AY2018/19 SEM 2

BT2102: Data Management and Visualisation

Business Analytics for Greendale College's Success

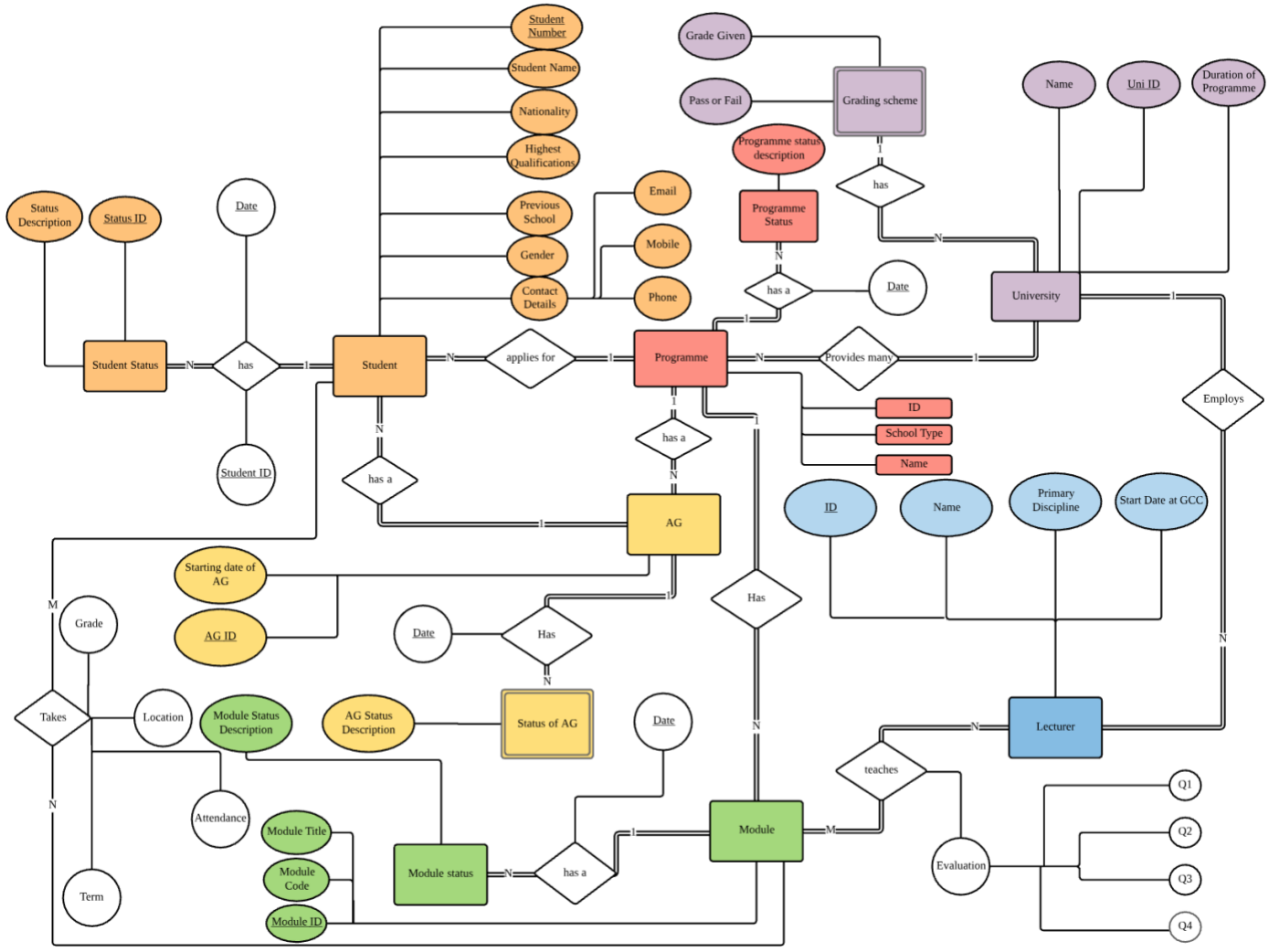
Assignment Part 1

Sean Low Chen Yi A0183743Y

Part 1

Conceptual design: An ER diagram, modelling the data requirements of the problem.

Details: Your ER diagram should clearly indicate key constraints, cardinality and participation. Your diagram should also be annotated with any other details necessary to interpret it correctly.



Part 2

Logical design: A list of tables, derived from the ER diagram.

