

CS 353

Database Systems  
Spring 2015

**SITS – Student Internship Tracking System**

Design Report

Group Members  
Sinem Sav - 21102143

Doğancan Demirtaş - 21201824

Can Akgün - 21102268

Table Of Contents

[1. REVISED E/R DIAGRAM 3](#_Toc417310369)

[2. RELATION SCHEMAS 4](#_Toc417310370)

[2.1 Faculty 4](#_Toc417310371)

[2.2 Department 5](#_Toc417310372)

[2.3 Person 6](#_Toc417310373)

[2.4 Student 7](#_Toc417310374)

[2.5 StudentAdvisor 8](#_Toc417310375)

[2.6 Secretary 9](#_Toc417310376)

[2.7 Company 10](#_Toc417310377)

[2.8 RegisteredCompany 11](#_Toc417310378)

[2.9 SelfFoundCompany 12](#_Toc417310379)

[2.10 Quota 13](#_Toc417310380)

[2.11 Opens 14](#_Toc417310381)

[2.12 Application 15](#_Toc417310382)

[2.13 QuotaApply 16](#_Toc417310383)

[2.14 DirectApply 17](#_Toc417310384)

[2.15 Announcement 18](#_Toc417310385)

[2.16 AppFeedbackAnnouncement 19](#_Toc417310386)

[2.17 GeneralAnnouncement 20](#_Toc417310387)

[3. FUNCTIONAL DEPENDENCIES AND NORMALIZATION 21](#_Toc417310388)

[4. FUNCTIONAL COMPONENTS 22](#_Toc417310389)

[4.1 Use Cases 22](#_Toc417310390)

[4.2 Scenarios 26](#_Toc417310391)

[5. User Interface Design and Corresponding SQL Statements 29](#_Toc417310392)

[5.1 Login 29](#_Toc417310393)

[5.2 My Applications - Student 30](#_Toc417310394)

[5.2 Show Quotas - Student 32](#_Toc417310395)

[5.3 Quota Apply - Student 34](#_Toc417310396)

[5.4 Show Companies - Student 35](#_Toc417310397)

[5.5 Add Company - Student 36](#_Toc417310398)

[5.6 Direct Apply - Student 37](#_Toc417310399)

[5.7 Announcements - Student 38](#_Toc417310400)

[5.8 Approve Company – Advisor 39](#_Toc417310401)

[5.9 See Quotas - Company 41](#_Toc417310402)

[5.10 Profile Settings - Company 42](#_Toc417310403)

[5.11 See Quotas - Secretary 43](#_Toc417310404)

[5.12 Add Quota - Secretary 44](#_Toc417310405)

[5.13 See Detailed Quota Applications - Secretary 45](#_Toc417310406)

[5.14 Approve Applications - Secretary 46](#_Toc417310407)

[5.15 See Announcements – Secretary 47](#_Toc417310408)

[5.16 Add Announcement – Secretary 48](#_Toc417310409)

[6. ADVANCED DATABASE COMPONENTS 49](#_Toc417310410)

[6.1 Reports 49](#_Toc417310411)

[6.1.1 Total Number Of Registered and Self-Found Companies 49](#_Toc417310412)

[6.1.2 Total Number of Applications 49](#_Toc417310413)

[6.1.3 Total Number of Approved Applications 49](#_Toc417310414)

[6.2 Views 49](#_Toc417310415)

[6.2.1 Secretary-Quota View 49](#_Toc417310416)

[6.2.2 Secretary-FilteredQuota View 50](#_Toc417310417)

[6.2.3 Student-Feedback Announcement View 50](#_Toc417310418)

[6.3 Triggers 50](#_Toc417310419)

[6.4 Constraints 51](#_Toc417310420)

[6.5 Stored Procedures 52](#_Toc417310421)

# 1. REVISED E/R DIAGRAM

# 2. RELATION SCHEMAS

## 2.1 Faculty

**Relational Model:**

Faculty(facultyName)

**Functional Dependencies:**

No functional dependency.

**Candidate Keys:**

{ (facultyName) }

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE faculty(

facultyName char(30) PRIMARY KEY

)ENGINE = InnoDB;

## 2.2 Department

**Relational Model:**

Department(deptName, facultyName)

**Functional Dependencies:**

deptName -> facultyName

**Candidate Keys:**

{ (deptName) }

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE department(

deptName char(5) PRIMARY KEY,

facultyName char(30),

FOREIGN KEY (facultyName) references faculty )ENGINE = InnoDB;

## 2.3 Person

**Relational Model:**

Person (userID, name, surname, password, phone, deptName)

**Functional Dependencies:**

userID -> name, surname, password, phone, deptName

**Candidate Keys:**

{ (userID) }

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE person(

userID int PRIMARY KEY AUTO\_INCREMENT,

name char(100) NOT NULL,

surname char(100) NOT NULL

password char(12) NOT NULL,

phone char(15),

deptName char(5),

FOREIGN KEY (deptName) references department

)ENGINE = InnoDB;

## 2.4 Student

**Relational Model:**

Student (userID, cgpa)

**Functional Dependencies:**

userID -> cgpa

**Candidate Keys:**

{ (userID) }

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE student(

userID int PRIMARY KEY,

cgpa float,

FOREIGN KEY (userID) references person

)ENGINE = InnoDB;

## 2.5 StudentAdvisor

**Relational Model:**

Student (userID)

**Functional Dependencies:**

No functional dependency.

**Candidate Keys:**

{ (userID) }

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE studentAdvisor(

userID int PRIMARY KEY,

FOREIGN KEY (userID) references person

)ENGINE = InnoDB;

## 2.6 Secretary

**Relational Model:**

Secretary (userID)

**Functional Dependencies:**

No functional dependency.

