by branchName;
```

JOIN ⇒ (clientId, accountNo, balance, branchName)

clientId	branchName
1	Star House
2	Pacific Place
4	Langham Place
3	Pacific Place
1	Star House
5	Pacific Place

group by

clientId	branchName
1	Star House
1	Star House
4	Langham Place
3	Pacific Place
2	Pacific Place
5	Pacific Place

distinct

clientId	branchName
1	Star House
4	Langham Place
3	Pacific Place
2	Pacific Place
5	Pacific Place

count

branchName	count
Star House	1
Langham Place	1
Pacific Place	3

 **Group by and aggregate functions apply to the join result.**



HAVING CLAUSE

The **having** clause allows a condition to be **applied to groups** rather than to individual tuples.

Query: Find the names and average balances of all branches where the average account balance is more than \$8000.

```
select branchName, avg(balance)
from Account
group by branchName
having avg(balance)>8000;
```

	accountNo	balance	branchName	
✓	A-102	40000	Star House	avg(65000.00)
	A-201	90000	Star House	
✓	A-222	75000	Pacific Place	avg(83333.33)
	A-215	75000	Pacific Place	
	A-224	100000	Pacific Place	
X	A-217	7500	Langham Place	avg(7500.00)

Any condition that appears in the **having** clause refers to the groups and is applied **after** the formation of the groups.

Any attribute in the **having** clause that is **not aggregated** must appear in the **group by** clause.



HAVING CLAUSE: EXAMPLE

Query: Find the branch names in Central and Western district where the average account balance is more than \$8000.

```
select branchName as branch
from Account natural join Branch
where district='Central and Western'
group by branchName
having avg(balance)>8000
```

First, find the records that satisfy the **where** clause predicate.

Then, form the groups (include only those tuples that satisfy the **where** clause predicate).

Finally, apply the **having** clause to *each group*.

SQL Evaluation Note

In most relational DBMSs, an alias defined in the **select** clause cannot be used in most other clauses. This is because the order of execution of a **select** statement is:

1. **from** clause
2. **where** clause
3. **group by** clause
4. **having** clause
5. **select** clause
6. **order by** clause

Consequently, a column or alias name defined in a clause executed later cannot be used in an earlier clause.



STRUCTURED QUERY LANGUAGE (SQL) EXERCISES 1, 2



EXAMPLE RELATIONAL SCHEMA AND DATABASE

Sailor(sailorId, sName, rating, age)

Boat(boatId, bName, color)

Reserves(sailorId, boatId, rDate)

Attribute names in italics are foreign key attributes.

Sailor

<u>sailorId</u>	sName	rating	age
22	Dustin	7	45
29	Brutus	1	33
31	Lubber	8	55
32	Andy	8	25
58	Rusty	10	35
64	Horatio	7	35
71	Zorba	10	16
74	Horatio	9	35
85	Art	3	25
95	Bob	3	63
99	Chris	10	30

11 tuples

Reserves

<u>sailorId</u>	<u>boatId</u>	<u>rDate</u>
22	101	10/10/17
22	102	10/10/17
22	103	08/10/17
22	104	07/10/17
31	102	10/11/17
31	103	06/11/17
31	104	12/11/17
64	101	05/09/17
64	102	08/09/17
74	103	08/09/17
99	104	08/08/17

11 tuples

Boat

<u>boatId</u>	bName	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red
105	Serenity	Cyan

5 tuples

EXERCISE 1

Find the boat name and the number of reservations for each red boat.

Sailor

<u>sailorId</u>	sName	rating	age
22	Dustin	7	45
29	Brutus	1	33
31	Lubber	8	55
32	Andy	8	25
58	Rusty	10	35
64	Horatio	7	35
71	Zorba	10	16
74	Horatio	9	35
85	Art	3	25
95	Bob	3	63
99	Chris	10	30

11 tuples

Reserves

<u>sailorId</u>	<u>boatId</u>	<u>rDate</u>
22	101	10/10/17
22	102	10/10/17
22	103	08/10/17
22	104	07/10/17
31	102	10/11/17
31	103	06/11/17
31	104	12/11/17
64	101	05/09/17
64	102	08/09/17
74	103	08/09/17
99	104	08/08/17

11 tuples

Boat

<u>boatId</u>	bName	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red
105	Serenity	Cyan

5 tuples

EXERCISE 1

Find the boat name and the number of reservations for each red boat.

👉 (Interlake, 3), (Marine, 3)

Is this a correct solution?

