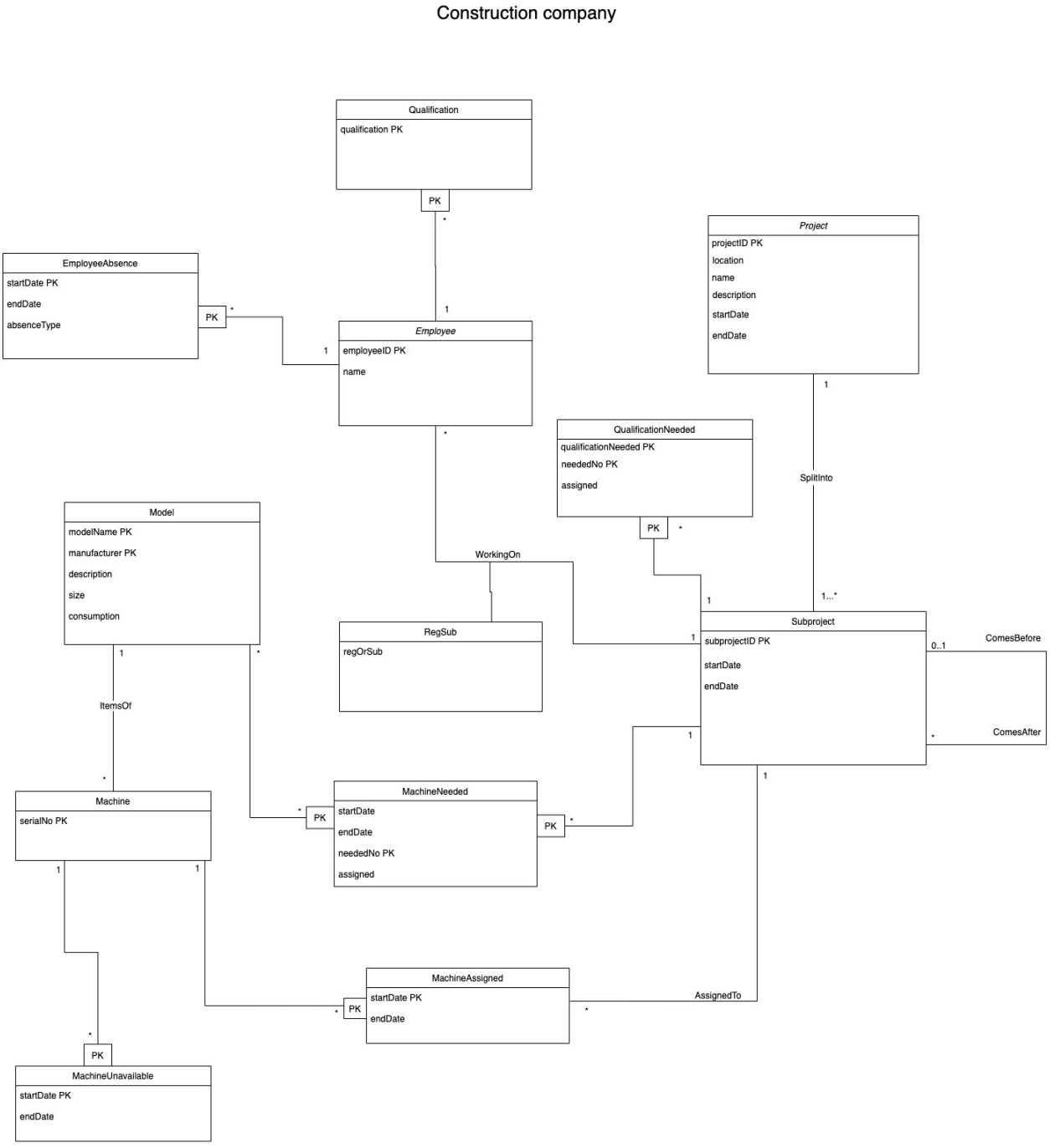
Databases
Project part 1

Construction company

UML



Relational data model

Projects(projectID, location, name, description, startDate, endDate)
Subprojects(subprojectID, projectID, startDate, endDate)
Dependency(subprojectID, comesAfterID)

Employees(employeeID, name)
WorkingOn(employeeID, subprojectID, regOrSub)
EmployeeAbsences(employeeID, startDate, endDate, absenceType)

Qualifications(qualification, employeeID)
QualificationsNeeded(neededNo, qualification, subprojectID, assigned)

Models(modelName, manufacturer, description, size, consumption)
Machines(serialNo, modelName, manufacturer)
MachineUnavailable(serialNo, startDate, endDate)

MachineNeeded (model, subprojectID, neededNo, assigned, startDate, endDate)
MachinesAssigned(serialNo, startDate, endDate, subprojectID)

Explanation of the solution

All operations in section 3 can be performed using this amazing database system.

We can insert new projects, subprojects and employees into their namesake relations.