**Candidate Keys:**

{ (userID) }

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE secretary(

userID int PRIMARY KEY,

FOREIGN KEY (userID) references person

)ENGINE = InnoDB;

## 2.7 Company

**Relational Model:**

Company (compID, name, password, address, phone, applicableDepts, status, supervisorName, supervisorPhone, city, country, evaluatorRating, studentRating, sector)

**Functional Dependencies:**

compID -> name, password, address, phone, applicableDepts, status, supervisorName, supervisorPhone, city, country, evaluatorRating, studentRating, sector)

**Candidate Keys:**

{ (compID) }

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE company(

compID int PRIMARY KEY AUTO\_INCREMENT,

name char(100) NOT NULL,

password char(12) NOT NULL,

address char(100),

phone char(15),

applicableDepts char(30),

status char(15) NOT NULL,

supervisorName char(30),

supervisorPhone char(15),

city char(20),

country char(20),

evaluatorRating int,

studentRating int,

sector char(20)

)ENGINE = InnoDB;

## 2.8 RegisteredCompany

**Relational Model:**

RegisteredCompany (compID, registrationDate, studentID)

**Functional Dependencies:**

compID -> registrationDate, studentID

**Candidate Keys:**

{ (compID) }

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE registeredCompany(

compID int PRIMARY KEY,

registrationDate TIMESTAMP,

FOREIGN KEY (compID) references company

)ENGINE = InnoDB;

## 2.9 SelfFoundCompany

**Relational Model:**

SelfFoundCompany (compID, advisorID)

**Functional Dependencies:**

compID -> advisorID

**Candidate Keys:**

{ (compID) }

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE selfFoundCompany (

compID int PRIMARY KEY,

FOREIGN KEY (compID) references company

FOREIGN KEY (advisorID) references studentAdvisor

)ENGINE = InnoDB;

## 2.10 Quota

**Relational Model:**

Quota (quotaID, internshipDuration, internshipStartDate, internshipEndDate, availableYears, status, quotaAmount, quotaDeadline,compID, generalAnnouncementID)

**Functional Dependencies:**

quotaID -> internshipDuration, internshipStartDate, internshipEndDate,availableYears, status,

quotaAmount, quotaDeadline, compID, generalAnnouncementID

**Candidate Keys:**

{ (quotaID) }

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE quota(

quotaID int PRIMARY KEY AUTO\_INCREMENT,

internshipDuration int NOT NULL,

internshipStartDate date NOT NULL,

internshipEndDate date NOT NULL,

availableYears int NOT NULL,

status char(30),

quotaAmount int NOT NULL,

quotaDeadline date NOT NULL,

FOREIGN KEY (generalAnnouncementID) references generalAnnouncement,

FOREIGN KEY (compID) references company

)ENGINE = InnoDB;

## 2.11 Opens

**Relational Model:**

Opens(quotaID, compID, deptName)

**Functional Dependencies:**

quotaID -> compID, deptName

**Candidate Keys:**

{ (quotaID) }

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE opens (

quotaID int,

compID int,

deptName char(5),

FOREIGN KEY (quotaID) references quota,

FOREIGN KEY (compID) references registeredCompany,

FOREIGN KEY (deptName) references department

)ENGINE = InnoDB;

## 2.12 Application

**Relational Model:**

Application (appID, appSubmitDate, approval, appType, secretaryID)

**Functional Dependencies:**

appID ->appSubmitDate, approval, appType, secretaryID

**Candidate Keys:**

{ (appID) }

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE application(

appID int PRIMARY KEY AUTO\_INCREMENT,

appSubmitDate TIMESTAMP,

approval char(12),

appType char(12),

secretaryID int,

FOREIGN KEY (secretaryID) references secretary

)ENGINE = InnoDB;

## 2.13 QuotaApply

**Relational Model:**

QuotaApply(appID, quotaID, compID, studentID, drawResult)

**Functional Dependencies:**

appID ->quotaID, compID, studentID, drawResult

**Candidate Keys:**

{ (appID) }

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE quotaApply(

appID int PRIMARY KEY,

quotaID int,

compID int,

studentID int,

drawResult int,

FOREIGN KEY (appID) references application,

FOREIGN KEY (quotaID) references quota,

FOREIGN KEY (compID) references registeredCompany,

FOREIGN KEY (studentID) references student

)ENGINE = InnoDB;

## 2.14 DirectApply

**Relational Model:**

DirectApply(appID, compID, studentID, internshipStartDate, internshipEndDate)

**Functional Dependencies:**

appID -> compID, studentID,internshipStartDate, internshipEndDate

**Candidate Keys:**

{ (appID) }

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE directApply(

appID int PRIMARY KEY,

compID int,

studentID int,

internshipStartDate date,

internshipEndDate date,

FOREIGN KEY (appID) references application,

FOREIGN KEY (compID) references registeredCompany,

FOREIGN KEY (studentID) references student

)ENGINE = InnoDB;

## 2.15 Announcement

**Relational Model:**

Announcement(announcementID, date)

**Functional Dependencies:**

announcementID -> date

**Candidate Keys:**

{ (secretaryID) }

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE announcement(

announcementID int PRIMARY KEY AUTO\_INCREMENT,

date TIMESTAMP

)ENGINE = InnoDB;

## 2.16 AppFeedbackAnnouncement

**Relational Model:**

AppFeedbackAnnouncement(announcementID, studentApproval, deadline, studentID, secretaryID)

**Functional Dependencies:**

announcementID -> studentApproval, deadline, studentID, secretaryID

**Candidate Keys:**

{ (announcementID) }

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE appFeedbackAnnouncement (

announcementID int PRIMARY KEY,

studentApproval char(3),

deadline date,

secretaryID int,

FOREIGN KEY (announcementID) references announcement,

FOREIGN KEY (secretaryID) references secretary,

FOREIGN KEY (studentID) references student

)ENGINE = InnoDB;

## 2.17 GeneralAnnouncement

**Relational Model:**

GeneralAnnouncement(announcementID, title, message, secretaryID)

**Functional Dependencies:**

announcementID -> title, message, secretaryID

**Candidate Keys:**

{ (announcementID) }

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE generalAnnouncement (

announcementID int PRIMARY KEY,

title char(30),

message varchar(7000),

secretaryID int,

FOREIGN KEY (announcementID) references announcement,

FOREIGN KEY (secretaryID) references secretary

)ENGINE = InnoDB;

# 3. FUNCTIONAL DEPENDENCIES AND NORMALIZATION

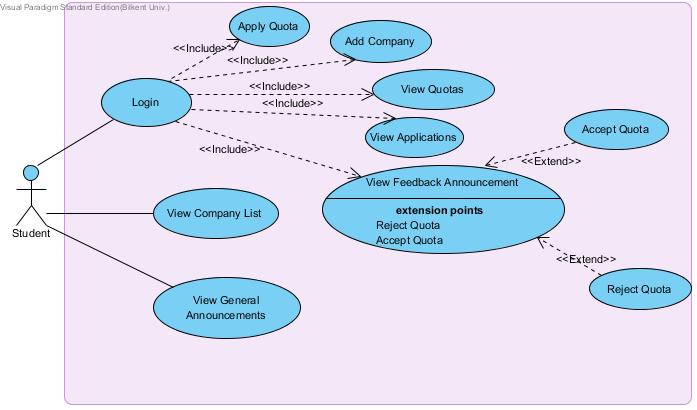
As it can be seen from previous section, all of our functional dependencies include left hand side as candidate key which is also superkey. As a consequent of this, all of our tables are in Boyce-Codd Normal Form and no decomposition is required.

