

E-Learning System Database

CPS 510
Assignment 10 – Project Documentation
Professor A.Abhari
Jasdeep Sekhon 501006972
Christopher Ingham 500707888

Why did we choose this DBMS?

Our application for CPS 510 will be implementing an E-Learning System also known as a Learning Management System. We chose this idea because we are interested in e-learning and have become very accustomed to this throughout COVID. E-learning is the future of education, and we are excited to learn more about DBMS and use our learning to implement our own model that incorporates many learning models used today.

"The *Research Institute of America* found that eLearning **increases retention rates 25% to 60%** while retention rates of face-to-face training are very low in comparison: 8% to 10%. This is because with eLearning students have more control over the learning process as well as the opportunity to revisit the training as needed". (https://www.shiftelearning.com/blog/bid/301248/15-facts-and-stats-that-reveal-the-power-of-elearning).

About DBMS

Udemy D2L | Brightspace

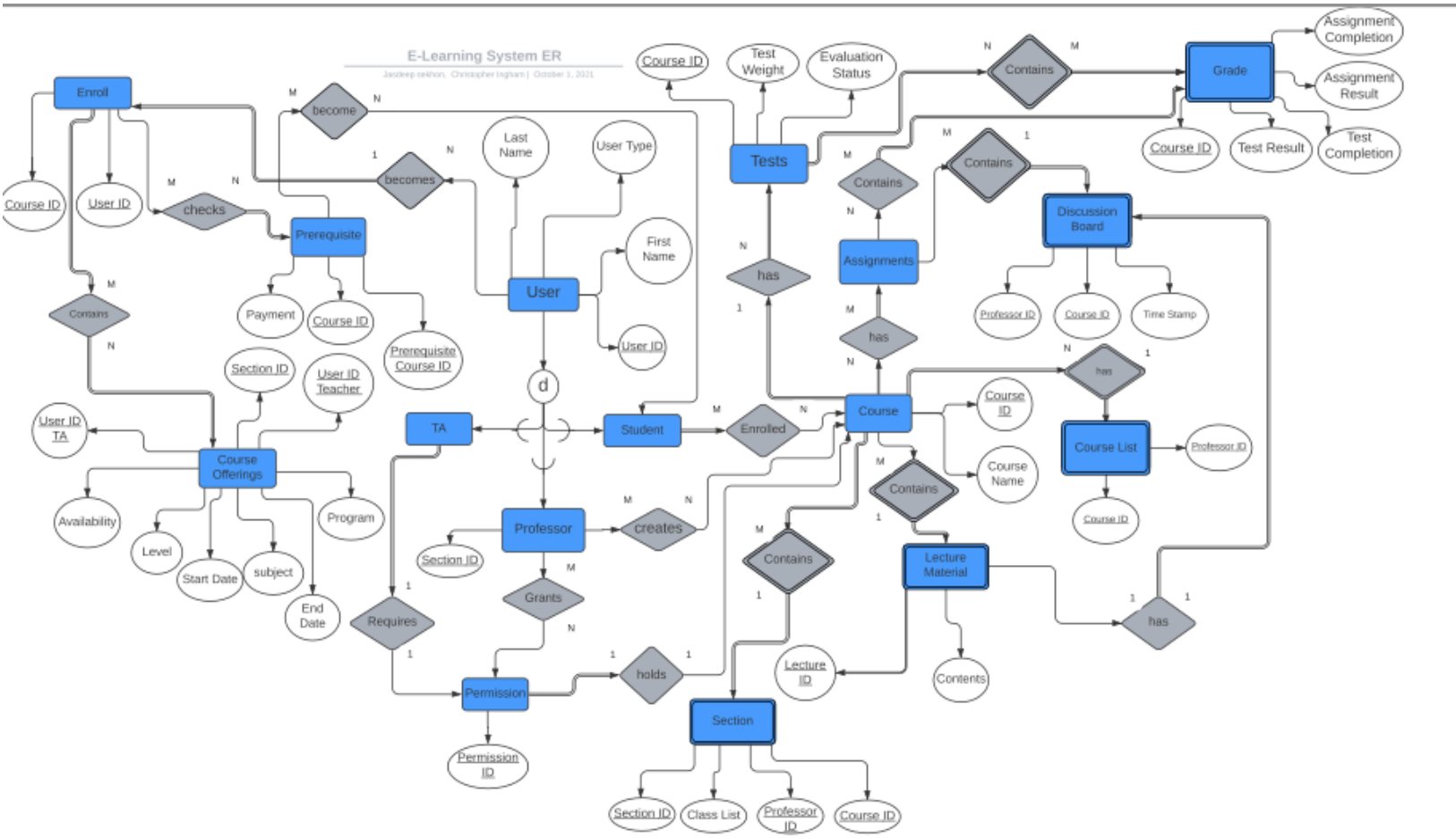
Our e-learning system has many functions that are interrelated but also serve different purposes in the DBMS. Our users will be Students, TA, and Teachers. Students will have usernames, access to specific assignments, submissions, course details, course material and grading. TA's will only have access to the courses they're enrolled in and only students who are a part of their sections. They will also only have access to assignments and grading pertaining to their section, not the course. Teachers will have access to most things, such as SectionID, UserID, Permissions, Course offerings, Discussion board for assignments, Discussion board for lectures, Course term and grading. There will also be an enroll function for UserID, CourseID, SectionType and Payment. Assignments will have CourseID, description and deadlines.

We want to make this e-learning site like EDX or Udemy. Our inspiration for choosing this application was specifically because we want to make a similar complex database which incorporates so many different functions, but it is an important database for students. We want our database system to contain many different courses by subject. Each of these courses will have students, TAs and professors and will function like a real course in a virtual environment. There will be assignments, quizzes, course material, discussions, lecture videos, grades, and these will be managed by either the prof or the TA depending on which section the professors and TAs are responsible for.

Table of Contents

Preparing the ER Model

The next step in our design was to prepare our ER model. This is one of the most crucial components of our implementation and design as it sets the tone for how our DBMS will function and form relations with different components of our System.



