Lab Exercise 1
Data Warehousing

CS ELEC 1C

Alessandro Andrei Araza Edison Javier Alyza Paige Ng

Normalized Schema Diagram

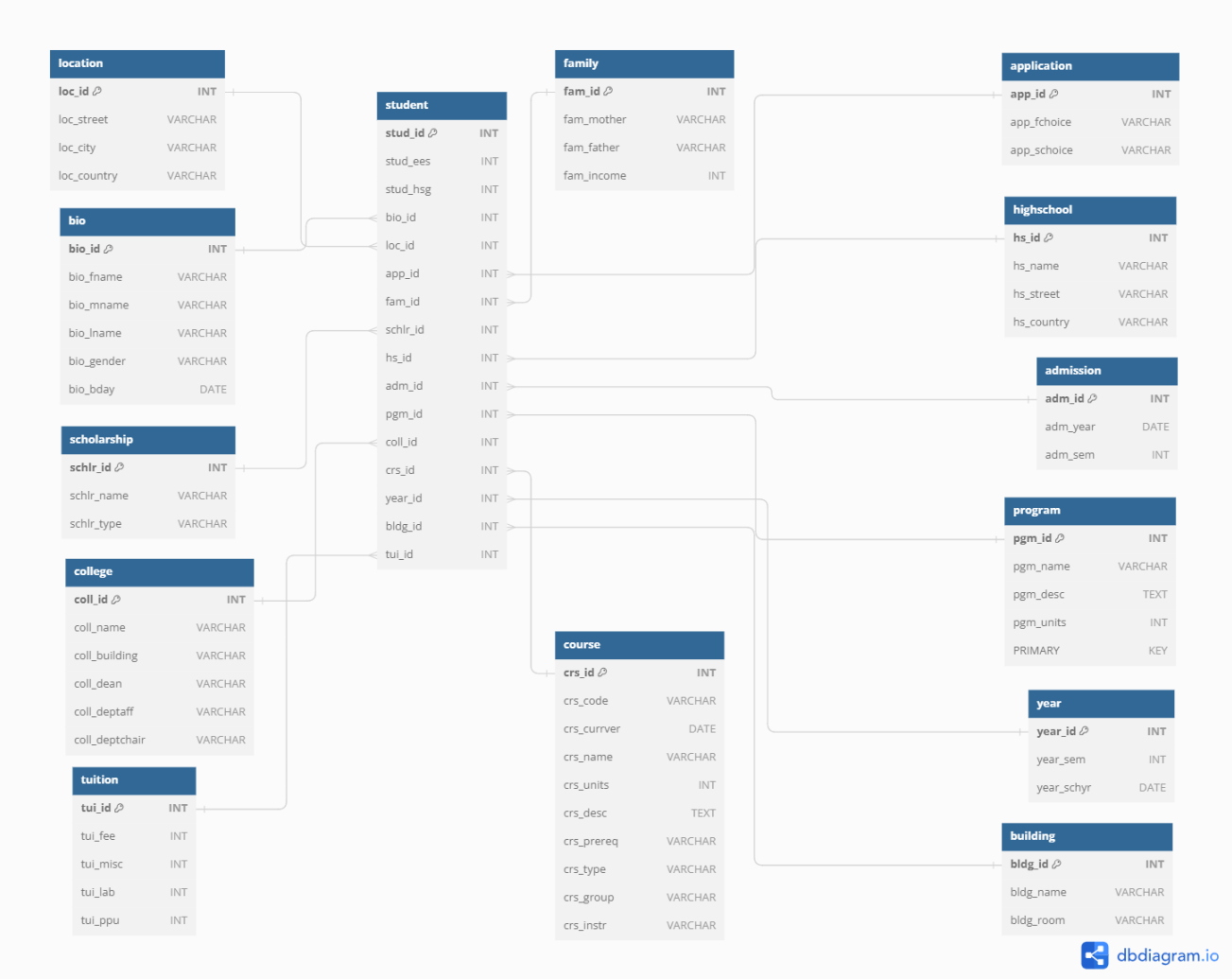


Figure 1.1: This schema was built using the engine of DB Diagram, the schema is available through this link

Questions

Question 1

What is the average tuition fee?

