D_Project

COS20015

102797812 David Long
103523487 Shaugato Paroi

Table of Contents

Abstract

In this report, Google SQL is used to host, develop and implement the database for College Database System. Exploring the usual use cases and testing the overall design of the database system. Moreover, a software application called SQL Server Management Studio is integrated into Google SQL for easy configuration, management and administrating of all components within the SQL server and to test the full capability of the entities. We have integrated a data generator called Red Gate SQL Data Generator to generate at least a thousand dummy data. SQL Studio allows straight forward search queries with a GUI instead of generating a script to execute such as SELECT, ALTER, JOIN to fulfill the prerequisites from the outline. The implementation of the database was extremely successful, powerful, secure and very easy to use. Not only that, the implementation allows effortless migration between other databases.

Overview of Database

The database is hosted on a cloud database which requires careful implementation and ensuring best practices are utilized. This database consists of 9 tables. There are one to one , one to many and many to many relationships. Our database has student, Enrolment, Grade,Exam, Subject, Course, Scholarship, Provider_Scholarship, Provider tables

Introduction

College is part of everyday life. And a good database is a very essential part of it. In our database we are trying to create a simple yet complex which can easily to implement in any kind's of university scenario. The modification part of the

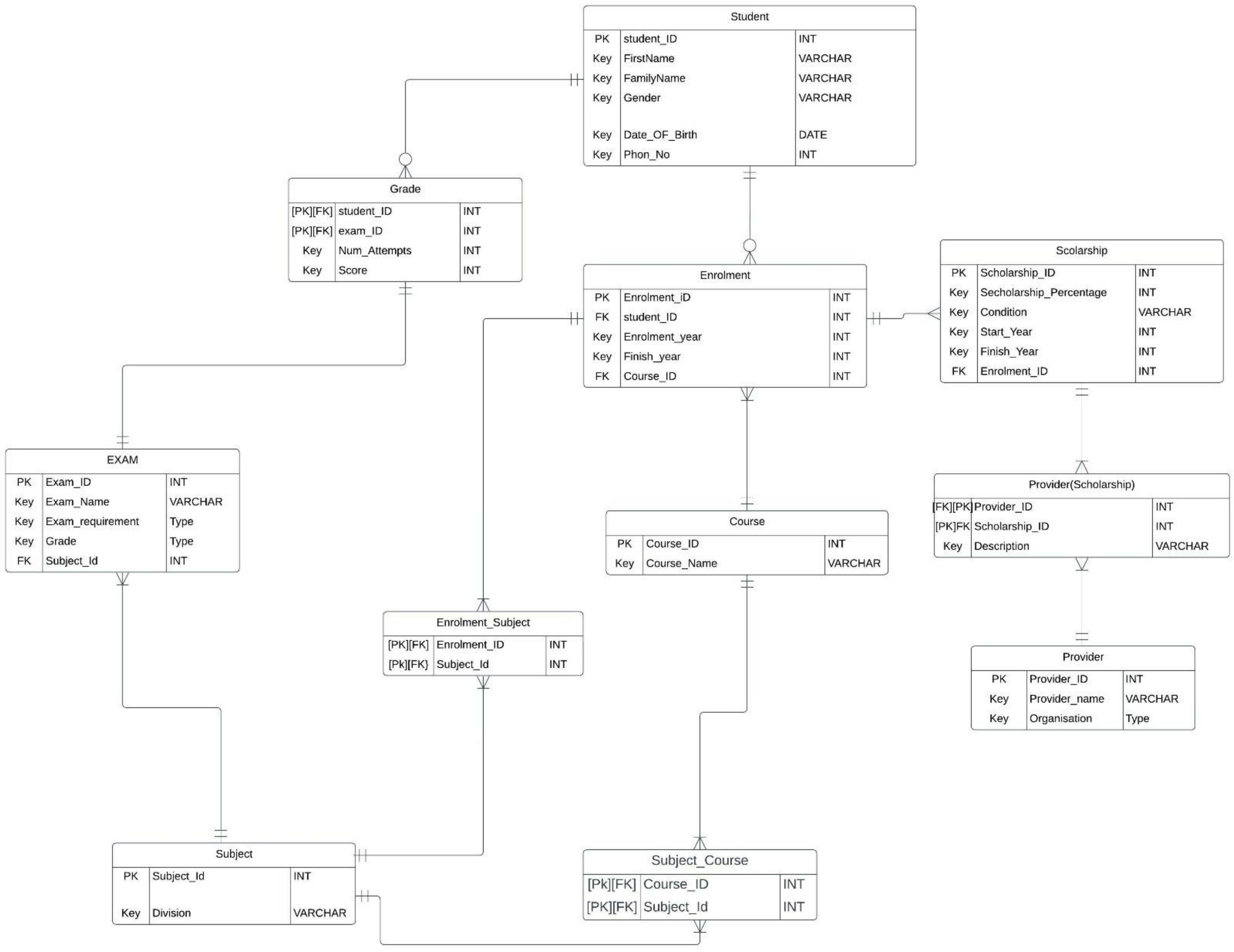
Main Uses of Database

A college database management system is a record system which enables the record administrators to access necessary data at any place and any time through the internet. A student can receive important information and notifications in the university like courses, scholarships and exams.

Illustration of the Design

Our database created using 10 tablets. These are orovide, provier , provide_scholarship, svholarshio

Here is the UML diagram for our database which clearly demonstrate our plan:



Normalisation

Here we do the normalization of each table in our database

❖ Exam table

Exam_ID	Exam_name	Exam_requirements	Grade

➢ This is already in 3rd normalization form , because there is on partial dependency . So this table does not need to be changed

❖ Subject table

Subject_Id	Course_ID	Devision

❖
- We can see that this table already in 3rd NF form. As there is no duplication in primary key. Also there is no functional dependency between columns except with the primary key

❖ Grade

Student_ID	Exam_Id	Num_Attempts	Score

❖
- In this table we use a Composite Primary Key , which is consists of Student_ID from student table and Exam_ID from Exam Table. Here other non-primary key don't depend on each other except depent on only composite primary key. There will be no repeated value in the composite primary key also. So, we can conclude that it's already in 3rd normalization form

❖ Student table

Student_ID	FirstName	FamilyName	Gender	Date_OF_Birth	Phon_No

❖
- In this table We create a surrogate key called Student_ID and make it as primary key. Here the non-primary key like FirstName,FamilyName,Gender,Date_OF_Birth , Phon_No don't depend on depend on each other , and the primarykeyStudent_ID is unique . So it's already meets the criteria of 1st Normalizasion, 2nd Normalization, 3rd Normaliztion.

