# Name of Your Company

BookYourInstructor LTD (Do not let the course eat you!)

# Project Title

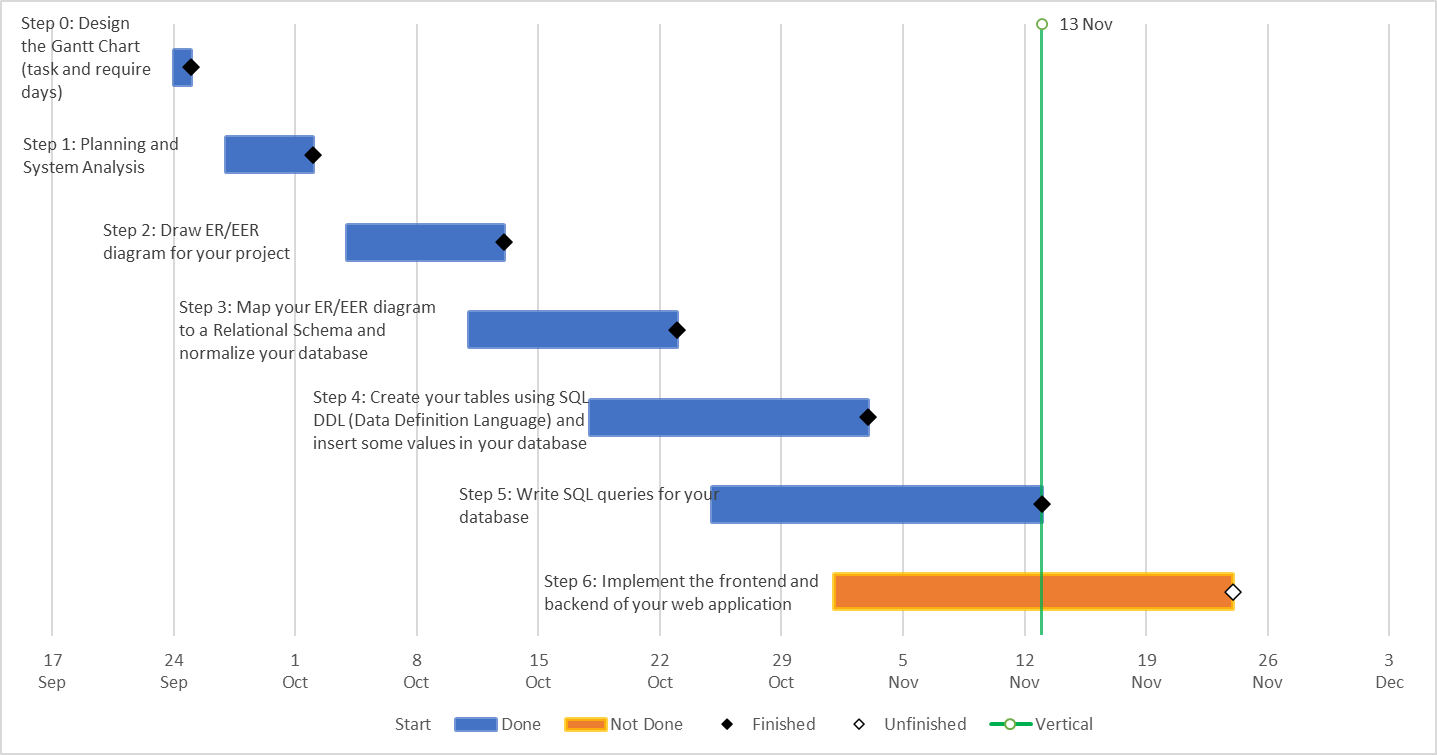
**Instructor Schedule Management Project**

# Team

* Roger Augusto Diaz Tazza (MySQL Expert)
* Rahul Devarajan Raj (MS SQL Server Expert)

# Weekly Meeting Hours

After an initial meeting, we decided to work every **MONDAY** from … **7 PM**…. to … **10 PM**…… via Zoom. Also, we designed a Gantt Chart that will help us to track our progress in the semester, as you can see in the following image:

**Figure 1: Gantt Chart for Instructor Schedule Management Project**

Source: Proper Elaboration. This Gantt Chart will be updated every week.

# Project Description

**Database description**

Consider a database in which data is recorded about the weekly schedule of instructors. The data requirements are summarized as follows:

* Each instructor is identified uniquely by instructor id.
* Each instructor has a name, a title (Dr., Bach, Prof., Ms., Mr.…), an email address, and an office phone number.
* An instructor has one or many schedules, whereas an instructor schedule belongs to one instructor. An instructor schedule is identified uniquely by a schedule id. Also, it has the day in which the instructor teaches lecture sessions, lab sessions, or gives assistance in an office session, and the period it is available for each activity (start and end time). Note: hours are expressed in 24 hours format, as well as there are no half hours, only whole hours (Example: 10:00 not 10:30)).
* An instructor can teach many sections and there cannot be any instructor who does not teach any section. Sections are taught by one instructor. Sections are identified uniquely by section id. Also, they have a semester and a delivery mode related.
* Each section belongs to one course. Each course has many sections and there cannot be any course that does not have any section. Courses are identified uniquely by course id. Also, it has a course name.
* A student must attend one section, whereas a section can be attended by many students. There cannot be any section without students. Note: It is assumed that if the student can attend a section, it is because is enrolled in that section. Students are identified uniquely by student id. Also, they have a name.
* A section may have many rooms available. One room is reserved for classes, another for office time, and for lab classes as well (room type). Also, one room may belong to many sections. A room is identified uniquely by room id and has a location as well.
* Each section and section schedule must have one room. Each room and section have one or many section schedules. Also, each section schedule and room must have 1 section. A section schedule is identified uniquely by schedule id. Also, it has a day, start and end time associated.
* An appointment occurs if there is a match between student availability and instructor office time. So, an instructor may meet with many students, whereas a student may meet with many instructors. Each instructor and one student may have many appointments, whereas an appointment belongs to one instructor and one student. Each appointment is identified uniquely by appointment id. Its haves the date of the appointment and the start and end time.

