You should work on the following assignments in fixed teams of two. Please note that *every* team member must be able to explain *all* solutions of the team of two. Please submit only one solution for each team of two.

**Deadline to upload your solution for assignments 1 and 2:**

**Sunday, 11:59 pm bevor the laboratory.**

The remaining assignments can be done during the laboratory.

If you have questions or need any help, use the forum in our EMIL room und help each other.

*Assignment 1: SQL-statements for the Student Information System*

Consider the following relational schema for the Student Information System (analogous to the Assignments from Lab 1 and 2):

**STUDENT**(studentID, fistName, lastName, dob, programID(FK))

**PROGRAM**(programID, name, requiredCPs)

**COURSE**(courseID, name, description, creditPoints, programID(FK))

**ATTEMPTS**(studentID(FK),courseID(FK), year, term, grade)

**PREREQUISITE**(advancedCourseID(FK), prerequisiteCourseID (FK))

* 1. Write SQL-statements that create the corresponding tables. Come up with reasonable constraints and datatypes for the fields of the tables.

Solution:

CREATE TABLE Program

(programID INT NOT NULL,

name VARCHAR(32) NOT NULL,

requiredCPs INT NOT NULL,

PRIMARY KEY (programID) );

CREATE TABLE Student

(studentID INT NOT NULL,

firstName VARCHAR(32) NOT NULL,

lastName VARCHAR(32) NOT NULL,

dob DATE NOT NULL,

programID INT NOT NULL,

PRIMARY KEY (studentID),

FOREIGN KEY (programID) REFERENCES Program(programID) );

CREATE TABLE Course

(CourseID INT NOT NULL,

name VARCHAR(32) NOT NULL,

description VARCHAR(100) NOT NULL,

creditPoints INT NOT NULL,

programID INT NOT NULL,

PRIMARY KEY (courseID),

FOREIGN KEY (programID) REFERENCES Program(programID) );

CREATE TABLE Attempts

(studentID INT NOT NULL,

courseID INT NOT NULL,

year INT NOT NULL,

term INT NOT NULL,

grade INT NOT NULL,

PRIMARY KEY (studentID, coursed, year, term),

FOREIGN KEY (studentID) REFERENCES Student(studentID),

FOREIGN KEY (courseID) REFERENCES Course(courseID) );

CREATE TABLE Prerequisite

(advancedCourse INT NOT NULL,

prerequisiteCourse INT NOT NULL,

PRIMARY KEY (advancedCourse, prerequisiteCourse),

FOREIGN KEY (advancedCourse) REFERENCES Course(courseID),

FOREIGN KEY (prerequisiteCourse) REFERENCES Course(courseID) );

* 1. Write SQL-queries that insert example data into your created tables. Make sure that each table contains at least 2 rows of data. Here are some sample data.

Table **STUDENT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| studentID | firstName | lastName | dob | programID |
| 123456 | John | Wayne | 11.05.1998 | 1 |
| 234567 | Anna | Meyer | 13.02.1999 | 1 |
| … | … | … | … | … |

Table **PROGRAM**

|  |  |  |
| --- | --- | --- |
| programID | Name | requiredCPs |
| 1 | Information Engineering | 120 |
| 2 | Renewable Energies | 110 |
| … | … | … |

Table **COURSE**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| courseID | Name | Description | creditPoints | programID |
| 4 | MA1 | Mathematics 1 | 8 | 1 |
| 9 | MA2 | Mathematics 2 | 8 | 1 |
| 13 | SS1 | Signals and Systems 1 | 6 | 1 |
| 15 | DB | Databases | 6 | 1 |
| … | … | … | … | … |

Table **PREREQUISITE**

|  |  |
| --- | --- |
| advancedCourse | prerequisiteCourse |
| 9 | 4 |
| 13 | 9 |
| 13 | 4 |
| … | … |

Table **ATTEMPTS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| studentID | courseID | Year | Term | grade |
| 123456 | 4 | 2021 | 1 | 7 |
| 123456 | 9 | 2021 | 2 | 9 |
| 123456 | 13 | 2022 | 1 | 3 |
| 123456 | 13 | 2022 | 2 | 6 |
| … | … | … | … | … |

* 1. Write a SQL-query for the created database that returns all students (first name + last name) that study the program “Information Engineering”.

SELECT s.firstName, s.lastName   
FROM student s, program p   
WHERE p.name = “Information Engineering” AND s.programID = p.ProgramID;

* 1. Write a SQL-query that returns the name of all courses that have prerequisite courses.

SELECT DISTINCT c.name   
FROM Course c, Prerequisite p   
WHERE c.courseID = p.advancedCourse;

* 1. Write a SQL-query that returns the sum of all credit points successfully achieved by student “John Wayne”. Keep in mind that the credit points only count when the student has an attempt with a grade of 5 or more points.

SELECT SUM(c.creditPoints)

FROM Course c, Attempts a, Student s

WHERE c.courseID = a.courseID

AND a.grade >= 5

AND a.studentID = s.studentID

AND s.firstName = “John”

AND s.lastName = “Wayne”;

* 1. If we have already talked about deleting data in the lecture: A student needs to be removed from the database. Write SQL-queries to remove the student with the name “John Wayne” from the database.

