CS-A1153 - Databases, Summer 2020
Project, part 2 (SQL)

Alberto Nieto Sandino : alberto.nietosandino@aalto.fi
Peiyi Zhao : peiyi.zhao@aalto.fi

August 28, 2020

Contents

1 UML diagram

The following is the UML diagram (Figure 1) that we develop for a database given the requirements of the project. Further explanations regarding the diagram and its components are given in the next section.

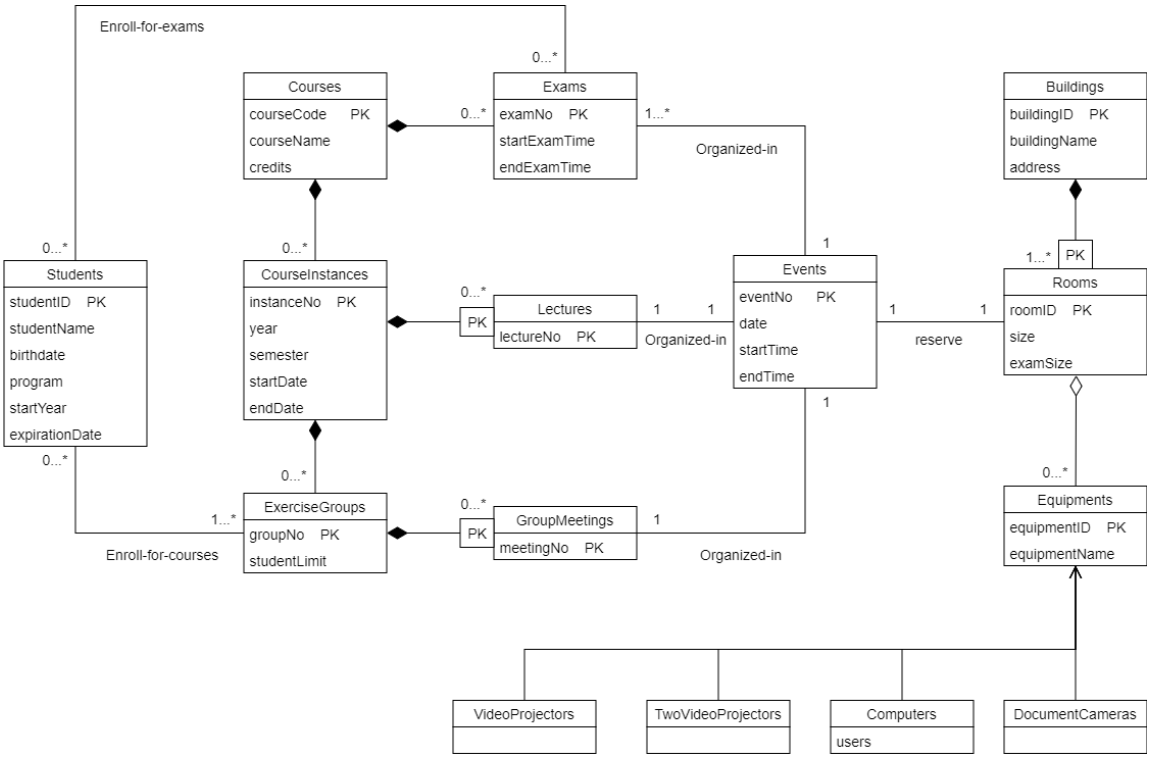

Figure 1: UML diagram for database

2 Relation schema converted from the UML diagram

Below is the relation schema for the UML diagram of the database.

- Courses (courseCode, courseName, credits)
- CourseInstances (instanceNo, year, semester, startDate, endDate, courseCode)
- ExerciseGroups (groupNo, studentLimit, instanceNo)

- Lectures (instanceNo, lectureNo, eventNo)
- GroupMeetings (groupNo, meetingNo, eventNo)
- Exams (examNo, startExamTime, endExamTime, courseCode, eventNo)

- Students (studentID, studentName, birthdate, program, startYear, expirationDate)
- Enroll-for-exams (studentID, examNo)
- Enroll-for-courses (studentID, groupNo)

- Buildings (buildingID, buildingName, address)
- Rooms (buildingID, roomID, size, examSize)

- Events (eventNo, date, startTime, endTime)
- Reserve (eventNo, buildingID, roomID)

- Equipments (equipmentID, equipmentName, buildingID, roomID)
- VideoProjectors (equipmentID)
- TwoVideoProjectors (equipmentID)
- Computers (equipmentID, users)
- DocumentCameras (equipmentID)

3 Explanations

Courses is a class including the common information for one course like code, name, and credits. CourseInstances has the basic information about each course implementation. It is part of the Courses, and it could not exist without Courses, so the relationship between them is composition. So does ExerciseGroup.

Exams belongs to Courses. Lectures belongs to CourseInstances. GroupMeetings belongs to ExerciseGroups. All the relationship between these three pairs is many-one relationship.

Events is a class to record the time information when some activity happens in a certain place at a certain time. So, one lecture or one GroupMeeting could be an event. But several exams may share an event because they could take place in one room at the same time.