# Assumptions about Cardinality and Participations

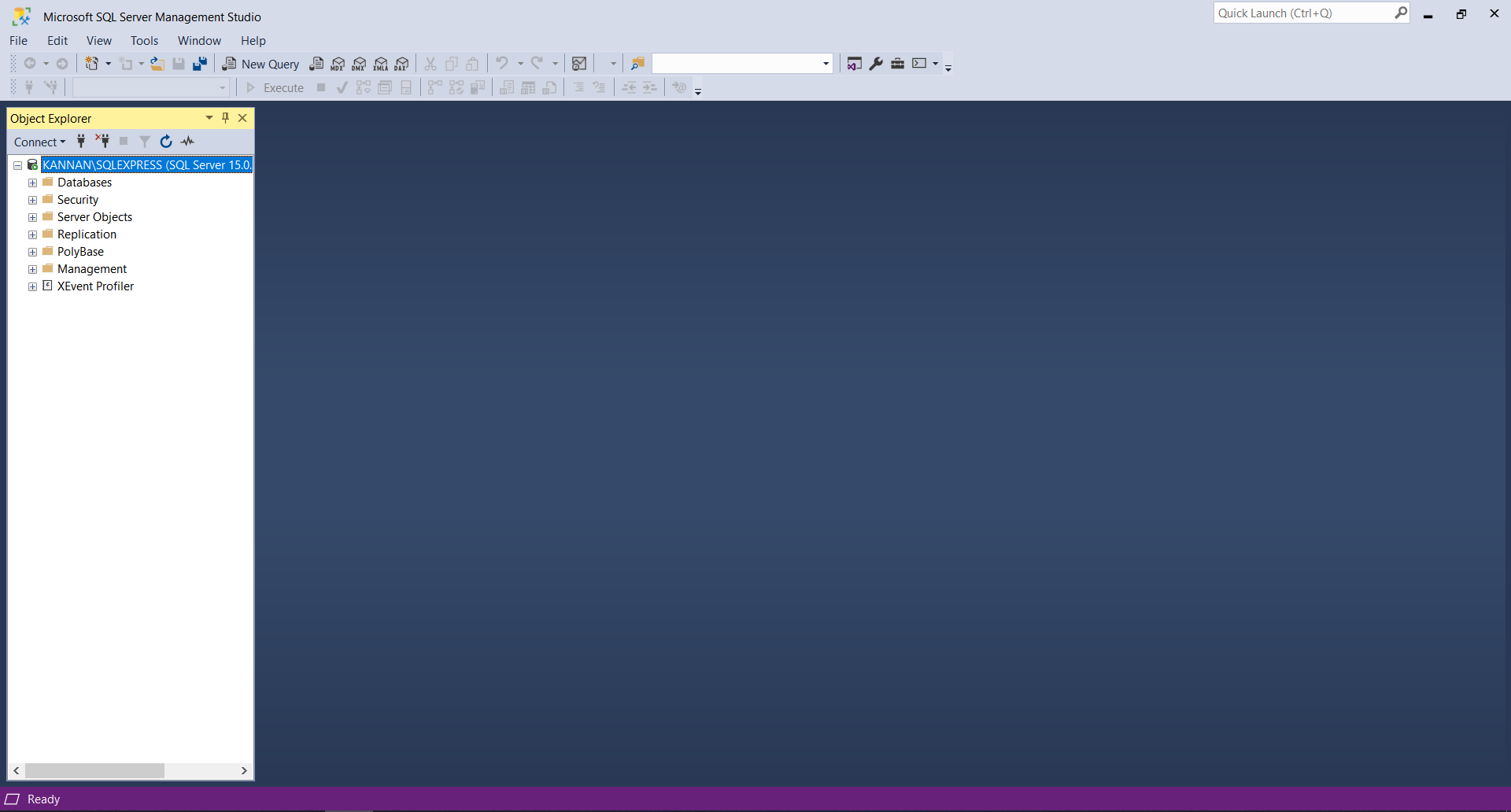
You can write all the assumptions about Cardinality and Participation (total/partial) here.

**Assumptions for Maximum/Minimum Cardinality:**

* One instructor must teach at least **1** section and at most **MANY** sections, whereas a section must be taught by at least **1** instructor and at most **1** instructor.
* One instructor must have at least **1** instructor schedule and at most **MANY** instructor schedules, whereas an instructor schedule must have at least **1** instructor and at most **1** Instructor.
* One section must belong to at least **1** course and at most **1** course, whereas a course must have at least **1** section and at most **MANY** sections.
* One student must attend at least **1** section and at most **1** section, whereas a section must be attended at least by **1** student and at most by **MANY** students.
* One section and one section schedule must have at least **1** room and at most **1** room.
* One room and one section must have at least **1** section schedule and at most **MANY** sections schedules.
* One section schedule and one room must have at least **1** section and at most **1** section.
* One instructor may meet with at least **1** student and at most **MANY** students, whereas a student may meet with at least **1** instructor and at most **MANY** instructors.
* One instructor and one student must have at least **1** appointment and at most **MANY** appointments, whereas an appointment may belong to at least **1** instructor and **1** student and at most **1** instructor and **1** student.

