

# ICS 321 Data Storage & Retrieval

## The Database Language SQL (iii)

Asst. Prof. Lipyeow Lim  
Information & Computer Science Department  
University of Hawaii at Manoa

# Bag Semantics in SQL

- SELECT-FROM-WHERE statements preserve duplicates by default, unless DISTINCT is given.
- Set operators UNION, INTERSECT, EXCEPT use **set semantics by default!**
- To use bag semantics: **UNION ALL, INTERSECT ALL, EXCEPT ALL.**

```
(SELECT title, year
FROM Movies)
UNION ALL
(SELECT movieTitle AS title, movieYear AS year
FROM StarsIn)
```

# Aggregate Operators

- SQL supports 5 aggregation operators on a column, say A,
  1. COUNT ( \* ), COUNT ( [DISTINCT] A )
  2. SUM ( [DISTINCT] A )
  3. AVG ( [DISTINCT] A )
  4. MAX ( A )
  5. MIN ( A )

# Aggregation Queries

- Q25: Find the average age of all sailors

```
SELECT AVG(S.age)  
FROM    Sailors S
```

- Q28: Count the number of sailors

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

- Find the age of the oldest sailor

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

## Q27: Find the name and age of the oldest sailor

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

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

- If there is an aggregation operator in the SELECT clause, then it can only have aggregation operators unless the query has a GROUP BY clause -- first query is illegal.

# Queries with GROUP BY and HAVING

SELECT	[DISTINCT] <i>target-list</i>
FROM	<i>relation-list</i>
WHERE	<i>qualification</i>
GROUP BY	<i>grouping-list</i>
HAVING	<i>group-qualification</i>

- The *target-list* contains (i) attribute names (ii) terms with aggregate operations (e.g., MIN (*S.age*)).
  - The list of attribute names in (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*.

# Conceptual Evaluation Strategy with GROUP BY and HAVING

- [Same as before] The cross-product of *relation-list* is computed, tuples that fail *qualification* are discarded, 'unnecessary' fields are deleted
- 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!*
  - In effect, an attribute in *group-qualification* that is not an argument of an aggregate op also appears in *grouping-list*. (SQL does not exploit primary key semantics here!)
- Aggregations in *target-list* are computed for each group
- One answer tuple is generated per qualifying group

Q32: Find age of the youngest sailor with age  $\geq$  18, for each rating with at least 2 such sailors

```
SELECT S.rating,  
        MIN(S.age) AS minage  
FROM Sailors S  
WHERE S.age  $\geq$  18  
GROUP BY S.rating  
HAVING COUNT (*) > 1
```

*Answer relation:*

rating	minage
3	25.5
7	35.0
8	25.5

*Sailors instance:*

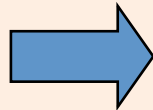
<u>sid</u>	sname	rating	age
22	dustin	7	45.0
29	brutus	1	33.0
31	lubber	8	55.5
32	andy	8	25.5
58	rusty	10	35.0
64	horatio	7	35.0
71	zorba	10	16.0
74	horatio	9	35.0
85	art	3	25.5
95	bob	3	63.5
96	frodo	3	25.5



# Conceptual Evaluation for Q32

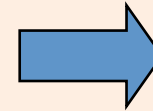
rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

Partition  
or  
GROUP BY



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

Eliminate groups  
Using HAVING clause



rating	minage
3	25.5
7	35.0
8	25.5

Perform aggregation  
on each group

# EVERY and ANY in HAVING clauses

```
SELECT S.rating, MIN(S.age) AS minage  
FROM Sailors S  
WHERE S.age >= 18  
GROUP BY S.rating  
HAVING COUNT (*) > 1 AND EVERY ( S.age <=60 )
```

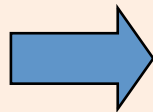
- **EVERY**: every row in the group must satisfy the attached condition
- **ANY**: at least one row in the group need to satisfy the condition

# Conceptual Evaluation with EVERY

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

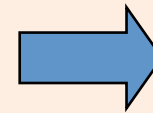
rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

Partition  
or  
GROUP BY



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

Eliminate groups  
Using HAVING clause



rating	minage
7	35.0
8	25.5

Perform aggregation  
on each group

What is the result of  
changing EVERY to ANY?

Find age of the youngest sailor for each rating with at least 2 sailors between 18 and 60

```
SELECT S.rating,  
        MIN (S.age) AS minage  
FROM Sailors S  
WHERE S.age >= 18 AND S.age <= 60  
GROUP BY S.rating  
HAVING COUNT (*) > 1
```

*Answer relation:*

rating	minage
3	25.5
7	35.0
8	25.5

*Sailors instance:*

<u>sid</u>	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

# Outer Joins

<b>S1</b>	<u>sid</u>	sname	rating	age
	22	Dustin	7	45.0
	31	Lubber	8	55.0
	58	Rusty	10	35.0

<b>R1</b>	<u>sid</u>	<u>bid</u>	<u>day</u>
	22	101	10/10/96
	58	103	11/12/96

- Regular join on sid: Sailor Lubber gets dropped.
- **Outer join:** Sailor rows without a matching Reserves row appear exactly once in the result, with the columns inherited from Reserves taking null values.
- **Left Outer Join :** Sailor rows w/o matching reservations appear in the result, but not vice versa
- **Right Outer Join:** Reservations w/o matching reservations appear in the result, but not vice versa

# Example of outer join

```
SELECT S1.*, R1.*  
FROM    Sailors S1 NATURAL OUTER JOIN Reserves R1
```

<b>S1</b>	<u>sid</u>	sname	rating	age
	22	Dustin	7	45.0
	31	Lubber	8	55.5
	58	Rusty	10	35.0

<b>R1</b>	<u>sid</u>	<u>bid</u>	<u>day</u>
	22	101	10/10/96
	58	103	11/12/96

## Result

- Note the nulls

sid	sname	rating	age	sid	bid	day
22	Dustin	7	45	22	101	10/10/96
31	Lubber	8	55.5	NULL	NULL	NULL
58	Rusty	10	35.0	58	103	11/12/96

# Insertion

```
INSERT INTO R(A1, A2, ...)  
  VALUES (v1, v2, ...);
```

```
INSERT INTO Studio(name)  
  SELECT DISTINCT studioname  
  FROM Movies  
  WHERE studioname NOT IN  
    (SELECT name  
     FROM Studio);
```

- If inserting results from a query, query must be evaluated prior to actual insertion

# Deletion

```
DELETE FROM R  
WHERE <condition>;
```

```
DELETE FROM StarsIn  
WHERE movieTitle = 'The Maltese Falcon' AND  
        MovieYear = 1942 AND  
        starName='Sydney Greenstreet';
```

- Deletion specified using a where clause.
- To delete a specific tuple, you need to use the primary key or candidate keys.



# Updates

```
UPDATE R  
SET <new value assignments>  
WHERE <condition>;
```

```
UPDATE MovieExec  
SET name='Pres. ' || name  
WHERE cert# IN (  
    SELECT presC#  
    FROM Studio );
```

- Tuples to be updated are specified using a where clause.
- To update a specific tuple, you need to use the primary key or candidate keys.