Project Report

James Kyle Harrison

Georgia Southern University

Table of Contents

Table of Figures

Introduction

Data Warehousing is the process of constructing and using a data warehouse typically in a professional environment so that new knowledge can be obtained from using the warehouse to quickly generate analytical results. Data warehouses are powerful in environments with a lot of operational data such as manufacturing systems and social media systems where there is so much data that a single-dimensional model just won't cut it. Although they can take a bit of time to construct, an upgrade to a multi-dimensional data warehouse is highly desirable. In order to keep up with business needs many are requiring data warehouses, which includes the University of Savannah at Richmond Hill who require a data warehouse for their graduated student data. Data warehouses are complicated in that they are made exclusively for a single parties use, meaning that one can't just take a pre-made data warehouse and use it for their own data. It requires planning and specifications in order to successfully create a new data warehouse. To help demonstrate why a data warehouse must be made from the ground up on a per-use basis and the basics of a data warehouse we shall create a data warehouse to meet the specifications of the University of Savannah at Richmond Hill. This document describes the creation of a data warehouse for the University of Savannah at Richmond Hill from creating a blueprint for a multidimensional model, to the implementation of a multidimensional star schema, and finally the creation of an interface for users to query the multidimensional model through an interpreter that converts the OLAP operations into SQL queries quickly and easily for online analytical processing.

Data Model Design

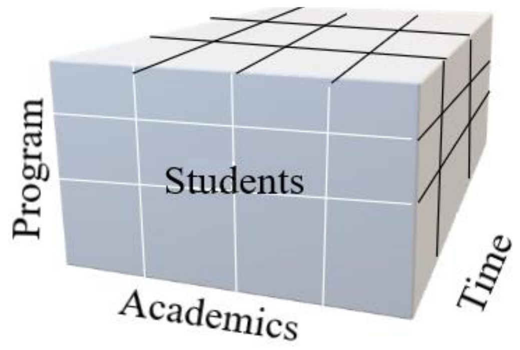
Schema

When it comes to designing a data warehouse for the University of Savannah at Richmond Hill (USRH), there is no denying that the subject of said data warehouse will be the students. With the selected data sample and the requirements of having to study the students by major, degrees, colleges, GPAs, and semesters there are a variety of ways that this can be handled. To create a solution that will meet its' parameters in a timely and efficient manner we will use attempt to go for the simplest means of reaching these goals. For this particular implementation we will move forward with developing a data warehouse using the star schema as the dimensions can be split in a simplistic manner a dictated by the groupings of analytical necessities in a hierarchy that will break down the data as low as necessary and no further.

Dimensions

This data warehouse has been elected to be a three-dimensional model to complete the specifications of the USRH. The requirements can be broken down into the dimensions of the Academics dimension representing the data dedicated to students and (for the purposes of analysis) GPA data, the Program which represents the combinations of degrees and majors that are possible by each of the specified colleges, and the Time dimension which represents the semester data such as which year and semester.

Figure 1: Students Three-Dimensional Model



To further explain the justification of this design, let us look at the dimension tables for each assigned dimension.

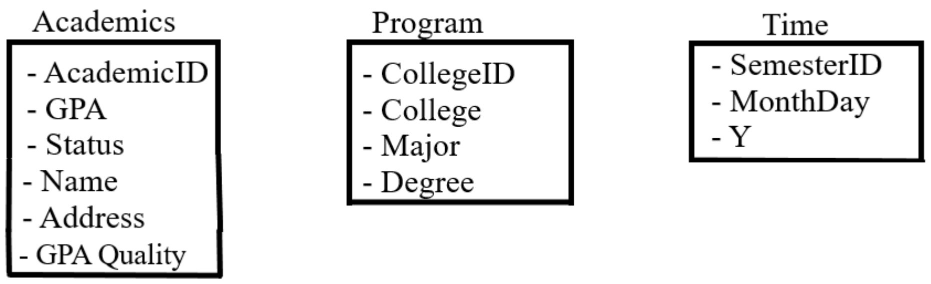


Figure 2: Dimensional Attributes

The Academics dimension uses the AcademicID attribute (otherwise known as ID in the dataset collected by university authorities) as a primary key attribute for the subject to allow for the use of all of the student enrollment information. The simplest justification for this dimension and its layout is that it allows filtering around the ‘what’ axis in this subject and creates the way for analysis on the subject of a per-student basis. This dimension is made in mind for filtering students on their GPA along with additional student personal information. The Program dimension uses the CollegeID attribute as its primary key for the subject relating to the school and degree. This ID will be used to filter for the subject’s major, degree, and college and its layout allows for filtering around the ‘where’ axis in the subject and creates a way for analysis on the premises of a per-college basis. This allows for the pre-requisite filtering requirements on a college, degree, or major basis. Although the granularity of this dimension can be changed by creating a sub-dimension for the college attribute based off of the major, the data reduction of listing the college is not necessarily required. Finally, the Time dimension uses the SemesterID attribute as it’s key for the subject of time to correctly filter using the semesters and time in general. This layout allows for filtering around the ‘when’ axis for the subject and will not need any changes to the granularity of time attributes as the analysis requirements only need to be met on a per-semester basis. This will meet the final requirement in the scope of analysis by allowing for filtering on the basis of semesters and year. With these three dimensions the following possible cuboids can be generated from analyses:

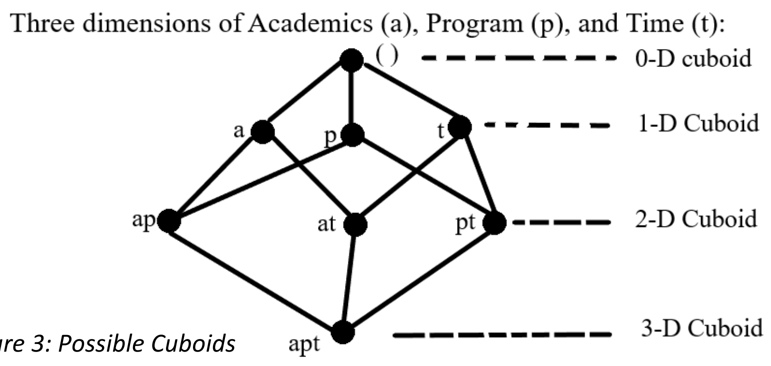

Three dimensions of Academics (a), Program (p), and Time (t):
( )
0-D cuboid
a
p
t
1-D Cuboid
ap
at
pt
2-D Cuboid
apt
3-D Cuboid

Figure 3: Possible Cuboids

Fact Table and Hierarchies

The fact table has a one-to-many relationship to the dimensions previously described, and thus the keys of these dimensions are within our fact table as this will allow for one to aggregate through some of the dimensions, but not all. Due to the lack of summable data within this data warehouse, the fact table will be just a set of keys to its dimensions and in result create a star schema with a fact table that will connect to the primary keys of each of the three dimensions by using them as foreign keys. This creates a fact table as described below:

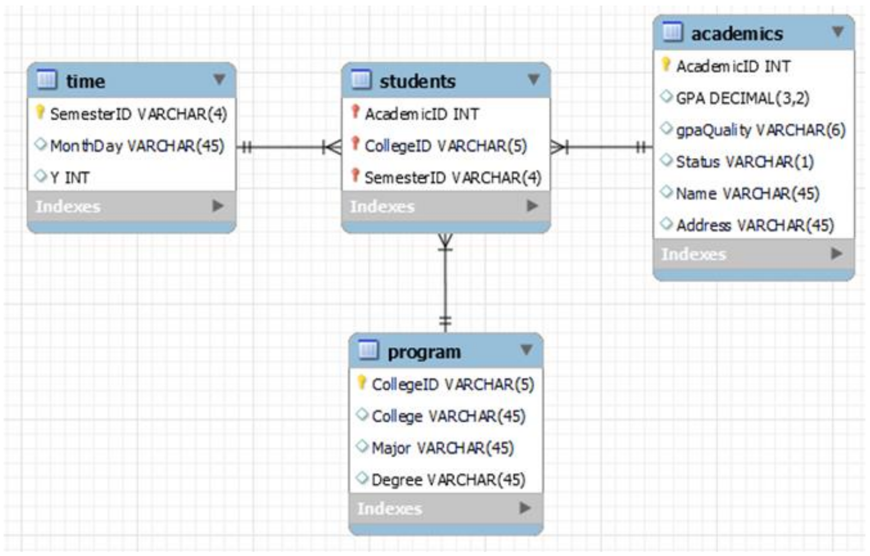


Figure 4: Fact Table

Now that the fact table and its connections to the dimension tables have been established let's take a look at the concept and schema hierarchies for each of the following dimensions. Starting with the time dimension's hierarchies we can see a naturally total order of all concepts where the Semester ID when matched with the subjects will lead into identifying the year and then semester in the concept hierarchy, and the schema hierarchy will follow a hierarchy that moves from the lowest concepts to the highest concept in a simple manner.

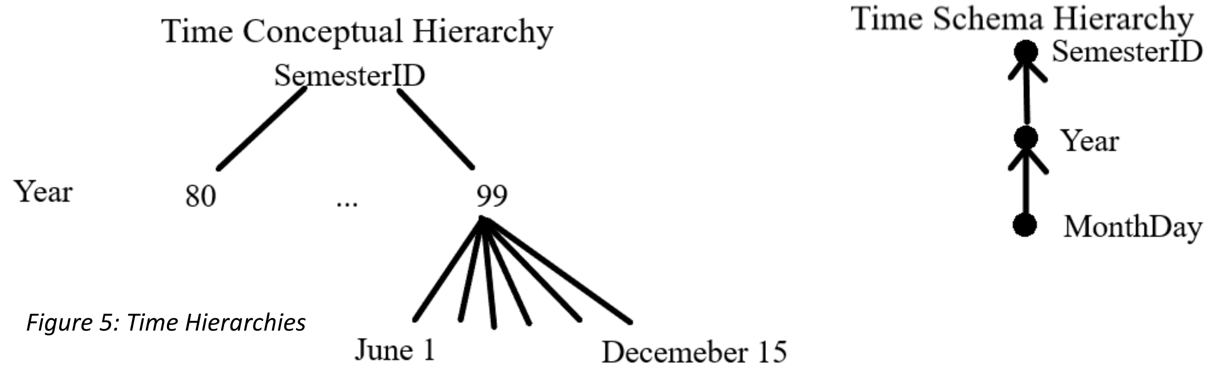


Figure 5: Time Hierarchies

The program dimensions hierarchies will also be quite similar to the time dimensions hierarchies in that it too uses a naturally total order of all concepts. The CollegeID when compared to the subjects will find the college that the student is in. From here we can filter out what major they are by looking at the majors in the school, and finally be able to do the same

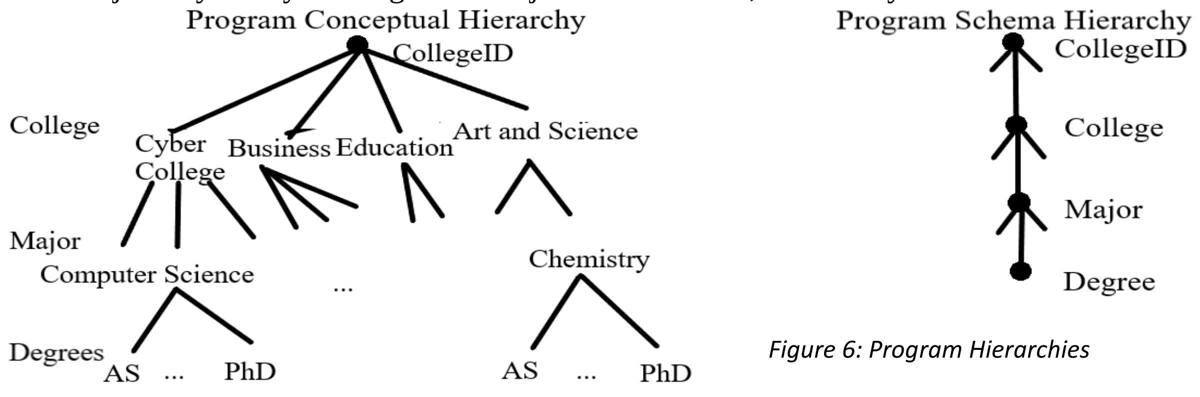
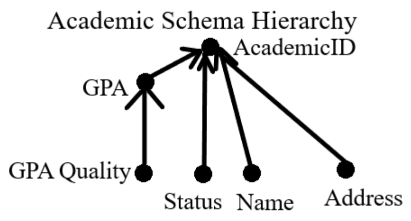


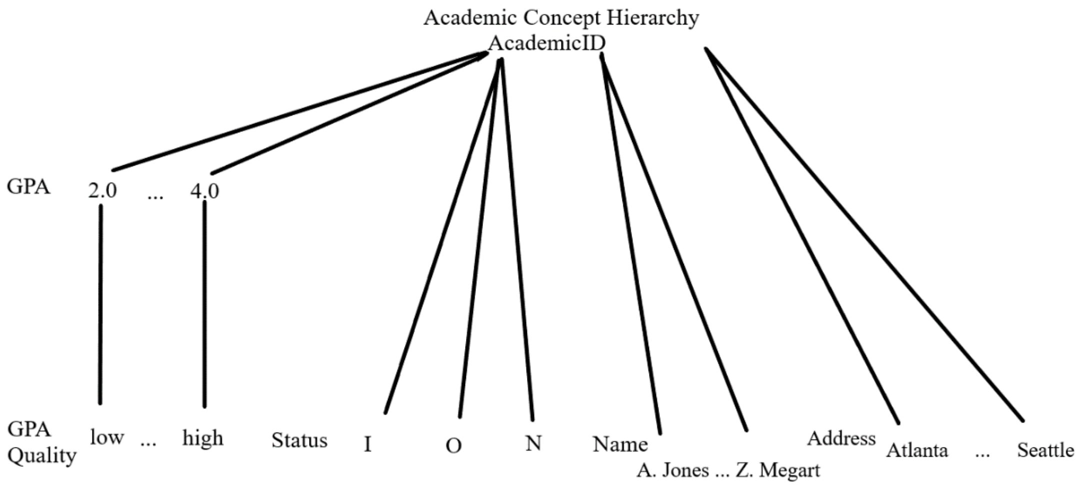
Figure 6: Program Hierarchies

with the degrees as not all degrees will apply to all majors (ex: you cannot get a PhD in the College of Education or MBA in Chemistry). This results in having another schema hierarchy where there exists an entire hierarchy that moves from the lowest concepts to the highest concept within the dimension.

