5. SQL Querying

SQL Outline:

1. Join
2. Nullvalues
3. Aggregation and Grouping
4. Operations on Sets
5. Subqueries
6. Orthogonality of Syntax
7. Views
8. Insert, Delete and Update
9. Referential Integrity
10. Trigger
11. Outlook: Analysis

Terminology

Rows of a table are also called *tuples* and columns of a table are called *attributes*. 
Join: RDB’s speciality to combine tables

How many people live in the capitals?

Problem: Table Country mentions capitals, but not population; table city mentions population, but does not tell us capitals! The join is the solution: we compute all possible pairs between rows in the two tables and select those pairs in which Country.Capital = City.Name!

```
SELECT A.Name, A.Capital, B.Inhabitants
  FROM Country A, City B
  WHERE A.Capital = B.Name;
```

<table>
<thead>
<tr>
<th>Country</th>
<th>Name</th>
<th>Code</th>
<th>Capital</th>
</tr>
</thead>
<tbody>
<tr>
<td>Austria</td>
<td>A</td>
<td>Vienna</td>
<td></td>
</tr>
<tr>
<td>Egypt</td>
<td>ET</td>
<td>Cairo</td>
<td></td>
</tr>
<tr>
<td>France</td>
<td>F</td>
<td>Paris</td>
<td></td>
</tr>
<tr>
<td>Germany</td>
<td>D</td>
<td>Berlin</td>
<td></td>
</tr>
<tr>
<td>Italy</td>
<td>I</td>
<td>Rome</td>
<td></td>
</tr>
<tr>
<td>Russia</td>
<td>RU</td>
<td>Moscow</td>
<td></td>
</tr>
<tr>
<td>Switzerland</td>
<td>CH</td>
<td>Bern</td>
<td></td>
</tr>
<tr>
<td>Turkey</td>
<td>TR</td>
<td>Ankara</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>City</th>
<th>Name</th>
<th>Country</th>
<th>Inhabitants</th>
<th>Longitude</th>
<th>Latitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>Berlin</td>
<td>D</td>
<td>3472</td>
<td>13.2</td>
<td>52.45</td>
<td></td>
</tr>
<tr>
<td>Freiburg</td>
<td>D</td>
<td>198</td>
<td>7.51</td>
<td>47.59</td>
<td></td>
</tr>
<tr>
<td>Karlsruhe</td>
<td>D</td>
<td>277</td>
<td>8.24</td>
<td>49.03</td>
<td></td>
</tr>
<tr>
<td>Munich</td>
<td>D</td>
<td>1244</td>
<td>11.56</td>
<td>48.15</td>
<td></td>
</tr>
<tr>
<td>Nuremberg</td>
<td>D</td>
<td>495</td>
<td>11.04</td>
<td>49.27</td>
<td></td>
</tr>
<tr>
<td>Paris</td>
<td>F</td>
<td>2125</td>
<td>2.48</td>
<td>48.81</td>
<td></td>
</tr>
<tr>
<td>Rome</td>
<td>I</td>
<td>2546</td>
<td>12.6</td>
<td>41.8</td>
<td></td>
</tr>
</tbody>
</table>

Energy Informatics, WS 2015/16
### Join variants

**Give me for each country its cities.**

```
SELECT A.CoName, B.L CiName
FROM Country A JOIN City B ON A.CoCode = B.CoCode
```

_in case we want to join with respect to equal column names we have a *natural join*_:

```
SELECT A.CoName, B.L CiName
FROM Country A NATURAL JOIN City B
```

_if we really want the *cartesian product*_:

```
SELECT A.CoName, B.L CiName
FROM Country A CROSS JOIN City B
```
Part 2: System Design

5. SQL Querying

How many people live in the capitals?

```sql
SELECT A.CoName, A.Capital, B.Inhabitants
FROM Country A JOIN City B
ON A.Capital = B.CiName;
```

<table>
<thead>
<tr>
<th>CoName</th>
<th>Capital</th>
<th>Inhabitants</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>Paris</td>
<td>2125</td>
</tr>
<tr>
<td>Germany</td>
<td>Berlin</td>
<td>3472</td>
</tr>
<tr>
<td>Italy</td>
<td>Rome</td>
<td>2546</td>
</tr>
</tbody>
</table>

What if we like to keep the information lost in case of missing join partners?

