

DATABASES PROJECT

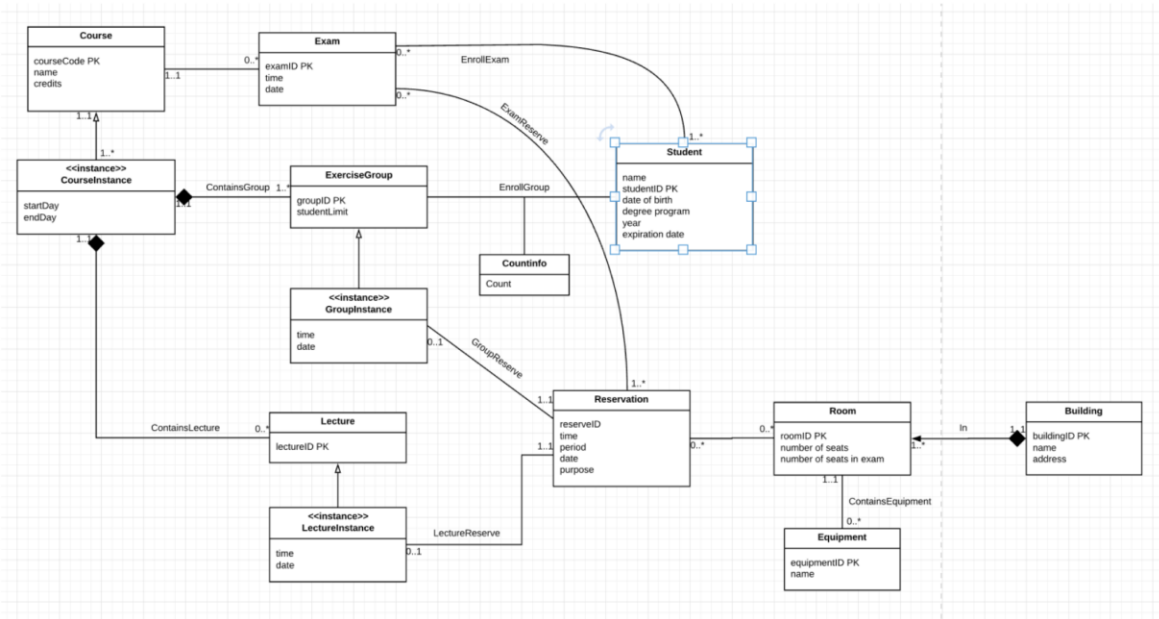
Binh Pham (788788, binh.pham@aalto.fi)

Khue Nguyen (789143, khue.nguyen@aalto.fi)

Long Nguyen (783903, long.l.nguyen@aalto.fi)

PART 1

★ UML:



★ Relations:

Course(courseCode, name, credits)

CourseInstance(courseInsID, courseID, startDay, endDay)

Exam(examID, courseID, startTime, endTime, date)

ExerciseGroup(groupID, courseInsID, studentLimit, numberOfStudents)

GroupInstance(groupInsID, groupID, startTime, endTime, date)

Lecture(lectureID, courseInsID)

LectureInstance(lectureInsID, lectureID, startTime, endTime, date)

Student(studentID, name, date of birth, degree program, year, expiration date)

Room(roomID, buildingID, number of seats, number of seats in exam)

Building(buildingID, name, address)

Equipment(equipmentID, roomID, name)

Reservation(reserveID, instanceID, roomID, startTime, endTime, date)

EnrollExam(studentID, examID)

EnrollGroup(studentID, groupID)

★ **Functional Dependencies:**

Relation Course: courseCode \rightarrow name credits

Relation Exam: examID \rightarrow time date

Relation ExerciseGroup: groupID \rightarrow studentLimit

Relation Room: roomID \rightarrow numberOfSeats numberOfSeatsInExam

Relation Building: buildingID \rightarrow name address

Relation Equipment: equipmentID \rightarrow name

Relation Reservation: reserveID \rightarrow buildingID roomID time period date purpose

roomID \rightarrow buildingID

Relation EnrollGroup: groupID \rightarrow count

Relation ContainsGroup: groupID \rightarrow courseCode startDate endDate

Relation ContainsLecture: lectureID \rightarrow courseCode startDate endDate

★ **Explanation:**

- ★ We represent Courses by class Course. The attributes of each Course are courseCode, name, and the number of credits (credits). The primary key of the relation Course is the courseCode, which separates one course from another as we assume each course has a unique code. Each course can have several instances, which are organized several times either in the same semester or in different semesters.
- We represent Course's instances by the class CourseInstance. CourseInstance is presented as a subclass of the superclass Course, as it has all the attributes that Course has, plus other attributes to specify an instance of a course. The attributes of this class are courseCode, name and credit inheriting from its superclass Course, startDate (the date the course instance starts), and endDate (the date the course instance ends).
- A course may have lectures, exercise groups, and exams.

- We represent the Lectures by the Lecture relation, which has the attribute lectureID. The primary key is lectureID. Similar to Course and CourseInstance, the same lecture may be organized several times. Therefore, we create the LectureInstance relation, which has all the attributes inherited from its superclass Lecture and two other attributes: time and date.
 - We represent the Exercise Groups by the ExerciseGroup relation, which has the attributes groupID and studentLimit (An exercise group may have a limit for the number of students). The primary key is groupID. As the exercise groups might meet for several times, we inherit from the superclass ExerciseGroup the subclass ExerciseInstance, which has two additional attributes date and time
 - We represent the Exams by the Exam relation, which has ExamID as its primary key and two other attributes: date and time.
- As Lectures and Exercise Groups belong to a Course instance, we represent the relationship between the classes Lecture, ExerciseGroup, and CourseInstance as compositions (denoting by a line with a black diamond adjacent to the container class - CourseInstance). The reasoning for this is that if a course instance is canceled, its lectures and exercise groups will cease to exist as well.
 - Apply the same logic for the requirement that Exams belong directly to the Course, there is a composition relationship between 2 classes Exam and Course, with the black diamond notation adjacent to class Course.
 -
 - The student enrolls for a course by enrolling for one of its exercise groups. We represent this by the relation EnrollGroup. We add the class Countinfo to count the number of students that have enrolled in the exercise group so that we can ensure that the number of students will not exceed the limit. Therefore, students cannot enroll in the exercise group if the attribute count of Countinfo is equal to the limit
 - The students must enroll for exams if they want to take it. Therefore, we create relation EnrollExam to represent this situation.
 - We represent Students by relation Student. The attributes of this relation are name, studentID, date – of – birth, degree – program, year and expiration – date. The primary key of this relation is studentID because every studentID is unique (no two identical studentID).
 - The university has several buildings, which contain rooms for different purposes (exams, teaching, exercises,...). We represent buildings by relation Building. Relation Building has three attributes: buildingID, name and address. For this relation, the buildingID is the primary key.

- We represent rooms by relation Room. Relation Room also has three attributes: roomID, number – of – seats and number – of – seats – in – exam. For this relation, roomID is the primary key. The information about each room also contains the equipment in the room. As the system must be able to store information also about different type of equipment, we create the relation Equipment, which help keep track with all the equipments in the room. The attributes of this relation are: equipmentID and name.
 - The relation from class Room to class Building is a composition as well, as all rooms must belong to some buildings, and if a building is destroyed, all of its rooms will disappear as well. The relation from class Building to class Room is a directed association, denoting by black triangle adjacent to the class Room. The directed association depicts a container-contained directional flow, or in other words, a building contains of several rooms.
-
- The database must contain information on the reservations of the rooms. We represent the reservations by relation Reservation. The attributes of this relation are reservationID (as primary key), roomID, buildingID, time, period (the amount of time the room is booked), date and purpose (exams, teaching, exercises,...).
 - We represent the reservation for Examination by the relation ExamReserve.
 - We represent the reservation for Exercise Group by the relation GroupReserve.
 - We represent the reservation for Lecture by the relation LectureReserve.
-
- Search for the course which are arranged in a certain time interval.
 - In relation CourseInstance, we have attributes startDate and endDate, which allow us to search for all the courses that are arranged in a certain time interval.
-
- Find out which exams a certain course has during a certain time interval.
 - In relation Exam, we have the attribute date for each exam. Therefore, we can find out which exams happened during a certain time interval.
 - The classes have been constructed so that they are ready to store all required information. All we need to do is inserting that information into the database. We also create the class Reservation to specially store the history of the reservations
 - Find out when a certain course has been arranged or when it will be arranged.
 - If we want to find out when a certain course has been or will be arranged, we can check the course instances of that course. In relation CourseInstance, we can find out when that course has been or will be arranged by checking the attribute startDate.
-
- Find the lectures belonging to a certain course instance.
 - We have the relation ContainsLecture. Therefore, we can find all the lectures belonging to a certain course instance.
-
- Find the exercise groups belonging to a certain course instance and find out, when and where a certain exercise group meets.

- We have the relation ContainsGroup. Therefore, we can find all the groups belonging to a certain course instance. If we want to find out when a certain exercise group meet, we can find all of it GroupInstance and then check their time and date attributes. If we want to find out where a certain exercise group meet, we should also find all of it GroupInstance and then check all the Reservation that include those group instances. In relation Reservation, we can find the room and the building where the exercise group meets
- Find a room which has at least a certain number of seats and which is free for reservation at a certain time.
- In relation Room, there is a number – of – seats attribute, which help us find a room which has at least a certain number of seats.
- In relation Reservation, we have attributes roomID and building ID. The roomID that exist in Reservation is the rooms that have been reserved. Therefore, if we want to check which room is free for reservation at a certain time, we have to find that time in Reservation, then find the roomIDs that are in relation Room but not yet in relation Reservation.
- Enroll a certain student to a certain exam or exercise group.
- We have the relation EnrollExam and EnrollGroup to enroll a certain student to a certain exam or exercise group.
- List all students who have enrolled for a certain course instance, exercise group or exam.
- We have the relations EnrollExam and EnrollGroup so that we can list all students who have enrolled for a certain exercise group or exam. We also have the relation ContainsExam and ContainsGroup so that we can trace the exam and exercise group back to the Course Instance that they belong. Therefore, we can list all students who have enrolled for a certain course instance.
- Find out which exercise groups at a certain course instance are not full yet.
- From a certain course instance, we can find out all the exercise groups belonging to the course. In order to check whether an exercise group is full or not, we have to check whether the Count attribute is still less than the studentLimit attribute of that exercise group. From that, we can sort out all the groups that are not full yet.

★ **Anomalies:**

There may be Redundancy and Update Anomaly when it comes to relation EnrollGroup(studentID, groupID, Count).

Firstly, the information about Count will be repeated many times as there are several students enroll in one group, which will cause Redundancy.

Secondly, when we want to update information Count, we will have to update it several times, which may cause Update Anomaly.