One event could reserve one room. So, there is a one-one association between Events and Rooms.

Students is a class which stores the information of students. It has two associations. One is with Exams. The other is with ExerciseGroups. These two associations could be used to register the course or exam for students.

Rooms is a part of Buildings and could not exist without Buildings. So, the relationship between them is composition. Equipments is also a part of Rooms, but it can exist without rooms. So, the relationship between them is aggregation.

VideoProjectors, TwoVideoProjectors and so on are one type of Equipments. So, they are the subclass of Equipments. And other type of information could be stored in the superclass Equipments.

The aforementioned UML diagram was designed so as to support a series of queries that were described in the project requirements. Following is an explanation of how each of these queries is supported by our designed database.

The database supports storing information about courses, course instances, students, lectures, exercises groups and their times, exams and enrollments. Courses, course instances, students, lectures, exercise groups and exams have their own classes where information about them is stored. Information about the meeting times for exercise groups is stored in the GroupMeetings class. Enrollments, both for exams and for courses, are represented as associations:

- Enrollment for an exam is an association between the Students and Exams class.
- Enrollment for a courses is an association between the Students and ExerciseGroups.

The database supports storing information about buildings, rooms and their reservations. Buildings and rooms store their information in the Buildings classes and Rooms classes separately. The class Events contains the date and time information of every activity including exams, lectures and group meetings. Each event has a unique EventNo. If several exams take place at the same time in one room, they will be regarded as one event and get the same eventNo. By joining the class Events and the class Reserve, we could find a certain event reserves a certain room in a certain time. includes reservations for exams, reservations for lectures and reservation for group meetings.

Courses are composed of the classes Exams and CourseInstances. A CourseInstances class is a given course organized in a certain semester. Each course instance has a unique instance number so that it could be recognized even if there are several course instances for the same course in one semester. Each of these course instances is composed of ExerciseGroups and Lectures. ExerciseGroups are composed of GroupMeetings which is associated with Events. Lectures is associated with events. These two latter associations make it possible to keep track of when a certain lecture or group meeting from a given course instance of a course took place or will take place at a specific time and date.

If we look at a certain time interval in events, we can have multiple lectures of different course instances and multiple group meetings of different exercise groups at the same time as long as their roomID is different. This is shown as the association reserve between Events and Rooms. Thus, for a given time interval, we can look up which courses have a lecture reservation in the Events class.

Through the association between Exams and Events, we can see which exams (of a given Course) take place at a certain time interval. In the Exams class, it is specified the officialduration of the exam (start and end), while in the reservation in Events, the time is longer since there is a need for practical arrangements both before and after the exam. In order to see when a certain course will or has been arranged, one can look up the course instance class. Since course is composed of course instance, it is possible to look up a certain course code (foreign key in courseInstance which references the primary key in the class Courses) and see the year and semester when course instances of that course may be taking place.

Course instances are composed of Lectures and Exercise groups. Thus, it is straightforward to look up which lectures are associated with which course instances.

Course instances are also composed of Exercised groups, thus we can search which groups belong to which course instance. Many exercise groups are possible for one course instance.

Once the exercise group is fixed, we can get the eventNo and find the time when this group meets by looking at Group meetings and its association with events. Through checking the eventNo in the class Reserve, we can find which room (in a certain building) the group meeting is taking place in.

To select a room with a certain number of seats and free at a certain time. We can first filter the Rooms class for rooms with a certain size (i.e. the number of seats). Then, through its association with Events, we can look at which times these selected rooms are not booked.

If we have a certain roomID with buildingID and a start and end time of a reservation from Events, we can look up if that corresponds to an exam, a lecture or group meeting (i.e. an exercise group) by joining these latter 3 classes with that time slot and seeing if there are any tuples in common. That is tuples where that would be associated with that time slot and room.

Enrolling of students take place with two associations:

- Association Enroll-for-courses which enrols the Student class into the ExerciseGroup, thus automatically signing up for a certain course.
- Association Enroll-for-exams which enrols the Student class into the Exam for a certain Course.

To list all students enrolled in a course instance, we can list all of the students who are enrolled in exerciseGroups of that certain course instance. To list all students enrolled in an exercise group, we can list the students present in that certain Exercise group (with unique groupNo). To list all students enrolled in an exam, we can use the Enroll-for-exams association where each student is associated with a unique exam.

To see if a certain ExerciseGroup is full, we can inspect its studentLimit attribute and count the tuples grouped by the groupNo. By comparing these two numbers, we could see whether the limit has been reached or if more students can be enrolled into that ExerciseGroup.

4 Normal form and functional dependencies

The following are the functional dependencies from our database.

- courseCode → courseName credits
- instanceNo → year semester startDate endDate courseCode
- groupNo → studentLimit instanceNo
- instanceNo lectureNo → eventNo
- groupNo meetingNo → eventNo
- examNo → startExamTime endExamTime courseCode eventNo
- studentID → studentName birthdate program startYear expirationDate
- buildingID → buildingName address
- buildingID roomID → size examSize

- eventNo → date startTime endTime
- equipmentID → equipmentName buildingID roomID

Because all the left side is a super key of the relation, it is in BCNF.