```
select bName count(*) as reservationCount
from Boat natural join Reserves
where color='red'
group by boatId;
```

Illegal!!!
Why?

👉 All non-aggregate attributes in the **select** clause **must** appear in the **group by** clause (i.e., **bName** must appear in the **group by** clause).

EXERCISE 1 (cont'd)

Find the boat name and the number of reservations for each red boat.

☞ (Interlake, 3), (Marine, 3)

```
select bName, count(*) as reservationCount
from Boat natural join Reserves
where color='red'
group by bName, boatId
```

sailorId	boatId	rDate	bName	color
22	102	10/10/17	Interlake	red
22	104	07/10/17	Marine	red
31	102	10/11/17	Interlake	red
31	104	12/11/17	Marine	red
64	102	08/09/17	Interlake	red
99	104	08/08/17	Marine	red

Reservations for red boats.

a group

a group

bName	reservationCount
Interlake	3
Marine	3

Name and count of the number of reservations for each red boat.



EXERCISE 1 (cont'd)

Find the boat name and the number of reservations for each red boat.

☞ (Interlake, 3), (Marine, 3)

Do you see any problems with this solution?

```
select bName, count(*) as reservationCount
from Boat natural join Reserves
where color='red'
group by bName
```

sailorId	boatId	rDate	bName	color
22	102	10/10/17	Interlake	red
22	104	07/10/17	Marine	red
31	102	10/11/17	Interlake	red
31	104	12/11/17	Marine	red
64	102	08/09/17	Interlake	red
99	104	08/08/17	Marine	red

Reservations for red boats.

a group

a group

bName	reservationCount
Interlake	3
Marine	3

Name and count of the number of reservations for each red boat.



EXERCISE 1 (cont'd)

Suppose we change the query to this.

Find the boat name and the number of reservations for each boat.

What is the result?

```
select bName, count(*) as reservationCount
from Boat natural join Reserves
group by bName
```

sailorId	boatId	rDate	bName	color
22	101	10/10/17	Interlake	blue
64	101	05/09/17	Interlake	blue
22	102	10/10/17	Interlake	red
31	102	10/11/17	Interlake	red
64	102	08/09/17	Interlake	red
22	103	08/10/17	Clipper	green
31	103	06/11/17	Clipper	green
74	103	08/09/17	Clipper	green
22	104	07/10/17	Marine	red
31	104	12/11/17	Marine	red
99	104	08/08/17	Marine	red

a group

a group

a group

bName	reservationCount
Interlake	5
Clipper	3
Marine	3

Since bName is not unique, grouping on it can get an incorrect result!



EXERCISE 1 (cont'd)

Find the boat name and the number of reservations for each boat.

Correct solution.

```
select bName, count(*) as reservationCount
from Boat natural join Reserves
group by bName, boatId
```

Recall: attributes in the group by clause do not have to appear in the select clause.

sailorId	boatId	rDate	bName	color
22	101	10/10/17	Interlake	blue
64	101	05/09/17	Interlake	blue
22	102	10/10/17	Interlake	red
31	102	10/11/17	Interlake	red
64	102	08/09/17	Interlake	red
22	103	08/10/17	Clipper	green
31	103	06/11/17	Clipper	green
74	103	08/09/17	Clipper	green
22	104	07/10/17	Marine	red
31	104	12/11/17	Marine	red
99	104	08/08/17	Marine	red

a group

a group

a group

a group

bName	reservationCount
Interlake	2
Interlake	3
Clipper	3
Marine	3



EXERCISE 2

Find the sailor id and number of reservations made for each sailor.

Sailor

<u>sailorId</u>	sName	rating	age
22	Dustin	7	45
29	Brutus	1	33
31	Lubber	8	55
32	Andy	8	25
58	Rusty	10	35
64	Horatio	7	35
71	Zorba	10	16
74	Horatio	9	35
85	Art	3	25
95	Bob	3	63
99	Chris	10	30

11 tuples

Reserves

<u>sailorId</u>	<u>boatId</u>	<u>rDate</u>
22	101	10/10/17
22	102	10/10/17
22	103	08/10/17
22	104	07/10/17
31	102	10/11/17
31	103	06/11/17
31	104	12/11/17
64	101	05/09/17
64	102	08/09/17
74	103	08/09/17
99	104	08/08/17

11 tuples

Boat

<u>boatId</u>	bName	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red
105	Serenity	Cyan

5 tuples

EXERCISE 2

Find the sailor id and number of reservations made for each sailor.

👉 (22, 4), (29, 0), (31, 3), (32, 0), (58, 0), (64, 2),
(71, 0), (74, 1), (85, 0), (95, 0), (99, 1)

```
select sailorId, count(sailorId) as reservationCount
from Reserves
group by sailorId;
```

How to include
all sailors?

sailorId	reservationCount
22	4
31	3
64	2
74	1
99	1

How about joining Sailor and Reserves?

