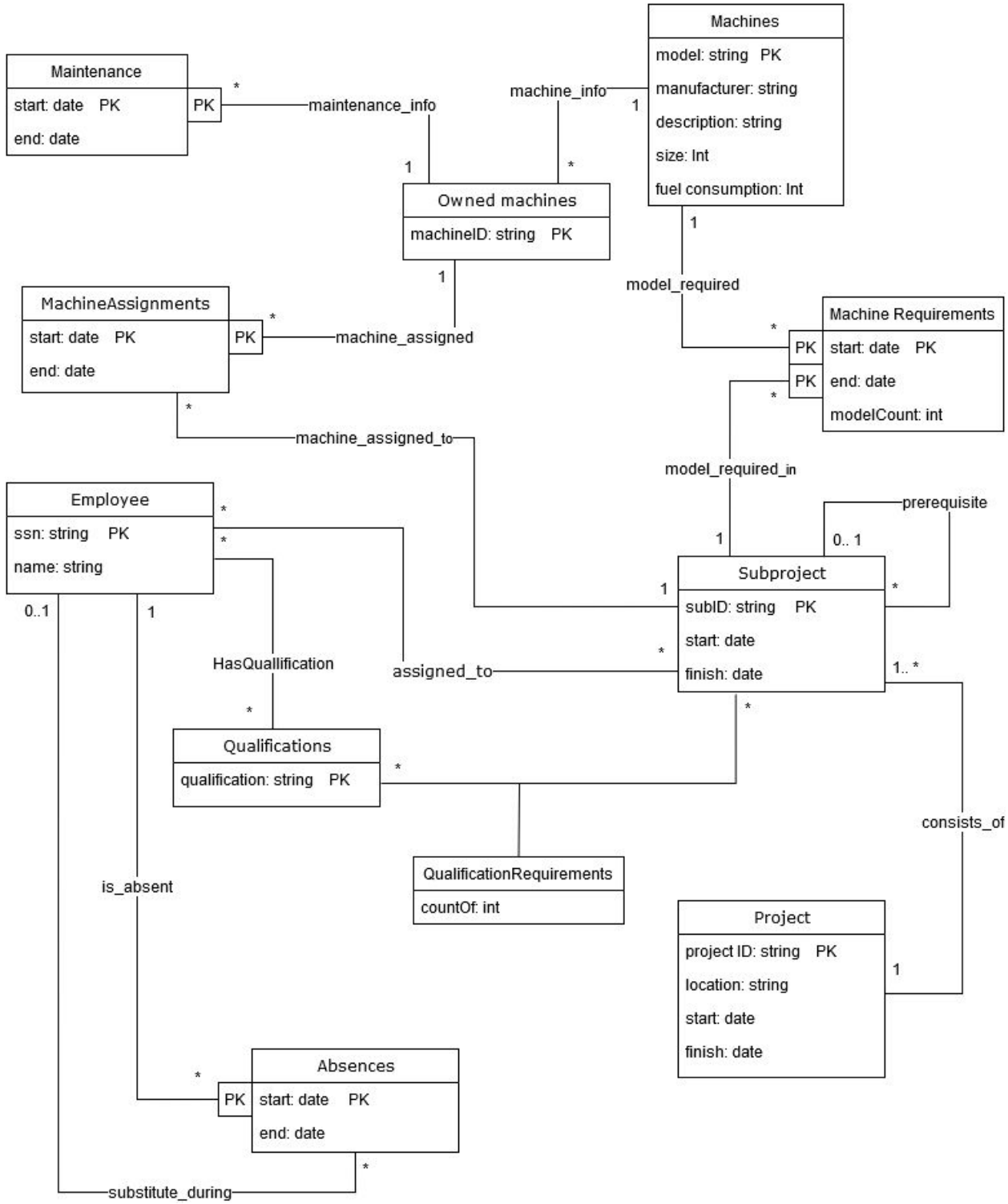
CS-A1150 Databases, Spring 2020 Project

Paul Toivonen, paul.toivonen@aalto.fi
Joel Hämäläinen, joel.hamalainen@aalto.fi
Amit Yadav, amit.yadav@aalto.fi