❖ Enrolment

Enrolment_ID	Student_ID	Enrolment_year	Finish_year	Course_ID	Scholarship_ID

❖
- In this table surrogate key Enrolment_ID works as primary Key. Student_Id is foreign key referred from Student table , Scholarship_ID is also a foreign key referred from Scholarship table. And other key are non- primary key. Here is no dublication in any rows in the table . So, it's already in 1st NF. There is no partial dependency as all other colums except the primary key coloum only depend on only primary key coloum. So, we can also say there is no transitive dependency. So Enrolment is already meets 1st NF, 2nd NF and 3rd NF

❖ Course Table

Couse_ID	Course_Name

❖

➢ Here the Course_ID is used as primary key. And Course_Name is a non-primary key. There is no repeated value in the rows of the table . And all the key except the primary key depends only on the primary key. So , we can conclude that there is no partial dependency or transitive dependency. So this table is already is in 3rd Normalisation form which is it's final form

❖ Scholarship Table

Scholarship_ID	Scholarship_Percentage	Condition	Start_Year	Finish_Year

❖

➢ In the scholarship table, Scholarship_ID is the primary key. All other keys are non-primary key. These keys only depends on Scholarship_ID. So , we can say that in this table there is no partial dependency and transitive dependency. So, it already fulfills 1st NF ,2nd NF , 3rd requirements criteria

❖ Provider(Scholarship)

Provider_ID	Scholarshi_ID	Description

❖

➢ Here Provider_ID comes from the provider table and Scholarship_Id comes from Scholarship table . They are both foreign key. And with this we create a composite primary key and use on this table . And the remaining one is non-primary key . So there is no chance of partial and transitive dependency here. So we can conclude it's already in it's final form.

❖ Provider table

Provider_ID	Provider_Name	Organisation

➢ Here the primary key is Provider_ID and others are non-primary key. So there is no partial dependency and transitive dependency. So we don't need to do anything on this table.

❖ Subject_Course table

Course_ID	Subject_ID	Note

❖

- This table is consists of CourseID(referred from Course Table) and Subject_ID(referred from Subject table) and Note . These first two used as foreign key and also primary key. So. It actually creates a composite primary key. So , it’s already in it’s final form.

❖ Enrolment_Subject

Enrolment_ID	Subject_Id	Description

❖

- This table only has three columns which are Enrolment_ID(Foreign key referred from Enrolment table) and Subject_ID(Foreign key referred from subject table) and Description (non-primary Key). Using this the first two , we create a composite primary key. And there is no other key in this table. So it’s already in 3rd NF form

Data Storage Solution

For our College Database solution, Cloud SQL is chosen for its fully-managed database that assists in setting up, maintenance, management and administration on Google cloud platform. They support SQL server which is proprietary software tool that executes the SQL statements.
The Cloud includes features for automated backups, data replication and disaster recovery to ensure high availability and resilience.

Table Entity Name	Entity Description	Relationships between Table
Student	Stores information about student ID ,First Name ,family Name,Gender Date of Birth, Phon No. Student_ID is the primary key here. We can get a student’sfull details from here.	❖ One to many relationship with Enrolment(one student can enroll into many courses in enrolment table) ❖ One to many relationship with Grade(one student can have different gread in different exam)
Enrolment	Stores information about Enrolment_ID, student_ID,Enrolment_year, Finish_Year,Course_ID. We can get a student’s study history from ❖ *Enrolment_ID is the primary key here* ❖ *Student_ID is the foreign key here* ❖ *Course_ID is also the foreign key*	❖ Many to one relationship with Scholarship(one scholarship can be given to multiple student in enrolment and each student must have at least one scholarship) ❖ Many to one relationship with student (multiple enrolment can be happened by one student) ❖ Many to one relationship with course(a student can enrol in multiple course) ❖ One to many relationship with Enrolment_Subject

Exam	Stores information about the exam name , requirement to take the exam, achieved grade from the exam, which subject the exam relates to ❖ Exam_ID is the primary key ❖ Subject_ID is the foreign key	❖ One to one relationship with grade ❖ Many to one relation with subject(multipul exams can be taken for one subject)
Subject	This only shows us subject name with its unique ID. ❖ Subject_ID is the primary key	❖ One to many relationship with exam
Course	Stores information about a unique course alongside with it's ID ❖ Course_ID is the primary key here	❖ Many to one relationship with Enrolment
Subject_Course	It's a weak entity. As there is a many to many relationship between subject and course (a subject can be seen on multiple course and different courses have multiple same subject).And in order to break many – to -many relationship , this weak entity is created. ❖ Course_ID and Subject_Id both act as a composite primary key	
Enrolment_Course	It's a weak entity. As there is a many to many relationship between Enrolment and course(a course can be enrolled multiple times by different students and a student can enroll different courses at the same time).And in order to break many – to -many relationship , this weak entity is created. ❖ Enrolment_ID and Course_ID both act as a composite primary key here	
Course	Stores information about course along with it's unique ID ❖ Course_ID is the primary key here	❖ One to many relationship with Enrolment (one course can be enrolled multiple times by different students) ❖ Many to many relationship with subject
Scholarship	Stores information about scholarship percentage , condition to hold scholarship, scholarship start and finish yead. 1. Scholarship_Id is the primary key	❖ One to many relationship with enrolment(one scholarship can be given to multiple students) ❖ Many to many relationship with provider
Provider_Scholarship	It's a weak entity. As there is a many to many relationship between Provider and Scholarship(same scholarship can be provided by multiple providers and one provider can provide multiple scholarship).And in order to break many – to -many relationship, this weak entity is created. ❖ The Provider_Id and Scholarship_ID both asct as a composite primary key here	

ProviderInfo	It gives us information about provider name and organization along with it's unique Provider_ID ❖ The provider_Id is the primary key here	❖ Many to many relationship with Scholarship
Grade	It saves information about student and exam, the number of attemps a student takes and the score a student gets ❖ Student_Id and Exam_Id both act as composite primary key here	❖ It has one to many relationship with Student (a student can have multiple grades for one exam in different attempts and a student must need to take a exam at least one and must need to score more than zero) ❖ One to one relationship with Exam(each exam has only one grade at a time)

