Lecture 10
Complex Queries in SQL

Nested Queries

- A nested query is a query that has another query embedded within it; the embedded query is called a subquery
- The embedded query can be a nested query itself
- A subquery typically appears within the WHERE clause of a query

Example 1

```
SELECT CId
FROM Climbers
WHERE Age > 40
INTERSECT
SELECT CId
FROM Climbs
WHERE RId = 1 ;
```

```
SELECT CId
FROM Climbers                    --outer query
WHERE Age > 40
AND CId IN (SELECT CId
            FROM Climbs          --subquery
            WHERE RId = 1) ;
```

Example 2

```
SELECT CId
FROM Climbers
WHERE Age < 40
MINUS
SELECT CId
FROM Climbs
WHERE RId = 1 ;
```

```
SELECT CId
FROM Climbers                          --outer query
WHERE Age < 40
AND CId  NOT IN (SELECT CId
                 FROM Climbs      --subquery
                 WHERE RId = 1) ;
```

Exercise

- Sailors: sid, sname, rating, age
- Boats: bid, bname, color
- Reserves: sid, bid, day

Exercise (Cont.)

1. Find the names of sailors who have reserved boat 103.
2. Find the names of sailors who have reserved a red boat.
3. Find the names of sailors who have not reserved a red boat.

Correlated Nested Queries

- The inner subquery has been completely independent of the outer query so far
- Inner subquery could depend on the row that is currently being examined in the outer query
- Correlated: subquery uses global variables in the outer query

EXISTS

- EXISTS allows to test whether a set is nonempty

```
SELECT CId
FROM Climbers c
WHERE EXISTS (SELECT *
              FROM Climbs b
              WHERE c.CId=b.CId AND b.RId = 1);
```

- Subquery depends on the current row c and must be re-evaluated for each row in Climbers

```
SELECT CId
FROM Climbers c
WHERE NOT EXISTS (SELECT *
                  FROM Climbs b
                  WHERE c.CId=b.CId);
```

UNIQUE

- UNIQUE will return true if no row appears twice in the answer set to the subquery
- Allows to test whether the result of a nested query is a set or a multiset

```
SELECT CId FROM Climbers c
WHERE UNIQUE
              (SELECT CId
               FROM Climbs b
               WHERE c.CId=b.CId AND RId = 1);
```

Comparison Operators

- IN, NOT IN, EXISTS, NOT EXISTS, UNIQUE, and NOT UNIQUE
- We can also use: <op> ANY, <op> ALL, where <op> is any of =, >, >=, <, <=, <>
- SOME is also available but it is just a synonym for ANY
- IN is equivalent to =ANY

Example 1

- What does the following mean in English?

```
SELECT CName, Age
FROM Climbers
WHERE Age >= ALL (SELECT Age
                  FROM Climbers);
```

CName	Age
Edmund	80

Example 2

- What does the following mean in English?

```
SELECT *
FROM Climbers
WHERE Age > ANY (SELECT Age
                 FROM Climbers
                 WHERE CName='Arnold');
```

CId	CName	Skill	Age
123	Edmund	EXP	80
313	Bridget	EXP	33
212	James	MED	27

Exercise

- Sailors: sid, sname, rating, age
- Boats: bid, bname, color
- Reserves: sid, bid, day

Exercise (Cont.)

- Find the names of sailors whose rating is better than some sailor called Smith.
- Find the names of sailors whose rating is better than every sailor called Smith.
- Find the names of sailors with the highest rating.

Division

- Which supplier has supplied all parts?

Supply Schema

sid (integer)	pid (integer)
101	1
102	1
101	3
103	2
102	2
102	3
102	4
102	5

pid (integer)
1
2
3
4
5

Contains

- Contains operator is used to compare two sets or multisets
- Contains operator compares two sets of values and returns true if one set contains all values in the other set
- Difficult to implement: most commercial DB systems do not have this operation

DIVISION in SQL

- The IDs of climbers who have climbed all routes (1)

```
SELECT CId
FROM Climbers c1
WHERE (SELECT RId
       FROM Climbs c2
       WHERE c1.CId=c2.CId)
      Contains
      (SELECT RId
       FROM Routes);
```

DIVISION in SQL (Cont.)

