Aalto University
CS-A1150 - Tietokannat

Assignment: Project Part 2

Jaakko Vauhkonen - jaakko.vauhkonen@aalto.fi
Aarno Veitola - aarno.veitola@aalto.fi

Returned: May 10, 2023

Contents

1 The UML and relational model (Project part 1) 3
1.1 UML diagram . . . 3
1.2 Relational model . . . 4
1.2.1 Classes . . . 4
1.2.2 Association relations . . . 5
1.3 Boyce-Codd normal form . . . 6
1.3.1 Functional dependencies . . . 6
1.3.2 Normal form . . . 7

2 Changes to part 1 9
2.1 Changes to the UML model . . . 9
2.2 Changes to the relational model . . . 9
2.2.1 Changes to classes . . . 10
2.2.2 Changes to association relations . . . 10

3 The SQL model 11
3.1 Creating the tables . . . 11
3.2 Indexing . . . 16
3.3 Views for the database . . . 17
3.4 Examples of SQL queries for the database . . . 18

1 The UML and relational model (Project part 1)

1.1 UML diagram

We begin by constructing a UML diagram which represents the structure of our database. The UML diagram is illustrated in figure 1.

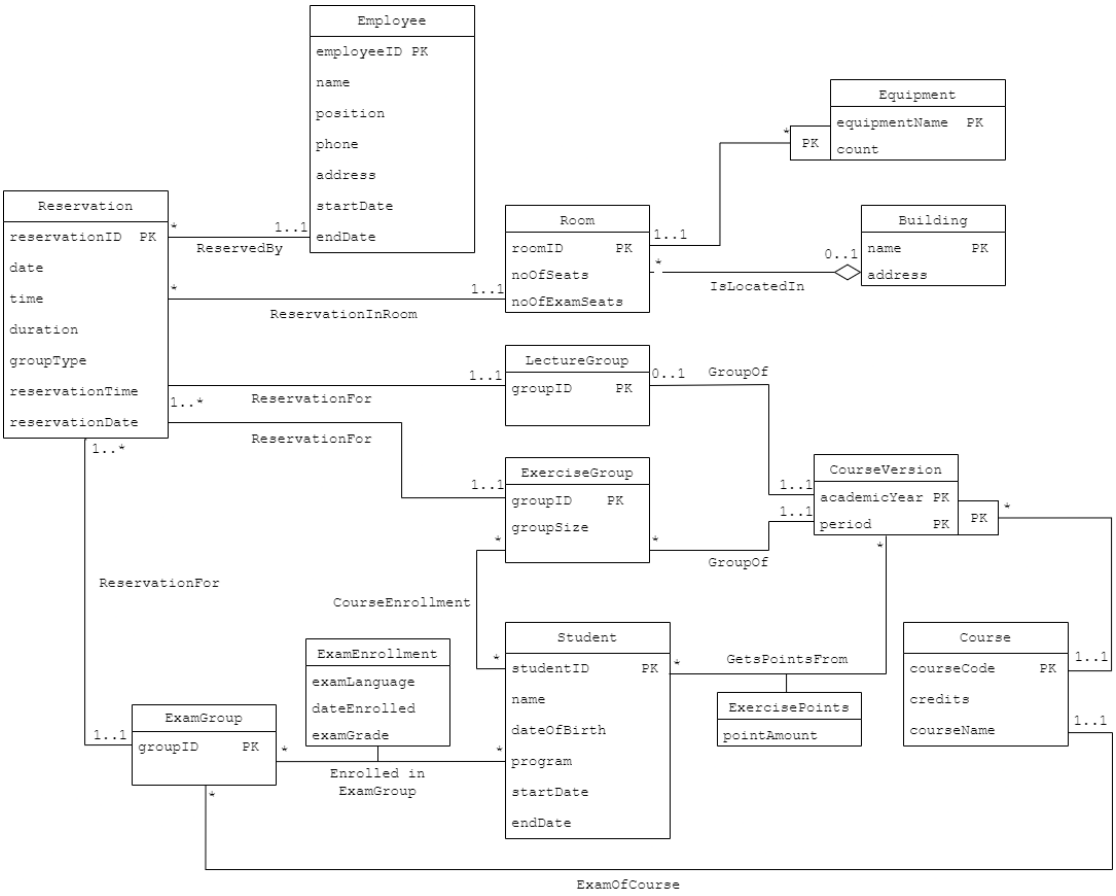

Figure 1: The UML diagram of our database.

Our implementation of the database is centered on the entity *Student*, which describes a certain student attending the school. The school curriculum is represented as instances of *Courses*, which have different versions, i.e. *CourseVersions*. The student enrolls in a course by enrolling in an *ExerciseGroup* specific to the *CourseVersion* in question. The specific version of the course also has one *LectureGroup* (or zero if the course has no lectures), which all students attend. A student's schedule depends on the *Reservations* made by the groups they are attending. For example, the dates and times for the lectures

are dictated by the *Reservations* the *LectureGroup* has. A *Course* can also have *ExamGroups*, which describe specific exam instances of the course in question. The school premises are described by instances of classes *Building* and *Room*, which represent the campus buildings and the reservable rooms they contain, respectively. The database also accounts for the *Equipment* in each reservable *Room*, and *Employees*, who make the *Reservations*.