```
select sailorId, count(sailorId) as reservationCount
from Sailor natural join Reserves
group by sailorId;
```

What's the
problem?

sailorId	reservationCount
22	4
31	3
64	2
74	1
99	1


EXERCISE 2 (cont'd)

Find the sailor id and number of reservations made for each sailor.

☞ (22, 4), (29, 0), (31, 3), (32, 0), (58, 0), (64, 2),
(71, 0), (74, 1), (85, 0), (95, 0), (99, 1)

sailorId	sName	rating	age	boatId	rDate
22	Dustin	7	45	101	10/10/17
22	Dustin	7	45	102	10/10/17
22	Dustin	7	45	103	08/10/17
22	Dustin	7	45	104	07/10/17
31	Lubber	8	55	102	10/11/17
31	Lubber	8	55	103	06/11/17
31	Lubber	8	55	104	12/11/17
64	Horatio	7	35	101	05/09/17
64	Horatio	7	35	102	08/09/17
74	Horatio	9	35	103	08/09/17
99	Chris	10	30	104	08/08/17
29	Brutus	1	33	-	-
32	Andy	8	25	-	-
58	Rusty	10	35	-	-
71	Zorba	10	16	-	-
85	Art	3	25	-	-
95	Bob	3	63	-	-

```
select sailorId, count(sailorId) as reservationCount
from Sailor natural join Reserves
group by sailorId;
```



sailorId	reservationCount
22	4
31	3
64	2
74	1
99	1

☞ Some Sailor tuples have no match in the Reserves relation.

How to deal with this problem?

EXERCISE 2 (cont'd)

Find the sailor id and number of reservations made for each sailor.

☞ (22, 4), (29, 0), (31, 3), (32, 0), (58, 0), (64, 2),
(71, 0), (74, 1), (85, 0), (95, 0), (99, 1)

```
select sailorId, count(boatId) as reservationCount
from Sailor natural left outer join Reserves
group by sailorId;
```

Recall: **left outer join** keeps all copies of the common attributes;
natural left outer join keeps only one copy of the common attributes.

Is this a
correct
solution?

No! Why?

```
select sailorId, count(sailorId) as reservationCount
from Sailor natural left outer join Reserves
group by sailorId;
```

Counting is done on the sailor ids and all
of them appear at least once in the result.

STRUCTURED QUERY LANGUAGE (SQL): **OUTLINE**

- ✓ SQL Basic Structure and Operations
- ✓ Additional Basic Operations
- ✓ Aggregate Functions
- ➔ **Nested Subqueries and Set Operations**
 - **Set Membership**
 - **Set Comparison**
 - **Empty Relation Test**
 - **Duplicate Tuples Test**
 - **With Clause**

Database Definition

Database Modification

Using SQL in Applications

NESTED SUBQUERIES

- Every SQL statement returns a relation as the result.
 - ☞ **A relation can be null or contain only a single, atomic value.**
- Consequently, a value or a set of values can be replaced with a SQL statement (i.e., with a subquery).
 - ☞ **The query is illegal if the subquery returns the wrong number of tuples or the wrong type for the comparison.**

```
select *
from Loan
where amount > 12000;
```

```
select *
from Loan
where amount > (select avg(amount)
from Loan);
```

This subquery *must* return a single, numeric value else it is *illegal*.

Subqueries are commonly used to test for set membership, do set comparison or determine set cardinality.

SET MEMBERSHIP: IN

Query: Find all clients who have both an account and a loan.

```

select distinct clientId
from Borrower
where clientId in (select clientId
                  from Depositor)
  
```

The **in** operator tests for the **presence** of set membership (i.e., selects clients in the Borrower set *only if* they are **in** the Depositor set).
Duplicates are retained.

The **set** of clients who have an account.

SET MEMBERSHIP: NOT IN

Query: Find all clients who have a loan, but do not have an account.

```
select distinct clientId
from Borrower
where clientId not in (select clientId
from Depositor);
```


The **not in** operator tests for the absence of set membership (i.e., selects clients in the Borrower set *only if* they are not in the Depositor set).
Duplicates are retained.

The **set** of clients who have an account.



SET COMPARISON: SOME

Query: Find the names of all branches that have greater assets than *some* (i.e., at least one) branch located in Central and Western.

 **Equivalent to:** Find the names of all branches that have greater assets than the **minimum** assets of any branch located in Central and Western.


```
select branchName
from Branch
where assets > some (select assets
from Branch
where district='Central and Western');
```

The **where** clause is true if the **assets** value of a **Branch** tuple is **greater than at least one member** of the set of all **assets** values of branches in Central and Western (i.e., **greater than the minimum assets** value in the set). **Duplicates are retained.**

The **set** of **assets** values of **all** branches in Central and Western.

SET COMPARISON: ALL

Query: Find the names of those branches that have greater assets than *all* branches located in Central and Western.

 **Equivalent to:** Find the names of all branches that have greater assets than the **maximum** assets of any branch located in Central and Western.

