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

Country

Name	Code	Capital
Austria	A	Vienna
Egypt	ET	Cairo
France	F	Paris
Germany	D	Berlin
Italy	I	Rome
Russia	RU	Moscow
Switzerland	CH	Bern
Turkey	TR	Ankara

City

Name	Country	Inhabitants	Longitude	Latitude
Berlin	D	3472	13,2	52,45
Freiburg	D	198	7,51	47,59
Karlsruhe	D	277	8,24	49,03
Munich	D	1244	11,56	48,15
Nuremberg	D	495	11,04	49,27
Paris	F	2125	2,48	48,81
Rome	I	2546	12,6	41,8

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;
```

Name	Capital	Inhabitants
France	Paris	2125
Germany	Berlin	3472
Italy	Rome	2546

Country

CoName	CoCode	Capital
Austria	A	Vienna
Egypt	ET	Cairo
France	F	Paris
Germany	D	Berlin
Italy	I	Rome
Russia	RU	Moscow
Switzerland	CH	Bern
Turkey	TR	Ankara

City

CiName	CoCode	Inhabitants	Longitude	Latitude
Berlin	D	3472	13,2	52,45
Freiburg	D	198	7,51	47,59
Karlsruhe	D	277	8,24	49,03
Munich	D	1244	11,56	48,15
Nuremberg	D	495	11,04	49,27
Paris	F	2125	2,48	48,81
Rome	I	2546	12,6	41,8

Join variants

Give me for each country its cities.

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

if we really want the *cartesian product*:

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

How many people live in the capitals?

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

CoName	Capital	Inhabitants
France	Paris	2125
Germany	Berlin	3472
Italy	Rome	2546

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

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

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

CoName	Capital	Inhabitants
Austria	Vienna	null
Egypt	Cairo	null
France	Paris	2125
Germany	Berlin	3472
Italy	Rome	2546
Russia	Moscow	null
Switzerland	Bern	null
Turkey	Ankara	null

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

CoName	Capital	Inhabitants
France	Paris	2125
Germany	Berlin	3472
Italy	Rome	2546
null	null	198
null	null	277
null	null	1244
null	null	495

use FULL 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 `null`, whenever one of the operands has value `null`.
- Arithmetic comparison expressions A=B, A<>B, A<B, etc. have truth-value `UNKNOWN`, whenever one of the operands has value `null`.
- SQL's logic is three-valued, i.e. has truth values (`t=TRUE, f=FALSE, u=UNKNOWN`).

AND	t	u	f
t	t	u	f
u	u	u	f
f	f	f	f

OR	t	u	f
t	t	t	t
u	t	u	u
f	t	u	f

NOT	
t	f
u	u
f	t

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

SELECT $A_1, \dots, A_n$	Result Attribute
FROM $R_1, \dots, R_m$	Tables used
WHERE F	Condition on tuples
GROUP BY $B_1, \dots, B_k$	Grouping attributes
HAVING G	Grouping condition
ORDER BY H	Sorting

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.

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

```
SELECT CiName FROM City
   WHERE Inhabitants > ANY
      (SELECT Inhabitants FROM City)
```

```
SELECT CiName FROM City
   WHERE Inhabitants > ALL
      (SELECT Inhabitants FROM City)
```

WRONG! -
all *other* cities!

```
SELECT CiName FROM City A
   WHERE Inhabitants > ALL
      (SELECT Inhabitants FROM City B
         WHERE A.CiName <> B.CiName)
```

- `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

Membership

CoCode	Organization	Status
A	EU	member
D	EU	member
D	WEU	member
ET	UN	member
I	EU	member
I	NAM	guest
TR	UN	member
TR	CERN	observer

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?.

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

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

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

Scalar Expressions

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

Location

CoCode	Continent	Percentage
D	Europe	100
F	Europe	100
TR	Asia	68
TR	Europe	32
ET	Africa	90
ET	Asia	10
RU	Asia	80
RU	Europe	20

```
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!