Details: The list of tables should include each table's name, all its attributes, the chosen data type for each attribute and an indication of any constraints on each attribute. The tables should be normalised to a level you deem suitable for the scenario to avoid data anomalies.

2.1: Table Summary

Underlined: Primary Key

Red coloured: Foreign Key

Red coloured and underlined: Primary and Foreign key

Table number	Table Details
	PROGRAMME DETAILS
1	**GreendaleDPDetails** (GreendaleDegreeProgrammeDetails) (ProgrammeID, ProgrammeName, PartnerUniversities, GradingSystem, ProgrammeDuration, SchoolType)
2	**GreendaleDPStatus** (GreendaleDegreeProgrammeStatus) (ProgrammeID, Programme_Status_Date, ProgrammeStatus, StatusDescription)
3	**AGofProgrammes** (ProgrammeID, AGID)
4	**ModulesinProgramme**(ProgrammeID, ModuleID, ModuleName)
	STUDENT DETAILS
5	**StudentDetails** (StudentID, FirstName, LastName, Nationality, HighestQualifications, PreviousSchool, Gender)
6	**StudentStatus** (StudentID, StatusID, Student_Status_Date, StatusDescription)
7	**StudentContactDetails** (StudentID, Email, Phone, Mobile)
8	**StudentProgramme** (StudentID, ProgrammeID)

	AG (Admission Group)
9	**AGDetails** (AGID, AG_StartDate)
10	**StudentsinAG** (AGID, StudentID)
11	**AGStatus** (AGID, StudentID, AG_Status_Date, StatusDescription)
	MODULE
12	**ModuleDetails** (ModuleID, ProgrammeID, ModuleCode, ModuleTitle)
13	**ModuleStatus** (ModuleID, Module_Date, StudentID, StatusDescription)
14	**StudentGrade** (ModuleID, StudentID, Grade, PassorFail, Term)
15	**StudentModuleAttendance** (ModuleID, StudentID, Location, Attendance, Term)
16	**ModulesbyLecturer** (ModuleID, LecturerID)
	LECTURERS
17	**LecturerDetails** (LecturerID, LecturerName, PrimaryDiscipline, GCC_StartDate)
18	**LecturerEvaluation** (LecturerID, ModuleID, StudentID, Q1Score, Q2Score, Q3Score, Q4Score)

2.2: Table Details

Table 1: **GreendaleDPDetails**

Attributes	Data Type	Constraints
ProgrammeID	int	
ProgrammeName	varchar(100)	
PartnerUniversities	varchar(100)	
GradingSystem	varchar(100)	
ProgrammeDuration	int	int in terms of number of Months
SchoolType	varchar(100)	

Table 2: **GreendaleDPStatus**

Attributes	Data Type	Constraints
ProgrammeID	int	
Programme_Status_Date	date	
ProgrammeStatus	varchar(100)	
StatusDescription	varchar(100)	

Table 3: **AGofProgrammes**

Attributes	Data Type	Constraints
ProgrammeID	int	
AGID	int	

Table 4: **ModulesinProgramme**

Attributes	Data Type	Constraints
ProgrammeID	int	
ModuleID	int	
ModuleName	varchar(100)	

Table 5: **StudentDetails**

Attributes	Data Type	Constraints
StudentID	char(8)	
FirstName	varchar(100)	
LastName	varchar(100)	
Nationality	varchar(100)	
HighestQualifications	varchar(100)	
PreviousSchool	varchar(100)	
Gender	varchar(10)	

Table 6: **StudentStatus**

Attributes	Data Type	Constraints
StudentID	char(8)	
StatusID	int(20)	
Student_Status_Date	date	
StatusDescription	varchar(100)	

Table 7: **StudentContactDetails**

Attributes	Data Type	Constraints
StudentID	char(8)	
Email	varchar(100)	
Phone	int(20)	
Mobile	int(20)	

Table 8: **StudentProgramme**

Attributes	Data Type	Constraints
StudentID	char(8)	
ProgrammeID	int	

Table 9: **AGDetails**

Attributes	Data Type	Constraints
AGID	int	
AG_StartDate	date	

Table 10: **StudentsinAG**

Attributes	Data Type	Constraints
AGID	int	
StudentID	char(8)	

Table 11: **AGStatus**

Attributes	Data Type	Constraints
AGID	int	
AG_Status_Date	date	
StudentID	char(8)	
StatusDescription	varchar(100)	'Active', 'Inactive' or 'Cancelled'