```
select branchName
from Branch
where assets > all (select assets
from Branch
where district='Central and Western');
```

The **where** clause is true if the **assets** value of a **Branch** tuple is greater than each of the members of the set of all **assets** values of branches in Central and Western (i.e., **greater than the maximum assets** value in the set). **Duplicates are retained.**

The **set** of **assets** values of all branches in Central and Western.

STRUCTURED QUERY LANGUAGE (SQL) EXERCISES 3, 4



EXERCISE 3

Find the records (tuples) of the sailors with the highest rating.

Sailor

<u>sailorId</u>	sName	rating	age
22	Dustin	7	45
29	Brutus	1	33
31	Lubber	8	55
32	Andy	8	25
58	Rusty	10	35
64	Horatio	7	35
71	Zorba	10	16
74	Horatio	9	35
85	Art	3	25
95	Bob	3	63
99	Chris	10	30

11 tuples

Reserves

<u>sailorId</u>	<u>boatId</u>	<u>rDate</u>
22	101	10/10/17
22	102	10/10/17
22	103	08/10/17
22	104	07/10/17
31	102	10/11/17
31	103	06/11/17
31	104	12/11/17
64	101	05/09/17
64	102	08/09/17
74	103	08/09/17
99	104	08/08/17

11 tuples

Boat

<u>boatId</u>	bName	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red
105	Serenity	Cyan

5 tuples

EXERCISE 3

Find the records (tuples) of the sailors with the highest rating.

☞ (58, Rusty, 10, 35), (71, Zorba, 10, 16), (99, Chris, 10, 30)

Is this a
correct
solution?
No! Why?

```
select *  
from Sailor  
where rating=max(rating);
```

There is no `max(rating)` value to compare in the where clause.
☞ The max rating value must be obtained by a select statement!

Is this a
correct
solution?
No! Why?

```
select *, max(rating)  
from Sailor;
```

A query that returns multiple tuples cannot contain an aggregate function.
☞ There are multiple tuples in the result, but only one max value!



EXERCISE 3 (cont'd)

Find the records (tuples) of the sailors with the highest rating.

👉 (58, Rusty, 10, 35), (71, Zorba, 10, 16), (99, Chris, 10, 30)

```
select *  
from Sailor  
where rating = (select max(rating)  
from Sailor);
```

sailorId	sName	rating	age
22	Dustin	7	45
29	Brutus	1	33
31	Lubber	8	55
32	Andy	8	25
58	Rusty	10	35
64	Horatio	7	35
71	Zorba	10	16
74	Horatio	9	35
85	Art	3	25
95	Bob	3	63
99	Chris	10	30

All sailors.

sailorId	sName	rating	age
58	Rusty	10	35
71	Zorba	10	16
99	Chris	10	30

Sailors with the maximum rating.

max(rating)
10

the maximum rating.

EXERCISE 3 (cont'd)

Use set membership

Find the records (tuples) of the sailors with the highest rating.

☞ (58, Rusty, 10, 35), (71, Zorba, 10, 16), (99, Chris, 10, 30)