1.2 Relational model

The relations of the model are written in form *R(A, B, C)* with key attributes underlined, and followed by an explanation of the relation.

1.2.1 Classes

Student(studentID, name, dateOfBirth, program, startDate, endDate), describes an individual student by their student ID, name, date of birth, study program, and the beginning and end of their right of study.

Course(courseCode, credits, courseName), provides the basic information of a course, i.e. the course code, credits, and the course name. It is assumed that a certain course can only have one name, i.e. a certain course is defined uniquely by its *courseCode*. This means that if two courses share their content but are completed in different languages, they will have their own course codes.

CourseVersion(courseCode, academicYear, period), specifies the course version by providing the period and year. The course can last multiple periods, i.e. the period can be for example ”IV” or ”III-V”. The same course will not be run twice simultaneously, meaning that the course code, year, and period will always lead to a unique course.

ExamGroup(groupID), describes a specific exam event for a course. The *groupID* helps differentiate course exams from each other as a single course often has multiple exams such as retakes. A certain *ExamGroup* can have multiple *Reservations*, i.e. the same exam can be held at different places at the same time, or at different times. This is useful if all the exam participants can't fit into a single room.

LectureGroup(groupID), describes a set of lectures for a given course. The group ID helps determine the reservations made for lectures of a certain course.

ExerciseGroup(groupID, groupSize), describes the set of exercise sessions for a given course. Each *Reservation* of an *ExerciseGroup* describes a different exercise session of

the same group. All exercise sessions of an *ExerciseGroup* are reserved separately.

Building(name, address), describes a building on campus. It is assumed that no two buildings have the same name.

Room(roomID, noOfSeats, noOfExamSeats), describes a room within a *Building*. It is assumed that no two rooms have the same *roomID*.

Equipment(equipmentName, roomID, count), describes a piece of equipment in a certain *Room*. The *equipmentName* does not take into consideration the specifics of certain equipment; e.g. if a *Room* has two projectors of different models, they fall under the *equipmentName* "projector" with a *count* of 2.

Employee(employeeID, name, position, phone, address, startDate, endDate), describes an employee by their employee ID, name, position, phone, address, and the start and end date of their contract. It is assumed a single employee can only have one contract at a time.

Reservation(reservationID, date, time, duration, groupType, reservationTime, reservationDate), describes a *Room* reservation made by a *LectureGroup*, *ExerciseGroup* or *ExamGroup*. Each *Reservation* includes a distinct *reservationID*, the *date*, *time* and *duration* for the reservation, the *groupType*, which is "exercise", "lecture" or "exam". The *reservationDate* and *reservationTime* are the date and time when the reservation was made.

1.2.2 Association relations

CourseEnrollment(studentID, groupID), tells who has enrolled in a particular course via the exercise group.

ExamEnrollment(studentID, groupID, examLanguage, dateEnrolled, examGrade), tells who has enrolled in an exam and which grade was received for the exam. The exam grade is NULL before the exam has been evaluated. Not attending an exam a student has enrolled in will also lead to a grade of NULL.

ExamOfCourse(groupID, courseCode), allows us to connect each exam event with the course it is for.

GetsPointsFrom(studentID, courseCode, academicYear, period, pointsAmount), tells how many exercise points a student has received during a particular course.

GroupOf(groupID, courseCode, academicYear, period), connects each lecture and exercise group to their respective courses. The group ID is unique for every single lecture and exercise group instance. This means that the group ID is as unique as a student ID and is not repeated across different versions of a course.

IsLocatedIn(roomID, name), describes in which building each room is located.

ReservationInRoom(reservationID, roomID), tells which room the reservation is for.

ReservedBy(reservationID, employeeID), tells who made the reservation.

ReservationFor(reservationID, groupID), tells which exam/lecture/exercise group the reservation is for.

1.3 Boyce-Codd normal form

1.3.1 Functional dependencies

The functional dependencies of every relation above are written below with the following syntax:

R(A,B,C):
A → B, C

Observe that relations with single attributes cannot have functional dependencies and are therefore left out of this analysis. The same applies to relations with only key attributes since they only have trivial functional dependencies, i.e. A, B, C → A, B, C.

Student(studentID, name, dateOfBirth, program, startDate, endDate):
studentID → name, dateOfBirth, program, startDate, endDate

Course(courseCode, credits, courseName):
courseCode → credits, courseName

ExerciseGroup(groupID, groupSize):
groupID → groupSize

Building(name, address):
name → address

Room(roomID, noOfSeats, noOfExamSeats):
roomID → noOfSeats, noOfExamSeats

Equipment(roomID, equipmentName, count):
roomID, equipmentName → count

Employee(employeeID, name, position, phone, address, startDate, endDate):
employeeID → name, position, phone, address, startDate, endDate

Reservation(reservationID, date, time, duration, groupType, reservationTime, reservationDate):
reservationID → date, time, duration, groupType, reservationTime, reservationDate

ExamOfCourse(groupID, courseCode):
groupID → courseCode

GetsPointsFrom(studentID, courseCode, academicYear, period, pointsAmount):
studentID, courseCode, academicYear, period → pointsAmount

GroupOf(groupID, courseCode, academicYear, period):
groupID → courseCode, academicYear, period

IsLocatedIn(roomID, name):
roomID → name

ReservationInRoom(reservationID, roomID):
reservationID → roomID

ReservedBy(reservationID, employeeID):
reservationID → employeeID

ReservationFor(reservationID, groupID):
reservationID → groupID

1.3.2 Normal form

From looking at the functional dependencies of the above relations, we can see that each relation has at most one functional dependency. From the dependencies, we can easily see that each closure of each left-hand side of the dependency contains all attributes of the relation.