We can fill missing partners columns by *null-values*!

```sql
SELECT A.CoName, A.Capital, B.Inhabitants
FROM Country A LEFT OUTER JOIN City B
ON A.Capital = B.CiName;
```

<table>
<thead>
<tr>
<th>CoName</th>
<th>Capital</th>
<th>Inhabitants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Austria</td>
<td>Vienna</td>
<td>null</td>
</tr>
<tr>
<td>Egypt</td>
<td>Cairo</td>
<td>null</td>
</tr>
<tr>
<td>France</td>
<td>Paris</td>
<td>2125</td>
</tr>
<tr>
<td>Germany</td>
<td>Berlin</td>
<td>3472</td>
</tr>
<tr>
<td>Italy</td>
<td>Rome</td>
<td>2546</td>
</tr>
<tr>
<td>Russia</td>
<td>Moscow</td>
<td>null</td>
</tr>
<tr>
<td>Switzerland</td>
<td>Bern</td>
<td>null</td>
</tr>
<tr>
<td>Turkey</td>
<td>Ankara</td>
<td>null</td>
</tr>
</tbody>
</table>

```sql
SELECT A.CoName, A.Capital, B.Inhabitants
FROM Country A RIGHT OUTER JOIN City B
ON A.Capital = B.CiName;
```

<table>
<thead>
<tr>
<th>CoName</th>
<th>Capital</th>
<th>Inhabitants</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>Paris</td>
<td>2125</td>
</tr>
<tr>
<td>Germany</td>
<td>Berlin</td>
<td>3472</td>
</tr>
<tr>
<td>Italy</td>
<td>Rome</td>
<td>2546</td>
</tr>
<tr>
<td>null</td>
<td>null</td>
<td>198</td>
</tr>
<tr>
<td>null</td>
<td>null</td>
<td>277</td>
</tr>
<tr>
<td>null</td>
<td>null</td>
<td>1244</td>
</tr>
<tr>
<td>null</td>
<td>null</td>
<td>495</td>
</tr>
</tbody>
</table>

Use OUTER JOIN to get the union of Left and RIGHT OUTER JOIN.
Nullvalues: The Case of Missing Information

The problem having a null-value

If for a tuple the value of an attribute is not known - what could be the reason for using null?

- A value exists, however not known at the moment,
- Value will exist in the future.
- Attribute-value for that tuple unknown, in principle.
- Attribute for that tuple not applicable.

Testing for null

SQL offers to test for null by using predicates IS NULL, respectively, IS NOT NULL in the WHERE-clause.

SELECT * FROM Country
    WHERE Capital IS NOT NULL
Null-values in expressions.

▶ In arithmetic expressions $A+B$, $A+1$, etc. the result is $\text{null}$, whenever one of the operands has value $\text{null}$.

▶ Arithmetic comparison expressions $A=B$, $A\neq B$, $A<B$, etc. have truth-value $\text{UNKNOWN}$, whenever one of the operands has value $\text{null}$.

▶ SQL’s logic is three-valued, i.e. has truth values ($t=\text{TRUE}$, $f=\text{FALSE}$, $u=\text{UNKNOWN}$).

<table>
<thead>
<tr>
<th>AND</th>
<th>t</th>
<th>u</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>t</td>
<td>u</td>
<td>f</td>
</tr>
<tr>
<td>u</td>
<td>u</td>
<td>u</td>
<td>f</td>
</tr>
<tr>
<td>f</td>
<td>f</td>
<td>f</td>
<td>f</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OR</th>
<th>t</th>
<th>u</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>t</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>u</td>
<td>t</td>
<td>u</td>
<td>u</td>
</tr>
<tr>
<td>f</td>
<td>t</td>
<td>u</td>
<td>f</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NOT</th>
<th>t</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>u</td>
<td>u</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>t</td>
<td></td>
</tr>
</tbody>
</table>

Avoid null-values whenever possible!
Simple Analysis: Aggregation and Grouping

Aggregation operators
COUNT, MIN, MAX, SUM and AVG.

```
SELECT COUNT(*), COUNT(CiName), COUNT(DISTINCT CoCode),
    MAX(Inhabitants), MIN(Inhabitants), AVG(Inhabitants)
FROM City
```

More on DISTINCT
```
SELECT CoCode
FROM City
```
```
SELECT DISTINCT CoCode
FROM City
```
DISTINCT here removes duplicate rows from the result table!
Forming groups of tuples.

