**Database Design**

Hours log:

09/25 – 30min (16:00-16:30) Hiranya (instructor’s office)

09/26 - 1h (21:30 - 22:30) – Deni

09/28- 30min (11:30 – 12:00)- Hiranya

09/29 - 30min (15:00 - 15:30) – Deni

09/30 - 30min (12:15 - 12:45) – Deni

10/01- 2h (15:00-17:00)- Hiranya

10/02 – 30min (15:10 – 15:40) – Deni (instructor’s office)

10/05 – 1h 30min (14:00 – 15:30) – Deni

10/06 – 1h 15min (12:45 – 14:00) – Deni & Hiranya

10/06 – 4h (15:45 – 19:45) – Deni

10/06 – 2h 30min (21:00 – 23:30) – Deni

10/07 – 4h (17:00-20:00) – Hiranya

10/09 – 1h (07:50 – 08:50) – Deni

10/09 – 2h (11:30 – 13:30) – Deni

# E/R Modeling

Since the application should be able to store students’ edited schedules (4-year plan), Figure 1.1 contains “Student” entity set and “StudentSchedule” weak entity sets. “StudentSchedule” is a weak entity sets because it is a subunit of “Student” and “Course” entity sets and it cannot be uniquely identified without using attributes from those two relations. Student’s personal schedule/plan will be stored in the “StudentSchedule” set.

**Figure 1.1**

Student (studentID, firstName, lastName, email, username, password)

The “Student” entity set has studentID, firstName, lastName, email, username, and password as attributes. “email” is the student’s ERAU unique email provided by the university. “username” is the first part of the email till the @ sign. This will allow student to sign in to his account by typing the full email or just the username part. “password” is the password specified by the student in order to access the account.

StudentSchedule (studentID, term, courseID, grade, status)

Weak entity set “StudentSchedule” has studentID (inherited from “Student”), courseID (inherited from “Course” entity seat through “Takes” relation), term, grade, and status as attributes. “term” specifies the term in which the course is taken. “grade” specifies the grade for the course. Grade will be entered by the student once the course is completed or if student wants to check possible GPA for the semester or CUM GPA. “status” specifies the current status of the course (transfer, taken, planned). StudentID and term determine the status. Therefore, those two attributes are the key for the relation.

HasSchedule (studentID, courseID, term)

 “ HasSchedule” relationship connects “Student” and “StudentSchedule” entity sets. This relation is needed since there is one-to-one relation between those two entity sets.

In this view there are two entity sets, “Student” and “Major”. “Student” entity set was already described and explained in the previous view. “Major” entity set assigns courses to the corresponding major.

**Figure 1.2**

Major (majorID, totalCredits, type, level)

“Major” entity set contains majorID, totalCredits, type, and level as attributes. “totalCredits” specifies the number of credits required to acquire the degree for that specific major. “level” specifies if it’s undergrad or grad degree. “type” specifies it it’s a major, minor, or dual major.

HasMajor (studentID, majorID)

 “ HasMajor” relationship connects “Student” and “Major” entity sets. This relation is needed since there is many-to-many relation between those two entity sets.

In this view there are two entity sets, “Student” and “Major”. “Student” and “Major” entity set was already described and explained in the previous view.

**Figure 1.3**

HasMinor (studentID, majorID)

 “ HasMinor” relationship connects “Student” and “Major” entity sets. This relation is needed since there is many-to-many relation between those two entity sets.

In this view there are two entity sets, “Course” and “Major”. “Major” entity set was already described and explained in the previous view. “Course” entity set contains all the courses and their data.

**Figure 1.4**

Course (courseID, fullName, credits, termOffered)

“Course” entity set contains courseID, fullName, credits, and termOffered as attributes. “fullName” specifies the full name of the course. “credit” specifies the number of credits for that course. “termOffered” specifies the term in which the course is offered (Spring, Fall, Summer A, All, etc.)

HasCourse (majorID, courseID)

 “ HasCourse” relationship connects “Course” and “Major” entity sets. This relation is needed since there is many-to-many relation between those two entity sets.

In this view there are two entity sets, “StudentSchedule” and “Course”. Both entity sets were already described and explained in the previous views.

**Figure 1.5**

Takes (courseID, studentID, term)

 “ Takes” relationship connects “StudentSchedule” and “Course” entity sets. This relation is needed since there is many-to-many relation between those two entity sets.

In this view there is one entity set, “Course”, that relates to itself. This entity set was already described and explained in the previous views.