- (2)

```
SELECT CId
FROM Climbers c1
WHERE NOT EXISTS
      (SELECT RId
       FROM Routes r
       WHERE NOT EXISTS
             (SELECT *
              FROM Climbs c2
              WHERE c1.CId=c2.CId
                    and c2.RId=r.RId));
```

← Routes not climbed by c1.

DIVISION in SQL (Cont.)

- (3) S1 contains S2=(S2-S1) is empty

```
SELECT CId
FROM Climbers c1
WHERE NOT EXISTS
      ((SELECT RId
        FROM Routes)
       EXCEPT
        (SELECT RId
         FROM Climbs c2
         WHERE c1.CId=c2.CId));
```

Aggregate Functions

- SQL supports 5 aggregate operations: COUNT, SUM, MAX, MIN, and AVG
- Aggregate functions perform some computations or summarization on data
- These functions can be used in the SELECT clause or in a HAVING clause (introduce later)

A Sample Table

Routes:

RId	RName	Grade	Rating	Height
1	Last Tango	II	12	100
2	Garden Path	I	2	60
3	The Sluice	I	8	60
4	Picnic	III	3	400

COUNT

```
SELECT COUNT(RId)
FROM Routes;
```

```
SELECT COUNT(Grade)
FROM Routes;
```

- Both of these queries return 4 as the result

COUNT(RId)
4

COUNT(GRADE)
4

COUNT (Cont.)

- If we use the keyword “DISTINCT”:

```
SELECT COUNT(DISTINCT Grade)
FROM Routes;
```

COUNT(GRADE)
3

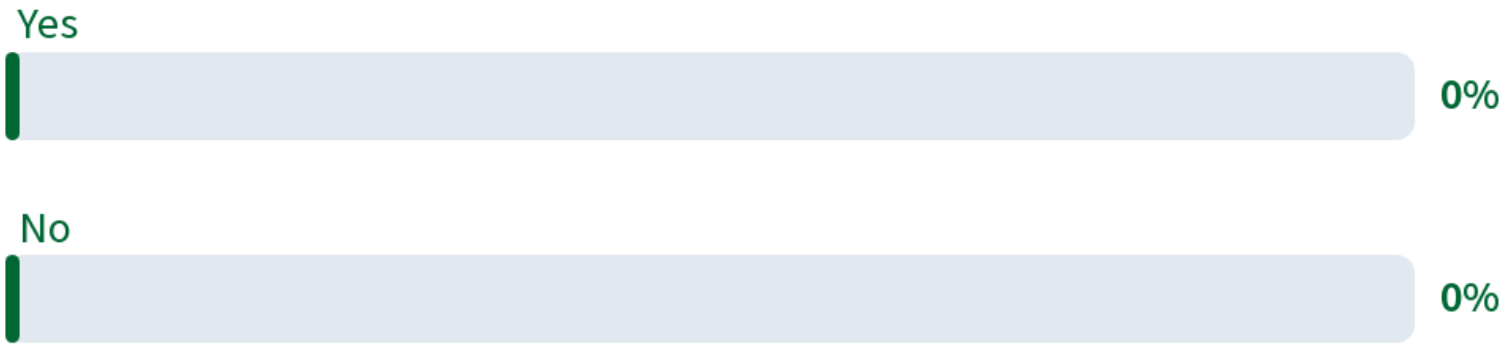
- Can also use SUM, AVG, MIN and MAX

Can we use aggregate function in WHERE clause?