# 4. FUNCTIONAL COMPONENTS

## 4.1 Use Cases

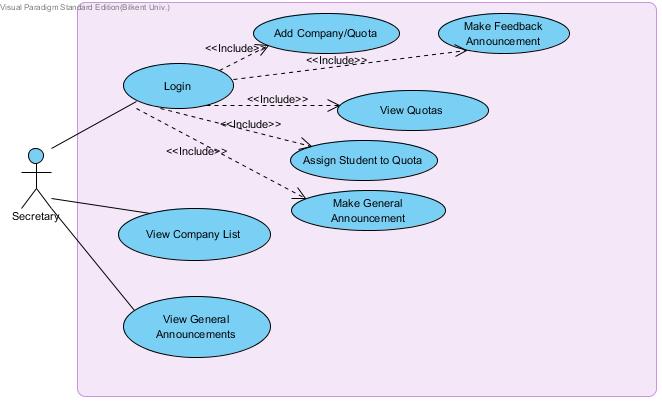
As indicated earlier in this report, Student Internship Tracking System has four different types of users namely students , secretaries, company staff and advisors(internship coordinators of department. In this section, specific features of the program according to user types will be specified using use case diagrams.

**Student:**

****

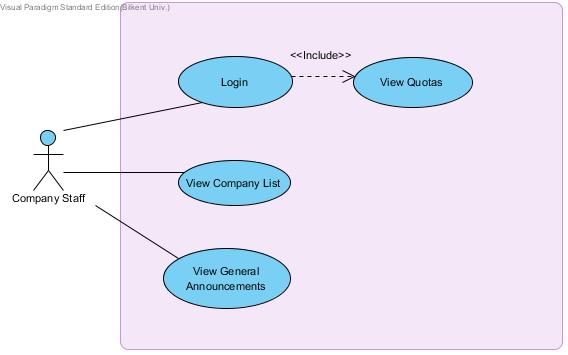
* Students have to login to the system for use cases specified with <<include>>.
* All students can view company list and general announcements from faculty and departments about internship processes regardless of their login operation.
* Students can view quotas of companies.
* Students can apply an existing company's quotas.
* Students can add their own internship company if it does not exist in the company list.
* Students can view their own applications.
* Students can view feedback announcements which are to specify status of their applications.
* After they view feedback announcements, students can accept or reject the quota of a specific company if they are approved. (Specified with <<extends>> in use case diagram).

**Secretary:**

****

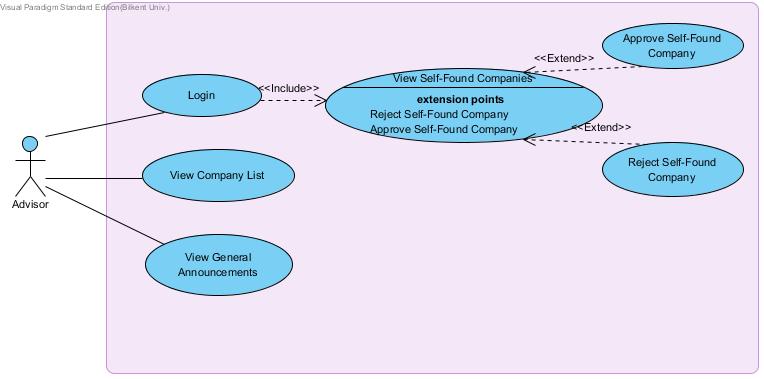
* Secretaries have to login to the system for use cases specified with <<include>>.
* All secretaries can view company list and general announcements from faculty and departments about internship processes regardless of their login operation.
* Secretaries can view quotas of companies.
* Secretaries can add company/quota pairs to the list.
* Secretaries can make feedback announcement which shows result of students' applications.
* Secretaries can make general announcements which can be seen from everybody regardless of login operation.
* Secretaries can see "open" quota and assign it to other students if the quota is rejected by a student.

**Company Staff:**

****

* Company staff have to login to the system for use case specified with <<include>>.
* Company staff can view company list and general announcements from faculty and departments about internship processes regardless of their login operation.
* Company staff can see current status of their quota.

**Advisor:**



* Advisors who are internship coordinators of the department also have to login to the system for use cases specified with <<include>>.
* Advisors can also view company list and general announcements from faculty and departments about internship processes regardless of their login operation.
* Advisors can view self-found companies which are not existing in company list but added by students.
* Advisors can approve or reject self-found company that they viewed.

## 4.2 Scenarios

In this section, some of the most important scenarios for the Student Internship Tracking System will be specified.

**Scenario 1**

**Scenario Name:** QUOTA APPLY

**Participating Actor**: Student

**Flow of Events:**

1. Student has to login into system by using user id and password.

2. Student will click on "Companies" tab.

3. Student will see the list of existing companies and quotas.

4. By clicking apply button next to any specific company, s/he will apply the quota and will wait for result of this application.

**Scenario 2**

**Scenario Name:** ADD SELF-FOUND COMPANY

**Participating Actor:** Student

**Flow of Events:**

1. Student has to login into system by using user id and password.

2. Student will click on "Companies" tab.

3. Student will see the list of existing companies and quotas.

4. In this screen, student will click on "Add Company" button and enter information about self-found company (Company Information, address, phone and website).

5. After specification of company, student will click on "add" button and will wait for approval of this company from advisor(internship coordinator).