UML Diagram


Maintenance
start: date PK
end: date
PK
maintenance_info
Machines
model: string PK
manufacturer: string
description: string
size: Int
fuel consumption: Int
machine_info
Owned machines
machineID: string PK
model_required
MachineAssignments
start: date PK
end: date
machine_assigned
Machine Requirements
start: date PK
end: date
modelCount: int
machine_assigned_to
model_required_in
prerequisite
Employee
ssn: string PK
name: string
Subproject
subID: string PK
start: date
finish: date
HasQuallification
assigned_to
Qualifications
qualification: string PK
QualificationRequirements
countOf: int
consists_of
is_absent
Project
project ID: string PK
location: string
start: date
finish: date
Absences
start: date PK
end: date
substitute_during

Relational Data Model

The UML diagram as a relation model:

Project(projectid, location, start, finish)
Subproject(subID, prerequisiteID, start, finish, partOfProject)
Employee(ssn, name)
Qualifications(qualification)
HasQualification(ssn, qualification)
QualificationRequirements(qualification, subID, countOf)
Absences(absentSSN, start, end, subSSN)
EmployeeAssignments(ssn, subID)
Machines(model, manufacturer, description, size, fuelConsumption)
OwnedMachines(machineID, model)
Maintenance(machineID, start, end)
MachineAssignments(machineID, start, end, subID)
MachineRequirements(model, start, subID, end, modelCount)

Solution Explanation

Remarks:

1. The following text is structured according to the order of the relations listed in the relational data model.
2. All the referenced relations, and attributes of those relations are in *cursive* to enhance the readability and understandability of the text.

Project is a relation that stores information regarding the projects of the company. The projects have a starting date and an end date which make it easy to filter projects according to time periods.

The *Subproject* relation stores information about subprojects that the main projects consist of. The subprojects also have a start date and an end date, so in case the project gets delayed it is easy to make changes to the schedule.

The relation between subprojects and projects, *consists_of*, has been combined into *Subproject* so that one of its attributes, *partOfProject*, is the id of the main project it is part of. This simplifies the schema of the database. Also, the prerequisite of a

subproject has been combined with the relation *Subproject* by adding an attribute called *prerequisiteID,* which can also be NULL in case the subproject has no prerequisites. The relations can be combined since both of the earlier mentioned relations are one-to-many associations by their nature. To find all the projects that depend on finishing subproject X, one would have to select all the subprojects with the attribute *PrerequisiteID* being X, from the *Subproject* relation.

Qualifications is a set of all possible qualifications an employee may have/acquire. Another option would have been to represent each qualification using subclasses, however this would have complicated the schema in the case there are a variable number of qualifications.

HasQualification is a relation that stores information about the current qualifications of all employees. To add new qualifications to an existing employee, a new tuple would be added with the social security number of the employee and the new qualification in string format.

QualificationRequirements is a relation that stores the count, *countOf,* for each qualification required in a subproject. Only stores information about the required qualifications. Other qualifications are left out.

The relations *is_absent* and *substitute_during* have been combined with Absences so that we have two relations; Employee and Absences that store all the information regarding these relations. Absences are identified by the absent workers social security number as well as the starting date of the absence since a person can have multiple absences. *subSSN* stores the *SSN* of the substitute employee.

EmployeeAssignments stores the subprojectID to which a specific employee has been assigned to. Since an employee is assigned to a subproject for its whole duration, we don't need to store the start or end dates of assignment. With a many-to-many association, we can store the history of subproject employees.

Machines is a relation for storing information regarding the specifics of machine models that the company might use in construction.

The company owned specific machines are listed in the relation *OwnedMachines* that ties the machine to the model specific information by combining *machine_info* into it. The specific machines are identified by *MachineID* and are instances of a machine model.

By having *Machines* and *OwnedMachines* as separate relations we are able to add totally new models and new instances of existing models to the database. To add a new model, one would add a tuple to *Machines*. To add a new item one would have to add a new tuple to *OwnedMachines* and check if the model already exists in *Machines*. In case no instance of the model is found, a new tuple would be added to *Machines* as well, regarding the model of the acquired machine.

Maintenance lists maintenance performed/scheduled on all the different company owned machines. The machine under maintenance is considered unavailable for work.

The same machine will be assigned to multiple subprojects in its lifetime so a separate relation, *MachineAssignments*, is needed for storing information about the time frame that a machine is assigned to a project. *MachineAssignments* is machine (item) specific and stores the start and end dates of the assignment so that it's easy to find the subproject that the machine was assigned to at a given point in time.