Schema Design

```
/*
Main function that ever other type uses as a main building block
Turns into Student, Professor and TA
*/
CREATE TABLE first_user(
    userID NUMBER PRIMARY KEY,
    userType NUMBER,
    firstName VARCHAR2 (50 CHAR),
    lastName VARCHAR2 (50 CHAR)
);

/*
Has courseID that is used in many methods
Has permissions that need to be met by TA and Prof permissions to be able teach
*/
CREATE TABLE Course(
    courseID NUMBER PRIMARY KEY NOT NULL,
    taPermissionCourse NUMBER NOT NULL,
    professorPermissionCourse NUMBER NOT NULL,
    courseName VARCHAR2 (50 CHAR)
);

/*
Has a ID
has userId that can be accessed to see name
*/
CREATE TABLE Student(
    studentID NUMBER PRIMARY KEY,
    userID NUMBER,
    FOREIGN KEY (userID) REFERENCES first_user(userID)
);
```

```
/*
Has a ID
has userId that can be accessed to see name
*/
CREATE TABLE Professor(
    professorID NUMBER PRIMARY KEY,
    userID NUMBER,
    FOREIGN KEY (userID) REFERENCES first_user(userID)
);

/*
Has a ID
has userId that can be accessed to see name
*/

CREATE TABLE TA(
    taID NUMBER PRIMARY KEY,
    userID NUMBER,
    FOREIGN KEY (userID) REFERENCES first_user(userID)
);
```

```
/*
Permission has two seperate keys, ta permissions and prof permissions
inherites TA ID and Professor ID and matches them to a courseID, and permission
Permission inherites CourseID so it can check the permission needed by TA and Prof for specific course
*/
CREATE TABLE Permission(
    profPermissionID NUMBER NOT NULL,
    taPermissionID NUMBER NOT NULL,
    professorID NUMBER,
    FOREIGN KEY (professorID) REFERENCES Professor(professorID),
    taID NUMBER,
    FOREIGN KEY (taID) REFERENCES TA(taID),
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID)
);

/*
This prequistie checks for certain course ID if the prerequite is satisfied
*/
CREATE TABLE Prerequisite(
    prerequisiteSatisfied VARCHAR2(5) PRIMARY KEY NOT NULL,
    payment NUMBER NOT NULL,
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID)
);
```

```
/*
Section ID shows the class list, who is teaching, who is ta and courseid
*/
CREATE TABLE Section(
    sectionID NUMBER PRIMARY KEY NOT NULL,
    classList VARCHAR2 (1000 CHAR),
    professorID NUMBER,
    FOREIGN KEY (professorID) REFERENCES Professor(professorID),
    taID NUMBER,
    FOREIGN KEY (taID) REFERENCES TA(taID),
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID)
);

/*
Contains Lecture Contents
*/
CREATE TABLE Lecture_Material(
    lectureID NUMBER PRIMARY KEY NOT NULL,
    lectureContents VARCHAR2 (1000 CHAR)
);

/*
Shows what Professors are teaching what courses
*/
CREATE TABLE Course_List(
    professorID NUMBER,
    FOREIGN KEY (professorID) REFERENCES Professor(professorID),
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID)
);

/*
Used to check score of specific test for a course for certain student
*/
CREATE TABLE Tests(
    testID NUMBER PRIMARY KEY NOT NULL,
    testWeight NUMBER NOT NULL,
    testEvaluationStatus VARCHAR2 (10 CHAR),
    sectionID NUMBER,
    FOREIGN KEY (sectionID) REFERENCES Section(sectionID),
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID),
    studentID NUMBER,
    FOREIGN KEY (studentID) REFERENCES Student(studentID)
);

/*
Used to check score of specific Assignment for a course for certain student
*/
CREATE TABLE Assignments(
    assignmentID NUMBER PRIMARY KEY NOT NULL,
    assignmentWeight NUMBER NOT NULL,
    assignmentEvaluationStatus VARCHAR2 (10 CHAR),
    sectionID NUMBER,
    FOREIGN KEY (sectionID) REFERENCES Section(sectionID),
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID),
    studentID NUMBER,
    FOREIGN KEY (studentID) REFERENCES Student(studentID)

);
```

Schema Design

```
/*
Says if the assignment/test has been completed and gives result
For each course and student
*/
CREATE TABLE Grades(
    assignmentCompletion VARCHAR2 (10 CHAR),
    assignmentResult NUMBER,
    testCompletion VARCHAR2 (10 CHAR),
    testResult NUMBER,
    sectionID NUMBER,
    FOREIGN KEY (sectionID) REFERENCES Section(sectionID),
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID),
    studentID NUMBER,
    FOREIGN KEY (studentID) REFERENCES Student(studentID)
);

/*
Discussion board for a certain section and certain section,
Also for certain assignments
*/
CREATE TABLE Discussion_Board(
    dicussionContent VARCHAR2(1000 CHAR),
    sectionID NUMBER,
    FOREIGN KEY (sectionID) REFERENCES Section(sectionID),
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID) ,
    assignmentID NUMBER,
    FOREIGN KEY (assignmentID) REFERENCES Assignments(assignmentID) ON DELETE CASCADE
);
/*
Course offerings basically shows based on prereqs for each students
what courses can be taken and what they are exactly.
*/
CREATE TABLE Course_Offerings(
    availablity VARCHAR2 (50 CHAR),
    levelOfCourse VARCHAR2 (50 CHAR),
    subject VARCHAR2 (50 CHAR),
    program VARCHAR2 (50 CHAR),
    professorID NUMBER,
    FOREIGN KEY (professorID) REFERENCES Professor(professorID),
    taID NUMBER,
    FOREIGN KEY (taID) REFERENCES TA(taID),
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID)
);

/*
Below tables are for relations that signify the many to many relationships
*/

CREATE TABLE enrolled (
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID),
    studentID NUMBER,
    FOREIGN KEY (studentID) REFERENCES Student(studentID)
);


CREATE TABLE checks (
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID),
    prerequisiteSatisfied VARCHAR2 (5 CHAR) NOT NULL,
    FOREIGN KEY(prerequisiteSatisfied) REFERENCES Prerequisite(prerequisiteSatisfied)
);

CREATE TABLE checks (
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID),
    prerequisiteSatisfied VARCHAR2 (5 CHAR) NOT NULL,
    FOREIGN KEY(prerequisiteSatisfied) REFERENCES Prerequisite(prerequisiteSatisfied)
);

CREATE TABLE creates (
    professorID NUMBER NOT NULL,
    FOREIGN KEY (professorID) REFERENCES Professor(professorID),
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID)
);
```

First_user table

- Contains primary key userID which will be used in the student, TA, and professor table
- userType refers to the number assigned to the TA, student, or professor
- firstName and lastName are attributes used to provide the names for the users

Course table

- courseID is the primary key in this table as it provides the specific course code for each of the courses and can be accessed by many of the tables in the schema design
- taPermissionCourse grants or denies ta permission to the course as well as the professorPermissionCourse
- courseName is the name of the course which can be expressed in char

Student table

- studentID is primary key which will distinguish each student from one another
- userID is the ID which was assigned to the user when they first created the account

Professor table

- professorID is primary key which will distinguish each professor from one another
- userID is the ID which was assigned to the user when they first created the account

TA table

- taID is primary key which will distinguish each ta from one another
- userID is the ID which was assigned to the user when they first created the account

Permission table

- Permission has two separate keys, ta permissions and professor permissions
- Inherits TA ID and professor ID and matches them to a courseID and permission
- Permission inherits courseID so it can check the permissions needed by TA and professor for specific course

Prerequisite table

- prerequisiteID is a key which will check if prerequisite is satisfied
- Payment is the fee for each course
- courseID is the course we are evaluating prerequisite for

Schema Design

Section Table
- sectionID is primary key for each section of course
- classList provides list of students registered in course
- professorID is the ID of the professor who oversees the course and has access and controls the course
- taID grants permission for TA for specific section to view assignments and marks
- courseID is the course the section is for

Lecture_Material Table
- lectureID is the primary key for the lecture material and determines lecture contents
- lecture content attribute contains lecture material

Course_List Table
- professorID shows details of professor who is teaching course
- courseID is the specific course the professor is teaching