Calculating only the average of the tuition fee itself, implying that the miscellaneous fees, lab fees, and price per unit would be disregard, means that this is simply getting the average of one table attribute.

Solution:

```
select avg(tui_fee) as "Average Tuition Fee" from tuition;
```

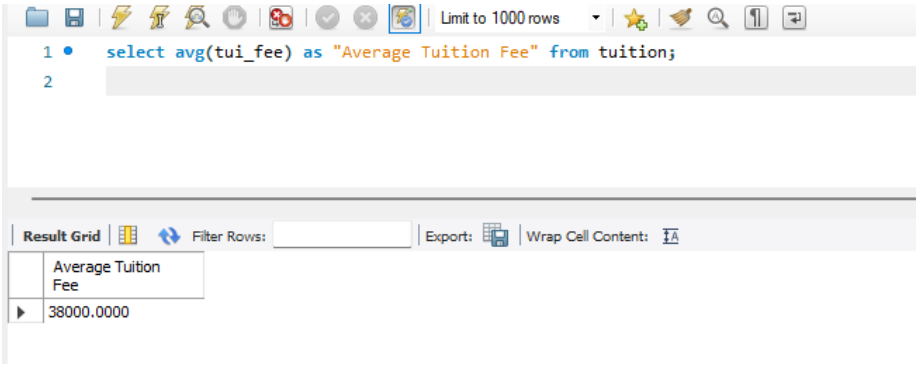


Figure 2.1: Question 1 Query and Output

Question 2

What is the average total school fees?

Using the `SUM` will add the various components of school fees and grouping it by Student ID will make it one row per student, and using the `AVG` will get the average total school fees in general

Solution:

```
CREATE VIEW student_fees AS
SELECT stud_id, SUM(tui_fee + tui_misc + tui_lab + tui_ppu)
AS total_school_fees
FROM student
JOIN tuition ON student.tui_id = tuition.tui_id
GROUP BY stud_id;

SELECT AVG(total_school_fees) AS "Average Total School Fees" FROM student_fees;
```

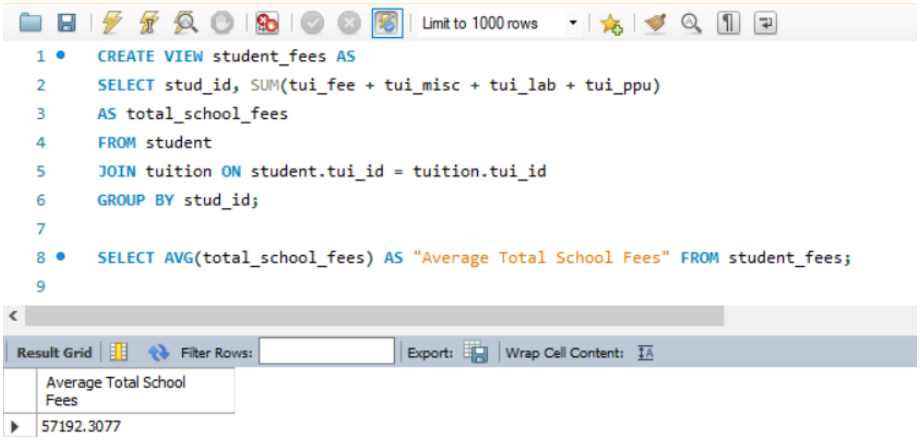


Figure 2.2: Question 2 Query and Output

Question 3

How many students are enrolled per subject?

We started by making a view called `enrolled` that contains the count of the number of `bio_ids` for each `crs_id` by using the `group by` keywords along with the `count()` function

Solution:

```
create view enrolled as
select count(bio_id) as Count, crs_id from student group by crs_id;

select sum(Count) as "Final Count", crs_code as "Course Code" from enrolled
join (select * from course) as cr on enrolled.crs_id = cr.crs_id group by crs_code;
```