Table 12: **ModuleDetails**

Attributes	Data Type	Constraints
ModuleID	int	
ProgrammeID	int	
ModuleCode	varchar(20)	
ModuleTitle	varchar(20)	

Table 13: **ModuleStatus**

Attributes	Data Type	Constraints
ModuleID	int	
Module_Date	date	
StudentID	char(8)	
StatusDescription	varchar(100)	'Active', 'Inactive' or 'Completed'

Table 14: **StudentGrade**

Attributes	Data Type	Constraints
ModuleID	int	
StudentID	char(8)	
Grade	varchar(20)	
PassorFail	varchar(4)	‘Pass’ or ‘Fail’

Table 15: **StudentModuleAttendance**

Attributes	Data Type	Constraints
ModuleID	int	
StudentID	char(8)	
Location	varchar(20)	
Attendance	boolean	

Table 16: **ModulesbyLecturer**

Attributes	Data Type	Constraints
ModuleID	int	
LecturerID	int	

Table 17: **LecturerDetails**

Attributes	Data Type	Constraints
LecturerID	int	
LecturerName	varchar(100)	
PrimaryDiscipline	varchar(100)	
GCC_StartDate	date	

Table 18: **LecturerEvaluation**

Attributes	Data Type	Constraints
ModuleID	int	
LecturerID	int	
StudentID	char(8)	
Q1Score	int(5)	int 1 to 5
Q2Score	int(5)	int 1 to 5
Q3Score	int(5)	int 1 to 5
Q4Score	int(5)	int 1 to 5

Part 3

Implementation: A list of SQL statements to create the database and tables.

Details: The statements you would use in your MySQL client to create the database and tables described in your logical design.

3.1 : Creating and using Database

```
CREATE DATABASE GreenDale;

USE GreenDale;
```

3.2 Creating Tables

Table 1: **GreendaleDPDetails**

```
CREATE TABLE `GreendaleDPDetails` (
  `ProgrammeID` int,
  `ProgrammeName` varchar(100),
  `PartnerUniversities` varchar(100),
  `GradingSystem` varchar(100),
  `ProgrammeDuration` int,
  `SchoolType` varchar(100),
  PRIMARY KEY (`ProgrammeID`)
);
```

Table 2: **GreendaleDPStatus**

```
CREATE TABLE `GreendaleDPStatus` (
  `ProgrammeID` int,
  `Programme_Status_Date` date,
  `ProgrammeStatus` varchar(100),
  `StatusDescription` varchar(100),
   PRIMARY KEY (`ProgrammeID`, `Programme_Status_Date`, `ProgrammeStatus`),
         FOREIGN KEY (ProgrammeID)
         REFERENCES GreendaleDPDetails(ProgrammeID));
```

Table 3: **AGofProgrammes**

```
CREATE TABLE `AGofProgrammes` (
  `ProgrammeID` int,
  `AGID` int,
  PRIMARY KEY (`ProgrammeID`,`AGID`),
         FOREIGN KEY (ProgrammeID)
         REFERENCES GreendaleDPDetails(ProgrammeID),
         FOREIGN KEY (AGID)
         REFERENCES AGDetails (AGID));
```

Table 4: **ModulesinProgramme**

```
CREATE TABLE `ModulesinProgramme` (
  `ProgrammeID` int,
  `ModuleID` int,
  `ModuleName` varchar(100),
  PRIMARY KEY (`ProgrammeID`, `ModuleID`),
         FOREIGN KEY (ProgrammeID)
         REFERENCES FROM GreendaleDPDetails (ProgrammeID),
         FOREIGN KEY (ModuleID)
         REFERENCES FROM ModuleDetails (ModuleID));
```

Table 5: **StudentDetails**

```
CREATE TABLE `StudentDetails` (
  `StudentID` char(8),
  `FirstName` varchar(100),
  `LastName` varchar(100),
  `Nationality ` varchar(50),
  `HighestQualifications` varchar(100),
  `PreviousSchool` varchar(100),
  `Gender` varchar(10),
  PRIMARY KEY (`StudentID`)
);
```

Table 6: **StudentStatus**

```
CREATE TABLE `StudentStatus` (
  `StudentID` char(8),
  `StatusID` int (20),
  `Student_Status_Date` date,
  `StatusDescription` varchar(100),
  PRIMARY KEY (`StudentID`, `StatusID`, `Student_Status_Date`),
        FOREIGN KEY (StudentID)
        REFERENCES StudentDetails (StudentID)
);
```