Tests Table
- testID is primary key and determines different attributes such as testWeight, testEvaluationStatus, and sectionID
- courseID determines sectionID
- studentID is used to determine if student complete test and their grade

Assignments Table
- assignmentID is primary key and determines different attributes such as assignmentWeight, assignmentEvaluationStatus and sectionID
- courseID determines sectionID
- studentID is used to determine if student completed assignment and their grade

Grades table
- assignmentCompletion determines if student completed assignment
- assignmentResult provides grade
- testCompletion/testResult are same as above
- sectionID determines which section student belongs to and courseID determines course

Discussion Board Table
- Contains discussionContent where students will be posting comments on course material
- Each discussion board will be for a specific section in the course
- assignmentID will determine which Discussion board students are posting too

Course_Offerings Table
- provides course offerings based on prereqs for each student.
- Availability – fall or winter
- levelofCourse – first, second, third or fourth year
- program – specific to students' program
- professorID – professor teaching course
- taID – ta teaching course
- courseID – contains all course information

Enrolled Table
- courseID – contains all course information
- studentID – student uses this ID to enroll

Checks Table
- courseID – contains all course information
- prerequisiteSatisfied – checks if prereq is satisfied

Creates Table
- professorID – professor teaching course
- courseID – contains all course information

Simple Queries

Resulting Tables

```
/*
Queries: Table 1
*/
SELECT *
FROM first_user
WHERE userType =1;

SELECT *
FROM first_user
WHERE userType =2;

SELECT *
FROM first_user
WHERE userType =3;

SELECT *
FROM first_user
ORDER BY userType DESC;

/*
Queries: Table 2
*/

SELECT CourseID, Coursename
FROM   Course
ORDER BY CourseID DESC;

/*
Queries: Table 3
*/
SELECT studentID,userID
FROM Student
ORDER BY userid DESC;


 /*
 Queries: Table 4
 */

 SELECT professorID, userID
 FROM PROFESSOR
 ORDER BY userID DESC;

/*
Queries: Table 5
*/

SELECT taID, userID
FROM ta
ORDER BY userID DESC;


/*
```

$\sigma_{userType=1}(first\_user)$

$\sigma_{userType=2}(first\_user)$

$\sigma_{userType=3}(first\_user)$

Table 1:

USERID	USERTYPE	FIRSTNAME	LASTNAME
3	1	Jasdeep	Sekhon
369369	1	Jessica	sekhon
112233	1	Joe	Rogan
999999	1	Jessica	Biel
123456	1	Tristan	Davidson
888888	1	Johnson	Smith
333333	1	Niki	Daphney
444444	1	Manjot	Singh
111111	1	Becky	Johal
528528	1	Nikola	Tesla

Download CSV

USERID	USERTYPE	FIRSTNAME	LASTNAME
999333	2	Abdol	Abhari
343434	2	Isaac	Woungang
898989	2	Jelena	Misic
898981	2	Lester	Hirakyi

USERID	USERTYPE	FIRSTNAME	LASTNAME
424242	3	Muhammed	Amir
121212	3	Tajali	Ahmed
848484	3	John	Fish
893838	3	Malahi	Kumon

USERID	USERTYPE	FIRSTNAME	LASTNAME
848484	3	John	Fish
121212	3	Tajali	Ahmed
424242	3	Muhammed	Amir
893838	3	Malahi	Kumon
898981	2	Lester	Hirakyi
898989	2	Jelena	Misic
343434	2	Isaac	Woungang
999333	2	Abdol	Abhari
528528	1	Nikola	Tesla
111111	1	Becky	Johal
444444	1	Manjot	Singh
333333	1	Niki	Daphney
888888	1	Johnson	Smith
123456	1	Tristan	Davidson
999999	1	Jessica	Biel
112233	1	Joe	Rogan
3	1	Jasdeep	Sekhon
369369	1	Jessica	sekhon

Table 2:

COURSEID	COURSENAME
633	Computer Security
510	Database Systems
420	Discrete Structures
393	Unix, C and C++

Download CSV

Table 3:

STUDENTID	USERID
999999	999999
888888	888888
528528	528528
444444	444444
369369	369369
333333	333333
123456	123456
112233	112233
111111	111111
3	3

Table 4:

PROFESSORID	USERID
999333	999333
898989	898989
898981	898981
343434	343434

Table 5:

TAID	USERID
893838	893838
848484	848484
424242	424242
121212	121212

Download CSV

Simple Queries

```sql
/*
Queries: Table 6
*/
SELECT courseID
FROM permission
where taid = 121212;
```

$\pi_{courseID}(\sigma_{taID=121212}\ permission)$

```sql
/*
Queries: Table 7
*/
SELECT classList, courseID, studentID
FROM   section, student
WHERE  sectionID = 1
       AND  professorID = 999333
ORDER BY StudentID ASC;
```

$\pi_{classList,\ courseID,\ studentID}(\sigma_{sectionID = 1 \wedge professorID = 999333}\ section) \bowtie student$

```sql
/*
Query for Lecture Materials Table
*/
SELECT *
FROM Lecture_Material, Course
WHERE courseName LIKE 'D%';
```

$\pi_{courseName = 'D\%'}\ Lecture\_Material \bowtie Course$

```sql
/*
Query for Course List Table
*/
SELECT DISTINCT *
FROM Course_List, Course;
```

$Course\_List \bowtie Course$

```sql
/*
Query for Tests Table
*/
SELECT Tests.courseID, Course.courseName, Tests.studentID,
Tests.testID, Tests.testWeight,
Tests.testEvaluationStatus, Grades.testResult
FROM Tests
INNER JOIN Course
ON Tests.courseID = Course.courseID
INNER JOIN Grades
ON Tests.courseID = Grades.courseID;
```

$\pi_{courseID,\ courseName,\ studentID,\ testID,\ testWeight}(Tests \bowtie Course \bowtie Grades)$
$(\sigma_{courseID}(Tests) = courseID(Course)\ \wedge\ (\sigma_{courseID}(Tests) = courseID(Grades)$

```sql
/*
Query for Assignments Table
*/
SELECT *
FROM Assignments
LEFT JOIN Course
ON Assignments.courseID = Course.courseID;
```

$Assignments \bowtie_{Assignments.courseID = Course.courseID} (Course)$

Simple Queries

```sql
/*
Query for Grades Table
*/
SELECT Grades.testCompletion, Grades.testResult, Tests.testWeight
FROM Grades
LEFT JOIN Tests
ON Grades.studentID = Tests.studentID ;

/*
Query for Discussion Board Table
*/
SELECT DISTINCT sectionID, discussionContent, courseID, assignmentID
FROM Discussion_Board;

/*
Query for Course Offerings Table
*/
SELECT *
FROM Course_Offerings
ORDER BY levelOfCourse;

/*
Query for relational Enrolled Table
*/
SELECT *
FROM enrolled
LEFT JOIN Course_List
ON enrolled.courseID = Course_List.courseID;
```

$\pi_{testcompletion,\ testResult,\ testWeight}$ (Grades ⋈ Tests) ($\sigma_{studentID}$ (Grades) = $_{studentID}$ (Tests)