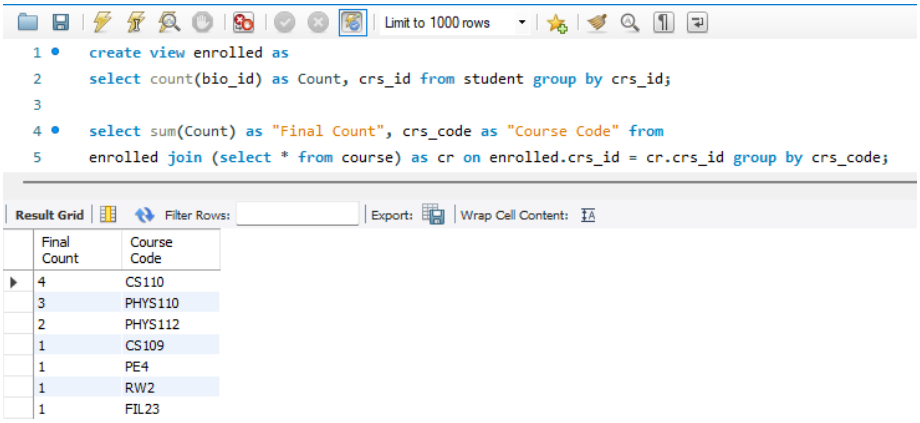


Figure 2.3: Question 3 Query and Output

Question 4

How many subjects does Taylor Sheesh have?

By using the `distinct` keyword to only get the courses that are connected to Taylor Sheesh's `bio_id`, we can acquire the count of the courses that Taylor is enrolled to without having to accidentally count duplicates of the same course.

A view called `sheesh` is also created to gather the details of Taylor Sheesh's bio

Solution:

```
create view sheesh as
select * from bio where bio_fname = "Taylor" and bio_lname = "Sheesh";

select count(distinct crs_id) as "Taylor's Subjects" from student where
bio_id = (select bio_id from sheesh);
```

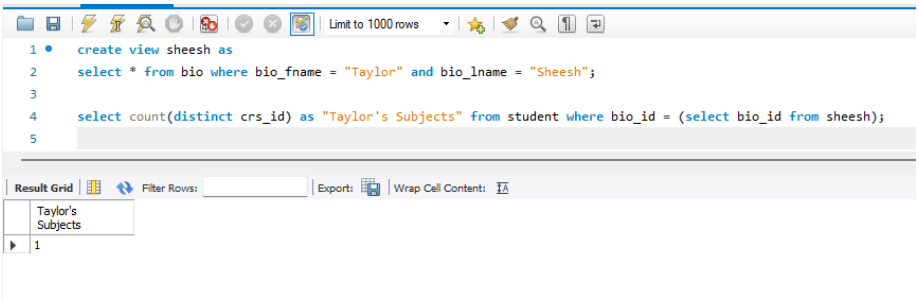


Figure 2.4: Question 4 Query and Output

Question 5

How many students have the same mother but different fathers?

Doing a self-join on family table created two instances to check if they have the same mother but different fathers with the help of the `DISTINCT` keyword to help eliminate duplicates

Solution:

```
CREATE VIEW same_mother AS
SELECT DISTINCT f1.fam_mother, f1.fam_father, student.fam_id
FROM family f1
JOIN student ON f1.fam_id = student.fam_id
JOIN family f2 ON f1.fam_mother = f2.fam_mother AND f1.fam_father != f2.fam_father;

SELECT COUNT(*) AS "Students with Same Mothers but Different Fathers" FROM same_mother;
```

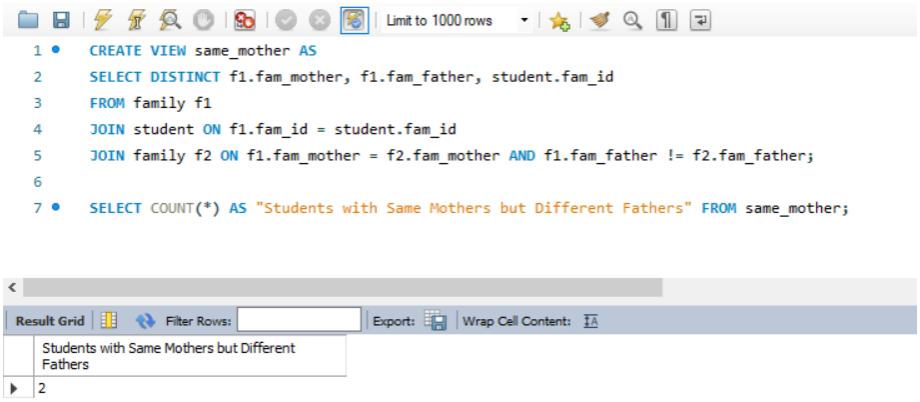


Figure 2.5: Question 5 Query and Output

Question 6

What are the combinations of semester and school year?

