

**Advanced Databases (ADB)**

October 2021

## SCHOOL OF INFOCOMM TECHNOLOGY

Diploma in Information Technology

**ADB Assignment 2**

**(40% of ADB Module)**

**24 January – 04 February 2022 (2 Weeks)**

**Deadline: 07 February 2022, 9:00 am**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ADB Class group** | T02 | | | |
| **Team Number** | T2 | | **Team  Grade** |  |
| **Tutor** | Mr Tan Choon Peng | | | |
| **Members** | **Student No.** | **Student Name** | **Individual Grade** | |
| S10208266K | Muhd Dandarawie Bin Rohani |  | |
| S10205104F | James Fong Ka Chun |  | |
| S10202961H | Chen Han |  | |
| S10205505J | Ellappan Keerthivasan |  | |
|  |  |  | |

Table of Contents

[SCHOOL OF INFOCOMM TECHNOLOGY - 1 -](#_Toc95100629)

[1) Team - 3 -](#_Toc95100630)

[(i) Final Business Domain Write-Up - 3 -](#_Toc95100631)

[(i.i) Muhd Dandarawie Bin Rohani - 3 -](#_Toc95100632)

[(i.ii) James Fong Ka Chun - 5 -](#_Toc95100633)

[(i.iii) Chen Han - 6 -](#_Toc95100634)

[(i.iv) Keerthivasan - 7 -](#_Toc95100635)

[(ii) Final ER Model And The List Of Mapped Relations - 9 -](#_Toc95100636)

[(ii.ii) List of mapped relations - 10 -](#_Toc95100637)

[(iii) How Your Cloud Database Design Meets The Requirements Of The Entire System - 12 -](#_Toc95100638)

[(iv) Evidence (screenshots) Of Implementation Of Cloud Storages (In Microsoft Azure Environment) - 14 -](#_Toc95100639)

[2) Individual - 18 -](#_Toc95100640)

[(i) Muhd Dandarawie Bin Rohani - 18 -](#_Toc95100641)

[(i.i) Denormalisation Application - 18 -](#_Toc95100642)

[(i.ii) Data Placement Strategy - 19 -](#_Toc95100643)

[(i.iii) Cloud Storage Strategies - 19 -](#_Toc95100644)

[(i.iv) Design/Schema on Cloud Storage Used - 21 -](#_Toc95100645)

[(i.v) Implementation of business queries - 22 -](#_Toc95100646)

[(ii) James - 28 -](#_Toc95100647)

[(ii.i) Denormalization Application - 28 -](#_Toc95100648)

[(ii.ii) Data Placement Strategy - 28 -](#_Toc95100649)

[(ii.iii) Cloud Storage Strategies - 29 -](#_Toc95100650)

[(ii.iv) Design/Schema on Cloud Storage Used - 30 -](#_Toc95100651)

[(ii.v) Implementation of business queries - 32 -](#_Toc95100653)

[(iii) Chen Han - 34 -](#_Toc95100654)

[(iii.i) Denormalization Application - 34 -](#_Toc95100648)

[(iii.ii) Data Placement Strategy - 35 -](#_Toc95100655)

[(iii.iii) Cloud Storage Strategies - 37 -](#_Toc95100656)

[(iii.iv) Design/Schema on Cloud Storage Used - 38 -](#_Toc95100657)

[(iii.v) Implementation of business queries - 40 -](#_Toc95100658)

[(iv) Keerthivasan - 48 -](#_Toc95100659)

[(iv.i) Implementation of Denormalization in Database - 48 -](#_Toc95100660)

[(iv.ii) Data Placement Strategy - 49 -](#_Toc95100661)

[(iv.iii) Cloud Storage Strategies - 50 -](#_Toc95100662)

[(iv.iii.i) SQL - 50 -](#_Toc95100663)

[(iv.iii.ii) NoSQL - 51 -](#_Toc95100664)

[(iv.iv) Design/Schema on Cloud Storage Used - 51 -](#_Toc95100665)

[(iv.v) Implementation of business queries after denormalization & fragmentation - 53 -](#_Toc95100666)

# 1) Team

## (i) Final Business Domain Write-Up

### (i.i) Muhd Dandarawie Bin Rohani

##### (i.i.i) Core Function: Gradings

1. Students graded according to score average from tests and assignments
2. Tutors feedback students on their assignments

Marking of assessments created by tutors

1. Tutors add announcements to modules for students

##### (i.i.ii) Description of Function:

1. Grades for students are separated into two categories, either current grade or cumulative grade, where current grade is the grade for their current semester and cumulative grade is calculated based on all semester grades. Current grades of the students are also stored in the StudentModule relationship.
2. The assignments that students have submitted can have written feedback from the tutors after grading.
3. Tutors have the ability to grade tests and assignments submitted by students
4. Tutors add announcements to module for students to view

##### (i.i.iii) Query:

1. Mr Tan would like to know the students who have failed the advanced databases module.

SELECT Student.StudentID AS ‘StudentID’,Student.StudentName AS ‘Name of Student’

FROM Student

INNER JOIN StudentModule

ON Student.StudentID = StudentModule.StudentID

INNER JOIN Module

ON StudentModule.ModuleID = Module.ModuleID

WHERE StudentModule.ModuleGrade = ‘0.0’

AND

Module.Name = ‘Advanced Databases’

### (i.ii) James Fong Ka Chun

##### (i.ii.i) Core Function: Assignment and Test Submission

Assignments are projects that students accomplish and submit the required files for.

Tests are graded sets of open-ended questions on the client for students to complete and submit.

Tutors create assignments and tests for students to complete.

Tutors can attach files and documents to these assignments and tests.

Students can submit these assignments and tests.

Students can also attach files and documents to their submissions.

##### (i.ii.ii) Description of Function:

This function can be split into two parts: Assignment and Test.

For Assignment, tutors can create an assignment for students to complete, which will affect the student’s grade for the module the assignment was created for. Tutors can assign and change the deadline and the percentage of marks the assignment contributes to its module. They can state the assignment requirements on the site itself or attach the assignment requirements to a separate file. Other files can also be added and removed by users to the assignment as necessary.

Students can make submissions for the created assignments, they can either make a text submission using the site’s built-in textbox or attach files to the submission. The date and time of submission is recorded as well. Tutors can score the submission, which is converted into a grade, and give feedback for the submission.

For Test, similarly to Assignment, tutors can create a test, give it details, assign a time limit for completion, and a percentile weightage for the module it is created for. Each test is made up of many questions with a predetermined answer and score. Files can also be attached to the question if needed.

Students can submit their tests either as a text submission or via attached files, and the date and time of submission will be recorded as well. Each submission can also be marked by tutors and have labels to signify if they are submissions for retests or not.

The answers that the students enter for each test submission are stored in a JSON file.

File or document submissions for both Assignment and Test are compressed into a single .zip file before being added as a value in a record to avoid redundancy.

##### (i.ii.iii) Query:

A tutor wants to view all the students that provided a submission for Test 2 from the ‘Advanced Databases’ module.

SELECT StudentName AS “Students with test submission”

FROM Student s

INNER JOIN TestSubmission ts

ON s.StudentID = ts.StudentID

INNER JOIN Test t

ON t.TestID = ts.TestID

INNER JOIN Module m

ON m.ModuleID = t.ModuleID

WHERE m.ModuleName = “Advanced Databases” AND t.TestName = “Test 2”

### (i.iii) Chen Han

##### (i.i.i) Core Function: Learning Materials

1. Tutor uploads learning materials
2. Student download learning materials
3. Tutor edits/deletes/release hidden learning materials

##### (i.i.ii) Description of Function:

* Tutor uploads learning materials such as Weekly lectures, practicals, common test and revision paper. When uploading a material, the tuor has to specify the type of the learning materials (e.g. Lecture Notes, Practicals or Common Test revision), this is to categorise the learning material and easy for the students to find and download.
* The tutor will also need to choose a name for this material and set a shown\_date which is the datetime to be shown to the students, so that the following week’s material will be unavailable for students to view. If the tutor wishes, he/she can also set the learning material available later even if the shown\_date is not up.
* Students can download the learning material by clicking on the file when they are enrolled in a module. The student has no right to change or delete the learning material on the page.
* Tutor is able to change the Learning Material after it has been uploaded, such as Material description and shown\_date, the tutor is also able to set those hidden learning materials to be shown to students.

##### (i.i.iii) Query:

1. A student would like to find all the learning materials that are “Weekly Tutorial”.

SELECT lm.\*, lmt.MaterialType

FROM LearningMaterial lm

INNER JOIN LearningMaterialType lmt

ON lm.TypeID = lmt.TypeID

WHERE lmt.MaterialType = ‘Weekly Tutorial’

AND lm.IsShown = 1

1. Tutor Ben would like to find out how many learning materials are uploaded by him in Module "Mobile Application Development".

SELECT COUNT(\*) AS "No. of learning materials Ben uploaded in MAD"

FROM LearningMaterial lm

INNER JOIN Module m

ON m.ModuleID = lm.ModuleID

INNER JOIN Tutor t

ON t.TutorID = lm.TutorID

WHERE t.TutorName = 'Ben' AND lm.ModuleID IN

(SELECT ModuleID FROM Module

WHERE ModuleName = 'MobileApplicationDevelopment')

### (i.iv) Keerthivasan

##### (i.iv.i) Core Function: Online Quiz

Tutor is able to create quizzes. The tutor will be able to create various different questions and categorize them while giving each question scores. The tutor will also be able to give a total percentage for the quiz.

Tutor is able to manage these quizzes as well. He will be able to change the time the quiz is released and be able to modify the questions and the scores of the questions BEFORE any student has taken it.

Tutor is able to validate/invalidate quizzes. For example, if a student is found cheating, they can change the quiz status to invalid for them and his marks will be invalidated and not counted towards the weightage. The tutor will also be able to allow for re-tries of the quiz.

Each student is able to take the quiz available to them for each of their modules.

Each student is available to check the results of their quizzes after completion.

##### (i.iv.ii) Description of Function:

First the tutor table will be required as the tutor will be the one creating the quizzes, overseeing them, checking results and possibly re-grading them.

The student table will be required as the students will be the ones taking the quiz itself, hence information about the student will be needed to record down who took the quiz.

The first main table for this function will be the Quiz Table, this table will contain most of the metadata of the quiz itself. For example, this table should include attributes such as

QuizID to identify the quiz, HostID to identify who created the quiz, Title to show the title of the quiz, Summary to summarize and mention key highlights of the quiz, Percentage to show the total weightage of the quiz, Published to see if the quiz is available or not, CreatedAt to show the datetime when the quiz was created, StartsAt to store the time the quiz open up and is available, EndsAt to store the datetime when the quiz closes.

The next table required will be QuizQuestion, this table will contain the questions related to the quiz. It should have the attributes as QuestionID to identify the question, QuizID to identify the parent quiz, CategoryID to identify the category of the question, Type to find out the type of questions mcq or open-ended, Level to find the difficulty of the quiz, Score to store the score of each question and Content to store the question itself.

Next table is the QuestionCategory table, this table will contain the categories for the questions. It should have the attributes CategoryID to have a unique identifier for each question and Category to store the categories of the question and QuestionID to identify the question.