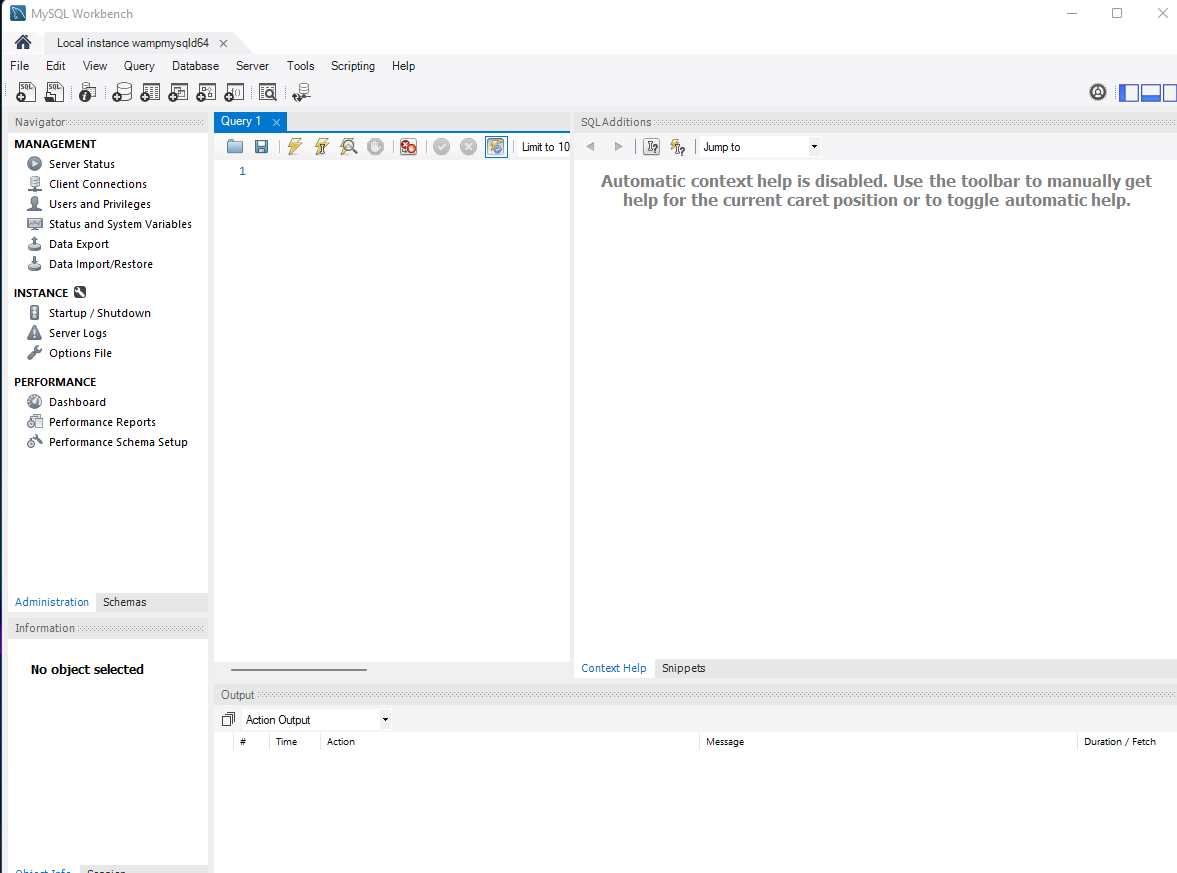
# Screenshots

**Figure 2: MS SQL Screenshot**



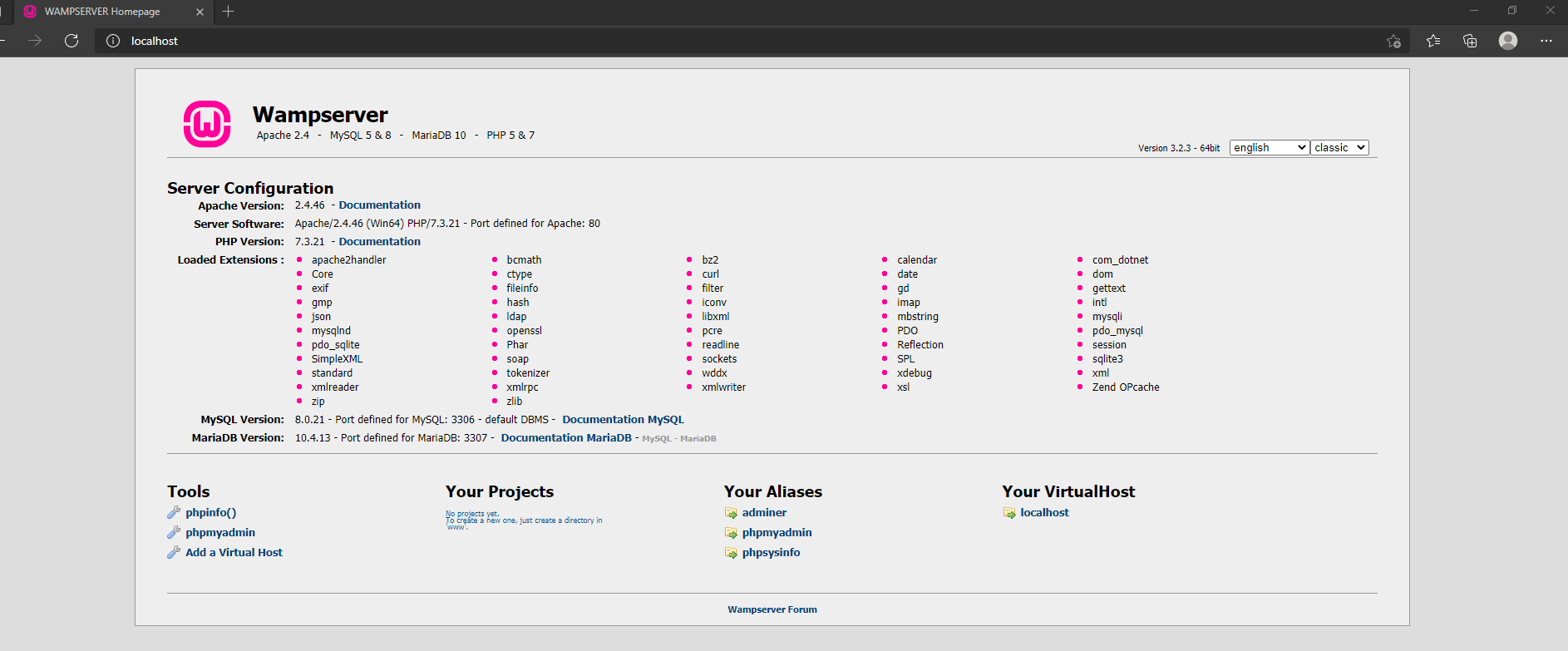
Source: Proper Elaboration.

**Figure 3: MySQL Screenshot**



Source: Proper Elaboration.

**Figure 4: WAMP Installed**



Source: Proper Elaboration.

# EER Modeling Diagram

In the following drawing canvas, EER Modeling shapes have been provided. You can copy and replicate them (Ctrl+C to copy and Ctrl+V to paste. You can also select a shape, then press the Ctrl button and drag and drop to copy a shape) and edit them to build your diagram.

Diagram

Description automatically generated

# ER-Model Mapping to Database Relational Schema

The relational Schema is written here:

* Step 1: Map strong entities (instructor, course, student, section schedule, and room)
* Instructor (InstructorId, InstructorName, OfficePhoneNum, EmailAddress, Title)
* Course (CourseId, CourseName)
* Student (StudentId, StudentName)
* SectionSchedule (SectionScheduleId, SectionScheduleStartTime, SectionScheduleEndTime, SectionScheduleDay)
* Room (RoomId, Location, RoomType)

*\*There are no specializations or generalizations. Derived attributes are not considered because they are not stored in the*

*Database.*

* Step 2: Map weak entities (course-section, instructor-student-appointment, and instructor-instructor schedule)
* Course\_Section (**CourseId**, SectionId, Semester, DeliveryMode)
* Instructor\_InstructorSchedule (**InstructorId**, InstructorScheduleId, Type, ScheduleStartTime, ScheduleEndTime, ScheduleDay)