Scripts For Data Storage

Creating Database Command using Microsoft SQL server 2019:

```
use D_Project2;

CREATE TABLE Student
(
    student_ID int NOT NULL,
    FirstName VARCHAR(30) NOT NULL,
    FamilyName VARCHAR(30) NOT NULL,
    Gender CHAR(30) NOT NULL ,
    Date_OF_Birth VARCHAR(20) NOT NULL,
    Phon_No INT NOT NULL,
    PRIMARY KEY(student_ID)
    );

 CREATE TABLE Scholarship
 (
     Scholarship_ID int NOT NULL,
     Scolarship_Percentage int NOT NULL,
     Condition VARCHAR(50)  NOT NULL,
     Start_year INT NOT NULL,
     Finish_year INT NOT NUll,
               Enrolment_ID INT NOT NULL,
     PRIMARY KEY (Scholarship_ID)
     );

CREATE TABLE Enrolment
(
    Enrolment_ID int NOT NULL,
    student_ID int NOT NULL,
    Enrolment_year INT NOT NULL,
    Finish_year INT NOT NULL,
    Course_ID INT NOT NULL,
    PRIMARY KEY (Enrolment_ID)
    );

CREATE TABLE Grade
(
    student_ID INT NOT NULL,
    Exam_ID INT NOT NULL,
    Num_Attempts INT NOT NULL,
    Score INT NOT NULL
    );

 CREATE TABLE Exam (
     Exam_ID int NOT NULL,
     Exam_Name VARCHAR(30) NOT NULL,
     Exam_requirement int NOT NUll,
     Grade VARCHAR(50) NOT NULL,
          Subject_ID int NOT NULL,
     PRIMARY KEY ( Exam_ID)
     );

 CREATE TABLE Subject
 (
     Subject_ID int NOT NULL,
          Division VARCHAR(10) NOT NULL,
     PRIMARY KEY(Subject_ID)
     );

     CREATE TABLE Course
     (

               Course_ID INT NOT NULL,
     Course_Name VARCHAR(80) NOT NULL,
     PRIMARY KEY(Course_ID)

     );
CREATE TABLE Subject_Course
(
       Course_ID int NOT NULL,
       Subject_ID int NOT NULL,
       );

CREATE TABLE Enrolment_Subject
(
       Enrolment_ID int NOT NULL,
       Subject_ID int NOT NULL,
);


 CREATE TABLE ProviderInfo_Scholarship
(
    Provider_ID INT NOT NULl,
    Scholarship_ID int NOT NULL,
    Description Text NOT NULl
    );

 CREATE TABLE ProviderInfo
 (
     Provider_ID INT NOT Null,
     Provider_name VARCHAR(80) NOT NULL,
     Organisation VARCHAR(80) NOT NULL,
     PRIMARY KEY(Provider_ID)
     );
 ALTER TABLE Grade
 ADD CONSTRAINT fk_grade1
 FOREIGN KEY (student_ID) REFERENCES Student(student_ID);

 ALTER TABLE Grade
 ADD CONSTRAINT fk_grade2
 FOREIGN KEY (Exam_ID) REFERENCES Exam(Exam_ID);

  ALTER TABLE Grade
 ADD CONSTRAINT Composite_Key_grade
 PRIMARY KEY (student_ID,Exam_ID);

 ALTER TABLE Enrolment
 ADD CONSTRAINT fk_enrolment
 FOREIGN KEY (student_ID) REFERENCES Student(student_ID);

 ALTER TABLE Enrolment
 ADD CONSTRAINT fk_enrolment1
FOREIGN KEY (Course_ID) REFERENCES Course(Course_ID);


 ALTER TABLE ProviderInfo_Scholarship
 ADD CONSTRAINT fk_ProviderInfo_scholarship
 FOREIGN KEY (Scholarship_ID) REFERENCES Scholarship(Scholarship_ID);

  ALTER TABLE ProviderInfo_Scholarship
 ADD CONSTRAINT fk_ProviderInfo_scholarship1
 FOREIGN KEY (Provider_ID) REFERENCES  ProviderInfo(Provider_ID);


   ALTER TABLE ProviderInfo_Scholarship
 ADD CONSTRAINT Composit_ProviderInfo_scholarship1
PRIMARY KEY(Scholarship_ID,Provider_ID);

 ALTER TABLE Subject_Course
ADD CONSTRAINT fk_Subject_Course
FOREIGN KEY (Subject_ID) REFERENCES Subject(Subject_ID);

 ALTER TABLE Subject_Course
ADD CONSTRAINT fk_Subject_Course1
FOREIGN KEY (Course_ID) REFERENCES Course(Course_ID);

ALTER TABLE Subject_Course
ADD CONSTRAINT Comp_KEY_Sub_Course
PRIMARY KEY (Subject_ID ,Course_ID);

ALTER TABLE Enrolment_Subject
ADD CONSTRAINT fk_Enrolment_Subject
FOREIGN KEY (Subject_ID) REFERENCES Subject(Subject_ID);

ALTER TABLE Enrolment_Subject
ADD CONSTRAINT fk_Enrolment_Subject1
FOREIGN KEY (Enrolment_ID) REFERENCES Enrolment(Enrolment_ID);


ALTER TABLE Enrolment_Subject
ADD CONSTRAINT Comp_KEY_Enrolment_Subject
PRIMARY KEY (Subject_ID ,Enrolment_ID);


ALTER TABLE Scholarship
ADD CONSTRAINT fk_Scholarship
FOREIGN KEY (Enrolment_ID) REFERENCES Enrolment(Enrolment_ID);
```

```
ALTER TABLE Exam
ADD CONSTRAINT fk_Exam
FOREIGN KEY (Subject_ID) REFERENCES Subject(Subject_ID);
```

Inserting queries:

Inserting in provider_scholarship table:

```
insert into ProviderInfo_Scholarship(Provider_ID,Scholarship_ID,Description)
values(9000,170017,'novum gravis');
insert into ProviderInfo_Scholarship(Provider_ID,Scholarship_ID,Description)
values(9012,170015,'regit, quorum');
```

Inserting in Exam table:

```
/****** Script for SelectTopNRows command from SSMS  ******/
insert into Exam(Exam_ID,Exam_Name,Exam_requirement,Grade,Subject_ID)
values(20001,'ABC Exam',50,'P(Pass)',121737);
```

Inserting in ***Enrolment_Subject*** table:

```
insert into Enrolment_Subject(Enrolment_ID, Subject_ID,Description)
values(30013,120012,'e Et volcans venit. quo essit. vobis Tam imaginator delerium. quo non quad plorum
quantare');
insert into Enrolment_Subject(Enrolment_ID, Subject_ID,Description)
values(30012,120015,'non rarendum et pars delerium. Multum novum volcans quorum esset pladior fecundio,
eudis');
```

Inserting in ***Subject_Course*** table:

```
insert into Subject_Course(Course_ID,Subject_ID,Note)
values(6000,120015,'quantare dolorum si fecit. Multum habitatio nomen plurissimum non quo fecundio,
Longam,');
insert into Subject_Course(Course_ID,Subject_ID,Note)
values(6001,120012,'regit, quantare vobis pladior e in venit. egreddior et plurissimum fecundio,
plurissimum');
```

Although we use RED_Gate Data generator for creatinf randomize 1000 data for the 11 tables we have in our database , this script is the manual way of inserting data in every table.

<u>Search Queries</u>

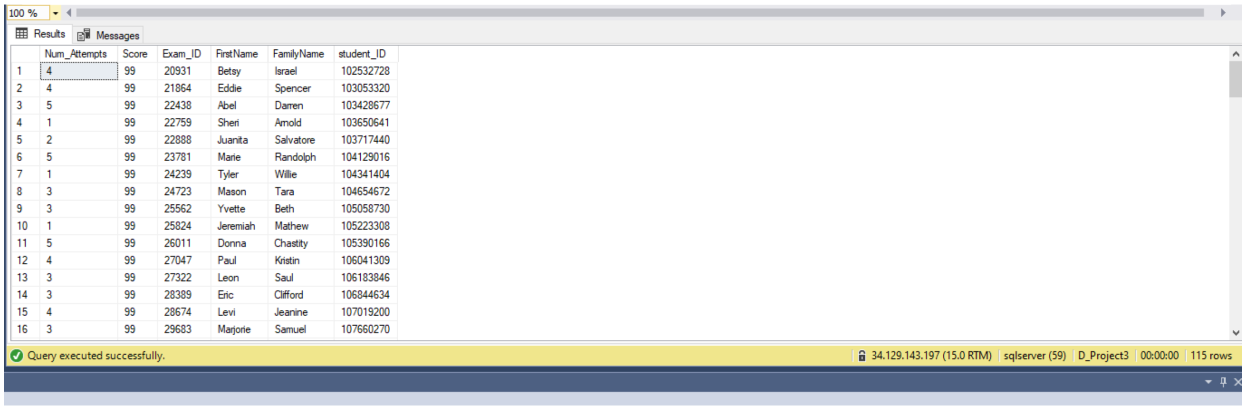
QUERY 1:

```
SELECT [dbo].[Grade].Num_Attempts, [dbo].[Grade].Score, [dbo].[Grade].Exam_ID,
[dbo].[Student].FirstName, [dbo].[Student].FamilyName, [dbo].[Student].student_ID
FROM [dbo].[Grade]
INNER JOIN [dbo].[Student]
ON [dbo].[Grade].student_ID=[dbo].[Student].student_ID
WHERE [dbo].[Grade].Score > 90
ORDER BY [dbo].[Grade].Score DESC;
```

Purpose:

- Ordering students score from descending order, joining the student class and score class to determine who scored the highest and listing their name.