$\pi_{sectionID,\ discussionContent,\ courseID,\ assignmentID}$ Discussion_Board

enrolled $\bowtie_{enrolled.courseID =}$ $_{Course\_List.courseID}$ (Course_List)

```sql
/*
Query for relational Checks Table
*/
SELECT enrolled.courseID, course.courseName, Course_List.professorID,  enrolled.studentID
FROM enrolled
INNER JOIN Course_List
ON enrolled.courseID = Course_List.courseID
INNER JOIN Course
ON enrolled.courseID = Course.courseID
WHERE enrolled.courseID LIKE '633';

/*
Query for relational Creates Table
*/
SELECT creates.professorID, first_user.firstName, first_user.lastName,
creates.courseID, course.courseName
FROM creates
INNER JOIN first_user
ON creates.professorID = first_user.userID
INNER JOIN course
ON creates.courseID = course.courseID
WHERE creates.professorID > '300000'
ORDER BY creates.professorID;
```

$\pi_{courseID,\ courseName,\ professorID,\ studentID}$ (Enrolled⋈Course_List) ($\sigma_{courseID}$(Enrolled) = $_{course\_List}$ (CourseID) ^ (Enrolled⋈Course) ($\sigma_{courseID(}$enrolled) = $_{courseID}$(Course)

$\pi_{creates.professorID,\ first\_user.firstName,\ first\_user.lastName,}$ $_{creates.courseID,\ course.courseName}$($\sigma_{creates.professorID > '300000'}$ (creates) $\bowtie_{creates.professorID = first\_user.userID}$(first_user) $\bowtie_{creates.courseID = course.courseID}$(course))

Advanced Queries

```sql
/*
View for Students Enrolled in CPS 633
*/

CREATE VIEW Students_enrolled_CPS_633
AS
SELECT classList, courseID
from SECTION
where courseID = 633;

SELECT *
FROM Students_enrolled_CPS_633

/*
View for Students Enrolled in CPS 510
*/

CREATE VIEW Students_enrolled_CPS_510
AS
SELECT classList, courseID
from SECTION
where courseID = 510;

SELECT *
FROM Students_enrolled_CPS_510

/*
View for Students Enrolled in CPS 420
*/

CREATE VIEW Students_enrolled_CPS_420
AS
SELECT classList, courseID
from SECTION
where courseID = 420;

SELECT *
FROM Students_enrolled_CPS_420

/*
View for students Assignment Grades
*/

CREATE VIEW students_grades
AS
SELECT DISTINCT student.studentID, Assignments.assignmentID, Grades.assignmentResult
from Student, Course, Assignments, Grades
WHERE
Student.studentID = Assignments.studentID AND
grades.studentID = Assignments.studentID;

SELECT *
FROM students_grades
```

```sql
/*
Checking which students are enrolled in CPS 510 and have completed their tests
(WHERE EXISTS)
*/

SELECT s.studentID
FROM Student s
WHERE EXISTS
(SELECT t.studentID
FROM Tests t
WHERE t.testEvaluationStatus = 'Complete' AND
t.courseID = 510
AND t.studentID = s.studentID);

/*
Checking which students did complete completed both their tests and assignments
(UNION)
*/

SELECT s.studentID
FROM Student s
WHERE EXISTS
(SELECT t.studentID
FROM Tests t
WHERE t.testEvaluationStatus = 'Complete'
AND t.studentID = s.studentID)

UNION

(SELECT s.studentID
FROM Student s
WHERE EXISTS
(SELECT a.studentID
FROM Assignments a
WHERE a.assignmentEvaluationStatus = 'Complete'
AND a.studentID = s.studentID));
```

$$\pi_{s.studentID}(\text{Student s} \bowtie_{s.studentID = t.studentID} \wedge \sigma_{t.testEvaluationStatus = \text{`Complete}} (\text{Tests t})) \wedge \sigma_{t.studentID} = s.studentID (\text{Tests t})$$

$$\pi_{s.studentID}(\text{Student s} \bowtie_{s.studentID = t.studentID} \wedge \sigma_{t.testEvaluationStatus = \text{`Complete}} (\text{Tests t})) \cup \pi_{student.studentID}(\text{Student s} \bowtie_{s.studentID = a.studentID} \wedge \sigma_{a.assignmentEvaluationStatus = \text{`Complete'}}(\text{Assignments a}))$$

Advanced Queries

```sql
/*
List of courses not offered in the Winter Term
(MINUS)
*/

SELECT c.subject, c.levelOfCourse, c.program, c.courseID
 FROM   Course_Offerings c
MINUS
SELECT c.subject, c.levelOfCourse, c.program, c.courseID
 FROM   Course_Offerings c
 WHERE  c.availablity = 'Winter Term';

 /*
 Find Average of all students on their test results
 (AVERAGE)
 */

 SELECT 'Average test result is ', AVG (testResult)
 FROM Grades;
  /*
 Find Average of all students in all sections on their assignment results
 (AVERAGE)
 */

SELECT 'Average assignment result is ', AVG (assignmentResult)
FROM Grades;

/*
List for each course, the number of students registered for the course
*/

SELECT COUNT( DISTINCT studentID) AS Number_Enrolled
FROM   enrolled

/*
GROUP BY studentID for average assignment results
*/
SELECT studentID, AVG(assignmentResult) AS Average_Assignment_Results
FROM Grades
GROUP BY studentID;
```

$\pi_{\text{c.subject, c.levelOfCourse, c.program, c.courseID}}(\text{Course\_Offerings c}) - (\pi_{\text{c.subject, c.levelOfCourse, c.program, c.courseID}}(\sigma_{\text{c.availability = 'Winter Term'}} \text{Course\_Offerings c}))$