To find out if a certain machine is free at a given time, one would have to cross check the *OwnedMachines* with the maintenance list and the machine assignments list over the given time period and minus the machines that are already assigned or under maintenance.

MachineRequirements stores information about the specific models of machinery that are needed during a subproject. It stores the model of the machine, start date, end date, the ID of the subproject and the amount of needed machines of the certain model(*modelCount*).

Functional dependencies and BCNF

The database has the following functional dependencies, for each relation:

Project(projectID, location, start, finish)
- (projectid → location, start, finish)

Subproject(subID, prerequisiteID, start, finish, partOfProject)
- (subID → prerequisiteID, start, finish, partOfProject)

Employee(ssn, name)
- (ssn → name)

Qualifications(qualification)
- None

HasQualification(ssn, qualification)

- None

QualificationRequirements(qualification, subID, countOf)

- (qualification, subID → countOf)

Absences(absentSSN, start, end, subSSN)

- (absentSSN, start → end, subSSN)

EmployeeAssignments(ssn, subID)

- None

Machines(model, manufacturer, description, size, fuelConsumption)

- (model → manufacturer, description, size, fuelConsumption)

OwnedMachines(machineID, model)

- (machineID → model)

Maintenance(machineID, start, end)

- (machineID, start → end)

MachineAssignments(machineID, start, end, subID)

- (machineID, subID → start, end)
- (machineID, start → end, subID)

ModelRequirements(model, start, subID, end, modelCount)

- (model, subID, start → end, modelCount)

As we can see above, all left sides of functional dependencies are keys. So the given structure is in Boyce-Codd Normal Form.

Creating the database in SQL

The database can be created in SQL using the following SQLite commands. The results of these commands after being run in SQLiteStudio are given after each one. In general, the relational model has been converted into SQL commands with the appropriate data types. The primary keys are the same as in the relational model, and the foreign keys are defined when an attribute references another primary key.
NOTE: Copy-pasting commands with quotes can cause problems. Retype quotes if this happens.

```
CREATE TABLE Projects (
        projectID TEXT PRIMARY KEY,
        location TEXT,
        start TEXT,
        finish TEXT
);
```

Result:
OK: Query finished in 0.034 second(s).

Reasoning:
The project ID is given in text to give more flexibility, and the location is the place name in text. SQLite handles dates as a text data type. The location and dates can be NULL, if they are not yet known. The dates are not compared, because comparison with NULL could lead to unexpected behaviour.

```
CREATE TABLE Subprojects (
        subID TEXT PRIMARY KEY,
        prerequisiteID TEXT REFERENCES Subprojects(subID),
        start TEXT,
        finish TEXT,
        partOfProject TEXT REFERENCES Projects(projectID)
);
```

Result:
OK: Query finished in 0.029 second(s).

Reasoning:
IDs and dates are text because of the same reasons as before. Because the subproject is part of a single project, the project it is part of is a foreign key. Same applies for the prerequisite project, which can also be NULL, when there is no prerequisite subproject for this subproject. Because the history of projects and subprojects needs to be kept, possibly also for other programs the IDs cannot be changed. Thus the default block policy on update and delete is appropriate.

```
CREATE TABLE Employees (
        ssn TEXT PRIMARY KEY,
        name TEXT NOT NULL
);
```

Result:
OK: Query finished in 0.030 second(s).

Reasoning:
The social security number can also contain letters, so it is a text type. An employee must have a name saved.

```
CREATE TABLE Qualifications (
        qualification TEXT PRIMARY KEY
);
```

Result:
OK: Query finished in 0.027 second(s).

Reasoning:
The qualification is the written name of a possible qualification.

```
CREATE TABLE HasQualification (
        ssn TEXT REFERENCES Employees(ssn),
        qualification TEXT REFERENCES Qualifications(qualification),
        PRIMARY KEY (ssn, qualification)
);
```

Result:
OK: Query finished in 0.024 second(s).
Reasoning:
HasQualification is a many- to many-relation, so the primary keys of both parts must be foreign keys. Together they form the primary key for this table. The data types are the same as the parents'.

```
CREATE TABLE QualificationRequirements (
        qualification TEXT REFERENCES Qualifications(qualification),
        subID TEXT REFERENCES Subprojects(subID),
        countOf INTEGER,
        PRIMARY KEY (qualification, subID)
);
```

Result:
OK: Query finished in 0.029 second(s).