Finally, there is the academics dimension, where the hierarchies are a bit more convoluted as this dimension is a partially ordered hierarchy, where only one set of footsteps is in a hierarchy. We are able to at least order GPA and the GPA Qualities such that this information can be derived from by declaring ranges for GPAs such that they can be categorized, however that does not seem to be the case for address and status based off of the provided data set. One would expect to be able to derive their student status of being an international or out-of-state student based off of their address, but that does not appear to be the case, and thus they must remain out-of-hierarchy.

Figure 7: Academic Hierarchies

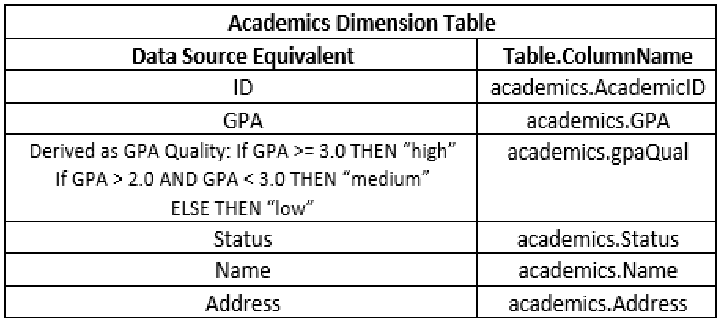




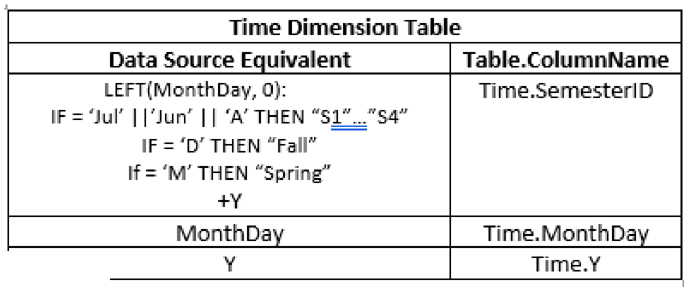
Data Model Implementation

Now that the modeling of the data warehouse has been completed it is time to being preparing to implement the model and the given data into the model. To prepare the data for the model we must first do a bit of extra processing on the data that we have been given. The data given comes from a few different sources that were able to successfully be merged into one file, however there are still some faults that can be found in this data. For example, the student IDs are not unique and there are a few entries that have the same ID; another issue that can be seen is also in the different ways that some of the student majors have been written ('business admin' vs 'b administration'). Although the IDs can be fixed using a function to ensure that all IDs are made unique, creating a trained neural network to deal with the pedantic differences in major declarations would require significantly more time than it would to just fix the few instances of this happening manually, thus the students' IDs will be fixed through automation and extreme outliers of the student majors will be changed manually.

The data is almost completely ready to be implemented into our data warehouse, however there are still attributes that must be addressed. Much of the data will fortunately be able to be converted directly from the dataset we were given, however there are a few things that must be derived and generated in order to properly follow the flow of this model. Starting with the simplest of the dimension tables, the Academics table will pull most of its data simply from the dataset with the exception being the GPA Quality. GPA Quality will be generated from the GPA itself and always be stored as 'high', 'medium', or 'low' to describe what GPA the student has. The time dimension table and program dimension table are however a bit more complicated, as unlike the Academic dimension table where we are already given a student's academic ID we will need to generate a SemesterID and CollegeID as the primary keys to describe the various combinations of the data within their tables. CollegeID will take the major and degree and combine them into an ID that is unique on a major and degree combination, and SemesterID shall do the same but with a combination of what semester and year combinations. The last part of data generation is in the Program dimension table, where we will derive what college a student is in by looking at their major according to the hierarchy that we defined earlier. With these values being generated, and the rest being pulled from our source data, we will finally be able to create the model in MySQL and begin loading the data into our model.

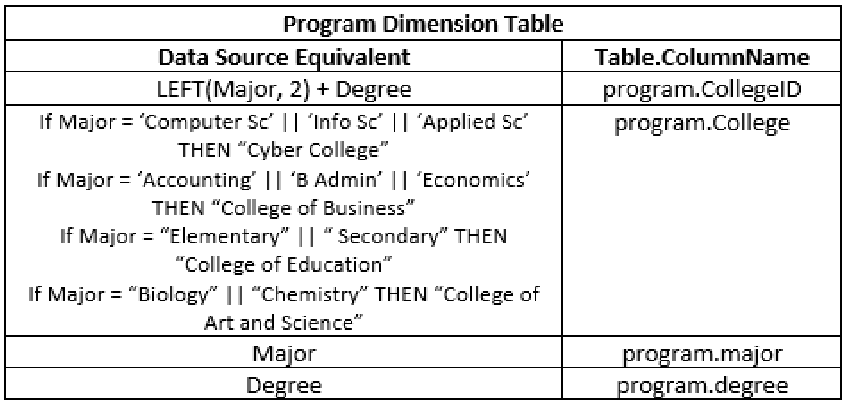


Academics Dimension Table	
Data Source Equivalent	**Table.ColumnName**
ID	academics.AcademicID
GPA	academics.GPA
Derived as GPA Quality: If GPA >= 3.0 THEN "high" If GPA > 2.0 AND GPA < 3.0 THEN "medium" ELSE THEN "low"	academics.gpaQual
Status	academics.Status
Name	academics.Name
Address	academics.Address



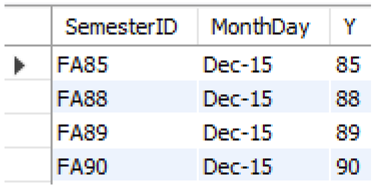
Time Dimension Table	
Data Source Equivalent	**Table.ColumnName**
LEFT(MonthDay, 0): IF = 'Jul' ||'Jun' || 'A' THEN "S1"..."S4" IF = 'D' THEN "Fall" If = 'M' THEN "Spring" +Y	Time.SemesterID
MonthDay	Time.MonthDay
Y	Time.Y

Figure 8: Dimension Tables

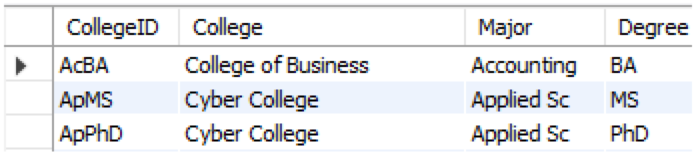


Program Dimension Table	
Data Source Equivalent	**Table.ColumnName**
LEFT(Major, 2) + Degree	program.CollegeID
If Major = 'Computer Sc' || 'Info Sc' || 'Applied Sc' THEN "Cyber College" If Major = 'Accounting' || 'B Admin' || 'Economics' THEN "College of Business" If Major = "Elementary" || " Secondary" THEN "College of Education" If Major = "Biology" || "Chemistry" THEN "College of Art and Science"	program.College
Major	program.major
Degree	program.degree

There is no denying between creating the model and loading the data that loading the data is a bit more advanced. With proper planning and connection of data types in order to create the primary key/foreign key connections between the three dimensional implementing the model is not too much of an issue, and as a result this model was able to be successfully implemented into MySQL such that we are able to recreate our star schema with our model that also has generational logic for the predescribed table attributes. With that being said, loading can finally begin to occur. There are various ETL tools to help with loading data quickly, however we will load the data without these tools simply by following the previously defined table/data source equivalents. Doing so will result in being able to query each table dimension to ensure that data has properly been generated as such:

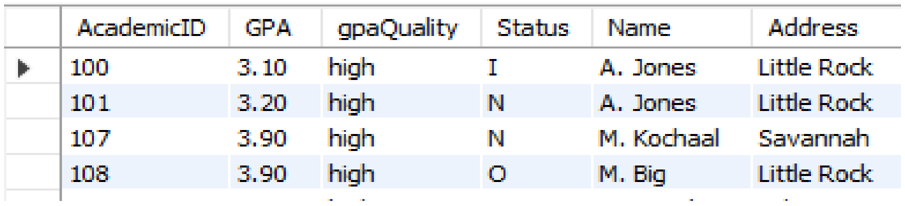


SemesterID	MonthDay	Y
FA85	Dec-15	85
FA88	Dec-15	88
FA89	Dec-15	89
FA90	Dec-15	90



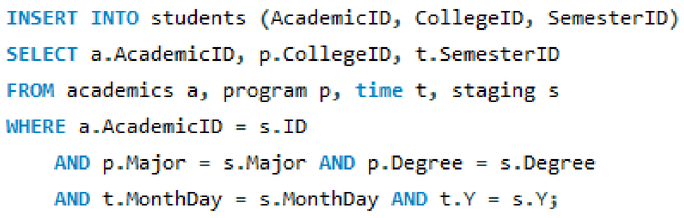
CollegeID	College	Major	Degree
AcBA	College of Business	Accounting	BA
ApMS	Cyber College	Applied Sc	MS
ApPhD	Cyber College	Applied Sc	PhD

Figure 9: Loaded Dimensions

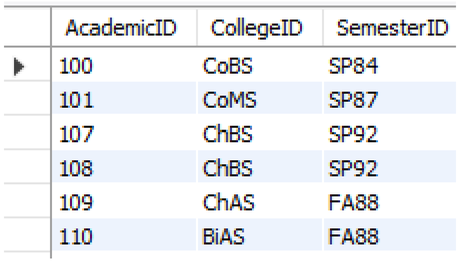


AcademicID	GPA	gpaQuality	Status	Name	Address
100	3.10	high	I	A. Jones	Little Rock
101	3.20	high	N	A. Jones	Little Rock
107	3.90	high	N	M. Kochaal	Savannah
108	3.90	high	O	M. Big	Little Rock

Populating the fact table is a bit more complicated than just importing the data unlike the dimensional tables, but still quite feasible. To begin, we must create a staging table that will take our raw data and be inserted one-to-one directly into it. With this staging table we can use the raw data in conjunction with our defined dimensional tables data to finally populate the fact table using a query to compare the student id (primary key for Academic table), the time (MonthDay and Y that create primary key SemesterID for the Time table), and major and degree information (which creates the primary key of CollegeID for the Program table) in a query (query used described below). After using the query we are able to now confirm that the correct data has been applied to the fact table.



```
INSERT INTO students (AcademicID, CollegeID, SemesterID)
SELECT a.AcademicID, p.CollegeID, t.SemesterID
FROM academics a, program p, time t, staging s
WHERE a.AcademicID = s.ID
    AND p.Major = s.Major AND p.Degree = s.Degree
    AND t.MonthDay = s.MonthDay AND t.Y = s.Y;
```



AcademicID	CollegeID	SemesterID
100	CoBS	SP84
101	CoMS	SP87
107	ChBS	SP92
108	ChBS	SP92
109	ChAS	FA88
110	BiAS	FA88