- Using the GROUP BY-clause we define a virtual structure on a table based on the values of the chosen attributes.
- Using the HAVING-clause only those groups are considered, which fulfill the condition stated in the HAVING-clause.

Important: in the SELECT-clause, attributes which are NOT used for grouping, are only allowed to appear as parameters of the aggregation operators!

```
SELECT CoCode, AVG(Inhabitants) FROM City
  GROUP BY CoCode

SELECT CoCode, MAX(Inhabitants) FROM City
  GROUP BY CoCode
  HAVING AVG(Inhabitants) < 2000
```
### SQL’s simple SFW-Expressions

<table>
<thead>
<tr>
<th>SQL Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SELECT</strong> $A_1, \ldots, A_n$</td>
<td>Result Attribute</td>
</tr>
<tr>
<td><strong>FROM</strong> $R_1, \ldots, R_m$</td>
<td>Tables used</td>
</tr>
<tr>
<td><strong>WHERE</strong> $F$</td>
<td>Condition on tuples</td>
</tr>
<tr>
<td><strong>GROUP BY</strong> $B_1, \ldots, B_k$</td>
<td>Grouping attributes</td>
</tr>
<tr>
<td><strong>HAVING</strong> $G$</td>
<td>Grouping condition</td>
</tr>
<tr>
<td><strong>ORDER BY</strong> $H$</td>
<td>Sorting</td>
</tr>
</tbody>
</table>

Evaluation strategy: FROM-clause first, then WHERE-clause, then GROUP-clause, then HAVING-clause, then ORDER-clause and finally SELECT-clause.
Tables are treated as sets of rows!

Set operators UNION, INTERSECT and MINUS.

Tables must have the same number of attributes and attributes on the same column-position must have *compatible* values.

```
SELECT CiName FROM City
INTERSECT
SELECT CoName FROM Country
```

```
SELECT CiName FROM City
MINUS
SELECT CoName FROM Country
```

```
SELECT CiName, 'City' AS Category FROM City
UNION
SELECT CoName, 'Country' AS Category FROM Country
```
Advanced Querying: Using Subqueries

A query is called *nested*, if its SELECT-, FROM-, WHERE-, or HAVING-clause does contain a SFW-expression - also called *subquery*.

To test the results of a subquery operators IN, ANY, ALL, UNIQUE, EXISTS and NOT can be used.

```sql
SELECT DISTINCT CiName FROM City
  WHERE CoCode IN
    (SELECT CoCode FROM Country WHERE Capital = 'Berlin')
```

```sql
SELECT CiName FROM City
  WHERE Population > ANY
    (SELECT Population FROM City)
```
SELECT CiName FROM City
    WHERE Population > ALL
        (SELECT Population FROM City)

WRONG! -
all other cities!

SELECT CiName FROM City A
    WHERE Population > ALL
        (SELECT Population FROM City B
            WHERE A.CiCode <> B.CiCode)

- A and B above are called *correlation variables* - the subquery is executed for each possible tuple of the outer table A; each such A-tuple is referenced by A in the subquery.