Moving on to the next table is the QuestionAnswer table. This table will store the answers of the questions. It should contain the QuestionID to identify the question, QuizID to identify the parent quiz, ChoiceID for a unique identifier for choice, CorrectChoice to hold the correct choice and Content to store the answer itself.

The next table is the StudentQuiz table this table will contain the data of the Students quiz. It should contain the StudentID a unique identifier for the student, QuizID to identify the quiz, SQuizID a unique identifier for the student quizm Status to see the status of the quiz if it is valid or not for weightage, TotalScore the total score for the student, StartedAt when the quiz was started, FinishedAt when the quiz was finished and Remarks to store remarks.

The final table needed will be the StudentAnswer table this table will be needed store the answers taken by the student during the quiz and see if they are right. It should contain the StudentID a unique identifier for the student, QuestionID to identify the question, ChoiceID for a unique identifier for choice, Answer to hold the students answer and Score the score for the answer.

Assumptions:

The quiz is multiple choices and only has 4 choices.

The students have all taken the quiz.

##### (i.iv.iii) Query:

1. Tutor Mr Ben would like to know how many students had gotten A for Quiz 1 uploaded for the Machine Learning module.

Select COUNT(\*) As “Number Of Students Who Got A”

FROM Module m INNER JOIN Quiz q on m.ModuleID = q.ModuleID

INNER JOIN StudentQuiz sq ON q.QuizID = sq.QuizID

WHERE m.ModuleName = “Machine Learning” AND q.Title = “Quiz 1” AND sq.TotalScore >= 80.

1. A student with the ID of S10205505 wants to check the results of their Quiz 1 in the machine learning module and see his answers and the correct answers to see which questions he had gotten wrong and correct and their respective scores

SELECT sa.QuestionID As “Question Number”, sa.Answer as “Student Answer”, qa.CorrectChoice as “Correct Answer”, sa.Score as “Student Score” FROM Module m INNER JOIN Quiz q on m.ModuleID = q.ModuleID

INNER JOIN StudentQuiz sq ON q.QuizID = sq.QuizID INNER JOIN StudentAnswer sa ON sq.SQuizID = sa.SQuizID INNER JOIN QuestionAnswer qa ON qa.ChoiceID = sa.ChoiceID WHERE m.ModuleName = “Machine Learning” AND q.Title = “Quiz 1” AND sq.StudentID = "S10205505" ORDER BY sa.QuestionID.

1. Tutor Mr Ben would like to know the average score of the students for Quiz 1 uploaded for the Machine Learning module.

Select AVG(sq.TotalScore) As “Average Marks”

FROM Module m INNER JOIN Quiz q on m.ModuleID = q.ModuleID

INNER JOIN StudentQuiz sq ON q.QuizID = sq.QuizID

WHERE m.ModuleName = “Machine Learning” AND q.Title = “Quiz 1”.

## (ii) Final ER Model And The List Of Mapped Relations

(ii.i) ER Model

*(Please see the attachment- “ADB Assignment 2 Final ER Model.png” for clearer picture )*

### (ii.ii) List of mapped relations

**Announcement**(<ModuleID>, AnnouncementID,AnnouncementDate,AnnouncementDescription)

**StudentContact**(<StudentID>,StudentName,StudentContact)

**StudentEmailAddress**(<StudentID>,StudentName,StudentEmailAddress)

**Tutor**(TutorID, TutorName, Password, Gender, TutorContact,TutorEmailAddress, AreaOfProfession,AcademicQualification)

**Student**(StudentID,StudentName,Password,CurrentGrade,CumulativeGrade,<CourseID>)

**Course**(CourseID,CourseName,IsOverseas)

**Module**(ModuleID,ModuleName, CreditUnit, <CourseID>)

**ModuleTutor**(<ModuleID>,<TutorID>,TeachingYear, Semester, TutorName, ModuleName,RoleOfTutor)

**StudentModule**(<StudentID>,<ModuleID>,Attempt,ModuleName,StudentName,Semester,StudyYear,ModuleGrade)

**Branch**(BranchID,BranchName,Country,Address)

**StudentBranch**(<BranchID>,<StudentID>,BranchName,Country,Address,StudentName)

**TutorBranch**(<BranchID>,<TutorID>,BranchName,Country,Address,TutorName)

**Administrator**(<AdminID>,EmailAddress,ContactNo,Password,AdminName)

**AdminManageCourse**(<AdminID>,<CourseID>,EmailAddress,CourseName,AdminName)

**Support**(SupportID,<AdminID>,<TutorID>,<StudentID>,SupportDetail,IsResolved,Feedback)

**Assignment**(AssignmentID, AssignmentName, AssignmentDetails, Percentage, Deadline, <ModuleID>)

**Test**(TestID, TestName, TestDetails, Percentage, TestTimeLimit, <ModuleID>)

**AssignmentSubmission**(AssignmentSubmissionID, AssignmentAttachment, AssignmentGrade, TotalScore, AssignmentSubmissionTime, Feedback, <AssignmentID>, <StudentID>,<TutorID>)

**TestSubmission**(TestSubmissionID, TestGrade, TotalScore, TestSubmissionTime, Feedback, IsRetest <TestID>, <StudentID>, <TutorID>)

**TestQuestion**(QuestionID, TestAttachment, Answer, Score, <TestID>)

**TestQuestionSubmission**(<TestID>, <TestSubmissionID>, SubmissionAnswers)

**CourseModule**(<ModuleID>, <CourseID>, ModuleName, CourseName, IsOversea )

**LearningMaterial**(MaterialID, <ModuleID>, ModuleName, <TutorID>, TutorName, <TypeID>, MaterialType, Description, FileName, MaterialName, DateTimeFileUploaded, DateTimeShown, IsShown)

**LearningMaterialType**(<TypeID>, MaterialType)

**StudentQuiz**(<StudentID>, <QuizID>, Status, TotalScore, StartedAt, FinishedAt, Remarks, <ModuleID>, ModuleName, Title)

**QuestionCategory**(CategoryID, <QuestionID>, Category)

**StudentAnswer**(<StudentID>,<QuizID>,<QuestionID>,<ChoiceID>, Answer, Score, CorrectChoice)

**QuestionAnswer**(<QuestionID>, <QuizID>, ChoiceID, CorrectChoice, Content)

**QuizQuestion**(QuestionID, <QuizID>, CategoryID, Type, Level, Score, Content, Category)

**Quiz**(QuizID, <ModuleID>,HostID, Title, Summary, Percentage, Published, CreatedAt, StartsAt, EndsAt, ModuleName)

## (iii) How Your Cloud Database Design Meets The Requirements Of The Entire System

Since the LMS requires the submission of assignments and assessments, it should also initially allow the tutors to create assignments and tests for the students to submit. In the database, there is an Assignment and Test table to store the information of the assignments and tests created. Each record in the Assignment table is a single assignment created by the tutors that has a unique AssignmentID to be referenced in the AssignmentSubmission table, in order to identify which assignment the submission record corresponds to. Each assignment’s name and details are stored in AssignmentName and AssignmentDetails. Since each assignment makes up a certain percentage of a student’s grade in a module, there is a Percentage attribute to store that value. The deadline for submission is also stored in the Deadline attribute. The Assignment table is created in the SQL format due to it needing to consistently receive submissions from students and updates from tutors in a timely manner. This ensures that when students submit their works, the database is updated in time to reflect a punctual submission. For AssignmentSubmission table, we use the Blob storage method as we need to store files and data in each record, which SQL databases cannot handle.

Similarly, for each test that the tutors create, a new record is added to the Test table, which stores the tutor’s assigned test name and details in TestName and TestDetails respectively. There is also a Percentage attribute that stores the percentage that the test contributes to the student’s overall grade for that module. The amount of time allotted for the test to be completed is stored in TestTimeLimit. The Test and TestSubmission tables are also stored in the SQL format to ensure that there is immediate consistency between the information in the cloud and information received locally. This is especially important due to the time limit that each test has, so if a TestSubmission record is updated slightly later than that of the expected end time of the test, then there could be issues regarding late submissions affecting the grade of the student.

Each Test is also made up of many TestQuestions, which have a set answer in the Answer attribute, and a score set to the Score attribute. Any pictures or files shown on the question are stored in a .zip file stored in TestAttachment. The answers for the TestQuestions that are provided in the TestSubmission by the students are stored in the TestQuestionSubmission table as SubmissionAnswers in a JSON file.

Both Assignment and Test table records can be partitioned by horizontally fragmentation on the module that they belong to, using the ModuleID foreign key. This allows for easier access to assignments and tests that belong to a certain module due to the high number of records that will be present in these tables.

The LMS also provides a place for the Tutor to upload their learning materials and the Students from different modules are able to download.  
Students who are entrolled to a Module, no matter which Course they are from (Some course might have shared the same Module, such as FI students and IT students might take Machine Learning module at the same time) they are able to view and download the learning materials uploaded by the Tutor in charge of the Module.   
Learning materials varies from Weekly Lecture Notes, Tutorials, Common Test paper, Practical documents, Lecture videos and etc. These file size sometimes can go up to Gigabytes. Thus, Blob storage is used to store all the large file objects and arrange them according to the Module on the Azure Blob Storage.  
When a tutor wants to upload a learning material, for example, Week 1 Tutorial for Machine Learning module, he can first create a new Learning Material type called “Tutoral”, and specify the Learning material name, description, the date where it should be shown to the students (A tutor can ‘hide’ all the upcoming material by setting a datetime for it to be visible for students). After that, he can select whichever file in his computer to upload. The file will then be uploaded to the Azure Blob Storage with relevant container name being defined. At the same time, a new record of the LearningMaterial is stored in the SQL database with the FileName attribtue being the clickable link generated to download the document in the Azure Blob Storage.

The LearningMaterial table are partitioned by horizontal fragmentation on the modules each learning material is created for by using the ModuleID foreign key. This allows the student to access the learning material easier and reduce the number of JOIN and WHERE clauses which in term, improves the query speed.

Lastly, since there can have more than one Tutor in charge of teaching a Module, the tutor can have the right to edit/delete/add a learning material record. Such operations includes editing the shown date that student are able to see the learning material. As sometime the learning material might need to be released earlier due to public holidays.

The LMS also requires students to be able to do online quizzes, hence it should also allow the tutors to create online quizzes for the students to do and submit. In the database, there is an Quiz table to store the information of the online quizzes created each record in the quiz table is a single quiz created by the tutor with unique QuizID to identify them. Details such Title, Summary, Percentage for weightage of quiz, published to see if the quiz is active, the quiz starting time and ending time are also there. Similar to quiz there is a StudentQuiz table which stores the studentid for identification with the quizid to make a composite primary key this table main purpose is to see if the quiz if valid for the student and the totalscore the student has gotten. The Quiz and StudentQuiz table are created in SQL format as it has a very strict structure that will not change furthermore, due to the time limit each quiz has the quiz submissions must be consistent and efficient the SQL provides this and prevents issues regarding late submissions or errors as the database will be updated consistently and on time.

Each quiz is made up of many questions and these questions also have questioncategories and questionanswers furthermore, each student has thier own studentanswer for each question. The quizquestion holds the questions, category, level of difficulty, score of the question and the category of the question. The questionanswer holds the questions correct choice and answer and each studentanswer holds their answer and score for the question. Due to the large amount of data from studentanswers and questions NoSQL database such as document database would be effective here as it is easily scalable.