**Scenario 3**

**Scenario Name:** ADD COMPANY/QUOTA

**Participating Actor:** Secretary

**Flow of Events:**

1. Secretary has to login into system by using user id and password.

2. Secretary will click on "Companies" tab.

3. Secretary will see the list of existing companies and quotas.

4. In this screen, secretary will click on "Add Company/Quota" button and enter information about company (Company Information, address, phone and website as well as quota information including deadline or max number of students that will be able to apply).

5. After specification of company, secretary will click on "add" button and this company with specified quota will be added to existing company list.

**Scenario 4**

**Scenario Name:** VIEW COMPANY QUOTA

**Participating Actor:** Company Staff

**Flow of Events:**

1. Company staff has to login into system by using user id and password.

2. Company staff will click on "See Quota Status" tab.

3. Company staff can see their quota status in this screen.

**Scenario 5**

**Scenario Name:** APPROVE/REJECT SELF-FOUND COMPANIES

**Participating Actor**: Advisor (Internship Coordinator)

**Flow of Events:**

1. Advisor has to login into system by using user id and password.

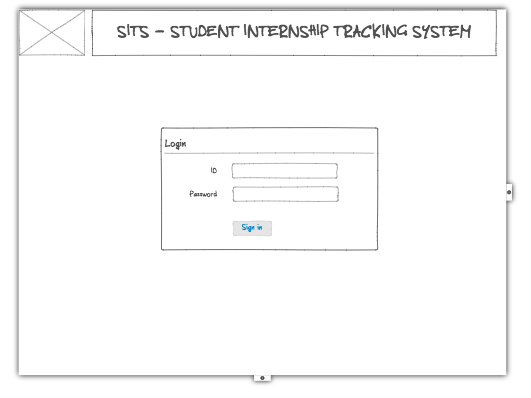
2. Advisor will click on "Companies" tab.

3. Advisor will select "Self-Found Companies" and click on "Filter" to see self-found companies only.

4. In the filtered list of "Self-Found Companies", advisor will click on either approve or reject buttons which are next to company name.

# 5. User Interface Design and Corresponding SQL Statements

## 5.1 Login



**Process:** The user enters his userID and password to sign in the Student Internship Tracking System.

**SQL Statements:**

Logging in:

SELECT userID, password

FROM Person

WHERE userID = @userID AND password = @password;

## SSD:Users:Dogancan:Desktop:cs353_git:interface mockups:My Applications - No Direct.png5.2 My Applications - Student

**Process:**

The user, in this case student, clicks on My Applications link on the navigation bar. The system lists all applications of that student. The student can cancel the quota application(s) or direct application via the action buttons.

**SQL Statements:**

Listing quota applications:

WITH

SELECT quotaID, count(\*) AS allApplications (quotaID, count)

FROM quotaApply

GROUP BY quotaID

SELECT name, city, quotaDeadline, internshipStartDate, internshipEndDate, quotaAmount - count, quotaAmount, status

FROM quotaApply NATURAL JOIN allApplications NATURAL JOIN quota NATURAL JOIN opens NATURAL JOIN company

WHERE userID = @userID;

Listing direct applications:

SELECT name, city, internshipStartDate, internshipEndDate, approval

FROM directApply NATURAL JOIN company NATURAL JOIN application

WHERE userID = @userID;

Cancelling quota application:

DELETE FROM quotaApply

WHERE appID = @appID;

Cancelling direct application:

DELETE FROM directApply

WHERE appID = @appID;

## SSD:Users:Dogancan:Desktop:cs353_git:interface mockups:Quotas.png5.2 Show Quotas - Student

**Process:**

Student clicks on Quotas link on the navigation bar. All quotas which are opened for his/her department is listed. S/he can apply for the quotas which are available to apply. Also when the student selects a specific city from dropdown menu, system displays all quotas from that selected city.

**SQL Statements:**

Listing all quotas:

WITH

SELECT quotaID, count(\*) AS allApplications (quotaID, count)

FROM quotaApply

GROUP BY quotaID

SELECT name, city, quotaDeadline, internshipStartDate, internshipEndDate, quotaAmount - count, quotaAmount, status, availableYears

FROM allApplications NATURAL JOIN quota NATURAL JOIN opens NATURAL JOIN company

WHERE deptName = @deptName;

Applying for an available quota:

INSERT INTO quotaApply VALUES (@quotaID, @compID, @userID, 0);

Populating dropdown list:

SELECT name

FROM company;

Filtering quotas with respect to city:

WITH

SELECT quotaID, count(\*) AS allApplications (quotaID, count)

FROM quotaApply

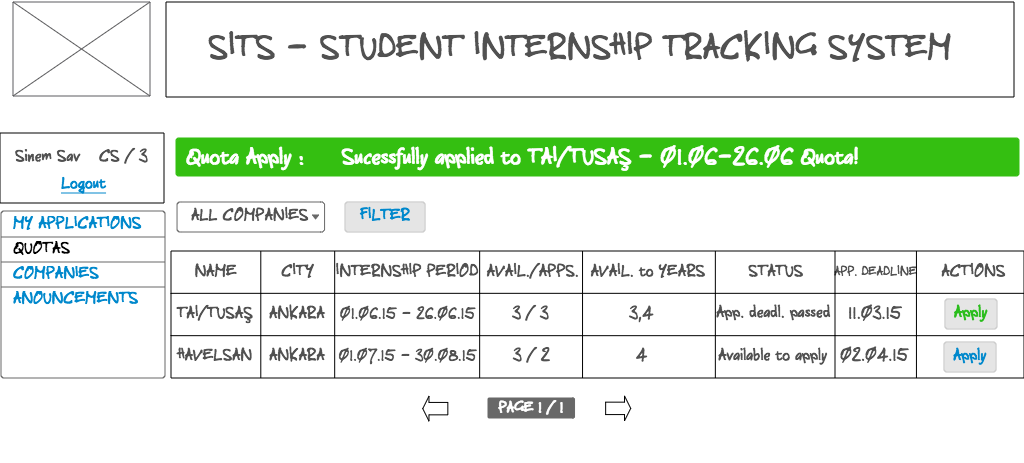
GROUP BY quotaID

SELECT name, city, quotaDeadline, internshipStartDate, internshipEndDate, quotaAmount - count, quotaAmount, status, availableYears

FROM allApplications NATURAL JOIN quota NATURAL JOIN opens NATURAL JOIN company

WHERE city = @city;

## 5.3 Quota Apply - Student



**Process:**

When student successfully applies for an available quota, a message is shown with company name and internship period.

