Project Phase II: University Database System

SOFE3700U:Data Management Systems

Group 13

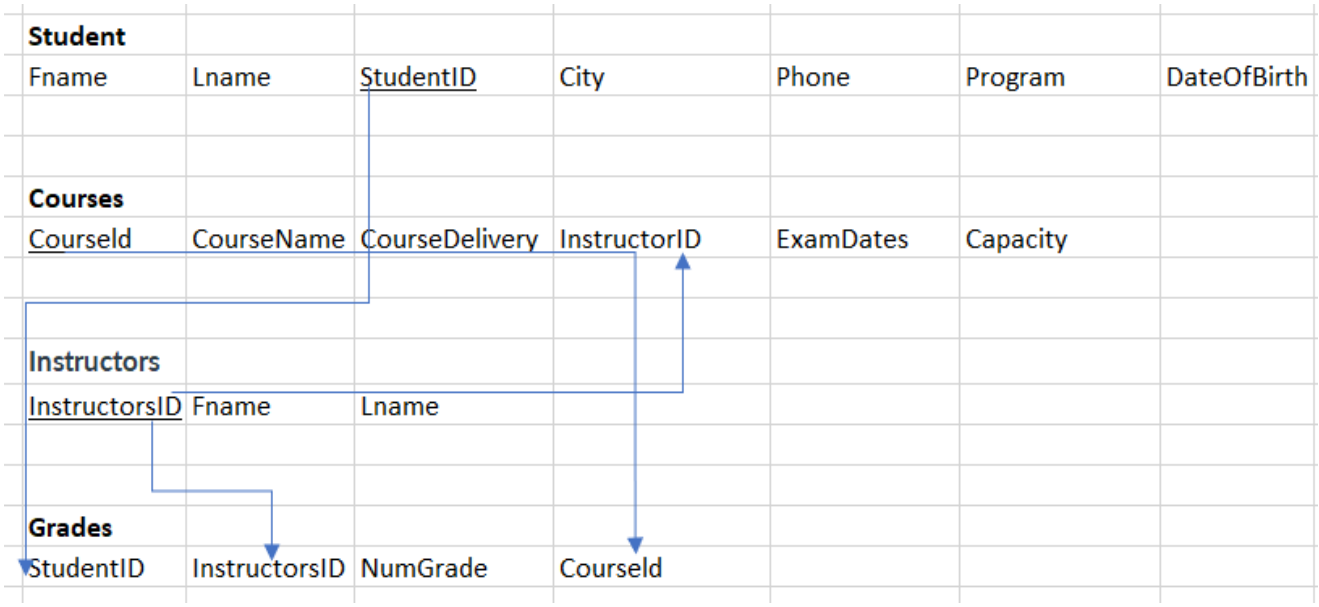
Nov. 7, 2021

Rayyan Mohammed(100752351)
Daniyal Khan (100750029)
Muhammad Zaeem Khalid (100746801)
Yousif Sarmad (100746887)
Charles Olagunju (100749818)

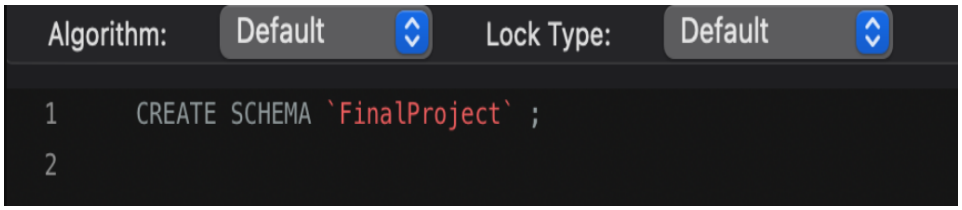
Deliverables:

•Part A: Relational Schema (5points)

- Create relational database schema for the proposed application area from Phase I:



- Actual SQL create table commands are required.



