ADVANCED SQL II

CS 564 - Fall 2018

ACKs: *Dan Suciu, Jignesh Patel, AnHai Doan*

WHAT IS THIS LECTURE ABOUT

- SQL: Aggregation
  - Aggregate operators
  - GROUP BY
  - HAVING
- SQL: Nulls
- SQL: Outer Joins

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 ;
```

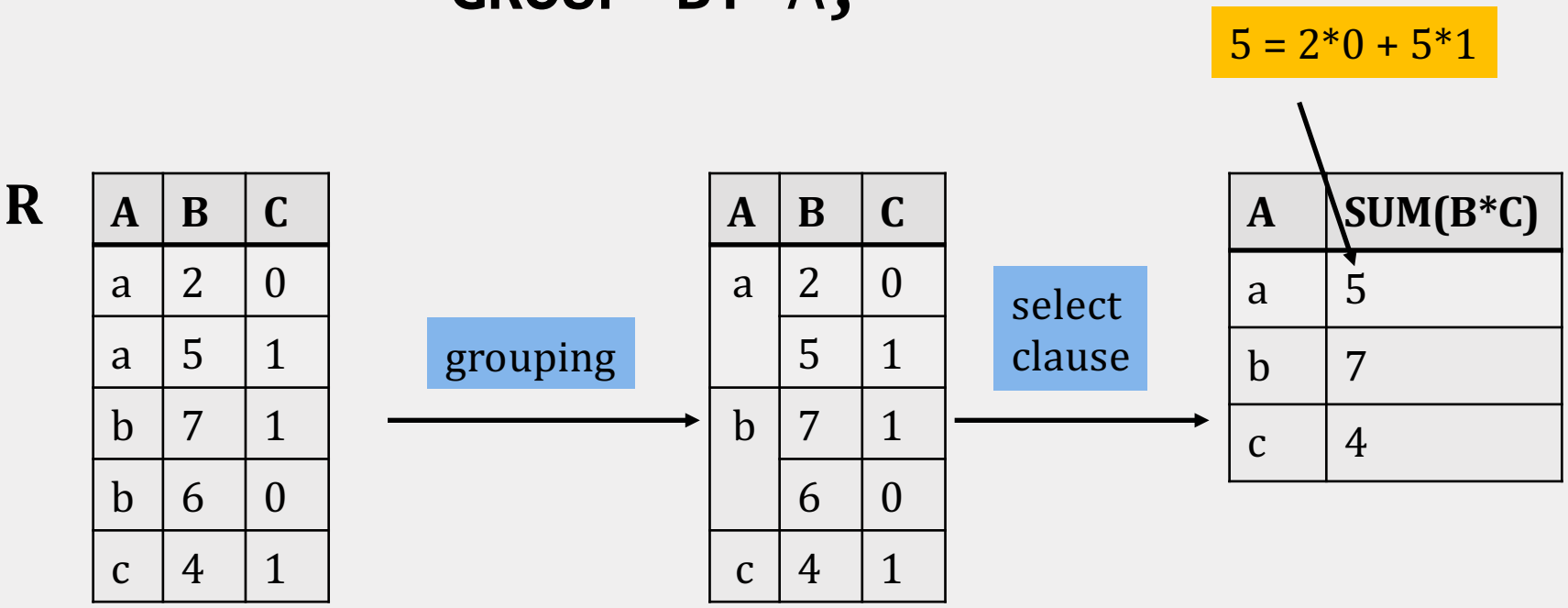
GROUP BY

- 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;
```



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

NULL VALUES

NULL VALUES

- tuples in SQL relations can have **NULL** as a value for one or more attributes
- The meaning depends on context:
  - Missing value: *e.g.* we know that Greece has some population, but we don't know what it is
  - Inapplicable: *e.g.* the value of attribute *spouse* for an unmarried person

NULL PROPAGATION

- When we do arithmetic operations using **NULL**, the result is again a **NULL**
  - (10 * x)+5 returns **NULL** if x = **NULL**
  - **NULL**/0 also returns **NULL**!

