SQL

SQL --- Structured Query Language

SQL is the most used database query language today.

- Standard since 80ties (1986); no changes last 10-20 years
- If there is the standard, „SQL“ is varies among different database systems
- Queries written for one database system (PosgreSQL) need not to work in another (MySQL), mostly because of usefull extensions of the SQL standard the database systems implement
- SQL contains also standards Data Manipulation Language (DML) and Data Definition Language (DDL)

Basic syntax:

- **SELECT** <attributes>
  FROM <relations>
  WHERE <condition>
  [ORDER BY attribute1, attribute2...]
  [LIMIT 100] [OFFSET 0]

Example of a SQL query

```
SELECT
     concat(e.firstname,’ ‘,e.lastname) AS ename,
     (CASE
          WHEN e.comm IS NULL THEN e.sal
          ELSE e.comm + e.sal
     ) AS ‘total_salary’
FROM emp
WHERE deptno>=20 AND lower(e.firstname)=‘john’
```

Multisets

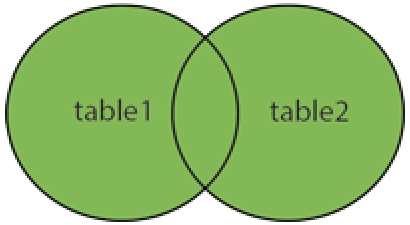
- SQL threats relations as multisets, i.e. multisets can contain duplicate rows (opposite to Datalogu).
- If you want to strip duplicates you must enforce it using some commands (**UNIQUE** constraint when creating table – more in DDL – or using **DISTINCT** keyword in queries).

JOINs

- join is the union of two tables; it is a subset of the cartesian product of tables specified by additional conditions for linking (cartesian product - each row with each)
- FULL JOIN
- INNER JOIN or just JOIN
- LEFT JOIN
- RIGHT JOIN

Cartesian product (FULL JOIN):

FULL OUTER JOIN



Name	Deptno
John	10
Thomas	20
Joe	40

X

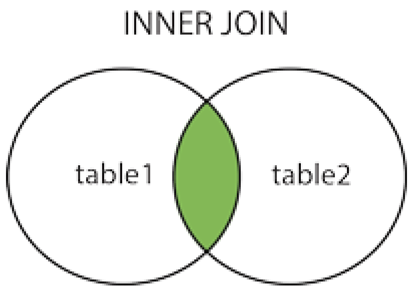
Deptno	Dept. name
10	Accounting
20	PR
30	Development

=

Name	Deptno	Deptno	Dept. name
John	10	10	Accounting
John	10	20	PR
John	10	30	Development
Thomas	20	10	Accounting
Thomas	20	20	PR
Thomas	20	30	Development
Joe	40	10	Accounting
Joe	40	20	PR
Joe	40	30	Development

```
SELECT * FROM emp, dept
```

INNER JOIN = JOIN:



Name	Deptno
John	10
Thomas	20
Joe	40

JOIN

Deptno	Dept. name
10	Accounting
20	PR
30	Development

=

Name	Deptno	Deptno	Dept. name
John	10	10	Accounting
~~John~~	~~10~~	~~20~~	~~PR~~
~~John~~	~~10~~	~~30~~	~~Development~~
~~Thomas~~	~~20~~	~~10~~	~~Accounting~~
Thomas	20	20	PR
~~Thomas~~	~~20~~	~~30~~	~~Development~~
~~Joe~~	~~40~~	~~10~~	~~Accounting~~
~~Joe~~	~~40~~	~~20~~	~~PR~~
~~Joe~~	~~40~~	~~30~~	~~Development~~

```
SELECT * FROM emp e, dept d
WHERE e.deptno = d.deptno
```

SELECT * FROM emp e
        JOIN dept d **ON** e.deptno = d.deptno

SELECT * FROM emp e **natural join** dept as d

INNER JOIN = JOIN:

Name	Deptno
John	10
Thomas	20
Joe	40

JOIN

Deptno	Dept. name
10	Accounting
20	PR
30	Development
10	Human res.

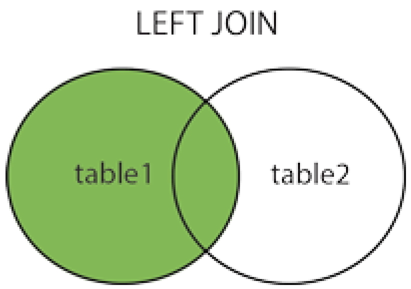
=

Name	Deptno	Deptno	Dept. name
John	10	10	Accounting
~~John~~	~~10~~	~~20~~	~~PR~~
~~John~~	~~10~~	~~30~~	~~Development~~
John	10	10	Human res.
~~Thomas~~	~~20~~	~~10~~	~~Accounting~~
Thomas	20	20	PR
~~Thomas~~	~~20~~	~~30~~	~~Development~~
~~Thomas~~	~~20~~	~~10~~	~~Human res.~~
~~Joe~~	~~40~~	~~10~~	~~Accounting~~
~~Joe~~	~~40~~	~~20~~	~~PR~~
~~Joe~~	~~40~~	~~30~~	~~Development~~
~~Joe~~	~~40~~	~~10~~	~~Human res.~~

```
SELECT *
FROM emp e
        JOIN dept d ON e.deptno = d.deptno
```

How would you write join using Datalog?