The StudentQuiz table can be partitioned by horizontal fragmentation on the modules each quiz are created for by using the ModuleID foreign key. This allows for easier access to online quizzes depending on their module as there will many records in this table and this will make it easier to identify records and prevent the need of queries.

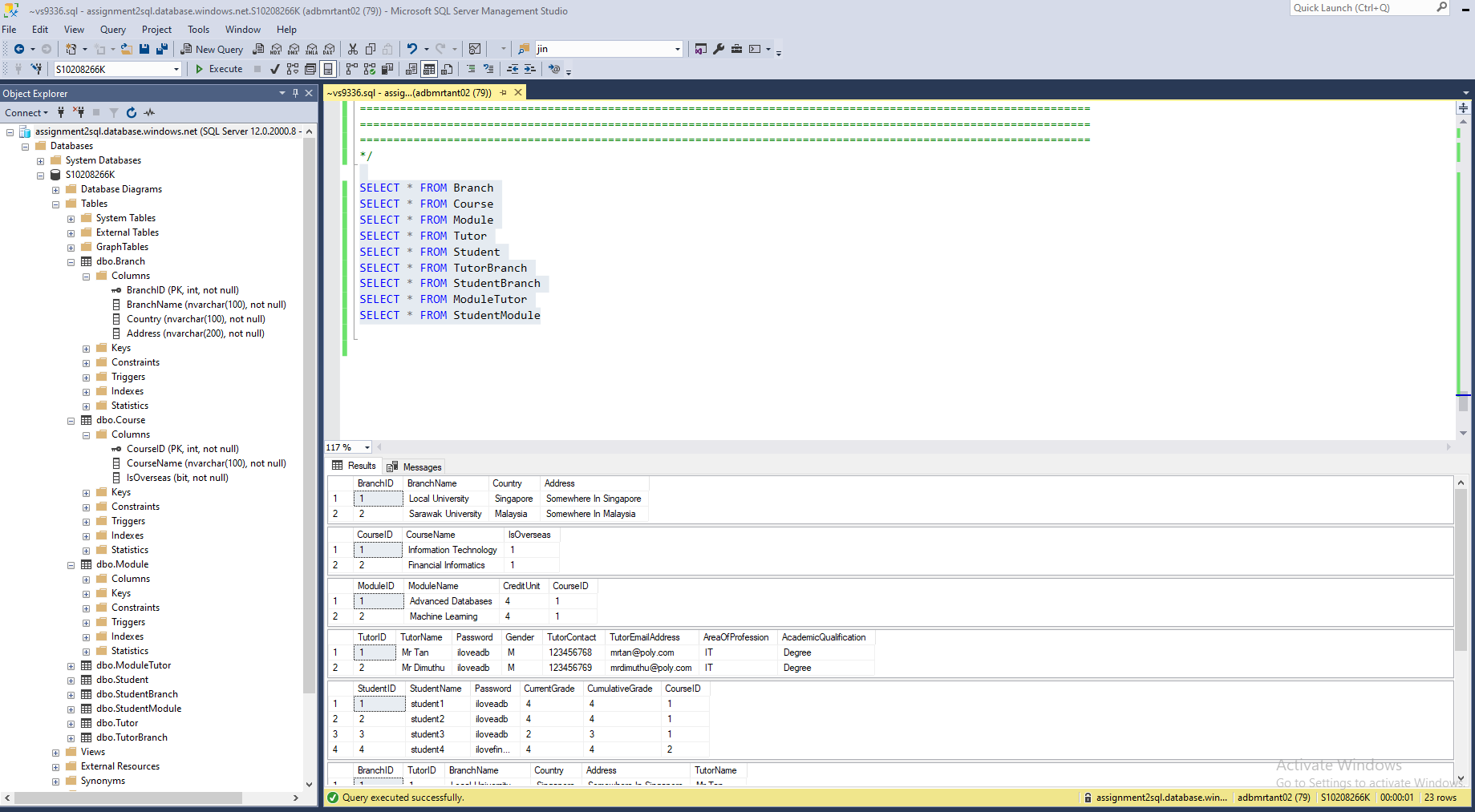
For grading of assessments, the AssignmentSubmission table consists of the TutorID, indicating the tutor grading the assignment. The AssignmentGrade and TotalScore attributes will contain values assigned by the tutors after marking and lastly the Feedback attribute where tutors can provide feedback to the students if there is a need for any.

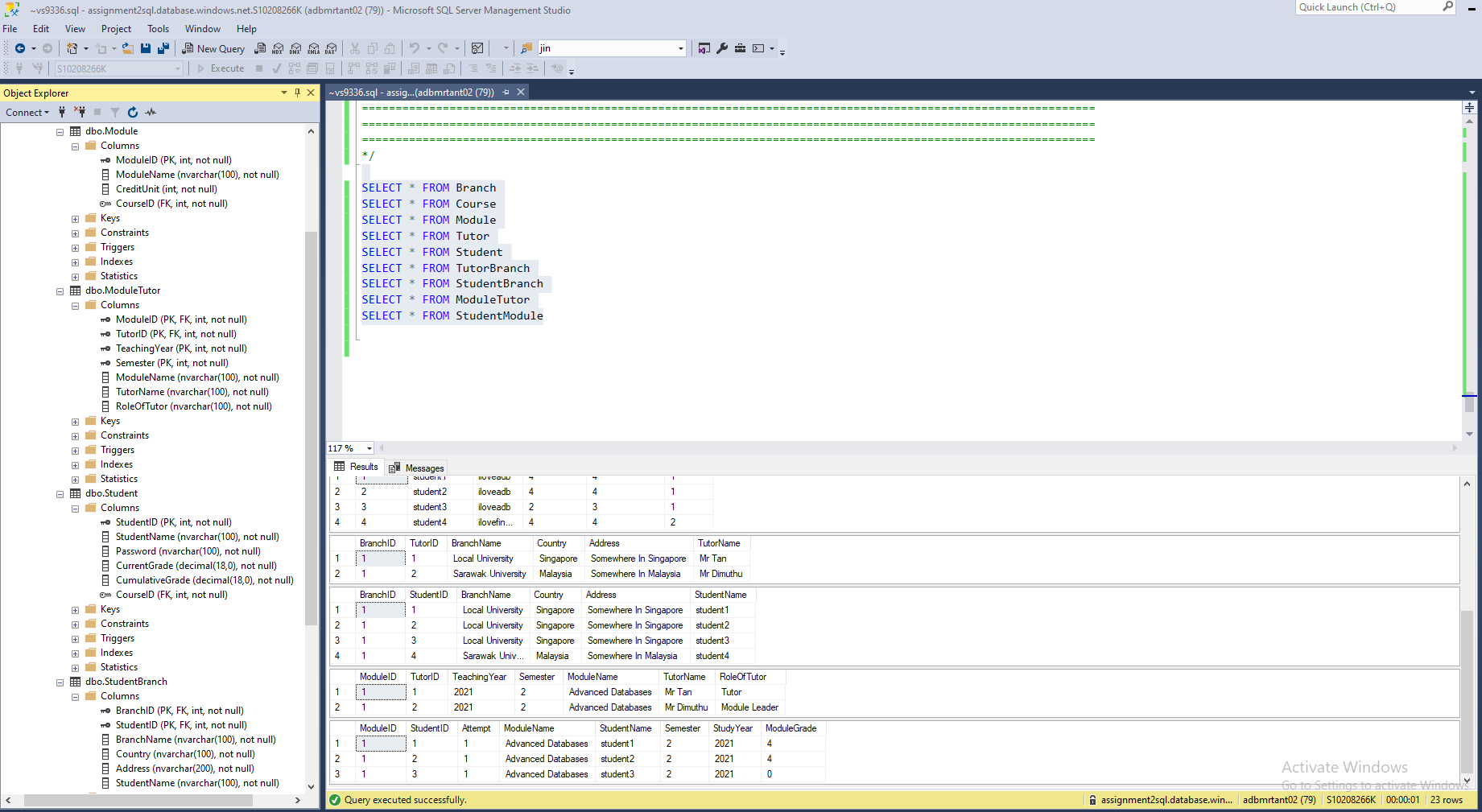
Similarly, the TestSubmission table consists of the same attributes as AssignmentSubmission to handle Tutor grading of tests and providing of feedback.

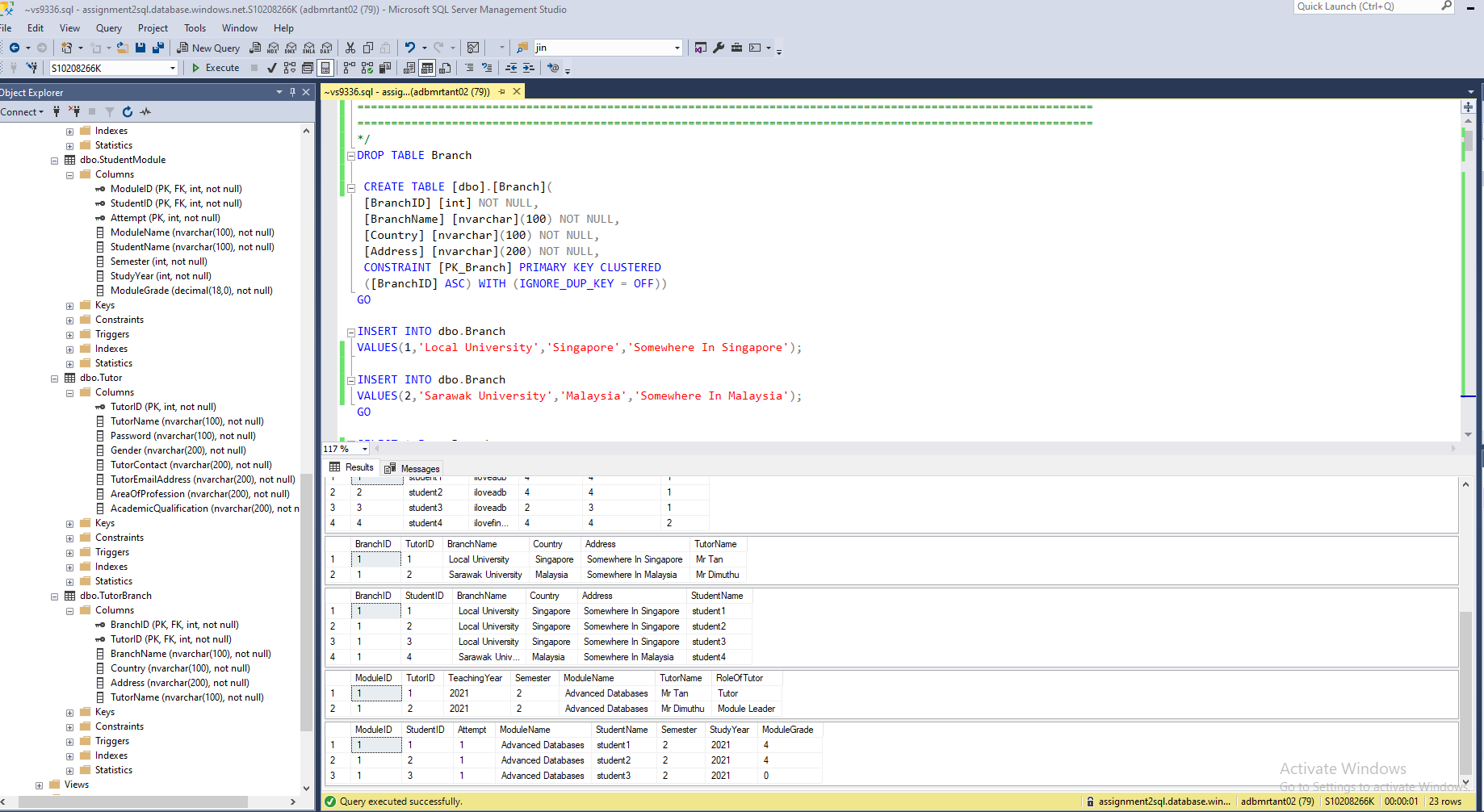
Announcements are paritioned based on the modules for example Advanced Databases partition and Machine Learning Partition. They will store values such as the date of the announcements and video notes, in the announcement description.