Figure 10: Loading the Fact Table

Interface Generation

Now that the Data Warehouse has properly been implemented, we need some sort of user-friendly interface to allow the privileged staff of USRH to properly use the Data Warehouse. The interface will be a basic template of the generated data cube made in Python such that a user can simply click what options they would like to generate reports. This template is, again, quite generic, but it allows for even the most basic user's to understand enough of what they are looking at to the point that they can easily query without any knowledge of programming languages as long as they understand the context of the data. To use this interface, first the user must choose their OLAP Operation on the Commands drop down box. The user will then fill the data on the columns that they would like to use for querying the data in conjunction with the OLAP Command. The options of selecting what to filter by can be determined by locating the properly labeled column that matches the criteria one would like to use, and then filling out the box below. For columns that have many of options, the user must fill them in by the proper names that are stored in the database, and if they want to filter by more

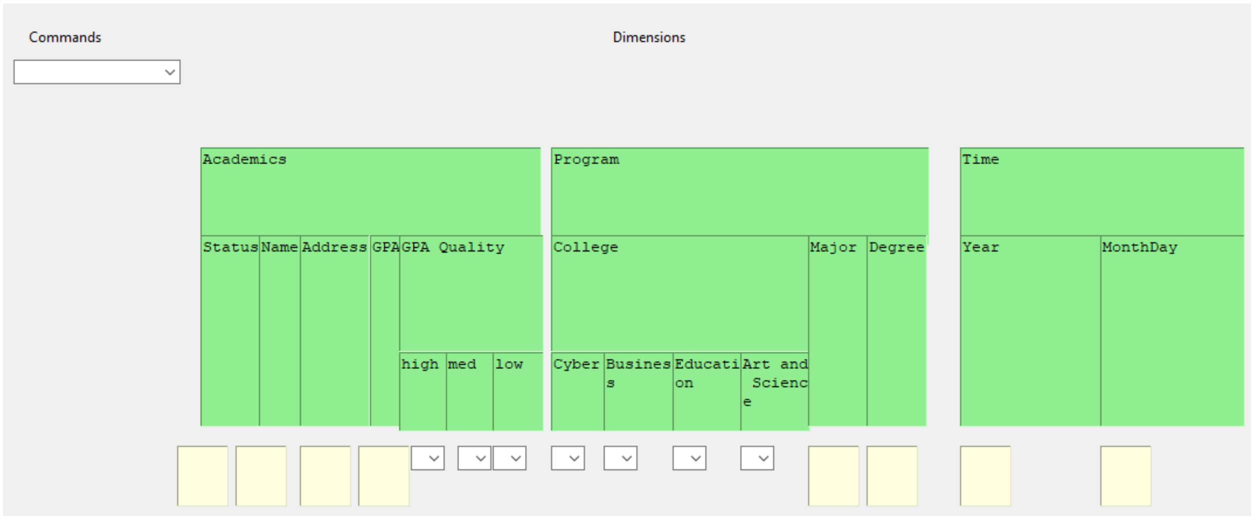
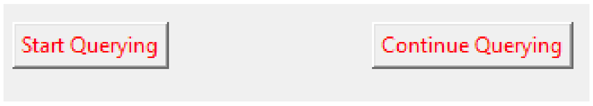


Figure 11: Interface

than one item in the column they must add an ‘ OR ‘ between the two items in the same box. Some of these boxes are also replaced with dropdown menus that give the selection of ‘1’ or ‘0’ as these represent the lowest answers that can be found in the hierarchy of that dimension. The user will select ‘1’ to filter by this option or a ‘0’ to not use it for querying; the user can also elect to not click on the box and leave it blank to also generate a query.

Once a user has successfully selected their required columns for a query they must simply press the ‘Start Querying’ button located at the bottom of the page to begin the creation and processing of the SQL query. This will be performed through the Mysql-connector-python driver, which creates the connection between the python interface and MySQL. With the connection being made the query results will be printed both in MySQL and also in Python through the tabulate package so that the information can be intuitively shown to the user. As for the querying commands themselves it is important for the user to already know the OLAP Operations and that rolling up/drilling down should be done first. The conversion of the OLAP operations to SQL statements is performed through a string of simple algorithms that will check what boxes have inputs in them and will fill and transform these inputs through various means with slightly altered functionalities depending on the OLAP Command that is user-assigned at the moment of pressing the button. For example, while performing the Roll-Up or Drill-Down operation one can fill the boxes with whatever data they want and it will only be read as an input in that box such that it will move to that hierarchy with no deeper understanding than there is an input in the box versus the Dice operation where the difference between selecting ‘1’ and a true input makes a difference as any inputs that are not ‘0’ or ‘1’ will be used as an input for what to query by.

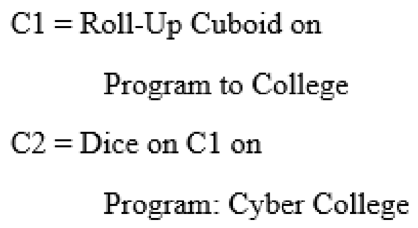
Figure 12: Interface Buttons



Along with the 'Start Querying' button you may also notice the 'Continue Querying' button nearby. These buttons function similarly, but the main difference here is that 'Continue Querying' functions in a way such that you are continuing to use the Cube generated from the previous query in conjunction with the current OLAP Operation. For best use of this interface, one should do a Roll-up or Drill-down using the 'Start Querying' button and then continue transforming the cube until desired results with the 'Continue Querying' button. Upon completion of a query there should be a table returned with all rows found that match the query as well as a count of how many entries that was total. In essence, the 'Start Querying' button only exists in the case of SQL statement creation error so that one may start a new Cube without having to close and reopen the application. To demonstrate, let us perform some testing.

Interface Testing

Let's take a look at the four provided test cases that were provided by the USRH starting with the first case where we must *get the Number of Graduate Students for the Cyber College.* First, we shall compute this task into a sequence of OLAP Operations and fortunately this task is not too complicated. In fact, it can be completed in as small as 2 OLAP operations of Rolling and Dicing on our Program dimension for where the students are in the Cyber College as demonstrated.



C1 = Roll-Up Cuboid on

Program to College

C2 = Dice on C1 on

Program: Cyber College

With these Operations decided, we then simply convert the operations into steps on the interface. We first do our Rollup operation to declare the depth of the hierarchy to be on the College level (as having any of the 4 'College' options during roll-up/drill-down will select that level in the hierarchy instead of querying) through the 'Start Query' button, and then change the Command to Dice with the same input in the Cyber College box of '1'. With both steps complete we can then go into the console and view the locally generated results, which includes the SQL statement generated to create this table and the table itself. At the bottom you can also see that it will generate the amount of rows that are located within this table, with our roll-up obtaining all of the entries, and the dice operation trimming it down to the 36 students that are within the Cyber College.

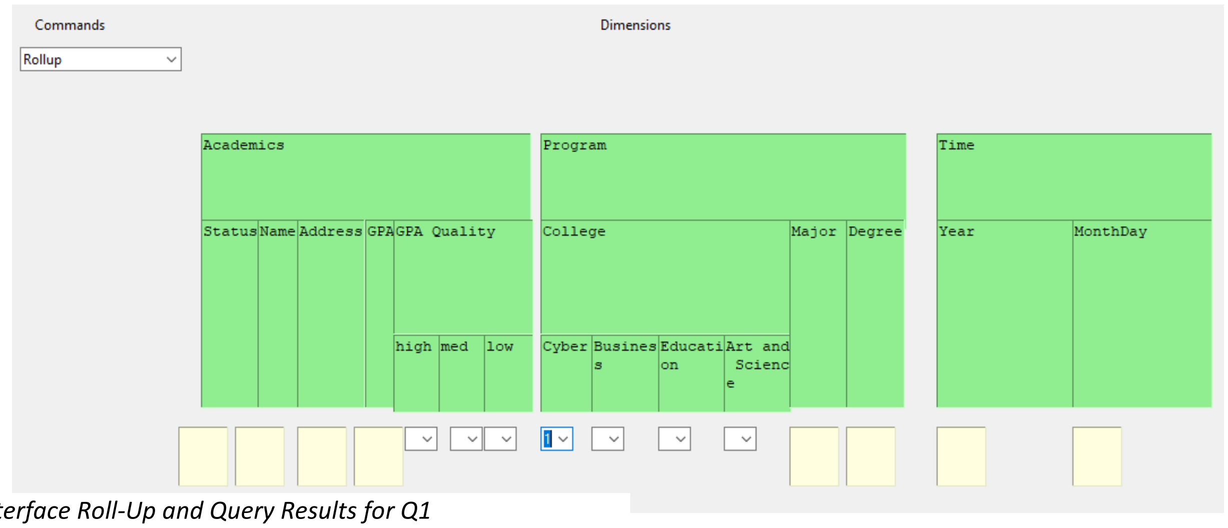
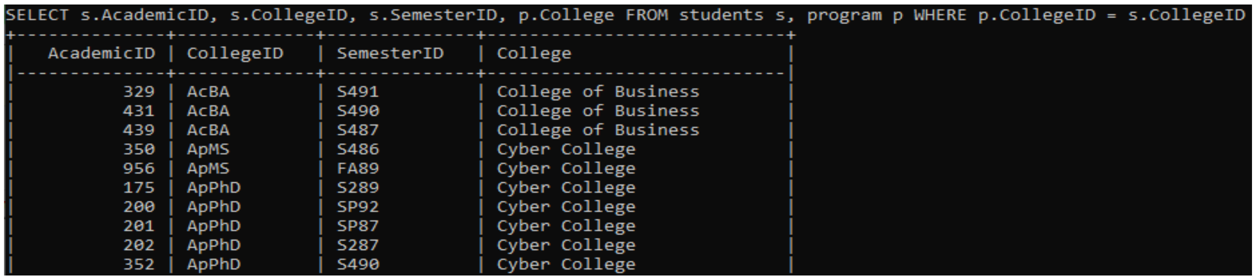


Figure 13: Interface Roll-Up and Query Results for Q1



```
SELECT s.AcademicID, s.CollegeID, s.SemesterID, p.College FROM students s, program p WHERE p.CollegeID = s.CollegeID
+-------------+-------------+-------------+----------------------------+
|   AcademicID | CollegeID   | SemesterID  | College                    |
|-------------+-------------+-------------+----------------------------|
|          329 | AcBA        | S491        | College of Business        |
|          431 | AcBA        | S490        | College of Business        |
|          439 | AcBA        | S487        | College of Business        |
|          350 | ApMS        | S486        | Cyber College              |
|          956 | ApMS        | FA89        | Cyber College              |
|          175 | ApPhD       | S289        | Cyber College              |
|          200 | ApPhD       | SP92        | Cyber College              |
|          201 | ApPhD       | SP87        | Cyber College              |
|          202 | ApPhD       | S287        | Cyber College              |
|          352 | ApPhD       | S490        | Cyber College              |
```