5 Relation schema in SQL

Below is the SQL code to create the relation schema in SQLiteStudio. The following is the creation of all the relations (i.e. tables) that are part of the database.

```
CREATE TABLE Courses (
    courseCode TEXT PRIMARY KEY,
    courseName TEXT NOT NULL,
    credits    REAL CHECK (credits > 0)
);

CREATE TABLE CourseInstances (
    instanceNo TEXT    PRIMARY KEY,
    year       INTEGER CHECK (year > 2000),
    semester   INTEGER CHECK (semester IN (1, 2) ),
    startDate  TEXT,
    endDate    TEXT,
    courseCode TEXT    REFERENCES Courses (courseCode)
);

CREATE TABLE ExerciseGroups (
    groupNo      TEXT    PRIMARY KEY,
    studentLimit INTEGER CHECK (studentLimit > 0),
    instanceNo   TEXT    REFERENCES CourseInstances (instanceNo)
);

CREATE TABLE Events (
    eventNo   TEXT PRIMARY KEY,
    date      TEXT,
    startTime TEXT,
    endTime   TEXT
);

CREATE TABLE Lectures (
    instanceNo TEXT,
    lectureNo  TEXT,
    eventNo    TEXT UNIQUE,
    PRIMARY KEY (
        instanceNo,
        lectureNo
    ),
    FOREIGN KEY (
        instanceNo
    )
    REFERENCES CourseInstances (instanceNo),
    FOREIGN KEY (
        eventNo
    )
    REFERENCES Events (eventNo)
);

CREATE TABLE GroupMeetings (
    groupNo   TEXT,
    meetingNo TEXT,
    eventNo   TEXT UNIQUE,
    PRIMARY KEY (
```

```
        groupNo,
        meetingNo
    ),
    FOREIGN KEY (
        groupNo
    )
    REFERENCES ExerciseGroups (groupNo),
    FOREIGN KEY (
        eventNo
    )
    REFERENCES Events (eventNo)
);

CREATE TABLE Exams (
    examNo        TEXT PRIMARY KEY,
    startExamTime TEXT,
    endExamTime   TEXT,
    courseCode    TEXT NOT NULL,
    eventNo       TEXT NOT NULL,
    FOREIGN KEY (
        courseCode
    )
    REFERENCES Courses (courseCode),
    FOREIGN KEY (
        eventNo
    )
    REFERENCES Events (eventNo)
);

CREATE TABLE Students (
    studentID      TEXT    PRIMARY KEY,
    studentName    TEXT,
    birthdate      TEXT,
    program        TEXT,
    startYear      INTEGER,
    expirationDate TEXT
);

CREATE TABLE EnrollForExams (
    studentID TEXT,
    examNo    TEXT,
    PRIMARY KEY (
        studentID,
        examNo
    ),
    FOREIGN KEY (
        studentID
    )
    REFERENCES Students (studentID),
    FOREIGN KEY (
        examNo
    )
    REFERENCES Exams (examNo)
);

CREATE TABLE EnrollForCourses (
    studentID TEXT,
    groupNo   TEXT,
    PRIMARY KEY (
        studentID,
        groupNo
    ),
    FOREIGN KEY (
        studentID
    )
    REFERENCES Students (studentID),
```

```
    FOREIGN KEY (
        groupNo
    )
    REFERENCES ExerciseGroups (groupNo)
);

CREATE TABLE Buildings (
    buildingID   TEXT PRIMARY KEY,
    buildingName TEXT,
    address      TEXT UNIQUE
);

CREATE TABLE Rooms (
    buildingID TEXT,
    roomID     TEXT,
    size       INTEGER,
    examSize   INTEGERR,
    PRIMARY KEY (
        buildingID,
        roomID
    ),
    FOREIGN KEY (
        buildingID
    )
    REFERENCES Buildings (buildingID),
    CHECK (size >= examSize)
);

CREATE TABLE Equipments (
    equipmentID   TEXT PRIMARY KEY,
    equipmentName TEXT NOT NULL,
    buildingID    TEXT,
    roomID        TEXT,
    FOREIGN KEY (
        buildingID,
        roomID
    )
    REFERENCES Rooms (buildingID,
    roomID)
);


CREATE TABLE VideoProjectors (
    equipmentID TEXT PRIMARY KEY,
    FOREIGN KEY (
        equipmentID
    )
    REFERENCES Equipments (equipmentID)
);

CREATE TABLE TwoVideoProjectors (
    equipmentID TEXT PRIMARY KEY,
    FOREIGN KEY (
        equipmentID
    )
    REFERENCES Equipments (equipmentID)
);

CREATE TABLE Computers (
    equipmentID TEXT PRIMARY KEY,
    users       TEXT NOT NULL,
    FOREIGN KEY (
        equipmentID
    )
    REFERENCES Equipments (equipmentID)
);
```

```
184 |
185 | CREATE TABLE DocumentCameras (
186 |     equipmentID TEXT PRIMARY KEY,
187 |     FOREIGN KEY (
188 |         equipmentID
189 |     )
190 |     REFERENCES Equipments (equipmentID)
191 | );
192 |
193 | CREATE TABLE Reserve (
194 |     eventNo    TEXT UNIQUE,
195 |     buildingID TEXT NOT NULL,
196 |     roomID     TEXT NOT NULL,
197 |     FOREIGN KEY (
198 |         eventNo
199 |     )
200 |     REFERENCES Events (eventNo),
201 |     FOREIGN KEY (
202 |         buildingID,
203 |         roomID
204 |     )
205 |     REFERENCES Rooms (buildingID,
206 |     roomID)
207 | );
```