★ **BCNF:**

Relation EnrollGroup(studentID, groupID, Count)

✱ FDs: $\text{groupID} \rightarrow \text{count}$

Calculate the closure of the left side: $\{\text{groupID}\}^+ = \{\text{groupID}, \text{count}\}$

Decompose EnrollGroup into $E1(\text{groupID}, \text{count})$ and $E2(\text{groupID}, \text{studentID})$

Now we check whether the new relations are in BCNF:

✓ $E1$ has only one non – trivial functional dependency $\text{groupID} \rightarrow \text{count}$. Calculate the closure of the left side: $\{\text{groupID}\}^+ = \{\text{groupID}, \text{count}\}$. The closure contains all attributes of $E1$.

✓ **$E1$ is in BCNF**

✓ $E2$ has no functional dependency. Thus, $E2$ is also in BCNF.

Therefore, we have two new relations: $E1(\text{groupID}, \text{count})$ and $E2(\text{groupID}, \text{studentID})$

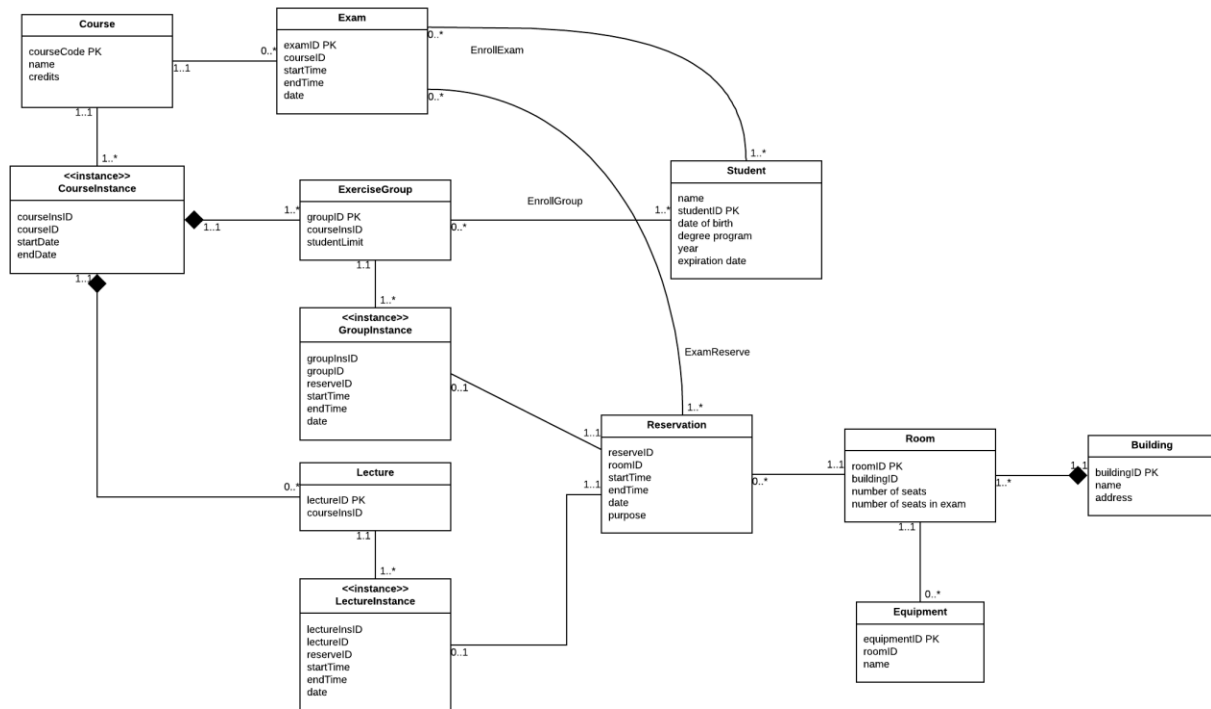
★ ***Decomposed relations***

$E1(\text{groupID}, \text{count})$

$E2(\text{groupID}, \text{studentID})$

PART 2

☺ ***CHANGES MADE AFTER THE SUBMISSION OF THE FIRST PART:***



- ✓ Class CourseInstance, GroupInstance and LectureInstance are not subclasses of Course, ExerciseGroup and Lecture, respectively. Therefore, in the UML diagram, we fix the relationship between CourseInstance and Course, LectureInstance and Lecture, GroupInstance and Group.
- ✓ In part one, the association between students and exercise group has the attribute “Count”. As this probably would not work as intended, in this part, we eliminate the “Count” attribute and decide to count tuples in order to keep track with the number of students in each exercise group.
- ✓ In part one, the association between reservation and room is many – many. In this part, we change the association to one – many.
- ✓ In relation Reservation in part one, we have only one attribute “time” to keep track of time. As it will cause difficulties with queries, we decide to use startTime and endTime instead.
- ✓ In part one, we missed the multiplicity between Exercise Group and Student. Therefore, we add that to this part as a supplement.

➤ **Relations:**

Course(courseCode, name, credits)

CourseInstance(courseInsID, courseID, startDate, endDate)

- We add attribute courseInsID to relation CourseInstance because just courseID, startDate and endDate is not enough as a key for CourseInstance

Exam(examID, courseID, startTime, endTime, date)

- We add attribute courseID to relation CourseInstance because the association between Exam and Course is one – many. Therefore, there is no need to create a separated relation for them.

ExerciseGroup(groupID, courseInsID, studentLimit)

- We eliminate the attribute Count in this relation.

GroupInstance(groupInsID, groupID, reserveID, startTime, endTime, date)

- We add the attribute reserveID in this relation because the association between GroupInstance and Reservation is one – many. Therefore, there is no need to create a separated table for this relation.

Lecture(lectureID, courseInsID)

- We add attribute courseInsID to relation CourseInstance because the association between Lecture and CourseInstance is one – many. Therefore, there is no need to create a separated relation for them.

LectureInstance(lectureInsID, lectureID, reserveID, startTime, endTime, date)

- Similar to GroupInstance, we add the attribute reserveID in this relation because the association between LectureInstance and Reservation is one – many. Therefore, there is no need to create a separated table for this relation.

Student(studentID, name, dateOfBirth, degreeProgram, year, expirationDate)

Room(roomID, buildingID, numberOfSeats, numberOfSeatInExam)

- In relation Room, we add the attribute buildingID because the association between Room and Building is one – many. Therefore, there is no need to create a separated table for this relationship.

Building(buildingID, name, address)

Equipment(equipmentID, roomID, name)

- In relation Equipment, we add the attribute roomID because the association between Equipment and Room is one – many. Therefore, there is no need to create a separated table for this relationship.

Reservation(reserveID, roomID, startTime, endTime, date)

- As mentioned above, we add startTime and endTime attributes to the table instead of time only. We also eliminated buildingID attribute.

EnrollExam(studentID, examID)

EnrollGroup(studentID, groupID)

- We eliminate the attribute Count.

ExamReserve(examID, reserveID)

☺ **ANSWERS FOR THE TASKS OF PART II**

1. Define the relation schema in SQL. The schema must contain the same information as the project part 1. Write the CREATE statements to the document. Use data types that are reasonable. Justify your solutions. In addition, insert enough initial data into your tables such that you can test the SQL queries described below.
2. Check the keys and other constraints in your database. Verify that the primary key and foreign keys are well defined.

(part 1 and 2 are combined here as we don't want our explanation to be interrupted)

❑ **Course (courseCode, name, credits)**

```
CREATE TABLE Course (  
    courseCode TEXT,  
    name TEXT,  
    credits INTEGER,  
    PRIMARY KEY (courseCode)  
);
```

The format for the code of a course would be a combination of 2 letters and 4 numbers, connecting by a hyphen. Therefore, *courseCode*'s type should be TEXT. It's quite obvious that the attribute *name* has the type TEXT, and attribute *credits* has type INTEGER. Each course has a unique code, so it makes sense to set *courseCode* as the primary key.

```
INSERT INTO Course VALUES ('DB-1123', 'Databases', 5);
INSERT INTO Course VALUES ('KA-1406', 'Programming I', 5);
INSERT INTO Course VALUES ('LN-1308', 'Calculus', 5);
INSERT INTO Course VALUES ('LC-1111', 'Finnish 1A', 2);
INSERT INTO Course VALUES ('CS-1010', 'Machine Learning', 3);
```

```
1 CREATE TABLE Course (
2   courseCode TEXT,
3   name TEXT,
4   credits INTEGER,
5   PRIMARY KEY (courseCode)
6 );
7
8
9 INSERT INTO Course VALUES ('DB-1123', 'Databases', 5);
10 INSERT INTO Course VALUES ('KA-1406', 'Programming I', 5);
11 INSERT INTO Course VALUES ('LN-1308', 'Calculus', 5);
12 INSERT INTO Course VALUES ('LC-1111', 'Finnish 1A', 2);
13 INSERT INTO Course VALUES ('CS-1010', 'Machine Learning', 3);
```

	courseCode	name	credits
1	DB-1123	Databases	5
2	KA-1406	Programming I	5
3	LN-1308	Calculus	5
4	LC-1111	Finnish 1A	2
5	CS-1010	Machine Learning	3

❑ **CourseInstance** (courseInsID, courseID, startDate, endDate)

```

CREATE TABLE CourseInstance (
    courseInsID TEXT,
    courseID TEXT,
    startDate TEXT,
    endDate TEXT,
    PRIMARY KEY (courseInsID),
    FOREIGN KEY (courseID) REFERENCES Course(courseCode)
    ON UPDATE CASCADE
    ON DELETE CASCADE,
    CHECK (endDate > startDate)
);

```

The format for *courseInsID* is 2 letters, taken from the letters used in the *courseID*, plus one number from 0-9 (as we assume, based on reality, that no course has more than 10 instances). The type of *courseInsID* then should be TEXT then. *courseID* is the foreign key referring to the attribute *courseCode* in the relation Course, so its type is also TEXT. As SQLite doesn't support the types DATE and TIME, attributes *startDay* and *endDay* are TEXT. Since the end of the course should be after its beginning, we need to check that the value of *endDay* is larger than that of *startDay*. The same logic is applied to other relations having attributes representing starting and ending time. *CourseInsID* is unique for each course instance, so it's suitable to be the primary key. Since every course instance must belong to some courses, there need to be a foreign key *courseID* referring to Course's *courseCode*. If there are any changes to the course, or the course is deleted from the system, all of its instances should be changed/deleted as well. The commands ON UPDATE CASCADE and ON DELETE UPDATE CASCADE, therefore, are used to maintain the cascade policy.

```

INSERT INTO CourseInstance VALUES ('DB1', 'DB-1123', '2019-08-13',
'2020-01-11');

```

```

INSERT INTO CourseInstance VALUES ('CS3', 'CS-1010', '2020-04-05',
'2020-07-13');