```
select *  
from Sailor  
where rating >= all (select rating  
from Sailor);
```

sailorId	sName	rating	age
22	Dustin	7	45
29	Brutus	1	33
31	Lubber	8	55
32	Andy	8	25
58	Rusty	10	35
64	Horatio	7	35
71	Zorba	10	16
74	Horatio	9	35
85	Art	3	25
95	Bob	3	63
99	Chris	10	30

All sailors.

sailorId	sName	rating	age
58	Rusty	10	35
71	Zorba	10	16
99	Chris	10	30

Sailors with the highest rating.

rating
7
1
8
8
10
7
10
9
3
3
10

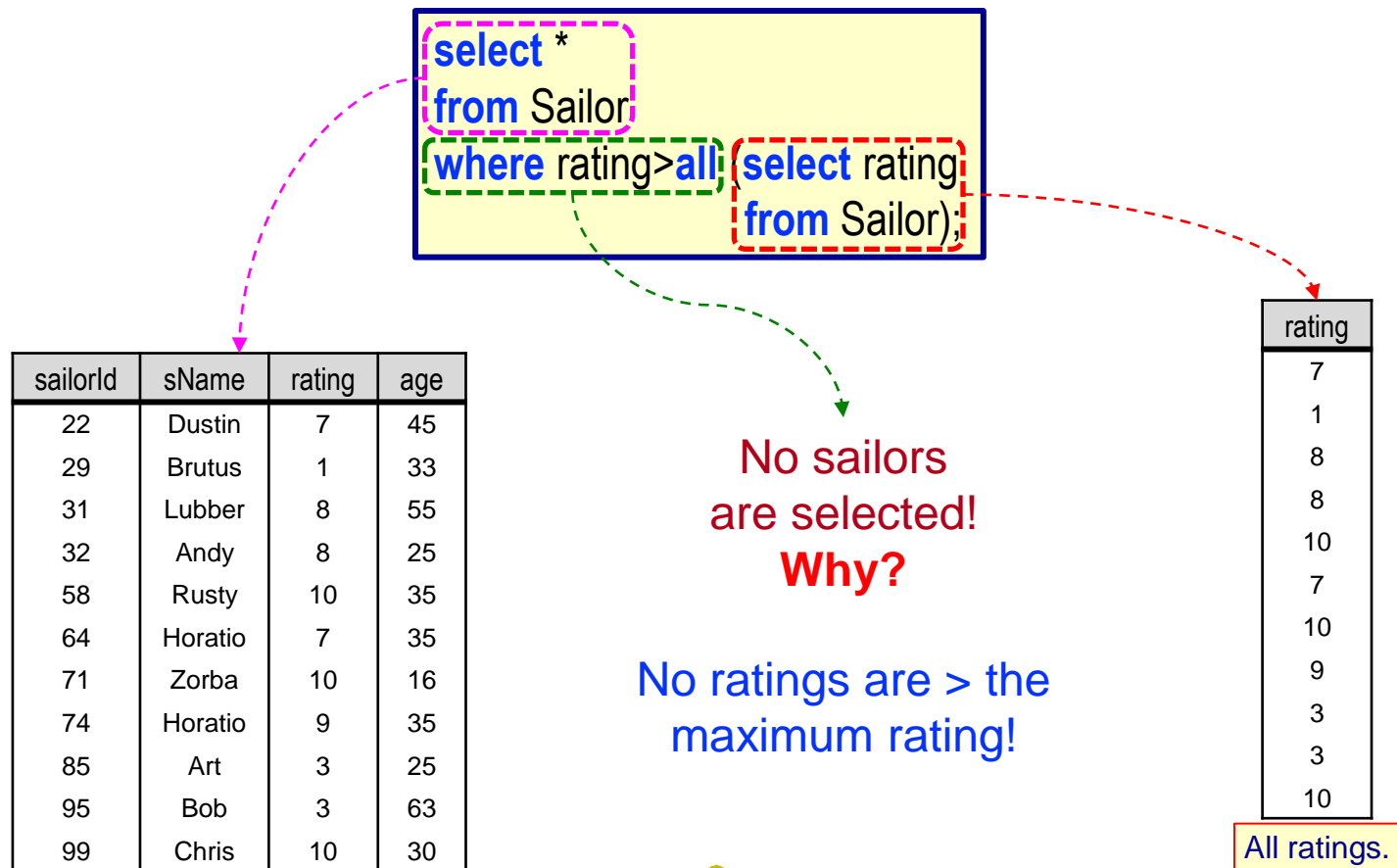
All ratings.



EXERCISE 3 (cont'd)

What is the result if we replace “>=all” with “>all”?

☞ Recall “>all” is equivalent to greater than the maximum.



EXERCISE 3 (cont'd)

What is the result if we replace “>=all” with “>=some”?

👉 Recall “>some” is equivalent to greater than the minimum.