6 Insert data into the database

Here, we insert all the tuples we may need to make use of the database and investigate its use cases.

```
01 | INSERT INTO Courses
02 | VALUES ('CS-A1150', 'Databases', 5);
03 |
04 | INSERT INTO Courses
05 | VALUES ('CS-E4800', 'Artificial Intelligence', 5);
06 |
07 | INSERT INTO Courses
08 | VALUES ('CS-E4610', 'Modern Database Systems', 5);
09 |
10 | INSERT INTO Courses
11 | VALUES ('CS-C3100', 'Computer Graphics', 5);
12 |
13 | INSERT INTO CourseInstances
14 | VALUES ('CS-A1150-2020-1', 2020, 1, '2020-09-03', '2020-11-20', 'CS-A1150');
15 |
16 | INSERT INTO CourseInstances
17 | VALUES ('CS-E4800-2020-1', 2020, 1, '2020-09-01', '2020-12-27', 'CS-E4800');
18 |
19 | INSERT INTO CourseInstances
20 | VALUES ('CS-E4610-2020-1', 2020, 1, '2021-09-07', '2021-11-23', 'CS-E4610');
21 |
22 | INSERT INTO CourseInstances
23 | VALUES ('CS-E4610-2020-2', 2020, 2, '2021-01-05', '2021-04-05', 'CS-E4610');
24 |
25 | INSERT INTO ExerciseGroups
26 | VALUES ('CS-A1150-group1', 10, 'CS-A1150-2020-1');
27 |
28 | INSERT INTO Events
29 | VALUES ('20200903L1', '2020-09-03', '09:00', '12:00');
30 |
31 | INSERT INTO Events
32 | VALUES ('20200904M1', '2020-09-04', '10:00', '11:00');
33 |
34 | INSERT INTO Events
```

```
35 | VALUES ('20201125E1', '2020-11-25', '08:30', '09:30');
36 |
37 | INSERT INTO Lectures
38 | VALUES ('CS-A1150-2020-1', 'lecture1', '20200903L1');
39 |
40 | INSERT INTO Lectures(instanceNo, lectureNo)
41 | VALUES ('CS-A1150-2020-1', 'lecture2');
42 |
43 | INSERT INTO GroupMeetings
44 | VALUES ('CS-A1150-group1', 'meeting1', '20200904M1');
45 |
46 | INSERT INTO GroupMeetings(groupNo, meetingNo)
47 | VALUES ('CS-A1150-group1', 'meeting2');
48 |
49 | INSERT INTO Exams
50 | VALUES ('CS-A1150-exam1', '08:40', '09:30', 'CS-A1150', '20201125E1');
51 |
52 | INSERT INTO Students
53 | VALUES ('112233', 'Teemu Teekkari', '1990-05-11', 'TIK', '2017', '2021-08-30');
54 |
55 | INSERT INTO Students
56 | VALUES ('123456', 'Tiina Teekkari', '1992-09-23', 'TUO', '2019', '2023-08-30');
57 |
58 | INSERT INTO Students
59 | VALUES ('224411', 'Maija Virtanen', '1991-12-05', 'AUT', '2018', '2022-08-30');
60 |
61 | INSERT INTO EnrollForExams
62 | VALUES ('112233', 'CS-A1150-exam1');
63 |
64 | INSERT INTO EnrollForExams
65 | VALUES ('224411', 'CS-A1150-exam1');
66 |
67 | INSERT INTO EnrollForCourses
68 | VALUES ('112233', 'CS-A1150-group1');
69 |
70 | INSERT INTO Buildings
71 | VALUES ('building1', 'Computer Science Building', 'Tietotekniikantalo, Konemiehentie
       2, 02150 Espoo');
72 |
73 | INSERT INTO Buildings
74 | VALUES ('building2', 'Engineering Physics Building', 'Tietotekniikantalo,
       Konemiehentie 4, 02150 Espoo');
75 |
76 | INSERT INTO Rooms
77 | VALUES ('building1', 'room101', 30, 20);
78 |
79 | INSERT INTO Rooms
80 | VALUES ('building1', 'C206', 45, 30);
81 |
82 | INSERT INTO Rooms
83 | VALUES ('building2', 'C310', 50, 30);
84 |
85 | INSERT INTO Equipments
86 | VALUES ('equipment1', 'MacComputer', 'building1', 'room101');
87 |
88 | INSERT INTO Equipments
89 | VALUES ('equipment2', 'SONY VideoProject', 'building1', 'room101');
90 |
91 | INSERT INTO VideoProjectors
92 | VALUES ('equipment2');
93 |
94 | INSERT INTO Computers
95 | VALUES ('equipment1', 'teachers');
96 |
97 | INSERT INTO Reserve
98 | VALUES ('20200903L1', 'building1', 'room101');
```

```
99 |
100 |  INSERT INTO Reserve
101 |  VALUES ('20201125E1', 'building1', 'C206');
```

