

ADVANCED SQL I

CS 564- Fall 2021

ACKs: Dan Suciu, Jignesh Patel, AnHai Doan

WHAT IS THIS LECTURE ABOUT

- **SQL: Aggregation**
 - Aggregate operators
 - GROUP BY
 - HAVING
- **SQL: Nested Queries**
 - IN/EXISTS/ALL
 - correlated queries

AGGREGATION

AGGREGATION

- **SUM, AVG, COUNT, MIN, MAX** can be applied to a column in a **SELECT** clause to produce that aggregation on the column
- **COUNT(*)** simply counts the number of tuples

```
SELECT AVG(Population)
FROM Country
WHERE Continent = 'Europe';
```

AGGREGATION: ELIMINATE DUPLICATES

We can use **COUNT(DISTINCT <attribute>)** to remove duplicate tuples before counting!

```
SELECT COUNT (DISTINCT Language)
FROM CountryLanguage ;
```

GROUP BY

- We may follow a **SELECT-FROM-WHERE** expression by **GROUP BY** and a list of attributes
- The relation is then grouped according to the values of those attributes, and any aggregation is applied only **within each group**

```
SELECT Continent, COUNT(*)  
FROM Country  
GROUP BY Continent;
```

GROUP BY: EXAMPLE

```
SELECT A, SUM(B * C)
FROM R
GROUP BY A;
```

R

A	B	C
a	2	0
a	5	1
b	7	1
b	6	0
c	4	1

grouping

A	B	C
a	2	0
	5	1
b	7	1
	6	0
c	4	1

SELECT
clause

$$5 = 2*0 + 5*1$$

A	SUM(B*C)
a	5
b	7
c	4

RESTRICTIONS

If any aggregation is used, then each element of the **SELECT** list must be either:

- aggregated, or
- an attribute on the **GROUP BY** list

This query is **wrong!!**

```
SELECT Continent, COUNT(Code)
FROM Country
GROUP BY Code;
```


GROUP BY + HAVING

The **HAVING** clause **always** follows a **GROUP BY** clause in a SQL query

- it applies to each group, and groups not satisfying the condition are removed
- it can refer only to attributes of relations in the **FROM** clause, as long as the attribute makes sense within a group

The HAVING clause applies **only** on aggregates!

HAVING: EXAMPLE

```
SELECT Language, COUNT(CountryCode) AS N  
FROM CountryLanguage  
WHERE Percentage >= 50  
GROUP BY Language  
HAVING N > 2  
ORDER BY N DESC ;
```

PUTTING IT ALL TOGETHER

```
SELECT [DISTINCT] S
FROM R, S, T ,...
WHERE C1
GROUP BY attributes
HAVING C2
ORDER BY attribute ASC/DESC
LIMIT N ;
```

CONCEPTUAL EVALUATION

1. Compute the **FROM-WHERE** part, obtain a table with all attributes in R,S,T,...
2. Group the attributes in the **GROUP BY**
3. Compute the aggregates and keep only groups satisfying condition **C2** in the **HAVING** clause
4. Compute aggregates in S
5. Order by the attributes specified in **ORDER BY**
6. Limit the output if necessary

NESTED QUERIES

NESTED QUERIES

A parenthesized SELECT-FROM-WHERE statement (*subquery*) can be used as a value in a:

- **FROM** clause
- **WHERE** clause

```
SELECT C.Name
FROM Country C
WHERE C.code =
    (SELECT C.CountryCode
     FROM City C
     WHERE C.name = 'Berlin');
```

outer query

inner query

NESTING

- We can write nested queries because the SQL language is **compositional**
- Everything is represented as a multiset
- Hence the output of one query can be used as the input to another (**nesting**)

NESTED QUERIES

Find all countries in Europe with population more than 50 million

```
SELECT C.Name
FROM (SELECT Name, Continent
      FROM Country
      WHERE Population >50000000) AS C
WHERE C.Continent = 'Europe' ;
```

USING WITH

Find all countries in Europe with population more than 50 million

```
WITH C AS (SELECT Name, Continent
            FROM Country
            WHERE Population >50000000)
SELECT C.Name
FROM C
WHERE C.Continent = 'Europe' ;
```

NESTED QUERIES

Find all countries in Europe with population more than the average population of a European country

```
SELECT C.Name
FROM Country C
WHERE C.Continent = 'Europe'
AND C.Population > (
    SELECT AVG(Population)
    FROM Country
    WHERE Continent = 'Europe') ;
```