```
select *  
from Sailor  
where rating >= some (select rating  
from Sailor);
```

sailorId	sName	rating	age
22	Dustin	7	45
29	Brutus	1	33
31	Lubber	8	55
32	Andy	8	25
58	Rusty	10	35
64	Horatio	7	35
71	Zorba	10	16
74	Horatio	9	35
85	Art	3	25
95	Bob	3	63
99	Chris	10	30

All sailors.

All sailors are
selected!
Why?

All ratings are >=
the minimum rating!

rating
7
1
8
8
10
7
10
9
3
3
10

All ratings.

EXERCISE 4

DO NOT
use JOIN

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

Use *only* set membership

Sailor

<u>sailorId</u>	sName	rating	age
22	Dustin	7	45
29	Brutus	1	33
31	Lubber	8	55
32	Andy	8	25
58	Rusty	10	35
64	Horatio	7	35
71	Zorba	10	16
74	Horatio	9	35
85	Art	3	25
95	Bob	3	63
99	Chris	10	30

11 tuples

Reserves

<u>sailorId</u>	<u>boatId</u>	<u>rDate</u>
22	101	10/10/17
22	102	10/10/17
22	103	08/10/17
22	104	07/10/17
31	102	10/11/17
31	103	06/11/17
31	104	12/11/17
64	101	05/09/17
64	102	08/09/17
74	103	08/09/17
99	104	08/08/17

11 tuples

Boat

<u>boatId</u>	bName	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red
105	Serenity	Cyan

5 tuples

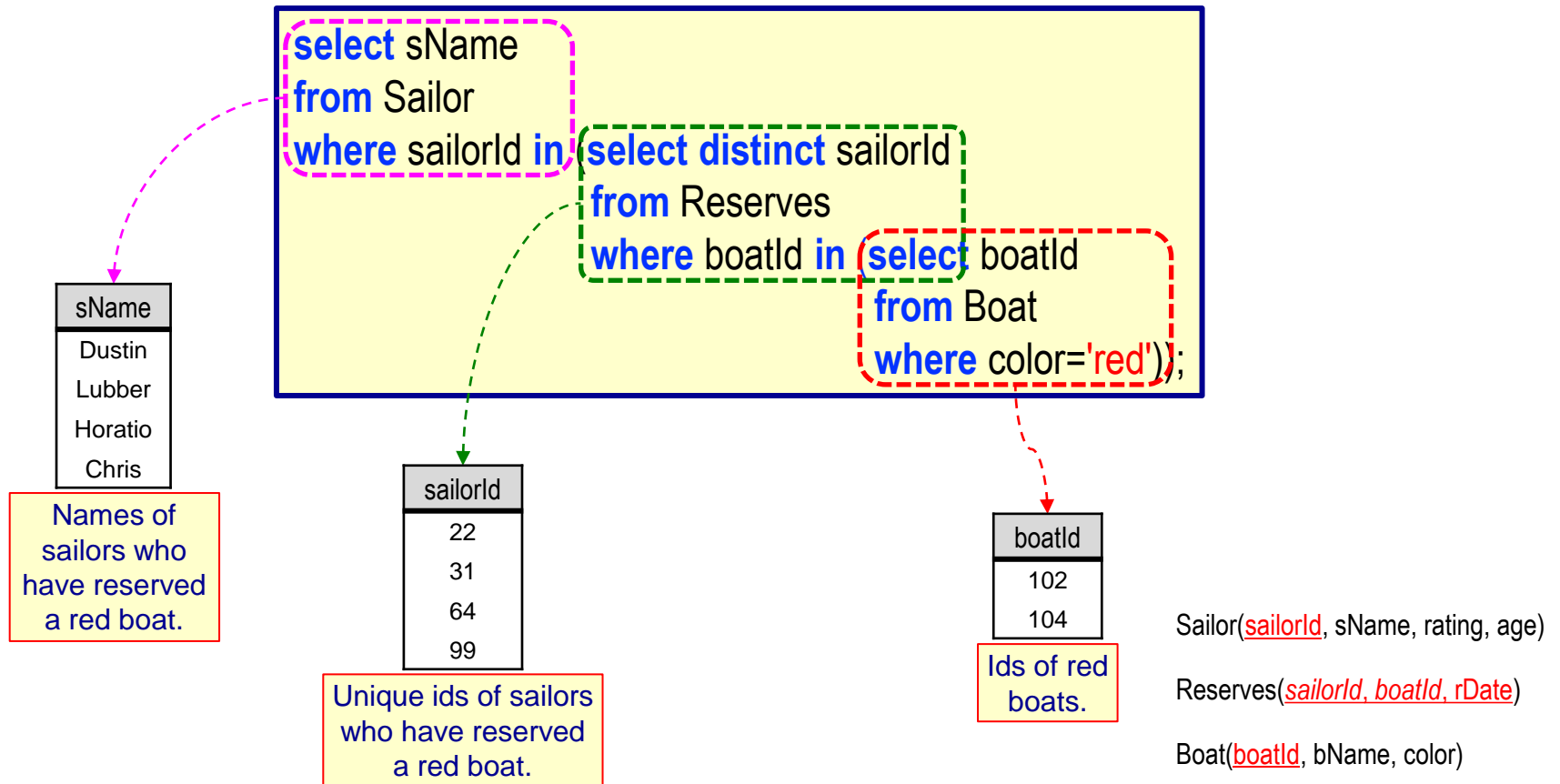
EXERCISE 4

DO NOT
use JOIN

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

Use *only* set membership

☞ **Dustin, Lubber, Horatio, Chris**



EXERCISE 4 (cont'd)

What if we replace the first in with **not in**?

Stated in words, what does this result represent?

```
select sName
from Sailor
where sailorId not in (select distinct sailorId
from Reserves
where boatId in (select boatId
from Boat
where color='red'))
```

sName
Brutus
Andy
Rusty
Zorba
Horatio
Art
Bob

Names of sailors who have not reserved a red boat (including reserved no boat).

sailorId
22
31
64
99

Unique ids of sailors who have reserved a red boat.

boatId
102
104

Ids of red boats.

Sailor(sailorId, sName, rating, age)
Reserves(sailorId, boatId, rDate)
Boat(boatId, bName, color)



EXERCISE 4 (cont'd)

What if we replace the second **in** with **not in**?

```
select sName
from Sailor
where sailorId in (select distinct sailorId
                  from Reserves
                  where boatId not in (select boatId
                                        from Boat
                                        where color='red'));
```

sName
Dustin
Lubber
Horatio
Horatio

Names of sailors who have reserved a boat other than a red boat (excludes sailors who have not reserved any boat).

sailorId
22
31
64
74

Ids of sailors who have reserved a boat other than a red boat.

Stated in words, what does this result represent?

boatId
102
104

Ids of red boats.

Sailor(sailorId, sName, rating, age)

Reserves(sailorId, boatId, rDate)

Boat(boatId, bName, color)



EXERCISE 4 (cont'd)

What if we replace both **in**'s with **not in**?

Stated in words, what does this result represent?