**SQL Statements:**

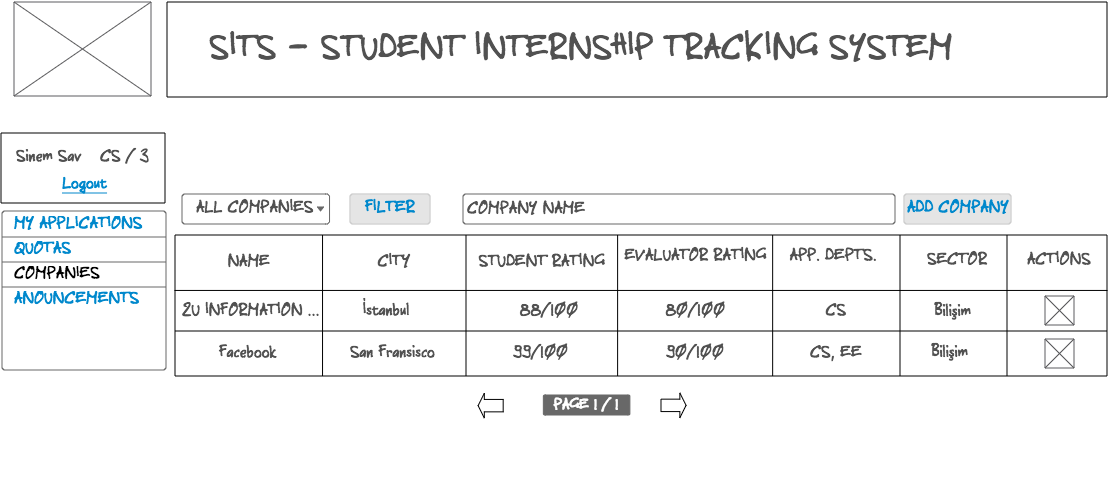
Displaying message:

SELECT name, internshipStartDate, internshipEndDate

FROM opens NATURAL JOIN quota NATURAL JOIN company

WHERE quotaID = @quotaID;

## 5.4 Show Companies - Student



**Process:**

When student clicks on Companies link on the navigation bar, system displays all companies. Student can filter the companies according to their cities. Also student can search a company with its name.

**SQL Statements:**

Displaying all companies:

SELECT name, city, studentRating, evaluatorRating, applicableDepts, sector

FROM company;

Filtering companies with their cities:

SELECT name, city, studentRating, evaluatorRating, applicableDepts, sector

FROM company

WHERE city = @city;

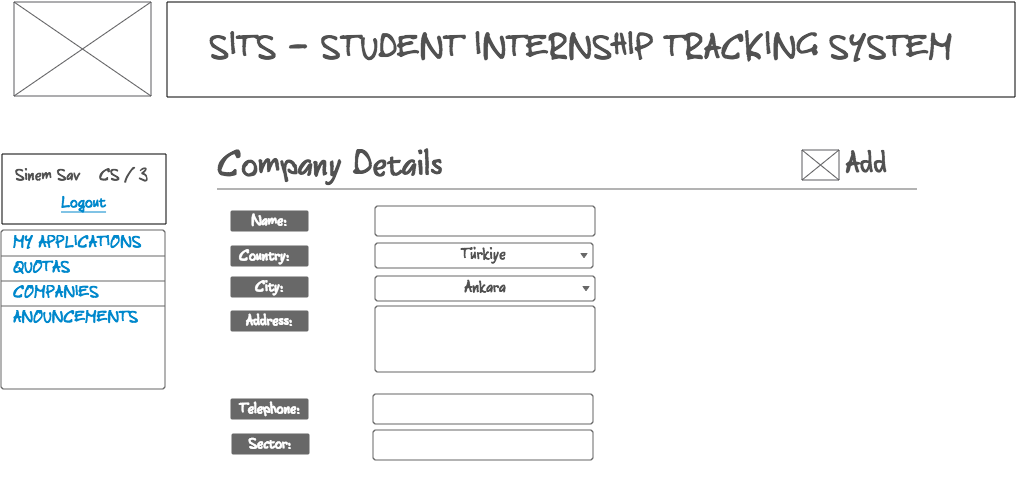
Filtering companies with company name:

SELECT name, city, studentRating, evaluatorRating, applicableDepts, sector

FROM company

WHERE name like ‘%search\_name’;

## 5.5 Add Company - Student



**Process:**

When student clicks on Add Company button in 5.4, system will display a form to add company details. NULL values will be initialized by other actors (secretary, company staff and advisor).

**SQL Statements:**

Adding company:

INSERT INTO selfFoundCompany VALUES (NULL);

//Trigger of this action is involved; a new tuple in company table is created. Then company table is updated with the rest of the form values.

UPDATE company

SET name=@name, password=@auto\_generated\_password, address=@address, phone=@telephone, status = ‘not approved’, city = @city, country = @country, sector = @sector;

## SSD:Users:Dogancan:Desktop:cs353_git:interface mockups:Direct Application.png5.6 Direct Apply - Student

**Process:**

When student clicks on action button in 5.4, the company should be applicable for that student’s department, this form is displayed. Student enters internship start and end dates.

**SQL Statements:**

Making direct apply:

INSERT INTO directApply VALUES (@compID, @userID, @internshipStartDate, @internshipEndDate);

## SSD:Users:Dogancan:Desktop:cs353_git:interface mockups:Anouncements.png5.7 Announcements - Student

**Process:**

When student clicks on Announcements button on the navigation bar, system displays current announcements for his/her department.

**SQL Statements:**

Displaying announcements:

SELECT title, message, date

FROM generalAnnouncement NATURAL JOIN announcement

WHERE deptName = (SELECT deptName

FROM student NATURAL JOIN person

WHERE userID = @userID);

## SSD:Users:Dogancan:Desktop:cs353_git:interface mockups:Approve Company - Advisor.png5.8 Approve Company – Advisor

**Process:**

When advisor clicks on Companies link on the navigation bar, all companies, registered and self-found companies, are listed. Advisor can approve/disapprove a company to the system by specifying its applicable departments.