7 Indexes and views

7.1 Indexes

In order to create reasonable indexing, we need to know or have an educated guess of what the typical queries to the database may be. Based on an educated guess, we know that the number of students in the database will be orders of magnitude larger than other agents in this database like courses, equipment, lectures and so on. Student's activities such as registering in courses and exams will also be more frequent than operations such as instantiating a course or registering a schedule. Thus, studentID can be a powerful attribute to use as an index in the relation Students.

```
01 |  CREATE INDEX idx_student ON Students (
02 |  studentID
03 |  );
```

Looking at the use cases (Section 8), we can see that the courseCode is quite an useful attribute when going through the Courses relation. Thus, we will make an index out of the courseCode attribute.

```
01 |  CREATE INDEX idx_courses ON Courses (
02 |  courseCode
03 |  );
```

An efficient way to access the relation CourseInstances would be to use year and semester as indexes. This would group tuples that are usually accessed together.

```
01 |  CREATE INDEX idx_course_instances ON CourseInstances (
02 |  year,
03 |  semester
04 |  );
```

There are some relations where the number of reads and writes may be lower, thus, creating an index for those (e.g. Buildings) may be counterproductive. There are other relations such as EnrollForCourses and EnrollForExams that will be accessed many times, therefore, they should use indexes. In both cases, using studentID as the key is what seems the most efficient.

```
01 |  CREATE INDEX idx_enroll_exams ON EnrollForExams (
02 |  studentID
03 |  );
04 |
05 |  CREATE INDEX idx_enroll_courses ON EnrollForCourses (
06 |  studentID
07 |  );
```

In both Events and Reserve indexing via the attribute eventNo will give the best efficiency.

```
01 |  CREATE INDEX idx_events ON Events (
02 |  eventNo
03 |  );
04 |  CREATE INDEX idx_reserve ON Reserve (
05 |  eventNo
06 |  );
```

7.2 Views

Based on the typical queries that we may expect, we can create views that will help simplify these queries and reduce their complexity.

```
CREATE VIEW LectureTimePlace AS
    SELECT instanceNo,
           lectureNo,
           date,
           startTime,
           endTime,
           buildingID,
           roomID
      FROM (
               SELECT instanceNo,
                      lectureNo,
                      Lectures.eventNo,
                      date,
                      startTime,
                      endTime
                 FROM Lectures
                      LEFT JOIN
                      Events ON Lectures.eventNo = Events.eventNo
           )
           AS LectureTime
           LEFT JOIN
           Reserve ON LectureTime.eventNo = Reserve.eventNo;

CREATE VIEW MeetingTimePlace AS
    SELECT groupNo,
           meetingNo,
           date,
           startTime,
           endTime,
           buildingID,
           roomID
      FROM (
               SELECT groupNo,
                      meetingNo,
                      GroupMeetings.eventNo,
                      date,
                      startTime,
                      endTime
                 FROM GroupMeetings
                      LEFT JOIN
                      Events ON GroupMeetings.eventNo = Events.eventNo
           )
           AS MeetingTime
           LEFT JOIN
           Reserve ON MeetingTime.eventNo = Reserve.eventNo;

CREATE VIEW ExamTimePlace AS
    SELECT courseCode,
           examNo,
           startExamTime,
           endExamTime,
           date,
           startTime,
           endTime,
           buildingID,
           roomID
      FROM (
               SELECT courseCode,
                      examNo,
                      Exams.eventNo,
                      startExamTime,
```

```
                        endExamTime,
                        date,
                        startTime,
                        endTime
                    FROM Exams
                        LEFT JOIN
                        Events ON Exams.eventNo = Events.eventNo
            )
            AS ExamTime
            LEFT JOIN
            Reserve ON ExamTime.eventNo = Reserve.eventNo;
```

8 Use cases and SQL-queries

Here, we can see some intersting use cases for the database. Other typical uses of the database concerning imputation and deletion of tuples in different relations can be observed in Section 6.

```
/* 1. Query all distinct courses (course code and name) taken place during the first
     semester of 2020 */
SELECT DISTINCT courseCode, courseName
  FROM CourseInstances AS CI,
       Courses AS C
 WHERE CI.courseCode = C.courseCode AND
       year = 2020 AND
       semester = 1;
```

	courseCode	courseName
1	CS-A1150	Databases
2	CS-E4800	Artificial Intelligence
3	CS-E4610	Modern Database Systems