* Instructor\_Student\_Appointment (**InstructorId, StudentId,** AppoinmentId, AppoinmentDay, AppoinmentStartTime, AppoinmentEndTime)

*\*There is no 1:1 binary relationship.*

* Step 3: Map 1: M binary relationship type (section-student, instructor-section)
* Student (StudentId, StudentName, **SectionId, CourseId**)
* Course\_Section (**CourseId**, SectionId, Semester, DeliveryMode, **InstructorId**)

*\*There are no M: N relationship and recursive relationship.*

* Step 4: Map n-ary relationship type (section-section schedule-room)
* Section\_SectionSchedule\_Room (**(CourseId, SectionId), SectionScheduleID, RoomID**)

*\*There are no multivalued attributes.*

Putting all together:

* Course (CourseId, CourseName)
* SectionSchedule (SectionScheduleId, SectionScheduleStartTime, SectionScheduleEndTime, SectionScheduleDay)
* Room (RoomId, Location, RoomType)
* Instructor (InstructorId, InstructorName, OfficePhoneNum, EmailAddress, Title)
* Instructor\_InstructorSchedule (**InstructorId**, InstructorScheduleId, Type, ScheduleStartTime, ScheduleEndTime, ScheduleDay)
* Course\_Section (**CourseId**, SectionId, Semester, DeliveryMode, **InstructorId**)
* Student (StudentId, StudentName, **SectionId, CourseId**)
* Instructor\_Student\_Appointment (**InstructorId, StudentId,** AppoinmentId, AppoinmentDay, AppoinmentStartTime, AppoinmentEndTime)
* Section\_SectionSchedule\_Room (**(CourseId, SectionId), SectionScheduleID, RoomID**)

# Normalization

All relations must be normalized up to BCNF. You must explain why you believe every relation in your database in normalized.