Table 7: **StudentContactDetails**

```
CREATE TABLE `StudentContactDetails` (
  `StudentID` char(8),
  `Email` varchar(100),
  `Phone` int(20),
  `Mobile` int(20),
  PRIMARY KEY (`StudentID`),
        FOREIGN KEY (StudentID)
        REFERENCES StudentDetails (StudentID));
```

Table 8: **StudentProgramme**

```
CREATE TABLE `StudentPogramme` (
  `StudentID` char(8),
  `ProgrammeID` int,
  PRIMARY KEY (`StudentID`, `ProgrammeID`),
        FOREIGN KEY (StudentID)
        REFERENCES StudentDetails (StudentID),
        FOREIGN KEY (ProgrammeID)
        REFERENCES GreendaleDPDetails (ProgrammeID)
);
```

Table 9: **AGDetails**

```
CREATE TABLE `AGDetails` (
  `AGID` int,
  `AG_StartDate` date,
  PRIMARY KEY (`AGID`),
);
```

Table 10: **StudentsinAG**

```
CREATE TABLE `StudentsinAG` (
  `AGID` int,
  `StudentID` char(8),
  PRIMARY KEY (`AGID`, `StudentID`),
        FOREIGN KEY (AGID)
        REFERENCES FROM AGDetails (AGID),
        FOREIGN KEY (StudentID)
        REFERENCES FROM StudentDetails (StudentID)
);
```

Table 11: **AGStatus**

```
CREATE TABLE `AGStatus` (
  `AGID` int,
  `AG_Status_Date` date,
  `StudentID` char(8),
  `Status Description` varchar(100),
  PRIMARY KEY (`AGID`,`AG_Status_Date`,`StudentID`),
        FOREIGN KEY (AGID)
        REFERENCES FROM AGDetails (AGID),
        FOREIGN KEY (StudentID)
        REFERENCES FROM StudentDetails (StudentID)
);
```

Table 12: **ModuleDetails**

```
CREATE TABLE `ModuleDetails` (
  `ModuleID` int,
  `ProgrammeID` int,
  `ModuleCode` varchar(20),
  `ModuleTitle` varchar(20),
  PRIMARY KEY (`ModuleID`, `ProgrammeID`),
        FOREIGN KEY (ProgrammeID)
        REFERENCES FROM GreendaleDPDetails (ProgrammeID)
);
```

Table 13: **ModuleStatus**

```
CREATE TABLE `ModuleStatus`(
  `ModuleID` int,
  `Module_Date` date,
  `StudentID` char(8),
  `StatusDescription` varchar(100),
  PRIMARY KEY (`ModuleID`, `Module_Date`,`StudentID`),
        FOREIGN KEY (ModuleID)
        REFERENCES FROM ModuleDetails (ModuleID),
        FOREIGN KEY (StudentID)
        REFERENCES FROM StudentDetails (StudentID)
);
```

Table 14: **StudentGrade**

```
CREATE TABLE `StudentGrade` (
  `ModuleID` int,
  `StudentID` char(8),
  `Grade` varchar(20),
  `PassorFail` varchar(20),
  `Term` int(5),
   PRIMARY KEY (`ModuleID`),
        FOREIGN KEY (ModuleID)
        REFERENCES ModuleDetails (ModuleID),
        FOREIGN KEY (StudentID)
        REFERENCES StudentDetails (StudentID)
);
```

Table 15: **StudentModuleAttendance**

```
CREATE TABLE `StudentModuleAttendance` (
  `ModuleID` int,
  `StudentID` char(8),
  `Location` varchar(20),
  `Attendance` boolean,
  `Term` int(5),
   PRIMARY KEY (`ModuleID`,`StudentID`,`Location`,`Term`),
          FOREIGN KEY (ModuleID)
          REFERENCES FROM ModuleDetails (ModuleID),
          FOREIGN KEY (StudentID)
          REFERENCES FROM StudentDetails (StudentID)
);
```

Table 16: **ModulesbyLecturer**

```
CREATE TABLE `ModulesbyLecturer` (
  `ModuleID` int,
  `LecturerID` int,
  PRIMARY KEY (`ModuleID`, `LecturerID`),
          FOREIGN KEY (ModuleID)
          REFERENCES FROM ModuleDetails (ModuleID),
          FOREIGN KEY (LecturerID)
          REFERENCES FROM LecturerDetails (LecturerID)
);
```