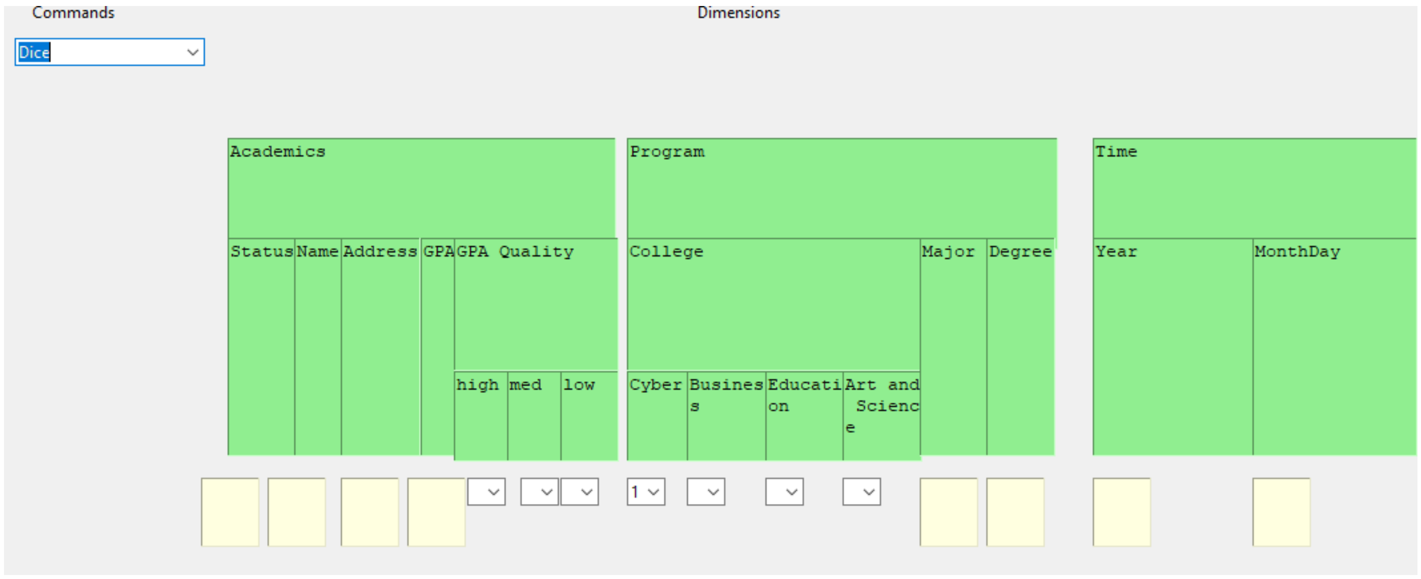
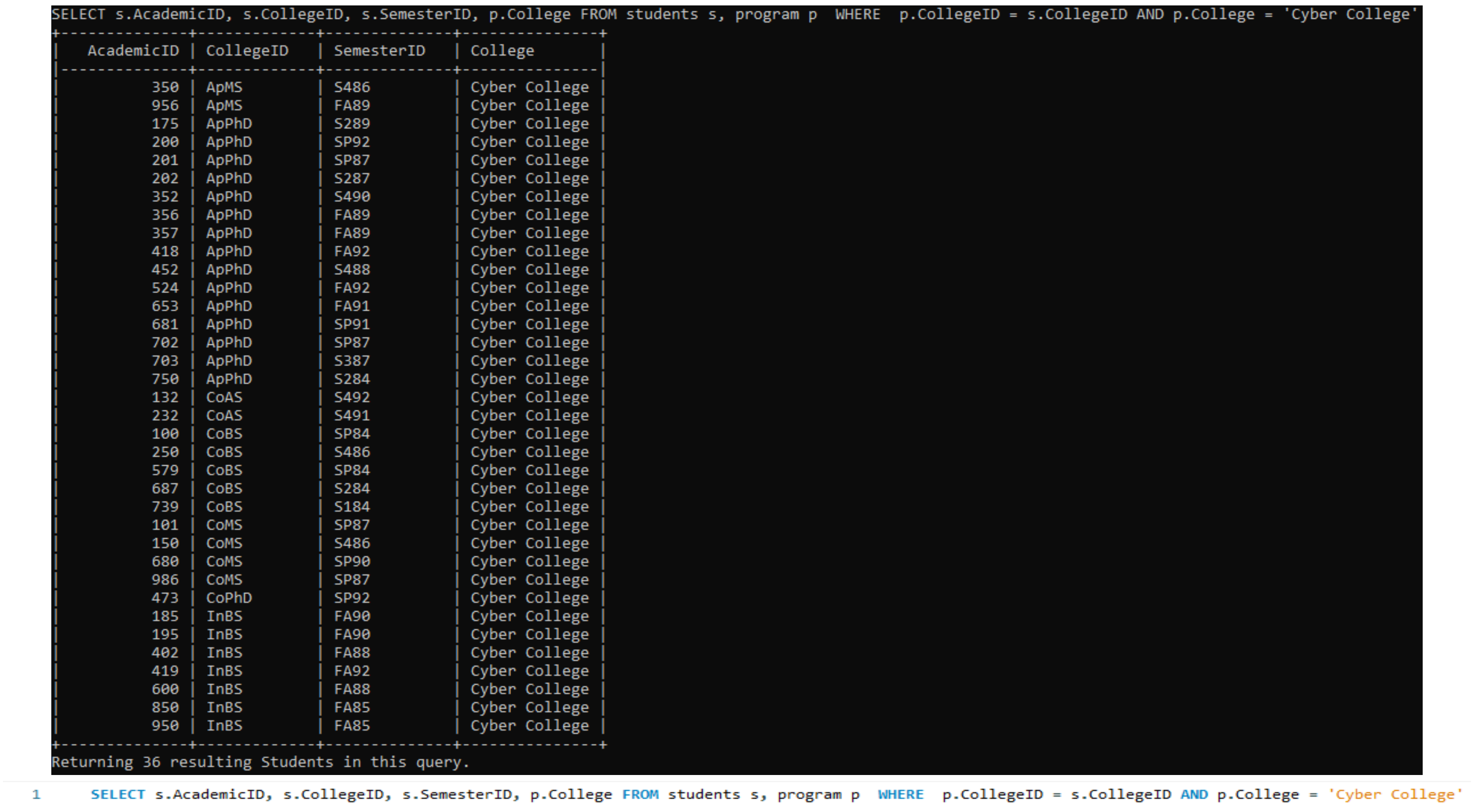


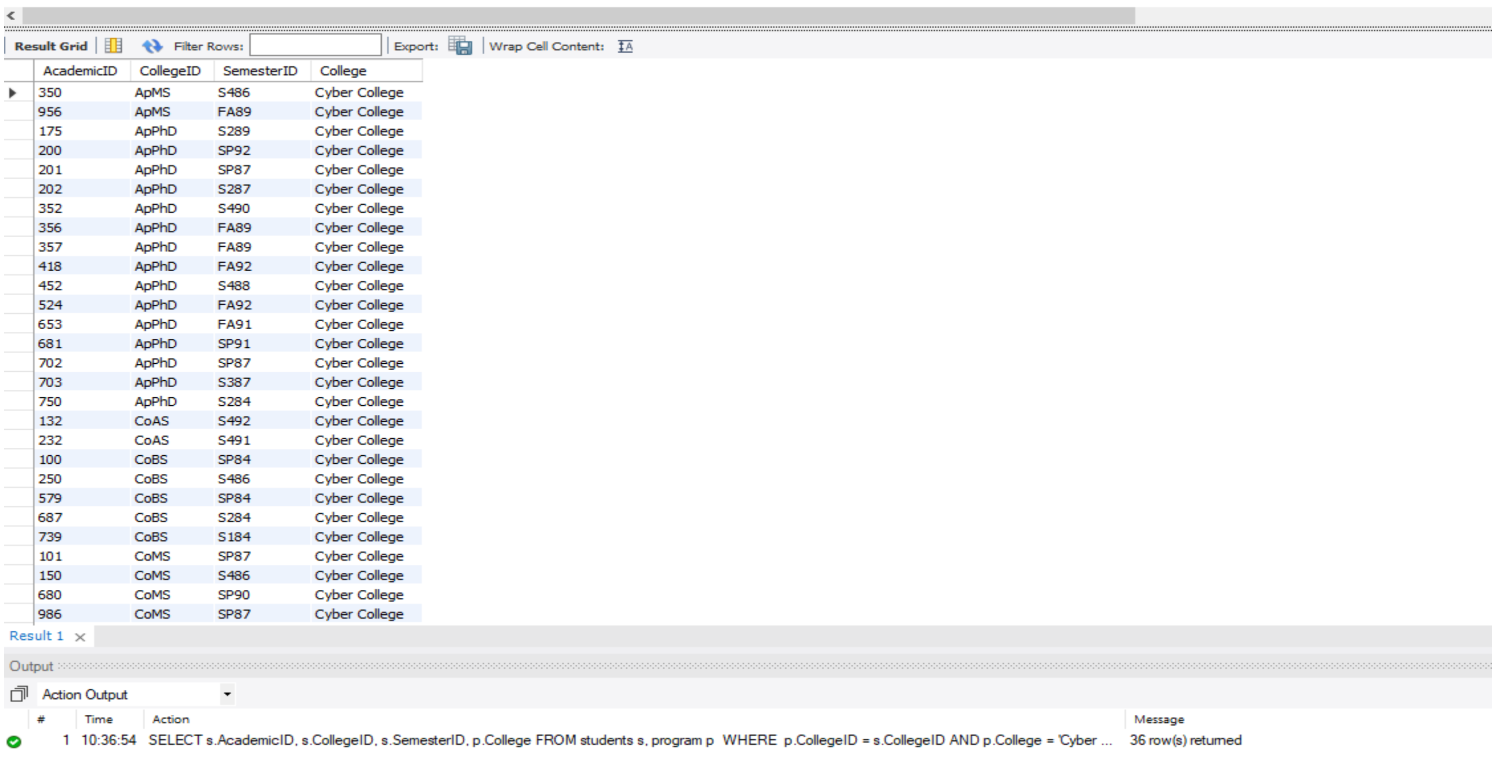
Figure 14: Interface Dice for Q1



```
SELECT s.AcademicID, s.CollegeID, s.SemesterID, p.College FROM students s, program p  WHERE  p.CollegeID = s.CollegeID AND p.College = 'Cyber College'
+-------------+-------------+--------------+--------------+
|  AcademicID | CollegeID   | SemesterID   | College      |
|-------------+-------------+--------------+--------------|
|         350 | ApMS        | S486         | Cyber College |
|         956 | ApMS        | FA89         | Cyber College |
|         175 | ApPhD       | S289         | Cyber College |
|         200 | ApPhD       | SP92         | Cyber College |
|         201 | ApPhD       | SP87         | Cyber College |
|         202 | ApPhD       | S287         | Cyber College |
|         352 | ApPhD       | S490         | Cyber College |
|         356 | ApPhD       | FA89         | Cyber College |
|         357 | ApPhD       | FA89         | Cyber College |
|         418 | ApPhD       | FA92         | Cyber College |
|         452 | ApPhD       | S488         | Cyber College |
|         524 | ApPhD       | FA92         | Cyber College |
|         653 | ApPhD       | FA91         | Cyber College |
|         681 | ApPhD       | SP91         | Cyber College |
|         702 | ApPhD       | SP87         | Cyber College |
|         703 | ApPhD       | S387         | Cyber College |
|         750 | ApPhD       | S284         | Cyber College |
|         132 | CoAS        | S492         | Cyber College |
|         232 | CoAS        | S491         | Cyber College |
|         100 | CoBS        | SP84         | Cyber College |
|         250 | CoBS        | S486         | Cyber College |
|         579 | CoBS        | SP84         | Cyber College |
|         687 | CoBS        | S284         | Cyber College |
|         739 | CoBS        | S184         | Cyber College |
|         101 | CoMS        | SP87         | Cyber College |
|         150 | CoMS        | S486         | Cyber College |
|         680 | CoMS        | SP90         | Cyber College |
|         986 | CoMS        | SP87         | Cyber College |
|         473 | CoPhD       | SP92         | Cyber College |
|         185 | InBS        | FA90         | Cyber College |
|         195 | InBS        | FA90         | Cyber College |
|         402 | InBS        | FA88         | Cyber College |
|         419 | InBS        | FA92         | Cyber College |
|         600 | InBS        | FA88         | Cyber College |
|         850 | InBS        | FA85         | Cyber College |
|         950 | InBS        | FA85         | Cyber College |
+-------------+-------------+--------------+--------------+
Returning 36 resulting Students in this query.
```

```
SELECT s.AcademicID, s.CollegeID, s.SemesterID, p.College FROM students s, program p  WHERE  p.CollegeID = s.CollegeID AND p.College = 'Cyber College'
```

Figure 15: Q1 Results in Python and MySQL



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

AcademicID	CollegeID	SemesterID	College
350	ApMS	S486	Cyber College
956	ApMS	FA89	Cyber College
175	ApPhD	S289	Cyber College
200	ApPhD	SP92	Cyber College
201	ApPhD	SP87	Cyber College
202	ApPhD	S287	Cyber College
352	ApPhD	S490	Cyber College
356	ApPhD	FA89	Cyber College
357	ApPhD	FA89	Cyber College
418	ApPhD	FA92	Cyber College
452	ApPhD	S488	Cyber College
524	ApPhD	FA92	Cyber College
653	ApPhD	FA91	Cyber College
681	ApPhD	SP91	Cyber College
702	ApPhD	SP87	Cyber College
703	ApPhD	S387	Cyber College
750	ApPhD	S284	Cyber College
132	CoAS	S492	Cyber College
232	CoAS	S491	Cyber College
100	CoBS	SP84	Cyber College
250	CoBS	S486	Cyber College
579	CoBS	SP84	Cyber College
687	CoBS	S284	Cyber College
739	CoBS	S184	Cyber College
101	CoMS	SP87	Cyber College
150	CoMS	S486	Cyber College
680	CoMS	SP90	Cyber College
986	CoMS	SP87	Cyber College

Result 1

Output

Action Output

#	Time	Action	Message
1	10:36:54	SELECT s.AcademicID, s.CollegeID, s.SemesterID, p.College FROM students s, program p WHERE p.CollegeID = s.CollegeID AND p.College = 'Cyber ...	36 row(s) returned

The results of this query can then be compared to the results of running the generated query on MySQL to show that the results are indeed the same with the order in which rows are displayed is identical and the amount of rows are also identical. This problem is quite simple in the grand scheme of requirements needed by the USRH so let's try a more difficult test case. The USRH has supplied us with a test case to *Obtain the Number of Graduates with a BS in Computer Science in the Summer of 1988,* however as there are no graduates with a BS in Computer Science during this time we shall change this to being placed during the Summer of 1984. With this said, we must now follow the same formula of converting the problem into OLAP operations, inserting the equivalents into the interface, and obtaining results given by the generated SQL query.