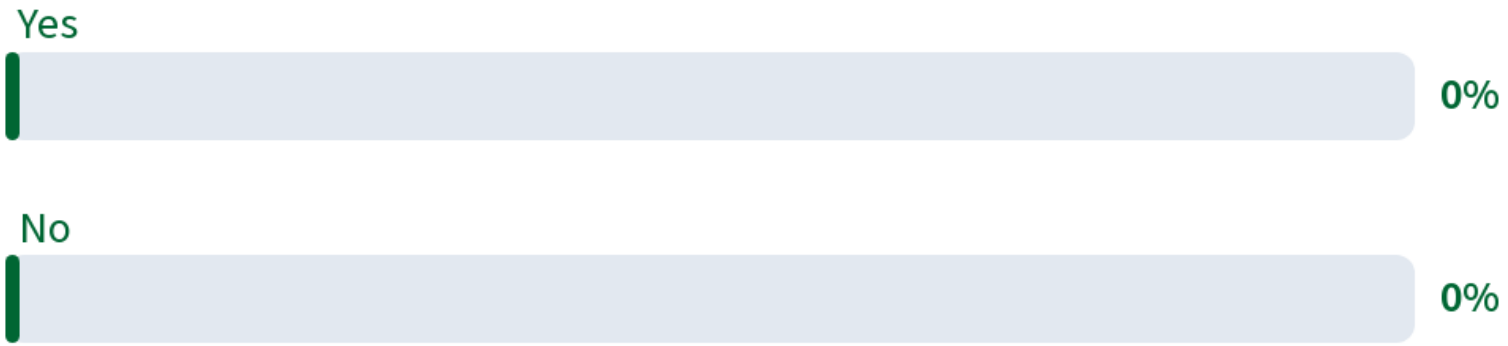
Yes

No

Can we use aggregate function in WHERE clause?



Can we use aggregate function in WHERE clause?



Exercise

- Sailors: sid, sname, rating, age
- Boats: bid, bname, color
- Reserves: sid, bid, day

Exercise (Cont.)

- Find the average age of all sailors.
- Find the average age of sailors with a rating of 10.
- Find the name and age of the oldest sailor.
- Count the number of sailors.
- Find the names of sailors who are older than the oldest sailor with a rating 10.

GROUP BY

- So far, aggregate operators have been applied to all qualifying tuples
- Sometimes we want to apply them to each of several groups of tuples
- The GROUP BY clause specifies the grouping attribute, which should also appear in the SELECT clause
- For example: “Print the number of routes in each grade.”

GROUP BY (Cont.)

```
SELECT Grade, COUNT(*)
FROM Routes
GROUP BY Grade;
```

GRADE	COUNT(*)
I	2
II	1
III	1

- The SELECT clause includes only the columns that appear in the GROUP BY statement and “aggregated” columns. So the following would generate an error

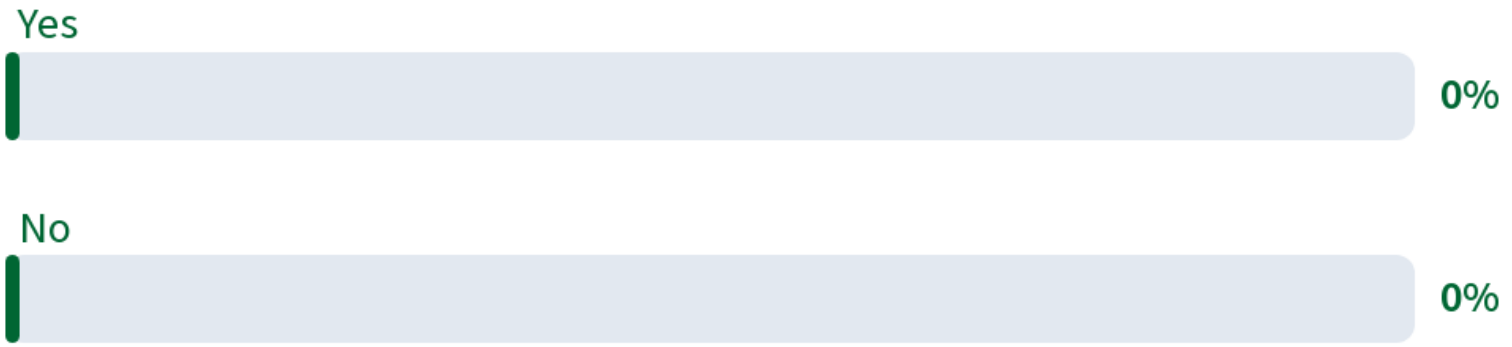
```
SELECT Grade, RName, COUNT(*)
FROM Routes
GROUP BY Grade;
```

Does grouping attribute need to be a key?