```

```

INSERT INTO CourseInstance VALUES ('CS1', 'CS-1010', '2019-08-15',
'2019-10-17');

```

```

INSERT INTO CourseInstance VALUES ('LC6', 'LC-1111', '2020-04-20',
'2020-07-03');

```

```
INSERT INTO CourseInstance VALUES ('KA2', 'KA-1406', '2020-01-13',
'2020-04-22');
```

```
1 CREATE TABLE CourseInstance (
2   courseInsID TEXT,
3   courseID TEXT,
4   startDate TEXT,
5   endDate TEXT,
6   PRIMARY KEY (courseInsID),
7   FOREIGN KEY (courseID) REFERENCES Course(courseCode)
8   ON DELETE CASCADE,
9   CHECK (endDate > startDate)
10 );
11
12 INSERT INTO CourseInstance VALUES ('DB1', 'DB-1123', '2019-08-13', '2020-01-11');
13 INSERT INTO CourseInstance VALUES ('CS3', 'CS-1010', '2020-04-05', '2020-07-13');
14 INSERT INTO CourseInstance VALUES ('CS1', 'CS-1010', '2019-08-15', '2019-10-17');
15 INSERT INTO CourseInstance VALUES ('LC6', 'LC-1111', '2020-04-20', '2020-07-03');
16 INSERT INTO CourseInstance VALUES ('KA2', 'KA-1406', '2020-01-13', '2020-04-22');
```

	courseInsID	courseID	startDate	endDate
1	DB1	DB-1123	2019-08-13	2020-01-11
2	CS3	CS-1010	2020-04-05	2020-07-13
3	CS1	CS-1010	2019-08-15	2019-10-17
4	LC6	LC-1111	2020-04-20	2020-07-03
5	KA2	KA-1406	2020-01-13	2020-04-22

❑ Exam (examID, courseID, startTime, endTime, date)

```
CREATE TABLE Exam (
    examID TEXT,
    courseID TEXT,
    startTime TEXT,
    endTime TEXT,
    date TEXT,
    PRIMARY KEY (examID),
    FOREIGN KEY (courseID) REFERENCES Course(courseCode)
    ON UPDATE CASCADE
    ON DELETE CASCADE,
    CHECK (endTime > startTime)
```

);

The type of *examID* is TEXT, since it's in the format of a letter following by 4 numbers. Attribute *courseID* refers to the attributes *courseCode* in Course, so it has type TEXT as well. We use the same logic as mentioned above to set up the type for *startTime*, *endTime* and *date* as TEXT, and add the condition that *endTime* has to be bigger (later) than *startTime*. (We can compare like this as in reality, no exams happen during the transition of one day to another). A unique ID is created for each exam, so the attribute *examID* is enough to be the primary key. All exams must belong to some courses, so it's necessary to set a foreign key *courseID* referring to the code of the course. Also, whenever an update or deletion is made on the course code, this change needs to be updated in the relation Exam as well, so we need to use ON UPDATE CASCADE and ON DELETE CASCADE.

```
INSERT INTO Exam VALUES ('A1022', 'DB-1123', '12:00', '15:00', '2020-03-03');
```

```
INSERT INTO Exam VALUES ('C2010', 'CS-1010', '08:00', '11:30', '2020-02-06');
```

```
INSERT INTO Exam VALUES ('L2011', 'LC-1111', '14:15', '17:20', '2020-05-07');
```

```
INSERT INTO Exam VALUES ('L2222', 'LC-1111', '10:00', '13:15', '2019-12-18');
```

```
INSERT INTO Exam VALUES ('S1234', 'KA-1406', '11:45', '14:15', '2020-08-13');
```

```
1 CREATE TABLE Exam (  
2 examID TEXT,  
3 courseID TEXT,  
4 startTime TEXT,  
5 endTime TEXT,  
6 date TEXT,  
7 PRIMARY KEY (examID),  
8 FOREIGN KEY (courseID) REFERENCES Course(courseCode)  
9 ON DELETE CASCADE,  
10 CHECK (endTime > startTime)  
11 );  
12  
13 INSERT INTO Exam VALUES ('A1022', 'DB-1123', '12:00', '15:00', '2020-03-03');  
14 INSERT INTO Exam VALUES ('C2010', 'CS-1010', '08:00', '11:30', '2020-02-06');  
15 INSERT INTO Exam VALUES ('L2011', 'LC-1111', '14:15', '17:20', '2020-05-07');  
16 INSERT INTO Exam VALUES ('L2222', 'LC-1111', '10:00', '13:15', '2019-12-18');  
17 INSERT INTO Exam VALUES ('S1234', 'KA-1406', '11:45', '14:15', '2020-08-13');
```

	examID	courseID	startTime	endTime	date
1	A1022	DB-1123	12:00	15:00	2020-03-03
2	C2010	CS-1010	08:00	11:30	2020-02-06
3	L2011	LC-1111	14:15	17:20	2020-05-07
4	L2222	LC-1111	10:00	13:15	2019-12-18
5	S1234	KA-1406	11:45	14:15	2020-08-13

❑ **ExerciseGroup (groupID, courseInsID, studentLimit)**

```
CREATE TABLE ExerciseGroup (
    groupID TEXT,
    courseInsID TEXT,
    studentLimit INTEGER,
    PRIMARY KEY (groupID),
    FOREIGN KEY (courseInsID) REFERENCES
    CourseInstance(courseInsID)
    ON UPDATE CASCADE
    ON DELETE CASCADE
);
```

groupID has type TEXT as its format is a “G” followed by 4 numbers. *courseInsID* refers to the relation CourseInstance’s *courseInsID*, therefore having type TEXT. *studentLimit* is a number so it has type INTEGER. Each group has a unique code, so attribute *groupID* is suitable to be the primary key. Each exercise group is created in accordance to a course instance, so we need to have a foreign key *courseInsID* referring to the code of the course instance. This attribute also needs to follow the cascade policy, so we add ON UPDATE CASCADE and ON DELETE CASCADE here as well.

```
INSERT INTO ExerciseGroup VALUES ('G1111', 'DB1', 30);
INSERT INTO ExerciseGroup VALUES ('G1231', 'CS1', 50);
INSERT INTO ExerciseGroup VALUES ('G1234', 'LC6', 25);
```

```
INSERT INTO ExerciseGroup VALUES ('G1298', 'KA2', 30);
INSERT INTO ExerciseGroup VALUES ('G1235', 'LC6', 25);
```

```
1 CREATE TABLE ExerciseGroup (
2   groupID TEXT,
3   courseInsID TEXT,
4   studentLimit INTEGER,
5   PRIMARY KEY (groupID),
6   FOREIGN KEY (courseInsID) REFERENCES CourseInstance(courseInsID) ON DELETE CASCADE
7 );
8
9 INSERT INTO ExerciseGroup VALUES ('G1111', 'DB1', 30);
10 INSERT INTO ExerciseGroup VALUES ('G1231', 'CS1', 50);
11 INSERT INTO ExerciseGroup VALUES ('G1234', 'LC6', 25);
12 INSERT INTO ExerciseGroup VALUES ('G1298', 'KA2', 30);
13 INSERT INTO ExerciseGroup VALUES ('G1235', 'LC6', 25); |
```

	groupID	courseInsID	studentLimit
1	G1111	DB1	30
2	G1231	CS1	50
3	G1234	LC6	25
4	G1298	KA2	30
5	G1235	LC6	25

❑ **GroupInstance (groupInsID, groupID, reserveID, startTime, endTime, date)**

```
CREATE TABLE GroupInstance (
    groupInsID TEXT,
    groupID TEXT,
    reserveID TEXT,
    startTime TEXT,
    endTime TEXT,
    date TEXT,
    PRIMARY KEY (groupInsID),
    FOREIGN KEY (groupID) REFERENCES ExerciseGroup(groupID)
    ON UPDATE CASCADE
```



```

        ON DELETE CASCADE,

        FOREIGN KEY (reserveID) REFERENCES Reservation(reserveID)

        ON DELETE SET NULL

        ON UPDATE CASCADE,

        CHECK (endTime > startTime)

);

```

The attribute *groupInstID* is constructed by adding a letter after the group's ID. By doing this, we can easily distinguish which instances belong to the same group. It makes sense to have *groupInstID*'s type as TEXT. *groupID* and *reserveID* refer to the attribute with the same name in relations ExerciseGroup and Reservation, respectively, so their types are both TEXT. The type (TEXT) and condition for *startTime*, *endTime*, and *date* are set using the same logic as mentioned in the relation Exam. A unique code is generated everytime a group instance is created, so *groupInstID* is enough for the primary key. A group instance needs to belong to some groups and have some reservations for it to happen. Accordingly, we need to create the foreign keys *groupID*, referring to ExerciseGroup's *groupID*, and *reserveID*, referring to Reservation's *reserveID*. If we update or delete the group's ID in relation ExerciseGroup, the foreign key *groupID* should be changed/deleted as well. Consequently, ON UPDATE CASCADE and ON DELETE CASCADE are needed here. However, if a reservation for an exercise group instance is removed, we can still reserve another date and time for that instance. Therefore, we should use ON DELETE SET NULL and ON UPDATE CASCADE for the foreign key *reserveID*.

```

INSERT INTO GroupInstance VALUES ('G1111A', 'G1111', 'RS100475',
'12:15', '13:45', '2020-06-14');

INSERT INTO GroupInstance VALUES ('G1234D', 'G1234', 'RS112233',
'16:00', '18:00', '2019-09-16');

INSERT INTO GroupInstance VALUES ('G1234F', 'G1234', 'RS334455',
'09:00', '11:00', '2020-06-24');

INSERT INTO GroupInstance VALUES ('G1298A', 'G1298', 'RS556677',
'08:15', '10:15', '2020-06-16');

INSERT INTO GroupInstance VALUES ('G1234P', 'G1234', 'RS778899',
'17:15', '19:30', '2020-04-06');

```

```

1 CREATE TABLE GroupInstance (
2   groupInsID TEXT,
3   groupID TEXT,
4   reserveID TEXT,
5   startTime TEXT,
6   endTime TEXT,
7   date TEXT,
8   PRIMARY KEY (groupInsID),
9   FOREIGN KEY (groupID) REFERENCES ExerciseGroup(groupID)
10  ON DELETE CASCADE,
11  FOREIGN KEY (reserveID) REFERENCES Reservation(reserveID)
12  ON DELETE SET NULL
13  ON UPDATE CASCADE,
14  CHECK (endTime > startTime)
15 );
16
17 INSERT INTO GroupInstance VALUES ('G1111A', 'G1111', 'RS100475', '12:15', '13:45', '2020-06-14');
18 INSERT INTO GroupInstance VALUES ('G1234D', 'G1234', 'RS112233', '16:00', '18:00', '2019-09-16');
19 INSERT INTO GroupInstance VALUES ('G1234F', 'G1234', 'RS334455', '09:00', '11:00', '2020-06-24');
20 INSERT INTO GroupInstance VALUES ('G1298A', 'G1298', 'RS556677', '08:15', '10:15', '2020-06-16');
21 INSERT INTO GroupInstance VALUES ('G1234P', 'G1234', 'RS778899', '17:15', '19:30', '2020-04-06');

```

	groupInsID	groupID	reserveID	startTime	endTime	date
1	G1111A	G1111	RS100475	12:15	13:45	2020-06-14
2	G1234D	G1234	RS112233	16:00	18:00	2019-09-16
3	G1234F	G1234	RS334455	09:00	11:00	2020-06-24
4	G1298A	G1298	RS556677	08:15	10:15	2020-06-16
5	G1234P	G1234	RS778899	17:15	19:30	2020-04-06

□ Lecture (lectureID, courseInsID)

```

CREATE TABLE Lecture (
    lectureID TEXT,
    courseInsID TEXT,
    PRIMARY KEY (lectureID),
    FOREIGN KEY (courseInsID) REFERENCES
    CourseInstance(courseInsID)
    ON UPDATE CASCADE
    ON DELETE CASCADE
);

```

The type for both *lectureID* and *courseInsID* is TEXT, as *lectureID*'s format is the letter "L" followed by 4 numbers and *courseInsID* refers to the attribute with the same name in CourseInstance.