Table 17: **LecturerDetails**

```
CREATE TABLE `LecturerDetails` (
  `LecturerID` int,
  `Lecturer Name` varchar(100),
  `Primary Discipline` varchar(100),
  `GCC_StartDate` date,
  PRIMARY KEY (`LecturerID`)
);
```

Table 18: **LecturerEvaluation**

```
CREATE TABLE `LecturerEvaluation` (
  `ModuleID` int,
  `LecturerID` int,
  `StudentID` char(8),
  `Q1Score` int(5),
  `Q2Score` int(5),
  `Q3Score` int(5),
  `Q4Score` int(5),
   PRIMARY KEY (`ModuleID`, `LecturerID`, `StudentID`),
         FOREIGN KEY (ModuleID)
         REFERENCES FROM ModuleDetails (ModuleID),
         FOREIGN KEY (LecturerID)
         REFERENCES FROM LecturerDetails (LecturerID),
         FOREIGN KEY (StudentID)
         REFERENCES FROM StudentDetails (StudentID));
```

Part 4

Queries: Your SQL queries to the questions posed at the end of the scenario.

Details: For each question listed at the end of the scenario, provide the SQL query/queries that would extract the data from your table structure to answer that question.

(a) Identifying students who aren't doing well

Q1:

Which students in an AG of a programme have failed a course or more than one course? Students who have failed are more likely to drop out of the programme, or are more likely to take a longer time to graduate. For each AG that is currently active, the students who are failing at least one module need to be highlighted to the Academic Success Team for them to intervene.

```
#1
#Which students in an AG of a programme have failed a course or more than one course? Students
#who have failed are more likely to drop out of the programme, or are more likely to take a longer time
#to graduate. For each AG that is currently active, the students who are failing at least one module need
#to be highlighted to the Academic Success Team for them to intervene.
CREATE OR REPLACE VIEW Students_fail_module AS(
SELECT
   AGStatus.AGID, StudentDetails.Firstname, StudentDetails.Lastname,
   StudentDetails.StudentID, AGofProgrammes.ProgrammeID
FROM
   StudentDetails
      LEFT JOIN
   StudentGrade ON StudentDetails.StudentID = StudentGrade.StudentID
      LEFT JOIN
   StudentStatus ON StudentDetails.StudentID = StudentStatus.StudentID
      LEFT JOIN
   ModuleStatus ON StudentDetails.StudentID = ModuleStatus.StudentID
      LEFT JOIN
   AGStatus ON StudentDetails.StudentID = AGStatus.StudentID
      LEFT JOIN
   AGofProgrammes ON AGStatus.AGID =    AGofProgrammes.AGID
WHERE
      ModuleStatus.StatusDescription = 'completed'
      AND StudentGrade.PassorFail = 'Fail'
      AND StudentStatus.StatusDescription = 'active'
      AND AGStatus.StatusDescription = 'active'
      AND StudentStatus.Student_Status_Date = (SELECT MAX(StudentStatus.Student_Status_Date) FROM StudentStatus)
      AND AGStatus.AG_Status_Date = (SELECT MAX(AGStatus.AG_Status_Date) FROM AGStatus)
GROUP BY AGID , Firstname , Lastname , StudentID, ProgrammeID
ORDER BY StudentID);
```

Q2:

Which students have failed two courses (or more) in two consecutive terms? These students are at much higher risk.

```
#2
#Finding the ModuleID, Module date of completion and
# studentID of active students who failed a completed module
CREATE OR REPLACE VIEW Students_fail_1 AS (
SELECT
   StudentGrade.StudentID , StudentGrade.ModuleID, ModuleStatus.Module_Date
FROM
   StudentGrade
      LEFT JOIN
   StudentStatus ON StudentGrade.StudentID = StudentStatus.StudentID
      LEFT JOIN
   ModuleStatus ON StudentGrade.ModuleID = ModuleStatus.ModuleID
WHERE
   ModuleStatus.StatusDescription = 'completed'
      AND StudentGrade.PassorFail = 'Fail'
      AND StudentStatus.StatusDescription = 'active'
GROUP BY StudentID, ModuleID, Module_Date
ORDER BY Module_Date);

#Replicating a similar table as Students_fail_1
CREATE OR REPLACE VIEW Students_fail_2 AS (
SELECT
   StudentGrade.StudentID , StudentGrade.ModuleID, ModuleStatus.Module_Date
FROM
   StudentGrade
      LEFT JOIN
   StudentStatus ON StudentGrade.StudentID = StudentStatus.StudentID
      LEFT JOIN
   ModuleStatus ON StudentGrade.ModuleID = ModuleStatus.ModuleID
WHERE
   ModuleStatus.StatusDescription = 'completed'
      AND StudentGrade.PassorFail = 'Fail'
      AND StudentStatus.StatusDescription = 'active'
GROUP BY StudentID, ModuleID, Module_Date
ORDER BY Module_Date);
```

```
#Filtering to produce students who failed modules over different terms
CREATE OR REPLACE VIEW Students_fail_consecutive AS
    (SELECT
        Students_fail_1.StudentID,
        Students_fail_1.ModuleID AS FirstModuleID,
        Students_fail_2.ModuleID AS SecondModuleID,
        Students_fail_1.Module_Date AS FirstModuleFinishDate,
        Students_fail_2.Module_Date AS SecondModuleFinishDate
    FROM
        Students_fail_1,
        Students_fail_2
    WHERE
        Students_fail_1.StudentID = Students_fail_2.StudentID
            AND (SecondModuleFinishDate > FirstModuleFinishDate
            OR (SecondModuleFinishDate = FirstModuleFinishDate
            AND FirstModuleID != SecondModuleID)));


#FINAL ANSWER
#Filtering to produce students who failed modules over consecutive terms
CREATE OR REPLACE VIEW Students_fail_consecutive_modules AS
    (SELECT DISTINCT
        (Students_fail_consecutive.StudentID),
        StudentDetails.FirstName,
        StudentDetails.LastName
    FROM
        StudentDetails,
        Students_fail_consecutive
    WHERE
        Students_fail_consecutive.SecondModuleFinishDate <= ADDDATE(
        Students_fail_consecutive.FirstModuleFinishDate, INTERVAL 6 MONTH)
            AND Students_fail_consecutive.StudentID = StudentDetails.StudentID
    GROUP BY StudentID , FirstName , LastName
    ORDER BY StudentID);
```

(b) Identifying modules that aren't doing well

Q3:

For each module, therefore, what is its passing rate, and is its passing rate improving over time and across different lecturers?

```
#3
#For each module, therefore, what is its passing rate, and is its passing rate improving over time and
#across different lecturers?

CREATE OR REPLACE VIEW Completed_modules AS
  (SELECT
     ModuleID, Module_Date
  FROM
     ModuleStatus
  WHERE
     StatusDescription = 'completed');


CREATE OR REPLACE VIEW Module_passes AS
  (SELECT
     Completed_modules.ModuleID,
     COUNT(StudentGrade.StudentID) AS No_of_passes,
     COUNT(StudentGrade.PassorFail) AS Total_no_of_students,
     LecturerDetails.LecturerID,
     LecturerDetails.LecturerName,
     Completed_modules.Module_Date
  FROM
     Completed_modules
        LEFT JOIN
     StudentGrade ON Completed_modules.ModuleID = StudentGrade.ModuleID
        LEFT JOIN
     ModulesbyLecturer ON ModulesbyLecturer.ModuleID = Completed_modules.ModuleID
        LEFT JOIN
     LecturerDetails ON LecturerDetails.LecturerID = ModulesbyLecturer.LecturerID
  WHERE
     PassorFail = 'Pass'
  GROUP BY Module_ID , Module_Date , LecturerID , LecturerName);


#Module pass rate
CREATE OR REPLACE VIEW Module_pass_rate AS
  (SELECT
     ModuleID,
     No_of_passes / Total_no_of_students * 100 AS Pass_rate
  FROM
     Module_passes
  GROUP BY ModuleID
  ORDER BY ModuleID);

#Module passrate overtime with different lecturers
CREATE OR REPLACE VIEW Module_pass_rate_overtime_withdiffLecturers AS
  (SELECT
     Module_passes.ModuleID,
     Module_pass_rate.Pass_rate,
     Module_passes.Module_Date,
     Module_passes.LecturerID,
     Module_passes.LecturerName
  FROM
     Module_passes
     LEFT JOIN
     Module_pass_rate ON Module_passes.ModuleID = Module_pass_rate.ModuleID
  ORDER BY ModuleID , Pass_rate);
```

Q4:

Which lecturers are the best to assign to a particular module and which lecturers aren't?

```
#4 Which lecturers are the best to assign to a particular module and which lecturers aren't?
#First way is : Judging by Pass rate (with reference to Q3) to find the best lecturer to be assigned
CREATE OR REPLACE VIEW Lecturer_best_passing AS
  (SELECT
    Module_passes.ModuleID,
    Module_pass_rate.Pass_rate,
    Module_passes.Module_Date,
    Module_passes.LecturerID,
    Module_passes.LecturerName
  FROM
    Module_passes
    LEFT JOIN
    Module_pass_rate ON Module_passes.ModuleID = Module_pass_rate.ModuleID
  ORDER BY Pass_rate DESC);

#Second way is : Judging by Evaluation to find the best lecturer to be assigned
CREATE OR REPLACE VIEW Lecturer_best_Evaluation AS
  (SELECT
    Module_passes.Module ID,
    Module_passes.LecturerName,
    Module_passes.LecturerID,
    (LecturerEvaluation.Q1Score
    + LecturerEvaluation.Q2Score
    + LecturerEvaluation.Q3Score
    + LecturerEvaluation.Q4Score) / 4 AS EvaluationScore,
    Module_passes.Module_Date
  FROM
    Module_passes
    LEFT JOIN
    LecturerEvaluation ON Module_passes.ModuleID = LecturerEvaluation.ModuleID
  GROUP BY ModuleID , LecturerName , LecturerID , Module_Date
  ORDER BY EvaluationScore DESC);
```

To determine if a lecturer is the best to be assigned to a particular module, we can judge this on 2 points:

1. The lecturer's passing rate (sorted in Descending order)
2. The lecturer's evaluation score (sorted in Descending order)

Q4 part 2

```
#First way is : Judging by Pass rate (with reference to Q3) to find the worst lecturer to be assigned
CREATE OR REPLACE VIEW Lecturer_worst_passing AS
   (SELECT
      Module_passes.ModuleID,
      Module_pass_rate.Pass_rate,
      Module_passes.Module_Date,
      Module_passes.LecturerID,
      Module_passes.LecturerName
   FROM
      Module_passes
      LEFT JOIN
      Module_pass_rate ON Module_passes.ModuleID = Module_pass_rate.ModuleID
   ORDER BY Pass_rate ASC);

#Second way is : Judging by Evaluation to find the worst lecturer to be assigned
CREATE OR REPLACE VIEW Lecturer_best_Evaluation AS
   (SELECT
      Module_passes.Module ID,
      Module_passes.LecturerName,
      Module_passes.LecturerID,
      (LecturerEvaluation.Q1Score
      + LecturerEvaluation.Q2Score
      + LecturerEvaluation.Q3Score
      + LecturerEvaluation.Q4Score) / 4 AS EvaluationScore,
      Module_passes.Module_Date
   FROM
      Module_passes
      LEFT JOIN
      LecturerEvaluation ON Module_passes.ModuleID = LecturerEvaluation.ModuleID
   GROUP BY ModuleID , LecturerName , LecturerID , Module_Date
   ORDER BY EvaluationScore ASC);
```

Similarly, to determine if a lecturer is the worst to be assigned to a particular module, we can judge this on the same 2 points:

1. The lecturer's passing rate (sorted in Ascending order)
2. The lecturer's evaluation score (sorted in Ascending order)

(c) Programmes that are doing well

Q5:

Which programmes and partners are the most successful at attracting students to Greendale?
Which programmes and partners are the least attractive?

```
# 5
# Which programmes and partners are the most successful at attracting students to Greendale?
# Which programmes and partners are the least attractive?

#Most successful means programme that attracts the most students to apply
CREATE OR REPLACE VIEW Most_Successful_Prog AS
   (SELECT
      GreendaleDPDetails.ProgrammeID,
      GreendaleDPDetails.ProgrammeName,
      GreendaleDPDetails.PartnerUniversities,
      COUNT(StudentStatus.StudentID) AS No_of_students
   FROM
      GreendaleDPDetails,
      StudentStatus
   WHERE
     StudentStatus.StatusID = (SELECT
           StatusID
        FROM
           StudentStatus
        WHERE
           StatusDescription = 'The application has been submitted')
   GROUP BY ProgrammeID , ProgrammeName , PartnerUniversities
   ORDER BY No_of_students DESC);

#Least successful means programme that attracts the least students to apply
CREATE OR REPLACE VIEW Least_Successful_Prog AS
   (SELECT
      GreendaleDPDetails.ProgrammeID,
      GreendaleDPDetails.ProgrammeName,
      GreendaleDPDetails.PartnerUniversities,
      COUNT(StudentStatus.StudentID) AS No_of_students
   FROM
      GreendaleDPDetails,
      StudentStatus
   WHERE
     StudentStatus.StatusID = (SELECT
           StatusID
        FROM
           StudentStatus
        WHERE
           StatusDescription = 'The application has been submitted')
   GROUP BY ProgrammeID , ProgrammeName , PartnerUniversities
   ORDER BY No_of_students ASC);
```