Reasoning:
This is the same situation as before, but it also has the count of the required employees as an integer.

```
CREATE TABLE Absences (
        absentSSN TEXT REFERENCES Employees(ssn),
        start TEXT,
        end TEXT NOT NULL,
        subSSN TEXT REFERENCES Employees(ssn),
        PRIMARY KEY (AbsentSSN, start)
);
```

Result:
OK: Query finished in 0.028 second(s).

Reasoning:
The absentee and the substitute must exist, so they are foreign keys and thus they take their data types. Dates are once again in text form. The end date must be known, so we can know when employees are available. The primary key consists of both the absentee and the start date, so the history of them can be recorded.

```
CREATE TABLE EmployeeAssignments (
        ssn TEXT REFERENCES Employees(ssn),
        subID TEXT REFERENCES Subprojects(subID),
        PRIMARY KEY (ssn, subID)
);
```

Result:
OK: Query finished in 0.025 second(s).

Reasoning:
This is once again a many- to many-relation. The foreign keys are also primary keys, and their data type is inherited.

```
CREATE TABLE Machines (
        model TEXT PRIMARY KEY,
        manufacturer TEXT,
        description TEXT,
        size INTEGER,
        fuelConsumption INTEGER
);
```

Result:
OK: Query finished in 0.030 second(s).

Reasoning:
The identifying model ID is text to give more flexibility. Machine info is of the relevant data type.

```
CREATE TABLE OwnedMachines (
        machineID TEXT PRIMARY KEY,
        model TEXT REFERENCES Machines(model)
);
```

Result:
OK: Query finished in 0.029 second(s).

Reasoning:
The individual machine ID is also given as text to give more flexibility. The model of this individual machine is located in the Machines table, so it is a foreign key.

```
CREATE TABLE Maintenance (
        machineID TEXT REFERENCES OwnedMachines(machineID),
        start TEXT,
        end TEXT,
        PRIMARY KEY (machineID, start)
);
```

Result:
OK: Query finished in 0.031 second(s).

Reasoning:
Machine ID is the individual machine from that table. The data type is inherited. Dates are again as text. Because of the history information, the identifying information is the machine and maintenance start date.

```
CREATE TABLE MachineAssignments (
        machineID TEXT REFERENCES OwnedMachines(machineID),
        start TEXT,
        end TEXT,
        subID TEXT REFERENCES Subprojects(subID),
        PRIMARY KEY(machineID, start)
);
```

Result:
OK: Query finished in 0.029 second(s).

Reasoning:
The assigned machine is from the machine table, and the subproject it is part of is from the subproject table as foreign keys and the inherited data type. Dates are in text format. The primary key includes both the machine ID and the start date, because one machine can be used many times in one subproject.

```
CREATE TABLE MachineRequirements (
        model REFERENCES Machines(model),
        start TEXT,
        subID TEXT REFERENCES Subprojects(subID),
        end TEXT,
        modelCount INTEGER,
        PRIMARY KEY (model, start, subID)
);
```

Result:
OK: Query finished in 0.026 second(s).

Reasoning:
The required models are from the model table, and the subproject it is needed in from the subproject table. Thus, they are foreign keys and their data type is the same as theirs. Dates as text, and the number required as an integer. The primary key includes the model, start date and subproject, because one subproject can require the same model many times and many models at the same time.

Typical queries, indexing and views

Typical queries in this database could be some of the following:

- Finding out basic information of employees, projects, machines
- Finding out all active projects or subprojects, or by start date
- Finding projects or subprojects that have finished during a certain time period
- Finding who has worked in a certain subproject or project
- Finding what projects an employee has been part of
- Finding which machines are available at a certain time
- Finding employees with a qualification who are available at a certain time
- Finding the models and the count of them used in a subproject
- Finding which subprojects require a specific subproject to be completed
- Finding out which subprojects are missing employees and which qualifications

Usability of indexing by typical query:

Basic information:
CREATE INDEX ProjectIndex ON Projects(projectID);
CREATE INDEX SubprojectIndex ON Subprojects(subID);
CREATE INDEX EmployeeIndex ON Employees(ssn);
CREATE INDEX MachineIndex ON Machines(model);
CREATE INDEX OwnedIndex ON OwnedMachines(machineID);

Result:
OK: Query finished in 0.075 second(s).