In order to solve this problem we must first start higher up in the hierarchy and work our way down, so we must start with the Major and Year and Drill-down to the Degree and Semester. When converted into the interface we will get 4 separate output queries with each query getting us closer from starting with 99 rows, to 4 rows, moving to the lower hierarchy, and finally return the 2 graduated students that obtained a BS in Computer Science in the Summer of 1984. The output is then shown being identitical in both Python and MySQL.

C1 = Roll-Up Cuboid on

Program to Major

Time to Y

C2 = Dice on C1

Program: Computer Sc

Time: 84

C3 = Drill-Down C2

Program to Degree

Time to MonthDay

C4 = Dice on C3

Program: BS

Time: Jun-1 OR Jul-4 OR Jul-15 OR Aug-15

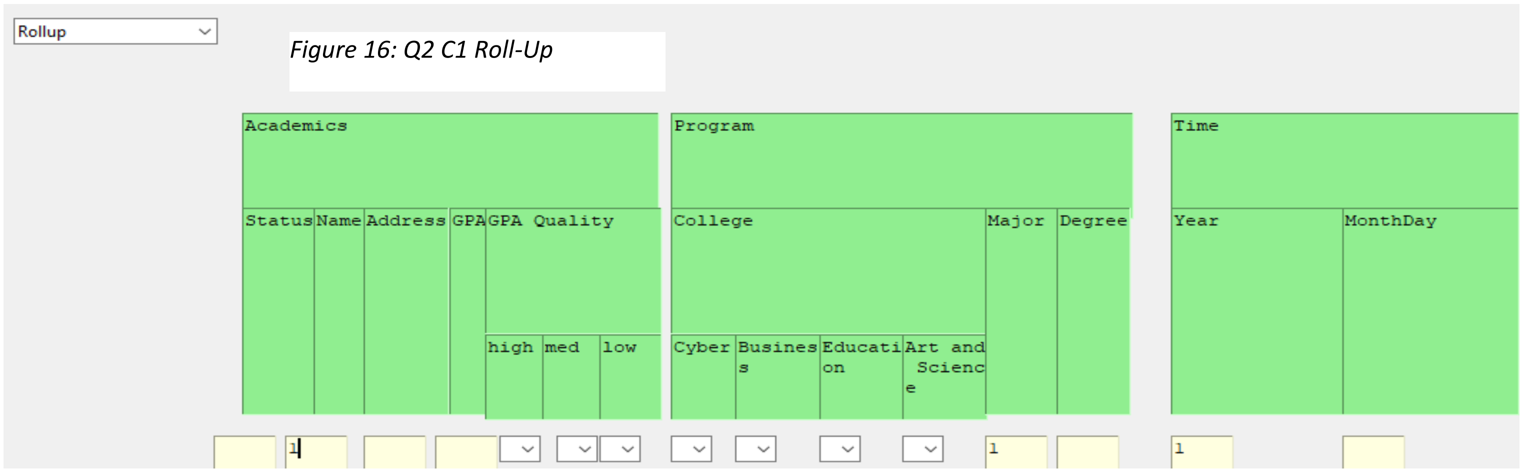
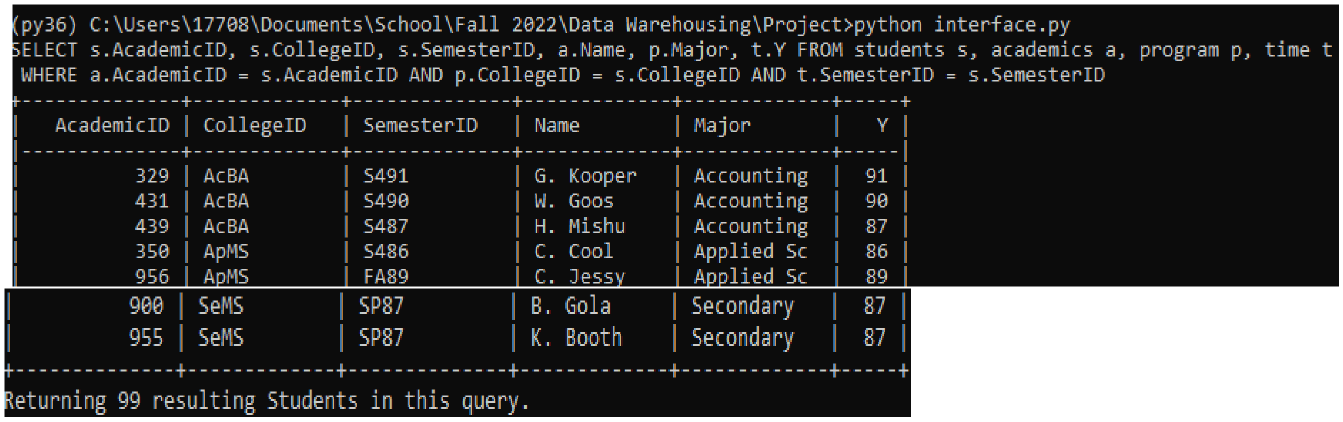
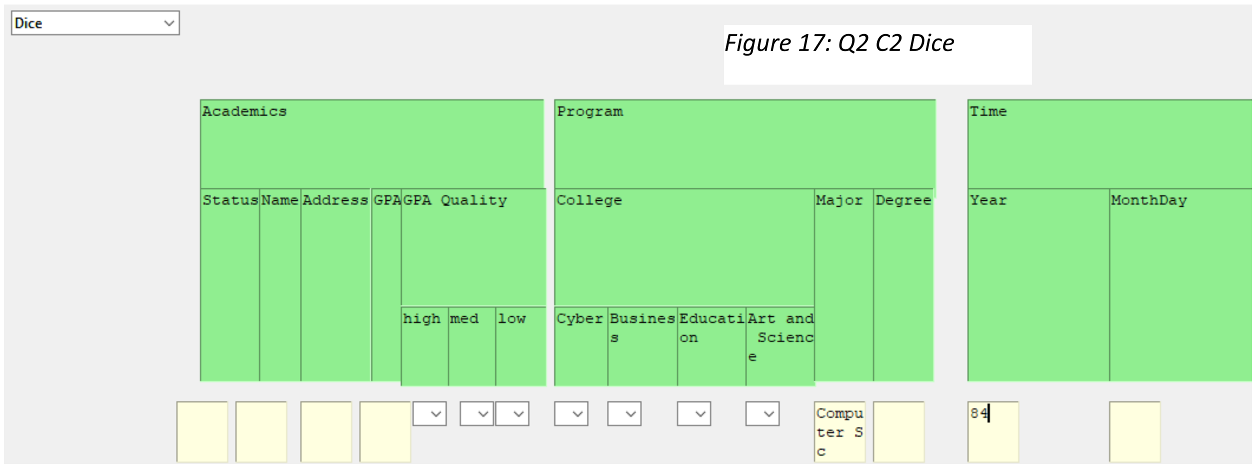


Figure 16: Q2 C1 Roll-Up

Rollup



```
(py36) C:\Users\17708\Documents\School\Fall 2022\Data Warehousing\Project>python interface.py
SELECT s.AcademicID, s.CollegeID, s.SemesterID, a.Name, p.Major, t.Y FROM students s, academics a, program p, time t
 WHERE a.AcademicID = s.AcademicID AND p.CollegeID = s.CollegeID AND t.SemesterID = s.SemesterID
+-------------+------------+-------------+-------------+-------------+-----+
|  AcademicID | CollegeID  | SemesterID  | Name        | Major       |  Y  |
|-------------+------------+-------------+-------------+-------------+-----|
|         329 | AcBA       | S491        | G. Kooper   | Accounting  |  91 |
|         431 | AcBA       | S490        | W. Goos     | Accounting  |  90 |
|         439 | AcBA       | S487        | H. Mishu    | Accounting  |  87 |
|         350 | ApMS       | S486        | C. Cool     | Applied Sc  |  86 |
|         956 | ApMS       | FA89        | C. Jessy    | Applied Sc  |  89 |
|         900 | SeMS       | SP87        | B. Gola     | Secondary   |  87 |
|         955 | SeMS       | SP87        | K. Booth    | Secondary   |  87 |
+-------------+------------+-------------+-------------+-------------+-----+
Returning 99 resulting Students in this query.
```

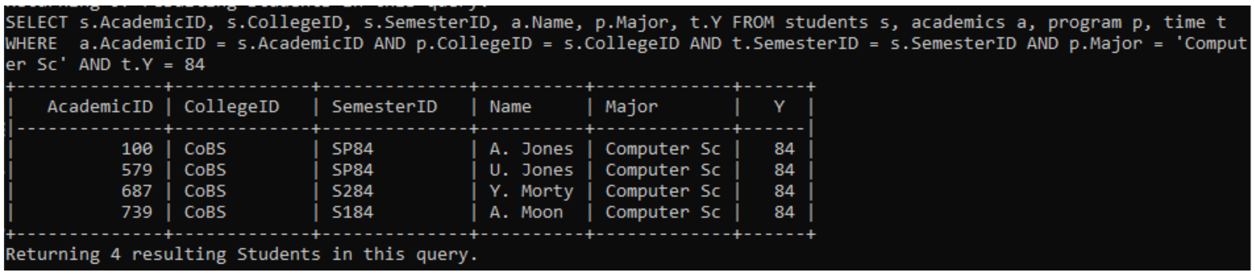


Dice

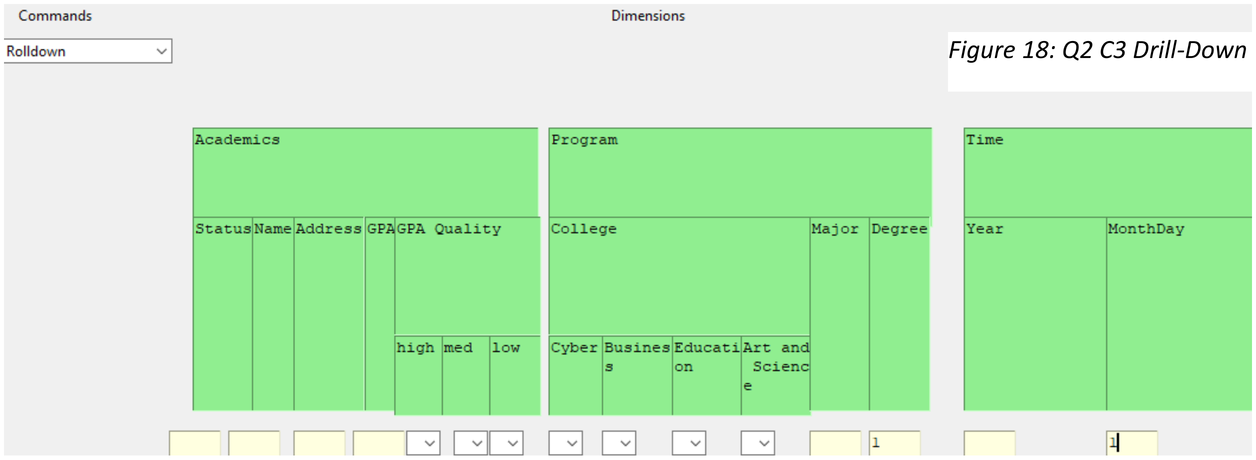
Major: Computer Sc

Year: 84

Figure 17: Q2 C2 Dice



```
SELECT s.AcademicID, s.CollegeID, s.SemesterID, a.Name, p.Major, t.Y FROM students s, academics a, program p, time t
WHERE  a.AcademicID = s.AcademicID AND p.CollegeID = s.CollegeID AND t.SemesterID = s.SemesterID AND p.Major = 'Comput
er Sc' AND t.Y = 84
+-------------+------------+-------------+----------+-------------+------+
|  AcademicID | CollegeID  | SemesterID  | Name     | Major       |   Y  |
|-------------+------------+-------------+----------+-------------+------|
|         100 | CoBS       | SP84        | A. Jones | Computer Sc |   84 |
|         579 | CoBS       | SP84        | U. Jones | Computer Sc |   84 |
|         687 | CoBS       | S284        | Y. Morty | Computer Sc |   84 |
|         739 | CoBS       | S184        | A. Moon  | Computer Sc |   84 |
+-------------+------------+-------------+----------+-------------+------+
Returning 4 resulting Students in this query.
```