CREATE TABLES

```
#!/bin/sh
#export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib
sqlplus64 "USERNAME/PASSWD@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)
(Host=oracle.scs.ryerson.ca)(Port=1521))(CONNECT_DATA=(SID=orcl)))" <<EOF

CREATE TABLE first_user(
    userID NUMBER PRIMARY KEY,
    userType NUMBER,
    firstName VARCHAR2 (50 CHAR),
    lastName VARCHAR2 (50 CHAR)
);

CREATE TABLE Course(
    courseID NUMBER PRIMARY KEY NOT NULL,
    taPermissionCourse NUMBER NOT NULL,
    professorPermissionCourse NUMBER NOT NULL,
    courseName VARCHAR2 (50 CHAR)
);

CREATE TABLE Student(
    studentID NUMBER PRIMARY KEY,
    userID NUMBER,
    FOREIGN KEY (userID) REFERENCES first_user(userID)
);

CREATE TABLE Professor(
    professorID NUMBER PRIMARY KEY,
    userID NUMBER,
    FOREIGN KEY (userID) REFERENCES first_user(userID)
);

CREATE TABLE TA(
    taID NUMBER PRIMARY KEY,
    userID NUMBER,
    FOREIGN KEY (userID) REFERENCES first_user(userID)
);



CREATE TABLE Permission(
    profPermissionID NUMBER NOT NULL,
    taPermissionID NUMBER NOT NULL,
    professorID NUMBER,
    FOREIGN KEY (professorID) REFERENCES Professor(professorID),
    taID NUMBER,
    FOREIGN KEY (taID) REFERENCES TA(taID),
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID)
);

CREATE TABLE Prerequisite(
    prerequisiteID NUMBER PRIMARY KEY NOT NULL,
    payment NUMBER NOT NULL,
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID)
);

CREATE TABLE Section(
    sectionID NUMBER PRIMARY KEY NOT NULL,
    classList VARCHAR2 (1000 CHAR),
    professorID NUMBER,
    FOREIGN KEY (professorID) REFERENCES Professor(professorID),
    taID NUMBER,
    FOREIGN KEY (taID) REFERENCES TA(taID),
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID)
);

CREATE TABLE Lecture_Material(
    lectureID NUMBER PRIMARY KEY NOT NULL,
    lectureContents VARCHAR2 (1000 CHAR)
);

CREATE TABLE Course_List(
    professorID NUMBER,
    FOREIGN KEY (professorID) REFERENCES Professor(professorID),
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID)
);

CREATE TABLE Tests(
    testID NUMBER PRIMARY KEY NOT NULL,
    testWeight NUMBER NOT NULL,
    testEvaluationStatus VARCHAR2 (20 CHAR),
    sectionID NUMBER,
    FOREIGN KEY (sectionID) REFERENCES Section(sectionID),
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID),
    studentID NUMBER,
    FOREIGN KEY (studentID) REFERENCES Student(studentID)
);

CREATE TABLE Assignments(
    assignmentID NUMBER PRIMARY KEY NOT NULL,
    assignmentWeight NUMBER NOT NULL,
    assignmentEvaluationStatus VARCHAR2 (20 CHAR),
    sectionID NUMBER,
    FOREIGN KEY (sectionID) REFERENCES Section(sectionID),
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID),
    studentID NUMBER,
    FOREIGN KEY (studentID) REFERENCES Student(studentID)
);

CREATE TABLE Grades(
    assignmentCompletion VARCHAR2 (10 CHAR),
    assignmentResult NUMBER,
    testCompletion VARCHAR2 (10 CHAR),
    testResult NUMBER,
    sectionID NUMBER,
    FOREIGN KEY (sectionID) REFERENCES Section(sectionID),
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID),
    studentID NUMBER,
    FOREIGN KEY (studentID) REFERENCES Student(studentID)
);

CREATE TABLE Discussion_Board(
    discussionContent VARCHAR2(1000 CHAR),
    sectionID NUMBER,
    FOREIGN KEY (sectionID) REFERENCES Section(sectionID),
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID) ,
    assignmentID NUMBER,
    FOREIGN KEY (assignmentID) REFERENCES Assignments(assignmentID) ON DELETE
CASCADE
);

CREATE TABLE Course_Offerings(
    availablity VARCHAR2 (50 CHAR),
    levelOfCourse VARCHAR2 (50 CHAR),
    subject VARCHAR2 (50 CHAR),
    program VARCHAR2 (50 CHAR),
    professorID NUMBER,
    FOREIGN KEY (professorID) REFERENCES Professor(professorID),
    taID NUMBER,
    FOREIGN KEY (taID) REFERENCES TA(taID),
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID)
);
CREATE TABLE enrolled (
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID),
    studentID NUMBER,
    FOREIGN KEY (studentID) REFERENCES Student(studentID)
);

CREATE TABLE checks (
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID),
    prerequisiteID  NUMBER NOT NULL,
    FOREIGN KEY(prerequisiteID) REFERENCES Prerequisite(prerequisiteID)
);

CREATE TABLE creates (
    professorID NUMBER NOT NULL,
    FOREIGN KEY (professorID) REFERENCES Professor(professorID),
    courseID NUMBER NOT NULL,
    FOREIGN KEY (courseID) REFERENCES Course(courseID)
);


exit;
EOF
```

DROP TABLES

```
#!/bin/sh
#export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib
sqlplus64 "USERNAME/PASSWD@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.scs.ryerson.ca)(Port=1521))
(CONNECT_DATA=(SID=orcl)))" <<EOF
DROP TABLE first_user CASCADE CONSTRAINTS;
DROP TABLE Course CASCADE CONSTRAINTS;
DROP TABLE Student CASCADE CONSTRAINTS;
DROP TABLE Professor CASCADE CONSTRAINTS;
DROP TABLE TA CASCADE CONSTRAINTS;
DROP TABLE Permission CASCADE CONSTRAINTS;
DROP TABLE Prerequisites CASCADE CONSTRAINTS;
DROP TABLE Section CASCADE CONSTRAINTS;
DROP TABLE Lecture_Material CASCADE CONSTRAINTS;
DROP TABLE Course_List CASCADE CONSTRAINTS;
DROP TABLE Tests CASCADE CONSTRAINTS;
DROP TABLE Assignments CASCADE CONSTRAINTS;
DROP TABLE Grades CASCADE CONSTRAINTS;
DROP TABLE Discussion_Board CASCADE CONSTRAINTS;
DROP TABLE Course_Offerings CASCADE CONSTRAINTS;
DROP TABLE enrolled CASCADE CONSTRAINTS;
DROP TABLE checks CASCADE CONSTRAINTS;
DROP TABLE creates CASCADE CONSTRAINTS;


exit;
EOF
```

POPULATE TABLES

```
#!/bin/sh
#export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib
sqlplus64 "USERNAME/PASSWD@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=oracle.scs.ryerson.ca)
(Port=1521))(CONNECT_DATA=(SID=orcl)))" <<EOF

insert into first_user VALUES (3, 1, 'Jasdeep','Sekhon');
insert into first_user values (369369, 01, 'Jessica', 'sekhon');
insert into first_user values (112233, 01, 'Joe', 'Rogan');
insert into first_user values (999999, 01, 'Jessica', 'Biel');
insert into first_user values (123456, 01, 'Tristan', 'Davidson');
insert into first_user values (888888, 01, 'Johnson', 'Smith');
insert into first_user values (333333, 01, 'Niki', 'Daphney');
insert into first_user values (444444, 01, 'Manjot', 'Singh');
insert into first_user values (111111, 01, 'Becky', 'Johal');
insert into first_user values (528528, 01, 'Nikola', 'Tesla');
insert into first_user values (999333, 02, 'Abdol', 'Abhari');
insert into first_user values (343434, 02, 'Isaac', 'Woungang');
insert into first_user values (898989, 02, 'Jelena', 'Misic');
insert into first_user values (898981, 02, 'Lester', 'Hirakvi');
insert into first_user values (424242, 03, 'Muhammed', 'Amir');
insert into first_user values (121212, 03, 'Tajali', 'Ahmed');
insert into first_user values (848484, 03, 'John', 'Fish');
insert into first_user values (893838, 03, 'Malahi', 'Kumon');

insert into Course values (510, 3, 2, 'Database Systems');
insert into Course values (633, 3, 2, 'Computer Security');
insert into Course values (393, 3, 2, 'Unix, C and C++');
insert into Course values (420, 3, 2, 'Discrete Structures');

insert into Student VALUES (3, 3);
insert into Student values (369369,369369);
insert into Student values (112233,112233);
insert into Student values (999999,999999);
insert into Student values (123456,123456);
insert into Student values (888888,888888);
insert into Student values (333333,333333);
insert into Student values (444444,444444);
insert into Student values (111111,111111);
insert into Student values (528528,528528);

insert into Professor values (999333, 999333);
insert into Professor values (343434, 343434);
insert into Professor values (898989, 898989);
insert into Professor values (898981, 898981);

insert into TA values (424242, 424242);
insert into TA values (121212, 121212);
insert into TA values (848484, 848484);
insert into TA values (893838,893838);
```