```
select sName
from Sailor
where sailorId not in (select distinct sailorId
                        from Reserves
                        where boatId not in (select boatId
                                             from Boat
                                             where color='red'));
```

sName
Brutus
Andy
Rusty
Zorba
Art
Bob
Chris

Names of sailors who have reserved only a red boat (i.e., Chris) or have reserved no boat.

sailorId
22
31
64
74

Ids of sailors who have reserved a boat other than a red boat.

boatId
102
104

Ids of red boats.

Sailor(sailorId, sName, rating, age)

Reserves(sailorId, boatId, rDate)

Boat(boatId, bName, color)



STRUCTURED QUERY LANGUAGE (SQL)

EXERCISE 4

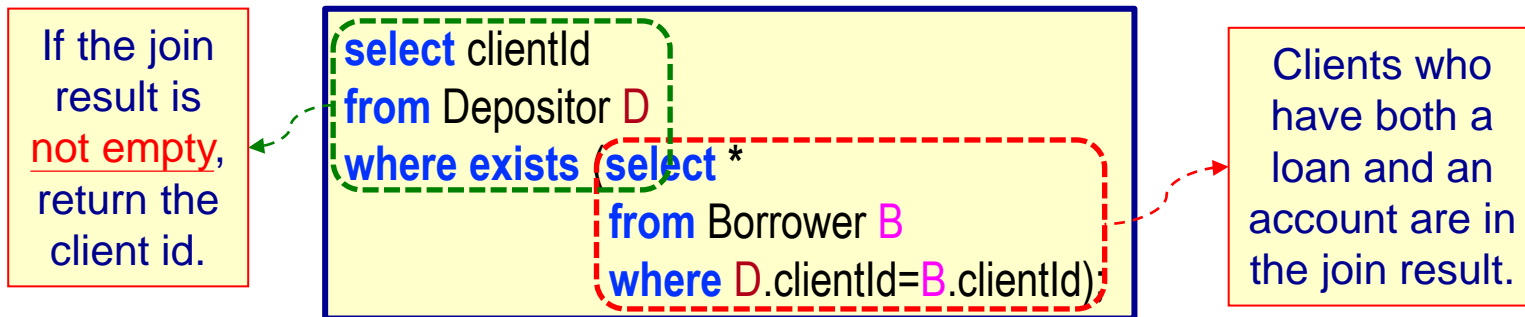
to be continued ...



EMPTY RELATION TEST

- The **exists** operator returns **true** if the subquery is *not empty* (i.e., the subquery **returns at least one tuple**).

Query: Find all clients who have both a loan and an account.



Scoping rules for correlation names (aliases) in subqueries.

- A correlation name defined in a subquery can be used only in the subquery itself or in any subquery contained in the subquery (e.g., **D** can be used in the nested **select**; **B** cannot be used in the outer **select**).
- Locally defined correlation names **override** globally defined names.

Not
implemented
in Oracle.

DUPLICATE TUPLES TEST: UNIQUE

- The **unique** operator tests for the *non existence* (i.e., absence) of duplicate tuples in a subquery.

👉 Returns **true** if the subquery contains **no duplicate tuples**.

Query: Find all clients who have *only one account* at the Star House branch.

See later slide for
an alternate way to
answer this query.

```

select D1.clientId
from Depositor D1
where unique (select D2.clientId
              from Account, Depositor D2
              where D1.clientId=D2.clientId
                 and D2.accountNo=Account.accountNo
                 and Account.branchName='Star House');

```

For each depositor
D1, check ...

Clients at Star House with
the same name as D1.

Find depositors with
the same name as D1.



DUPLICATE TUPLES TEST: REVISITED

- The **group by** and **having** clauses can test for the *non existence* (absence) and *existence* (presence) of duplicate tuples.

Query: Find all clients who have *only one account* at the Star House branch.

```
select clientId
from Depositor D, Account A
where D.accountNo=A.accountNo
      and branchName='Star House'
group by clientId
having count(*)=1;
```

Query: Find all clients who have *at least two accounts* at the Star House branch.

How would you answer this query?



SUBQUERIES IN THE FROM CLAUSE

- The **from** clause can contain a subquery expression.

Why?

Query: Find the name(s) of branches whose average balance is greater than the average account balance.

```
select branchName
from (select branchName, avg(balance) as avgBalance
      from Account
      group by branchName) result
where avgBalance > (select avg(balance)
                   from Account);
```

The average balance
of all accounts.

result

branchName	avg Balance
Star House	65000.00
Langham Place	7500.00
Pacific Place	83333.33

The **result** relation
contains the branch
name and average
balance of each branch.

 The relation **result** is called a **derived (temporary) relation**.

SUBQUERIES IN THE FROM CLAUSE (cont'd)

Query: Find the name and average balance of branches with the maximum average account balance.