Past, present and future projects can be stored in our database, and each is given its own projectID to differentiate the projects. Because we are using projectIDs, we can store projects with the same location, name and dates into the system without having any problems. (We assume 2^64 bits will be enough to store all projectIDs for the time being.) Past project information is retrieved very easily. Project and Subproject have an association SplitInto which describes which projects the subprojects are a part of.

The temporal order of the subproject is stored using the Dependency relation. From there we can retrieve the projects whose start depends on this project, and it is possible to store information about multiple subprojects being dependant on one subprojects, but it's not possible to have one subproject depend on two other subprojects. These are always a part of the same project.

Each employee can have many qualifications, which can be stored in the Qualifications relation. We thought about creating a separate relation for the names of the qualifications, in which case the names of the qualifications would have been kept even though the name would have been deleted from the Qualification relation. But because we didn't need to store anything else about the qualifications, we decided against it. It would also have forced us to store information twice. (In the qualifications relation and in the relations with the separate names.) We still decided to keep it as its own class instead of an attribute to Employee, because since an Employee can have several qualifications, this would have caused a redundancy.

When an employee needs to be assigned to a subproject, their employeeID is inserted to the WorkingOn association together with the subprojectID. Also, the knowledge about if they're assigned as a regular worker or as a substitute to a regular worker is inserted to the same relation. The employee is assigned for the same dates as the subproject is going on.

If a model is to be reserved for a subproject, the name of the model, the manufacturer, the subproject and the date of when a machine is needed is inserted into the machinesNeeded relation. If a free machine is found, the assigned attribute is yes, if no free machines are found, it gets the value no. This way we can easily find the models that are still needed for the project.

A machine can also be assigned to a subproject directly, without having to reserve a model. In that case, the machineID is inserted into the machinesAssigned relation. We can also find information about previously assigned machines. We thought it was natural for machines to only work at one project at a time. We implemented the system according to that, with the machine and startDate as primary key.

MachineUnavailable is for storing information about periods when we know the machine will be unavailable, Eg. for maintenance. The machine and startDate are the primary keys, since one machine can have several maintenances, but not in the same day.

Because the primary key of subprojects is not dependent on the start or end dates, we can easily change the dates. Also, the dependency isn't dependent on the dates. The only problem will be, that the dates have to be changed in many different places, Eg. in other projects, for reserved machines and employeeAbsence would have to be checked. But that is a problem that can't be fixed. Either way, the users would have to discuss machine reservations and employee absences, before being certain that the database can be updated on all aspects.

The absences are stored in the EmployeeAbsences relation, which has startDate and Employee as the primary key, to make it possible for an employee to have several absences. It also has the additional information endDate, and the type of absence (sick leave, vacation) the employee is taking.

The information about if all employees and machines have been assigned has been stored in the QualificationsNeeded and MachinesNeeded. This way we can easily find those models or qualifications who may not have an employee or a machine assigned to them yet. There we can also store how many machines/workers with some qualifications are needed for this project. This is stored in the attribute neededNo. If we need 3 machines, each one will be added to the relation with numbers 1, 2 and 3 respectively. At first we considered storing all the qualifications in one place, but this turned out much more complicated than this solution. Since qualification only has one attribute, it is simpler to have a separate relation to Employee, where an employee can have several qualifications, and a separate one for qualifications that a subproject needs, since those may not exist among the employees. It does not matter either, if eg. the only electrician is deleted, since no additional information is lost. Because of these reasons we eventually decided on this solution.

For receiving information about free employees, the data has to be combined from the relations, Employees, EmployeeAbsences (to get the absences) and WorkingOn (to get the info about who's already assigned to a project). When we take the difference between all employees and the employees that are in either EmployeeAbsences or WorkingOn or both, we receive the available (free) employees. Using set operations, freeEmployees = Employees \ (EmployeeAbsences ∪ Employs), were Employees, EmployeeAbsences and WorkingOn represent the EmployeeID attribute of those relations.

For finding free machines, the solution is very similar. We get the information about when machines are unavailable from MachineUnavailable, when machines are under maintenance, and from MachineAssigned, when the machine is needed in a project.

The WorkingOn relation is our only association class. We needed to store the information about if the employees act as regular workers or substitutes in a certain project, so we created a class for that purpose. We realized that the class didn't need any unique component of a primary key

in addition to the primary key created by employees (employeeID) and subproject (subprojectID), so we were able to make it into an association class.

We considered giving Machine and Subproject an identifier that was only unique within the model/project, but decided against it, to make it simpler. Instead we explain the relationship between Machine and Model, and Subproject and Project, with an association. Now they both instead have an ID that is unique within all the models and projects.

At first, we tried to manage with as few classes as possible to not make the UML too complicated. I believe it was a good idea because it enabled us to see the project as a whole. There were some small pieces of information that couldn't be stored in the existing classes, but just by creating more relations, we were able to store that information. It seemed like creating more classes/relations usually solves those problems.

There were quite a lot of problems with some technicalities of the UML design, like where the PK:s were supposed to be positioned and how associations are made into relations. Our UML was quite messy at first, but after clearing up some of those problems, the UML became much clearer.