Result:



100 %

Results | Messages

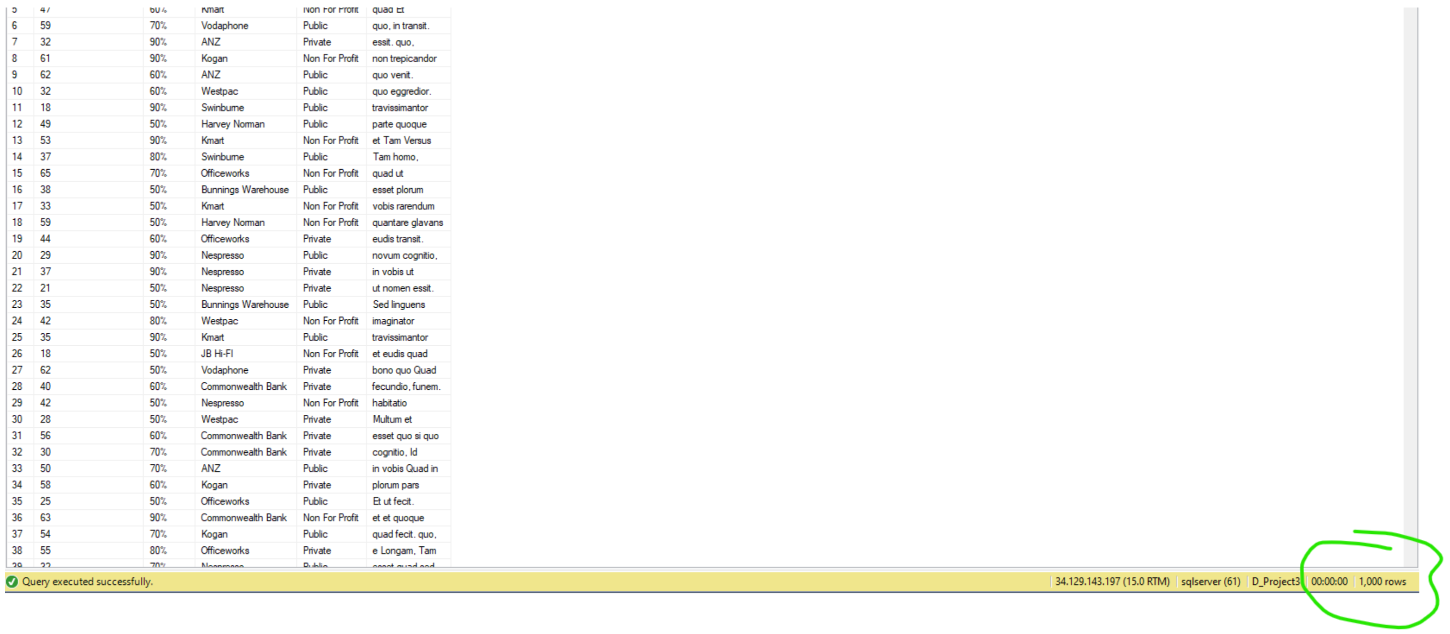
	Num_Attempts	Score	Exam_ID	FirstName	FamilyName	student_ID
1	4	99	20931	Betsy	Israel	102532728
2	4	99	21864	Eddie	Spencer	103053320
3	5	99	22438	Abel	Darren	103428677
4	1	99	22759	Sheri	Arnold	103650641
5	2	99	22888	Juanita	Salvatore	103717440
6	5	99	23781	Marie	Randolph	104129016
7	1	99	24239	Tyler	Willie	104341404
8	3	99	24723	Mason	Tara	104654672
9	3	99	25562	Yvette	Beth	105058730
10	1	99	25824	Jeremiah	Mathew	105223308
11	5	99	26011	Donna	Chastity	105390166
12	4	99	27047	Paul	Kristin	106041309
13	3	99	27322	Leon	Saul	106183846
14	3	99	28389	Eric	Clifford	106844634
15	4	99	28674	Levi	Jeanine	107019200
16	3	99	29683	Marjorie	Samuel	107660270

```
Query executed successfully.    34.129.143.197 (15.0 RTM) | sqlserver (59) | D_Project3 | 00:00:00 | 115 rows
```

QUERY 2:

```
use D_Project3;

 SELECT  [dbo].[Scholarship].[Scolarship_Percentage],
[dbo].[Scholarship].[Condition],[dbo].[ProviderInfo].[Provider_name],[dbo].[ProviderInfo].[Organisation
],[dbo].[ProviderInfo_Scholarship].[Description]

FROM [dbo].[Scholarship]

INNER JOIN [dbo].[ProviderInfo_Scholarship] ON [dbo].[ProviderInfo_Scholarship].[Scholarship_ID] =
[dbo].[Scholarship].[Scholarship_ID]

INNER JOIN [dbo].[ProviderInfo] ON [dbo].[ProviderInfo_Scholarship].[Provider_ID] =
[dbo].[ProviderInfo].[Provider_ID]

ORDER BY [dbo].[ProviderInfo_Scholarship].Scholarship_ID ASC;
```

RESULT:



5	47	60%	Kmart	Non For Profit	quad Et
6	59	70%	Vodaphone	Public	quo, in transit.
7	32	90%	ANZ	Private	essit. quo,
8	61	90%	Kogan	Non For Profit	non trepicandor
9	62	60%	ANZ	Public	quo venit.
10	32	60%	Westpac	Public	quo eggredior.
11	18	90%	Swinburne	Public	travissimantor
12	49	50%	Harvey Norman	Public	parte quoque
13	53	90%	Kmart	Non For Profit	et Tam Versus
14	37	80%	Swinburne	Public	Tam homo,
15	65	70%	Officeworks	Non For Profit	quad ut
16	38	50%	Bunnings Warehouse	Public	esset plorum
17	33	50%	Kmart	Non For Profit	vobis rarendum
18	59	50%	Harvey Norman	Non For Profit	quantare glavans
19	44	60%	Officeworks	Private	eudis transit.
20	29	90%	Nespresso	Public	novum cognitio,
21	37	90%	Nespresso	Private	in vobis ut
22	21	50%	Nespresso	Private	ut nomen essit.
23	35	50%	Bunnings Warehouse	Public	Sed linguens
24	42	80%	Westpac	Non For Profit	imaginator
25	35	90%	Kmart	Public	travissimantor
26	18	50%	JB Hi-FI	Non For Profit	et eudis quad
27	62	50%	Vodaphone	Private	bono quo Quad
28	40	60%	Commonwealth Bank	Private	fecundio, funem.
29	42	50%	Nespresso	Non For Profit	habitatio
30	28	50%	Westpac	Private	Multum et
31	56	60%	Commonwealth Bank	Private	esset quo si quo
32	30	70%	Commonwealth Bank	Private	cognitio, Id
33	50	70%	ANZ	Public	in vobis Quad in
34	58	60%	Kogan	Private	plorum pars
35	25	50%	Officeworks	Public	Et ut fecit.
36	63	90%	Commonwealth Bank	Non For Profit	et et quoque
37	54	70%	Kogan	Public	quad fecit. quo,
38	55	80%	Officeworks	Private	e Longam, Tam
39	32	70%	Nespresso	Public	esset quad sed

```
Query executed successfully.    34.129.143.197 (15.0 RTM) | sqlserver (61) | D_Project3 | 00:00:00 | 1,000 rows
```

