TECHNICAL UNIVERSITY
OF CRETE (TUC)
Department ECE
COURSE: COURSE NO 303 - DATABASES

Reference Phase 1: Explanation of SQL functions

User Group 37

Team Members:	Papoutsakis Nikolaos	2019030206
	Siganos Socrates	2019030097

SQL Functions

- 2.1

  Student Entry `create_students(entries, entry_date)`

The function creates a certain number of new students who share a certain year of enrolment. For each student, a new record is created in both the `Student and Person` tables. To create new e-mails, the `TRANSLATE_GREEK_ENGLISH(text)` function is used, which uses the student's Greek full name to create an alphanumeric in English. The AMKA is a random unique number.

Adding Professors `create_professors(entries)`
Similar to the previous function, it inserts a certain number of teachers into the tables `Professor, Person`.

Lab Staff Introduction `create_lab_teachers(entries)`
Similar to the previous functions, it inserts a specific number of lab personnel into the `LabTeacher, Person` tables.

- 2.2

The ***insert_random_grades(semester_id)*** function inserts random grades for courses in a given semester. It generates random grades for exams and labs, calculates the final grade based on the percentage of exams, and updates the Register table accordingly. The function takes as input the id of the semester and performs the grade generation and updates using a loop.

E.g. SELECT insert_random_grades (24);

- 2.3

The ***create_program(...)*** function creates a curriculum with the arguments listed in the pronunciation. The function works differently depending on the type of program, as well as the year of the program.

Introduction of the Standard Curriculum

In case there is no formal program entry in the Program table, the first program is created based on the function arguments, the total number of students belonging to the program is calculated, all students with an enrollment year greater than or equal to the program year are entered in the Joins table, and all courses are entered in the ProgramOffersCourse table for that program. In case there is already at least one entry of a previous formal program, some cases can be distinguished. 1) If the program year is less than that of the most recent formal program, the function terminates the program insertion process. 2) If the year is the same as that of the most recent program, then the existing program is modified based on the given arguments, the total number of students is recalculated in case new students were createdand the new students are inserted in the Joins table for the existing program. 3) If the year is older than the most recent standard program, a new program is created to which the students in the previous standard program with an intake year greater than or equal to the year of the new program are transferred (Update JOINS), potential new students are entered into the program, the total number of students in the new program is calculated, and the new number of students for the previous standard program is recalculated, and the entries are created in the ProgramOffersCourse table.

Introduction of Foreign Language Curriculum

Importing a foreign language programme works similarly to the above procedure except that the year of the programme may not be the same as an existing foreign language programme. Also, each program contains foreign language students only in the specific year of admission, not later. The number of students is calculated based on the foreign language students in the year and from the random number of graduate students admitted to the program. Finally, enrollments are created for all courses in the ProgramOffersCourse table.

Introduction of Seasonal Curriculum

For the introduction of a seasonal programme, a random number of students who are not enrolled in a curriculum are selected. To create units, the create_custom_unit() function is used which creates a unit for a particular seasonal program, which contains the courses given as an argument. Multiple units may be created for a seasonal program.

- 3.1

The ***find_information_using_am(student_am text)*** function returns information about a student based on the student's registration ID (AM). Retrieving the information is done with a simple JOIN the Student and Person tables. The function takes as input the student's am and the function returns Table with the student's details.

E.g. SELECT * FROM find_information_using_am('2015000001');;

- 3.2

The ***get_students_fullname_from_course(code character)*** function returns the AM, first name, and last name fields from the Person, Student, and Register tables, along with the CourseRun and Semester tables. The information is retrieved using partial INNER JOIN and based on the associations between entities in the database. The function takes as input the course code and returns a table with the names of the students participating in ascending order based on their AM (ASC).

E.g. SELECT * FROM get_students_fullname_from_course('ΠΛΗ 102');;

- 3.3

The ***get_fullname_and_positions()*** function retrieves information by **UNIONing** multiple tables in the system using the corresponding AMKA fields. It joins the Person table with the Student, Professor and LabTeacher tables based on AMKA and eventually returns a table with all the persons registered in the database with the position of each one next to it.

E.g. SELECT get_fullname_and_positions();.