For example, we have the following dependency from the relation *GroupOf*:

groupID → courseCode, academicYear, period

We can calculate the closure $\{groupID\}^+$ easily. From the only dependency, we get

$\{groupID\}^+ =$ groupID, courseCode, academicYear, period. This contains all attributes of GroupOf, which is a condition for the relation to be in Boyce-Codd normal form.

The same applies to all of the dependencies in section 1.3.1. This means that our database is already in BCNF and there is no need for decomposition.

Since all of the relations in our database are in BCNF, we have removed all redundancy caused by functional dependence. BNCF is a condition that also guarantees we have removed update and deletion anomalies in addition to the redundancy.

2 Changes to part 1

2.1 Changes to the UML model

The updated UML model used to construct the updated relational model and SQL model is presented in Figure 2. The details of the structural changes to the model are further discussed in Section 2.2.

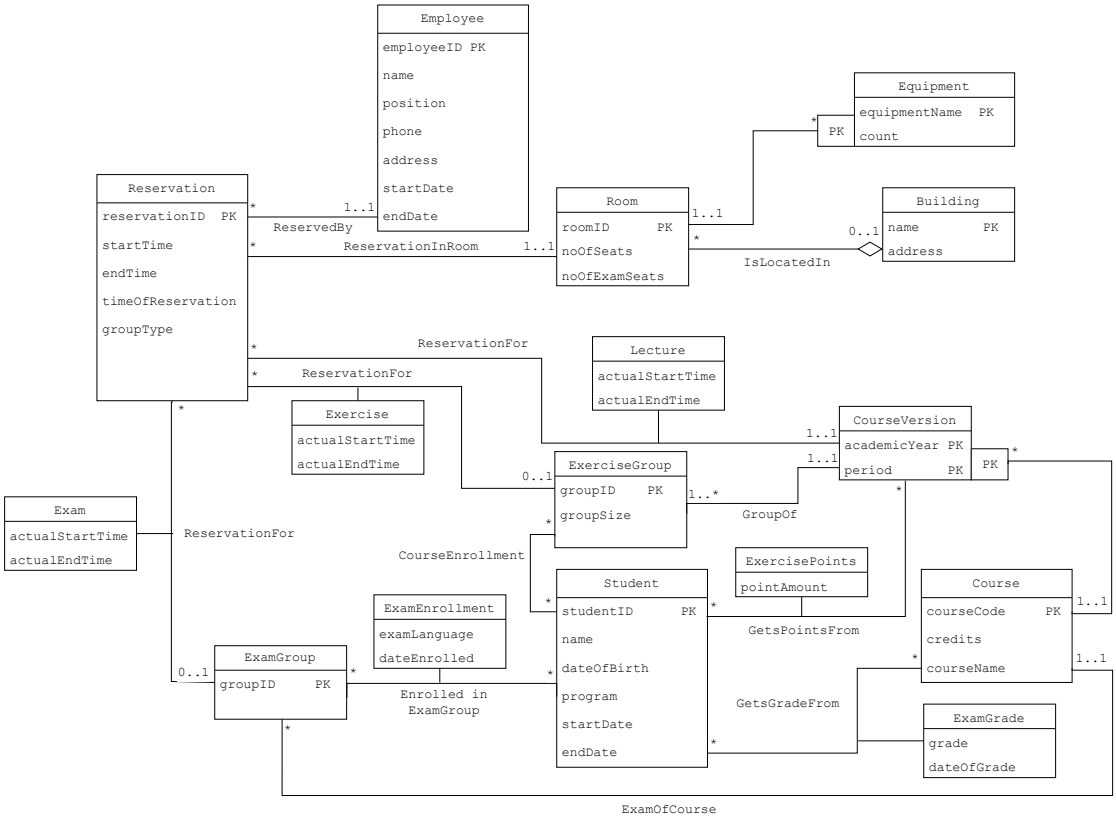

Figure 2: The updated UML diagram of our database.

2.2 Changes to the relational model

Here we have listed the relational model used in the second part of the project. There may be brief explanations of the changes made after the description of some classes.

The relations of the model are written in form *R(A, B, C)* with key attributes underlined.

2.2.1 Changes to classes

Student(studentID, name, dateOfBirth, program, startDate, endDate)

Course(courseCode, credits, courseName)

CourseVersion(courseCode, academicYear, period)

ExamGroup(groupID, courseCode), *ExamGroup* has inherited the attribute *courseCode* to get rid of the redundant class *ExamOfCourse*.