1. Course (CourseId, CourseName)

* **Check 1NF:**
* The Course relation has a primary key which is composed by CourseId.
* CourseName is an atomic attribute because there is only one single name for each course.
* **Because all attributes are atomic and no multiple valued attributes exist in the Course relation, we define it is already in 1NF.**
* **Check 2NF:**
* Course relation is already in 1NF.
* Because CourseId is a single key, we do not need to analyze the 2NF for this relation.
* **Because there is no composite key, the Course relation is already in 2NF.**
* **Check 3NF:**
* Course relation is already in 2NF.
* There are only two attributes, so we cannot identify any transitivity dependency. Also, there is not any non-prime attribute determined by another non-prime attribute.
* **Because there is no transitive dependency and non-prime** **attribute is determined by another non-prime attribute, the Course relation is already in 3NF.**

1. SectionSchedule (SectionScheduleId, SectionScheduleStartTime, SectionScheduleEndTime, SectionScheduleDay)

* **Check 1NF:**
* The SectionSchedule relation has a primary key which is composed by SectionScheduleId.
* SectionScheduleStartTime, SectionScheduleEndTime and SectionScheduleDay are atomic attributes because there cannot be more than one start, end time and day for each section session at the same time.
* **Because all attributes are atomic and no multiple valued attributes exist in the SectionSchedule relation, we define it is already in 1NF.**
* **Check 2NF:**
* SectionSchedule relation is already in 1NF.
* Because SectionScheduleId is a single key, we do not need to analyze the 2NF for this relation.
* **Because there is no composite key, the SectionSchedule relation is already in 2NF.**
* **Check 3NF:**
* SectionSchedule relation is already in 2NF.
* The following functional dependency is identified: SectionScheduleId - > (SectionScheduleStartTime, SectionScheduleEndTime, SectionScheduleDay). There are not transitive functional dependencies identified. Also, there is not any non-prime attribute determined by another non-prime attribute.
* **Because there is no transitive dependency and non-prime attribute is determined by another non-prime attribute, the SectionSchedule relation is already in 3NF.**

1. Room (RoomId, Location, RoomType)

* **Check 1NF:**
* The Room relation has a primary key which is composed by RoomId.
* A room cannot be in different locations (faculty buildings) at the same time, so location is an atomic attribute. Also, each room can be either a lab room, office room, or classroom, but it can't be any of them at the same time. So, RoomType is also an atomic attribute.
* **Because all attributes are atomic and no multiple valued attributes exist in the Room relation, we define it is already in 1NF.**
* **Check 2NF:**
* Room relation is already in 1NF.
* Because RoomId is a single key, we do not need to analyze the 2NF for this relation.
* **Because there is no composite key, the Room relation is already in 2NF.**
* **Check 3NF:**
* Room relation is already in 2NF.
* The following functional dependency is identified: RoomId - > (Location, RoomType). There are not transitive functional dependencies identified. Also, there is not any non-prime attribute determined by another non-prime attribute.
* **Because there is no transitive dependency and non-prime attribute is determined by another non-prime attribute, the Room relation is already in 3NF.**

1. Instructor (InstructorId, InstructorName, OfficePhoneNum, EmailAddress, Title)

* **Check 1NF:**
* The Instructor relation has a primary key which is composed by InstructorId.
* We are considering the instructor’s name as a whole name even if it is composed by first and last name, and an instructor cannot have more than one name at the same time, so InstructorName is an atomic attribute. Also, we are assuming that an instructor can have only one phone number and one email address at the same time. For that reason, OfficePhoneNum and EmailAddress are atomic attributes. In the case of Title, even if an instructor may have many titles in their background (Bach, PhD, or Magister), we are storing the highest degree achieved. So, Title is an atomic attribute as well.
* **Because all attributes are atomic and no multiple valued attributes exist in the Instructor relation, we define it is already in 1NF.**
* **Check 2NF:**
* Instructor relation is already in 1NF.
* Because InstructorId is a single key, we do not need to analyze the 2NF for this relation.
* **Because there is no composite key, the Instructor relation is already in 2NF.**
* **Check 3NF:**
* Instructor relation is already in 2NF.
* The following functional dependencies are identified: InstructorId -> (InstructorName, OfficePhoneNum, EmailAddress, Title); OfficePhoneNum -> (InstructorId, InstructorName, EmailAddress, Title); and EmailAddress - > (InstructorId, InstructorName, OfficePhoneNum, Title). As OfficePhoneNum and EmailAddress can be candidate keys, they can be prime attributes. So, we do not consider them for the transitivity dependencies. Also, there is not any non-prime attribute determined by another non-prime attribute.
* **Because there is no transitive dependency and non-prime attribute is determined by another non-prime attribute, the Instructor relation is already in 3NF**

1. Instructor\_InstructorSchedule (**InstructorId**, InstructorScheduleId, Type, ScheduleStartTime, ScheduleEndTime, ScheduleDay)

* **Check 1NF:**
* The Instructor\_InstructorSchedule relation has a composite primary key which is composed by InstructorId and InstructorScheduleId.
* Each instructor schedule is assigned for many types (Lab, Office, and Classroom), but not at the same time. So, Type is an atomic attribute. ScheduleStartTime, ScheduleEndTime and ScheduleDay are atomic attributes because there cannot be more than one start, end time and day for each instructor session at the same time.
* **Because all attributes are atomic and no multiple valued attributes exist in the Instructor\_InstructorSchedule relation, we define it is already in 1NF.**
* **Check 2NF:**
* Instructor-InstructorSchedule relation is already in 1NF.
* There are not any partial functional dependencies identified. All no-prime attributes are determined by only the entire primary key rather than a part of the primary key.
* **Because there are no partial functional dependencies, the Instructor-InstructorSchedule relation is already in 2NF.**
* **Check 3NF:**
* Instructor-InstructorSchedule relation is already in 2NF.
* The following functional dependency is identified: (InstructorId, InstructorScheduleId) - > (Type, ScheduleStartTime, ScheduleEndTime, ScheduleDay). There are not transitive functional dependencies identified. Also, there is not any non-prime attribute determined by another non-prime attribute.
* **Because there is no transitive dependency and non-prime attribute is determined by another non-prime attribute, the Instructor-InstructorSchedule relation is already in 3NF.**