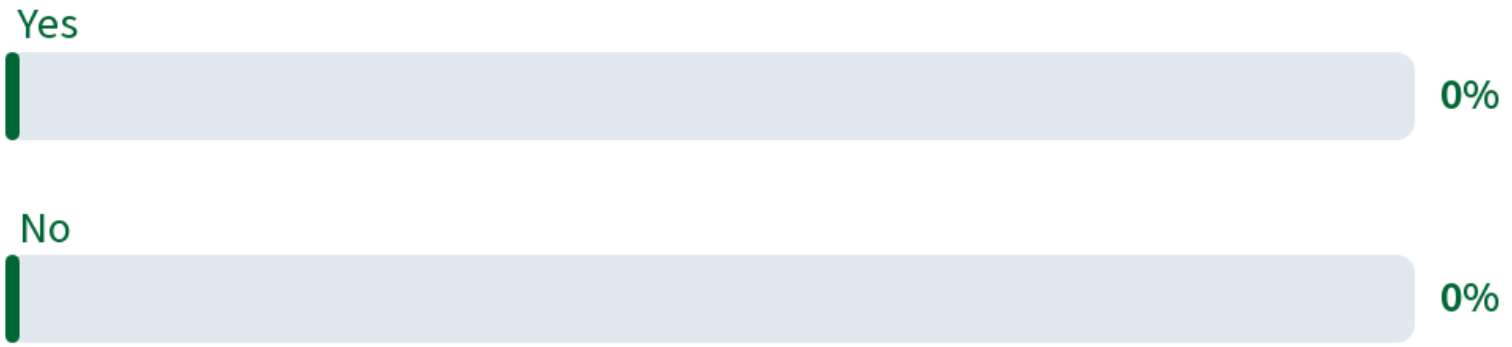
Yes

No

Does grouping attribute need to be a key?

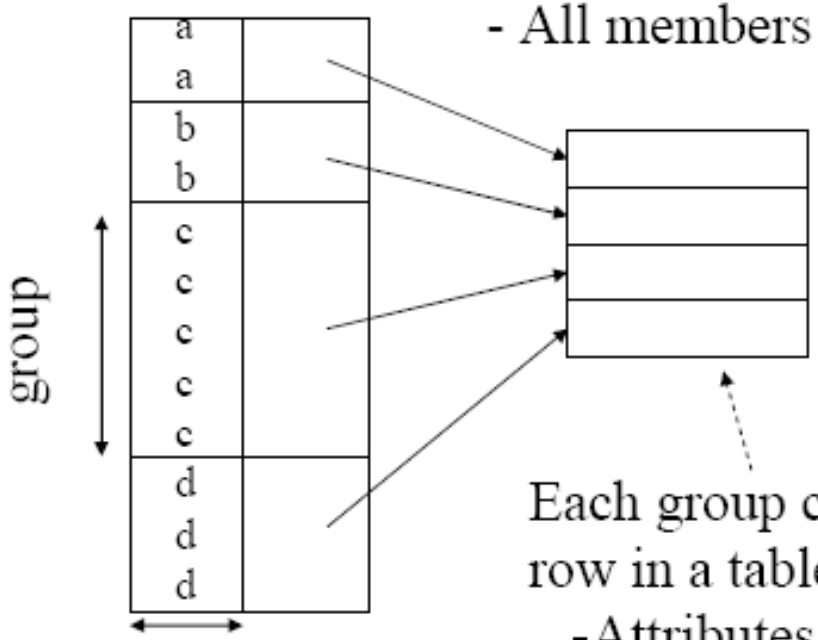


Does grouping attribute need to be a key?