POPULATE TABLES

```
insert into Permission values (12345,12354,999333,424242,510);
insert into Permission values (54321,12435,343434,121212,633);
insert into Permission values (99999,88888,898989,848484,420);
insert into Permission values (99989,88828,898981,893838,393);

insert into Prerequisite values (123, 1000, 510);
insert into Prerequisite values (456, 1000, 633);
insert into Prerequisite values (789, 1000, 420);
insert into Prerequisite values (111, 1000, 393);

insert into Section values(1,'Jessica Sekhon, Joe Rogan',999333,424242,510);
insert into Section values (2, 'Jessica Biel, Tristan Davidson, Johnson Smith', 343434,121212, 633);
insert into Section values (3, 'Niki Daphney, Manjot Singh',898981,893838,393);
insert into Section values (4, 'Becky Johal, Nikola Tesla', 898989, 848484, 420);

insert into Lecture_Material values (20210910, 'Course Outline');


insert into Course_List values (999333, 510);
insert into Course_List values (343434, 633);
insert into Course_List values (898981,393);
insert into Course_List values (898989,420);

insert into Tests values(20210912, 25, 'Complete',1,510,369369);
insert into Tests values (20212939, 25, 'Complete', 1, 510,112233);
insert into Tests values (20211011, 40, 'Complete',2, 633, 999999);
insert into Tests values (19239123, 35, 'Complete',2, 633, 888888);
insert into Tests values (12389123, 20, 'Not Complete', 2, 633, 123456);
insert into Tests values (123923929, 10, 'Complete', 3, 393,333333 );
insert into Tests values (12189238, 10, 'Complete',3, 393, 444444);
insert into Tests values (19238192, 50, 'Complete',4, 420, 111111);
insert into Tests values (12192839, 50, 'Complete',4, 420,528528);

insert into Assignments values(20210912, 25, 'Complete',1,510,369369);
insert into Assignments values (20212939, 25, 'Complete', 1, 510,112233);
insert into Assignments values (20211011, 40, 'Complete',2, 633, 999999);
insert into Assignments values (19239123, 35, 'Complete',2, 633, 888888);
insert into Assignments values (12389123, 20, 'Not Complete', 2, 633, 123456);
insert into Assignments values (123923929, 10, 'Complete', 3, 393,333333 );
insert into Assignments values (12189238, 10, 'Complete',3, 393, 444444);
insert into Assignments values (19238192, 50, 'Complete',4, 420, 111111);
insert into Assignments values (12192839, 50, 'Complete',4, 420,528528);

insert into Grades values ('Yes',90,'Yes',88,1, 510,369369);
insert into Grades values ('Yes',69,'Yes',96,1, 510,112233);
insert into Grades values ('Yes',92,'Yes',75,2, 633,999999);
insert into Grades values ('Yes',90,'Yes',90,2, 633,888888);
insert into Grades values ('No',92, 'No', 88,2, 633,123456);
insert into Grades values ('Yes',99,'Yes',89,3, 393,333333);
insert into Grades values ('Yes',66,'Yes',99,3, 393,444444);
insert into Grades values ('Yes',85,'Yes',87,4, 420,111111);
insert into Grades values ('Yes',89,'Yes',98,4, 420,528528);

insert into Discussion_Board values ('Database ER Diagram Discussion', 1, 510,20210912);
insert into Discussion_Board values ('SQL Tables and Queries Discussion', 1, 510, 20212939);
insert into Discussion_Board values ('SetUID Lab Discussion',2,633,20211011);
insert into Discussion_Board values ('MD5 Collision Attack',2, 633, 19239123);
insert into Discussion_Board values ('Strong Induction Questions',4,420,19238192);

insert into Course_Offerings values('Winter Term', '3rd Year','Database Systems','Computer Science',999333,424242, 510);
insert into Course_Offerings values('Winter Term', '3rd Year','Computer Security','Computer Science',343434,121212, 633);
insert into Course_Offerings values ('Fall Term', '2nd year', 'Unix, C and C++', 'Computer Science',898981, 893838, 393);
insert into Course_Offerings values ('Fall Term', '2nd year', 'Discrete Structures', 'Computer Science',898989,848484 , 420);


insert into enrolled values(510,369369);
insert into enrolled values(510,112233);
insert into enrolled values (633,999999);
insert into enrolled values(633,123456);
insert into enrolled values(633,888888);
insert into enrolled values(393,333333);
insert into enrolled values(393,444444);
insert into enrolled values(420,111111);
insert into enrolled values(420,528528);

insert into checks values(510,123);
insert into checks values(633,456);
insert into checks values(393,789);
insert into checks values(420,111);

insert into creates values(999333,510);
insert into creates values(343434,633);
insert into creates values(898981,393);
insert into creates values(898989,420);
```

QUERY TABLES

```
#!/bin/sh
#export LD_LIBRARY_PATH=/usr/lib/oracle/12.1/client64/lib
sqlplus64 "USERNAME/PASSWD@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)
(Host=oracle.scs.ryerson.ca)(Port=1521))(CONNECT_DATA=(SID=orcl)))" <<EOF

/*
Queries: Table 1
*/
SELECT *
FROM first_user
WHERE userType =1;

SELECT *
FROM first_user
WHERE userType =2;

SELECT *
FROM first_user
WHERE userType =3;

SELECT *
FROM first_user
ORDER BY userType DESC;

/*
Queries: Table 2
*/

SELECT CourseID, Coursename
FROM   Course
ORDER BY CourseID DESC;

/*
Queries: Table 3
*/
SELECT studentID,userID
FROM Student
ORDER BY userid DESC;

/*
Queries: Table 4
*/

SELECT professorID, userID
FROM PROFESSOR
ORDER BY userID DESC;

/*
Queries: Table 5
*/

SELECT taID, userID
FROM ta
ORDER BY userID DESC;


/*
Queries: Table 6
*/
SELECT courseID
FROM permission
where taid = 121212;

/*
Queries: Table 7
*/
SELECT classList, courseID, studentID
FROM   section, student
WHERE  sectionID = 1
       AND  professorID = 999333
ORDER BY StudentID ASC;


/*
Query for Lecture Materials Table
*/
SELECT *
FROM Lecture_Material, Course
WHERE courseName LIKE 'D%';

/*
Query for Course List Table
*/
SELECT DISTINCT *
FROM Course_List, Course;

/*
Query for Tests Table
*/
SELECT Tests.courseID, Course.courseName, Tests.studentID, Tests.testID,
Tests.testWeight, Tests.testEvaluationStatus, Grades.testResult
FROM Tests
INNER JOIN Course
ON Tests.courseID = Course.courseID
INNER JOIN Grades
ON Tests.courseID = Grades.courseID;

/*
Query for Assignments Table
*/
SELECT *
FROM Assignments
LEFT JOIN Course
ON Assignments.courseID = Course.courseID;

/*

/*
Query for Grades Table
*/
SELECT Grades.testCompletion, Grades.testResult, Tests.testWeight
FROM Grades
LEFT JOIN Tests
ON Grades.studentID = Tests.studentID ;

/*
Query for Discussion Board Table
*/
SELECT DISTINCT sectionID, discussionContent, courseID, assignmentID
FROM Discussion_Board;

/*
Query for Course Offerings Table
*/
SELECT *
FROM Course_Offerings
ORDER BY levelOfCourse;

/*
Query for relational Enrolled Table
*/
SELECT *
FROM enrolled
LEFT JOIN Course_List
ON enrolled.courseID = Course_List.courseID;

/*
Query for relational Checks Table
*/
SELECT enrolled.courseID, course.courseName, Course_List.professorID,
enrolled.studentID
FROM enrolled
INNER JOIN Course_List
ON enrolled.courseID = Course_List.courseID
INNER JOIN Course
ON enrolled.courseID = Course.courseID
WHERE enrolled.courseID LIKE '633';

/*
Query for relational Creates Table
*/
SELECT creates.professorID, first_user.firstName, first_user.lastName,
creates.courseID, course.courseName
FROM creates
INNER JOIN first_user
ON creates.professorID = first_user.userID
INNER JOIN course
ON creates.courseID = course.courseID
WHERE creates.professorID > '300000'
ORDER BY creates.professorID;

/*
Checking which students are enrolled in CPS 510 and have completed their tests
(WHERE EXISTS)
*/

SELECT s.studentID
FROM Student s
WHERE EXISTS
(SELECT t.studentID
FROM Tests t
WHERE t.testEvaluationStatus = 'Complete' AND
t.courseID = 510
AND t.studentID = s.studentID);
```