Reasoning:
To find one single tuple, it is efficient to use an index to find a unique primary key. Because assignments of machines and employees are added frequently, it is not efficient to use an index, because it would always update both the index and table.

Project and subproject by start or end date:
Because there can be many projects and subprojects at the same time, and they can be unclustered and distributed randomly, even with an index almost all disk pages must be retrieved.

Employees in a subproject or project:
CREATE INDEX EmployeeWorkedAt ON EmployeeAssignments(ssn);

Result:
OK: Query finished in 0.016 second(s).

Reasoning:
Because employees are assigned to one subproject at a time, there are fairly few tuples for one employee. If there are many searches, and not too many inserts an index is efficient.

Machines available by date:
Dates can once again be randomly distributed, so all disk pages must be retrieved, and an index is not efficient.

Employees with qualification at a time:
There can be many employees with the desired qualification, and they can be randomly distributed. The same is true for the subproject times. Thus, an index is not efficient.

Models used in a subproject:
Because the required machines are not clustered, they can once again be randomly distributed, and an index would not be efficient.

Subproject prerequisites:
Because many different subprojects at different locations can require the desired subproject, an index would not be efficient.

Missing employees at a subproject:
Assignments and requirements can be randomly distributed.

One possible view would be one that gives all available employees right now. It can be created by using the following commands:

```
CREATE VIEW AvailableEmployees AS
        SELECT Employees.ssn, Employees.name, HasQualification.qualification
        FROM Employees, HasQualification
        WHERE Employees.ssn = HasQualification.ssn
        EXCEPT
        SELECT Employees.ssn, Employees.name, HasQualification.qualification
        FROM Employees, HasQualification, EmployeeAssignments, Absences
        WHERE Employees.ssn = HasQualification.ssn AND Employees.ssn =
                (SELECT EmployeeAssignments.ssn
                FROM EmployeeAssignments, Subprojects
                WHERE EmployeeAssignments.subID = Subprojects.subID AND start <=
                date('now') AND finish >= date('now')
                ) OR Employees.ssn =
                        (SELECT absentSSN
                        FROM Absences
                        WHERE start <= date('now') AND end >= date('now')
                        );
```

Use cases

1. Start database

Insert projects, machines and employees, assign them and get results.

```
INSERT INTO Projects
VALUES ('PR01','Espoo', '2019-03-09', '2020-05-24');

INSERT INTO Subprojects
VALUES
        ('SPR01-01', NULL, '2019-03-09', '2019-04-20', 'PR01'),
        ('SPR01-02', 'SPR01-01', '2019-04-20', '2020-05-24', 'PR01');

INSERT INTO Employees
VALUES
        ('100584-163A', 'Lauri Loronen'),
        ('210264-14H', 'Pasi Sorjonen'),
        ('240383-107M', 'Pasi Rantanen'),
        ('190892-113I', 'Jaakko Virtanen'),
        ('040686-034K', 'Vadim Oblist'),
        ('081091-141L', 'Joona Silanen');

INSERT INTO Qualifications
VALUES
        ('designer'),
        ('supervisor'),
        ('worker'),
        ('caster'),
        ('carpenter');

INSERT INTO HasQualification
VALUES
        ('100584-163A', 'designer'),
        ('210264-14H', 'supervisor'),
        ('240383-107M', 'worker'),
        ('190892-113I', 'worker'),
        ('040686-034K', 'worker'),
        ('081091-141L', 'worker'),
```

```
        ('081091-141L', 'caster');

INSERT INTO QualificationRequirements
VALUES
        ('designer', 'SPR01-01', 1),
        ('supervisor', 'SPR01-01', 1),
        ('worker', 'SPR01-01', '3');

INSERT INTO Absences
VALUES ('240383-107M', '2019-03-12', '2019-03-15', '040686-034K');

INSERT INTO EmployeeAssignments
VALUES
        ('100584-163A', 'SPR01-01'),
        ('210264-14H', 'SPR01-01'),
        ('240383-107M', 'SPR01-01'),
        ('190892-113I', 'SPR01-01'),
        ('081091-141L', 'SPR01-01');

INSERT INTO Machines
VALUES
        ('PK-130-F', 'BOSCH', 'Power drill', 130, 55),
        ('RD180-M', 'Faust', 'Grinder', 180, 40),
        ('JD Devastator 250', 'John Deer', 'Bulldozer', 250, 135);

INSERT INTO OwnedMachines
VALUES
        ('PD01', 'PK-130-F'),
        ('PD02', 'PK-130-F'),
        ('BD01', 'JD Devastator 250');

INSERT INTO Maintenance
VALUES ('PD01', '2018-07-15', '2018-07-17');

INSERT INTO MachineAssignments
VALUES
        ('PD01', '2019-04-20', '2019-04-30', 'SPR01-02'),
        ('PD02', '2019-04-30', '2019-05-15', 'SPR01-02'),
        ('BD01', '2019-03-09', '2019-04-10', 'SPR01-01');
```

```
INSERT INTO MachineRequirements
VALUES
        ('JD Devastator 250', '2019-03-09', 'SPR01-01', '2019-04-10', 1),
        ('PK-130-F', '2019-04-20', 'SPR01-02', '2019-04-30', '1'),
        ('PK-130-F', '2019-04-30', 'SPR01-02', '2019-05-15', '1');
```

