HACETTEPE UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

BBM473: DATABASE LABORATORY, SPRING 2023

PHASE 2 - Stored Procedures, Views, Triggers, and SQL queries

---

Group ID = 20

Student names:	*Student numbers:*
Mert DOĞRAMACI	21946055
Özgün AKYÜZ	21827005

Course Management System

Functionalities

1. Procedures:

- *insert_[table name] stored procedures*

The purpose of an insert stored procedure is to provide a standardized and efficient way to add new data to a database table. Instead of writing individual SQL statements to insert each new record, a stored procedure can be created to handle this process. When a new record needs to be added to the table, the application or user can call the insert stored procedure with the required parameters, such as the values for each column in the table. The stored procedure then executes the SQL statements necessary to insert the new record, providing consistency and security by ensuring that the data is properly validated, formatted, and sanitized before being added to the database.

- *update_[table name] stored procedures*

The purpose of an update stored procedure is to provide a standardized and efficient way to modify existing data in a database table. Instead of writing individual SQL statements to update each record, a stored procedure can be created to handle this process. When a record needs to be updated in the table, the application or user can call the update stored procedure with the required parameters, such as the ID of the record to be updated and the new values for each column in the table. The stored procedure then executes the SQL statements necessary to update the record, providing consistency and security by ensuring that the data is properly validated, formatted, and sanitized before being modified in the database.

- *delete_[table name] stored functions*

The purpose of a delete stored procedure is to provide a standardized and efficient way to remove data from a database table. Instead of writing individual SQL statements to delete each record, a stored procedure can be created to handle this process. When a record needs to be deleted from the table, the application or user can call the delete stored procedure with the required parameters, such as the ID of the record to be deleted. The stored procedure then executes the SQL statements necessary to delete the record, providing consistency and security by ensuring that the data is properly validated and sanitized before being removed from the database. Delete stored procedures can also be used to perform additional operations before or after the delete, such as updating related tables or logging the deletion event. This helps to ensure data integrity and consistency throughout the database. Additionally, delete stored procedures

can be optimized for performance by using techniques such as optimized queries or cascading deletes, which can improve the speed and efficiency of the deletion process.

- *Triggers*

In our work, we used the trigger functions to automatically increment the primary key fields in our tables.

2. Views:

2.1. student_grades

This view returns the grades of the entire students that are registered in the system. Combining this view with a "where" statement allows us to see all the grades of the students that are enrolled in the same section. It combines the "student" and "student_enrolls_section" tables. The student table returns basic information such as ID, name, surname, departmentID, studentID, etc., and the student_enrolls_section table returns pairs of studentID and sectionIDs with grade information. So, we can use it to see the grades for all students enrolled in a specific section via the following query:

```
SELECT * FROM student_grades WHERE sectionID = 3;
```

2.2. instructor_sections

This view returns all sections taught by all instructors. Combining this view with a "where" statement allows us to see all the sections that are taught by the same instructor. It combines the "instructor", "instructor_teaches_section", and "section" tables. The instructor returns basic information such as ID, name, surname, departmentID, etc.; the instructor_teaches_section returns pairs of instructorID and sectionIDs; and the section returns information such as semester, year, etc. So, we can use it to see all sections taught by a specific instructor via the following query:

```
SELECT * FROM instructor_sections WHERE instructor_ID = 1;
```

The last three views are for gaining some statistics that any user may be curious about.

2.3. department_course_stats

This view returns statistics about the courses offered by a specific department, such as the average ECTS credits and the number of sections offered. It combines the "department", "course", and "section" tables. The department table returns ID and department name; the course table returns course title and ECTS; and the section table returns section IDs. So, the query can count the sections by using the distinct section IDs. Also, it can calculate the average of the ECTS for the courses. Finally, we can see the average ECTS and number of sections that are offered by a specific department. In short, we can use it to retrieve statistics about the courses offered by a specific department, such as the average ECTS credits and the number of sections offered, via the following query:

```
SELECT * FROM department_course_stats WHERE department_name =
                    'Computer Engineering';
```

2.4. student_submission_stats

This view returns statistics about the submissions made by a specific student, such as the number of submissions and the average marks received. Combining this view with a "where" statement allows us to see the specific submissions and the average marks received that belong to the same student. It combines the "student", "student_has_submission", and "submission" tables. The student table returns ID, name, surname, etc.; the student_has_submission table returns the pairs of studentID and submissionIDs, etc.; and the submission table returns the submission ID and mark to the submission, etc. So, the query can count the submissions by using the distinct submission IDs. Also, it can calculate the average of the marks that are gained from the submissions. In short, we can use it to retrieve statistics about the submissions made by a specific student, such as the number of submissions and the average marks received, via the following query:

```
SELECT * FROM student_submission_stats WHERE student_ID = 1;
```

2.5. department_section_stats

This view returns the statistics about the sections offered by all departments, such as the average grade and the number of students enrolled. Combining this view with a "where" statement allows us to see the statistics about the specific department instead of all of them. It combines the “department”, “course”, “section”, and “student_enrolls_section” tables. The department table returns ID, and name; the course table returns ID; the section table returns ID; the student_enrolls_section table returns pairs of studentID and sectionIDs, and grade. Also, it can calculate the average of the grades that are gained by the students and count of the students that

are enrolled in that section. In short, we can use it to retrieve statistics about the sections offered by a specific department, such as the average grade and the number of students enrolled, via the following query:

```
SELECT * FROM department_section_stats WHERE department_name =
                    'Computer Engineering';
```

Revision of The Tables

- The “type” attribute of the course entity is renamed as “isCompulsory” because we considered it is better to use this attribute as a boolean.
- We removed the user table and created two separate tables as student and instructor. Because of the limitations of PostgreSQL, we are not allowed to use inheritance and foreign keys at the same time. So, we removed the inheritance relation between user, student, and instructor.
- Since we removed the user table, we have to divide the login_credentials and contact_info tables into two tables such as student_login_credentials - instructor_login_credentials and student_contact_info - instructor_contact_info. Because we were not able to use the userID attribute.