QUERY TABLES

```
/*
Checking which students did complete completed both their tests and assignments
(UNION)
*/
*/

SELECT s.studentID
FROM Student s
WHERE EXISTS
(SELECT t.studentID
FROM Tests t
WHERE t.testEvaluationStatus = 'Complete'
AND t.studentID = s.studentID)

UNION

(SELECT s.studentID
FROM Student s
WHERE EXISTS
(SELECT a.studentID
FROM Assignments a
WHERE a.assignmentEvaluationStatus = 'Complete'
AND a.studentID = s.studentID));

/*
List of courses not offered in the Winter Term
(MINUS)
*/

SELECT c.subject, c.levelOfCourse, c.program, c.courseID
 FROM   Course_Offerings c
MINUS
SELECT c.subject, c.levelOfCourse, c.program, c.courseID
 FROM   Course_Offerings c
 WHERE  c.availablity = 'Winter Term';

 /*
 Find Average of all students on their test results
 (AVERAGE)
 */

 SELECT 'Average test result is ', AVG (testResult)
 FROM Grades;
  /*
 Find Average of all students in all sections on their assignment results
 (AVERAGE)
 */

SELECT 'Average test result is ', AVG (assignmentResult)
FROM Grades;

/*
List for each course, the number of students registered for the course
*/

SELECT COUNT( DISTINCT studentID) AS Number_Enrolled
FROM   enrolled
```

Table 1: First_user
userID → userType, firstName, lastName

Table 2: Course
courseID → courseName, taPermissionCourse, professorPermissionCourse

Table 3: Student
userID → studentID

Table 4: Professor
userID → professorID

Table 5: TA
userID → taID

Table 6: Permission
courseID → taPermissionID, profPermissionID, taID, professorID

Table 7: Prerequisite
prerequisiteID → courseID, payment

Table 8: Section
sectionID→ classList, taID, courseID

Table 9: Lecture_Material
lectureID → lectureContents

Table 10: Course_List
courseID → professorID

Table 11: Tests
testID → testWeight, testEvalualationStatus, sectionID, studentID, courseID, studentID

Table 12: Assignments
assignmentID → assignmentWeight, assignmentEvaluationStatus, sectionID, studentID, courseID

Table 13: Grades

- No FD since no primary key

Table 14: Discussion_Board

assignmentID → discussionContent, sectionID, courseID

Table 15: Course_Offerings

courseID → professorID, taID, availability, levelOfCourse, subject, program

Table 16: Enrolled

studentID → courseID

Table 17: Checks

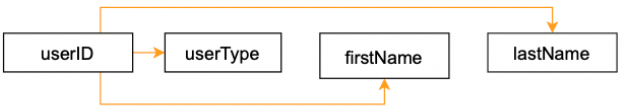
courseID → prerequisiteID

Table 18: Creates

courseID → professorID

Normalization of Database: 3NF Form

Table 1: First_user



FD: userID --> userType, firstName, lastName

Table 2: Course

FD: courseID --> courseName, taPermissionCourse, professorPermissionCourse

Table 3: Student

FD: userID --> studentID

Table 4: Professor

FD: userID --> professorID

Table 5: TA

FD: userID --> taID

Table 6: Permission

FD: courseID --> taPermissionID, profPermissionID, taID, professorID

Table 7: Prerequisites

FD :prerequisiteID --> courseID, payment

Table 8: Section

FD: sectionID --> classList, taID, courseID

3NF Form:

As database requirements increase, Database complexity increases. Therefore we must follow normalization.

What is 3NF?
When an attribute is dependent on Non-prime attribute (transitive dependency)

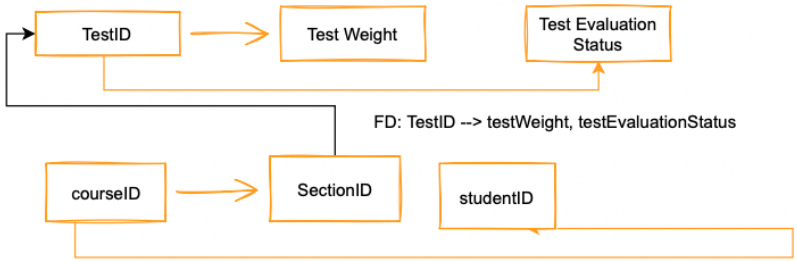
Table 9: lecture_Material

FD: lectureID --> lectureContents

Table 10:Course_List

FD: courseID --> professorID

Table 11: Tests



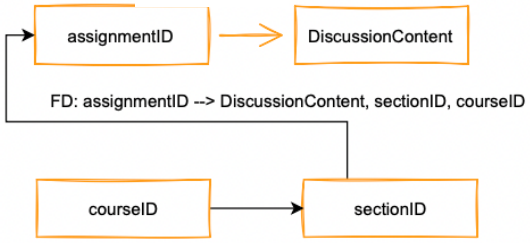
FD: TestID --> SectionID, CourseID, studentID

For tests,

courseID also determines sectionID and studentID.

Therefore, we must put it in a separate table.

Table 12: Assignments

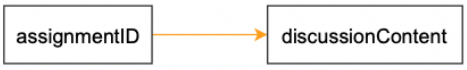


FD: courseID --> sectionID

For assignments,

courseID also determines sectionID. We must put these relations separately.