**SQL Statements:**

Displaying companies:

SELECT name, city, studentRating, evaluatorRating, applicableDepts, sector

FROM company;

Filtering companies as Self Found Company:

SELECT name, city, studentRating, evaluatorRating, applicableDepts, sector

FROM company

WHERE advisorID is not NULL;

Filtering companies with company name:

SELECT name, city, studentRating, evaluatorRating, applicableDepts, sector

FROM company

WHERE name like ‘%search\_name’;

Initializing applicable departments of a company and approving that company:

UPDATE company

SET applicableDepts = @dropdownApplicableDepts, status = ‘approved’

WHERE compID = @compID;

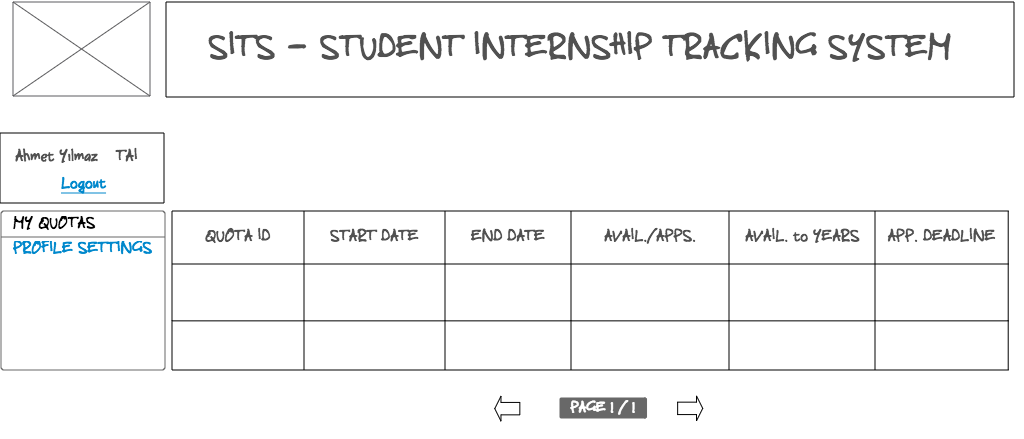
NOTE: Since the status of a company is initialized as ‘not approved’, there is no need to update that value when advisor disapproves the company.

UPDATE selfFoundCompany

SET advisorID = @userID

WHERE compID = @compID;

## 5.9 See Quotas - Company



**Process:**

When company staff logins and clicks on My Quotas link on the navigation bar, system displays all quotas, which belong to that company.

**SQL Statements:**

Displaying quotas:

WITH

SELECT quotaID, count(\*) AS allApplications (quotaID, count)

FROM quotaApply

GROUP BY quotaID

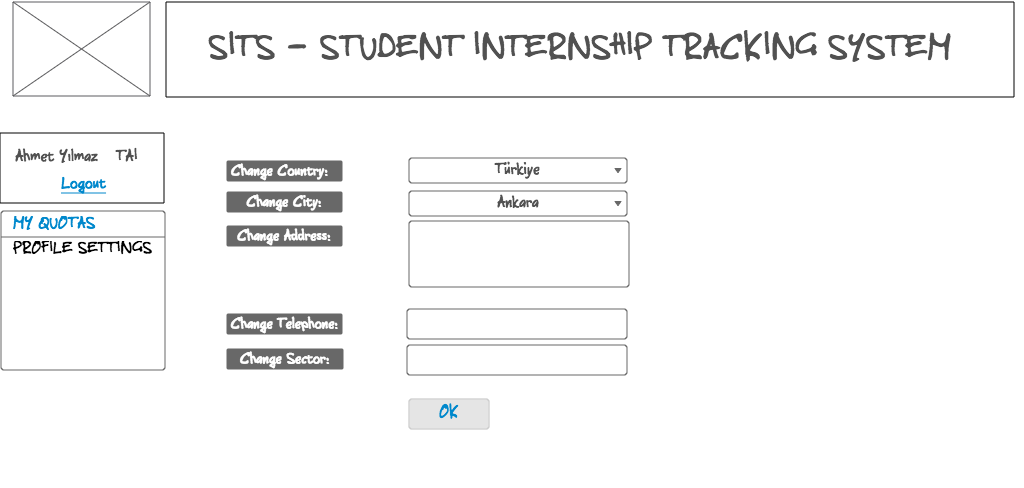
SELECT quotaID, quotaDeadline, internshipStartDate, internshipEndDate, quotaAmount - count, quotaAmount, availableYears

FROM allApplications NATURAL JOIN quota NATURAL JOIN opens

WHERE compID = @compID;

@compID -> Obtained from login credentials.

## 5.10 Profile Settings - Company



**Process:**

When company staff clicks on Profile Settings link on the navigation bar, system displays respective forms to update company attributes.

**SQL Statements:**

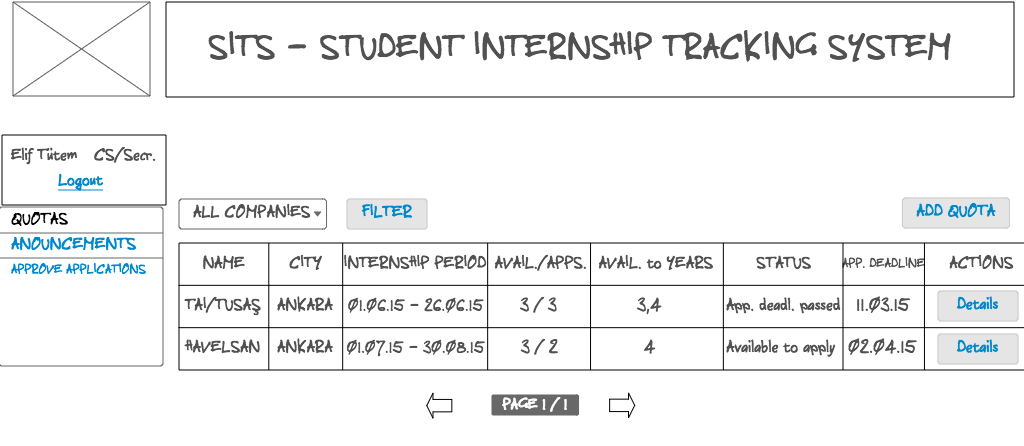
Updating company attributes:

UPDATE company

SET country = @country, city = @city, address = @address, phone = @telephone, sector = @sector

WHERE compID = @compID;

## 5.11 See Quotas - Secretary



**Process:**

When secretary logins to the system and clicks on Quotas link on the navigation bar, system displays all the quotas which are opened to his/her department. S/he can filter the quotas according to companies’ cities. Also s/he can add quota by simply clicking on Add Quota button on the right. Finally, s/he can see the specific quota details by clicking on Details button on each row.

