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

```sql
SELECT AVG(Population)
FROM Country
WHERE Continent = 'Europe';
```
AGGREGATION: ELIMINATE DUPLICATES

We can use \texttt{COUNT(DISTINCT <attribute>)} to remove duplicate tuples before counting!

\begin{verbatim}
SELECT COUNT (DISTINCT Language) 
FROM CountryLanguage ;
\end{verbatim}
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.

```sql
SELECT Continent, COUNT(*)
FROM Country
GROUP BY Continent;
```
## GROUP BY: EXAMPLE

### SQL Query

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

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

### Grouping

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

### Select Clause

- **a**: $5 = 2 \times 0 + 5 \times 1$
- **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**!!

```sql
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`
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 \( x = \text{NULL} \)

A query produces a tuple in the answer **only if** its truth value in the **WHERE** clause is **TRUE** (1)
The truth value of a \textbf{WHERE} clause is computed using the following rules:

- \textbf{C1 AND C2} \quad ----> \quad min\{\ value(C1), \ value(C2) \} \\
- \textbf{C1 OR C2} \quad ----> \quad max\{ \ value(C1), \ value(C2) \} \\
- \textbf{NOT C} \quad ----> \quad 1- \ value(C)
SELECT * 
FROM R 
WHERE (R.A>0) AND ((R.B<5) OR (NOT R.C=3));

tuple (1, NULL, NULL)

the expression is UNKNOWN!
What will happen in the following query?

```sql
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:

- \( x \text{ IS NULL} \)
- \( x \text{ IS NOT NULL} \)

```
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

```sql
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:

```sql
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

```sql
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
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>5</td>
</tr>
<tr>
<td>b</td>
<td>5</td>
</tr>
<tr>
<td>c</td>
<td>6</td>
</tr>
</tbody>
</table>

S
<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>200</td>
</tr>
<tr>
<td>5</td>
<td>300</td>
</tr>
<tr>
<td>7</td>
<td>400</td>
</tr>
</tbody>
</table>

A | C |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>100</td>
</tr>
<tr>
<td>a</td>
<td>300</td>
</tr>
<tr>
<td>b</td>
<td>300</td>
</tr>
<tr>
<td>c</td>
<td>NULL</td>
</tr>
</tbody>
</table>
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