Creates a schema called FinalProject

Algorithm: Default Lock Type: Default

```sql
CREATE TABLE `FinalProject`.`Student` (
  `StudentID` INT NOT NULL,
  `Fname` VARCHAR(45) NOT NULL,
  `Lname` VARCHAR(45) NOT NULL,
  `City` VARCHAR(45) NOT NULL,
  `Phone` INT NOT NULL,
  `DateOfBirth` DATETIME NOT NULL,
  `Program` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`StudentID`));

```

SQL create table command for Student table

```sql
CREATE TABLE `FinalProject`.`Instructors` (
  `InstructorID` INT NOT NULL,
  `Fname` VARCHAR(45) NOT NULL,
  `Lname` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`InstructorID`));

```

SQL create table command for Instructor table

Algorithm: Default Lock Type: Default

```sql
CREATE TABLE `FinalProject`.`Courses` (
  `CourseID` INT NOT NULL,
  `CouseName` VARCHAR(45) NOT NULL,
  `CourseDelivery` VARCHAR(45) NOT NULL,
  `Capacity` INT NOT NULL,
  `ExamDates` DATETIME NOT NULL,
  `InstructorID` INT NOT NULL,
  PRIMARY KEY (`CourseID`));

```

SQL create table command for Courses table

```
ALTER TABLE `FinalProject`.`Courses`
ADD INDEX `InstructorID_idx` (`InstructorID` ASC) VISIBLE;
;
ALTER TABLE `FinalProject`.`Courses`
ADD CONSTRAINT `InstructorID`
  FOREIGN KEY (`InstructorID`)
  REFERENCES `FinalProject`.`Instructors` (`InstructorID`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION;

```

Adds the foreign key “InstructorID” to the Courses table

```
CREATE TABLE `FinalProject`.`Grades` (
    `StudentID` INT NOT NULL,
    `CourseID` VARCHAR(45) NOT NULL,
    `InstructorID` INT NOT NULL,
    `NumGrade` INT NOT NULL);

```

SQL create table command for Grades table

```
ALTER TABLE `FinalProject`.`Grades`
ADD CONSTRAINT `StudentID`
  FOREIGN KEY (`StudentID`)
  REFERENCES `FinalProject`.`Student` (`StudentID`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
ADD CONSTRAINT `CourseID`
  FOREIGN KEY (`CourseID`)
  REFERENCES `FinalProject`.`Courses` (`CourseID`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
ADD CONSTRAINT `InstructorID1`
  FOREIGN KEY (`InstructorID`)
  REFERENCES `FinalProject`.`Instructors` (`InstructorID`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION;

```

Adds the foreign keys “InstructorID”, “CourseID”, and “StudentID” to the Grades table

Part B: Sample Data (2points)

- Populate your database with sample data. Each relation should contain at least 6 tuples. Make sure that the populated data is suitable for the type of queries in Part C.

Student

StudentID	Fname	Lname	City	Phone	DateOfBirth	Program
10001	Rayyan	Mohammed	Toronto	416-875-2145	2001-10-23	Engineering
10002	Daniyal	Khan	Toronto	647-854-7865	2001-03-26	Business
10003	Charles	Olagunju	Pickering	209-451-8765	2001-07-24	Health Science
10004	Yousif	Sarmad	Oshawa	905-324-7851	1992-01-01	Engineering
10005	Zaeem	Khalid	Oshawa	905-789-3214	1990-05-12	Business
10006	John	Doe	Mississauga	209-147-8532	1983-12-15	Nursing

Courses

CourseID	CourseName	CourseDelivery	Capacity	ExamDates	InstructorID
ENG3200	Intro to Engineering	In Person	140	2021-12-15	20001
ENG3850	Engineering Economics	In Person	140	2021-12-12	20001
BUS2840	Accounting Principles	Hybrid	200	2021-12-13	20002
BUS1470	Intro to Business	Online	500	2021-12-09	20002
HSCI4800	Life Science Advanced	Hybrid	200	2021-12-12	20004
NUR1000	Nursing Principles	In Person	100	2021-12-15	20006
HSCI1000	Intro to Life Science	Online	500	2021-12-05	20003
NUR4000	Long Term Care	Hybrid	200	2021-12-07	20005