A unique code is generated for each lecture, so *lectureID* is the primary key for this relation. All lectures have to belong to some course instances, so it's necessary to have the foreign key

courseInsID referring to CourseInstance's *courseInsID*. If the course instance is updated or removed, the same action needs to be mimicked at the foreign key, according to cascade policy.

```
INSERT INTO Lecture VALUES ('L2366', 'CS3');
INSERT INTO Lecture VALUES ('L3784', 'LC6');
INSERT INTO Lecture VALUES ('L4836', 'CS3');
INSERT INTO Lecture VALUES ('L2222', 'CS1');
INSERT INTO Lecture VALUES ('L7636', 'DB1');
```

```
1 CREATE TABLE Lecture (
2   lectureID TEXT,
3   courseInsID TEXT,
4   PRIMARY KEY (lectureID),
5   FOREIGN KEY (courseInsID) REFERENCES CourseInstance(courseInsID) ON DELETE CASCADE
6 );
7
8 INSERT INTO Lecture VALUES ('L2366', 'CS3');
9 INSERT INTO Lecture VALUES ('L3784', 'LC6');
10 INSERT INTO Lecture VALUES ('L4836', 'CS3');
11 INSERT INTO Lecture VALUES ('L2222', 'CS1');
12 INSERT INTO Lecture VALUES ('L7636', 'DB1');
```

	lectureID	courseInsID
1	L2366	CS3
2	L3784	LC6
3	L4836	CS3
4	L2222	CS1
5	L7636	DB1

❑ **LectureInstance (lectureInsID, lectureID, reserveID, startTime, endTime, date)**

```
CREATE TABLE LectureInstance (
    lectureInsID TEXT,
    lectureID TEXT,
```

```

        reserveID TEXT,
        startTime TEXT,
        endTime TEXT,
        date TEXT,
        PRIMARY KEY (lectureInsID),
        FOREIGN KEY (lectureID) REFERENCES Lecture(lectureID)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
        FOREIGN KEY (reserveID) REFERENCES Reservation(reserveID)
        ON DELETE SET NULL
        ON UPDATE CASCADE,
        CHECK (endTime > startTime)
    );

```

The same logic when constructing relation GroupInstance is used here: *lectureInsID* is *lectureID* followed by a letter, *lectureID* and *reserveID* refer to attributes in other relations (Lecture and Reservation). Therefore, all of their types, as well as the type of *startTime*, *endTime*, and *date*, are TEXT. From the ID of the lecture, a unique ID is constructed for each lecture instance, which is called *lectureInsID* and is the primary key of the relation. A lecture instance needs to belong to some lectures and has reservation, so foreign keys *lectureID* and *reserveID* are created. If the lecture ID is changed/deleted, the same change/deletion should happen in the foreign key as well. However, if the reservation for the lecture is removed, we can still reserve another time. Accordingly, we use ON UPDATE CASCADE, ON DELETE CASCADE and ON DELETE SET NULL like above.

```

INSERT INTO LectureInstance VALUES ('L2222A', 'L2222', 'RS090517',
    '13:00', '15:00', '2019-09-15');

INSERT INTO LectureInstance VALUES ('L2366B', 'L2366', 'RS231111',
    '17:00', '19:00', '2020-06-23');

INSERT INTO LectureInstance VALUES ('L2366D', 'L2366', 'RS121212',
    '14:00', '16:00', '2020-06-15');

INSERT INTO LectureInstance VALUES ('L3784E', 'L3784', 'RS756234',
    '14:20', '16:20', '2020-04-05');

INSERT INTO LectureInstance VALUES ('L7636A', 'L7636', 'RS067584',
    '13:30', '15:30', '2020-05-07');

```

```

1 CREATE TABLE LectureInstance (
2   lectureInsID TEXT,
3   lectureID TEXT,
4   reserveID TEXT,
5   startTime TEXT,
6   endTime TEXT,
7   date TEXT,
8   PRIMARY KEY (lectureInsID),
9   FOREIGN KEY (lectureID) REFERENCES Lecture(lectureID)
10  ON DELETE CASCADE,
11  FOREIGN KEY (reserveID) REFERENCES Reservation(reserveID)
12  ON DELETE SET NULL
13  ON UPDATE CASCADE,
14  CHECK (endTime > startTime)
15 );
16
17 INSERT INTO LectureInstance VALUES ('L2222A', 'L2222', 'RS090517', '13:00', '15:00', '2019-09-15');
18 INSERT INTO LectureInstance VALUES ('L2366B', 'L2366', 'RS231111', '17:00', '19:00', '2020-06-23');
19 INSERT INTO LectureInstance VALUES ('L2366D', 'L2366', 'RS121212', '14:00', '16:00', '2020-06-15');
20 INSERT INTO LectureInstance VALUES ('L3784E', 'L3784', 'RS756234', '14:20', '16:20', '2020-04-05');
21 INSERT INTO LectureInstance VALUES ('L7636A', 'L7636', 'RS067584', '13:30', '15:30', '2020-05-07');

```

	lectureIns	lectureID	reserveID	startTime	endTime	date
1	L2222A	L2222	RS090517	13:00	15:00	2019-09-15
2	L2366B	L2366	RS231111	17:00	19:00	2020-06-23
3	L2366D	L2366	RS121212	14:00	16:00	2020-06-15
4	L3784E	L3784	RS756234	14:20	16:20	2020-04-05
5	L7636A	L7636	RS067584	13:30	15:30	2020-05-07

□ Student (studentID, name, dateOfBirth, degreeProgram, year, expirationDate)

```

CREATE TABLE Student (
    studentID TEXT,
    name TEXT,
    dateOfBirth TEXT,
    degreeProgram TEXT CHECK (degreeProgram IN ('Bachelor',
'Master', 'PhD')),
    year INTEGER CHECK (year < 7),
    expirationDate TEXT,
    PRIMARY KEY (studentID)
);

```

The student's ID is usually a sequence of 6 numbers, but sometimes it can contain letters, for example for students in open university. Therefore, its type should be TEXT. It's obvious that

the type of *name*, *dateOfBirth*, *degreeProgram* and *expirationDate* should be TEXT, and that the attribute *year* should be INTEGER. Additionally, we add some conditions. The degree program can only be “Bachelor”, “Master”, or “PhD”, and a student only has at most 7 years to finish his or her degree. The student ID is unique and enough to distinguish students. Therefore, it’s the primary key of the relation Student.

```
INSERT INTO Student VALUES ('783903', 'Long', '1999-13-08',
'Bachelor', 2, '2026-08-30');

INSERT INTO Student VALUES ('788788', 'Binh', '2001-14-02',
'Bachelor', 2, '2026-08-30');

INSERT INTO Student VALUES ('678173', 'Matthew', '1988-09-09',
'Bachelor', 4, '2024-08-29');

INSERT INTO Student VALUES ('283740', 'Kris', '1992-14-06',
'Master' , 1, '2022-04-20');

INSERT INTO Student VALUES ('230984', 'Vera', '1997-11-01',
'Master' , 3, '2025-01-21');
```

```
1 CREATE TABLE Student (
2 studentID TEXT,
3 name TEXT,
4 dateOfBirth TEXT,
5 degreeProgram TEXT CHECK (degreeProgram IN ('Bachelor', 'Master', 'PhD')),
6 year INTEGER CHECK (year < 7),
7 expirationDate TEXT,
8 PRIMARY KEY (studentID)
9 );
10
11 INSERT INTO Student VALUES ('783903', 'Long', '1999-13-08', 'Bachelor', 2, '2026-08-30');
12 INSERT INTO Student VALUES ('788788', 'Binh', '2001-14-02', 'Bachelor', 2, '2026-08-30');
13 INSERT INTO Student VALUES ('678173', 'Matthew', '1988-09-09', 'Bachelor', 4, '2024-08-29');
14 INSERT INTO Student VALUES ('283740', 'Kris', '1992-14-06', 'Master' , 1, '2022-04-20');
15 INSERT INTO Student VALUES ('230984', 'Vera', '1997-11-01', 'Master' , 3, '2025-01-21');
```

	studentID	name	dateOfBirth	degreePr	year	expirationDat
1	783903	Long	1999-13-08	Bachelor	2	2026-08-30
2	788788	Binh	2001-14-02	Bachelor	2	2026-08-30
3	678173	Matthew	1988-09-09	Bachelor	4	2024-08-29
4	283740	Kris	1992-14-06	Master	1	2022-04-20
5	230984	Vera	1997-11-01	Master	3	2025-01-21

❑ Room (roomID, buildingID, numberOfSeats, numberOfSeatsInExam)

```

CREATE TABLE Room (
    roomID TEXT,
    buildingID TEXT,
    numberOfSeats INTEGER,
    numberOfSeatsInExam INTEGER,
    PRIMARY KEY (roomID),
    FOREIGN KEY (buildingID) REFERENCES Building(buildingID)
    ON UPDATE CASCADE
    ON DELETE CASCADE,
    CHECK (numberOfSeatsInExam < numberOfSeats)
);