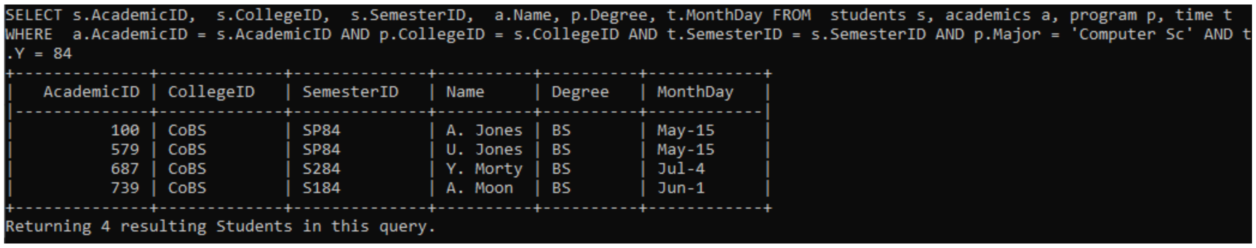


Rolldown

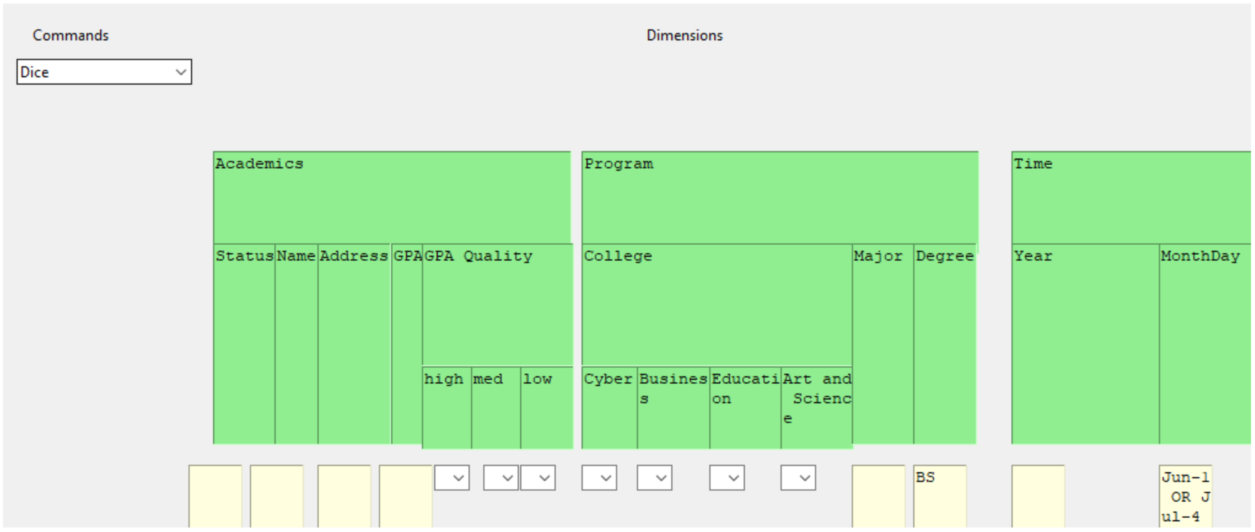
Degree: 1

MonthDay: 1

Figure 18: Q2 C3 Drill-Down

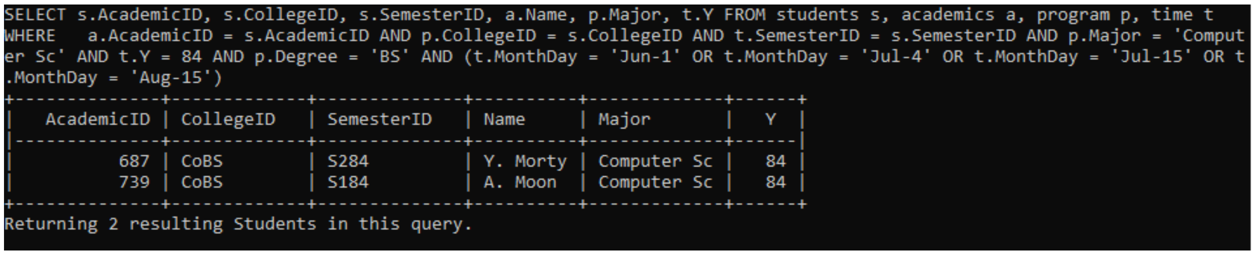


```
SELECT s.AcademicID,  s.CollegeID,  s.SemesterID,  a.Name, p.Degree, t.MonthDay FROM  students s, academics a, program p, time t
WHERE  a.AcademicID = s.AcademicID AND p.CollegeID = s.CollegeID AND t.SemesterID = s.SemesterID AND p.Major = 'Computer Sc' AND t
.Y = 84
+-------------+------------+--------------+----------+----------+------------+
|  AcademicID | CollegeID  | SemesterID   | Name     | Degree   | MonthDay   |
|-------------+------------+--------------+----------+----------+------------|
|         100 | CoBS       | SP84         | A. Jones | BS       | May-15     |
|         579 | CoBS       | SP84         | U. Jones | BS       | May-15     |
|         687 | CoBS       | S284         | Y. Morty | BS       | Jul-4      |
|         739 | CoBS       | S184         | A. Moon  | BS       | Jun-1      |
+-------------+------------+--------------+----------+----------+------------+
Returning 4 resulting Students in this query.
```

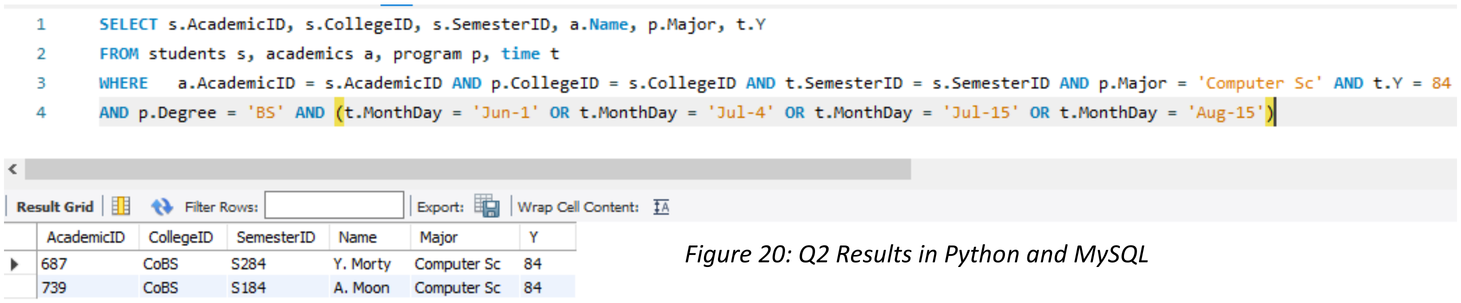


Degree: BS
MonthDay: Jun-1 OR Jul-4

Figure 19: Q2 C4 Dice



```
SELECT s.AcademicID, s.CollegeID, s.SemesterID, a.Name, p.Major, t.Y FROM students s, academics a, program p, time t
WHERE   a.AcademicID = s.AcademicID AND p.CollegeID = s.CollegeID AND t.SemesterID = s.SemesterID AND p.Major = 'Comput
er Sc' AND t.Y = 84 AND p.Degree = 'BS' AND (t.MonthDay = 'Jun-1' OR t.MonthDay = 'Jul-4' OR t.MonthDay = 'Jul-15' OR t
.MonthDay = 'Aug-15')
+-------------+--------------+---------------+-----------+--------------+------+
|  AcademicID | CollegeID    | SemesterID    | Name      | Major        |   Y  |
|-------------+--------------+---------------+-----------+--------------+------|
|         687 | CoBS         | S284          | Y. Morty  | Computer Sc  |   84 |
|         739 | CoBS         | S184          | A. Moon   | Computer Sc  |   84 |
+-------------+--------------+---------------+-----------+--------------+------+
Returning 2 resulting Students in this query.
```



```sql
SELECT s.AcademicID, s.CollegeID, s.SemesterID, a.Name, p.Major, t.Y
FROM students s, academics a, program p, time t
WHERE   a.AcademicID = s.AcademicID AND p.CollegeID = s.CollegeID AND t.SemesterID = s.SemesterID AND p.Major = 'Computer Sc' AND t.Y = 84
AND p.Degree = 'BS' AND (t.MonthDay = 'Jun-1' OR t.MonthDay = 'Jul-4' OR t.MonthDay = 'Jul-15' OR t.MonthDay = 'Aug-15')
```

AcademicID	CollegeID	SemesterID	Name	Major	Y
687	CoBS	S284	Y. Morty	Computer Sc	84
739	CoBS	S184	A. Moon	Computer Sc	84

Figure 20: Q2 Results in Python and MySQL

The next test query is to *obtain the Graduates with a High GPA in the College of Business*, which once again is a simple query that can be completed in few OLAP operations.

C1 = Roll-up Cuboid on

Academics to gpaQuality

Program to College

C2 = Dice on C1

Academics: high

Program: College of Business

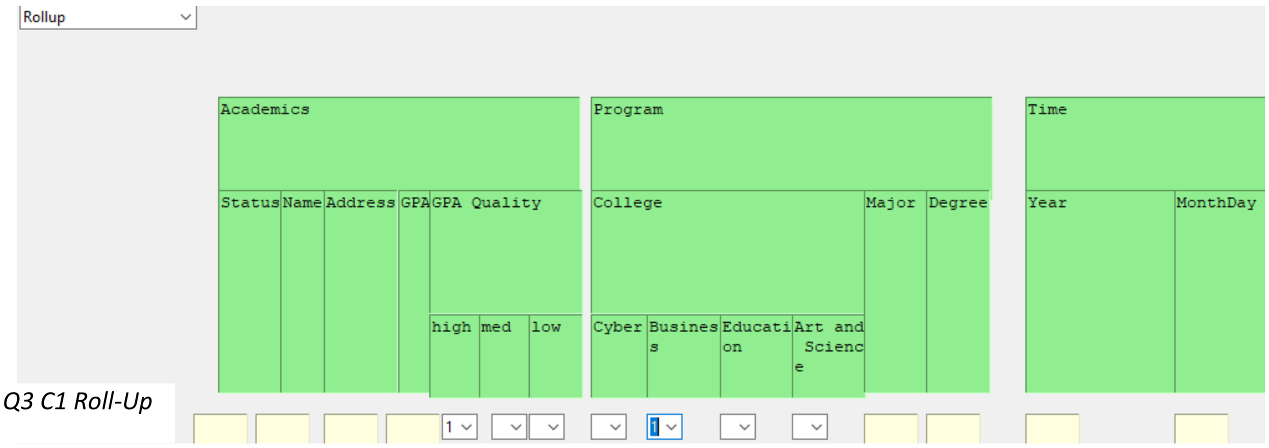
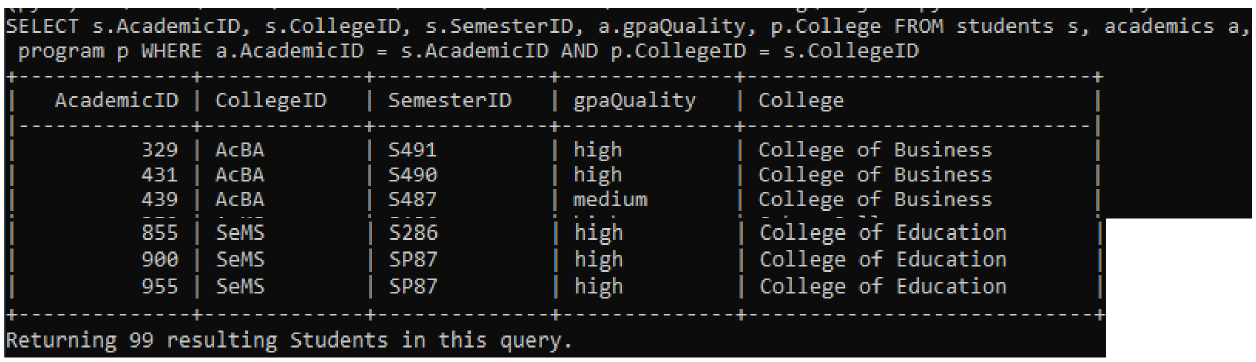