Instructors

InstructorID	Fname	Lname
20001	Khalid	Hafeez
20002	Usman	Aziz
20003	Farhan	Mohammed
20004	David	Suzuki
20005	LeBron	James
20006	Franklin	Torgbo

Grades

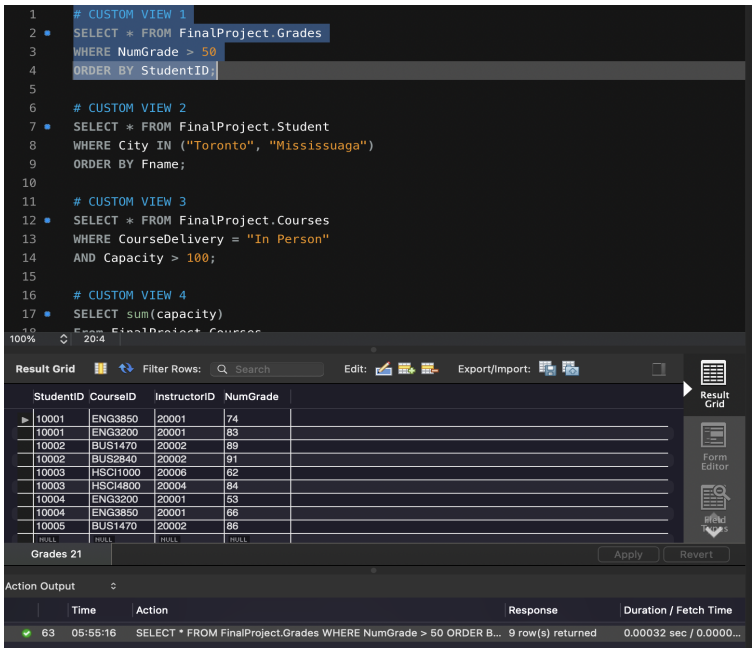
StudentID	CourseID	InstructorID	NumGrade
10001	ENG3200	20001	83
10001	ENG3850	20001	74
10002	BUS2840	20002	91
10002	BUS1470	20002	89
10003	HSCI4800	20004	84
10003	HSCI1000	20006	62
10004	ENG3200	20001	53
10004	ENG3850	20001	66
10005	BUS2840	20002	41
10005	BUS1470	20002	86
10006	NUR1000	20006	33
10006	NUR4000	20005	27

Part C: Views

(3points)

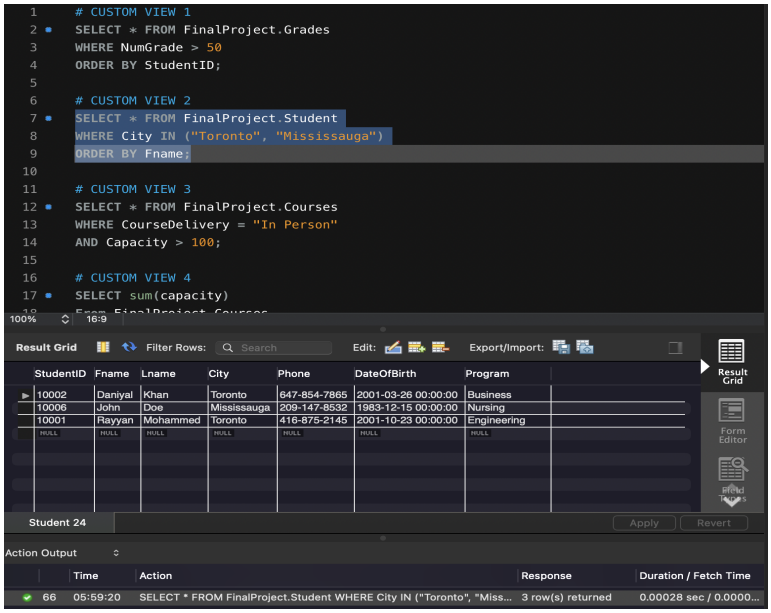
- Create (write English description and SQL syntax) of 10 views that a user of the database system would find useful. From these 10 views, the first 5 are common for all groups, and you can create your own views for the remaining ones.
- ▪View 1: Computes a join of at least three tables
- ▪View 2: Uses nested queries with the ANY or ALL operator and uses a GROUP BY clause
- ▪View3: A correlated nested query
- ▪View 4: Uses a FULL JOIN
- ▪View 5: Uses nested queries with any of the set operations UNION, EXCEPT, or INTERSECT