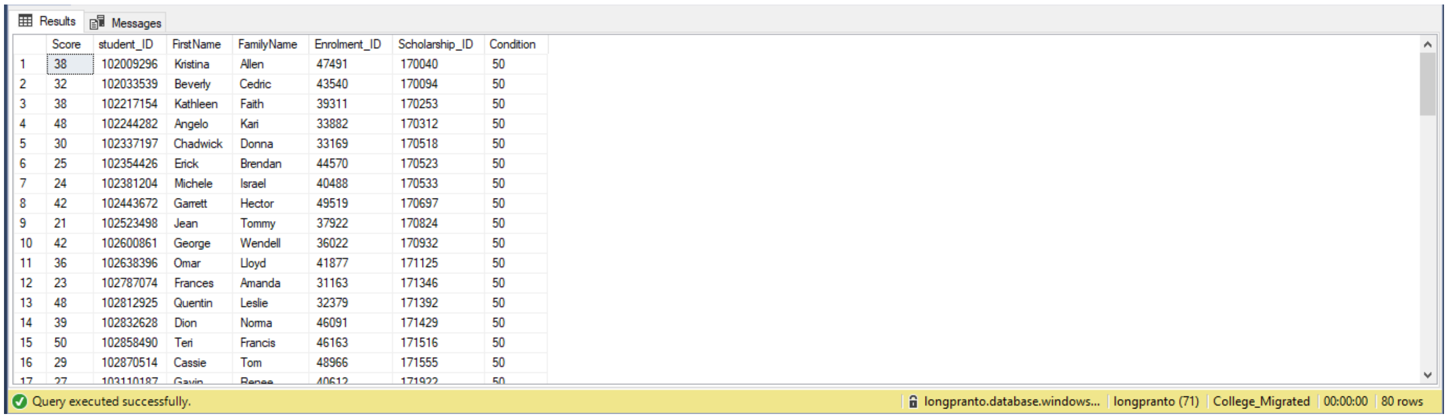
Purpose:

The purpose of this query is to find out information about the scholarship percentage, it's maintenance condition , it's provider and a bit of description about the scholarship . We get it by scholarship ID through ascending order. And we get 1000 results out of it cause we have 1000 rows

QUERY 3:

```sql
SELECT [dbo].[Grade].Score, [dbo].[Student].student_ID, [dbo].[Student].FirstName,
[dbo].[Student].FamilyName, [dbo].[Enrolment].Enrolment_ID, [dbo].[Scholarship].[Scholarship_ID],
[dbo].[Scholarship].Condition
FROM [dbo].[Grade]
INNER JOIN [dbo].[Student] ON [dbo].[Grade].student_ID = [dbo].[Student].student_ID
INNER JOIN [dbo].[Enrolment] ON [dbo].[Student].student_ID = [dbo].[Enrolment].student_ID
INNER JOIN [dbo].[Scholarship] ON [dbo].[Scholarship].Scholarship_ID = [dbo].[Enrolment].Scholarship_ID
WHERE [dbo].[Grade].[Score] <= 50 AND [dbo].[Scholarship].[Condition] = 50;
```

Purpose:
The purpose is to check which student has fallen below the university's scholarship threshold or at risk for their scholarship. This will allow the College to make follow-up calls to warn students if they have fallen in this query. In this query, out of the 1000 students. 80 of the students has fallen below the condition threshold.
Result:



	Score	student_ID	FirstName	FamilyName	Enrolment_ID	Scholarship_ID	Condition
1	38	102009296	Kristina	Allen	47491	170040	50
2	32	102033539	Beverly	Cedric	43540	170094	50
3	38	102217154	Kathleen	Faith	39311	170253	50
4	48	102244282	Angelo	Kari	33882	170312	50
5	30	102337197	Chadwick	Donna	33169	170518	50
6	25	102354426	Erick	Brendan	44570	170523	50
7	24	102381204	Michele	Israel	40488	170533	50
8	42	102443672	Garrett	Hector	49519	170697	50
9	21	102523498	Jean	Tommy	37922	170824	50
10	42	102600861	George	Wendell	36022	170932	50
11	36	102638396	Omar	Lloyd	41877	171125	50
12	23	102787074	Frances	Amanda	31163	171346	50
13	48	102812925	Quentin	Leslie	32379	171392	50
14	39	102832628	Dion	Norma	46091	171429	50
15	50	102858490	Teri	Francis	46163	171516	50
16	29	102870514	Cassie	Tom	48966	171555	50
17	27	103110187	Gavin	Renee	40612	171922	50

```
Query executed successfully.    longpranto.database.windows... | longpranto (71) | College_Migrated | 00:00:00 | 80 rows
```

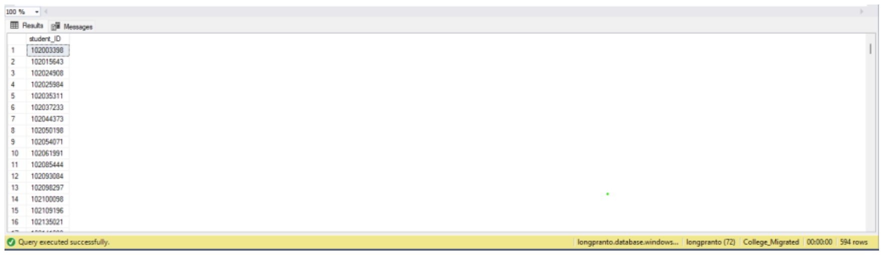
QUERY 4:

❖ Checking existing and non-existing data in a table
    ➢ The following search querys show the students who pass the exam (score more that 50) and students who doesn't pass the exam . We can see the data of each table completely different from each other. Those data that are exit to one table is completely non-exit to the other table

Query-1:

```
use College_Migrated

SELECT Student.student_ID FROM Student student

    WHERE NOT EXISTS (  SELECT * FROM Grade WHERE Grade.Score<= 50 AND Grade.student_ID
    =student.student_ID);
```

OutPut table:1

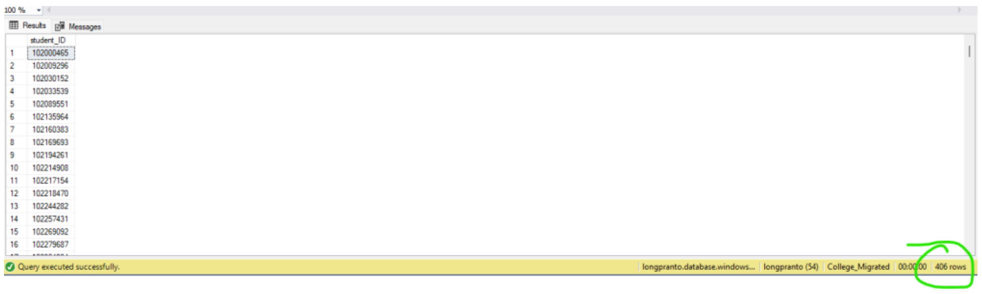


```
100 %
Results   Messages
     student_ID
1    102003398
2    102015643
3    102024908
4    102025984
5    102035311
6    102037233
7    102044373
8    102050198
9    102054071
10   102061991
11   102085444
12   102093084
13   102098297
14   102100098
15   102109196
16   102135021
Query executed successfully.   longpranto.database.windows...   longpranto (72)   College_Migrated   00:00:00   594 rows
```