```

The attribute *roomID* is formed with the letter “R” followed by 4 numbers, so its type should be TEXT. *buildingID* refers to the same-name attribute in relation Building, and it has type TEXT. Both *numberOfSeats* and *numberOfSeatsInExam* have type INTEGER. The condition that needs to be checked is that the number of seats in an exam is always smaller or equal to the number of seats in that room, as the students need to sit far away from each other during exams. Each room has a unique ID, so *roomID* is enough to be the primary key for Room. A room should belong to some buildings, so it’s necessary to have the foreign key *buildingID* referring to the attribute *buildingID* of Building. This foreign key has to follow cascade policy as well, since if a building is updated/removed, all rooms inside of it are updated/ceased to exist as well.

```

INSERT INTO Room VALUES ('R0002', 'B03', 50, 30);
INSERT INTO Room VALUES ('R7832', 'B11', 100, 50);
INSERT INTO Room VALUES ('R0005', 'B03', 30, 15);
INSERT INTO Room VALUES ('R1314', 'B01', 200, 150);
INSERT INTO Room VALUES ('R3334', 'B08', 20, 10);

```

```

1 CREATE TABLE Room (
2   roomID TEXT,
3   buildingID TEXT,
4   numberOfSeats INTEGER,
5   numberOfSeatsInExam INTEGER,
6   PRIMARY KEY (roomID),
7   FOREIGN KEY (buildingID) REFERENCES Building(buildingID)
8   ON DELETE CASCADE,
9   CHECK (numberOfSeatsInExam < numberOfSeats)
10
11 );
12
13 INSERT INTO Room VALUES ('R0002', 'B03', 50, 30);
14 INSERT INTO Room VALUES ('R7832', 'B11', 100, 50);
15 INSERT INTO Room VALUES ('R0005', 'B03', 30, 15);
16 INSERT INTO Room VALUES ('R1314', 'B01', 200, 150);
17 INSERT INTO Room VALUES ('R3334', 'B08', 20, 10);

```

	roomID	buildingID	numberOfSeats	numberOfSeatsInExam
1	R0002	B03	50	30
2	R7832	B11	100	50
3	R0005	B03	30	15
4	R1314	B01	200	150
5	R3334	B08	20	10

❑ Building (buildingID, name, address)

```

CREATE TABLE Building (
    buildingID TEXT,
    name TEXT,
    address TEXT,
    PRIMARY KEY (buildingID)
);

```

The attribute *buildingID* has type TEXT, and is constructed by the letter “B” followed by 2 numbers (as we assume the university has less than 100 buildings, so it’s more efficient using just 2 numbers). The *name* and *address* of the building should, of course, be TEXT. The ID of the building is unique enough to be the relation’s primary key.


```

INSERT INTO Building VALUES ('B01', 'Otakaari 1', '111 Otaniemi');

INSERT INTO Building VALUES ('B03', 'CS Building', 'Jamerantaival 1');

INSERT INTO Building VALUES ('B11', 'Alvari', '84 Otaniemi');

INSERT INTO Building VALUES ('B08', 'Library', '99 Central City');

INSERT INTO Building VALUES ('B02', 'Undergraduate Center', '123 Hakaniemi');

```

```

1 CREATE TABLE Building (
2   buildingID TEXT,
3   name TEXT,
4   address TEXT,
5   PRIMARY KEY (buildingID)
6 );
7
8 INSERT INTO Building VALUES ('B01', 'Otakaari 1', '111 Otaniemi');
9 INSERT INTO Building VALUES ('B03', 'CS Building', 'Jamerantaival 1');
10 INSERT INTO Building VALUES ('B11', 'Alvari', '84 Otaniemi');
11 INSERT INTO Building VALUES ('B08', 'Library', '99 Central City');
12 INSERT INTO Building VALUES ('B02', 'Undergraduate Center', '123 Hakaniemi');

```

	buildingID	name	address
1	B01	Otakaari 1	111 Otaniemi
2	B03	CS Building	Jamerantaival 1
3	B11	Alvari	84 Otaniemi
4	B08	Library	99 Central City
5	B02	Undergraduate Center	123 Hakaniemi

❑ Equipment (equipmentID, roomID, name)

```

CREATE TABLE Equipment (
    equipmentID TEXT,
    roomID TEXT,
    name TEXT,
    PRIMARY KEY (equipmentID),
    FOREIGN KEY (roomID) REFERENCES Room(roomID)

```

```
        ON DELETE SET NULL
        ON UPDATE CASCADE,
    );
```

equipmentID has type TEXT and is constructed by the letter “E” followed by 4 numbers. The name of the equipment has type TEXT. *roomID* also has type TEXT since it refers to the attribute roomID in Room. Each equipment has a unique ID when purchased, so *equipmentID* is enough to be Equipment’s primary key. We also create the foreign key *roomID* referring to Room’s *roomID* to show that the equipment needs to belong to some rooms. We use ON UPDATE CASCADE because if the ID of the room is updated, the foreign key should mimic this change as well. However, if the room is removed, we shouldn’t remove the equipment once belonged to the room, but rather set *roomID* as NULL as the equipment now is stored somewhere else for later use (for example the school’s storage).

```
INSERT INTO Equipment VALUES ('E1234', 'R1314', 'projector');
INSERT INTO Equipment VALUES ('E0003', 'R1314', 'speaker');
INSERT INTO Equipment VALUES ('E1308', 'R3334', 'projector');
INSERT INTO Equipment VALUES ('E0001', 'R0002', 'table');
INSERT INTO Equipment VALUES ('E8923', 'R0005', 'chair');
```

```
1 CREATE TABLE Equipment (
2   equipmentID TEXT,
3   roomID TEXT,
4   name TEXT,
5   PRIMARY KEY (equipmentID),
6   FOREIGN KEY (roomID) REFERENCES Room(roomID)
7 );
8
9 INSERT INTO Equipment VALUES ('E1234', 'R1314', 'projector');
10 INSERT INTO Equipment VALUES ('E0003', 'R1314', 'speaker');
11 INSERT INTO Equipment VALUES ('E1308', 'R3334', 'projector');
12 INSERT INTO Equipment VALUES ('E0001', 'R0002', 'table');
13 INSERT INTO Equipment VALUES ('E8923', 'R0005', 'chair');
```

	equipmentID	roomID	name
1	E1234	R1314	projector
2	E0003	R1314	speaker
3	E1308	R3334	projector
4	E0001	R0002	table
5	E8923	R0005	chair

□ **Reservation** (reserveID, roomID, startTime, endTime, date)

```
CREATE TABLE Reservation (
    reserveID TEXT,
    roomID TEXT,
    startTime TEXT,
    endTime TEXT,
    date TEXT,
    PRIMARY KEY (reserveID),
    FOREIGN KEY (roomID) REFERENCES Room(roomID)
    ON UPDATE CASCADE
    ON DELETE CASCADE,
    CHECK (endTime > startTime)
);
```

The *reserveID* is a string with the first two characters “RS”, followed by 6 numbers. Consequently, it should have type TEXT. *roomID* refers to the same-name attribute in Room so it has type TEXT as well. Once again, the logic and concept about time in SQLite is applied here to define the type (TEXT) and condition of attributes *startTime*, *endTime* and *date*. The reservation ID is unique for each reservation, so it’s enough to be the primary key. We have a foreign key *roomID* referring to the ID of the room that is reserved. If there are any changes happen to the room’s ID, the foreign key should mimic those changes as well.

```

INSERT INTO Reservation VALUES ('RS182347', 'R0002', '09:00',
'12:00', '2019-09-14');

INSERT INTO Reservation VALUES ('RS020233', 'R1314', '14:30',
'16:00', '2020-06-22');

INSERT INTO Reservation VALUES ('RS100475', 'R7832', '12:15',
'13:45', '2020-06-14');

INSERT INTO Reservation VALUES ('RS172897', 'R0005', '09:20',
'11:15', '2020-04-04');

INSERT INTO Reservation VALUES ('RS000992', 'R3334', '10:15',
'11:30', '2020-05-06');


INSERT INTO Reservation VALUES ('RS090517', 'R0002', '13:00',
'15:00', '2019-09-15');

INSERT INTO Reservation VALUES ('RS231111', 'R1314', '17:00',
'19:00', '2020-06-23');

INSERT INTO Reservation VALUES ('RS121212', 'R7832', '14:00',
'16:00', '2020-06-15');

INSERT INTO Reservation VALUES ('RS756234', 'R0005', '14:20',
'16:20', '2020-04-05');

INSERT INTO Reservation VALUES ('RS067584', 'R3334', '13:30',
'15:30', '2020-05-07');


INSERT INTO Reservation VALUES ('RS112233', 'R0002', '16:00',
'18:00', '2019-09-16');

INSERT INTO Reservation VALUES ('RS334455', 'R1314', '09:00',
'11:00', '2020-06-24');

INSERT INTO Reservation VALUES ('RS556677', 'R7832', '08:15',
'10:15', '2020-06-16');

INSERT INTO Reservation VALUES ('RS778899', 'R0005', '17:15',
'19:30', '2020-04-06');

```

```

1 CREATE TABLE Reservation (
2   reserveID TEXT,
3   roomID TEXT,
4   startTime TEXT,
5   endTime TEXT,
6   date TEXT,
7   PRIMARY KEY (reserveID),
8   FOREIGN KEY (roomID) REFERENCES Room(roomID)
9   ON DELETE CASCADE,
10  CHECK (endTime > startTime)
11 );
12
13 INSERT INTO Reservation VALUES ('RS182347', 'R0002', '09:00', '12:00', '2019-09-14');
14 INSERT INTO Reservation VALUES ('RS020233', 'R1314', '14:30', '16:00', '2020-06-22');
15 INSERT INTO Reservation VALUES ('RS100475', 'R7832', '12:15', '13:45', '2020-06-14');
16 INSERT INTO Reservation VALUES ('RS172897', 'R0005', '09:20', '11:15', '2020-04-04');
17 INSERT INTO Reservation VALUES ('RS000992', 'R3334', '10:15', '11:30', '2020-05-06');

```

```

1 INSERT INTO Reservation VALUES ('RS090517', 'R0002', '13:00', '15:00', '2019-09-15');
2 INSERT INTO Reservation VALUES ('RS231111', 'R1314', '17:00', '19:00', '2020-06-23');
3 INSERT INTO Reservation VALUES ('RS121212', 'R7832', '14:00', '16:00m', '2020-06-15');
4 INSERT INTO Reservation VALUES ('RS756234', 'R0005', '14:20', '16:20', '2020-04-05');
5 INSERT INTO Reservation VALUES ('RS067584', 'R3334', '13:30', '15:30', '2020-05-07');
6
7 INSERT INTO Reservation VALUES ('RS112233', 'R0002', '16:00', '18:00', '2019-09-16');
8 INSERT INTO Reservation VALUES ('RS334455', 'R1314', '09:00', '11:00', '2020-06-24');
9 INSERT INTO Reservation VALUES ('RS556677', 'R7832', '08:15', '10:15', '2020-06-16');
10 INSERT INTO Reservation VALUES ('RS778899', 'R0005', '17:15', '19:30', '2020-04-06');