Custom Views 1:



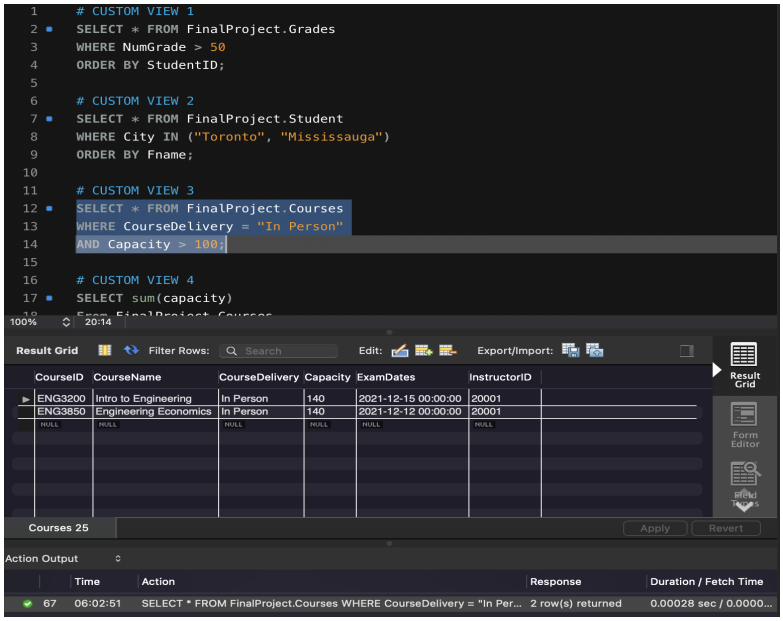
This view displays the Grades table for the students who have a grade higher than 50. It orders the table by the StudentID. This view is useful to our project because it helps us to see which students are passing the courses.

Custom Views 2:



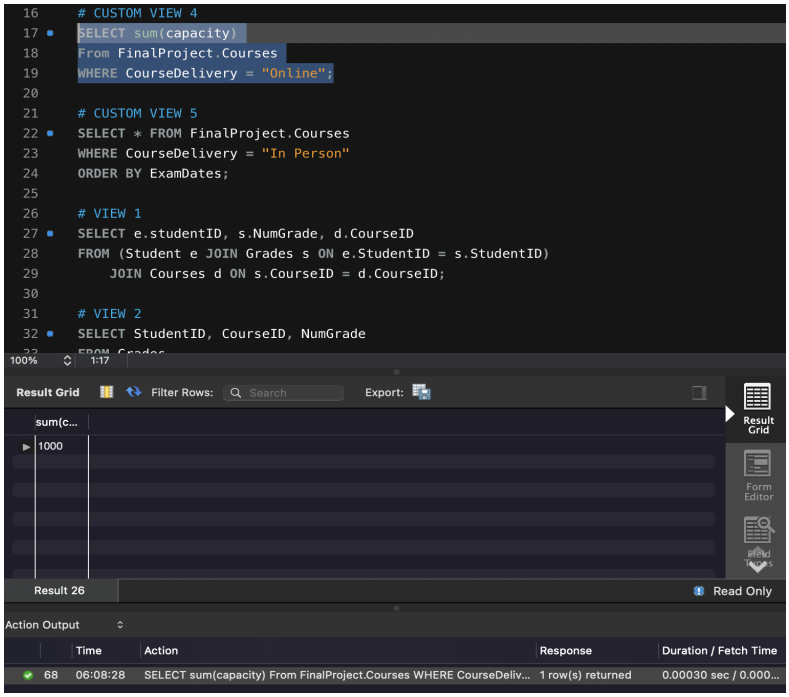
This view gets information from the Student table and if they live in the city Toronto or Mississauga the data is ordered by their first name. This view is key to our project because it allows us to see which students need to commute to classes.