1. Course\_Section (**CourseId**, SectionId, Semester, DeliveryMode, **InstructorId**)

* **Check 1NF:**
* The Course\_Section relation has a composite primary key which is composed by CourseId and SectionId.
* Each semester can have only one name (Winter 2022, Fall 2021, etc.) at a time. So, Semester is an atomic attribute. We are assuming each section in a course can have only one delivery mode at the same time, either semi presential, presential or online. So, DeliveryMode is an atomic attribute. Each instructor is identified but only one Id. There cannot be one instructor with more than one identification. So, InstructorId is an atomic attribute.
* **Because all attributes are atomic and no multiple valued attributes exist in the Course-Section relation, we define it is already in 1NF.**
* **Check 2NF:**
* Course\_Section relation is already in 1NF.
* There are not any partial functional dependencies identified. All no-prime attributes are determined by only the entire primary key rather than a part of the primary key.
* **Because there are no partial functional dependencies, the Course\_Section relation is already in 2NF.**
* **Check 3NF:**
* Course\_Section relation is already in 2NF.
* The following functional dependency is identified: (CourseId, SectionId) - > (Semester, DeliveryMode, InstructorId). There are not transitive functional dependencies identified. Also, there is not any non-prime attribute determined by another non-prime attribute.
* **Because there is no transitive dependency and non-prime attribute is determined by another non-prime attribute, the Course-Section relation is already in 3NF.**

1. Student (StudentId, StudentName, **SectionId, CourseId**)

* **Check 1NF:**
* The Student relation has a primary key which is composed by StudentId.
* We are considering the student’s name as a whole name even if it is composed by first and last name, and a student cannot have more than one name at the same time, so StudentName is an atomic attribute. Also, we are assuming that each section of a course and the course itself cannot have two identifiers at the same time. For that reason, SectionId and CourseId are atomic attributes.
* **Because all attributes are atomic and no multiple valued attributes exist in the Student relation, we define it is already in 1NF.**
* **Check 2NF:**
* Student relation is already in 1NF.
* Because StudentId is a single key, we do not need to analyze the 2NF for this relation.
* **Because there is no composite key, the Student relation is already in 2NF.**
* **Check 3NF:**
* Student relation is already in 2NF.
* The following functional dependency is identified: (StudentId) - > (StudentName, SectionId, CourseId). There are not transitive functional dependencies identified. Also, there is not any non-prime attribute determined by another non-prime attribute.
* **Because there is no transitive dependency and non-prime attribute is determined by another non-prime attribute, the Student relation is already in 3NF.**

1. Instructor\_Student\_Appointment (**InstructorId, StudentId,** AppoinmentId, AppoinmentDay, AppoinmentStartTime, AppoinmentEndTime)

* **Check 1NF:**
* The Instructor\_Student\_Appointment relation has a composite primary key which is composed by InstructorId, StudentId and AppointmentId.
* AppoinmentDay, AppoinmentStartTime and AppoinmentEndTime are atomic attributes because there cannot be more than one start, end time and day for each appointment session at the same time.
* **Because all attributes are atomic and no multiple valued attributes exist in the Instructor\_Student\_Appointment relation, we define it is already in 1NF.**
* **Check 2NF:**
* Instructor\_Student\_Appointment relation is already in 1NF.
* There are not any partial functional dependencies identified. All no-prime attributes are determined by only the entire primary key rather than a part of the primary key.
* **Because there are no partial functional dependencies, the Instructor\_Student\_Appointment relation is already in 2NF.**
* **Check 3NF:**
* Instructor\_Student\_Appointment relation is already in 2NF.
* The following functional dependency is identified: (InstructorId, StudentId, AppointmentId) - > (AppoinmentDay, AppoinmentStartTime, AppoinmentEndTime). There are not transitive functional dependencies identified. Also, there is not any non-prime attribute determined by another non-prime attribute.
* **Because there is no transitive dependency and non-prime attribute is determined by another non-prime attribute, the Instructor\_Student\_Appointment relation is already in 3NF.**

1. Section\_SectionSchedule\_Room (**(CourseId, SectionId), SectionScheduleID, RoomID**)

* **Check 1NF:**
* The Section\_SectionSchedule\_Room relation has a composite primary key which is composed by CourseId, SectionId, SectionScheduleId and RoomId.
* The attributes of this relation are keys, and they have already only one value at the same time. So, all are atomic attributes.
* **Because all attributes are atomic and no multiple valued attributes exist in the Section\_SectionSchedule\_Room relation, we define it is already in 1NF.**
* **Check 2NF:**
* Section\_SectionSchedule\_Room relation is already in 1NF.
* As there are not any no-prime attributes in this relation, no partial functional dependencies can be identified.
* **Because there are no partial functional dependencies, the Section\_SectionSchedule\_Room relation is already in 2NF.**
* **Check 3NF:**
* Section\_SectionSchedule\_Room relation is already in 2NF.
* As there are noy any no-prime attributes in this relation, no transitive functional dependencies can be identified.
* **Because there is no transitive dependency and non-prime attribute is determined by another non-prime attribute, the Section\_SectionSchedule\_Room relation is already in 3NF.**

# Determining Data Types (Domain) and Constraints

You explain why you choose a certain data type for a field and why you apply certain constraints

1. **Course Table**

CourseId – INT PRIMARY KEY – As CourseId is a whole number and uniquely identify a course. We specify PRIMARY KEY to maintain the entity integrity constraint. Example 2200,2300

CourseName – VARCHAR (40) NOT NULL UNIQUE – Used to denote a course name and it is not null and unique, because it cannot exist one course with no name or two names at the same time. Example: Database, Introduction to Programming

1. **SectionSchedule Table**

SectionScheduleId – INT PRIMARY KEY- SectionScheduleId is a whole number and uniquely identify a section session. We specify PRIMARY KEY to maintain the entity integrity constraint. Example: 1, 2 ,3, 4, etc.