**Figure 1.6**

PreRequisites (courseID, preRequisiteCourse)

 “ PreRequisites” relationship connects “Course” entity sets to itself by specifying which course(s) is pre-requisite for selected course. This relation is needed since this is many-to-many relation.

In this view there is one entity set, “Course”, that relates to itself. This entity set was already described and explained in the previous views.

**Figure 1.7**

CoRequisites (courseID, coRequisiteCourse)

 “ CoRequisites” relationship connects “Course” entity sets to itself by specifying which course(s) is co-requisite for selected course. This relation is needed since this is many-to-many relation.

In this view there are two entity sets, “Course” and “CourseOffering”. “Course” entity set was already described and explained in the previous view. “CourseOffering” is a weak entity set (subunit of “Course”) and contains schedule for the current and future terms.

**Figure 1.8**

CourseOffering (courseID, term, section, instructorID, courseNumber, lectureDays, timeSlot, classroom)

“CourseOffering” entity set contains courseID, term, section, instructorID, courseNumber, lectureDays, timeSlot, classroom as attributes. “term” specifies the specific term for that course (S14 = Spring 2014, F15 = Fall 2015, etc.). “section” specifies the specific section for the course if there is more sections offered. “courseNumber” is unique identifier for the course on section. It is the same for the same course offered in different semesters. “lectureDays” specified which days are lectures for that class (MoWeFri, TuTh, Tu, Th, etc.). “timeSlot” specifies at what time is the lecture (8:00-8:50, etc.). “classroom” specifies the classroom number for that course (LB 162, etc.).

OfferedIn (courseID, term, section)

 “ OfferedIn” relationship connects “Course” and “CourseOffering” entity sets. This relation is needed since there is many-to-many relation between those two entity sets.

In this view there are two entity sets, “Student” and “Instructor”. Both entity sets were already described and explained in the previous view.

**Figure 1.9**

Instructor (instructorID, firstName, lastName, email, phoneNumber, isPC, username, password)

“Instructor” entity set has studentID, firstName, lastName, email, phoneNumber, isPC, username, and password as attributes. “email” is the instructor’s ERAU unique email provided by the university. “username” is the first part of the email till the @ sign. This will allow instructor to sign in to his account by typing the full email or just the username part. “password” is the password specified by the instructor in order to access the account. “phoneNumber” is instructor’s office phone number. “isPC” is field that specifies if the instructor is program coordinator as well.

Advises (studentID, instructorID, type)

 “Advises” relationship connects “Instructor” and “Student” entity sets. This relation is needed since there is many-to-many relation between those two entity sets. “type” attribute specifies if the advisor is primary or secondary if student has more than one advisor.

In this view there are two entity sets, “Instructor” and “Availability”. “Instructor” entity set was already described and explained in the previous views. “Availability” entity set specifies the availability for a specific instructor in a specific term.

**Figure 1.10**

Availability (instructorID, term, hours)

“Availability” entity set contains instructorID, term, and hours as attributes. “term” specifies the specific term. “hours” specifies how many credit hours the instructor can teach in that specific term.

IsAvailable (instructorID, term)

 “ IsAvailable” relationship connects “Instructor” and “Availability” entity sets. This relation is needed since there is many-to-many relation between those two entity sets.

In this view there are two entity sets, “Instructor” and “CourseOffering”. Both entity sets were already described and explained in the previous views.

**Figure 1.11**

Teaches (courseID, term, section, instructorID)

 “Teaches” relationship connects “Instructor” and “CourseOffering” entity sets. This relation is needed since there is many-to-many relation between those two entity sets.

In this view there are two entity sets, “Student” and “CourseOffering”. Both entity sets were already described and explained in the previous views.

**Figure 1.12**

EnrolledIn (studentID, courseID, term, section)

 “ EnrolledIn” relationship connects “Student” and “CourseOffering” entity sets. This relation is needed since there is many-to-many relation between those two entity sets.

# Local Relational Schemas

**Figure 1.1**

The ER schema can be translated in the following relations. The logic behind creating relational schemas from ER views can be found in the “E/R Modeling” section.

Relations:

Student (studentID, firstName, lastName, email, username, password)

StudentSchedule (studentID, courseID, term, grade, status)

HasSchedule (courseID, term)

The following are dependencies created from the original problem:

studentID → firstName                                           3NF (left side super key)

studentID → lastName 3NF (left side super key)

studentID → email 3NF (left side super key)

email → username 3NF (left side super key)

username → password 3NF (left side super key)

username → studentID 3NF (left side super key)

studentID, courseID, term → grade 3NF (left side super key)

studentID, courseID, term → status 3NF (left side super key)

In the “Student” relation, studentID, username, and email are keys (each one separately, not all together). Those attributes imply all other attributes in the relation. Therefore all dependencies are in 3NF since left side is a super key (every key is a super key as well). Consequently, the relation remains the same as declared above.