Functional dependencies:

employeeID → name
employeeID absenceStartDate → absenceEndDate absenceType
qualification employeeID →
projectID → location name description startDate endDate
subprojectID → startDate endDate projectID
comesAfterID → comesBeforeID
qualificationNeeded neededNo subprojectID → assigned
employeeID subprojectID → regOrSub
modelName manufacturer → description size consumption
serialNo → modelName manufacturer
serialNo unavailableStartDate → unavailableEndDate
serialNo assignedStartDate → assignedEndDate subprojectID
modelName manufacturer subprojectID neededNo → neededStartDate neededEndDate assigned

Anomalies and redundancy

In the Model table, we have the attribute manufacturer. Multiple models can have the same manufacturer, which causes a redundancy. We decided to not make a separate table for the manufacturer since it has no additional information.

BCNF

Projects(<u>projectID</u>, location, name, description, startDate, endDate)
The only and non-trivial dependency present inProjects:
projectID → location name description startDate endDate
$\{projectID\}^+ = \{projectID,location,name,description,startDate,endDate\}$
The closure of projectID contains all attributes. Projects is already in BCNF.

Subprojects(<u>subprojectID</u>, projectID, startDate, endDate)
Non-trivial dependencies that are present in subprojects:
subprojectID → startDate endDate projectID
$\{subprojectID\}^+ = \{subprojectID, subStartDate, subEndDate, projectID\}$
$\{subprojectID\}^+$ Contains all attributes of subprojects and there are no other non-trivial dependencies. Subprojects is in BCNF.

Dependency(<u>subprojectID</u>, <u>comesAfterID</u>)
There are no non-trivial dependencies. Dependency is in BCNF

Employees(<u>employeeID</u>, name)
employeeID → name
$\{employeeID\}^+ = \{employeeID, name\}$
$\{employeeID\}^+$ Contains all the attributes of Employees. There are no other non-trivial dependencies. Employees is in BCNF

Employs(<u>employeeID,</u> <u>subprojectID</u>, regOrSub)
employeeID subprojectID → regOrSub
$\{employeeID\ subprojectID\}^+ = \{employeeID, subprojectID, regOrSub\}$
$\{employeeID\ subprojectID\}^+$ Contains all attributes of Employs. There are no other non-trivial dependencies. Employs is in BCNF

EmployeeAbsences(<u>employeeID</u>, <u>startDate</u>, endDate, absenceType)
employeeID startDate → endDate absenceType
$\{employeeID\ startDate\}^+ = \{employeeID, startDate, endDate, absenceType\}$
$\{employeeID\ startDate\}^+$ contains all attributes of EmployeeAbsences. There are no other non-trivial dependencies. EmployeeAbsences is in BCNF

Qualifications(<u>qualification</u>, <u>employeeID</u>)
No non-trivial dependencies. Qualifications is in BCNF

QualificationsNeeded(<u>neededNo</u>, <u>qualification</u>, <u>subprojectID</u>, assigned)
neededNo qualification subprojectID → assigned
$\{neededNo\ qualification\ subprojectID\}^+ = \{neededNo\ qualification\ subprojectID\ assigned\}$

{neededNo qualification subprojectID}$^+$ Contains all attributes of QualificationsNeeded. No other non-trivial dependencies. QualificationsNeeded is in BCNF

Models(<u>modelName</u> , <u>manufacturer,</u> description, size, consumption)
modelName manufacturer → description, size, consumption
{modelName manufacturer}$^+$ = {modelName manufacturer description, size, consumption}
{modelName manufacturer}$^+$ Contains all attributes of Models. No other non-trivial dependencies. Models is in BCNF.

Machines(<u>serialNo</u>, modelName, manufacturer)
serialNo → modelName, manufacturer
{serialNo}$^+$ = {serialNo, modelName, manufacturer}
{serialNo}$^+$ Contains all attributes of Machines. No other non-trivial dependencies. Machines is in BCNF.

MachineUnavailable(<u>startDate</u>, endDate, <u>serialNo</u>)
startDate serialNo→ endDate
{startDate serialNo}$^+$ = {serialNo, startDate, endDate}
{startDate serialNo}$^+$ Contains all attributes of MachinesUnavailable. No other non-trivial dependencies. MachinesUnavailable is in BCNF.

MachinesNeeded(<u>model</u>, <u>subprojectID</u>, <u>neededNo</u>, assigned, startDate, endDate)
model subprojectID neededNo → assigned, startdate endDate
{model, subprojetID, neededNo}$^+$ = {model, subprojectID, neededNo , startdate, endDate, assigned}
{model, subprojetID, neededNo}$^+$ Contains all attributes of MachinesNeeded. No other non-trivial dependencies. MachinesNeeded is in BCNF.

MachinesAssigned(<u>serialNo</u>, subprojectID, <u>startDate</u>, endDate)
serialNo startDate→ subprojectID endDate
{serialNo, startDate}$^+$ = {serialNo, startDate, subProjectID, endDate}
{serialNo, startDate}$^+$ Contains all attributes of MachinesAssigned. No other non-trivial dependencies. MachinesAssigned is in BCNF.

Our whole relational model is in BCNF.