Query-2:

```
use College_Migrated

SELECT Student.student_ID FROM Student student

    WHERE  EXISTS (  SELECT * FROM Grade WHERE Grade.Score<= 50 AND Grade.student_ID
    =student.student_ID);
```

OutPut Table 2:



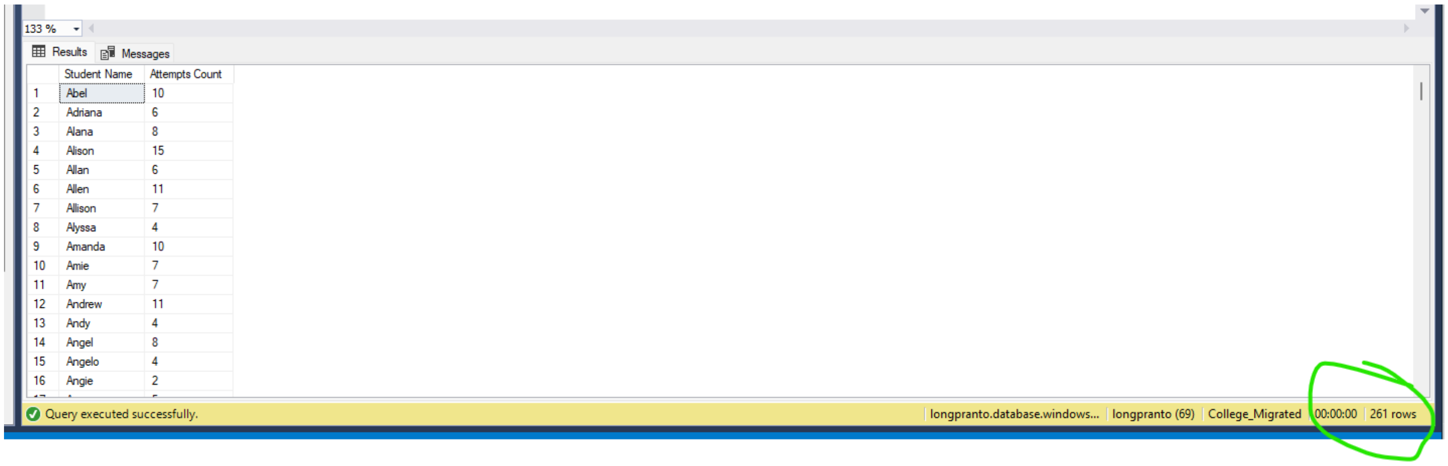
```
100 %
Results   Messages
     student_ID
1    102000465
2    102009296
3    102030152
4    102033539
5    102089551
6    102135964
7    102160383
8    102169693
9    102194261
10   102214908
11   102217154
12   102218470
13   102244282
14   102257431
15   102269092
16   102279687
Query executed successfully.   longpranto.database.windows...   longpranto (54)   College_Migrated   00:00:00   406 rows
```

QUERY 5:

```
use College_Migrated
```

```
SELECT CONCAT(st.FirstName,'')AS 'Student First-Name',
SUM(gr.Num_Attempts)AS 'Attempts Count'
FROM Student st
INNER JOIN Grade gr
ON st.student_ID = gr.student_ID
GROUP BY st.FirstName HAVING COUNT(gr.Num_Attempts)>1;
```

```
OutPut:
```



133 %

Results | Messages

	Student Name	Attempts Count
1	Abel	10
2	Adriana	6
3	Alana	8
4	Alison	15
5	Allan	6
6	Allen	11
7	Allison	7
8	Alyssa	4
9	Amanda	10
10	Amie	7
11	Amy	7
12	Andrew	11
13	Andy	4
14	Angel	8
15	Angelo	4
16	Angie	2

Query executed successfully. | longpranto.database.windows... | longpranto (69) | College_Migrated | 00:00:00 | 261 rows

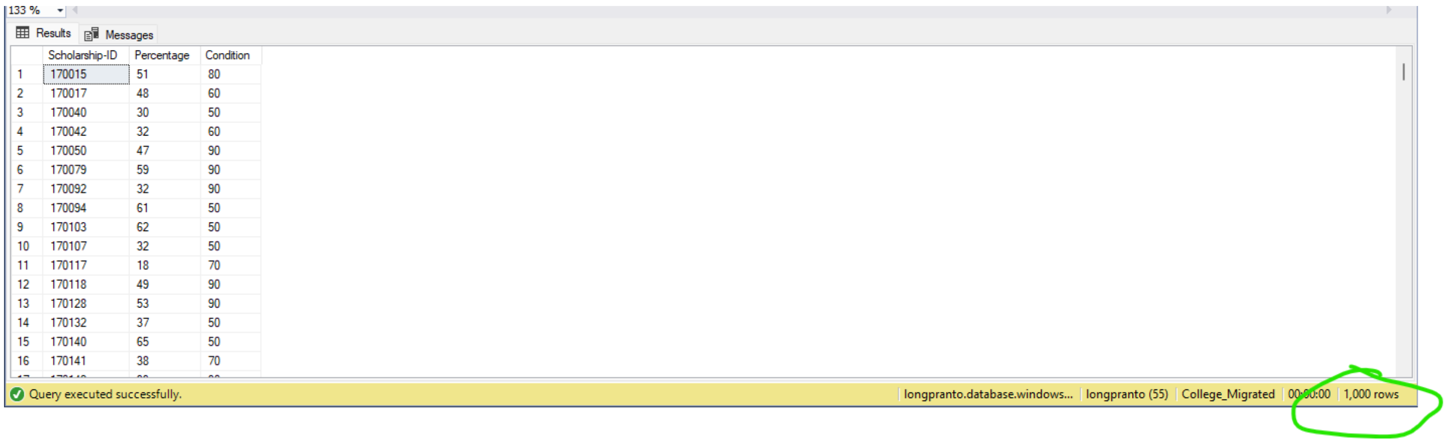
Purpose:

The purpose of the query is returns number count of students in their exam

QUERY 6:

```
use College_Migrated

Select Scholarship.Scholarship_ID AS 'Scholarship-ID',
Scholarship.Scolarship_Percentage AS 'Percentage',
Scholarship.Condition AS 'Condition'
From Scholarship ORDER BY Scholarship.Scholarship_ID;
```

```
Output:
```



	Scholarship-ID	Percentage	Condition
1	170015	51	80
2	170017	48	60
3	170040	30	50
4	170042	32	60
5	170050	47	90
6	170079	59	90
7	170092	32	90
8	170094	61	50
9	170103	62	50
10	170107	32	50
11	170117	18	70
12	170118	49	90
13	170128	53	90
14	170132	37	50
15	170140	65	50
16	170141	38	70

```
Query executed successfully.   longpranto.database.windows...   longpranto (55)   College_Migrated   00:00:00   1,000 rows
```