**SQL Statements:**

Listing all quotas:

We use view in section 6.2.1 for this operation.

Filtering quotas with respect to city:

We use view in section 6.2.2 for this operation.

## SSD:Users:Dogancan:Desktop:cs353_git:interface mockups:Secretary - Add Quota.png5.12 Add Quota - Secretary

**Process:**

When secretary clicks on Add Quota button in 5.11, system displays respective forms to add a quota for a company.

**SQL Statements:**

Adding quota:

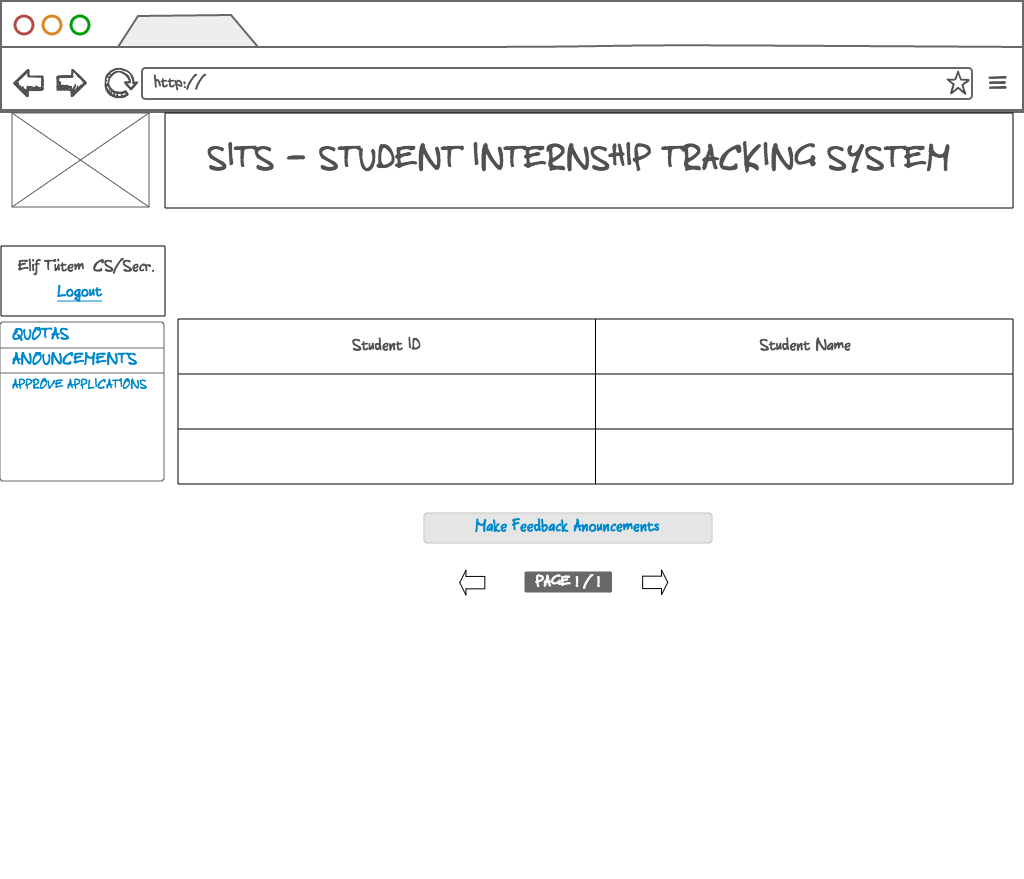
INSERT INTO opens VALUES (@compID, @deptName);

// Trigger of this action is invoked; a new tuple in quota table is created. Then quota table is updated with the rest of the form values.

UPDATE quota

SET internshipDuration = @duration, internshipStartDate = @internshipStartDate, internshipEndDate = @internshipEndDate, avaliableYears =@avaliableYears, status = @status, quotaAmount = @quotaAmount, quotaDeadline = @quotaDeadline;

## 5.13 See Detailed Quota Applications - Secretary



**Process:**

When secretary clicks on Details button in 5.11, system displays applicants of that quota.

**SQL Statements:**

Displaying applicants of a specific quota:

SELECT userID, name

FROM student NATURAL JOIN person NATURAL JOIN quotaApply

WHERE quotaID = @quotaID;

Making feedback announcements:

INSERT INTO appFeedbackAnnouncement VALUES (NULL, @auto\_deadline, @userID, @userID);

## SSD:Users:Dogancan:Desktop:cs353_git:interface mockups:Secretary Approve Applications.png5.14 Approve Applications - Secretary

**Process:**

When secretary clicks on Approve Applications button on the navigation bar, system displays all applications (quota and direct). Secretary can approve/disapprove this applications via the action buttons in each row.

**SQL Statements:**

Displaying all applications:

SELECT compID, userID, appType

FROM application NATURAL JOIN directApply NATURAL JOIN quotaApply;

Approving an application:

UPDATE application

SET approval = ‘approved’

WHERE appID = @appID;

Disapproving an application:

UPDATE application

SET approval = ‘disapproved’

WHERE appID = @appID;

## SSD:Users:Dogancan:Desktop:cs353_git:interface mockups:Secretary Anouncements.png5.15 See Announcements – Secretary

**Process:**

When secretary clicks on Announcements button on the navigation bar, system displays all announcements. Secretary can add announcement by simple clicking on Add Announcement button on the right.