GROUP BY (Cont.)

- Divide rows into groups based on subset of attributes;
- All members of a group agree on those attributes



Each group can be described by a single row in a table with attributes limited to:

- -Attributes all group members share (listed in GROUP BY clause)
- -Aggregates over group

HAVING

- Eliminate unwanted groups (analogous to WHERE clause)
- “HAVING” is used to restrict the groups that appear in the result
- The attribute(s) appearing in the HAVING clause must appear as the argument to an aggregation operator, or it must appear in GROUP BY clause

A Sample Table

Routes:

RId	RName	Grade	Rating	Height
1	Last Tango	II	12	100
2	Garden Path	I	2	60
3	The Sluice	I	8	60
4	Picnic	III	3	400

Example 1

```
SELECT Height, AVG(Rating)
FROM Routes
GROUP BY Height
HAVING Height < 300;
```

HEIGHT	AVG(RATING)
60	5
100	12

Example 2

```
SELECT Height, AVG(Rating)
FROM Routes
GROUP BY Height
HAVING MAX(Rating) < 10;
```

HEIGHT	AVG(RATING)
60	5
400	3

Exercise

- Sailors: sid, sname, rating, age
- Boats: bid, bname, color
- Reserves: sid, bid, day

Exercise (Cont.)

- Find the age of the youngest sailor for each rating level.
- Find the age of the youngest sailor who is at least 18 years old for each rating level with at least two such sailors.
- For each red boat, find the number of reservations for this boat.
- Find the average age of sailors for each rating level where the rating is greater than 10.
- Find the average age of sailors who are at least 18 years old for each rating level that has has at least two sailors.

Null Values

- The value of an attribute can be
  - *unknown* (e.g., a rating has not been assigned)
  - *inapplicable* (e.g., no spouse)
  - *unavailable or withheld* (e.g., a person has a home phone but does not want it to be listed
- SQL provides a special value *null* for such situations
- The presence of *null* complicates many issues
  - When a NULL is involved in a comparison operation, the result is considered to be UNKNOWN (it may be TRUE or FALSE)
  - NULL values are usually discarded when aggregate functions are applied to a particular column (attribute)
  - If NULLs exist in the grouping attribute, a **separate group** is created for all tuples with a NULL value in the grouping attribute

ORDER BY

- Allows to order tuples of a query by the values of one or more attributes
- The **default** order is in **ascending** order of values
- We can use keyword DESC for descending order of values
- The keyword ASC can be used to specify ascending order explicitly

Joined Tables in SQL

- Specify a table resulting from a join operation in the FROM clause of a query
- Easier to comprehend than mixing together all the select and join conditions in the WHERE clause
- Allows users to specify JOIN, NATRUAL JOIN, and various types of OUTER JOIN

JOIN

- Retrieve the name and address of every employee who works for the ‘Research’ department

```
SELECT Name, Address
FROM  Employee JOIN Department ON Dno=Dnumber
WHERE Dname='Research';
```

NATRUAL JOIN

- No join condition is specified
- Equality is implied for each pair of attributes with the same name
- Each such pair of attributes is included only once in the resulting relation

```
SELECT Name, Address
FROM  Employee NATURAL JOIN Department AS Dept
      (Dname,Dno,Mssn,Msdate)
WHERE Dname='Research';
```

OUTER JOIN

- A variant of the inner join that relies on null values:

```
SELECT *
FROM Climbers
     NATURAL LEFT OUTER JOIN Climbs;
```

- Tuples of Climbers that do not match some tuple in Climbs would normally be excluded from the result
- The “left” outer join preserves them with null values for the missing Climbs attributes

Result of Left Outer Join

CId	CName	Skill	Age	RId	Date	Duration
123	Edmund	EXP	80	1	10/10/88	5
123	Edmund	EXP	80	3	11/08/87	1
214	Arnold	BEG	25	2	08/07/92	2
313	Bridget	EXP	33	1	12/08/89	5
313	Bridget	EXP	33	1	06/07/94	3
212	James	MED	27	null	null	null