ExerciseGroup(groupID, courseCode, academicYear, period, groupSize), inherited the attributes *courseCode*, *academicYear* and *period* to get rid of the redundant class *GroupOf*.

Building(name, address)

Room(roomID, noOfSeats, noOfExamSeats)

Equipment(equipmentName, roomID, count)

Employee(employeeID, name, position, phone, address, startDate, endDate)

Reservation(reservationID, reserverID, roomID, startTime, endTime, timeOfReservation), inherited the attributes *reserverID* and *roomID* to get rid of the redundant classes *ReservationFor* and *ReservationInRoom*. The time and date of reservation, as well as the start and end time of the reservation, are now described by the attributes *startTime*, *endTime* and *timeOfReservation*, which are all of the form 'YYYY-MM-DD HH:MM'.

2.2.2 Changes to association relations

Association relations *ExamOfCourse*, *GroupOf*, *ReservationInRoom* and *ReservationBy* have been removed/replaced due to redundancy.

CourseEnrollment(studentID, groupID)

ExamEnrollment(studentID, groupID, examLanguage, dateEnrolled)

ExamGrade(studentID, courseCode, dateOfGrade, grade), this association class has been created to move the attribute grade from *ExamEnrollment* to its own class, since the information of the exam enrollment and grade are not related, and should therefore not be contained in the same class. *dateOfGrade* specifies the date the grade was given on, and allows us to have multiple grades for a given course.

ExercisePoints(studentID, courseCode, academicYear, period, pointsAmount)

IsLocatedIn(roomID, buildingName)

Lecture(reservationID, courseCode, academicYear, period, actualStartTime, actualEndTime), this is a new class that replaces *LectureGroup* and represents a single lecture of a certain *CourseVersion*.

Exercise(reservationID, groupID, actualStartTime, actualEndTime), this is a new class that replaces the class *ReservationFor*. It describes an exercise session of a certain course.

Exam(reservationID, groupID, actualStartTime, actualEndTime)), this is a new class that replaces the class *ReservationFor*. It describes an exam event of a certain course.

3 The SQL model

3.1 Creating the tables

The data types used for our statements are TEXT, INTEGER and REAL. The majority of our attributes are ones that require the use of letters and therefore must be strings. These are all given the data type TEXT. All numeric attributes such as group sizes, number of seats in a room or exam grades all have the data type INTEGER. These all need to be represented as whole numbers so it is easier to constrict them into integers instead of reals. We also choose to use this data type for the *academicYear* attribute in our tables. The only numeric attribute with the data type REAL instead of INTEGER is *pointsAmount* in the *ExercisePoints* table. This is because we want to allow students to receive half a point etc. from their exercises. Since DATE is not a data type supported by SQLite we use TEXT for all dates in the database.

All of the attributes of the tables have been constrained by the NOT NULL constraint. We don't want any NULL values in the database since we find all the attributes to be important enough in the current form of the database that it is reasonable to demand values for all the attributes. For example, deleting an employee that has made reservations before would leave NULL values to the *employeeID* in the Reservation table. We could go around this by moving the *employeeID* to some other employee or simply prevent deleting employees from the database to have a more complete history of both reservations and employees on record.

We also have constraints on the numeric attributes. Most of these are made to ensure the values are positive or non-negative, like *pointsAmount* from *ExercisePoints* or the *academicYear* from *CourseVersion*. The exam grades are constricted to be integers

between zero and five as this is the scale used for grading. We have also limited the values of the examLanguage from table *ExamEnrollment* to be Finnish, Swedish or English as they are the only accepted language options for our exams.

The primary keys have been taken directly from the revised version of relational model of the database in Section 2.2. The primary keys are implemented in all of the create table statements.

The foreign keys have been well defined to increase the functionality of our database. The foreign keys reference the parent tables where data will be updated. An example of this could be the *CourseVersion* table which has the foreign key *courseCode* from the *Course* table. This helps us by ensuring a *courseCode* update for a course is reflected also in the different versions of that course. This also prevents from accidentally creating a *courseVersion* for a course that doesn't exist, i.e. a course which has no course code. Such cases have been considered for all attributes of all tables in our database.