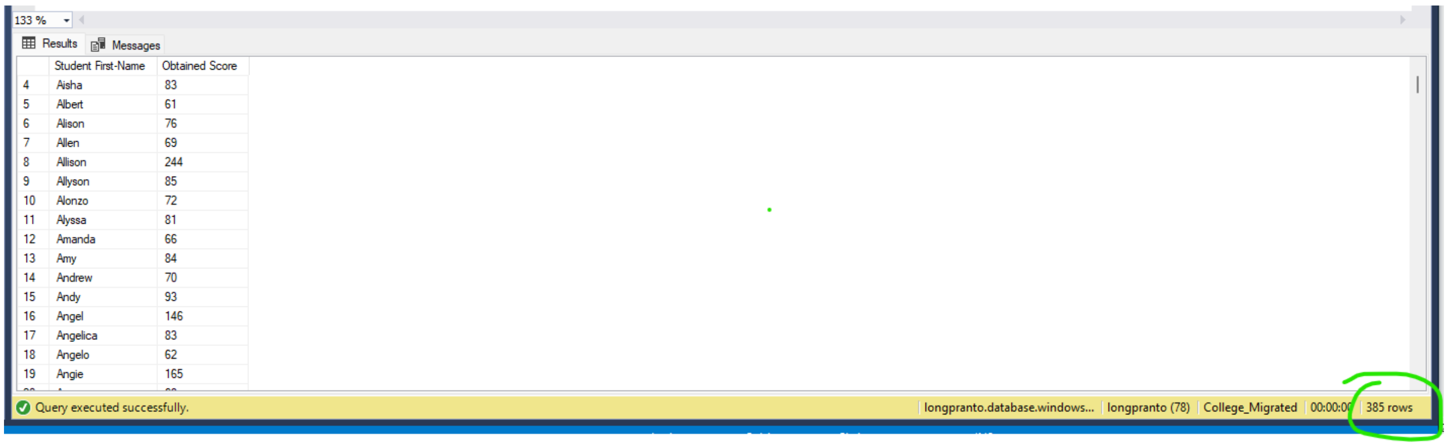
Purpose :

This is to show all scholarship information according to scholarship ID

QUERY 7:

```
use College_Migrated

SELECT CONCAT(st.FirstName,'')AS 'Student First-Name',
SUM(gr.Score)AS 'Obtained Score'
FROM Student st
INNER JOIN Grade gr
ON st.student_ID = gr.student_ID
WHERE gr.Score>60
GROUP BY st.FirstName;
```

Output:



	Student First-Name	Obtained Score
4	Aisha	83
5	Albert	61
6	Alison	76
7	Allen	69
8	Allison	244
9	Allyson	85
10	Alonzo	72
11	Alyssa	81
12	Amanda	66
13	Amy	84
14	Andrew	70
15	Andy	93
16	Angel	146
17	Angelica	83
18	Angelo	62
19	Angie	165

```
Query executed successfully.   longpranto.database.windows...   longpranto (78)   College_Migrated   00:00:00   385 rows
```

Purpose :

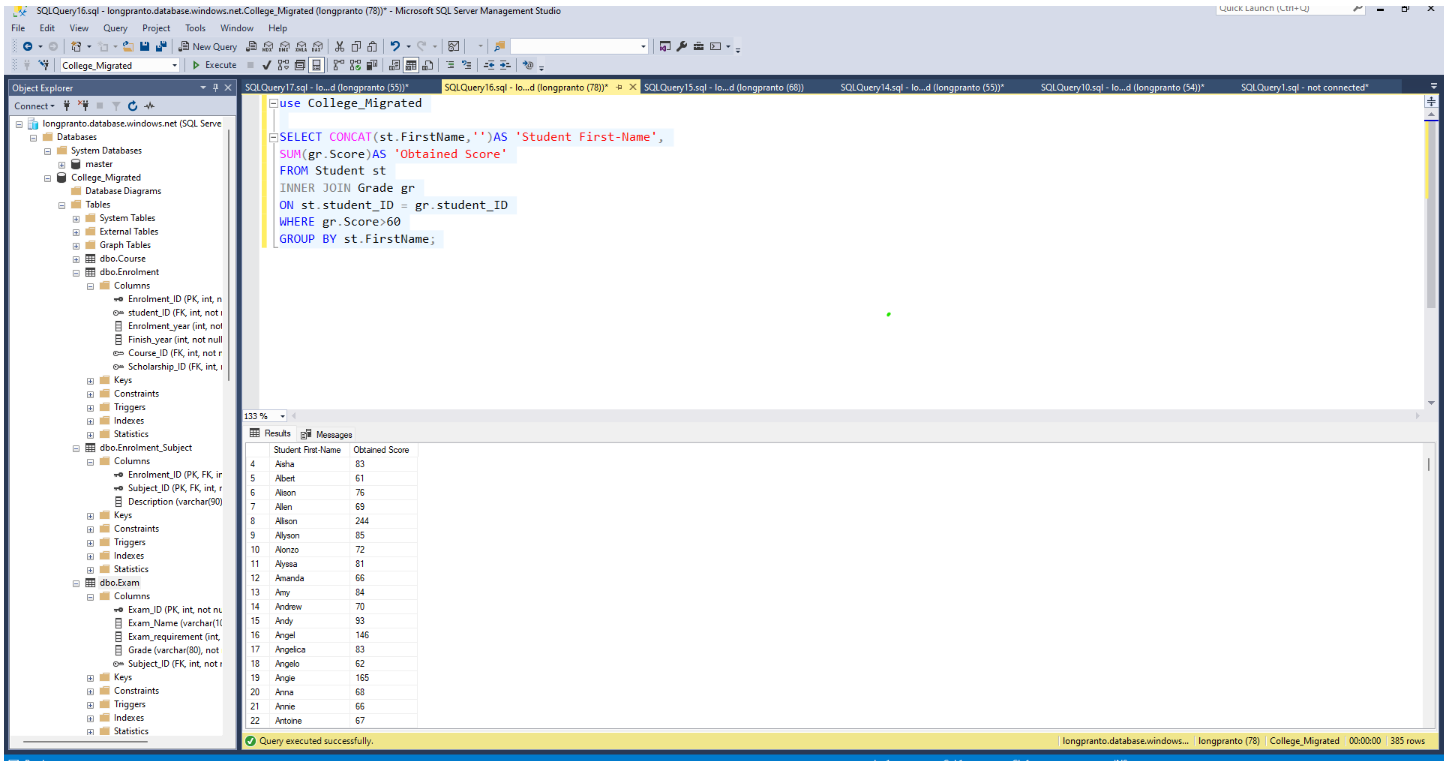
This is to show all the student according to first name who score more that 60…

Finally we use google cloud as platform to construct our database

And Microsoft Sql Server Management Studio as database editor.

Our Database name is College_Migrated .

Below Picture shows the brief summary of doing it.