```

	reserveID	roomID	startTime	endTime	date
1	RS182347	R0002	09:00	12:00	2019-09-14
2	RS020233	R1314	14:30	16:00	2020-06-22
3	RS100475	R7832	12:15	13:45	2020-06-14
4	RS172897	R0005	09:20	11:15	2020-04-04
5	RS000992	R3334	10:15	11:30	2020-05-06
6	RS090517	R0002	13:00	15:00	2019-09-15
7	RS231111	R1314	17:00	19:00	2020-06-23
8	RS121212	R7832	14:00	16:00m	2020-06-15
9	RS756234	R0005	14:20	16:20	2020-04-05
10	RS067584	R3334	13:30	15:30	2020-05-07
11	RS112233	R0002	16:00	18:00	2019-09-16
12	RS334455	R1314	09:00	11:00	2020-06-24
13	RS556677	R7832	08:15	10:15	2020-06-16
14	RS778899	R0005	17:15	19:30	2020-04-06

❑ EnrollExam (studentID, examID)

```

CREATE TABLE EnrollExam (
    studentID TEXT,
    examID TEXT,
    PRIMARY KEY (studentID, examID)
);

```

Attributes *studentID* and *examID* must be consistent with attribute *studentID* of relation Student and *examID* of relation Exam. Therefore, they have type TEXT. Either of the attribute is not unique enough to distinguish an enrollment for an exam, so we need to use both of them to form the primary key. (Because one student can enroll for more than one exam and one exam has more than one student).

```

INSERT INTO EnrollExam VALUES ('783903', 'A1022');
INSERT INTO EnrollExam VALUES ('678173', 'L2222');
INSERT INTO EnrollExam VALUES ('788788', 'S1234');
INSERT INTO EnrollExam VALUES ('230984', 'L2011');
INSERT INTO EnrollExam VALUES ('283740', 'C2010');

```

```

1 CREATE TABLE EnrollExam (
2   studentID TEXT,
3   examID TEXT,
4   PRIMARY KEY (studentID, examID)
5 );
6
7 INSERT INTO EnrollExam VALUES ('783903', 'A1022');
8 INSERT INTO EnrollExam VALUES ('678173', 'L2222');
9 INSERT INTO EnrollExam VALUES ('788788', 'S1234');
10 INSERT INTO EnrollExam VALUES ('230984', 'L2011');
11 INSERT INTO EnrollExam VALUES ('283740', 'C2010');

```

	studentID	examID
1	783903	A1022
2	678173	L2222
3	788788	S1234
4	230984	L2011
5	283740	C2010

❑ **EnrollGroup (studentID, groupID)**

```
CREATE TABLE EnrollGroup (
    studentID TEXT,
    groupID TEXT,
    PRIMARY KEY (studentID, groupID)
);
```

Attributes *studentID* and *groupID* must be consistent with attribute *studentID* of relation Student and *groupInstID* of relation GroupInstance. Therefore, they have type TEXT. Either of the attribute is not unique enough to distinguish an enrollment for a group, so we need to use both of them to form the primary key. (Because one student may enroll for several group, and one group may have several students).

```
INSERT INTO EnrollGroup VALUES ('783903', 'G1111');
INSERT INTO EnrollGroup VALUES ('788788', 'G1231');
INSERT INTO EnrollGroup VALUES ('678173', 'G1234');
INSERT INTO EnrollGroup VALUES ('283740', 'G1298');
INSERT INTO EnrollGroup VALUES ('230984', 'G1235');
INSERT INTO EnrollGroup VALUES ('783903', 'G1231');
```

```

1 CREATE TABLE EnrollGroup (
2 studentID TEXT,
3 groupID TEXT,
4 PRIMARY KEY (studentID, groupID)
5 );
6
7 INSERT INTO EnrollGroup VALUES ('783903', 'G1111');
8 INSERT INTO EnrollGroup VALUES ('788788', 'G1231');
9 INSERT INTO EnrollGroup VALUES ('678173', 'G1234');
10 INSERT INTO EnrollGroup VALUES ('283740', 'G1298');
11 INSERT INTO EnrollGroup VALUES ('230984', 'G1235');

```

```

1 INSERT INTO EnrollGroup VALUES ('783903', 'G1231');

```

	studentID	groupID
1	783903	G1111
2	788788	G1231
3	678173	G1234
4	283740	G1298
5	230984	G1235
6	783903	G1231

❑ ExamReserve (examID, reserveID)

```

CREATE TABLE ExamReserve (
examID TEXT,
reserveID TEXT,
PRIMARY KEY (examID, reserveID)
);

```


Attributes *examID* and *reserveID* must be consistent with attribute *examID* of relation Exam and *reserveID* of relation Reservation. Therefore, they have type TEXT. Either of the attribute is not unique enough to distinguish a reservation for an exam, so we need to use both of them to form the primary key. (Because one exam can have several reservations as it can be organized several times during the year, or organized in several rooms at the same time. Plus, for an exam reservation, there can be more than one exam organized at the same time, under the same reservation).

```
INSERT INTO ExamReserve VALUES ('A1022', 'RS000992');
INSERT INTO ExamReserve VALUES ('C2010', 'RS000992');
INSERT INTO ExamReserve VALUES ('L2222', 'RS172897');
INSERT INTO ExamReserve VALUES ('L2011', 'RS182347');
INSERT INTO ExamReserve VALUES ('S1234', 'RS020233');
```

```
1 CREATE TABLE ExamReserve (
2 examID TEXT,
3 reserveID TEXT,
4 PRIMARY KEY (examID, reserveID)
5 );
6
7 INSERT INTO ExamReserve VALUES ('A1022', 'RS000992');
8 INSERT INTO ExamReserve VALUES ('C2010', 'RS000992');
9 INSERT INTO ExamReserve VALUES ('L2222', 'RS172897');
10 INSERT INTO ExamReserve VALUES ('L2011', 'RS182347');
11 INSERT INTO ExamReserve VALUES ('S1234', 'RS020233');
```

	examID	reserveID
1	A1022	RS000992
2	C2010	RS000992
3	L2222	RS172897
4	L2011	RS182347
5	S1234	RS020233

3. Which kind of SQL queries might be typical for the database? Create reasonable indexing which supports the purposeful use of the database. In addition add at least one useful view definition to your database.

★ **INDEXING:**

For most relation, we will create index on the primary key because queries that involve primary key is common. Moreover, an index on primary key will only yields a single page because primary key is unique.

- Course(courseCode, name, credits)

→ For relation Course, we will create index on courseCode

```
CREATE INDEX CourseIndex ON Course(courseCode);
```

- CourseInstance(courseInsID, courseID, startDate, endDate)

→ For relation CourseInstance, we will create index on both courseInsID

```
CREATE INDEX CourseInsIndex ON CourseInstance(courseInsID);
```

- Exam(examID, courseID, startTime, endTime, date)

→ For relation Exam, we will create index on examID

```
CREATE INDEX ExamIndex ON Exam(examID);
```

- ExerciseGroup(groupID, courseInsID, studentLimit)

→ For relation ExerciseGroup, we will create index on groupID

```
CREATE INDEX GroupIndex ON ExerciseGroup(groupID);
```

- GroupInstance(groupInsID, groupID, reserveID, startTime, endTime, date)

→ For relation GroupInstance, we will create index on groupInsID

```
CREATE INDEX GroupInsIndex ON GroupInstance(groupInsID);
```

- LectureInstance(lectureInsID, lectureID, reserveID, startTime, endTime, date)

→ For relation LectureInstance, we will create index on lectureInsID

```
CREATE INDEX LectureInsIndex ON LectureInstance(lectureInsID);
```

- Student(studentID, name, dateOfBirth, degreeProgram, year, expirationDate)

→ For relation Student, we will create index on studentID

```
CREATE INDEX StudentIndex ON Student(studentID);
```

- Room(roomID, buildingID, numberOfSeats, numberOfSeatInExam)

→ For relation Room, we will create index on roomID

```
CREATE INDEX RoomIndex ON Room(RoomID);
```

- Building(buildingID, name, address)

→ For relation Building, we will create index on buildingID

```
CREATE INDEX BuildingIndex ON Building(buildingID);
```

- Equipment(equipmentID, roomID, name)

→ For relation Equipment, we will create index on equipmentID

```
CREATE INDEX EquipmentIndex ON Equipment(equipmentID);
```

- Reservation(reserveID, roomID, startTime, endTime, date)

→ For relation Reservation, we will create index on reserveID

```
CREATE INDEX ReserveIndex ON Reservation(reserveID);
```

★ **VIEW:**

- We will construct views for all students of each Degree Program.
- For example, we will construct a view Bachelor giving the studentID, name, dateOfBirth, degreeProgram, year, expirationDay of all the students who study Bachelor degree.

```
1 CREATE VIEW Bachelor AS
2 SELECT studentID, name, dateOfBirth, degreeProgram, year, expirationDate
3 FROM Student
4 WHERE degreeProgram = 'Bachelor'
```

```
CREATE VIEW Bachelor AS

SELECT  studentID,   name,   dateOfBirth,   degreeProgram,   year,
expirationDate

FROM Student

WHERE degreeProgram = 'Bachelor';
```

- We will construct a view for all the rooms in each Building
- For example, we will construct a view CSBuilding giving the roomID, buildingID, numberOfSeats, numberOfSeatsInExam of all the rooms in CS Building.

```
1 CREATE VIEW CSBuilding AS
2     SELECT roomID, buildingID, numberOfSeats, numberOfSeatsInExam
3     FROM Room, Building
4     WHERE Building.name = 'CS Building'
5     AND Room.buildingID = Building.buildingID
```

```
CREATE VIEW CSBuilding AS

SELECT roomID, buildingID, numberOfSeats, numberOfSeatsInExam

FROM Room, Building

WHERE Building.name = 'CS Building'

AND Room.buildingID = Building.buildingID;
```

4. Create some typical use cases for the database. Write a description of the use case, explain which information needs to be selected/updated/inserted and write the SQL queries and possible other commands that are needed to fulfill the use case. The number of SQL queries (SQL statements which begin with the word SELECT and end with semicolon) written in the document must be at least 15. (One use case may contain several queries.) Try to find at least one use case where you need to use a little bit more complicated SQL query, for example, including join of several tables, grouping and calculating a value of an aggregate function. In addition to queries, many of your use cases probably need other SQL commands, like inserts or deletes.

★ **TYPICAL USE CASES FOR THE DATABASES**

★ SELECTED:

- ✓ Find the studentID, student name, courseCode, name, credits of all the courses that a specific student is attending.