Getting the unique combinations of semester and school year values can be easily acquired by using the `group by` keyword while querying. This table would show all combinations of school year and semester present in the database.

Solution:

```
create view uniq as
select count(*) from year group by year_sem, year_schyr;

select count(*) as Combinations from uniq;
```

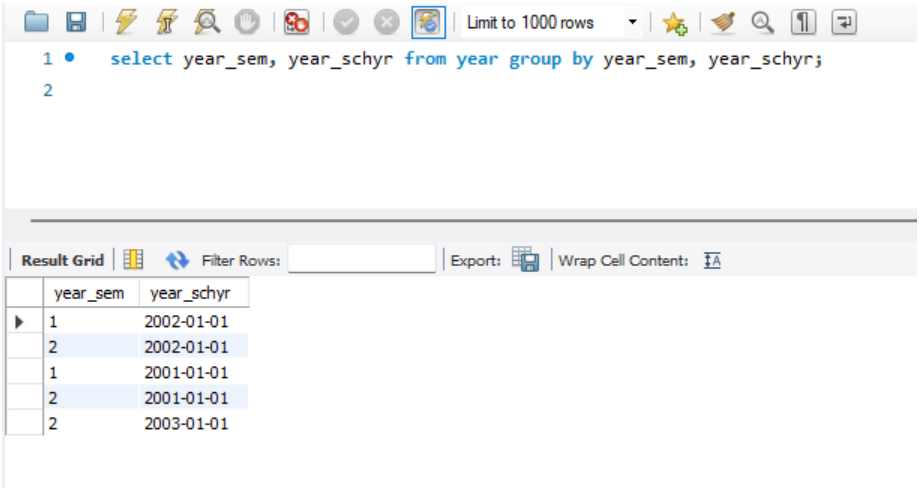


Figure 2.6: Question 6 Query and Output

Question 7

How many students does International School of the National Artistic Arts University have?

The I.S.N.A.A.U. has a total of 4 unique students in the database

We acquired this data by first creating the view `isnaau` to store the details of the school in the `highschool` table by matching its name in the records

This view would then be used to get the `hs_id` attribute and match it with the student records in the `student` table. To make sure that the `count()` function wouldn't recount duplicates of the same student, using the `distinct` keyword is a must.

Solution:

```
create view isnaau as
select * from highschool where hs_name
= "International School of the National Artistic Arts University";

select count(distinct stud_id) as "Students" from student
where hs_id = (select hs_id from isnaau);
```

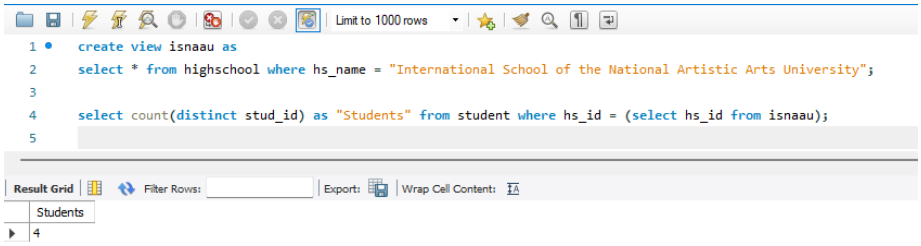


Figure 2.7: Question 7 Query and Output

Question 8

How many courses are held in all the buildings?