The full list of all create table commands is presented below:

```
CREATE TABLE Student (
    studentID  TEXT NOT NULL,
    name       TEXT NOT NULL,
    dateOfBirth TEXT NOT NULL,
    program    TEXT NOT NULL,
    startDate  TEXT NOT NULL,
    endDate    TEXT NOT NULL,
    CHECK (startDate < endDate),
    PRIMARY KEY (studentID)
);

CREATE TABLE Course (
    courseCode TEXT   NOT NULL,
    credits   INTEGER NOT NULL CHECK (credits >= 0),
    courseName TEXT   NOT NULL,
    PRIMARY KEY (courseCode)
);

CREATE TABLE CourseVersion (
    courseCode  TEXT      NOT NULL,
    academicYear INTEGER  NOT NULL CHECK (academicYear > 0),
    period      TEXT      NOT NULL,
    FOREIGN KEY (courseCode) REFERENCES Course (courseCode),
```

```
    PRIMARY KEY (courseCode, academicYear, period)
);

CREATE TABLE ExamGroup (
    groupID    TEXT NOT NULL,
    courseCode TEXT NOT NULL,
    PRIMARY KEY (groupID),
    FOREIGN KEY (courseCode) REFERENCES Course (courseCode)
);

CREATE TABLE ExerciseGroup (
    groupID       TEXT    NOT NULL,
    courseCode    TEXT    NOT NULL,
    academicYear  INTEGER NOT NULL,
    period        TEXT    NOT NULL,
    groupSize     INTEGER NOT NULL CHECK (groupSize >= 0),
    PRIMARY KEY (groupID)
    FOREIGN KEY (courseCode, academicYear, period) REFERENCES CourseVersion
        (courseCode, academicYear, period)
);

CREATE TABLE Building (
    name    TEXT NOT NULL PRIMARY KEY,
    address TEXT NOT NULL
);

CREATE TABLE Room (
    roomID       TEXT     NOT NULL PRIMARY KEY,
    noOfSeats    INTEGER  NOT NULL CHECK (noOfSeats >= 0),
    noOfExamSeats INTEGER NOT NULL CHECK (noOfExamSeats >= 0)
);

CREATE TABLE Equipment (
    equipmentName TEXT    NOT NULL,
    roomID       TEXT     NOT NULL,
    count        INTEGER  NOT NULL CHECK (count > 0),
    FOREIGN KEY (roomID) REFERENCES Room (roomID),
    PRIMARY KEY (equipmentName, roomID)
);
```

```
CREATE TABLE Employee (
    employeeID TEXT NOT NULL,
    name       TEXT NOT NULL,
    position   TEXT NOT NULL,
    phone      TEXT NOT NULL,
    address    TEXT NOT NULL,
    startDate TEXT NOT NULL,
    endDate    TEXT NOT NULL,
    PRIMARY KEY (employeeID)
);

CREATE TABLE Reservation (
    reservationID    TEXT NOT NULL,
    reserverID       TEXT NOT NULL,
    roomID           TEXT NOT NULL,
    startTime        TEXT NOT NULL,
    endTime          TEXT NOT NULL,
    timeOfReservation TEXT NOT NULL,
    PRIMARY KEY (reservationID)
    FOREIGN KEY (reserverID) REFERENCES Employee (employeeID)
    FOREIGN KEY (roomID) REFERENCES Room (roomID)
);

CREATE TABLE CourseEnrollment (
    studentID  TEXT NOT NULL,
    groupID    TEXT NOT NULL,
    PRIMARY KEY (studentID, groupID)
    FOREIGN KEY (studentID) REFERENCES Student (studentID)
    FOREIGN KEY (groupID) REFERENCES ExerciseGroup (groupID)
);

CREATE TABLE ExamEnrollment (
    studentID   TEXT NOT NULL,
    groupID     TEXT NOT NULL,
    examLanguage TEXT NOT NULL CHECK (examLanguage IN ("Finnish", "Swedish",
        "English")),
    dateEnrolled TEXT NOT NULL,
    FOREIGN KEY (studentID) REFERENCES Student (studentID),
    FOREIGN KEY (groupID) REFERENCES ExamGroup (groupID),
    PRIMARY KEY (studentID, groupID)
```

```
);

CREATE TABLE ExamGrade (
    studentID  TEXT    NOT NULL,
    courseCode TEXT    NOT NULL,
    dateOfGrade TEXT   NOT NULL,
    grade      INTEGER NOT NULL CHECK (grade >= 0 AND grade <= 5),

    PRIMARY KEY (studentID, courseCode, dateOfGrade),
    FOREIGN KEY (studentID) REFERENCES Student (studentID),
    FOREIGN KEY (courseCode) REFERENCES Course (courseCode)
);

CREATE TABLE ExercisePoints (
    studentID      TEXT    NOT NULL,
    courseCode     TEXT    NOT NULL,
    academicYear   INTEGER NOT NULL,
    period         TEXT    NOT NULL,
    pointsAmount   REAL    NOT NULL CHECK (pointsAmount >= 0),
    PRIMARY KEY (studentID, courseCode, academicYear, period)
    FOREIGN KEY (studentID) REFERENCES Student (studentID)
    FOREIGN KEY (courseCode, academicYear, period) REFERENCES CourseVersion
        (courseCode, academicYear, period)
);

CREATE TABLE IsLocatedIn (
    roomID         TEXT NOT NULL,
    buildingName   TEXT NOT NULL,
    PRIMARY KEY (roomID)
    FOREIGN KEY (roomID) REFERENCES Room (roomID)
    FOREIGN KEY (buildingName) REFERENCES Building (name)
);

CREATE TABLE Lecture (
    reservationID TEXT    NOT NULL,
    courseCode     TEXT    NOT NULL,
    academicYear   INTEGER NOT NULL,
    period         TEXT    NOT NULL,
    actualStartTime TEXT  NOT NULL,
    actualEndTime TEXT    NOT NULL,
```

```
    PRIMARY KEY (reservationID)
    FOREIGN KEY (reservationID) REFERENCES Reservation (reservationID)
    FOREIGN KEY (courseCode, academicYear, period) REFERENCES CourseVersion
        (courseCode, academicYear, period)
);

CREATE TABLE Exercise (
    reservationID  TEXT NOT NULL,
    groupID        TEXT NOT NULL,
    actualStartTime TEXT NOT NULL,
    actualEndTime  TEXT NOT NULL,
    PRIMARY KEY (reservationID)
    FOREIGN KEY (reservationID) REFERENCES Reservation (reservationID)
    FOREIGN KEY (groupID) REFERENCES ExerciseGroup(groupID)
);

CREATE TABLE Exam (
    reservationID  TEXT NOT NULL,
    groupID        TEXT NOT NULL,
    actualStartTime TEXT NOT NULL,
    actualEndTime  TEXT NOT NULL,
    PRIMARY KEY (reservationID, groupID)
    FOREIGN KEY (reservationID) REFERENCES Reservation (reservationID)
    FOREIGN KEY (groupID) REFERENCES ExamGroup (groupID)
);
```