For example, a student with studentID 678173 want to find the his studentID, his name, courseCode, name and credits of all the courses he is enrolling for.

```
1 SELECT Student.studentID, Student.name, courseCode, name, credits
2 FROM Course, CourseInstance, ExerciseGroup, EnrollGroup, Student
3 WHERE Course.courseCode = CourseInstance.courseID
4 AND CourseInstance.courseInsID = ExerciseGroup.courseInsID
5 AND ExerciseGroup.groupID = EnrollGroup.groupID
6 AND EnrollGroup.studentID = Student.studentID
7 AND EnrollGroup.studentID = '678173'
```

```
SELECT Student.studentID, Student.name, courseCode, name, credits
FROM Course, CourseInstance, ExerciseGroup, EnrollGroup, Student
WHERE Course.courseCode = CourseInstance.courseID
AND CourseInstance.courseInsID = ExerciseGroup.courseInsID
AND ExerciseGroup.groupID = EnrollGroup.groupID
AND EnrollGroup.studentID = Student.studentID
AND EnrollGroup.studentID = '678173';
```

RESULT :

	studentID	name	courseCode	name:1	credits
1	678173	Matthew	LC-1111	Finnish 1A	2

- ✓ Find studentID, name of all students that attends a specific course.

For example, we want to find studentID, name of all students that enroll in the course with courseID 'CS-1010'

```
1 SELECT studentID, name
2 FROM Student, EnrollGroup, ExerciseGroup, CourseInstance, Course
3 WHERE Student.studentID = EnrollGroup.studentID
4 AND EnrollGroup.groupID = ExerciseGroup.groupID
5 AND ExerciseGroup.courseInsID = CourseInstance.courseInsID
6 AND CourseInstance.courseID = Course.courseCode
7 AND courseCode = 'CS-1010' |
```

```
SELECT studentID, name
FROM Student, EnrollGroup, ExerciseGroup, CourseInstance, Course
```

```

WHERE Student.studentID = EnrollGroup.studentID
AND EnrollGroup.groupID = ExerciseGroup.groupID
AND ExerciseGroup.courseInsID = CourseInstance.courseInsID
AND CourseInstance.courseID = Course.courseCode
AND courseCode = 'CS-1010';

```

RESULT :

	studentID	name
1	788788	Binh

- ✓ Find all the groupID, studentLimit and the number of students of all the exercise groups of a specific course that are still available, which means the number of students is not yet exceed the student limit.
- ✓ For example, Find all the groupID, studentLimit and the number of students of all the exercise groups of courseID 'KA-1406'

```

1 SELECT ExerciseGroup.groupID, studentLimit, COUNT(DISTINCT studentID)
2 FROM ExerciseGroup JOIN EnrollGroup ON ExerciseGroup.groupID = EnrollGroup.groupID
3 WHERE ExerciseGroup.courseInsID IN (SELECT courseInsID
4                                     FROM CourseInstance
5                                     WHERE courseID = 'KA-1406')
6 GROUP BY ExerciseGroup.groupID
7 HAVING COUNT(DISTINCT studentID) < studentLimit |

```

```

SELECT groupID, studentLimit, COUNT(DISTINCT studentID)

FROM ExerciseGroup JOIN EnrollGroup ON ExerciseGroup.groupID =
EnrollGroup.groupID

WHERE ExerciseGroup.courseInsID IN (SELECT courseIns
                                     FROM CourseInstance
                                     WHERE courseID = 'KA-1406')

GROUP BY groupID

HAVING COUNT(DISTINCT studentID) < studentLimit;

```

RESULT :

	groupID	studentLimit	COUNT(DISTINCT studentID)
1	G1298	30	1

- ✓ Find all the rooms that are available at a specific time that have at least n number of seats.

For example, a professor wants to find all the rooms that are available between 8:00 and 13:00 in '2019-09-14' and have the number of seats greater or equal to 45

```

1 SELECT roomID, numberOfSeats, numberOfSeatsInExam
2 FROM Room
3 WHERE Room.roomID NOT IN (SELECT roomID
4                           FROM Reservation
5                           WHERE startTime >= '08:00'
6                           AND endTime <= '13:00'
7                           AND date = '2019-09-14')
8 AND Room.numberOfSeats >= 45

```

```

SELECT roomID, numberOfSeats, numberOfSeatsInExam
FROM Room
WHERE Room.roomID NOT IN (SELECT roomID
                           FROM Reservation
                           WHERE startTime >= '08:00'
                           AND endTime <= '13:00'
                           AND date = '2019-09-14')
AND Room.numberOfSeats >= 45;

```

RESULT :

	roomID	numberOfSeats	numberOfSeatsInExam
1	R7832	100	50
2	R1314	200	150

- ✓ A student wants to find the buildingID, name and address of the building that have the room his/her lecture will happen.

For example, a student wants to find the buildingID, the name and the address of the building that the lecture instance 'L222A' will happen.

```
1 SELECT Building.buildingID, Building.name, address
2 FROM Building, Room, LectureInstance, Reservation
3 WHERE Reservation.reserveID = LectureInstance.reserveID
4 AND Reservation.roomID = Room.roomID
5 AND Room.buildingID = Building.buildingID
6 AND LectureInstance.lectureInsID = 'L222A'
```

```
SELECT buildingID, name, address
FROM Building, Room, LectureInstance, Reservation
WHERE Reservation.reserveID = LectureInstance.reserveID
AND Reservation.roomID = Room.roomID
AND Room.buildingID = Building.buildingID
AND LectureInstance.lectureInsID = 'L222A';
```

RESULT:

	buildingID	name	address
1	B03	CS Building	Jamerantaival 1

- ✓ Find all the room that has a specific equipment

For example, the teacher wants to find all the rooms that have the projector for his/her presentation.

```
1 SELECT Room.roomID, numberOfSeats, numberOfSeatsInExam
2 FROM Room, Equipment
3 WHERE Equipment.roomID = Room.roomID
4 AND Equipment.name = 'projector'
```

```
SELECT roomID, numberOfSeats, numberOfSeatsInExam
FROM Room, Equipment
WHERE Equipment.roomID = Room.roomID
AND Equipment.name = 'projector';
```

RESULT:

	roomID	numberOfSeats	numberOfSeatsInExam
1	R1314	200	150
2	R3334	20	10

- ✓ Find studentID, name, date of birth, degree program, year, expiration date and the sum of credits of all the course he/she is taking.

```

1 SELECT studentID, Student.name, dateOfBirth, degreeProgram, year, expirationDate, SUM(Course.credits)
2 FROM Course, CourseInstance, ExerciseGroup, EnrollmentGroup, Student
3 WHERE Course.courseCode = CourseInstance.courseID
4 AND CourseInstance.courseInsID = ExerciseGroup.courseInsID
5 AND ExerciseGroup.groupID = EnrollmentGroup.groupID
6 AND EnrollmentGroup.studentID = Student.studentID
7 GROUP BY Course.courseCode |

```

```

SELECT studentID, Student.name, dateOfBirth, degreeProgram, year,
expirationDate, SUM(Course.credits)

```

```

FROM Course, CourseInstance, ExerciseGroup, EnrollmentGroup, Student

```

```

WHERE Course.courseCode = CourseInstance.courseID

```

```

AND CourseInstance.courseInsID = ExerciseGroup.courseInsID

```

```

AND ExerciseGroup.groupID = EnrollmentGroup.groupID

```

```

AND EnrollmentGroup.studentID = Student.studentID

```

```

GROUP BY Course.courseCode;

```

RESULT :

	studentID	name	dateOfBirth	degreePr	year	expirationDat	SUM(Co
1	788788	Binh	2001-14-02	Bachelor	2	2026-08-30	3
2	783903	Long	1999-13-08	Bachelor	2	2026-08-30	5
3	283740	Kris	1992-14-06	Master	1	2022-04-20	5
4	230984	Vera	1997-11-01	Master	3	2025-01-21	4

*Find all the studentID and name of all students that have enrolled for more than n credits.

For example, we want to find all the studentID and name of all students that have enrolled for more than 3 credits.

```

1 SELECT studentID, Student.name, dateOfBirth, degreeProgram, year, expirationDate, SUM(Course.credits)
2 FROM Course, CourseInstance, ExerciseGroup, EnrollmentGroup, Student
3 WHERE Course.courseCode = CourseInstance.courseID
4 AND CourseInstance.courseInsID = ExerciseGroup.courseInsID
5 AND ExerciseGroup.groupID = EnrollmentGroup.groupID
6 AND EnrollmentGroup.studentID = Student.studentID
7 GROUP BY Course.courseCode
8 HAVING SUM(Course.credits) > 3 |

```

```

SELECT studentID, Student.name, dateOfBirth, degreeProgram, year,
expirationDate, SUM(Course.credits)

```

```

FROM Course, CourseInstance, ExerciseGroup, EnrollGroup, Student
WHERE Course.courseCode = CourseInstance.courseID
AND CourseInstance.courseInsID = ExerciseGroup.courseInsID
AND ExerciseGroup.groupID = EnrollGroup.groupID
AND EnrollGroup.studentID = Student.studentID
GROUP BY Course.courseCode
HAVING SUM(Course.credits) > 3;

```

RESULT:

	studentID	name	dateOfBirth	degreePr	year	expirationDat	SUM(Coi
1	783903	Long	1999-13-08	Bachelor	2	2026-08-30	5
2	283740	Kris	1992-14-06	Master	1	2022-04-20	5
3	230984	Vera	1997-11-01	Master	3	2025-01-21	4

- ✓ Find the buildingID, name, address and the total number of rooms of all buildings.

```

1 SELECT Building.buildingID, Building.name, address, COUNT(DISTINCT Room.roomID)
2 FROM Building, Room
3 WHERE Building.buildingID = Room.buildingID
4 GROUP BY Building.buildingID |

```

```

SELECT Building.buildingID, Building.name, address,
COUNT (DISTINCT Room.roomID)
FROM Building, Room
WHERE Building.buildingID = Room.buildingID
GROUP BY Building.buildingID;

```

RESULT:

	buildingID	name	address	COUNT(DISTINCT Room.roomID)
1	B01	Otakaari 1	111 Otaniemi	1
2	B03	CS Building	Jamerantaival 1	2
3	B08	Library	99 Central City	1
4	B11	Alvari	84 Otaniemi	1

*The staff want to find the building with more than one room to be the main buildings of the University.

```
1 SELECT Building.buildingID, Building.name, address, COUNT(DISTINCT Room.roomID)
2 FROM Building, Room
3 WHERE Building.buildingID = Room.buildingID
4 GROUP BY Building.buildingID
5 HAVING COUNT(DISTINCT Room.roomID) > 1
```

```
SELECT Building.buildingID, Building.name, address,
COUNT(DISTINCT Room.roomID)
FROM Building, Room
WHERE Building.buildingID = Room.buildingID
GROUP BY Building.buildingID
HAVING COUNT(DISTINCT Room.roomID) > 1;
```

RESULT :

	buildingID	name	address	COUNT(DISTINCT Room.roomID)
1	B03	CS Building	Jamerantaival 1	2

- ✓ Find the studentID, name of all the students that enroll in both course A and course B.