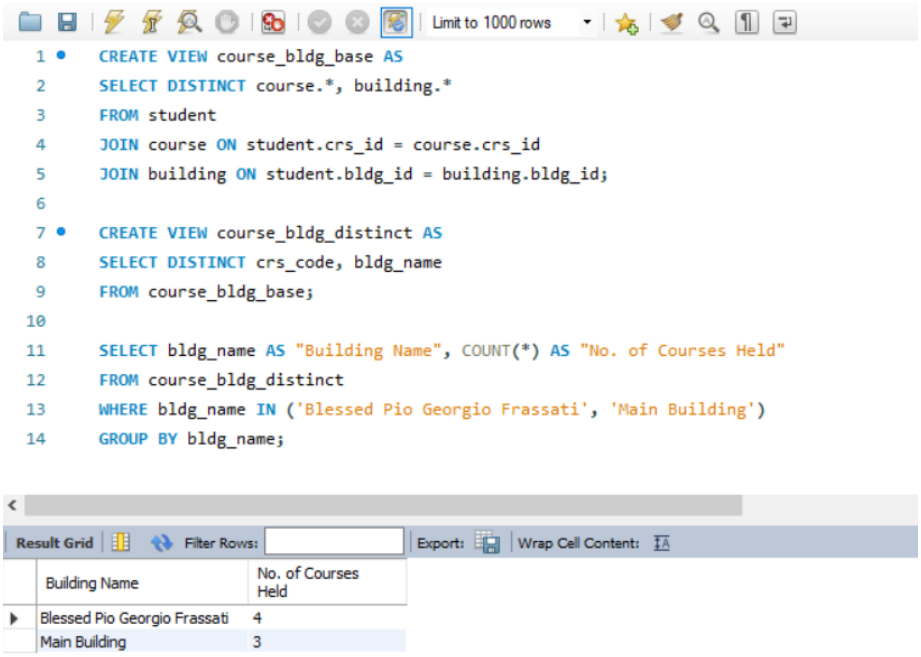
Two views were created: (1) For joining the course and building tables and (2) For eliminating duplicates, then it counted the courses per building it is being held

Solution:

```
CREATE VIEW course_bldg_base AS
SELECT DISTINCT course.*, building.*
FROM student
JOIN course ON student.crs_id = course.crs_id
JOIN building ON student.bldg_id = building.bldg_id;

CREATE VIEW course_bldg_distinct AS
SELECT DISTINCT crs_code, bldg_name
FROM course_bldg_base;

SELECT bldg_name AS "Building Name", COUNT(*) AS "No. of Courses Held"
FROM course_bldg_distinct
WHERE bldg_name IN ('Blessed Pio Georgio Frassati', 'Main Building')
GROUP BY bldg_name;
```



```
1  CREATE VIEW course_bldg_base AS
2  SELECT DISTINCT course.*, building.*
3  FROM student
4  JOIN course ON student.crs_id = course.crs_id
5  JOIN building ON student.bldg_id = building.bldg_id;
6
7  CREATE VIEW course_bldg_distinct AS
8  SELECT DISTINCT crs_code, bldg_name
9  FROM course_bldg_base;
10
11 SELECT bldg_name AS "Building Name", COUNT(*) AS "No. of Courses Held"
12 FROM course_bldg_distinct
13 WHERE bldg_name IN ('Blessed Pio Georgio Frassati', 'Main Building')
14 GROUP BY bldg_name;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Building Name	No. of Courses Held
Blessed Pio Georgio Frassati	4
Main Building	3

Figure 2.8: Question 8 Query and Output

Question 9

What is the average family income per scholarship?

Note:-

We decided not to separate them by scholarship type; i.e. merit-based, needs-based, etc. Since the question only refers to "**per scholarship**", we considered them to be one, regardless if it is merit or needs-based.

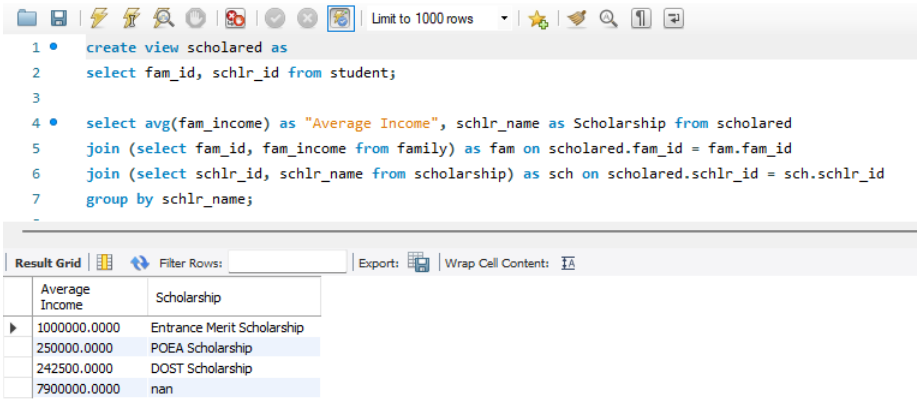
We isolated the students' `fam_id` and `schlr_id` foreign keys. Then query for a joint table that joins the `fam_income` and `schlr_name` from the already aforementioned IDs of the family and scholarship tables. By using the `group by` keywords along with the `avg()` function to the `fam_income` attribute, we can query for the average household income for each scholarship.

Solution:

```
create view scships as
select * from scholarship group by schlr_name;

create view scholared as
select fam_id, schlr_id from student;

select avg(fam_income) as "Average Income", schlr_name as Scholarship from scholared
join (select fam_id, fam_income from family) as fam
    on scholared.fam_id = fam.fam_id
join (select schlr_id, schlr_name from scholarship) as sch
    on scholared.schlr_id = sch.schlr_id
group by schlr_name;
```



```
create view scholared as
select fam_id, schlr_id from student;

select avg(fam_income) as "Average Income", schlr_name as Scholarship from scholared
join (select fam_id, fam_income from family) as fam on scholared.fam_id = fam.fam_id
join (select schlr_id, schlr_name from scholarship) as sch on scholared.schlr_id = sch.schlr_id
group by schlr_name;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Average Income	Scholarship
1000000.0000	Entrance Merit Scholarship
250000.0000	POEA Scholarship
242500.0000	DOST Scholarship
7900000.0000	nan

Figure 2.9: Question 9 Query and Output

Question 10

Whose family has the least amount of money left after paying total school fees?

Start with a view named `base` where all the necessary IDs can be fetched. We can use this view for the next view called `details` to join in one table all the needed details such as the family income, course units, and all the fees in the tuition table. This would make it easier later to total expenses and leftover money since everything is all in one table.

The `expenses` view cleans up the tables from earlier and provides a cleaner version of the name, as well as the calculation of the total for each scholarship adjusted for the enrolled units of the student.

We can then process the `expenses` view further by using the `sum()` function and grouping them by student names since students can be enrolled in more than one course, this would add up hence using the `sum()` function to aggregate all their expenses. If we subtract the sum total of all their expenses from their family income, we get the amount of money they have left after paying total school fees. We can name this as the `total` view

Knowing which family would have the least amount of money after paying is a matter of using the `min()` function to the `total` view

Solution:

```
create view base as
select stud_id, bio_id, fam_id, crs_id, tui_id as tuiid from student;

create view details as
select * from base
right join
(select bio_id as biod, bio_fname as fname, bio_lname as lname from bio where bio_id = bio_id)
as bioo on base.bio_id = bioo.biod
right join
(select fam_id as famid, fam_income as Income from family) as fam on base.fam_id = fam.famid
right join
(select crs_id as crsid, crs_units as Units from course) as crs on base.crs_id = crs.crsid
right join
(select * from tuition) as tui on base.tuiid = tui.tui_id;

create view expenses as
select concat(fname, " ", lname) as Name, Income,
(tui_fee + tui_misc + tui_lab + tui_ppu*Units) as Total from details;

create view total as
select Name, Income as "Family Income", sum(Total) as "Sum Total", (Income - sum(Total))
```

```
as Money from expenses group by Name;

select * from total where Money = (select min(Money) from total);
```