Custom Views 3:



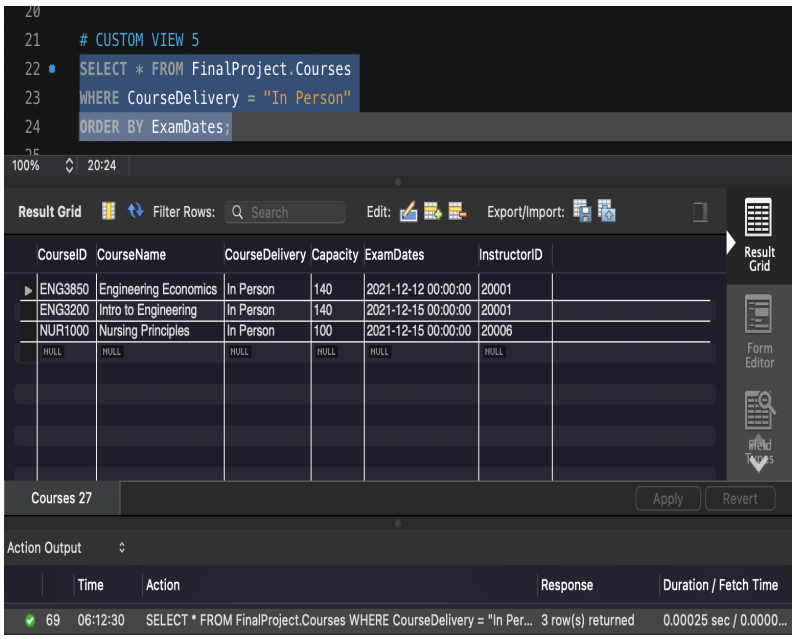
This view gets information from the Courses table and displays the in person classes with a capacity greater than 100. This view is useful because we can see which classes are in person and how many are attending with new covid laws and restrictions.

Custom Views 4:



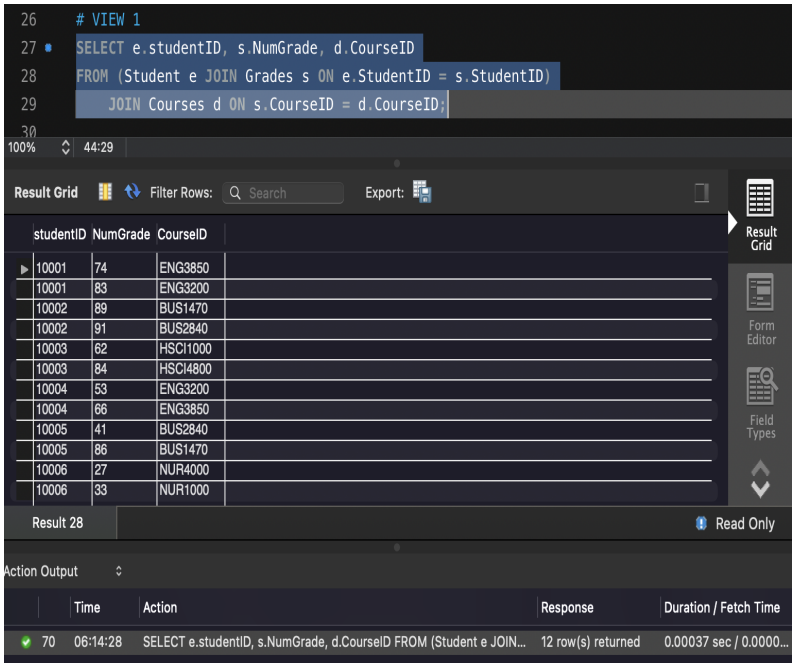
This view checks the total capacity under the Courses table where the classes are online. It's useful to see which classes are online and determine the amount of students attending.