UNNESTING

Unnesting means to find an equivalent SQL query that does not use nesting!

```
SELECT C.Name
FROM (SELECT Name, Continent
      FROM Country
      WHERE Population > 50000000) AS C
WHERE C.Continent = 'Europe' ;
```

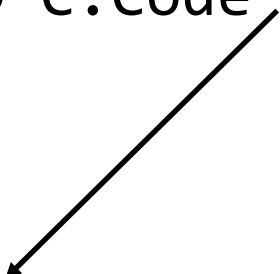
```
SELECT Name
FROM Country
WHERE Population > 50000000
AND Continent = 'Europe' ;
```

↓ unnesting

SET-COMPARISON OPERATOR: IN

*Find all countries in Europe that have **some** city with population more than 5 million*

```
SELECT C.Name
FROM Country C
WHERE C.Continent = 'Europe'
AND C.Code IN (SELECT CountryCode
                FROM City
                WHERE Population > 5000000);
```



checks whether the value is in the table returned by the subquery

SET-COMPARISON OPERATOR: EXISTS

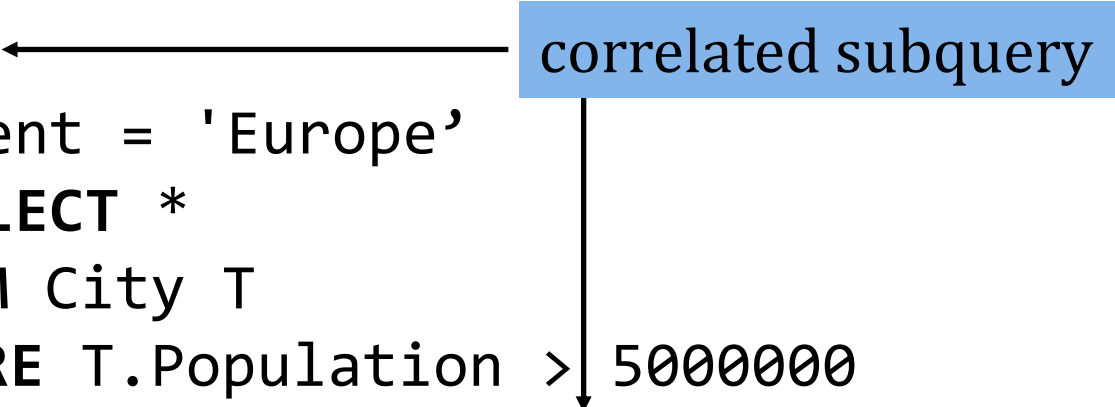
*Find all countries in Europe that have **some** city with population more than 5 million*

```
SELECT C.Name
FROM Country C
WHERE C.Continent = 'Europe'
AND EXISTS (SELECT *
             FROM City T
             WHERE T.Population > 5000000
             AND T.CountryCode = C.Code);
```

CORRELATED SUBQUERIES

- A **correlated subquery** uses values defined in the outer query
- The inner subquery gets executed multiple times!

```
SELECT C.Name
FROM Country C ← correlated subquery
WHERE C.Continent = 'Europe'
AND EXISTS (SELECT *
            FROM City T
            WHERE T.Population > 5000000
            AND T.CountryCode = C.Code);
```



SET-COMPARISON OPERATORS: EXISTS

*Find all countries in Europe that have **all** cities with population less than 1 million*

```
SELECT C.Name
FROM Country C
WHERE C.Continent = 'Europe'
AND NOT EXISTS (SELECT *
                  FROM City T
                  WHERE T.Population > 1000000
                  AND T.CountryCode = C.Code);
```

SET-COMPARISON OPERATOR: ANY

*Find all countries in Europe that have **some** city with population more than 5 million*

```
SELECT C.Name
FROM Country C
WHERE C.Continent = 'Europe'
AND 5000000 <= ANY (SELECT T.Population
                     FROM City T
                     WHERE T.CountryCode = C.Code);
```

The operator before **ANY** must be a comparison operator!

SET-COMPARISON OPERATORS: ALL

*Find all countries in Europe that have **all** cities with population less than 1 million*

```
SELECT C.Name
FROM Country C
WHERE C.Continent = 'Europe'
AND 1000000 > ALL (SELECT T.Population
                    FROM City T
                    WHERE T.CountryCode = C.Code);
```

The operator before **ALL** must also be a comparison operator!