3.2 Indexing

Typical use cases and SQL queries for our database involve the parent tables *Student*, *Employee*, *Course*, and *Reservation*. Therefore they are also the most relevant classes for our database to index. Typical use cases can include adding tuples like students to the database, creating new *courseVersions*, compiling statistics like the number of registrations for a course or the number of students that passed. Reservations are also important as hosting lectures or exams require information about the availability of rooms. More specific use cases are presented in section 3.4.

The indices are created around the primary keys of the classes except for the *roomReservationIndex*. This index is created to help speed up queries related to finding reservations in a certain room and at a particular time. This kind of index is useful as there will only

be one reservation at a time in a particular room unless there are exams that can be held simultaneously. Still in this case there will only be a few exams at once at best. This index is created as the number of tuples in the reservation table will certainly be very large. The reservation table will be updated very often and two indices with three different attributes mean additional updates. This is a sacrifice we will have to make to help with queries related to reservations.

Related to the above, we do not want to add too many indices as they complicate updating. We will not add indices to classes that are less interesting or to which fewer queries are expected.

The SQL commands for creating the indices are below:

```
CREATE INDEX studentIndex ON student(studentID);
CREATE INDEX courseIndex ON course(courseCode);
CREATE INDEX versionIndex ON courseVersion(courseCode, academicYear, period);
CREATE INDEX reservationIndex ON reservation(reservationID);
CREATE INDEX roomReservationIndex ON reservation(roomID,startTime);
CREATE INDEX employeeIndex ON employee(employeeID);
```

3.3 Views for the database

The command below creates a view *BestGrades* which lists the best grade a student has passed a course with. This requires an exam grade that is above zero. All courses for all students who have passed at least one course have been listed. Such a view is useful in queries related to calculating the amount of students that have passed a course, calculating the GPA for students, calculating total credits for students etc.

The *EquipmentByRoom* view contains all equipment inserted into the database, the number of equipment, and their respective rooms. This can be useful in locating equipment or doing inventory.

The *ReservationsByBuilding* contains all reservations in every building and the time they are reserved. This is very useful to check the availability of rooms and checking which reservations are on-going at a specific time.

```
CREATE VIEW BestGrades AS
    SELECT name, Student.studentID, courseCode, MAX(grade) AS finalGrade
    FROM ExamGrade, Student
    WHERE Student.studentID = ExamGrade.studentID AND grade > 0
```

```
    GROUP BY courseCode, Student.studentID;

CREATE VIEW EquipmentByRoom AS
    SELECT Room.roomID, equipmentName, count
    FROM Equipment, Room
    WHERE Equipment.roomID = Room.roomID
    ORDER BY Room.roomID;

CREATE VIEW ReservationsByBuilding AS
    SELECT name, Reservation.roomID, startTime, endTime
    FROM Building, IsLocatedIn, Reservation
    WHERE name = buildingName
    AND IsLocatedIn.roomID = Reservation.roomID
    ORDER BY name
```

3.4 Examples of SQL queries for the database

We have listed 15 SQL queries below, that output useful data from the database. The queries include a brief comment describing the implementation of the query and some general applications. After each query there is a table containing the output of the query.

1. This query can be used to attain a list of grades for a certain student. The query below searches all the grades of the student "Teemu Teekkari". If the student has taken an exam multiple times, this query outputs the highest grade of all the exams.

```
-- *Returns the grades of a student by name*
-- It is assumed that no ExamGrade do not contain illegal tuples,
-- and the legality of the tuple is checked when they are added to ExamGrade.
-- This query picks the top grades of all courses from all attempts
-- of a single student.
SELECT courseCode, MAX(grade) AS finalGrade
FROM ExamGrade, Student
WHERE Student.name = "Teemu Teekkari" AND Student.studentID =
    ExamGrade.studentID
GROUP BY courseCode;
```