Figure 21: Q3 C1 Roll-Up



```
SELECT s.AcademicID, s.CollegeID, s.SemesterID, a.gpaQuality, p.College FROM students s, academics a,
 program p WHERE a.AcademicID = s.AcademicID AND p.CollegeID = s.CollegeID
+-------------+-------------+--------------+-------------+--------------------------+
|  AcademicID | CollegeID   | SemesterID   | gpaQuality  | College                  |
|-------------+-------------+--------------+-------------+--------------------------|
|         329 | AcBA        | S491         | high        | College of Business      |
|         431 | AcBA        | S490         | high        | College of Business      |
|         439 | AcBA        | S487         | medium      | College of Business      |
|         855 | SeMS        | S286         | high        | College of Education     |
|         900 | SeMS        | SP87         | high        | College of Education     |
|         955 | SeMS        | SP87         | high        | College of Education     |
+-------------+-------------+--------------+-------------+--------------------------+
Returning 99 resulting Students in this query.
```

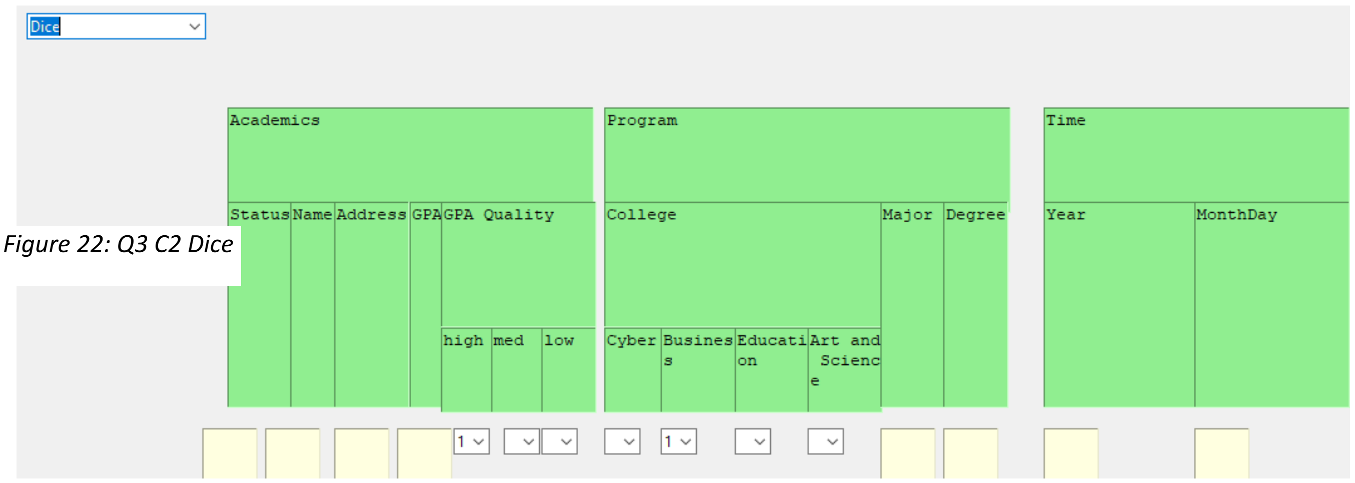
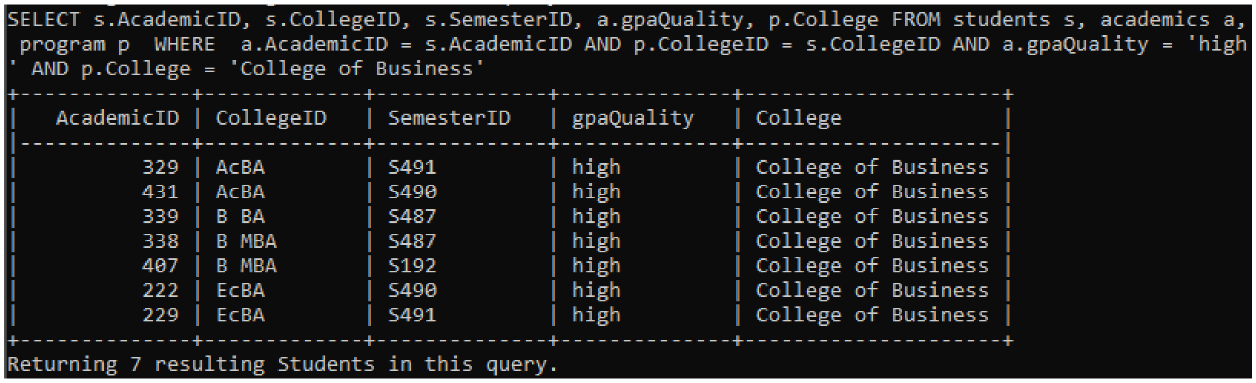
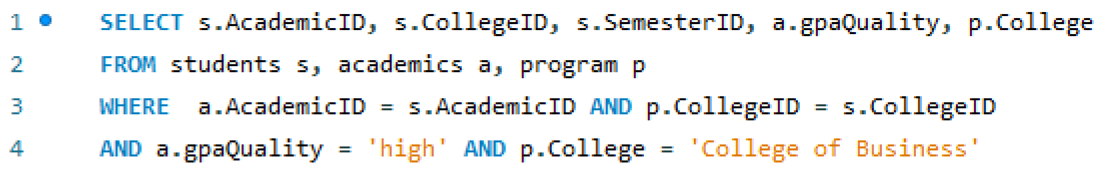


Figure 22: Q3 C2 Dice

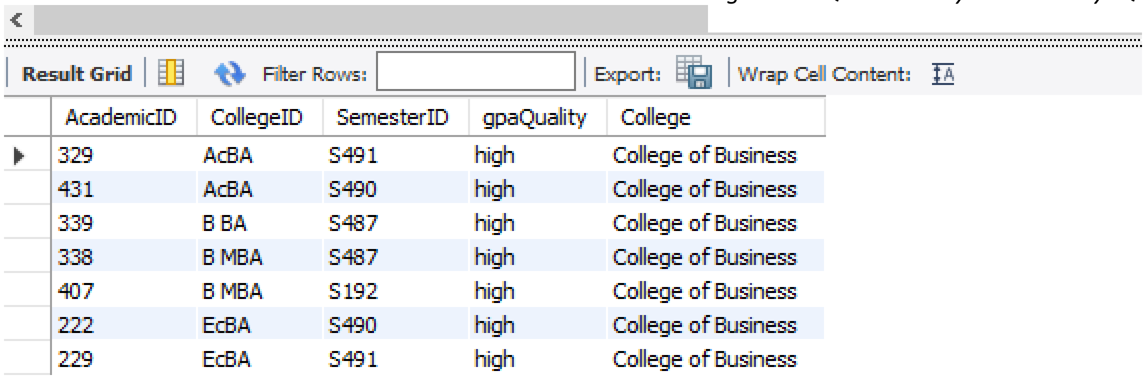


```
SELECT s.AcademicID, s.CollegeID, s.SemesterID, a.gpaQuality, p.College FROM students s, academics a,
 program p  WHERE  a.AcademicID = s.AcademicID AND p.CollegeID = s.CollegeID AND a.gpaQuality = 'high
' AND p.College = 'College of Business'
+-------------+------------+-------------+-------------+--------------------+
|   AcademicID | CollegeID  | SemesterID  | gpaQuality  | College            |
|-------------+------------+-------------+-------------+--------------------|
|          329 | AcBA       | S491        | high        | College of Business |
|          431 | AcBA       | S490        | high        | College of Business |
|          339 | B BA       | S487        | high        | College of Business |
|          338 | B MBA      | S487        | high        | College of Business |
|          407 | B MBA      | S192        | high        | College of Business |
|          222 | EcBA       | S490        | high        | College of Business |
|          229 | EcBA       | S491        | high        | College of Business |
+-------------+------------+-------------+-------------+--------------------+
Returning 7 resulting Students in this query.
```



```
SELECT s.AcademicID, s.CollegeID, s.SemesterID, a.gpaQuality, p.College
FROM students s, academics a, program p
WHERE  a.AcademicID = s.AcademicID AND p.CollegeID = s.CollegeID
AND a.gpaQuality = 'high' AND p.College = 'College of Business'
```

Figure 23: Q3 Results Python and MySQL



AcademicID	CollegeID	SemesterID	gpaQuality	College
329	AcBA	S491	high	College of Business
431	AcBA	S490	high	College of Business
339	B BA	S487	high	College of Business
338	B MBA	S487	high	College of Business
407	B MBA	S192	high	College of Business
222	EcBA	S490	high	College of Business
229	EcBA	S491	high	College of Business

The final test case is to *get the Number of International students graduated in the year 89, 90, and 91*. To make this query a little more difficult we will also add the additional final step of wanting to view Graduates in the year 91 separately, meaning we will solve the initial problem and then add a Slice OLAP operation step on to the end to test the additional capabilities of the interface (which will be noted as C3). The MySQL equivalent shown will be only the solution to the above test case meaning it will be equivalent to C2.