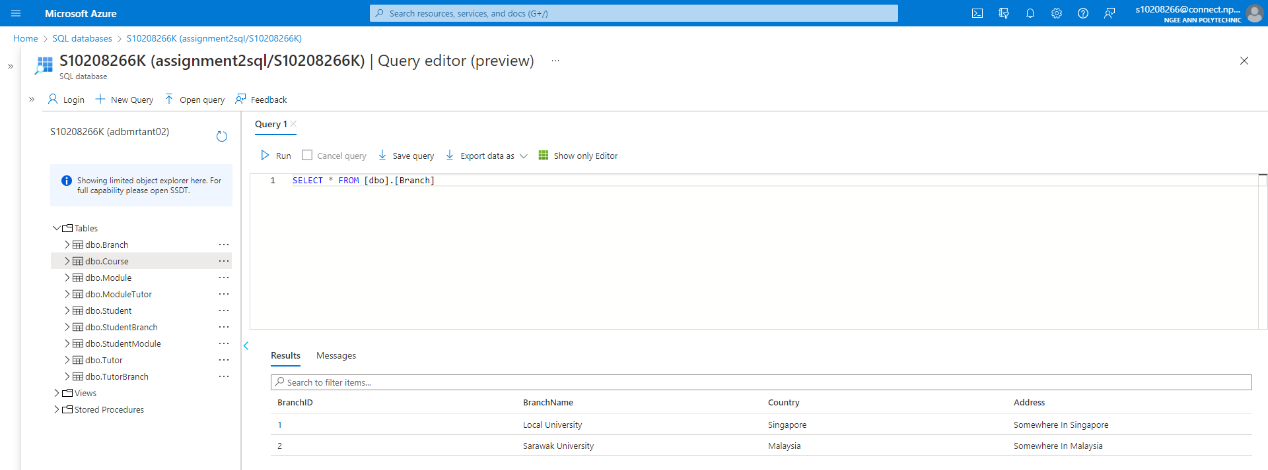
## (iv) Evidence (screenshots) Of Implementation Of Cloud Storages (In Microsoft Azure Environment)

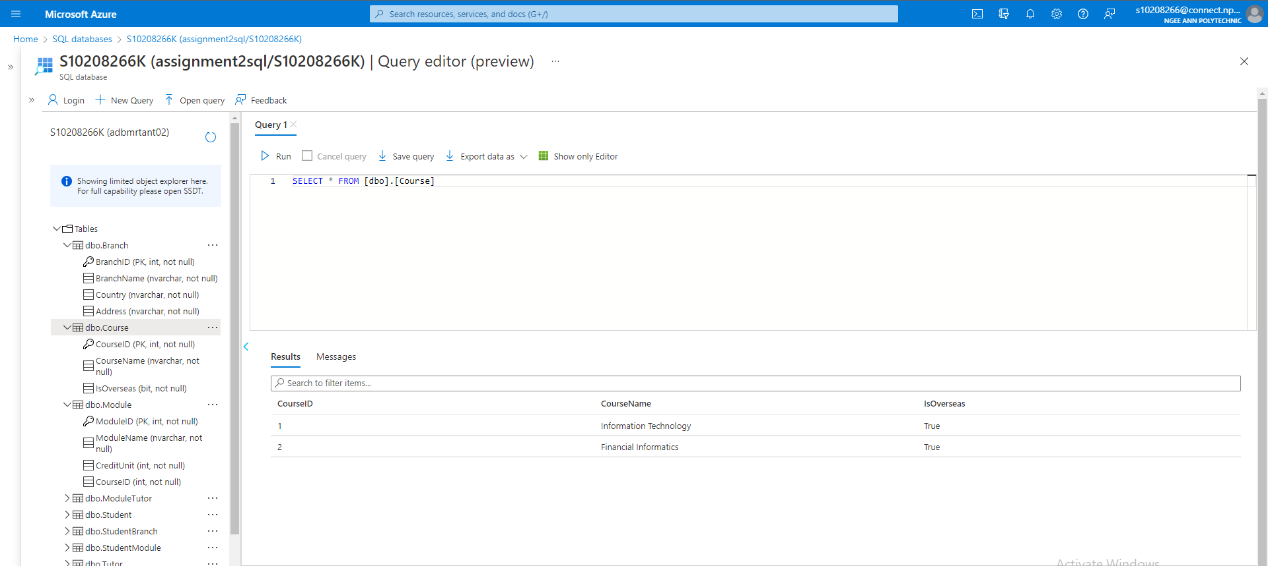
SQL:

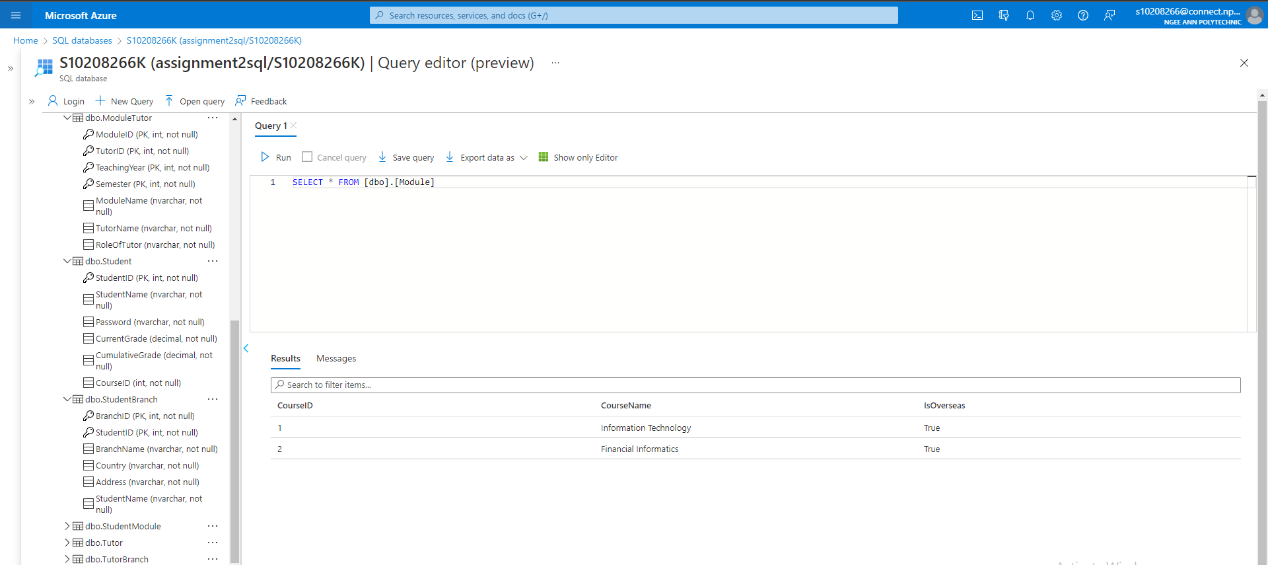


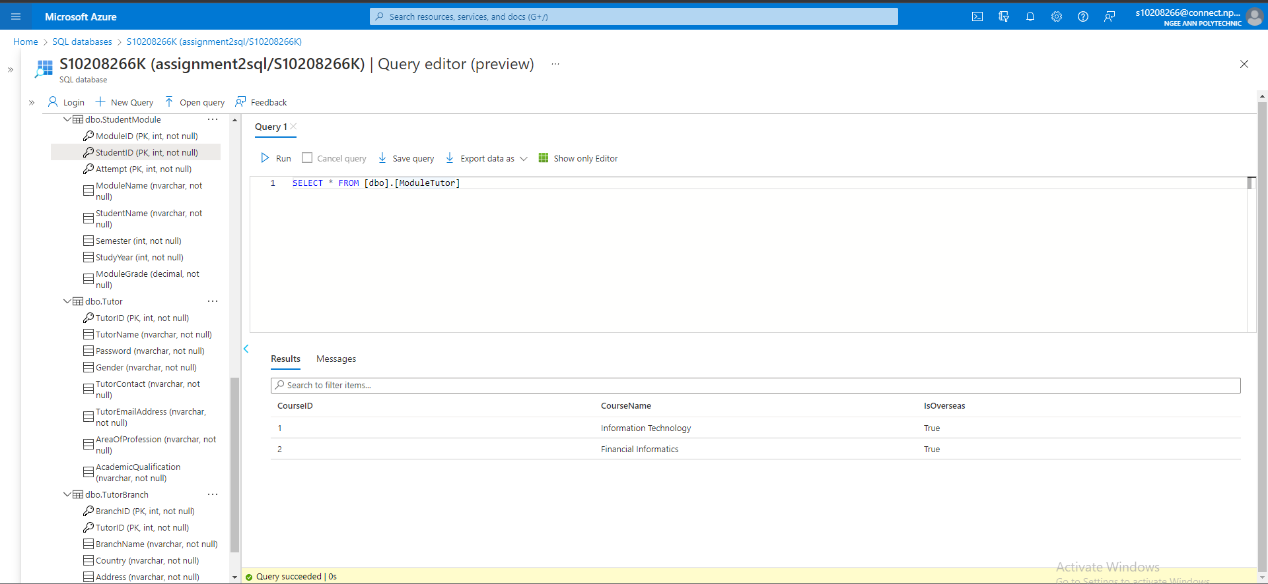


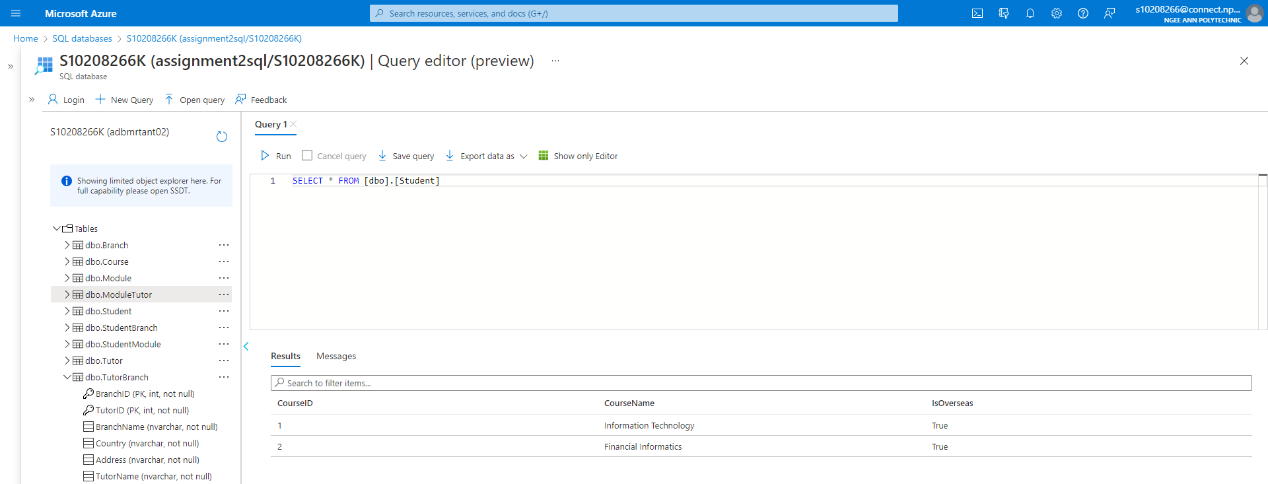




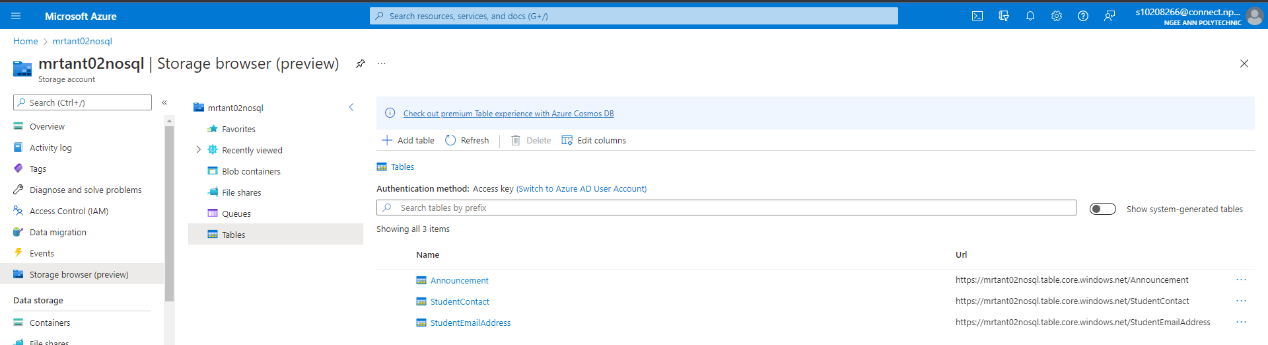


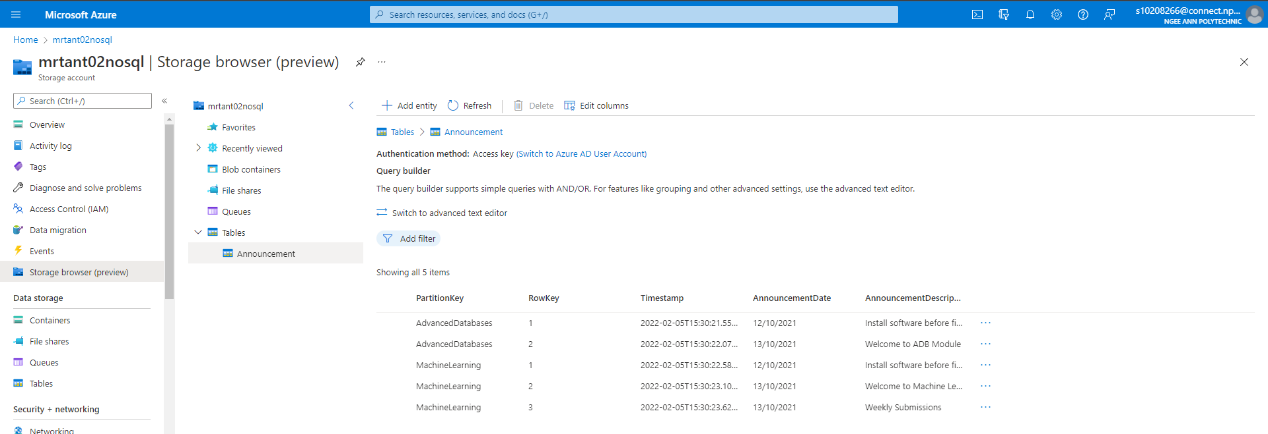


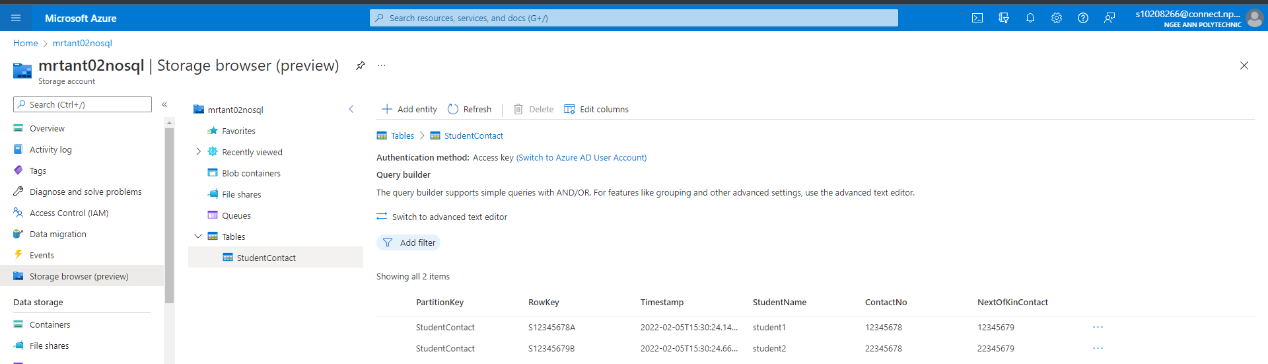


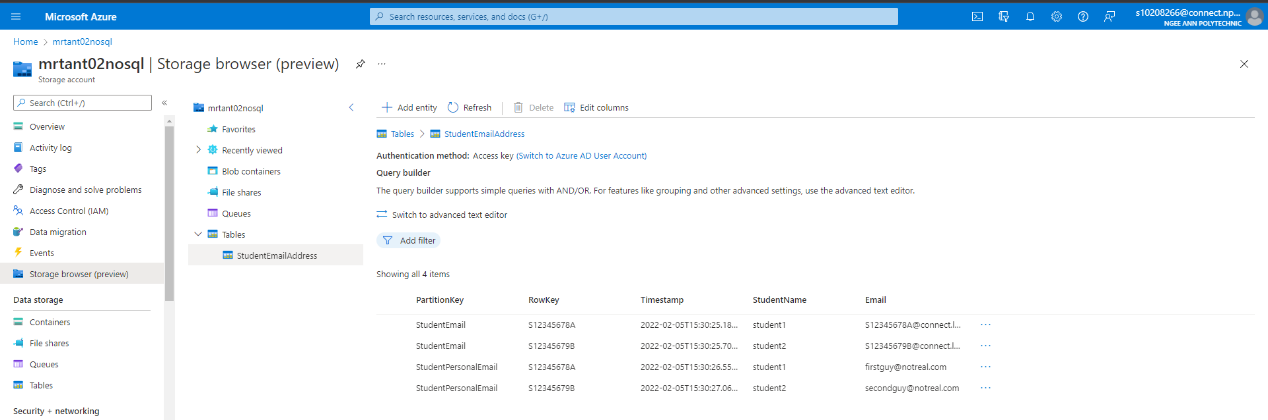


NoSQL









# 2) Individual

## (i) Muhd Dandarawie Bin Rohani

### (i.i) Denormalisation Application

- **StudentContact**(<StudentID>, StudentName,StudentContact)

- **StudentEmailAddress**(<StudentID>, StudentName,StudentEmailAddress)

Two tables, StudentContact and StudentEmailAddress, created so that students can have more than one contact number and email address linked to them. The rows in both tables do not need an additional key to be combined into composite keys as for example a student named Ali, with a StudentID of S10302122Q can have two rows in StudentContact:

S10302122Q, student1,92135572

S10302122Q, student2,82467596

Where both rows are unique. There shouldn’t be duplicates and if there is it should be handled in the front end.

- **StudentModule**(<StudentID>,<ModuleID>,Attempt,ModuleName,StudentName,Semester,StudyYear,ModuleGrade)

ModuleGrade stored in StudentModule table to independently store grades of different students taking different modules in a separate table from Student and Module to avoid cluttering and removing JOIN between the two tables.

- **ModuleTutor**(<ModuleID>,<TutorID>,TeachingYear,Semester,TutorName,ModuleName,RoleOfTutor)

Denormalisation done between Module and Tutor entity to store values such as TeachingYear, Semester and RoleOfTutor. This is done to avoid unnecessary data table entries in Module and Tutor tables while ensuring less use of join statements by including TeachingYear and Semester attributes in the table. TeachingYear and Semester are combined with ModuleID and TutorID as Composite keys to ensure unique entries. TeachingYear and Semester attributes are then used in Announcement table as shown below as foreign keys to link Announcements to ModuleTutor as tutors assign announcements to modules.

- **Announcement**(<ModuleID>, AnnouncementID,AnnouncementDate,AnnouncementDescription)

Announcement table, with its own unique AnnouncementID paired with ModuleID foreign key, contains date and description where date will retrieve the current date the row was created or updated so that it can be reused, the same as description.

- **TutorBranch**(<BranchID>, <TutorID>, BranchName, Country,Address,TutorName)

A branch can have many tutors, so in order to reduce data in Branch table, we create another table called TutorBranch to store the tutors working for the branch.

- **AdminManageCourse**(<AdminID>,<CourseID>,AdminName,EmailAddress,CourseName)

A denormalised table to include information regarding details of admins that are managing the courses. This is to assist in queries whereby there is a need to retrieve admin details for students to tutors to seek support.

##### (i.i.i)Denormalisation examples from previous queries done in Assignment checkpoint 2:

Students who have failed Advanced databases module:

**- StudentModule**(<StudentID>,<ModuleID>, Attempt, ModuleName,StudentName,Semester,StudyYear, ModuleGrade)

Before I continue explaining, we assume Mr Tan is looking for students who have failed Advanced databases module at least once.

Now with a new primary key called Attempt, we can straight away determine from the StudentModule table itself whether or not the student has failed as if the student attempt is more than 1, we can assume the student has failed the module. Previously, we required 2 inner joins between three tables, Student, StudentModule and Module itself before we can pull out the table values we are looking for.

### (i.ii) Data Placement Strategy

To recap my part in this assignment, it will be mainly focused on tutors grading students and students having access to their grades.

My choice regarding database design strategy would be distributed databases. This is so that we will be able to cater to branches that are overseas. Each region will have an extra database as a backup, replicated from its original database. However, only some tables will be replicated from one region to the other.

Examples of tables that should continuously be in sync with other databases across all regions will be Course Table, Module Table and Branch Table.