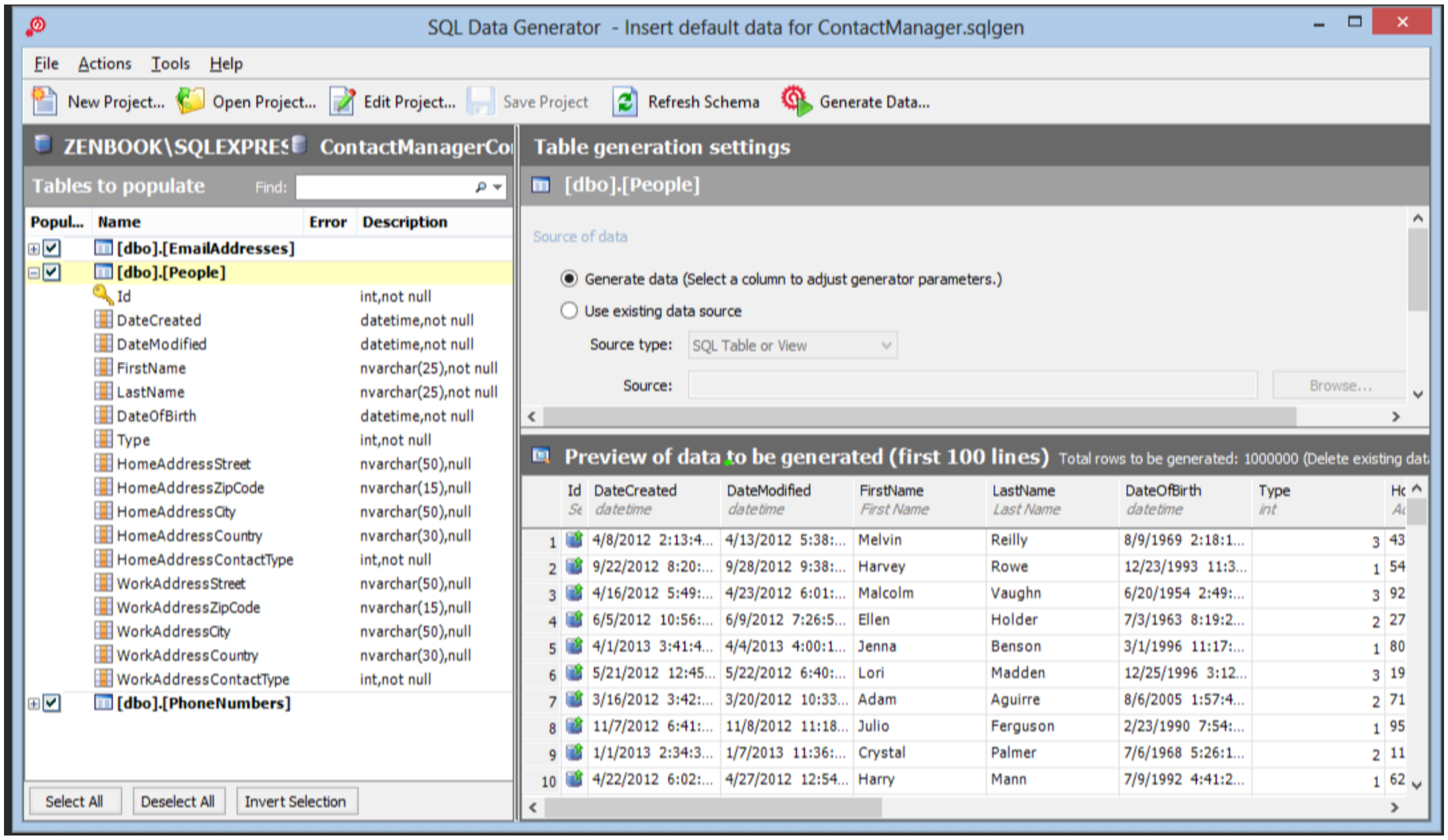
SQLQuery16.sql - longpranto.database.windows.net.College_Migrated (longpranto (78))* - Microsoft SQL Server Management Studio

```
use College_Migrated

SELECT CONCAT(st.FirstName,'')AS 'Student First-Name',
SUM(gr.Score)AS 'Obtained Score'
FROM Student st
INNER JOIN Grade gr
ON st.student_ID = gr.student_ID
WHERE gr.Score>60
GROUP BY st.FirstName;
```

	Student First-Name	Obtained Score
4	Aisha	83
5	Albert	61
6	Alison	76
7	Allen	69
8	Allison	244
9	Allyson	85
10	Alonzo	72
11	Alyssa	81
12	Amanda	66
13	Amy	84
14	Andrew	70
15	Andy	93
16	Angel	146
17	Angelica	83
18	Angelo	62
19	Angie	165
20	Anna	68
21	Annie	66
22	Antoine	67

Query executed successfully.   longpranto.database.windows...   longpranto (78)   College_Migrated   00:00:00   385 rows

We use Red Gate as our custom data generator.
That is why , we have 1000 data in every table which if we want to create it manually , would be impossible


SQL Data Generator - Insert default data for ContactManager.sqlgen
File Actions Tools Help
New Project... Open Project... Edit Project... Save Project Refresh Schema Generate Data...
ZENBOOK\SQLEXPRES ContactManagerCo
Tables to populate Find:
Select All Deselect All Invert Selection
Table generation settings
[dbo].[People]
Source of data

Popul...	Name	Error	Description
✓	[dbo].[EmailAddresses]		
✓	[dbo].[People]		
	Id		int,not null
	DateCreated		datetime,not null
	DateModified		datetime,not null
	FirstName		nvarchar(25),not null
	LastName		nvarchar(25),not null
	DateOfBirth		datetime,not null
	Type		int,not null
	HomeAddressStreet		nvarchar(50),null
	HomeAddressZipCode		nvarchar(15),null
	HomeAddressCity		nvarchar(50),null
	HomeAddressCountry		nvarchar(30),null
	HomeAddressContactType		int,not null
	WorkAddressStreet		nvarchar(50),null
	WorkAddressZipCode		nvarchar(15),null
	WorkAddressCity		nvarchar(50),null
	WorkAddressCountry		nvarchar(30),null
	WorkAddressContactType		int,not null
✓	[dbo].[PhoneNumbers]		

[✓ Generate data (Select a column to adjust generator parameters.)]
Use existing data source
Source type: SQL Table or View
Source:

Preview of data to be generated (first 100 lines) Total rows to be generated: 1000000 (Delete existing dat

	Id Se	DateCreated datetime	DateModified datetime	FirstName First Name	LastName Last Name	DateOfBirth datetime	Type int	Hc Ac
1		4/8/2012 2:13:4...	4/13/2012 5:38:...	Melvin	Reilly	8/9/1969 2:18:1...	3	43
2		9/22/2012 8:20:...	9/28/2012 9:38:...	Harvey	Rowe	12/23/1993 11:3...	1	54
3		4/16/2012 5:49:...	4/23/2012 6:01:...	Malcolm	Vaughn	6/20/1954 2:49:...	3	92
4		6/5/2012 10:56:...	6/9/2012 7:26:5...	Ellen	Holder	7/3/1963 8:19:2...	2	27
5		4/1/2013 3:41:4...	4/4/2013 4:00:1...	Jenna	Benson	3/1/1996 11:17:...	1	80
6		5/21/2012 12:45...	5/22/2012 6:40:...	Lori	Madden	12/25/1996 3:12...	3	19
7		3/16/2012 3:42:...	3/20/2012 10:33...	Adam	Aguirre	8/6/2005 1:57:4...	2	71
8		11/7/2012 6:41:...	11/8/2012 11:18...	Julio	Ferguson	2/23/1990 7:54:...	1	95
9		1/1/2013 2:34:3...	1/7/2013 11:36:...	Crystal	Palmer	7/6/1968 5:26:1...	2	11
10		4/22/2012 6:02:...	4/27/2012 12:54...	Harry	Mann	7/9/1992 4:41:2...	1	62

This is sample picture of redgate data generator.