Custom Views 5:



This view gets information from the Courses table and displays the in person classes, ordering them by the exam dates. This view is useful for students that need to go in on the dates published.

View 1:



```
# VIEW 1
SELECT e.studentID, s.NumGrade, d.CourseID
FROM (Student e JOIN Grades s ON e.StudentID = s.StudentID)
    JOIN Courses d ON s.CourseID = d.CourseID;
```

Result Grid

studentID	NumGrade	CourseID
10001	74	ENG3850
10001	83	ENG3200
10002	89	BUS1470
10002	91	BUS2840
10003	62	HSCI1000
10003	84	HSCI4800
10004	53	ENG3200
10004	66	ENG3850
10005	41	BUS2840
10005	86	BUS1470
10006	27	NUR4000
10006	33	NUR1000

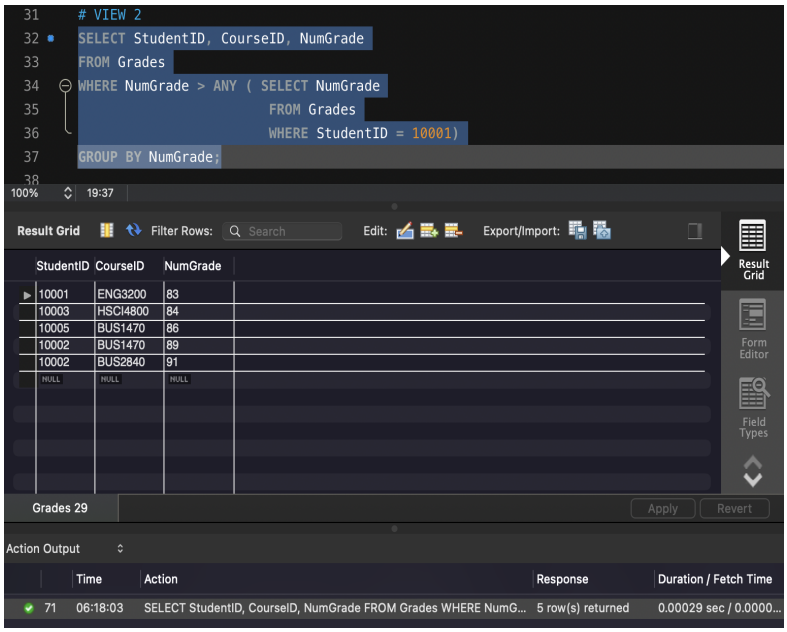
Result 28 — Read Only

Action Output

	Time	Action	Response	Duration / Fetch Time
70	06:14:28	SELECT e.studentID, s.NumGrade, d.CourseID FROM (Student e JOIN...	12 row(s) returned	0.00037 sec / 0.0000...

This view joins Students, Grades and Courses table together while displaying StudentID, NumGrade and CourseID Points.

View 2:



```
# VIEW 2
SELECT StudentID, CourseID, NumGrade
FROM Grades
WHERE NumGrade > ANY ( SELECT NumGrade
                       FROM Grades
                       WHERE StudentID = 10001)
GROUP BY NumGrade;
```

Result Grid

StudentID	CourseID	NumGrade
10001	ENG3200	83
10003	HSCI4800	84
10005	BUS1470	86
10002	BUS1470	89
10002	BUS2840	91
NULL	NULL	NULL