courseCode	finalGrade
CS-A1120	4
MS-C1342	5
NBE-C2102	5
PHYS-A0140	5
SCI-C0200	5

2. This query outputs a list of all exam results for a given course by name. This data can be further used to, for example, calculate the number of students who have passed a certain course. The below query retrieves this data for the course "Tietokannat".

```
-- *All exam results for a given course by course name*
-- Returns all exam results for a given course by its name.
SELECT ExamGrade.studentID, ExamGrade.grade
FROM ExamGrade, Course
WHERE ExamGrade.courseCode = Course.courseCode
AND Course.courseName = "Tietokannat"
ORDER BY grade DESC;
```

studentID	grade
302329292	4
987654322	1
639340498	0

3. This query filters the exam results of only the students who have passed the course "Tietokannat", and then outputs the number of those students. This query can also be used to search with other course names.

```
-- *The number of students who have passed a certain course*
-- Returns the number of students who have passed a certain course.
SELECT COUNT(grade) AS noOfPasses
FROM ExamGrade, Course
WHERE ExamGrade.courseCode = Course.courseCode
AND Course.courseName = "Tietokannat"
AND grade > 0;
```

noOfPasses
2

4. This query is similar to the last query, but now it outputs the number of students who have failed the course. These queries can be used to determine the percentage of passes and fails for a certain course, and further to assess the difficulty of that course.

```
-- *The number of students who have failed a certain course*
-- Returns the number of students who have failed a certain course.
SELECT COUNT(grade) AS noOfFails
FROM ExamGrade, Course
WHERE ExamGrade.courseCode = Course.courseCode
AND Course.courseName = "Tietokannat"
AND grade = 0;
```

noOfFails
1

5. The output of this query are the names, student IDs and grades for all students who have passed the course "Tietokannat". The result of this query can be used to determine the mean grade for the course in question, or for any other course.

```
-- *All students that have passed a certain course by course name*
-- Returns the name and student ID of all students who have passed
-- the specified course by grade 1 or higher.
SELECT name, Student.studentID, grade
FROM Student, Course, ExamGrade
WHERE Student.studentID = ExamGrade.studentID
AND ExamGrade.courseCode = Course.courseCode
AND Course.courseName = "Tietokannat"
AND grade > 0;
```

name	studentID	grade
Kimmo Kauppatieteilijä	987654322	1
Tiina Teekkari	302329292	4

6. This query lists all the courses completed by each student and the credits they receive from each course. This query can be further used to determine the amount of credits each student has completed.

```
-- *All completed courses with credits for all students*
-- Returns the name, student ID of all students, and the
-- course code and credits for all courses they have passed.
```

```
SELECT DISTINCT name, Student.studentID, Course.courseCode, credits
FROM Student, Course, ExamGrade
WHERE Student.studentID = ExamGrade.studentID
AND ExamGrade.courseCode = Course.courseCode
AND grade > 0;
```

name	studentID	courseCode	credits
Teemu Teekkari	123456789	MS-C1342	5
Teemu Teekkari	123456789	NBE-C2102	5
Teemu Teekkari	123456789	CS-A1120	5
Teemu Teekkari	123456789	SCI-C0200	10
Teemu Teekkari	123456789	PHYS-A0140	5
Kiia Kemisti	576855966	MS-C1342	5
Kiia Kemisti	576855966	NBE-C2102	5
Matti Meikäläinen	639340498	NBE-C2102	5
Kari Konelainen	987654321	PHYS-A0110	5
Kimmo Kauppatieteilijä	987654322	CS-A1150	5
Tiina Teekkari	302329292	CS-A1150	5

7. This query uses the previous query as a subquery and further calculates the sum of all credits from all courses each student has completed.

```
-- *Total amount of credits for each student*
-- Returns the name, student ID and the total amount of credits
-- for each student.
SELECT name, studentID, SUM(credits)
FROM (SELECT DISTINCT name, Student.studentID, Course.courseCode, credits
      FROM Student, Course, ExamGrade
      WHERE Student.studentID = ExamGrade.studentID
      AND ExamGrade.courseCode = Course.courseCode
      AND grade > 0)
GROUP BY studentID;
```

name	studentID	SUM(credits)
Teemu Teekkari	123456789	30
Tiina Teekkari	302329292	5
Kiia Kemisti	576855966	10
Matti Meikäläinen	639340498	5
Kari Konelainen	987654321	5
Kimmo Kauppatieteilijä	987654322	5

8. This query returns the GPA of each student. The query uses a subquery which first finds all the exam grades larger than 1, and groups the data with respect to the course code and student ID. The outer query then takes the average of these grades with respect to the student ID.

```
-- *Students' GPAs*
-- Returns the name, student ID and GPA of all students.
-- The GPA is calculated from the exam grades of
-- passed courses.
SELECT name, studentID, AVG(finalGrade) AS GPA
FROM (SELECT name, Student.studentID, courseCode, MAX(grade) AS finalGrade
      FROM ExamGrade, Student
      WHERE Student.studentID = ExamGrade.studentID AND grade > 0
      GROUP BY courseCode, Student.studentID)
GROUP BY studentID;
```

name	studentID	GPA
Teemu Teekkari	123456789	4.8
Tiina Teekkari	302329292	4
Kiia Kemisti	576855966	1.5
Matti Meikäläinen	639340498	3
Kari Konelainen	987654321	5
Kimmo Kauppatieteilijä	987654322	1