Figure 2: Output of query 1

```
/* 2. Query all courses that contain the word 'Database' in their name */
SELECT courseName
  FROM Courses
 WHERE courseName LIKE '%Database%';
```

	courseName
1	Databases
2	Modern Database Systems

Figure 3: Output of query 2

```

/* 3. Find how many courses take place per semester and year*/
SELECT year,
       semester,
       COUNT(instanceNo) AS total
  FROM CourseInstances
 GROUP BY year,
          semester;
```

```
/* 4. Find how many instances of each course are scheduled.
Those courses with no instances scheduled must also be listed.
Order the courses based on the number of instances in a descending fashion. */
SELECT courseName as Course,
```

	year	semester	total
1	2020	1	3
2	2020	2	1

Figure 4: Output of query 3

```
05 |         COUNT(instanceNo) AS Instances
06 |   FROM Courses
07 |         LEFT OUTER JOIN
08 |         CourseInstances ON CourseInstances.courseCode = Courses.courseCode
09 |  GROUP BY courseName
10 |  ORDER BY instances DESC;
```

	Course	Instances
1	Modern Database Systems	2
2	Artificial Intelligence	1
3	Databases	1
4	Computer Graphics	0

Figure 5: Output of query 4

```
01 | /* 5. Query the exact number and the limit number of students in one group */
02 | SELECT EnrollForCourses.groupNo,
03 |         COUNT(studentID),
04 |         studentLimit
05 |   FROM EnrollForCourses,
06 |         ExerciseGroups
07 |  WHERE EnrollForCourses.groupNo = ExerciseGroups.groupNo
08 |  GROUP BY EnrollForCourses.groupNo;
```

	groupNo	COUNT(studentID)	studentLimit
1	CS-A1150-group1	1	10

Figure 6: Output of query 5

```
01 | /* 6. Query the exact number of students in one exam */
02 | SELECT examNo,
03 |         COUNT(studentID)
04 |   FROM EnrollForExams
05 |  GROUP BY examNo;
```

	examNo	COUNT(studentID)
1	CS-A1150-exam1	2

Figure 7: Output of query 6

```
01 | /* 7. Query the exact number of students in a study group */
02 | SELECT groupNo,
03 |         COUNT(studentID)
04 |   FROM EnrollForCourses
05 |  GROUP BY groupNo;
```

```
01 | /* 8. Find the course codes and names of all the course that are organized during
       the second semester,
02 |  but not in the second semester. */
03 | SELECT courseCode,
04 |         courseName
```

	groupNo	COUNT(studentID)
1	CS-A1150-group1	1

Figure 8: Output of query 7

```
  FROM Courses
 WHERE courseCode IN (
           SELECT courseCode
             FROM courseInstances
            WHERE semester = 1
       )
AND
       courseCode NOT IN (
           SELECT courseCode
             FROM courseInstances
            WHERE semester = 2
       );
```

	courseCode	courseName
1	CS-A1150	Databases
2	CS-E4800	Artificial Intelligence

Figure 9: Output of query 8

```
/* 9. Query the number of computers in one room which could used by students to have
     a lecture or exam */
SELECT buildingID,
       roomID,
       COUNT(Computers.equipmentID)
  FROM Computers,
       Equipments
 WHERE users = 'students' AND
       Computers.equipmentID = Equipments.equipmentID
 GROUP BY buildingID,
          roomID;
```

buildingID	roomID	COUNT(Computers.equipmentID)

Figure 10: Output of query 9

```
/* 10. Find a room which has at least 20 seats and which is free for reservation at
    a certain time (2020-09-03 at 9:00 to 10:00).
List the room ID and the address of the room where the building is located.
*/

/* In order to select those rooms without a reservation, 3 different conditions can
    occur:
    - The room has no reservations for that day
    - The room has a reservation in that day, but not within the specified times
    - The room has no reservations at all (it is not found in the Reserve relation).
   Besides this, we just have to make sure that the room has space for more than 20
    people.
In this first query, one of the rooms has an event in that day and within the given
    timeframe.
The others are available.
*/

SELECT roomID,
```

```
15 |         buildingName,
16 |         address
17 |    FROM Rooms,
18 |         Buildings
19 |   WHERE Rooms.buildingID = Buildings.buildingID AND
20 |         size >= 20
21 | EXCEPT
22 | SELECT Reserve.roomID,
23 |         buildingName,
24 |         address
25 |    FROM Reserve,
26 |         Events,
27 |         Rooms,
28 |          Buildings
29 |   WHERE Rooms.buildingID = Buildings.buildingID AND
30 |         Reserve.eventNo = Events.eventNo AND
31 |         Reserve.roomID = Rooms.roomID AND
32 |         date = '2020-09-03' AND
33 |         ( (TIME('09:00') >= startTime AND
34 |            TIME('09:00') < endTime) OR
35 |           (TIME('10:00') > startTime AND
36 |            TIME('10:00') <= endTime) OR
37 |           (TIME('09:00') < startTime AND
38 |            TIME('10:00') > endTime) );
```

	roomID	buildingName	address
1	C206	Computer Science Building	Tietotekniikantalo, Konemiehentie 2, 02150 Espoo
2	C310	Engineering Physics Building	Tietotekniikantalo, Konemiehentie 4, 02150 Espoo