For example, we want to find the studentID and name all the students that enroll in both course DB-1123 and course CS-1010

```
1 SELECT studentID, name
2 FROM Student
3 WHERE studentID IN (SELECT Student.studentID
4 FROM Student, CourseInstance, ExerciseGroup, EnrollGroup
5 WHERE CourseInstance.courseInsID = ExerciseGroup.courseInsID
6 AND ExerciseGroup.groupID = EnrollGroup.groupID
7 AND EnrollGroup.studentID = Student.studentID
8 AND CourseInstance.courseID = 'DB-1123')
9 AND studentID IN (SELECT Student.studentID
10 FROM Student, CourseInstance, ExerciseGroup, EnrollGroup
11 WHERE CourseInstance.courseInsID = ExerciseGroup.courseInsID
12 AND ExerciseGroup.groupID = EnrollGroup.groupID
13 AND EnrollGroup.studentID = Student.studentID
14 AND CourseInstance.courseID = 'CS-1010')
```

```
SELECT studentID, name
FROM Student
WHERE studentID IN (SELECT Student.studentID
FROM Student, CourseInstance, ExerciseGroup, EnrollGroup
WHERE CourseInstance.courseInsID = ExerciseGroup.courseInsID
```

```

AND ExerciseGroup.groupID = EnrollGroup.groupID
AND EnrollGroup.studentID = Student.studentID
AND CourseInstance.courseID = 'DB-1123')
AND studentID IN (SELECT Student.studentID
FROM Student, CourseInstance, ExerciseGroup, EnrollGroup
WHERE CourseInstance.courseInsID = ExerciseGroup.courseInsID
AND ExerciseGroup.groupID = EnrollGroup.groupID
AND EnrollGroup.studentID = Student.studentID
AND CourseInstance.courseID = 'CS-1010');

```

RESULT:

	studentID	name
1	783903	Long

- ✓ Find the examID, startTime, endTime and date of all the exams that have reservation in a specific room.

For example, we want to find the examID, startTime, endTime and date of all the exams that have reservation in room R3334

```

1 SELECT Exam.examID, Exam.startTime, Exam.endTime, Exam.date
2 FROM Exam, Room, ExamReserve, Reservation
3 WHERE Room.roomID = Reservation.roomID
4 AND Reservation.reserveID = ExamReserve.reserveID
5 AND ExamReserve.examID = Exam.examID
6 AND Room.roomID = 'R3334'

```

```

SELECT Exam.examID, Exam.startTime, Exam.endTime, Exam, date
FROM Exam, Room, ExamReserve, Reservation
WHERE Room.roomID = Reservation.roomID
AND Reservation.reserveID = ExamReserve.reserveID
AND ExamReserve.examID = Exam.examID
AND Room.roomID = 'R3334';

```

RESULT :

	examID	startTime	endTime	date
1	A1022	12:00	15:00	2020-03-03
2	C2010	08:00	11:30	2020-02-06

- ✓ Find the degree program and the total number of students in each program.

```
1 SELECT degreeProgram, COUNT(DISTINCT studentID)
2 FROM Student
3 GROUP BY degreeProgram
```

```
SELECT degreeProgram, COUNT(DISTINCT studentID)
FROM Student
GROUP BY degreeProgram;
```

RESULT :

	degreeProgram	COUNT(DISTINCT studentID)
1	Bachelor	3
2	Master	2

- ✓ Find the examID, courseID, name, startTime, endTime, date of the exam and the number of students enrolled in that exam.

```
1 SELECT Exam.examID, Course.courseCode, Course.name, COUNT(DISTINCT Student.studentID)
2 FROM Exam, Course, Student, EnrollExam
3 WHERE Course.courseCode = Exam.courseID
4 AND Exam.examID = EnrollExam.examID
5 AND EnrollExam.studentID = Student.studentID
6 GROUP BY Exam.examID
```

```
SELECT Exam.examID, Course.courseCode, Course.name,
COUNT(DISTINCT Student.studentID)
FROM Exam, Course, Student, EnrollExam
WHERE Course.courseCode = Exam.courseID
AND Exam.examID = EnrollExam.examID
```

```
AND EnrollExam.studentID = Student.studentID  
GROUP BY Exam.examID;
```

RESULT :

	examID	courseCode	name	COUNT(DISTINCT Student.studentID)
1	A1022	DB-1123	Databases	1
2	C2010	CS-1010	Machine Learning	1
3	L2011	LC-1111	Finnish 1A	1
4	L2222	LC-1111	Finnish 1A	1
5	S1234	KA-1406	Programming I	1

★ INSERT:

- ✓ Enroll a specific student to an exam. And then find the studentID and name of all the students that have enrolled in that exam.








For example, we want to enroll student with studentID 678173 to an exam with ID A1022

```
1 INSERT INTO EnrollExam(studentID,examID) VALUES ('678173', 'A1022');
```

```
INSERT INTO EnrollExam(studentID,examID)  
VALUES ('678173', 'A1022');
```

Check the relation EnrollExam afterwards:

```
1 SELECT * FROM EnrollExam;
```

					1		
	studentID	examID					
1	783903	A1022					
2	678173	L2222					
3	788788	S1234					
4	230984	L2011					
5	283740	C2010					
6	678173	A1022					

Then, we want to find the studentID and name of all the students that have enrolled in exam with ID A1022

```
1 SELECT studentID, name
2 FROM Student, EnrollExam
3 WHERE Student.studentID = EnrollExam.studentID
4 AND EnrollExam.examID = 'A1022' |
```

```
SELECT studentID, name
FROM Student, EnrollExam
WHERE Student.studentID = EnrollExam.studentID
AND EnrollExam.examID = 'A1022';
```

RESULT:

	studentID	name
1	783903	Long
2	678173	Matthew

★ **UPDATE:**

- ✓ A student finished his/her bachelor degree and start his/her master degree. Therefore, we need to update information in relation Student. Then we want to find all information about that student.

```

1 UPDATE Student
2 SET degreeProgram = 'Master'
3 WHERE studentID = '678173'

```

UPDATE Student









SET degreeProgram = 'master'

WHERE studentID = '678173';

```

1 SELECT *
2 FROM Student
3 WHERE studentID = '678173'

```

<div>      <div>1</div>    </div> <div>Total rows loaded: 1</div>						
	studentID	name	dateOfBirth	degreeProgram	year	expirationDate
1	678173	Matthew	1988-09-09	Master	4	2024-08-29


SELECT *

FROM Student

WHERE studentID = '678173';

Check the table Student afterwards:


```
1 SELECT * FROM Student;
```

 Total rows loaded: 5						
	studentID	name	dateOfBirth	degreePr	year	expirationDat
1	788788	Binh	2001-14-02	Bachelor	2	2026-08-30
2	678173	Matthew	1988-09-09	Master	4	2024-08-29
3	283740	Kris	1992-14-06	Master	1	2022-04-20
4	230984	Vera	1997-11-01	Master	3	2025-01-21
5	783903	Long	1999-13-08	Bachelor	2	2026-08-30

- ✓ A student wants to remove his/her registration from a group exercise and enroll in another group exercise.

```
1 UPDATE EnrollGroup
2 SET groupID = 'G1234'
3 WHERE studentID = '783903'
4 AND groupID = 'G1111'
```

```
UPDATE EnrollGroup
SET groupID = 'G1234'
WHERE studentID = '783903'
AND groupID = 'G1111';
```

*Find the studentID and name of all the students who enroll in the group 'G1234'

```
1 SELECT Student.studentID, name
2 FROM Student, EnrollGroup
3 WHERE Student.studentID = EnrollGroup.studentID
4 AND groupID = 'G1234';
```

```
SELECT Student.studentID, name
```

```
FROM Student, EnrollGroup
WHERE Student.studentID = EnrollGroup.studentID
AND groupID = 'G1234';
```

*Check the table EnrollGroup afterwards:

```
1 SELECT * FROM EnrollGroup;
```



	studentID	groupID
1	783903	G1234
2	788788	G1231
3	678173	G1234
4	283740	G1298
5	230984	G1235
6	783903	G1231

★ DELETE:

- ✓ An equipment is broken. Therefore, we have to bring it out of the room and delete it from the database. Then we want to find the equipmentID and name of all the equipment in that room.

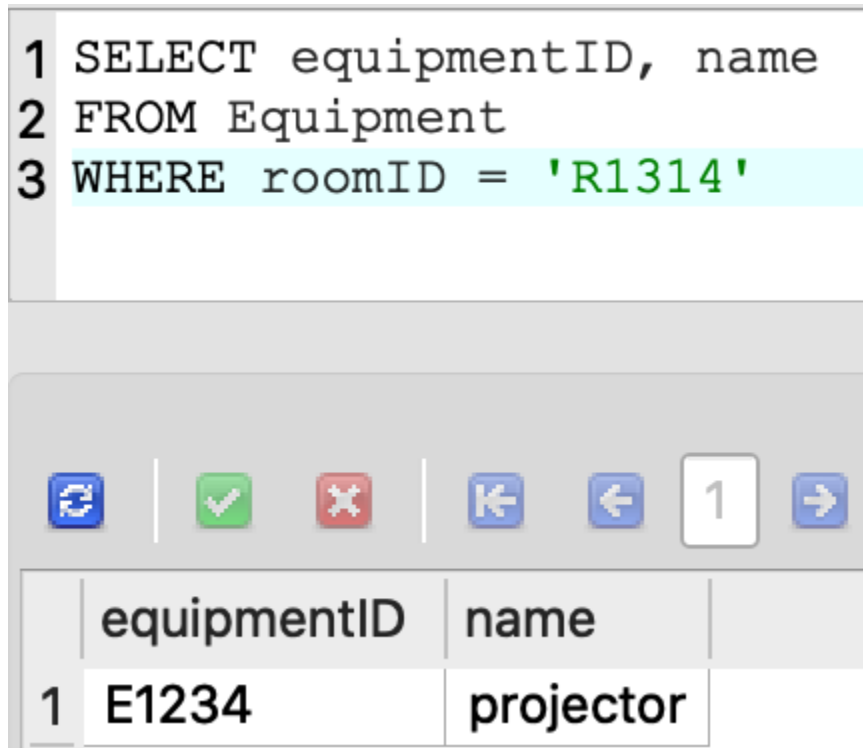
```
1 DELETE FROM Equipment
2 WHERE equipmentID = 'E0003'
3 AND roomID = 'R1314' |
```

```
DELETE FROM Equipment
```

```
WHERE equipmentID = 'E0003'

AND roomID = 'R1314';
```

*Find the equipmentID and name of all the equipment in that room.



```
SELECT equipmentID, name

FROM Equipment

WHERE roomID = 'R1314';
```

5. Run the statements (creating the tables, inserting example data in them and the SQL queries) in SQLiteStudio environment. Attach the .sql file to your submission. Insert the listing of the SQL command you run and their outputs in your document (another file).