9. This query returns all information of the equipment the room with ID "U2", but can be used to find the equipment of other rooms. This information is relevant, for example, when making a reservation, since you might want to reserve a room that has precisely the equipment you desire.

```
-- *All equipment located in a certain room*
-- Returns the name and count of all equipment
-- located in a certain room by room ID.
SELECT equipmentName, count
FROM Equipment
WHERE roomID = 'U2';
```

equipmentName	count
Blackboard	4
Projector	1

10. This query returns all the rooms contained in the building "Kandidaattikeskus". The query can be used to find the rooms in any building on campus. This data is useful if, for example, you want to make a reservation to a certain building and thus want to find the reservable rooms in that building.

```
-- *All rooms in a certain building*
-- Returns the room ID for all rooms located
-- in a building by building name
SELECT roomID
FROM Building, Room, IsLocatedIn
WHERE Room.roomID = IsLocatedIn.roomID
AND buildingName = name
AND name = 'Kandidaattikeskus';
```

roomID
U2
M131
Y347
U202
A

11. This query returns all the reservations made for the room "U202", but can be generally used to find all the reservations made to any room. This data can be useful if, for example, you are interested in reserving a certain room and want to find all the active reservations for that room to see if it is available.

```
-- *The reservation history of a room*
-- Returns all reservations made to a certain room by the room ID.
SELECT reservationID, reserverID, startTime, endTime, timeOfReservation
FROM Reservation
WHERE roomID = 'U202';
```

reservationID	reserverID	startTime	endTime	timeOfReservation
123593	204585959	2023-05-09 11:30	2023-05-09 14:30	2023-05-08 18:19
123594	204585959	2023-05-10 11:30	2023-05-10 14:30	2023-05-08 18:20

12. This query returns the number of students who have enrolled in a certain course. This data is relevant for the organizers of the course.

```
-- *The number of students who have enrolled in a specific course*
```

```
-- Counts the number of students that have enrolled in a certain
-- CourseVersion.
SELECT COUNT(Student.studentID) AS studentCount
FROM CourseEnrollment, ExerciseGroup, Student
WHERE Student.studentID = CourseEnrollment.studentID
AND CourseEnrollment.groupID = ExerciseGroup.groupID
AND ExerciseGroup.courseCode = 'PHYS-A0130'
AND ExerciseGroup.academicYear = 2023
AND ExerciseGroup.period = 'III';
```

studentCount
2

13. This query returns all the exercise points for a certain course. This data is needed in grading.

```
-- *The exercise points for a certain course*
-- Returns all students' exercise points by a specific course version
SELECT name, Student.studentID, pointsAmount
FROM ExercisePoints, Student
WHERE Student.studentID = ExercisePoints.studentID
AND ExercisePoints.courseCode = 'CS-A1150'
AND ExercisePoints.academicYear = 2023
AND ExercisePoints.period = 'III-V';
```

name	studentID	pointsAmount
Teemu Teekkari	123456789	95
Tiina Teekkari	302329292	23

14. This query returns the information of all reservations made by a single employee. This data is relevant if, for example, you want to remove an employee from the database. In this case you may want to change the reservations from one employee to another.

```
-- *All reservations made by an employee*
-- Returns all the reservations made by an employee by their employee ID
SELECT reservationID, startTime, endTime, roomID, buildingName,
    Building.address
FROM Employee, Reservation, Building, IsLocatedIn
WHERE employeeID = '204585959'
```

```
AND employeeID = reserverID
AND Reservation.roomID = IsLocatedIn.roomID
AND buildingName = Building.name;
```

reservationID	startTime	endTime	roomID	buildingName	address
123593	2023-05-09 11:30	2023-05-09 14:30	U202	Kandidaattikeskus	Otakaari 1
123594	2023-05-10 11:30	2023-05-10 14:30	U202	Kandidaattikeskus	Otakaari 1
134564	2023-05-09 8:30	2023-05-09 12:30	Y347	Kandidaattikeskus	Otakaari 1
134212	2023-05-10 8:00	2023-05-10 10:15	U2	Kandidaattikeskus	Otakaari 1
134213	2023-05-11 10:00	2023-05-11 12:15	U2	Kandidaattikeskus	Otakaari 1

15. This query returns all the essential information of the lectures of a given course. This data can be used to, for example, construct schedules for students and teachers.

```
-- *All lecture times of a given course*
-- Returns the time and place for all lectures of a course.
SELECT actualStartTime, actualEndTime, Reservation.roomID, buildingName,
    address
FROM Lecture, Reservation, IsLocatedIn, Building
WHERE Lecture.reservationID = Reservation.reservationID
AND IsLocatedIn.roomID = Reservation.roomID
AND buildingName = Building.name
AND courseCode = 'MS-C1342' AND academicYear = 2022 AND period = 'IV';
```

actualStartTime	actualEndTime	roomID	buildingName	address
2023-05-10 8:15	2023-05-10 10:00	U2	Kandidaattikeskus	Otakaari 1
2023-05-11 10:15	2023-05-11 12:00	U2	Kandidaattikeskus	Otakaari 1