LEFT [OUTER] JOIN:



Name	Deptno
John	10
Thomas	20
Joe	40

LEFT JOIN

Deptno	Dept. name
10	Accounting
20	PR
30	Development
10	Human res.

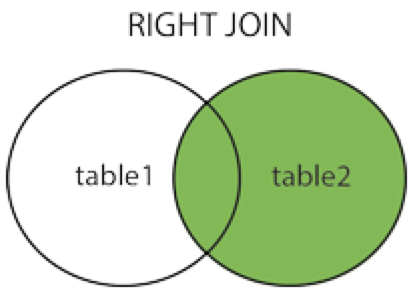
=

Name	Deptno	Deptno	Dept. name
John	10	10	Accounting
John	10	10	Human res.
Thomas	20	20	PR
Joe	40	null	null

```
SELECT *
FROM emp as e
        LEFT JOIN dept as d
                ON e.deptno = d.deptno
```

How would you write LEFT JOIN using Datalogu?

RIGHT [OUTER] JOIN:



Deptno	Dept. name
10	Accounting
20	PR
30	Development
10	Human res.

RIGHT JOIN

Name	Deptno
John	10
Thomas	20
Joe	40

=

Name	Deptno	Deptno	Dept. name
John	10	10	Accounting
John	10	10	Human res.
Thomas	20	20	PR
Joe	40	null	null

The same as LEFT JOIN, just in opposite direction

```
SELECT *
FROM dept AS d
        RIGHT JOIN emp AS e ON e.deptno = d.deptno
```

Operators, expressions, functions

- You can use many operators in WHERE clause:
  - =, <>, >, <, >=, <=, BETWEEN, LIKE, IN, IS NULL, IS NOT NULL
  - AND, OR, ! (NOT)
- Also it is possible to use arithmetic expressions and many more functions
  - E.g. *concat(e.firstname,' ', e.lastname)*
  - Functions to work with date and time
  - List of supported functions depends on a database system
    - https://www.postgresql.org/docs/current/static/functions.html

Inner SELECT (subselect)

- SELECT name FROM emp e WHERE e.ID IN (SELECT ID FROM managers)
- SELECT name FROM emp e WHERE EXISTS (SELECT * FROM managers WHERE id=e.id)

- in the case of nested selects, one must be careful about efficiency
  - **JOIN operations can optimize the database system quite well**
  - (if you have a properly designed DB - more on that later)
  - even EXISTS and NOT EXISTS are straightforward
  - optimizing IN and NOT IN can be a problem, it's easier to write an "inefficient" query
- Note that EXISTS can always be rewritten as JOIN; NOT EXISTS as a difference (EXCEPT)

UNION, EXCEPT

- SELECT name
  FROM emp_dallas WHERE sal>=1000

  UNION [ALL]

  SELECT name
  FROM emp_huston WHERE sal>=500

- Number and type of attributes in SELECT clause must be the same

Auxiliary tables and CTE

```
WITH emp_houston AS (
        SELECT * FROM emp as e, dept as d
        WHERE e.deptno=d.deptno and d.dname=‘houston’
)
SELECT * FROM emp_houston WHERE sal>=1000
```

-
  ```
  CREATE TEMPORARY TABLE emp_houston (
        SELECT * FROM emp as e, dept as d
        WHERE e.deptno=d.deptno and d.dname=‘houston’
  );
  ```

-
  ```
  SELECT * FROM emp_houston WHERE sal>=1000
  ```

PostgreSQL

- we will work with the PostgreSQL database system during the exercises
- Most database systems have a client-server architecture
  - Server
    - contains data
    - understands SQL queries
    - clients connect to it mostly via socket (TCP/IP), or named pipes or other channels supported by the OS
  - client
    - An application that needs to work with data
    - sends queries to server in SQL language
    - displays/processes response from the server
- Some features can be implemented on any page (e.g., pagination).

PostgreSQL

- We have two terminal windows open on cvika.dcs.fmph.uniba.sk
- In one window, we edit the file with the assignment, e.g. vim queries.sql
- In the second window, run the edited file (all queries in it) with the command

  psql -f queries.sql

  - It doesn't hurt to have a third window where we run PSQL and use it to view the contents of the database and debug queries

- Everyone is working on their database, which is automatically selected after running PSQL

Woriking with PostgreSQL console

- PostgreSQL console (interactive terminal) with the psql command
- You can then write queries in the console, e.g. **SELECT * FROM emp;**
- Interesting special commands:
    - \d emp or \d+ emp - Displays the table structure
    - \d - Displays a list of tables in the current database
    - \db - Displays a list of databases
    - \c emp - connects to the EMP database
    - \q – exit console
- Console documentation: http://www.postgresql.org/docs/current/static/app-psql.html