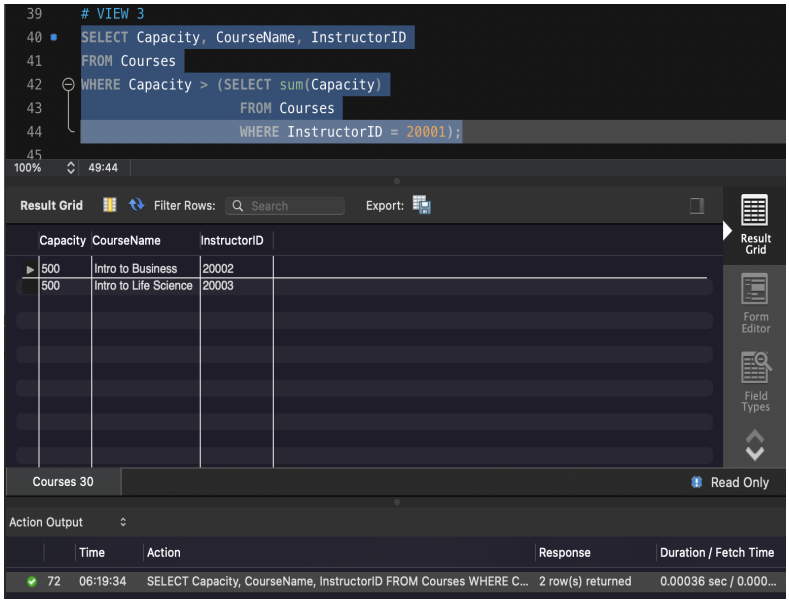
Grades 29 — Apply — Revert

Action Output

	Time	Action	Response	Duration / Fetch Time
71	06:18:03	SELECT StudentID, CourseID, NumGrade FROM Grades WHERE NumG...	5 row(s) returned	0.00029 sec / 0.0000...

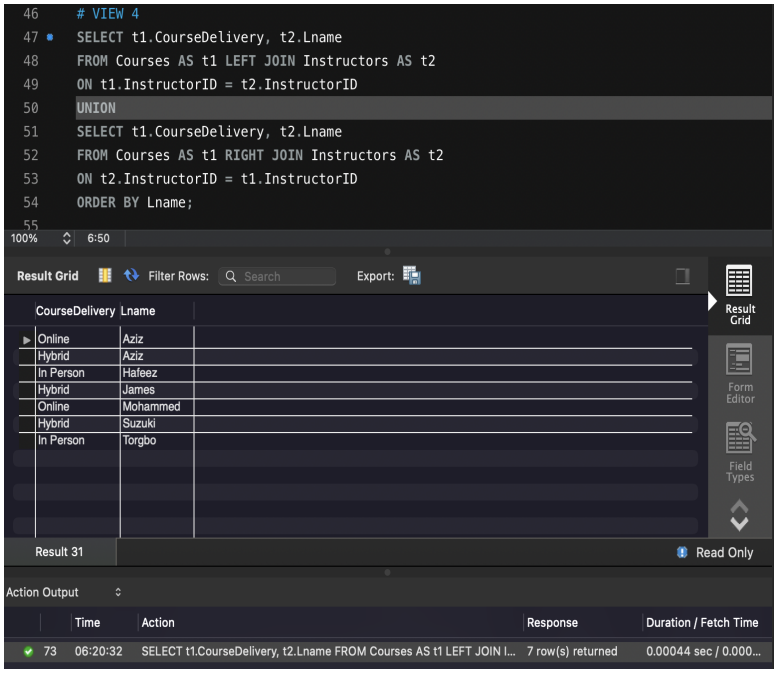
This view displays any student who has a higher grade than the lowest grade for student “10001” which is 74.