Tables that require backup to their own backup database near their region e.g Singapore has 3 independent zones, so one will be used for their original database and the other zone as their backup.

Examples will be TutorBranch, StudentBranch, Tutor, Student, ModuleTutor and StudentModule.

### (i.iii) Cloud Storage Strategies

##### (i.iii.i)NoSQL:

**- Announcement**(<ModuleID>, AnnouncementID,AnnouncementDate,AnnouncementDescription)

Announcement in NoSQL is different from the Relational mapping shown above. It will contain values

such as ‘AdvancedDatabases’ for PartitionKey and numerals starting from 1 for RowKey. This is because it is not necessary for Announcement table to have a relation with ModuleID to link announcements, instead it can be handled front-end.

- **StudentContact**(<StudentID>,StudentName,StudentContact)

StudentContact table in NoSQL will have ‘StudentContact’ as its PartitionKey and ‘StudentID’ as its RowKey. It is possible to implement NoSQL strategy for the StudentContact table because student contact details has no relation with other tables in the databases.

- **StudentEmailAddress**(<StudentID>,StudentName,StudentEmailAddress)

StudentEmailAddress table in NoSQL will have ‘StudentEmailAddress’ as its PartitionKey and ‘StudentID’ as its RowKey. Similar to StudentContact, StudentEmailAddress does not have a relation with other tables in the databases so it is entirely possible to implement NoSQL strategy to this table.

Basically, as shown later on, these three tables will be using AzureTableStorage service to store its data in NoSQL format.

##### (i.iii.ii)SQL:

**- Tutor**(TutorID, TutorName, Password, Gender, TutorContact,TutorEmailAddress, AreaOfProfession,AcademicQualification)

The tutor table will use SQL as the there isn’t a need for the table to be highly scalable. The only instances of interaction with this table is when tutors join or leave, change they contact or email address.

**- Student**(StudentID,StudentName,Password,CurrentGrade,CumulativeGrade,<CourseID>)

For the purpose of this assignment, I assume the client keeps all the Student data who have attended their university, even after their graduation. Here, it seems like it will make more sense to use NoSQL as scalability is required to store data of past students. However, most attributes in Student work in tandem with other attributes from other tables for example CurrentGrade and CumulativeGrade. Hence, it is not recommended to store the data inside NoSQL. However, it is possible to transfer data of past students to a NoSQL database, that will scale horizontally as time passes by, more students will graduate and more student data needs to be stored. But simplifying our approach to doing entity relationship diagram deem the action unnecessary.

**- Course**(CourseID,CourseName,IsOverseas)

My assumption is that realistically there won’t be a point whereby scaling of courses offered is required, hence there isn’t a need to implement NoSQL for Course table.

**- Module**(ModuleID,ModuleName, CreditUnit, <CourseID>)

A course can have many modules, meaning there are a lot more modules than there are courses. Hence scalability has to be considered for the Module table, as the more courses there are, the more rows that will exist in the Module table. However, because of how both module and courses have relation between itself and other entities, it is best to assign SQL storage strategy for the Module table for the relations.

**- ModuleTutor**(<ModuleID>,<TutorID>,TeachingYear, Semester, TutorName, ModuleName,RoleOfTutor)

A denormalised table formed between the relation Module and Tutor. We cannot assign NoSQL format for this table as NoSQL is non relational while ModuleTutor is a relational database table.

**- StudentModule**(<StudentID>,<ModuleID>,Attempt,ModuleName,StudentName,Semester,StudyYear,ModuleGrade)

A denormalised table made to store data of students taking different modules to avoid cluttering of data to occur in either Student or Module table. As this table has relations with Student Table and Module table, it is not recommended that this table uses NoSQL technology.

**- Branch**(BranchID,BranchName,Country,Address)

There isn’t a need to set up Branch table as NoSQL because there can only be a few branches and it won’t reach a point where it needs to expand horizontally for efficiency.

**- StudentBranch**(<BranchID>,<StudentID>,BranchName,Country,Address,StudentName)

**- TutorBranch**(<BranchID>,<TutorID>,BranchName,Country,Address,TutorName)

Similar tables that link student and tutors tables to their branches. We have to consider how far StudentBranch will expand due to the number of students taking courses from the university, inclusive of local and overseas students. The reason why I did not end up with using NoSQL for this table is because of the relation between student/tutor and branch. As NoSQL is non relational, it is not beneficial for NoSQL to be used in this scenario.

**- Administrator**(<AdminID>,EmailAddress,ContactNo,Password,AdminName)

**- AdminManageCourse**(<AdminID>,<CourseID>,EmailAddress,CourseName,AdminName)

**- Support**(SupportID,<AdminID>,<TutorID>,<StudentID>,SupportDetail,IsResolved,Feedback)

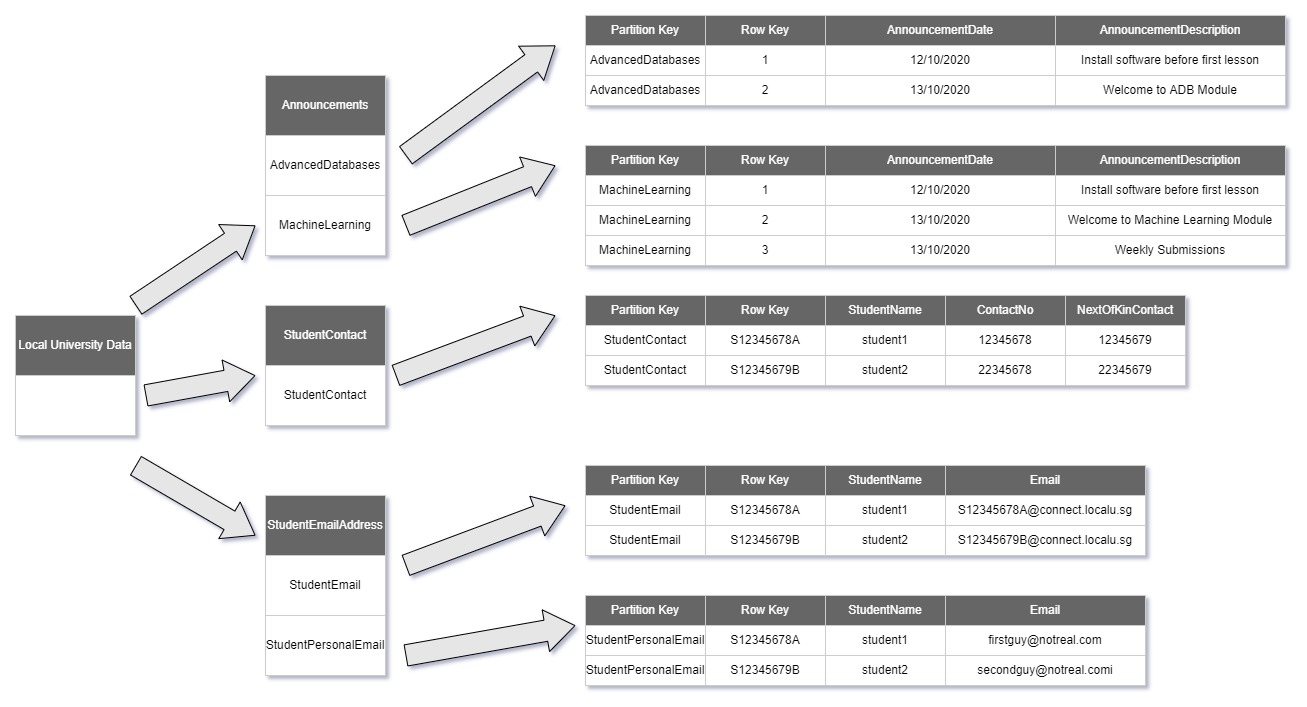
For Administrator, AdminManageCourse and Support tables, SQL is the best choice as relations are needed for the tables to work. Furthermore, it is not recommended to implement NoSQL as we need consistency across the tables in querying data as retrievals of data in AdminManageCourse is required when Students or Tutors request support.

For SQL tables, we will consider Shared application with separate databases. This is because we have a local university and a university overseas. We need to have the option whereby if needed, we can have more separated databases spread through regions/locations where the schools are offered overseas.

### (i.iv) Design/Schema on Cloud Storage Used

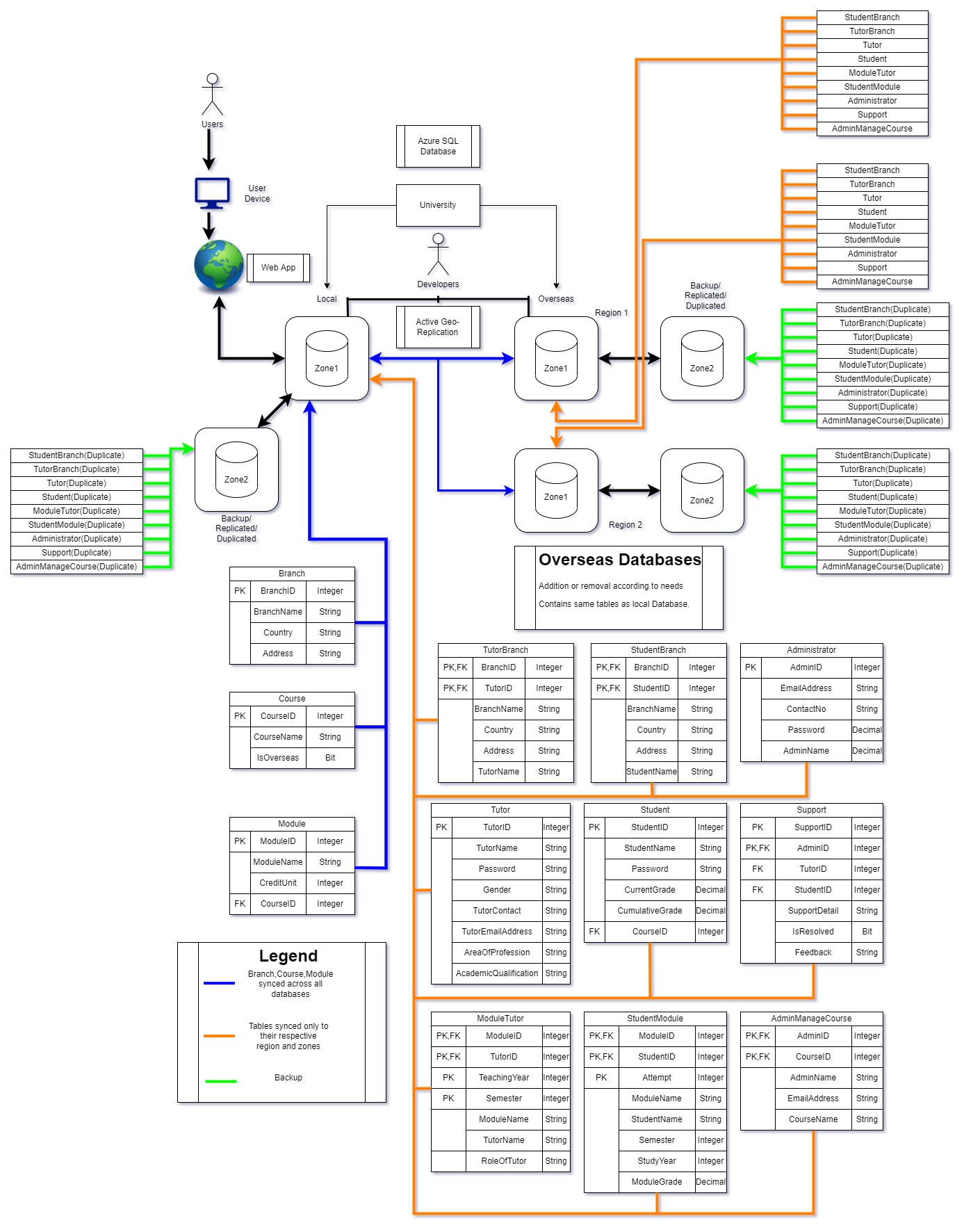
##### (i.iv.i)NoSQL:

AzureTableStorage



##### (i.iv.ii)SQL:

AzureSQL Database



### (i.v) Implementation of business queries

##### (i.v.i)NoSQL/LINQ/AzureTableStorage/Key-Value:

Announcement

// Retrieve all entities in Advanced Databases partition

var advancedDatabasesQuery1 = (from announcements in table.CreateQuery<Announcements>()

where announcements.PartitionKey == "AdvancedDatabases" select announcements);

foreach (Announcements theAnnouncements in advancedDatabasesQuery1)

{

Response.Write("PartitionKey: ");

Response.Write(theAnnouncements.PartitionKey + "<br>");

Response.Write("RowKey: ");

Response.Write(theAnnouncements.RowKey + "<br>");

Response.Write("Announcement Date: ");

Response.Write(theAnnouncements.AnnouncementDate + "<br>");

Response.Write("Announcement Description: ");

Response.Write(theAnnouncements.AnnouncementDescription + "<br>");

Response.Write("<br>");

}

// Retrieve announcement(s) where date is 12/10/2021

var advancedDatabasesQuery2 = (from announcements in table.CreateQuery<Announcements>()

where announcements.AnnouncementDate == Convert.ToDateTime("12/10/2021") select announcements);

foreach (Announcements theAnnouncements in advancedDatabasesQuery2)

{

Response.Write("PartitionKey: ");

Response.Write(theAnnouncements.PartitionKey + "<br>");

Response.Write("RowKey: ");

Response.Write(theAnnouncements.RowKey + "<br>");

Response.Write("Announcement Date: ");

Response.Write(theAnnouncements.AnnouncementDate + "<br>");

Response.Write("Announcement Description: ");

Response.Write(theAnnouncements.AnnouncementDescription + "<br>");

Response.Write("<br>");

}

// Retrieve all entities in machine learning partition

var machineLearningQuery1 = (from machineLearning in table.CreateQuery<MachineLearning>()

where machineLearning.PartitionKey == "MachineLearning" select machineLearning);

foreach (MachineLearning theMachineLearning in machineLearningQuery1)

{

Response.Write("PartitionKey: ");

Response.Write(theMachineLearning.PartitionKey + "<br>");

Response.Write("RowKey: ");

Response.Write(theMachineLearning.RowKey + "<br>");

Response.Write("Announcement Date: ");

Response.Write(theMachineLearning.AnnouncementDate + "<br>");

Response.Write("Announcement Description: ");

Response.Write(theMachineLearning.AnnouncementDescription + "<br>");

Response.Write("<br>");

}

StudentContact

// Retrieve all entities in StudentContact partition

var studentContactQuery1 = (from studentContact in table.CreateQuery<StudentContact>()

where studentContact.PartitionKey == "StudentContact" select studentContact);

foreach (StudentContact theStudentContacts in studentContactQuery1)

{

Response.Write("PartitionKey: ");

Response.Write(theStudentContacts.PartitionKey + "<br>");

Response.Write("RowKey: ");

Response.Write(theStudentContacts.RowKey + "<br>");

Response.Write("Student Name: ");

Response.Write(theStudentContacts.StudentName + "<br>");

Response.Write("Contact Number: ");

Response.Write(theStudentContacts.ContactNo + "<br>");

Response.Write("Next Of Kin Contact: ");

Response.Write(theStudentContacts.NextOfKinContact + "<br>");

Response.Write("<br>");

}

// Retrieve student from StudentContact where Row Key is S12345679B

var studentContactQuery2 = (from studentContact in table.CreateQuery<StudentContact>()

where studentContact.RowKey == " S12345679B " select studentContact);

foreach (StudentContact theStudentContacts in studentContactQuery2)

{

Response.Write("PartitionKey: ");

Response.Write(theStudentContacts.PartitionKey + "<br>");

Response.Write("RowKey: ");

Response.Write(theStudentContacts.RowKey + "<br>");

Response.Write("StudentName: ");

Response.Write(theStudentContacts.StudentName + "<br>");

Response.Write("Contact Number: ");

Response.Write(theStudentContacts.ContactNo + "<br>");

Response.Write("Next Of Kin Contact: ");

Response.Write(theStudentContacts.NextOfKinContact + "<br>");

Response.Write("<br>");

}

StudentEmailAddress

// Retrieve all entities in StudentEmail partition

var studentEmailQuery1 = (from studentEmail in table.CreateQuery<StudentEmail>()

where studentEmail.PartitionKey == "StudentEmail" select studentEmail);

foreach (StudentEmail theStudentEmail in studentEmailQuery1)

{

Response.Write("PartitionKey: ");

Response.Write(theStudentEmail.PartitionKey + "<br>");

Response.Write("RowKey: ");

Response.Write(theStudentEmail.RowKey + "<br>");

Response.Write("Student Name: ");

Response.Write(theStudentEmail.StudentName + "<br>");

Response.Write("School Email Address Of Student: ");

Response.Write(theStudentEmail.Email + "<br>");

Response.Write("<br>");

}

// Retrieve entity where school email address of student is “S12345678A@connect.localu.sg”

var studentEmailQuery2 = (from studentEmail in table.CreateQuery<StudentEmail>()

where studentEmail.Email == "S12345678A@connect.localu.sg" select studentEmail);

foreach (StudentEmail theStudentEmail in studentEmailQuery2)

{

Response.Write("PartitionKey: ");

Response.Write(theStudentEmail.PartitionKey + "<br>");

Response.Write("RowKey: ");

Response.Write(theStudentEmail.RowKey + "<br>");

Response.Write("Student Name: ");

Response.Write(theStudentEmail.StudentName + "<br>");

Response.Write("School Email Address Of Student: ");

Response.Write(theStudentEmail.Email + "<br>");

Response.Write("<br>");

}

// Retrieve all entities in StudentPersonalEmail partition

var studentPersonalEmailQuery1 = (from studentPersonalEmail in table.CreateQuery<StudentPersonalEmail>()

where studentPersonalEmail.PartitionKey == "StudentPersonalEmail" select studentPersonalEmail);

foreach (StudentPersonalEmail theStudentPersonalEmail in studentPersonalEmailQuery1)

{

Response.Write("PartitionKey: ");

Response.Write(theStudentPersonalEmail.PartitionKey + "<br>");

Response.Write("RowKey: ");

Response.Write(theStudentPersonalEmail.RowKey + "<br>");

Response.Write("Student Name: ");

Response.Write(theStudentPersonalEmail.StudentName + "<br>");

Response.Write("School Email Address Of Student: ");

Response.Write(theStudentPersonalEmail.Email + "<br>");

Response.Write("<br>");

}

// Retrieve entity where Row Key is “S12345678A”

var studentPersonalEmailQuery2 = (from studentPersonalEmail in table.CreateQuery<StudentPersonalEmail>()

where studentPersonalEmail.RowKey == "S12345678A" select studentPersonalEmail);

foreach (StudentPersonalEmail theStudentPersonalEmail in studentPersonalEmailQuery2)

{

Response.Write("PartitionKey: ");

Response.Write(theStudentPersonalEmail.PartitionKey + "<br>");

Response.Write("RowKey: ");

Response.Write(theStudentPersonalEmail.RowKey + "<br>");

Response.Write("Student Name: ");

Response.Write(theStudentPersonalEmail.StudentName + "<br>");

Response.Write("School Email Address Of Student: ");

Response.Write(theStudentPersonalEmail.Email + "<br>");

Response.Write("<br>");

}

##### (i.v.ii) SQL/AzureSQLDatabase

// Retrieve students who require repeat of ‘Advanced Databases’ Module for Year 2021 Semester 2:

// Assume ADB ModuleID = 1.

// Assume client manually inputs semester and studyyear where the values are 2 and 2021 respectively

CREATE PROC uspRetrieveRepeatStudents(@ModuleName VARCHAR(100), @Semester INT, @StudyYear INT)

AS

IF Exists(SELECT \* FROM StudentModule WHERE ModuleName = @ModuleName

AND Semester = @Semester

AND StudyYear = @StudyYear

AND ModuleGrade = 0)

BEGIN

SELECT \* FROM StudentModule WHERE ModuleName = @ ModuleName

AND Semester = @Semester

AND StudyYear = @StudyYear

AND ModuleGrade = 0)

// If client wants a printed result

DECLARE @printRetrieveRepeatStudents xml = (SELECT \* FROM StudentModule

WHERE ModuleName = @ ModuleName

AND Semester = @Semester

AND StudyYear = @StudyYear

AND ModuleGrade = 0

AND Semester = @Semester FOR XML AUTO)

PRINT CONVERT(NVARCHAR(MAX), @printRetrieveRepeatStudents)

END

ELSE

BEGIN

DECLARE @RetrieveRepeatStudentsNull Varchar(200)

SET @RetrieveRepeatStudentsNull = ‘No records exists!’

Print @RetrieveRepeatStudentsNull

END

EXEC uspRetrieveRepeatStudents @ModuleName = ‘Advanced Databases’,@Semester = 2,@StudyYear=2021

// Retrieve all tutors teaching Advanced Databases Module Overseas

// Assume ADB ModuleID = 1.

CREATE PROC uspRetrieveModuleTutorOverseas(@ModuleName VARCHAR(100),@CurrentLocation VARCHAR(100))

AS

DECLARE @command varchar(1000)

SELECT @command = ‘USE ? SELECT \* FROM ModuleTutor

INNER JOIN TutorBranch

ON ModuleTutor.TutorID = TutorBranch.TutorID

WHERE ModuleTutor.ModuleName = @ModuleName

AND

TutorBranch.Country != @CurrentLocation’

EXEC sp\_Msforeachdb @command

EXEC uspRetrieveModuleTutorOverseas @ModuleName = ‘Advanced Databases’, @CurrentLocation = ‘Singapore’

## (ii) James

### (ii.i) Denormalization Application

**Assignment**(AssignmentID, AssignmentName, AssignmentDetails, Percentage, Deadline, <ModuleID>)

**Test**(TestID, TestName, TestDetails, Percentage, TestTimeLimit, <ModuleID>)

There is no denormalization used in the Assignment and Test tables as there is hardly any chance that any Assignment or Test shares similar attribute values with each other. And if they did, the effects of the redundant data when creating queries of any kind, including those that require joins, would be minimal. For example, two or more Test records can have similar Percentage or TestTimeLimit values, but repetition of these values in different records would not matter in the context of assignments and tests, as there are a limited number of values that can be used in that way. If they did matter, then a join to ModuleID of the Module entity would be mandatory to filter the records retrieved in that query, regardless of whether these values are redundant or not due to the sheer number of records that would be present either way.