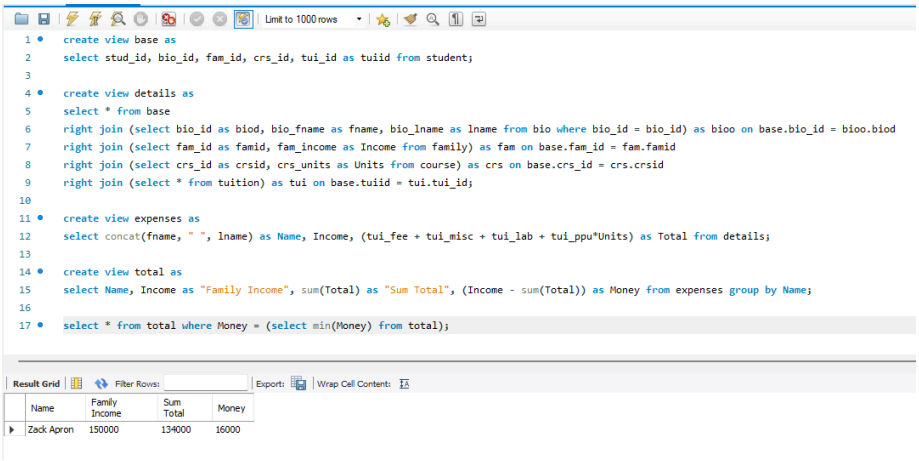


Figure 2.10: Question 10 Query and Output

Question 11

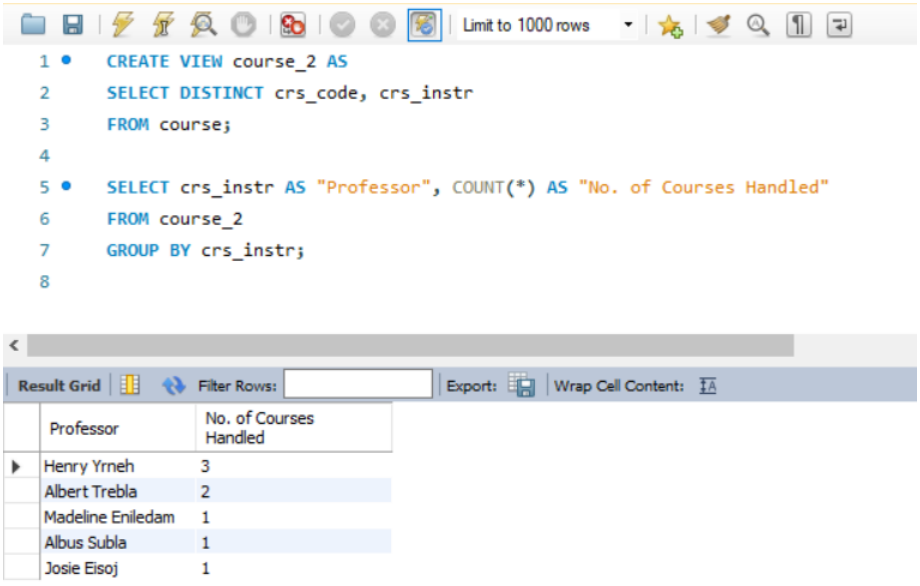
How many courses do each professor handle?

A view was created using the `DISTINCT` keyword to eliminate duplicate values and be used as the basis for counting the courses each professor handles

Solution:

```
CREATE VIEW course_2 AS
SELECT DISTINCT crs_code, crs_instr
FROM course;

SELECT bldg_name AS "Professor", COUNT(*) AS "No. of Courses Handled"
FROM course_2
GROUP BY crs_instr;
```



```
CREATE VIEW course_2 AS
SELECT DISTINCT crs_code, crs_instr
FROM course;

SELECT crs_instr AS "Professor", COUNT(*) AS "No. of Courses Handled"
FROM course_2
GROUP BY crs_instr;
```

Professor	No. of Courses Handled
Henry Yrneh	3
Albert Trebla	2
Madeline Eniledam	1
Albus Subla	1
Josie Eisoj	1

Figure 2.11: Question 11 Query and Output

Question 12

What is the average high school grade for each school?