Result:
OK: Query finished in 0.251 second(s). Rows affected: 43

```
SELECT * FROM Employees;
```

Result:
OK: Query finished in 0.004 second(s).

ssn	name
100584-163A	Lauri Loronen
210264-14H	Pasi Sorjonen
240383-107M	Pasi Rantanen
190892-113I	Jaakko Virtanen
040686-034K	Vadim Oblist
081091-141L	Joona Silanen

2. Find out the locations of projects

Get all locations of projects and print them.

```
SELECT DISTINCT location FROM Projects;
```

Result:
OK: Query finished in 0.002 second(s).
location
Espoo

3. Find ongoing subprojects

Get the id of all subprojects that are currently active.

```
SELECT subID FROM Subprojects
WHERE START <= DATE("now") AND finish >= DATE("now");
```

Result:
OK: Query finished in 0.002 second(s).
subID
SPR01-02

4. List subprojects of project

List all subprojects and their dates of project PR01, sorted by starting date.

```
SELECT subID, start, finish FROM Subprojects
WHERE partOfProject = "PR01" ORDER BY start;
```

Result:
OK: Query finished in 0.001 second(s).

subID	start	finish
SPR01-01	2019-03-09	2019-04-20
SPR01-02	2019-04-20	2020-05-24

5. Finding subprojects that have finished in a certain time period

Find out finished subprojects between dates 2019-02-01 and 2019-10-01, ordered by start date.

```
SELECT subID, start, finish FROM Subprojects
WHERE start >= “2019-02-01” AND finish <= “2019-10-01” ORDER BY start;
```

Result:
OK: Query finished in 0.009 second(s).

subID	start	finish
SPR01-01	2019-03-09	2019-04-20

6. Subprojects requiring a certain subproject to be completed

Find out which subprojects require subproject SPR01-01 to be completed.

```
SELECT subID FROM Subprojects WHERE prerequisiteID = "SPR01-01";
```

Result:
OK: Query finished in 0.000 second(s).

subID
SPR01-02

7. List all employees working on a subproject

List all employees working on subproject SPR01-01.

```
SELECT ssn, name FROM Employees, EmployeeAssignments
WHERE EmployeeAssignments.ssn = Employees.ssn AND subID = "SPR01-01";
```

Result:
Query finished in 0.002 second(s).

ssn	name
100584-163A	Lauri Loronen
210264-14H	Pasi Sorjonen
240383-107M	Pasi Rantanen
190892-113I	Jaakko Virtanen
081091-141L	Joona Silanen

8. List all machines assigned to a subproject

Find out all machines and their info, assigned to subproject SPR01-01.

```
SELECT OwnedMachines.machineID, model, start, end
FROM MachineAssignments, OwnedMachines
WHERE OwnedMachines.machineID = MachineAssignments.machineID
AND subID = "SPR01-01";
```

Result:
OK: Query finished in 0.002 second(s).

machineID	model	start	end
BD01	JD Devastator 250	2019-03-09	2019-04-10

9. Absences and substitutes of an employee

Find out all absences and substitutes for these absences for employee Pasi Rantanen (240383-107M).

```
SELECT start, end, name FROM Absences, Employees
WHERE absentSSN = “240383-107M” AND subSSN = ssn;
```