- 3.4

The ***get_obligatory_courses(student_am, program_id)*** function returns the courses that are mandatory for a particular student in a particular program of study. First, it selects the required courses from the Course table and excludes courses that the student has 'passed' based on the Register, Student, Joins and ProgramOffersCourse tables.

E.g. SELECT * FROM get_obligatory_courses('2010000001', 1);;

- 3.5

The get_sector_labs() function returns the sectors sorted in descending order by the number of labs that have been run for each sector. The total number of labs for each sector is obtained by joining the Sector and Lab tables and joining Lab with CourseRun. The resulting total of records is the total number of workshops that have been conducted, which are grouped by the sector to which they belong.

- 3.6

The ***get_qualified_students(program_id)*** function returns all students belonging to a particular program of study who are qualified for graduation. That is, they have passed the minimum required number of courses, completed the minimum required number of credits, and in case the program requires a thesis, if the student has completed it with a grade >= 5.

- 3.7

The ***get_lab_hours_per_teacher()*** function retrieves the total lab hours for each teacher. By using INNER JOIN we were able to get all the Persons that exist in the database and are in the Supports table (since we are talking about the workload of the lab instructors). Since there are Person records in the database that do not correspond to 1 of the 3 Person categories, we added using UNION union those Person records where they do not have an identity with workload being 0.

E.g SELECT get_lab_hours_per_teacher();;

- 3.8

The ***get_all_courses_related_to(c_code)*** function retrieves all courses related to a particular course by its code. We use recursion to be able to recursively identify dependent and recommended courses. The function returns all courses that are in some way, directly or indirectly, related to the one given as an argument.

E.g. SELECT get_all_courses_related_to('THΛ 302');

- 3.9

The get_professors() function returns the details of all professors who have taught at least one course for each different type of curriculum (standard, seasonal, foreign language). For standard and foreign language programmes, it checks whether the courses conducted within the 10 semesters of the programme were taught by a particular professor, while for the seasonal programme, it checks the courses of the given semester. If a teacher's entry is found in all 3 cases, the function will return the teacher's data.

- 4.1.1-4.1.2 (Trigger)

The ***future_sem_trigger*** trigger acts on the Semester table. Its job is to update the Semester table correctly in order to have the university function correctly. Initially, the function *check_future_semester()* has been implemented*, which* implements the functionality of the trigger. More specifically, in case of inserting a future Semester, it is called to make decisions whether the insertion is correct or not, based on some commands that check the dates. Finally, in case of updating this future semester to present, the previous present semester is automatically assigned to past.

E.g

```
-- Insert New Future Semester
INSERT INTO "Semester"
(semester_id, academic_year, academic_season, start_date, end_date, semester_status)
VALUES (27, 2023, 'winter', '2023-10-01', '2024-01-15', 'future');;

-- Set it 'present'
UPDATE "Semester"
SET semester_status = 'present'
WHERE semester_id = 27
```

- 4.1.3 (Trigger)

The trigger ***create_proposed_registries_trigger*** acts on the Semester table. The function that implements the trigger is create_proposed_registries(). When a future semester becomes present, this function creates proposed student registrations in the current semester's courses. This is done by updating the CourseRun table and inserting new registrations into the Register table, excluding students who have 'pass' status and previously shared courses.

- 4.1.4 (Trigger)

The ***calculate_final_student_grades_trigger*** is responsible for being triggered when a previous semester transitions from present to past state and is called to calculate the final student grades for that semester. The function insert_random_grades(semester_id) of query 2.1 is suitable for this function. It is worth noting that for a final grade to be calculated, the course registration must be in 'approved' status.

- 4.2

The check for the import of programs has not been implemented in a trigger as its functionality has already been implemented in the create_program(...)function of exercise 2.3.The check for the maximum allowed number of committee members is implemented by the trigger committee_trigg which is activated after the import of members in the Committee table. Using the function check_committee_num if the new table resulting after the imports exceeds the maximum number of members of the thesis, the import will not be performed.

- 5.1 (View)

The ***semester_table*** view shows the course codes and titles, as well as the names of the instructors for the semester courses of the current semester. To implement this we had to link several tables together and check that AMKA is not null and that the semester stauts is the present one. In more detail we used INNER JOIN between the tables and also used the string_agg() function which combines the names of the professors into one string per course.

E.g. SELECT * FROM semester_table;