SectionScheduleStartTime - TIME (0) NOT NULL – Used to denote section session start time and is not null, because there cannot exist any section session without a start time. Example: 10:00:00

SectionScheduleEndTime - TIME (0) NOT NULL – Used to denote section session end time and is not null, because there cannot exist any section session without an end time. Example: 14:00:00

SectionScheduleDay – VARCHAR (15) NOT NULL – Used to denote section session day and is not null, because there cannot exist any section session without a day. Example: Monday

1. **Room Table**

RoomId – INT PRIMARY KEY – RoomId is a whole number and uniquely identify a room. We specify PRIMARY KEY to maintain the entity integrity constraint. Example: 404, 500, 200 or room 1000.

Location - CHAR (1) NOT NULL -Used to denote faculty building and cannot be null, because there cannot exist any room without a physical location. Example: A, B, C, or D

RoomType - CHAR (1) NOT NULL – Used to denote the room type and cannot be null, because there cannot exist any room that is not used for any lab, office, or class purposes. Example: L for lab, O for office and C for class.

1. **Instructor Table**

InstructorId – INT PRIMARY KEY – InstructorId is a whole number and uniquely identify an instructor. We specify PRIMARY KEY to maintain the entity integrity constraint. Example: 4001 (4 digits as StudentId but start with 4)

InstructorName - VARCHAR (40) NOT NULL – Used to denote the instructor’s name and cannot be null, because there cannot exist any instructor without a name. We are not considered UNIQUE constraint because it can exist two or more instructors with the same name. Example: Rahul Smith

OfficePhoneNum - CHAR (12) NOT NULL UNIQUE, -Used to denote instructor office phone number and cannot be null, because there cannot exist any instructor without an office phone number. It is also unique because we are assuming there is no instructor with more than one office phone number at the same time. Example: (778) 999-8756 (Assuming there are no international numbers, extensions, etc.).

EmailAddress - VARCHAR (50) NOT NULL UNIQUE, -Used to denote instructor email address and cannot be null, because there cannot exist any instructor without an email address. Also, it needs to be UNIQUE, because we are assuming there is no instructor with more than one email address at the same time. Example: john@gmail.com

Title - CHAR (3) -Used to denote instructor title. Example: MG (for magister), BA (for bachelor), PHD (for doctor).

1. **Instructor\_InstructorSchedule Table**

InstructorId – ID - Used to identify the instructor. Example: 4001(4 digits as students but start with 4)

InstructorScheduleId - INT- Used to identify the instructor schedule. Example: 40011

InstructorScheduleType - CHAR (1) NOT NULL – Used to denote instructor schedule type and is not null, because there cannot be any instructor schedule that is not assigned to any lab, office, or class purposes. Example: L for lab, O for office and C for class.

InstructorScheduleStartTime - TIME (0) NOT NULL - Used to denote instructor session start time and is not null, because there cannot exist any instructor session without a start time. Example: 10:00:00

InstructorScheduleEndTime - TIME (0) NOT NULL - Used to denote instructor session end time and is not null, because there cannot exist any instructor session without an end time. Example: 14:00:00

InstructorScheduleDay - VARCHAR (15) NOT NULL - Used to denote instructor session day and is not null, because there cannot exist any instructor session without a day. Example: Monday

We used FOREIGN KEY (InstructorId) REFERENCES Instructor (InstructorId) to maintain the reference integrity constraint between both tables and to be sure that there is no orphan record in the instructor schedule table. And we specify ON DELETE CASCADE, to maintain the foreign key rule in the foreign key of the Instructor Schedule table if the correspond parent record in the Instructor table is deleted. We specify ON DELETE UPDATE, to maintain the foreign key rule in the foreign key of the Instructor Schedule Table if the correspond parent record in the Instructor table is updated.

Both InstructorId and InstructorScheduleId composed the primary key and are used to identify uniquely the instructor schedule and to maintain the entity integrity constraint.

1. **Course\_Section Table**

CourseId - INT - Used to identify the course. Example: 2200, 2300

SectionId - INT - Used to identify the section. Example: 1,2 and 3 (There cannot be more than 3 sections per course).

Semester - VARCHAR (10) – Used to denote the semester which belongs the section and course. Example: SUMMER2021, FALL2021, WINTER2021, etc.

DeliveryMode - VARCHAR (15) – Used to denote the delivery mode of each section and course. Example: Online, Presential, Semi-Presential.