DELETE FROM attempts a

WHERE a.studentID IN (

SELECT s.studentID FROM Student s

WHERE s.firstName = “John”

AND s.lastName = “Wayne”);

DELETE FROM Student

WHERE firstName = “John”

AND lastName = “Wayne”;

*Assignment 2: SQL-statements for a Shipping company*

A shipping company wants to use a SQL-database to keep track of its ships and employed sailors based on the following relation schema:

**HARBOR**(harborID, location, establishedIn)

**SAILOR**(sailorID, lastName, dob, trainedAt(FK -> harborID))

**SHIP**(shipID, name, grossWeight, launchDate, baseHarbor(FK -> harborID))

**HIRE**(sailor(FK -> sailorID), ship(FK -> shipID), startOfService, annualSalary)

You can use the provided SQL-script for creating the tables and inserting some data in the tables.

* 1. Create a SQL-query that returns the dob (date of birth) of sailors in descending order that were hired on August 3rd, 2012.

SELECT lastname, dob

from sailor s, hire h

where s.sailorID = h.sailorID AND

h.startOfService = '2012-08-03'

ORDER BY dob DESC;

* 1. Create a SQL-query that returns all information of sailors that were hired between July 3rd, 2011, and September 3rd, 2012 and which last name starts with a ‘J’.

SELECT \*

from sailor s, hire h

where s.sailorID = h.sailorID AND

h.startOfService BETWEEN '2011-07-03' AND '2012-09-03'

AND lastname like 'J%';

* 1. Create a SQL-query that returns for each ship the sum of the annual salary of every sailor who is hired for that ship.

SELECT s.name, SUM(h.annualSalary)

FROM ship s, hire h

WHERE s.shipID = h.shipID

GROUP BY s.shipID;

* 1. Create a SQL-query that returns the location of all harbors that are not base harbor to any ship in the database.

Select h.location

FROM Harbour h

where h.harbourID NOT IN (SELECT s.baseharbour from Ship s);

* 1. Create a SQL-query that returns the shipId, ship name and the number of sailors who are hired on the ship and earn maximum 42.000$.

select h.shipid, s.name, COUNT(h.sailorid)

FROM ship s, hire h

WHERE h.shipid = s.shipid

GROUP BY h.shipid

HAVING MAX(h.annualSalary) < 42000;

* 1. Describe in your own words the result of the following query:

**SELECT DISTINCT** h1.location

**FROM** SHIP s1, SHIP s2, HARBOR h1, HARBOR h2

**WHERE** s1.baseHarbor = h1.harborID

**AND** s2.baseHarbor = h2.harborID

**AND** s1.launchDate = s2.launchDate

**AND** h1.location = h2.location

**AND** h1.harborID != h2.harborID;

Name of all Locations at which two Harbours are situated, which are the baseHarbour for two different Ships with the same launchingDate.

*Assignment 3: SQL-statements for the COMPANY example from Elmasri also used in the lecture*

Let’s have a look on the COPMPANY example from the book „Fundamentals of Database Systems“ from Elmasri which is also used in the lecture. Given is the database schema in Figure 1 and the database state in Figure 2.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Figure : Schema diagram for the COMPANY relational database schema (see Elmasri, page 71)

Ein Bild, das Text, Quittung enthält.

Automatisch generierte Beschreibung

Figure : A possible database state for the COMPANY relational database schema (see Elmasri, page 72)

Write SQL statement for the following tasks:

3.1 Retrieve the names of all employees in department 5 who work more than 10 hours

per week on a project.  
  
select fname, lname

FROm Employee, works\_on

WHERE ssn = essn AND dno = 5 AND Hours > 10;

3.2 List the names of all employees who have a dependent with the same first name as

themselves.

SELECT fname, lname

FROM employee, dependent

WHERE Dependent\_name = fname AND ssn=ESSN;

3.3 Find the names of all employees who are directly supervised by ‘Franklin Wong’.

SELECT e2.fname, e2.lname

FROM employee e1, employee e2

WHERE e1.fname = 'Franklin' and e1.lname = 'Wong'

AND e1.ssn=e2.Super\_ssn;

3.4 Suppose that the EMPLOYEE table’s constraint EMPSUPERFK as specified in Figure 3 is changed to read as follows:

**CONSTRAINT** EMPSUPERFK

**FOREIGN KEY** (Super\_ssn) **REFERENCES** EMPLOYEE(Ssn)

**ON DELETE CASCADE ON UPDATE CASCADE**;

Answer the following questions:

* What happens when the following command is run on the database state shown in Figure 2?

**DELETE** FROM EMPLOYEE **WHERE** Lname = ‘Borg’ ;

Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails (`company\_2023`.`department`, CONSTRAINT `DEPTMGRFK` FOREIGN KEY (`Mgr\_ssn`) REFERENCES `employee` (`Ssn`) ON DELETE RESTRICT ON UPDATE CASCADE)

* Is it better to **CASCADE** or **SET NULL** in case of EMPSUPERFK constraint **ON**

**DELETE**?

🡪 **SET NULL**

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Figure : Database schema 2 (see Elmasri, page 95)

3.5 For each project, list the project name and the total hours per week (by all

employees) spent on that project.

SELECT pname, SUM(hours)

FROM project, works\_on

where pnumber = pno

group by pname;

3.6 Retrieve the average salary of all female employees.

3.7. Write SQL statements to create a table EMPLOYEE\_BACKUP to back up the

EMPLOYEE table shown.

CREATE TABLE EMPLOYEE\_BACKUP SELECT \* FROM employee;

3.8. For each department, whose average employee salary is more than $30,000,

retrieve the department name and the number of employees working for that

department.

SELECT dname, dno, count(ssn), AVG(salary)

FROM employee, department

where dno = dnumber

GROUP BY (dno)

having AVG(salary) > 30000;