Table 14:Discussion_Board



FD: assignmentID --> DiscussionContent

Normalization of Database: 3NF Form

Table 15: Course_Offerings

FD: courseID --> professorID, taID, availability, levelOfCourse, subject, program

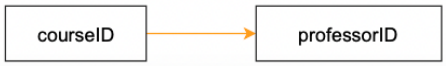
Table 16: Enrolled

FD: studentID --> courseID

Table 17: Checks

FD: courseID --> prerequisiteID

Table 18: Creates



FD: courseID --> professorID

We will be using Bernstein's algorithm to normalize the following tables in 3NF.

3NF vs BCNF

3NF: A partial key (prime attribute) can *also* depend on an attribute that is ***not*** a superkey
BCNF: Every partial key (prime attribute) can *only* depend on a superkey,

Table 11 in 3NF Form (using Bernstein's algorithm)

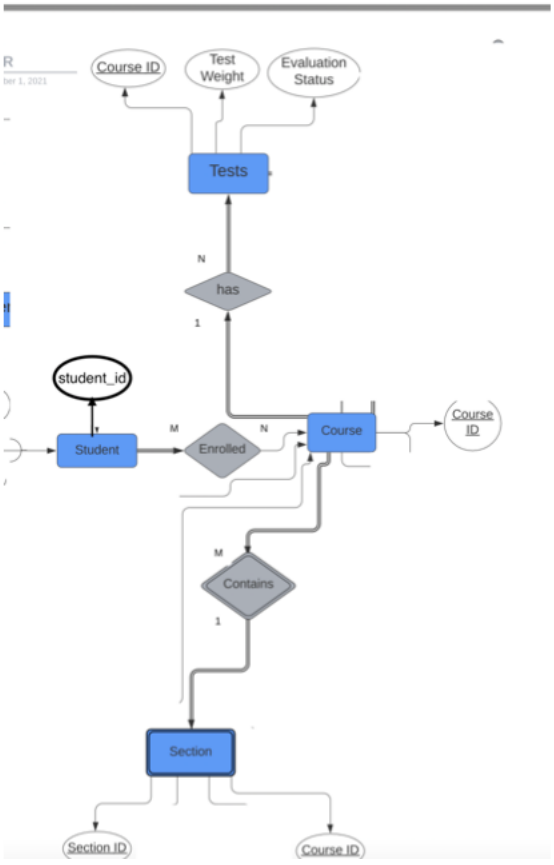
Why is this not in 3NF?
testName → testEvaluationStatus
testEvaluationStatus is transitively dependant on testName

For 3NF we need to make sure any non prime attributes are dependent on a prime attribute. In this case we have a non prime attribute dependent on another non prime attribute.

Step 1. Determine Functional Dependencies

testID → testWeight
courseID → sectionID
sectionID → courseID
studentID, sectionID → testEvaluationStatus
sectionID → testName
testName → testID
testName → testEvaluationStatus

testID = t
testWeight = w
courseID = c
sectionID = s
studentID = i
testEvaluationStatus = e
testName = n



Step 2. Break RHS and find redundancies

FD = {
t → w,
c → s,
s → c,
si → e,
s → n,
n → t
n → e
}

t → w: t += {t} we do not get w, so not redundant
c → s: c+= {c} we do not get s, so not redundant
s → c: s += {s,n,t,w,e} we do not get c, so not redundant
si → e: si += {s,i,c,n,t,w} we do not get e, so not redundant
s → n: s += {s,c} we do not get n, so not redundant
n → t: n+= {n} we do not get t so not redundant
n → e: e+= {n,t,w} we do not n so not redundant
So after removing redundancies, FD = {t → w, c → s, s → c, si → e, s → n, n → t, n → e}

Step 2b. Find and remove partial dependencies (or minimizing LHS)
si → e: s+= {c,s,n,t,w} i+= {i} fully dependant since we do not get e

If we find a partial dependency, we can remove one of the attributes from the LHS. In this case we checked all the LHS with composite attributes and we couldn't minimize LHS so they are 2NF. For the rest of FDs because they have one attribute on LHS they are already minimized. So FDs are:

t → w
c → s
s → c
si → e
s → n
n → t
n → e

Step 3: Find keys

A formula to make this process simple is:
If an attribute is not in LHS and is not in LHR: it is part of the key
If an attribute is only in LHS but not in RHS: it is part of the key
If an attribute is only in RHS and not in LHS: it is NOT part of the key

Firstly, find which one are not keys

→ w, n are on RHS but not in LHS
→ therefore candidate keys are: testID, courseID, student_ID

Step 4: Make Tables

- **Here we can make this 3nf by creating a separate exam table for the functional dependency** testEvaluationStatus → testName

We have FD = {t → w, c → s, s → c, si → e, s → n, e → n, n → t}

We make a relation for each functional dependency:
t → w : R1(t,w)
c → s : R2 (c, s)
s → c : R3 (s, c)
si → n : R4 (s,n,i)
s → n: R5 (s,n)
n → t : R6 (n,t)
n → e: R7 (e,n)

Now to make it lossless join and dependency preserving if there is no key in the Relations R1 to R4 we need to add at least one key. So we can add any of the HSRT or HSRC to decomposition so we have

R1(t,w) with FD: t → w
R2 (c, s) with FD: c → s
R3 (s, c) with FD: s → c
R4 (s,n,i) with FD: si → n
R5 (s,n) with FD:s → n
R6 (n,t) with FD : n → t
R7 (n,e) with FD: n → e
R8 (t,s,c) with no FD

BCNF DECOMPOSITION

Check if table is in BCNF
t → w,
c → s,
s → c,
si → e,
s → n,
n → t

A relation is in BCNF if and only if every non trivial, left irreducible FD has a candidate key as its determinant

So we need to test if the test if the hand sides are the keys are not

t+ = {t} testID is a key
c+ = {c} courseID is a key
s+ = {s,n,t,w) sectionID is a key
si+ = {s,i,n,t,w,c} sectionID and studentID are both keys
s+ = {s,c} sectionID is a key
n+= {n,t} testName is not a key. So it is not BCNF with respect to n → t

Since R is not BCNF with respect to n → t and
n+ = {n,t) decompose R in R11 and R12

R11 = {w,c,s,e,i}
R12 = {n,t} which is BCNF

Now the question is if R11 is in BCNF with FD11
FD11 = { t → w, c → s, s → c, si ---> e, s---> n}
si += {s,i,e,c,n,t,w} it is key
s+= {s,n,c,t,w} it is key

t += {t,w} is not key so split R11 to R111 and R112

R111 = (c,s,e,i)
With FD111 {c → s, si---> e, s ---> c}

R112 = (t,w) is BCNF

With F112 = (t → w)

Final BCNF schema for R IS

R1 = (n,t)
R2 = (t,w)
R3 = (c,s,e,i)

Final Remarks

Overall, this course as well as this assignment has taught us a lot about SQL and DBMS. Nonetheless, it also taught us how to design and implement a DBMS through scheme design, ER modelling, queries, application development and normalization. It was nice to see our project grow and take fold as we progressed through the semester. We were able to incorporate academic feedback and apply it every week to make sure we were applying the correct principles as well as producing a product which resembled what we had envisioned early on.

We hope to take these skills, and the ability to analyze user requirements and apply them to future projects or even in implementation of a database management system.