C1 = Roll-Up Cuboid on

Academics to Status

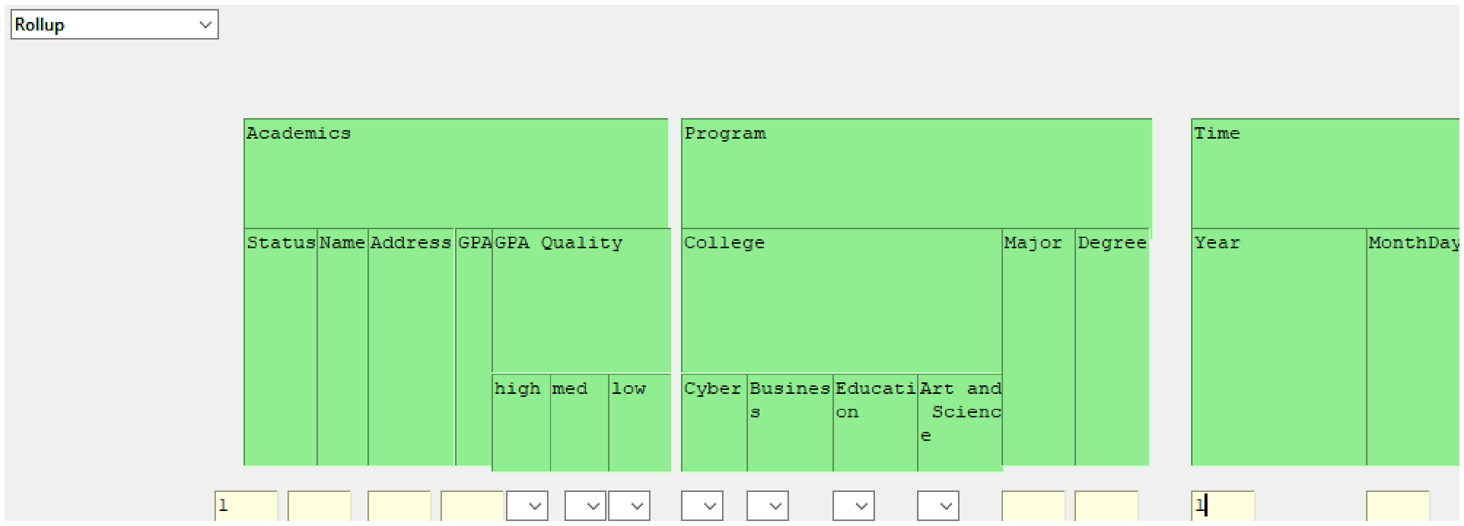
Time to Y

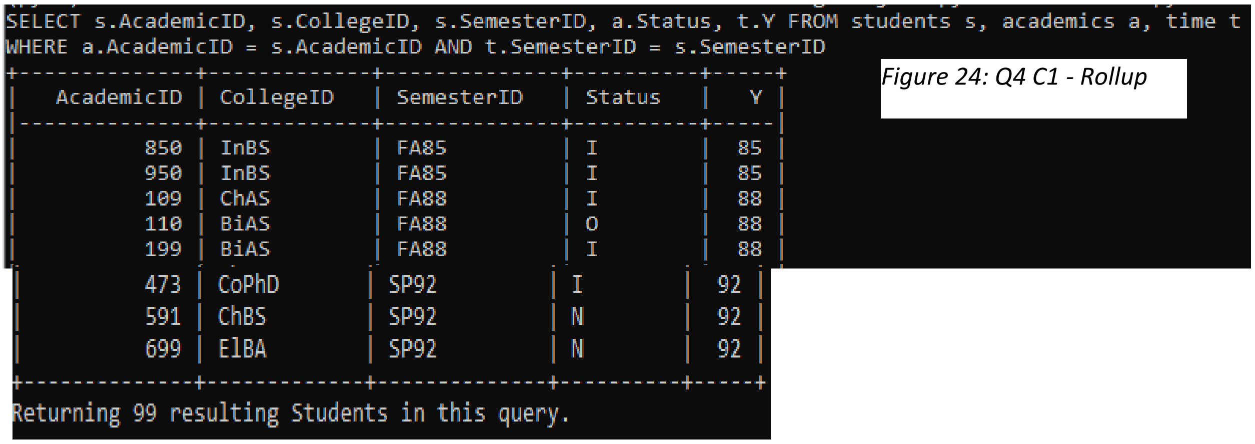
C2 = Dice C1 on

Academics: I, Time: 89 or 90 or 91

C3 = Slice C2 on

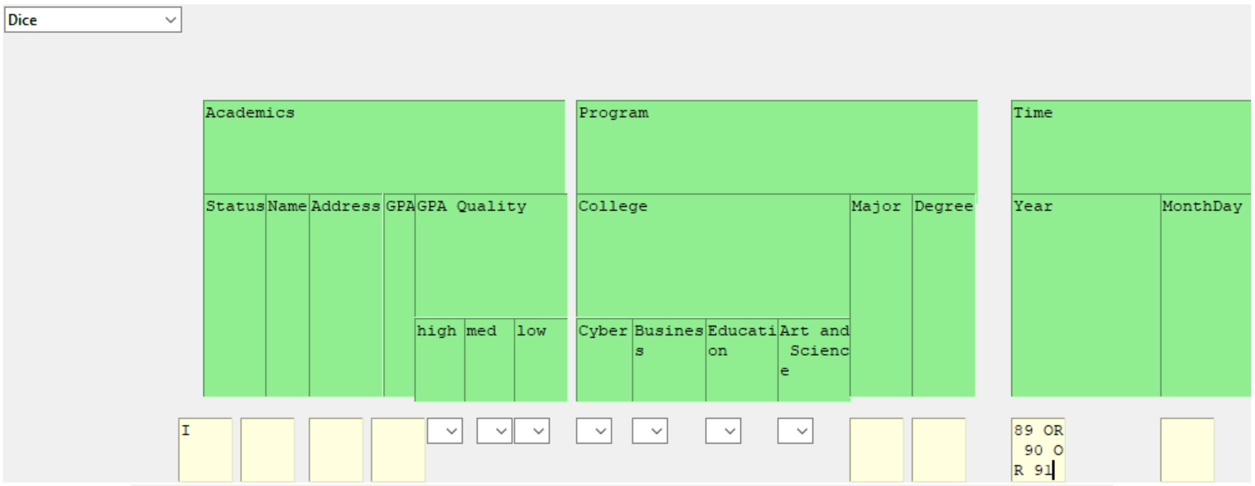
Time: 91





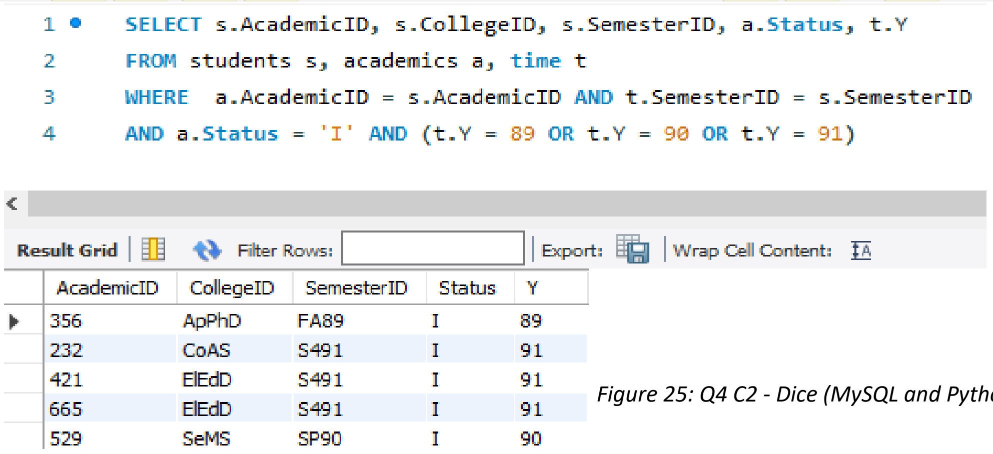
```
SELECT s.AcademicID, s.CollegeID, s.SemesterID, a.Status, t.Y FROM students s, academics a, time t
WHERE a.AcademicID = s.AcademicID AND t.SemesterID = s.SemesterID
+-------------+------------+-------------+----------+-----+
|   AcademicID | CollegeID  | SemesterID  | Status   |   Y |
|-------------+------------+-------------+----------+-----|
|          850 | InBS       | FA85        | I        |  85 |
|          950 | InBS       | FA85        | I        |  85 |
|          109 | ChAS       | FA88        | I        |  88 |
|          110 | BiAS       | FA88        | O        |  88 |
|          199 | BiAS       | FA88        | I        |  88 |
|          473 | CoPhD      | SP92        | I        |  92 |
|          591 | ChBS       | SP92        | N        |  92 |
|          699 | ElBA       | SP92        | N        |  92 |
+-------------+------------+-------------+----------+-----+
Returning 99 resulting Students in this query.
```

Figure 24: Q4 C1 - Rollup



I

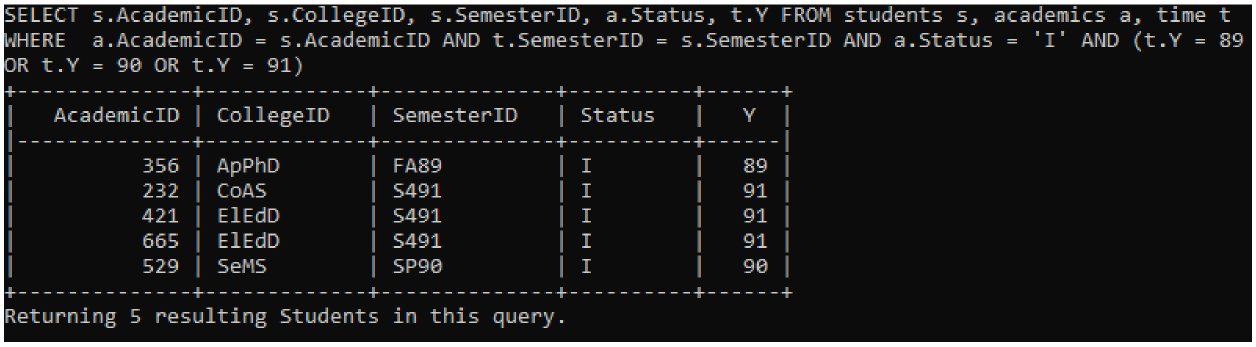
89 OR 90 OR 91



```sql
SELECT s.AcademicID, s.CollegeID, s.SemesterID, a.Status, t.Y
FROM students s, academics a, time t
WHERE  a.AcademicID = s.AcademicID AND t.SemesterID = s.SemesterID
AND a.Status = 'I' AND (t.Y = 89 OR t.Y = 90 OR t.Y = 91)
```

AcademicID	CollegeID	SemesterID	Status	Y
356	ApPhD	FA89	I	89
232	CoAS	S491	I	91
421	ElEdD	S491	I	91
665	ElEdD	S491	I	91
529	SeMS	SP90	I	90

Figure 25: Q4 C2 - Dice (MySQL and Python Results)



```
SELECT s.AcademicID, s.CollegeID, s.SemesterID, a.Status, t.Y FROM students s, academics a, time t
WHERE  a.AcademicID = s.AcademicID AND t.SemesterID = s.SemesterID AND a.Status = 'I' AND (t.Y = 89
OR t.Y = 90 OR t.Y = 91)
+-------------+-------------+--------------+----------+-----+
|  AcademicID | CollegeID   | SemesterID   | Status   |   Y |
|-------------+-------------+--------------+----------+-----|
|         356 | ApPhD       | FA89         | I        |  89 |
|         232 | CoAS        | S491         | I        |  91 |
|         421 | ElEdD       | S491         | I        |  91 |
|         665 | ElEdD       | S491         | I        |  91 |
|         529 | SeMS        | SP90         | I        |  90 |
+-------------+-------------+--------------+----------+-----+
Returning 5 resulting Students in this query.
```

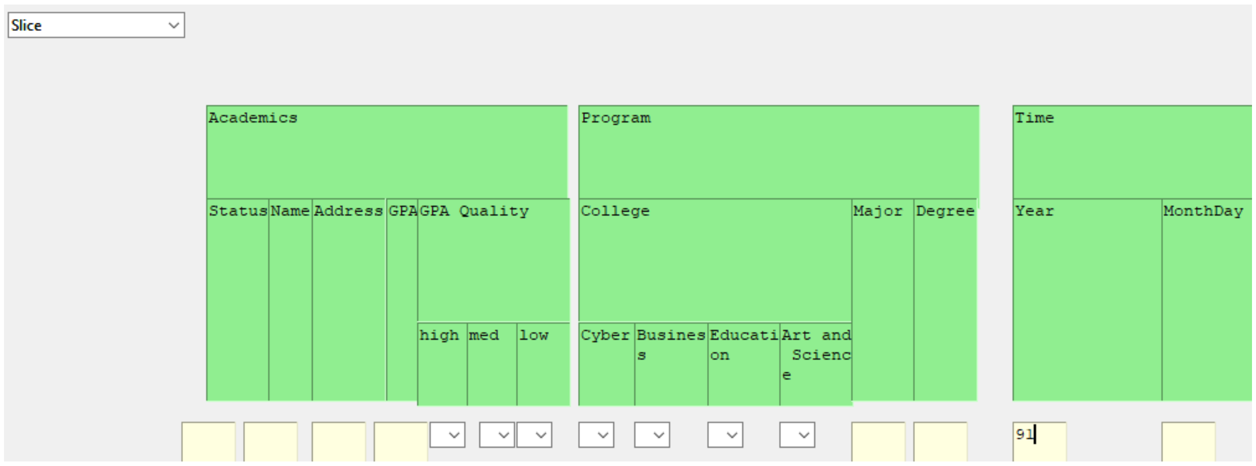
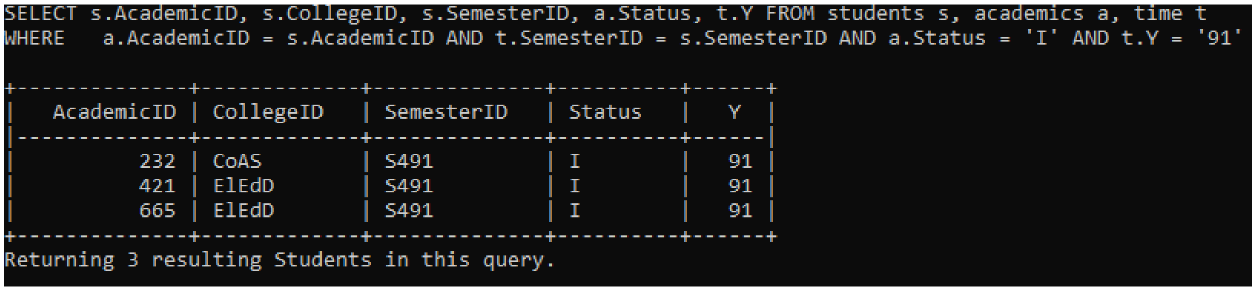


Figure 26: Q4 C3 Slice (Test of additional operation)



```
SELECT s.AcademicID, s.CollegeID, s.SemesterID, a.Status, t.Y FROM students s, academics a, time t
WHERE   a.AcademicID = s.AcademicID AND t.SemesterID = s.SemesterID AND a.Status = 'I' AND t.Y = '91'

+--------------+-------------+--------------+----------+------+
|   AcademicID | CollegeID   | SemesterID   | Status   |   Y  |
|--------------+-------------+--------------+----------+------|
|          232 | CoAS        | S491         | I        |   91 |
|          421 | ElEdD       | S491         | I        |   91 |
|          665 | ElEdD       | S491         | I        |   91 |
+--------------+-------------+--------------+----------+------+
Returning 3 resulting Students in this query.
```

Conclusion

In this document we were able to successfully show the process of creating a data warehouse to meet the specifications of its' use. The strengths of upgrading to a data warehouse are abundantly clear when comparing the systems created here versus implementing the raw data into a single table. Creating a schema of tables compared to a single table of entries allows for much quicker analysis by minimizing the rows and columns through the use of dimension tables. These strengths are also amplified that much further with the ability to create an interface that works in tandum with the data warehouse. Having an interface to create SQL statement from OLAP operations allows for not only the generation of queries that meet ones' requirements quicker, but is also is advantageous in getting those who know nothing about querying or SQL to be able to pick up the concepts quickly. Perhaps the greatest advantage of the upgrade to a data warehouse is how it is forward-going. As long as data goes through the transformations necessary to be in the format given by the USRH's initial data, new data can be added to the data warehouse quickly and continue to be queried by the interface in the future.