**SQL Statements:**

Displaying all announcements:

SELECT title, message, date

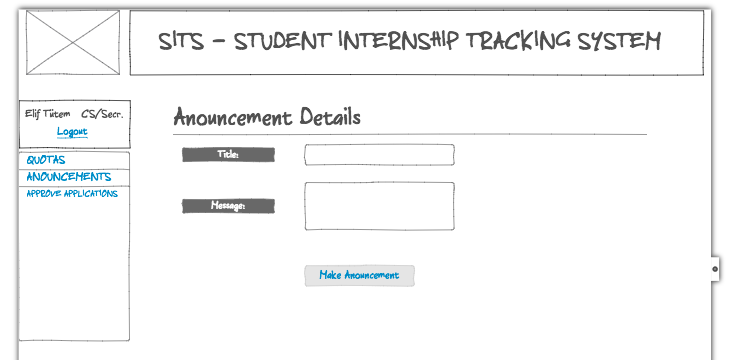
FROM generalAnnouncement NATURAL JOIN announcement

WHERE deptName = (SELECT deptName

FROM secretary NATURAL JOIN person

WHERE userID = @userID);

## 5.16 Add Announcement – Secretary



**Process:**

When secretary clicks on Add Announcement button in 5.15, system displays respective forms to get general announcement title and message.

**SQL Statements:**

Making an general announcement:

INSERT INTO generalAnnouncement VALUES (@title, @message, @userID);

# 6. ADVANCED DATABASE COMPONENTS

## 6.1 Reports

### 6.1.1 Total Number Of Registered and Self-Found Companies

WITH self\_found\_companies (comp\_id, comp\_name, numberCompanies)

AS (SELECT count(compID)

FROM Company

WHERE status = "not approved"),

WITH registered\_companies (comp\_id, comp\_name, numberCompanies)

AS (SELECT count(compID)

FROM Company

WHERE status = " approved")

SELECT S.numberCompanies, R.numberCompanies

FROM self\_found\_companies S, registered\_companies R;

### 6.1.2 Total Number of Applications

SELECT count(\*)

FROM Application;

### 6.1.3 Total Number of Approved Applications

SELECT count(appID)

FROM Application

WHERE approval = "approved";

## 6.2 Views

### 6.2.1 Secretary-Quota View

A secretary cannot view any quota which is not from his/her department (thus, cannot approve or reject these quotas). Therefore, we will use a view for this operation and use secretary id to get department:

WITH

SELECT quotaID, count(\*) AS allApplications (quotaID, count)

FROM quotaApply

GROUP BY quotaID

CREATE VIEW secretary\_quotas\_view AS

SELECT name, city, quotaDeadline, internshipStartDate, internshipEndDate, quotaAmount - count, quotaAmount, status, availableYears

FROM allApplications

WHERE deptName = @deptName;

### 6.2.2 Secretary-FilteredQuota View

This view is same as the previous one except it uses filtering operation:

WITH

SELECT quotaID, count(\*) AS allApplications (quotaID, count)

FROM quotaApply

GROUP BY quotaID

CREATE VIEW secretary\_filteredQuota\_view AS

SELECT name, city, quotaDeadline, internshipStartDate, internshipEndDate, quotaAmount - count, quotaAmount, status, availableYears

FROM allApplications NATURAL JOIN quota NATURAL JOIN opens NATURAL JOIN company

WHERE city = @city AND deptName = @deptName;

### 6.2.3 Student-Feedback Announcement View

A student cannot view any application feedback announcement which doesn't belong to them(doesn't include result of his/her application). Therefore, we will use a view for this operation and use student id to get correct feedback announcement:

CREATE VIEW student\_feedbackAnnouncement\_view AS

SELECT studentApproval, deadline

FROM AppFeedbackAnnouncement

WHERE studentID = @studentID;

## 6.3 Triggers

* As our application table is a dynamic table and need to be updated when a student makes a quota or direct application, we will update this table by using trigger and add this application.
* If a student cancels an application, it is deleted from directApply or quotaApply table which requires deletion from Application table also. This process will be managed by a trigger.
* If a student, secretary or advisor is added and it is not existing in Person table, it should be firstly added to the Person table which again will be managed by a trigger.
* When a registered or self-found company is added it should first added to Company table with a trigger (if it is a self-found company, its’ status will be “not approved” initially).
* Similarly, when a feedback announcement or general announcement is made by a secretary, it should be first added to Announcements table with a trigger.
* If a tuple is deleted from Person table, and it is existing in subclasses(sub-tables of Person) it should be deleted from them also.
* If a tuple is deleted Announcement table and it is existing in subclasses(sub-tables of Announcement) it should be deleted from them also.
* If a tuple is deleted from Company table, and it is existing in subclasses(sub-tables of Company) it should be deleted from them also.
* When a secretary opens a quota (adds a tuple to opens table in database), it should be first added to Quota table with a trigger.
* When a self-found company is approved by advisor, it should be added to registered company's table which will be done by trigger.

## 6.4 Constraints

* Student Internship Tracking System cannot be used without login operation except open services (View Company List and General Announcements).
* Status of any company cannot be null since it indicates whether the company is registered or self-found (waiting for approval).
* Password of any user cannot be less than 6 characters and more than 12 characters.
* Internship end date cannot be earlier than its start date.
* Student who make a quota application cannot make direct application and should cancel the quota application to make direct application (other way around is also not applicable).
* Available years of a quota cannot include years less than 1 and more than 3.
* For each company, only one id and password will exist for login operation.
* Duration of any internship must be bigger than 20 weekdays.
* Students cannot apply to quotas which are not applicable for their department.
* Students cannot apply to quotas if deadline is passed.
* Secretaries cannot see applications of departments other than their own department.
* If the deadline of a quota application is not passed, secretaries cannot make feedback announcements for these applications (cannot announce result before the deadline).

## 6.5 Stored Procedures

* Our company list page is available for anyone regardless of their login operation. Thus, we plan to use stored procedure to get all companies.
* When a student, secretary or advisor is added we should check if they exist in Person table and if not we should first add them to the Person table. This process is same for all insertion into these tables and because of this; we will use a stored procedure.
* Similarly, when a registered or self-found company is added it should first added to Company table which will be a stored procedure.
* When a quota is rejected by a student, it will be automatically open quota and this procedure will be a stored procedure.