In “StudentSchedule” relation closure, studentID, courseID, term attributes imply all other attributes. Since studentID, courseID, and term is a super key this automatically makes all dependencies being 3NF. Therefore, the relation remains the same as declared above.

**Figure 1.2**

The ER schema can be translated in the following relations. The logic behind creating relational schemas from ER views can be found in the “E/R Modeling” section. “Student” relation is the same as in the Figure 1.1. Therefore, dependencies for that relation are not shown.

Relations:

Student (studentID, firstName, lastName, email, username, password)

Major (majorID, totalCredits, type, level)

HasMajor (studentID, majorID)

The following are dependencies created from the original problem:

majorID → type 3NF (left side super key)

majorID → level 3NF (left side super key)

majorID → totalCredits 3NF (left side super key)

In the “Program” relation, programID implies all the other attributes in the relation. Therefore, all other additional non-trivial dependencies derived from the closure include key on the left hand side, which automatically makes all dependencies being 3NF. Therefore, the relation remains the same as declared above.

**Figure 1.3**

The ER schema can be translated in the following relations. The logic behind creating relational schemas from ER views can be found in the “E/R Modeling” section. “Student” relation is the same as in the Figure 1.1. “Major” relation is the same as in the Figure 1.2. Therefore, dependencies for those relations are not shown.

Relations:

Student (studentID, firstName, lastName, email, username, password)

Major (majorID, totalCredits, type, level)

HasMinor (studentID, majorID)

**Figure 1.4**

The ER schema can be translated in the following relations. The logic behind creating relational schemas from ER views can be found in the “E/R Modeling” section. “Major” relation is the same as in the Figure 1.2. Therefore, dependencies for that relation are not shown.

Relations:

Major (majorID, totalCredits, type, level)

Course (courseID, fullName, credits, termOffered)

HasCourse (majorID, courseID)

The following are dependencies created from the original problem:

courseID → fullName 3NF (left side super key)

courseID → credits 3NF (left side super key)

courseID → termOffered 3NF (left side super key)

In the “Course” relation, courseID attribute implies all the other attributes in the relation. Therefore, all additional non-trivial dependencies derived from the closure include key on the left hand side, which automatically makes all dependencies being 3NF. Therefore, the relation remains the same as declared above.

**Figure 1.5**

The ER schema can be translated in the following relations. The logic behind creating relational schemas from ER views can be found in the “E/R Modeling” section. “StudentSchedule” relation is the same as in the Figure 1.1. “Course” relation is the same as in the Figure 1.4. Therefore, dependencies for that relation are not shown.

Relations:

StudentSchedule (studentID, courseID, term, grade, status)

Course (courseID, fullName, credits, termOffered)

Takes (courseID, studentID, term)

**Figure 1.6**

The ER schema can be translated in the following relations. The logic behind creating relational schemas from ER views can be found in the “E/R Modeling” section. “Course” relation is the same as in the Figure 1.4. Therefore, dependencies for that relation are not shown.

Relations:

Course (courseID, fullName, credits, termOffered)

PreRequisites (courseID, preRequisiteCourse)

**Figure 1.7**

The ER schema can be translated in the following relations. The logic behind creating relational schemas from ER views can be found in the “E/R Modeling” section. “Course” relation is the same as in the Figure 1.4. Therefore, dependencies for that relation are not shown.

Relations:

Course (courseID, fullName, credits, termOffered)

CoRequisites (courseID, coRequisiteCourse)

**Figure 1.8**

The ER schema can be translated in the following relations. The logic behind creating relational schemas from ER views can be found in the “E/R Modeling” section. “Course” relation is the same as in the Figure 1.4. Therefore, dependencies for that relation are not shown.

Relations:

Course (courseID, fullName, credits, termsOffered)

CourseOffering (courseID, term, section, instructorID, courseNumber, lectureDays, timeSlot, classroom)

OfferedIn (courseID, term, section)

The following are dependencies created from the original problem:

courseID, term, section → courseNumber 3NF (left side super key)

courseID, term, section → lectureDays 3NF (left side super key)

courseID, term, section → timeSlot 3NF (left side super key)

courseID, term, section → classroom 3NF (left side super key)

courseID, term, section → instructorID 3NF (left side super key)

In the “CourseOffering” relation, courseID, term, and section attributes together imply all the other attributes. Therefore, those attributes together form a relation key. All additional non-trivial dependencies derived from the closure include key on the left hand side, which automatically makes all dependencies being 3NF. Therefore, the relation remains the same as declared above.