View 3:



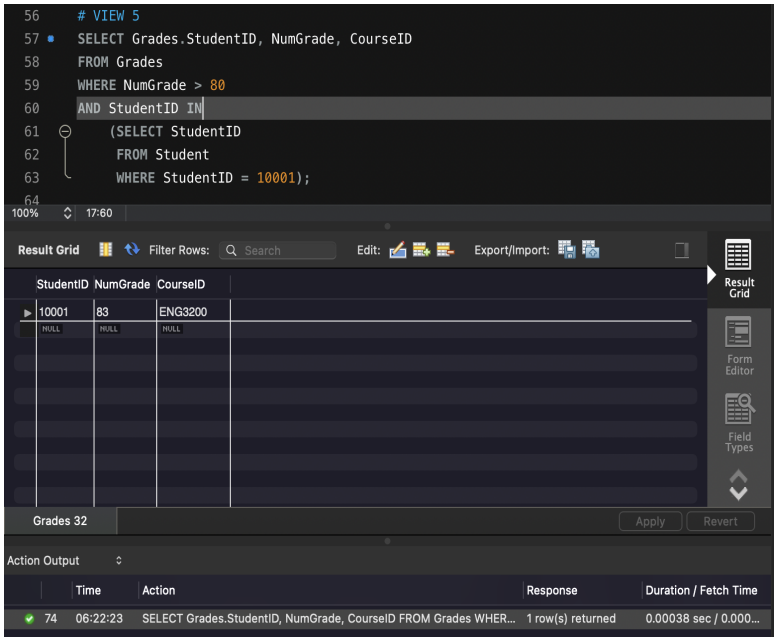
Displays the Capacity, InstructorID and CourseName for the instructors who have a higher total capacity than instructor “20001” which is 380.

View 4:



Displays Left, Right Join and Union to emulate a Full Join in MySQL Workbench. Displays course delivery and last name by combining Courses table and Instructors table.

View 5:



Displays emulation of an Intercept in MySQL Workbench by using IN. Displays StudentID,NumGrade and CourseID where the student with id “10001” got a higher grade than 80.

ALL VIEWS:

```
1    # CUSTOM VIEW 1
2    SELECT * FROM FinalProject.Grades
3    WHERE NumGrade > 50
4    ORDER BY StudentID;
5
6    # CUSTOM VIEW 2
7    SELECT * FROM FinalProject.Student
8    WHERE City IN ("Toronto", "Mississauga")
9    ORDER BY Fname;
10
11   # CUSTOM VIEW 3
12   SELECT * FROM FinalProject.Courses
13   WHERE CourseDelivery = "In Person"
14   AND Capacity > 100;
15
16   # CUSTOM VIEW 4
17   SELECT sum(capacity)
18   From FinalProject.Courses
19   WHERE CourseDelivery = "Online";
20
21   # CUSTOM VIEW 5
22   SELECT * FROM FinalProject.Courses
23   WHERE CourseDelivery = "In Person"
24   ORDER BY ExamDates;
25
```

```sql
# VIEW 1
SELECT e.studentID, s.NumGrade, d.CourseID
FROM (Student e JOIN Grades s ON e.StudentID = s.StudentID)
    JOIN Courses d ON s.CourseID = d.CourseID;

# VIEW 2
SELECT StudentID, CourseID, NumGrade
FROM Grades
WHERE NumGrade > ANY ( SELECT NumGrade
                       FROM Grades
                       WHERE StudentID = 10001)
GROUP BY NumGrade;

# VIEW 3
SELECT Capacity, CourseName, InstructorID
FROM Courses
WHERE Capacity > (SELECT sum(Capacity)
                  FROM Courses
                  WHERE InstructorID = 20001);

# VIEW 4
SELECT t1.CourseDelivery, t2.Lname
FROM Courses AS t1 LEFT JOIN Instructors AS t2
ON t1.InstructorID = t2.InstructorID
UNION
SELECT t1.CourseDelivery, t2.Lname
FROM Courses AS t1 RIGHT JOIN Instructors AS t2
ON t2.InstructorID = t1.InstructorID
ORDER BY Lname;
```

```sql
# VIEW 5
SELECT Grades.StudentID, NumGrade, CourseID
FROM Grades
WHERE NumGrade > 80
AND StudentID IN
    (SELECT StudentID
     FROM Student
     WHERE StudentID = 10001);
```

Part D: E-R diagram: (5points)

- Create an ER schema diagram for your project database.