Figure 11: Output of query 10

```
01 | /* 11. In this second query, there are no reservations for any Event on 2020-09-05,
       so all the rooms that
02 | have been inserted in the database are present. No matter if they have any scheduled
       event at some point or not.*/
03 | SELECT roomID,
04 |         buildingName,
05 |         address
06 |    FROM Rooms,
07 |         Buildings
08 |   WHERE Rooms.buildingID = Buildings.buildingID AND
09 |         size >= 20
10 | EXCEPT
11 | SELECT Reserve.roomID,
12 |         buildingName,
13 |         address
14 |    FROM Reserve,
15 |         Events,
16 |         Rooms,
17 |          Buildings
18 |   WHERE Rooms.buildingID = Buildings.buildingID AND
19 |         Reserve.eventNo = Events.eventNo AND
20 |         Reserve.roomID = Rooms.roomID AND
21 |         date = '2020-09-05' AND
22 |         ( (TIME('09:00') >= startTime AND
23 |            TIME('09:00') < endTime) OR
24 |           (TIME('10:00') > startTime AND
25 |            TIME('10:00') <= endTime) OR
26 |           (TIME('09:00') < startTime AND
27 |            TIME('10:00') > endTime) );
```

```
01 | /* 12. Find out for which purpose a certain room is reserved at a certain time.
02 | For example, is there some event in room C206 from the Computer Science Building
03 | on 2020-11-25 between 08:30 and 10:30?*/
```

	roomID	buildingName	address
1	C206	Computer Science Building	Tietotekniikantalo, Konemiehentie 2, 02150 Espoo
2	C310	Engineering Physics Building	Tietotekniikantalo, Konemiehentie 4, 02150 Espoo
3	room101	Computer Science Building	Tietotekniikantalo, Konemiehentie 2, 02150 Espoo

Figure 12: Output of query 11

```
04 |
05 | /* The query should return empty if no events are scheduled, or it will return the
       EventNo
06 | that identifies which kind of event is scheduled and its schedule.
07 | If there is an event that overlaps the time that is given, it will be shown as well.
08 | No matter how small the time overlap is.*/
09 | SELECT DISTINCT eventNo,
10 |                 startTime,
11 |                 endTime
12 |   FROM Events,
13 |        Reserve,
14 |        Rooms,
15 |        Buildings
16 |  WHERE Events.eventNo = Reserve.eventNo AND
17 |        Reserve.roomID = Rooms.roomID AND
18 |        Rooms.buildingID = Buildings.buildingID AND
19 |        Rooms.roomID = 'C206' AND
20 |        Buildings.buildingName = 'Computer Science Building' AND
21 |        date = '2020-11-25' AND
22 |        (
23 |            (TIME(startTime) <= TIME('08:30') AND TIME(endTime) > TIME('08:30') )
24 |            OR
25 |            (TIME(startTime) > TIME('08:30') AND TIME(startTime) < TIME('10:30') )
26 |         );
```

	eventNo	startTime	endTime
1	20201125E1	08:30	09:30

Figure 13: Output of query 12

```
01 | /* 13. Now, we will check another room (e.g. room101) of the same building for the
       same time schedule.
02 | Since the previous one had an Event (Exam because of the code).
03 | */
04 | SELECT DISTINCT eventNo,
05 |                 startTime,
06 |                 endTime
07 |   FROM Events,
08 |        Reserve,
09 |        Rooms,
10 |        Buildings
11 |  WHERE Events.eventNo = Reserve.eventNo AND
12 |        Reserve.roomID = Rooms.roomID AND
13 |        Rooms.buildingID = Buildings.buildingID AND
14 |        Rooms.roomID = 'room101' AND
15 |        Buildings.buildingName = 'Computer Science Building' AND
16 |        date = '2020-11-25' AND
17 |        (
18 |            (TIME(startTime) <= TIME('08:30') AND TIME(endTime) > TIME('08:30') )
19 |            OR
20 |            (TIME(startTime) > TIME('08:30') AND TIME(startTime) < TIME('10:30') )
21 |         );
```

```
01 | /* As we can see, no tuples are returned and, thus, we can book this room for this
       time schedule freely.*/
```

eventNo	startTime	endTime

Figure 14: Output of query 13

```
02 |
03 |
04 | /* 14. List all exams that are scheduled, the course name and where and when it is
       scheduled to take place.
05 | Now, we can use the view that we created earlier 'ExamTimePlace' to make the query
       much simpler.*/
06 | SELECT examNo,
07 |        courseName,
08 |        date,
09 |        startExamTime AS start,
10 |        endExamTime AS end,
11 |        roomId,
12 |        buildingName
13 |   FROM ExamTimePlace,
14 |        Courses,
15 |        Buildings
16 |  WHERE ExamTimePlace.courseCode = Courses.courseCode AND
17 |        ExamTimePlace.buildingID = Buildings.buildingID;
```

	examNo	courseName	date	start	end	roomId	buildingName
1	CS-A1150-exam1	Databases	2020-11-25	08:40	09:30	C206	Computer Science Building