**Figure 1.9**

The ER schema can be translated in the following relations. The logic behind creating relational schemas from ER views can be found in the “E/R Modeling” section. . “Student” relation is the same as in the Figure 1.1. Therefore, dependencies for that relation are not shown.

Relations:

Student (studentID, firstName, lastName, email, username, password)

Instructor (instructorID, firstName, lastName, email, phoneNumber, isPC, username, password)

Advises (studentID, instructorID, type)

The following are dependencies created from the original problem:

instructorID → firstName                                           3NF (left side super key)

instructorID → lastName 3NF (left side super key)

instructorID → email 3NF (left side super key)

instructorID → phoneNumber 3NF (left side super key)

instructorID → isPC 3NF (left side super key)

email → username 3NF (left side super key)

username → password 3NF (left side super key)

username → instructorID 3NF (left side super key)

studentID, instructorID → type 3NF (left side super key)

In the “Instructor” relation, instructorID username, and email are keys (each one separately, not all together). Those attributes imply all other attributes in the relation. Therefore all dependencies are in 3NF since left side is a super key. Consequently, the relation remains the same as declared above.

**Figure 1.10**

The ER schema can be translated in the following relations. The logic behind creating relational schemas from ER views can be found in the “E/R Modeling” section. . “Instructor” relation is the same as in the Figure 1.9, while “CourseOffering” relation is the same as in the Figure 1.8. Therefore, dependencies for those relations are not shown.

Relations:

Instructor (instructorID, firstName, lastName, email, phoneNumber, isPC, username, password)

Availability (instructorID, term, hours)

IsAvailable (instructorID, term)

The following are dependencies created from the original problem:

instructorID, term → hours 3NF (left side super key)

In the “Availability” relation, courseID, and term attributes together imply all the other attributes. Therefore, those attributes together form a relation key. All additional non-trivial dependencies derived from the closure include key on the left hand side, which automatically makes all dependencies being 3NF. Therefore, the relation remains the same as declared above.

**Figure 1.11**

The ER schema can be translated in the following relations. The logic behind creating relational schemas from ER views can be found in the “E/R Modeling” section. . “Instructor” relation is the same as in the Figure 1.9, while “CourseOffering” relation is the same as in the Figure 1.8. Therefore, dependencies for those relations are not shown.

Relations:

Instructor (instructorID, firstName, lastName, email, phoneNumber, isPC, username, password)

CourseOffering (courseID, term, section, instructorID, courseNumber, lectureDays, timeSlot, classroom)

Teaches (courseID, term, section, instructorID)

**Figure 1.12**

The ER schema can be translated in the following relations. The logic behind creating relational schemas from ER views can be found in the “E/R Modeling” section. . “Student” relation is the same as in the Figure 1.1, while “CourseOffering” relation is the same as in the Figure 1.4. Therefore, dependencies for those relations are not shown.

Relations:

Student (studentID, firstName, lastName, email, advisor)

CourseOffering (courseID, term, section, instructorID, courseNumber, lectureDays, timeSlot, classroom)

EnrolledIn (studentID, courseID, term, section)

# Global Relational Schema

In the figure below, a global relational schema can be seen. The schema was created by combining the views from “Local Relational Schema” section. Only key attributes for each entity set are shown for better readability. For other attributes check the corresponding relations.

Student (studentID, firstName, lastName, email, username, password)

StudentSchedule (studentID, courseID, term, grade, status)

Course (courseID, fullName, credits, termOffered)

CourseOffering (courseID, term, section, instructorID, courseNumber, lectureDays, timeSlot, classroom)

Major (majorID, totalCredits, type, level)

Instructor (instructorID, firstName, lastName, email, phoneNumber, isPC, username, password)

Availability (instructorID, term, hours)

HasSchedule (courseID, term)

HasMajor (studentID, majorID)

HasMinor (studentID, majorID,)

HasCourse (majorID, courseID)

Takes (courseID, studentID, term)

PreRequisites (courseID, preRequisiteCourse)

CoRequisites (courseID, coRequisiteCourse)

OfferedIn (courseID, term, section)

Advises (studentID, instructorID, type)

IsAvailable (instructorID, availabilityID)

Teaches (courseID, term, section, instructorID)

EnrolledIn (studentID, courseID, term, section)