- In general, if there are several outer tables, the subquery is executed for each combination of the respective correlation variables.
SELECT CoName FROM Country A  
   WHERE UNIQUE  
       (SELECT CiName FROM City B  
        WHERE A.CoCode = B.CoCode)

SELECT CoName FROM Country A  
   WHERE 1 = 
       (SELECT COUNT(*) FROM City B  
        WHERE A.CoCode = B.CoCode)
Division of Tables

<table>
<thead>
<tr>
<th>CoCode</th>
<th>Organization</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>EU</td>
<td>member</td>
</tr>
<tr>
<td>D</td>
<td>EU</td>
<td>member</td>
</tr>
<tr>
<td>D</td>
<td>WEU</td>
<td>member</td>
</tr>
<tr>
<td>ET</td>
<td>UN</td>
<td>member</td>
</tr>
<tr>
<td>I</td>
<td>EU</td>
<td>member</td>
</tr>
<tr>
<td>I</td>
<td>NAM</td>
<td>guest</td>
</tr>
<tr>
<td>TR</td>
<td>UN</td>
<td>member</td>
</tr>
<tr>
<td>TR</td>
<td>CERN</td>
<td>observer</td>
</tr>
</tbody>
</table>

Describe!

```
SELECT DISTINCT CoCode FROM Membership M
WHERE NOT EXISTS
    ((SELECT Organization FROM Membership WHERE CoCode = 'A')
     MINUS
     (SELECT Organization FROM Membership WHERE CoCode = M.CoCode))
```

We compute all countries which are member in at least those organizations, in which Austria a member is.

This is similar to usual Division - why?.

Energy Informatics, WS 2015/16
Equality of tables

Remember, sets $A$, $B$ are equal iff $A \subseteq B$ and $B \subseteq A$;

$A \subseteq B$ iff $A - B = \emptyset$.

Which countries exactly have the same organization as Austria?

```
SELECT DISTINCT CoCode FROM Membership M WHERE
   NOT EXISTS
   ((SELECT Organization FROM Membership WHERE CoCode = 'A')
       MINUS
   (SELECT Organization FROM Membership WHERE CoCode = M.CoCode))
AND NOT EXISTS
   ((SELECT Organization FROM Membership WHERE CoCode = M.CoCode)
       MINUS
   (SELECT Organization FROM Membership WHERE CoCode = 'A'))
```
Nice Syntax: Orthogonality Applies

- A table-expressions can appear wherever a table could appear.
- A scalar expression can appear wherever a scalar value can appear.
- A boolean expression can appear wherever a boolean value can appear.
# Table Expressions

```sql
SELECT Name
FROM (SELECT CiName AS Name
FROM City UNION
SELECT CoName AS Name
FROM Country) T
```

```sql
SELECT SUM(CitySlicker)
FROM (SELECT CoCode, MAX(Inhabitants) AS CitySlicker
FROM City
GROUP BY CoCode) T
```
Scalar Expressions

```sql
SELECT CoName,
    (SELECT SUM(Inhabitants) FROM City B
        WHERE B.CoCode = A.CoCode)
    AS CoInhabitants
FROM Country A
```

### Location

<table>
<thead>
<tr>
<th>CoCode</th>
<th>Continent</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Europe</td>
<td>100</td>
</tr>
<tr>
<td>F</td>
<td>Europe</td>
<td>100</td>
</tr>
<tr>
<td>TR</td>
<td>Asia</td>
<td>68</td>
</tr>
<tr>
<td>TR</td>
<td>Europe</td>
<td>32</td>
</tr>
<tr>
<td>ET</td>
<td>Africa</td>
<td>90</td>
</tr>
<tr>
<td>ET</td>
<td>Asia</td>
<td>10</td>
</tr>
<tr>
<td>RU</td>
<td>Asia</td>
<td>80</td>
</tr>
<tr>
<td>RU</td>
<td>Europe</td>
<td>20</td>
</tr>
</tbody>
</table>

```sql
SELECT DISTINCT CoCode, Percentage FROM Location
    WHERE Continent = 'Asia' AND
        Percentage <
            (SELECT Percentage FROM Location
                WHERE CoCode = 'TR' AND Continent = 'Asia')
```
Boolean Expressions

Assume: INSERT INTO Country VALUES ('Wonderland', 'WO', null)

```
SELECT CiName FROM City
    WHERE CiName NOT IN (SELECT Capital FROM Country)
```

Result: empty table.

```
SELECT CiName FROM City A
    WHERE NOT EXISTS (SELECT Capital FROM Country
                      WHERE Capital = A.CiName )
```

Result: Freiburg, Munich, Nuremberg, Karlsruhe.

Give the reasons!