```
select branchName, avgBalance
from (select branchName, avg(balance) as avgBalance
      from Account
      group by branchName) result
where avgBalance=(select max(avgBalance)
                  from result);
```

result

branchName	avg Balance
Star House	65000.00
Langham Place	7500.00
Pacific Place	83333.33

The maximum average balance in the **result** relation.

The **result** relation contains the branch name and average balance of each branch.

Oracle Note

This query is *not allowed in Oracle* due to Oracle's scoping rules.
(The scope of the **result** relation is restricted to the outer select clause.)

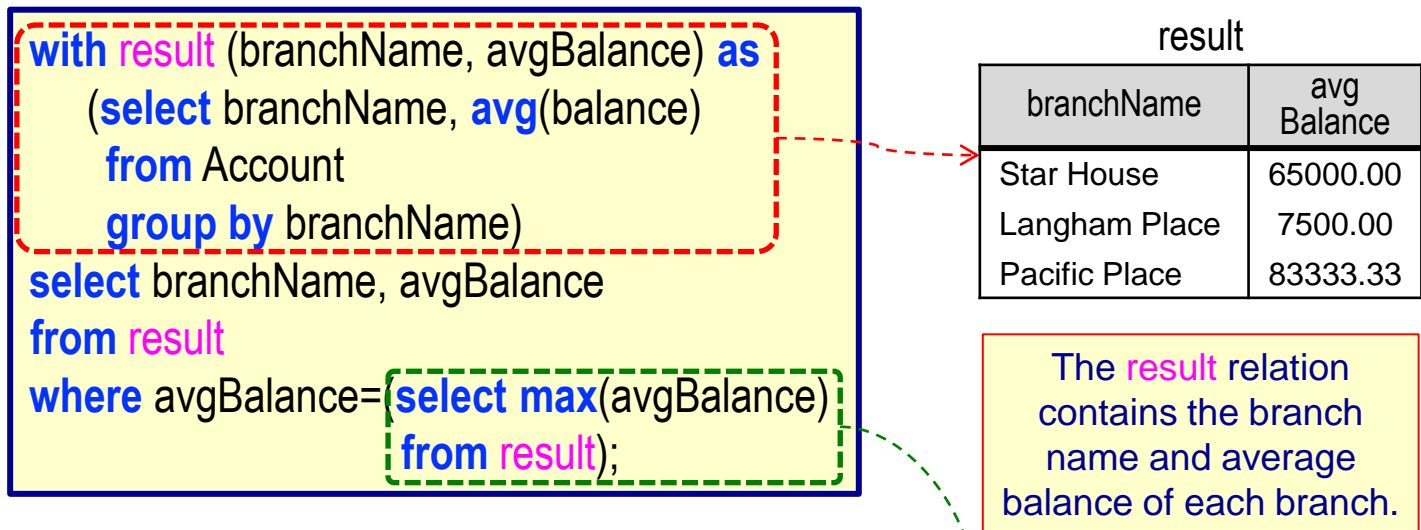
See the next slide.



WITH CLAUSE

- Allows a **derived (temporary) relation** to be defined that is available only to the query in which the **with** clause occurs.

Query: Find the name and average balance of branches with the maximum average account balance.



Oracle Note
This query *is allowed* in Oracle.

The maximum average balance in the **result** relation.



EXERCISE 4

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

Use exists

☞ Dustin, Lubber, Horatio, Chris

```
select sName
from Sailor S
where exists (select *
              from Reserves natural join Boat
              where Reserves.sailorId=S.sailorId
              and color='red');
```

sailorId	sName
22	Dustin
29	Brutus
31	Lubber
32	Andy
58	Rusty
64	Horatio
71	Zorba
74	Horatio
85	Art
95	Bob
99	Chris

Reserves natural join Boat where color='red'				
boatId	sailorId	rDate	bName	color
102	22	10/10/17	Interlake	red
102	64	08/09/17	Interlake	red
102	31	10/11/17	Interlake	red
104	22	07/10/17	Marine	red
104	99	08/08/17	Marine	red
104	31	12/11/17	Marine	red

Sailor(sailorId, sName, rating, age)

Reserves(sailorId, boatId, rDate)

Boat(boatId, bName, color)



EXERCISE 4

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

Use with clause

☞ Dustin, Lubber, Horatio, Chris

```
with RedBoatReservations (sailorId) as
(select sailorId
 from Reserves natural join Boat
 where color='red')
select distinct sName
from Sailor natural join RedBoatReservations;
```

sailorId	sName
22	Dustin
29	Brutus
31	Lubber
32	Andy
58	Rusty
64	Horatio
71	Zorba
74	Horatio
85	Art
95	Bob
99	Chris

RedBoatReservations
sailorId
22
64
31
22
99
31

Sailor(sailorId, sName, rating, age)

Reserves(sailorId, boatId, rDate)

Boat(boatId, bName, color)



STRUCTURED QUERY LANGUAGE (SQL) EXERCISES 5, 6, 7

Upload your completed exercise
worksheet to Canvas by **11 p.m.**
of Feb 24th

