MDA104: Tutorial 4
Analytical queries in SQL

Vlastislav Dohnal

Online app to practice SQL

- Use our simple app
  - https://disa.fi.muni.cz/projects/MDA104/
  - accessible from the university's network (use MUNI VPN).

- **OR** use your own instance of PostgreSQL
  - Download data from IS.

Extended GROUP BY

- In OLAP, you may use CUBE, ROLLUP and GROUPING SETS in GROUP BY.
- Let's have relations

  course (*code*, *title*, *credits*, *type_of_completion*)

  enrollment (*učo*, *code*, *type_of_completion*)

  student (*učo*, *first_name*, *last_name*)
- In SQL, write queries that return:
  - A last name initial and the number of students with that initial, incl. the grand total.
  - The number of enrollments per course and provides a **grand total** across all courses, where you print the course code and title of it along with the number of enrollments.
  - Count the number of students per course and type of completion, including subtotals, i.e., data for a pivot table.
  - Calculate the **total number of credits** achieved by students – grouped by individual students and courses and then including subtotals and grand total. Print course title and student name instead of their IDs.

Analytic Functions – Windowing Functions

- Windowing functions are aggregate functions applied with OVER (….) in SELECT
  - E.g. SELECT emp_id, salary, **AVG(salary) OVER (PARTITION BY dept)** FROM employee;
  - It allows to compare "aggregate" values with stored values (rows) directly.
  - E.g., SELECT emp_id, salary,
    AVG(salary) OVER (PARTITION BY dept) AS avg,
    salary / avg AS ratio
    FROM employee;

- For the same schema (see previous slide), write an SQL query that
  - Ranks students based on the total number of credits they have earned from highest to lowest. Print the student učo, name, the number of credits, and their rank.
    - Mind "ORDER BY" can follow "PARTITION BY" in OVER (…) and PARTITION BY may be omitted.
  - Compares each student's earned credits to the average across all students. Print the student's učo, name, earned credits, overall average, and the difference as number of credits.

Analytic Functions – Windowing Functions (cont.)

- Let's have relations

  course (*code*, *title*, *credits*, *type_of_completion*)

  enrollment (*učo*, *code*, *type_of_completion*)

  student (*učo*, *first_name*, *last_name*)

  grading (*učo*, *code*, *graded_on*, *grade*) – *grade (A-F,X,-) received on the date*

- In SQL, write queries that return:
  - Assign each student a percentile rank based on the total number of credits they have earned.
  - The most recent grade for each student in each course. Print the student's name, course title, and grade.

Recursive Queries

- Recursive queries solve the limit on the number of joined tables in FROM clause.
- Let's have relations

  course (*code*, *title*, *credits*, *type_of_completion*)

  enrollment (*učo*, *code*, *type_of_completion*)

  student (*učo*, *first_name*, *last_name*)

  prereq (*code*, *prereq*) *-- a prereq is a course code that must be passed before code.*

- In SQL, write queries that return:
  - (easy) code and title of each course and its direct prerequisite (code) if any.
  - (joins) two levels of prerequisites for the course code PB168
    - Print the PB168 code and its prerequisites and the prerequisites of the prerequisites.
  - (recursion) the same with a recursive query