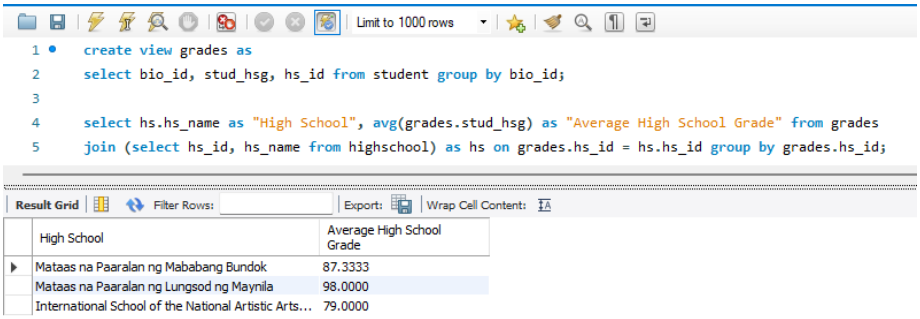
I needed to first create a `grades` view to get the IDs and attributes needed for further querying. These columns include the bio ID, student high school grade, high school ID and group them by `bio_id` to remove any duplicates of the same student (should there ever be one).

Using the `grades` view, we can leverage this to query for an `avg()` by joining the `hs_name` to the table. This would allow us to show more clearly in the output which school has which average grade. By using the `avg()` function with the `group by grades.hs_id`, we can get the average **per school** instead of the overall average of all the records.

Solution:

```
create view grades as
select bio_id, stud_hsg, hs_id from student group by bio_id;

select hs.hs_name as "High School", avg(grades.stud_hsg) as
"Average High School Grade" from grades
join (select hs_id, hs_name from highschool) as hs
    on grades.hs_id = hs.hs_id group by grades.hs_id;
```



```
create view grades as
select bio_id, stud_hsg, hs_id from student group by bio_id;

select hs.hs_name as "High School", avg(grades.stud_hsg) as "Average High School Grade" from grades
join (select hs_id, hs_name from highschool) as hs on grades.hs_id = hs.hs_id group by grades.hs_id;
```

High School	Average High School Grade
Mataas na Paaralan ng Mababang Bundok	87.3333
Mataas na Paaralan ng Lungsod ng Maynila	98.0000
International School of the National Artistic Arts...	79.0000

Figure 2.12: Question 12 Query and Output

Thoughts and Experience

During Lab Exercise #1, we engaged in the hands-on application of SQL queries. It was a valuable learning experience as we practiced writing SQL statements to retrieve, filter, and aggregate data from various database tables. Last year, this concept was taught to us, so it was a struggle to review and apply these topics to the task. The exercise also allowed the group to solidify their understanding of SQL fundamentals, such as SELECT statements, JOIN operations, GROUP BY clauses, and we can't forget the other keywords like DISTINCT, AS, and many more. We also gained confidence in constructing queries to extract meaningful insights from a database. Lab Exercise #1 provided a practical foundation for working with databases and was a significant step forward in the group's SQL journey.

Participating in the Laboratory Exercise on normalization and MySQL queries has been an enlightening experience that has significantly contributed to our understanding of the Data Warehousing curriculum. This practical exercise reinforced the theoretical concepts we've learned in class. It allowed us to appreciate the real-world applications of data normalization and the importance of well-structured databases in data warehousing.

One of the key takeaways from this exercise was realizing how critical proper data normalization is in designing a database schema for a data warehouse. It became clear that organizing data efficiently by eliminating data redundancy reduces storage requirements and enhances data integrity and query performance. It aligns with the fundamental principles of data warehousing, which aim to create a centralized repository of high-quality data that supports analytical and reporting needs.

Working with MySQL queries in the context of the exercise was equally insightful. We had the opportunity to practice crafting SQL statements to retrieve, filter, and manipulate data from normalized tables. This hands-on experience highlighted the significance of query optimization in data warehousing, as efficient queries are essential for extracting meaningful insights from large datasets. The practical aspect of writing these queries deepened our understanding and honed our SQL skills, a crucial tool for any data warehousing professional.

Overall, this Laboratory Exercise was a bridge between the theoretical knowledge acquired in class and its practical application in data warehousing. It emphasized the importance of data modeling, normalization, and efficient querying, all of which are foundational concepts in the curriculum. We feel better equipped to tackle complex data warehousing projects, confident in our abilities to design well-structured databases and write optimized SQL queries to extract valuable information from them. This hands-on experience has truly enriched our learning journey in Data Warehousing.