InstructorId - INT - Used to identify the instructor. Example: 4001 (4 digits as students but start with 4)

We used FOREIGN KEY (InstructorId) REFERENCES Instructor (InstructorId) and FOREIGN KEY (CourseId) REFERENCES Course (CourseId) to maintain the reference integrity constraint between those tables and to be sure that there is no orphan record in the course section table. And we specify ON DELETE CASCADE when we refer to the Course table and ON DELETE SET NULL when we refer the Instructor table, to maintain the foreign key rule in the foreign keys of the Course Section table if the correspond parent record in the Instructor or Course table is deleted. We specify ON DELETE UPDATE, to maintain the foreign key rule in the foreign key of the Course Section table if the correspond parent record in the Instructor table is updated.

Both CourseId and SectionId composed the primary key and are used to identify uniquely the section of any course and to maintain the entity integrity constraint.

1. **Student Table**

StudentId - INT – PRIMARY KEY- StudentId is a whole number and uniquely identify a student. We specify PRIMARY KEY to maintain the entity integrity constraint. Example: 3003 (4 digits as instructors but start with 3)

StudentName - VARCHAR (40) NOT NULL – Used to denote the student’s name and is not null, because there cannot be any student without a name. We are not considered UNIQUE constraint because it can exist two or more students with the same name. Example: Diego Perez

SectionId - INT - Used to identify the section. Example: 1,2 and 3.

CourseId - INT - Used to identify the course. Example: 2200, 2300

We used FOREIGN KEY (CourseId, SectionId) REFERENCES Course\_Section (CourseId, SectionId) to maintain the reference integrity constraint between both tables and to be sure that there is no orphan record in the student table. And we specify ON DELETE SET NULL, to maintain the foreign key rule in the foreign key of the Student table if the correspond parent record in the Course\_Section table is deleted.

1. **Instructor\_Student\_Appointment Table**

InstructorId - INT – Used to identify the instructor. Example: 4001 (4 digits as students but start with 4)

StudentId - INT - Used to identify the student. - Example: 3003 (4 digits as instructors but start with 3)

AppointmentId - INT – Used to identify the appointment session-- Example: 1, 2, 3 ,4 ...

AppoinmentDay - VARCHAR (15) NOT NULL - Used to denote an appointment session day and is not null, because there cannot exist any appointment session without a day-- Example: Monday

AppoinmentStartTime - TIME (0) NOT NULL - Used to denote an appointment session start time and is not null, because there cannot exist any appointment session without a start time. Example: 10:00:00

AppoinmentEndTime - TIME (0) NOT NULL - Used to denote an appointment session end time and is not null, because there cannot exist any appointment session without an end time. Example: 14:00:00

We used FOREIGN KEY (InstructorId) REFERENCES Instructor (InstructorId) and FOREIGN KEY (StudentId) REFERENCES Student (StudentId) to maintain the reference integrity constraint between those tables and to be sure that there is no orphan record in the Instructor\_Student\_Appointment table. And we specify ON DELETE CASCADE in both references, to maintain the foreign key rule in the foreign keys of the Instructor\_Student\_Appointment table if the correspond parent record in the Instructor or Student table is deleted. We specify ON DELETE UPDATE, to maintain the foreign key rule in the foreign key of the Instructor Student Appointment table if the correspond parent record in the Instructor table is updated.

InstructorId, StudentId, and AppointmentId composed the primary key and are used to identify uniquely the appointment session and to maintain the entity integrity constraint.

1. **Section\_SectionSchedule\_Room Table**

CourseId - INT - Used to identify the course. Example: 2200, 2300

SectionId - INT - Used to identify the section. Example: 1,2 and 3 (There cannot be more than 3 sections per course).

SectionScheduleId – INT- Used to identify the section session. Example: 1, 2 ,3, 4, etc.

RoomId - INT- Used to identify the room. Example: 404, 500, 200 or room 1000.

We used FOREIGN KEY (CourseId, SectionId) REFERENCES Course\_Section (CourseId, SectionId), FOREIGN KEY (SectionScheduleId) REFERENCES SectionSchedule (SectionScheduleId), and FOREIGN KEY (RoomId) REFERENCES Room (RoomId) to maintain the reference integrity constraint between all the tables and to be sure that there is no orphan record in the Section\_Schedule\_Room table. And we specify ON DELETE CASCADE in all references, to maintain the foreign key rule in the foreign keys of the Section\_Schedule\_Room table if the correspond parent record in the Course\_Section, SectionSchedule or Room table is deleted.

CourseId, SectionId, SectionScheduleId, and RoomId composed the primary key and are used to identify uniquely the section schedule room table and to maintain the entity integrity constraint.

# Creating Database and Tables - SQL DDL

You do not need to copy SQL commands here. Save your SQL commands in a script file and just mention the name of the file here. Make sure the script file is stored besides this document within the same folder.

The commands for creating the database and the tables are stored in the script file with name **“Project\_CreationTables”.**

The commands for dropping the tables are stored in the script file with name **“Project\_DroppingTables”.**

# Inserting Values in Tables

You do not need to copy SQL commands here. Save your SQL commands in a script file and just mention the name of the file here. Make sure the script file is stored beside this document within the same folder.

The commands for inserting values to the tables are stored in the script file with name **“Project\_InsertionValues”.**

# SQL Queries

You do not need to copy SQL commands here. Save your SQL commands in a script file and just mention the name of the file here. Make sure the script file is stored beside this document within the same folder.

The commands for querying are stored in the script file with name **“Project\_Queries”.**

# Views

You do not need to copy SQL commands here. Save your SQL commands in a script file and just mention the name of the file here. Make sure the script file is stored beside this document within the same folder.

The commands for querying are stored in the script file with name **“Project\_Queries”.**