Figure 15: Output of query 14

```
01 | /* 15. List all students who have enrolled for a exam in the Databases course.
02 | List the student name, the program they are in, which exam they are enrolled in (
       multiple exams
03 | per year in a course are possible) and the date of the exam.*/
04 | SELECT studentName,
05 |        program,
06 |        examNo,
07 |        date
08 |   FROM EnrollForExams,
09 |        Exams,
10 |        Courses,
11 |        Students,
12 |        Events
13 |  WHERE EnrollForExams.studentID = Students.studentID AND
14 |        EnrollForExams.examNo = Exams.examNo AND
15 |        Exams.courseCode = Courses.courseCode AND
16 |        Exams.eventNo = Events.eventNo AND
17 |        Courses.courseName = 'Databases';
```

	studentName	program	examNo	date
1	Teemu Teekkari	TIK	CS-A1150-exam1	2020-11-25
2	Maija Virtanen	AUT	CS-A1150-exam1	2020-11-25

Figure 16: Output of query 15

```
01 | /* 16. For a particular course instance (e.g. 'CS-A1150-2020-1'), list the exercise
       groups that are not full yet.
02 | List how many students are enrolled in those groups that are not full.*/
03 | SELECT EnrollForCourses.groupNo,
04 |        COUNT(studentID) AS totalStudents
```

```
  FROM EnrollForCourses,
       ExerciseGroups
 WHERE EnrollForCourses.groupNo = ExerciseGroups.groupNo
 GROUP BY EnrollForCourses.groupNo
HAVING COUNT(studentID) < ExerciseGroups.studentLimit;
```

	groupNo	totalStudents
1	CS-A1150-group1	1

Figure 17: Output of query 16

```
/* 17. Find out, when a certain course has been arranged or when it will be arranged
    */
SELECT courseName,
       year,
       semester,
       startDate,
       endDate
  FROM CourseInstances,
       Courses
 WHERE CourseInstances.courseCode = Courses.courseCode
 ORDER BY startDate, endDate;

```

	courseName	year	semester	startDate	endDate
1	Artificial Intelligence	2020	1	2020-09-01	2020-12-27
2	Databases	2020	1	2020-09-03	2020-11-20
3	Modern Database Systems	2020	2	2021-01-05	2021-04-05
4	Modern Database Systems	2020	1	2021-09-07	2021-11-23

Figure 18: Output of query 17

```
/* As we can see the course Modern Database Systems is organized twice in 2020.
The course instances that are listed are ordered by their start date (starting the
    earliest).
If they would have the same start date, the order would be determined by the end
    date (earliest first). */

/* 18. Delete student given a studentID ('112233') from course registration in
    course with course code CS-A1150.
After deletion, the student is no longer registered in the exercise group and, thus,
    not in the course either.*/
DELETE FROM EnrollForCourses
      WHERE studentID = '112233' AND
            groupNo LIKE 'CS-A1150%';

/*19. In this case, the teacher plans to organise a course instance.
He need query whether there is already an course instance for the Computer Graphics
    recently.
If there is schedule, the teacher wants to update the startDate and endDate.
If there is no schedule, the teacher wants to insert one.*/
SELECT Courses.courseCode,
       instanceNo,
       credits,
       year,
       semester,
       startDate,
       endDate
  FROM Courses
       LEFT OUTER JOIN
       CourseInstances ON Courses.courseCode = CourseInstances.courseCode
 WHERE courseName = 'Computer Graphics';
```

```
26 |  -- Now the teacher finds theres is no such course instance, so he wants to insert
       one.
27 | INSERT INTO CourseInstances VALUES (
28 |                                 'CS-C3100-2020-1',
29 |                                 2020,
30 |                                 1,
31 |                                 '2021-01-05',
32 |                                 '2020-04-20',
33 |                                 'CS-C3100'
34 |                               );
35 | -- Later the teacher wants to change the start date, he wants to update it.
36 | UPDATE CourseInstances
37 |    SET startDate = '2021-01-02'
38 |  WHERE instanceNo = 'CS-C3100-2020-1';
39 |
40 | -- When we execute the query now, we obtain the following
41 | SELECT Courses.courseCode,
42 |        instanceNo,
43 |        credits,
44 |        year,
45 |        semester,
46 |        startDate,
47 |        endDate
48 |   FROM Courses
49 |        LEFT OUTER JOIN
50 |        CourseInstances ON Courses.courseCode = CourseInstances.courseCode
51 |  WHERE courseName = 'Computer Graphics';
```

	courseCo	instanceNo	credits	year	semester	startDate	endDate
1	CS-C3100	CS-C3100-2020-1	5	2020	1	2021-01-02	2020-04-20

Figure 19: Output of query 19