Q6:

Which partners and programmes have the highest graduation rates? Which partners and programmes have the highest graduation rates within the allotted time for that programme?

```
# 6
# Which partners and programmes have the highest graduation rates? Which partners and
# programmes have the highest graduation rates within the allotted time for that programme?

CREATE OR REPLACE VIEW Grad_rates AS
  (SELECT
      GreendaleDPDetails.ProgrammeID,
      GreendaleDPDetails.ProgrammeName,
      GreendaleDPDetails.PartnerUniversities,
      (select COUNT(StudentStatus.StudentID)
      FROM
      StudentStatus
      WHERE
      StudentStatus.StatusDescription = 'withdraw'
      OR
      StudentStatus.StatusDescription = 'graduated'
      OR
      StudentStatus.StatusDescription = 'transfer'
      OR
      StudentStatus.StatusDescription = 'waiting' ) as No_of_original_students,
      (select COUNT(StudentStatus.StudentID)
      FROM
      StudentStatus
      WHERE
      StudentStatus.StatusDescription = 'graduated') as No_of_graduated
      FROM
      GreendaleDPDetails
      GROUP BY
      ProgrammeID, ProgrammeName, PartnerUniversities
      ORDER BY
      No_of_graduated DESC);

#The partners and programmes with the highest graduation rates.
CREATE OR REPLACE VIEW Highest_Grad_Rate AS
  (SELECT
      ProgrammeID,
      ProgrammeName,
      PartnerUniversities,
      No_of_graduated / No_of_original_students * 100 as GraduationRate
      FROM
      Grad_rates
      GROUP BY ProgrammeID, ProgrammeName, PartnerUniversities
      ORDER BY GraduationRate DESC);
```

```
#Student acceptance date
CREATE OR REPLACE VIEW Student_acceptance_date AS
  (SELECT
    StudentID,
    Student_Status_Date AS AcceptanceDate
  FROM
    StudentStatus
  WHERE
    StatusDescription = 'The student has accepted the offer.');


#Student graduation date
CREATE OR REPLACE VIEW Student_graduation_date AS
  (SELECT
  StudentID,
  Student_Status_Date AS GraduationDate
  FROM
  StudentStatus
  WHERE
  StatusDescription = 'graduated');

#Student graduation date
CREATE OR REPLACE VIEW TimeTakenToGraduate AS
  (SELECT
  Student_graduation_date.StudentID,
  timestampdiff(Month, Student_acceptance_date.AcceptanceDate, Student_graduation_date.GraduationDate) AS
  time_before_graduation
  FROM
  Student_acceptance_date,
  Student_graduation_date
  WHERE
  Student_graduation_date.StudentID = Student_acceptance_date.StudentID);


CREATE OR REPLACE VIEW Students_with_allottedGraduation AS
  (SELECT
    TimeTakenToGraduate.StudentID,
    StudentDetails.StudentName,
    GreendaleDPDetails.ProgrammeID,
    GreendaleDPDetails.ProgrammeName,
    GreendaleDPDetails.PartnerUniversities
  FROM
    TimeTakenToGraduate
      LEFT JOIN
    StudentDetails ON TimeTakenToGraduate.StudentID = StudentDetails.StudentID
      LEFT JOIN
    StudentProgramme ON TimeTakenToGraduate.StudentID = StudentProgramme.StudentID
      LEFT JOIN
    GreendaleDPDetails ON StudentProgramme.ProgrammeID = GreendaleDPDetails.ProgrammeID
  WHERE
    TimeTakeToGraduate.time_before_graduation = GreendaleDPDetails.ProgrammeDuration);

#Partners and programmes with the highest graduation rates within the allotted time
CREATE OR REPLACE VIEW graduation_rate AS
  (SELECT
    Students_with_allottedGraduation.ProgrammeID,
    Students_with_allottedGraduation.ProgrammeName,
    Students_with_allottedGraduation.PartnerUniversities,
    (COUNT(Students_with_allocatedGraduation.StudentID) /
    COUNT(Student_acceptance_date.StudentID)) * 100 AS Graduation_rate
  FROM
    Student_acceptance_date,
    Students_with_allottedGraduation
  GROUP BY ProgrammeID , ProgrammeName , PartnerUniversities
  ORDER BY Graduation_rate);
```