Result:

```
OK: Query finished in 0.000 second(s).
start       end         name
2019-03-12  2019-03-15  Vadim Oblist
```

10. Projects an employee has been part of

Find out all projects employee Pasi Sorjonen (210264-14H) has been part of.

```
SELECT partOfProject FROM Subprojects, EmployeeAssignments
WHERE EmployeeAssignments.subID = Subprojects.subID AND ssn = “210264-14H”;
```

Result:

```
OK: Query finished in 0.002 second(s).
partOfProject
PR01
```

11. Machines available during certain time

Find out all machines that are available in 2019-04-25.

```
SELECT machineID, model
FROM OwnedMachines WHERE machineID NOT IN (
        SELECT OwnedMachines.machineID
        FROM OwnedMachines, MachineAssignments, Maintenance
        WHERE OwnedMachines.machineID = MachineAssignments.machineID
        AND OwnedMachines.machineID = Maintenance.machineID
        AND (
                MachineAssignments.start < "2019-04-25"
                OR Maintenance.start < "2019-04-25"
        )
        AND (
        MachineAssignments.end > "2019-04-25"
        OR Maintenance.end > "2019-04-25"
        )
);
```

Result:
OK: Query finished in 0.001 second(s).

machineID	model
PD02	PK-130-F
BD01	JD Devastator 250

12. Available employees with qualification at a certain time

Find out which employees, having qualification worker are available during 2019-03-10.

```
SELECT ssn, name, qualification FROM Employees, HasQualification
WHERE Employees.ssn = HasQualification.ssn
AND qualification = "worker"
AND Employees.ssn NOT IN (
        SELECT ssn FROM EmployeeAssignments, Subprojects
        WHERE EmployeeAssignments.subID = Subprojects.subID
        AND start <= "2019-03-10" AND finish >= "2019-03-10"
)
AND Employees.ssn NOT IN (
        SELECT absentSSN FROM Absences
        WHERE start <= "2019-03-10" AND end >= "2019-03-10"
);
```

Result:
OK: Query finished in 0.002 second(s).

ssn	name	qualification
040686-034K	Vadim Oblist	worker

13. Find assigned qualifications along with ‘how many’ for a subproject

Find the qualifications and how many of them are assigned for project ‘SPR01-01’.

```
SELECT qualification, count(*) FROM HasQualification
WHERE ssn in
        (SELECT ssn FROM EmployeeAssignments WHERE subID=’SPR01-01’)
GROUP BY qualification;
```

Result:
OK: Query finished in 0.000 second(s).

qualification	count(*)
caster	1
designer	1
supervisor	1
worker	3

14. Find required qualifications along with ‘how many’ for a subproject

Find the qualifications and the count of them required in subproject ‘SPR01-01’.

```
SELECT qualification, countOf FROM QualificationRequirements
WHERE subID='SPR01-01' ;
```

Result:
OK: Query finished in 0.002 second(s).

qualification	countOf
designer	1
supervisor	1
worker	3

15. Checking data integrity

Try to insert and update incorrect data. This should give an error. Update correct data and get results.

1. INSERT INTO HasQualification
   Values
   ('100000-000A', 'carpenter');
   Result:
   FOREIGN KEY constraint failed
2. INSERT INTO OwnedMachines
   Values
   ('PD03', 'PK-000-A');
   Result:
   FOREIGN KEY constraint failed

3. UPDATE Subprojects
   SET partOfProject = "NEWPROJECT"
   WHERE subID = "SPR01-01";
   Result: Error: FOREIGN KEY constraint failed
4. UPDATE Absences
   SET end = NULL
   WHERE absentSSN = "240383-107M"
   AND start = "2019-03-12";
   Result: Error: NOT NULL constraint failed: Absences.end
5. UPDATE Absences
   SET end = "2019-04-01"
   WHERE absentSSN = "240383-107M"
   AND start = "2019-03-12";
   Result: OK: Query finished in 0.014 second(s). Rows affected: 1

16. Find when a machine was last used and in which subproject and project.

```
SELECT min(end), subID, partOfproject
FROM MachineAssignments as M, Subprojects as S
WHERE machineID='PD01' and M.subID=S.subID
```

Result:
OK: Query finished in 0.001 second(s).

min(“end”)	subID	partOfproject
2019-04-30	SPR01-02	PR01