- String concatenation also results in **NULL** when one of the operands is **NULL**
  - 'Wisconsin' || **NULL**|| '-Madison' returns **NULL**

COMPARISONS WITH NULL

- The logic of conditions in SQL is 3-valued logic:
  - **TRUE** = 1
  - **FALSE** = 0
  - **UNKNOWN** = 0.5
- When any value is compared with a **NULL**, the result is **UNKNOWN**
  - *e.g.* x > 5 is **UNKNOWN** if x = **NULL**
- A query produces a tuple in the answer only if its truth value in the **WHERE** clause is **TRUE** (1)

3-VALUED LOGIC

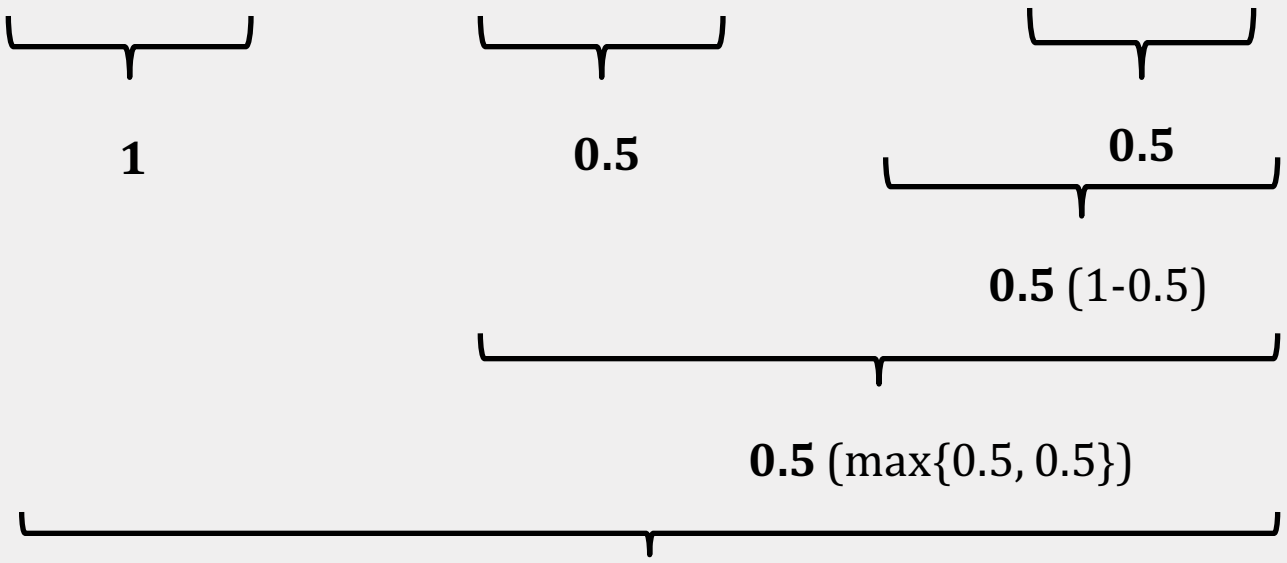
The truth value of a **WHERE** clause is computed using the following rules:

- C1 **AND** C2 ----> *min*{ value(C1), value(C2) }
- C1 **OR** C2 ----> *max*{ value(C1), value(C2) }
- **NOT** C ----> 1- value(C)

3-VALUED LOGIC: EXAMPLE

tuple (1, NULL, NULL)

```
SELECT *
FROM R
WHERE (R.A>0) AND ((R.B<5) OR (NOT R.C=3));
```



the expression is **UNKNOWN**!

COMPLICATIONS

What will happen in the following query?

```
SELECT COUNT(*)
FROM Country
WHERE IndepYear > 1990 OR IndepYear <= 1990 ;
```

It will not count the rows with NULL!

TESTING FOR NULL

We can test for **NULL** explicitly:

- `x IS NULL`
- `x IS NOT NULL`

```
SELECT COUNT(*)
FROM Country
WHERE IndepYear > 1990 OR IndepYear <= 1990
OR IndepYear IS NULL;
```

OUTER JOINS

INNER JOINS

The joins we have seen so far are inner joins

```
SELECT C.Name AS Country, MAX(T.Population) AS N
FROM Country C, City T
WHERE C.Code = T.CountryCode
GROUP BY C.Name;
```

Alternative syntax:

```
SELECT C.Name AS Country, MAX(T.Population) AS N
FROM Country C
INNER JOIN City T ON C.Code = T.CountryCode
GROUP BY C.Name;
```

We can simply also write **JOIN**

LEFT OUTER JOINS

A left outer join includes tuples from the left relation even if there's no match on the right! It fills the remaining attributes with NULL

```
SELECT C.Name AS Country, MAX(T.Population)
FROM Country C
LEFT OUTER JOIN City T
    ON C.Code = T.CountryCode
GROUP BY C.Name ;
```

LEFT OUTER JOIN: EXAMPLE

```
SELECT A, C
FROM R LEFT OUTER JOIN S
ON R.B = S.B
```

R

A	B
a	2
a	5
b	5
c	6

S

B	C
2	100
3	200
5	300
7	400

A	C
a	100
a	300
b	300
c	**NULL**

OTHER OUTER JOINS

- Left outer join:
  - include the left tuple even if there is no match
- Right outer join:
  - include the right tuple even if there is no match
- Full outer join:
  - include the both left and right tuples even if there is no match