**AssignmentSubmission**(AssignmentSubmissionID, AssignmentAttachment, AssignmentGrade, TotalScore, AssignmentSubmissionTime, Feedback, <AssignmentID>, <StudentID>,<TutorID>)

**TestSubmission**(TestSubmissionID, TestGrade, TotalScore, TestSubmissionTime, Feedback, IsRetest <TestID>, <StudentID>, <TutorID>)

There is denormalization used in the creation of the AssignmentSubmission and TestSubmission tables. Instead of having a separate table for Feedback, which would take up more disk space and require queries to use join statements in order to reference it, we implement it as an attribute within these two tables instead, reducing the complexity of any queries that need to reference it and reducing disk space usage.

**TestQuestion**(QuestionID, TestAttachment, Answer, Score, <TestID>)

Since each TestQuestion's attribute values do not unnecessarily repeat themselves, then there is no need for denormalization. Each TestAttachment also consists of a single .zip file that contains all the files or documents that the student submits, removing the need for redundant records to display all the files submitted for a single question.

**TestQuestionSubmission**(<TestID>, <TestSubmissionID>, SubmissionAnswers)

No denormalization is used on this table.

### (ii.ii) Data Placement Strategy

The most optimal data placement alternative for these tables would be to partition them with horizontal fragmentation on the different modules offered in the university. Since all five tables are prone to having constant updates in data, it would be very costly to have a replicated data placement system due to all the nodes the database has been replicated at being constantly updated with new records. If a centralized alternative was used, where the data is placed in a single database, it would make accessing data slow and unreliable as many users from different faculties will be retrieving and sending new records into the database at any one time.

If we fragment the tables horizontally based on each module in each faculty, this will make data much easier to retrieve as the query does not have to run through as many records to return the ones requested. Instead of running through a whole university’s worth of records, the query sent to a fragmented set of data for a certain module in a certain faculty or school would just run through that much smaller set of data, making it faster and cheaper to make requests to the database. If data records need to be stored, only that partition of the tables needs to be updated instead of having to update all the partitions, reducing update costs as well. This also causes the data retrieved to be more available, as if one partition fails, it will not affect the other partitions’ performance. The data will also be more reliable as the database updates more quickly because of the lower number of users that are accessing the data at any one time.

##### (ii.ii.i)Examples of Partitioned Data Placement

For example, the Test table can be partitioned with horizontal fragmentation based on each record’s ModuleID foreign key. Each partition has a ModuleID assigned to it, which in turn can reference the course of study that the module belongs to, allowing us to partition the records by course if needed.

In this example, we can assume that the table records are fragmented horizontally on the records’ ModuleIDs. One fragment could be that of ModuleID 4, which corresponds to a certain module, say, Advanced Databases. And another fragment could be that of ModuleID 5, which corresponds to Machine Learning. This means that when data needs to be retrieved or updated, the query can leave out a WHERE statement to specify which ModuleID the record belongs to, making data retrieval and updates faster. Each fragment can be named according to the module it belongs to, like TestADB or TestML.

The SQL query to find the test records for Advanced Databases would therefore look something like this: SELECT \* FROM Test, in which the partitioned table TestADB would be selected,

instead of SELECT \* FROM Test WHERE ModuleID = 4, in which the Test table is selected.

### (ii.iii) Cloud Storage Strategies

The tables Test, TestSubmission, TestQuestion, Assignment, and AssignmentSubmission can be stored in the SQL format as all the records in these tables have the exact same columns. There is no way for these records to deviate from the format that they are in, making them a good fit for the SQL relational database format.

For AssignmentSubmission and TestQuestion, since the uploaded files are compressed into the .zip format before being inserted into the record, there should be no deviation from the standard record format no matter what files are being inserted by the user.

The TestSubmissionQuestion table can be saved as a NoSQL MongoDB document database. Since the number of questions in each TestSubmissionQuestion record can vary, SQL format cannot be used as not all records will have the same columns. MongoDB is chosen as there will be an extremely large amount of data in the TestSubmissionQuestion table: there will be one record for each student taking part in each test, so if we have many tests and many students in the university, then there will be a very large amount of TestSubmissionQuestion records. Due to the way MongoDB scales with the amount of records this also makes it a good choice as compared to Blob Storage or Table Storage.

##### (ii.iii.i)SQL:

**Assignment**(AssignmentID, AssignmentName, AssignmentDetails, Percentage, Deadline, <ModuleID>)

**Test**(TestID, TestName, TestDetails, Percentage, TestTimeLimit, <ModuleID>)

**TestSubmission**(TestSubmissionID, TestGrade, TotalScore, TestSubmissionTime, Feedback, IsRetest <TestID>, <StudentID>, <TutorID>)

There is a need for the Assignment, Test, and TestSubmission tables to have a relationship with the Module table due to how these tables are partitioned, so we use SQL so that JOIN can be used for the queries when partitioning. As mentioned, these tables contain records that do not deviate from the set format, so the use of an SQL format would be useful when storing the table in this data. The data in these tables will also be reliable as compared to NoSQL storage methods, which is important especially for TestSubmission as submissions have to be received at a certain time for tests, so if the values are updated lately, then there might be inconsistencies when referring to the database to check who has or has not submitted.

**AssignmentSubmission**(AssignmentSubmissionID, AssignmentAttachment, AssignmentGrade, TotalScore, AssignmentSubmissionTime, Feedback, <AssignmentID>, <StudentID>,<TutorID>)

**TestQuestion**(QuestionID, TestAttachment, Answer, Score, <TestID>)

Like the Assignment, Test, and TestSubmission tables, the AssignmentSubmission and TestQuestion records are standardized in format. There are also foreign keys used to reference other tables for their values. Even though there is the file’s name in the record, all the files or documents submitted will first be compressed into a .zip file by the LMS application itself before being inserted into the database. This keeps the records standardized as the only files in the records are .zip files no matter what files are being uploaded by the user. Query can be used to enter the file into the database at the same time as the record entry.

##### (ii.iii.ii)NoSQL:

**TestQuestionSubmission**(<TestID>, <TestSubmissionID>, SubmissionAnswers)

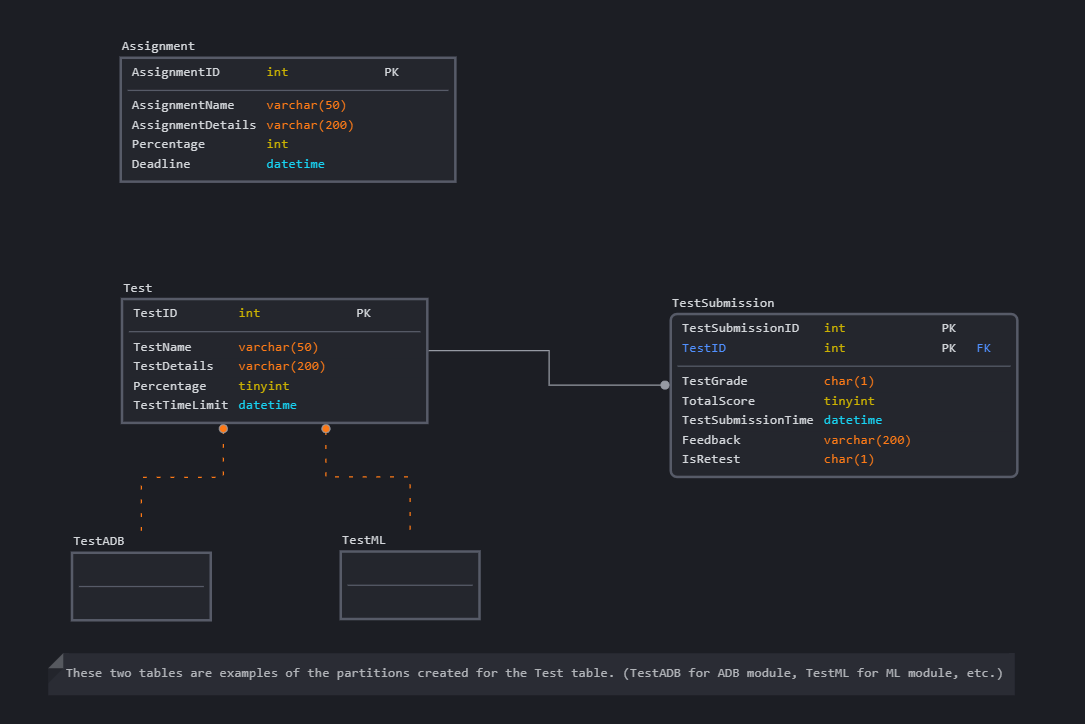
TestQuestionSubmission is the table that stores all the answers that the students provide whenever they submit a test. This means whenever a TestSubmission is created, a new TestQuestionSubmission is created. The SubmissionAnswers attribute is supposed to contain all the answers for a specific TestSubmission record. Therefore, there is no definite number of answers that can be provided by the student, so there cannot be a specific number of columns to fit a variable number of answers.

Thus, NoSQL is used as there is no column restrictions on NoSQL records. Since records will come in for each student taking the test, for every test, this means that there will be an extremely large number of records in the TestQuestionSubmission table. Thus, we can use the Document Database storage format, through MongoDB. In this way, each record is stored in a JSON format, and each SubmissionAnswers value can be an array containing all the answers. This makes it so that there is no one number of answers that can be inserted, but there will always be an array to store them, no matter the length. This makes it much simpler to store data without the need for redundant information in an SQL format.

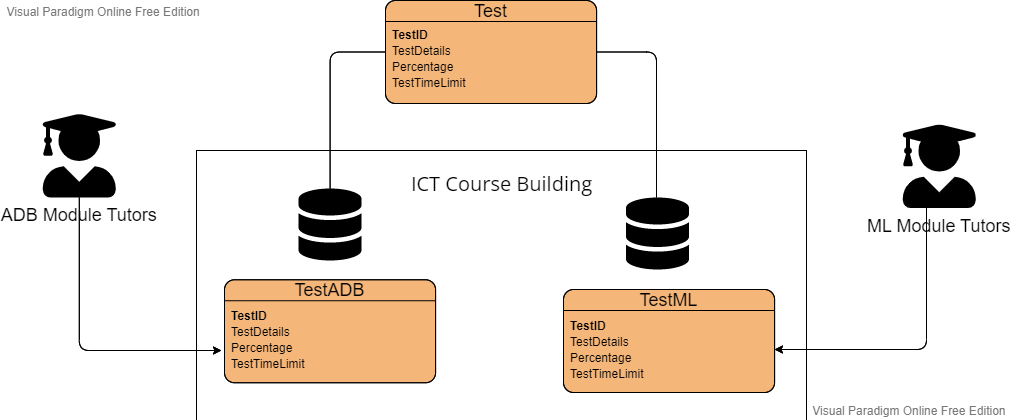
### (ii.iv) Design/Schema on Cloud Storage Used

##### (ii.iv.i) SQL:

Using the Azure SQL Database we implement the database according to this schema.



Detailed View of how Test table is horizontally fragmented into different tables for each module.



##### (ii.iv.ii) NoSQL:

How each record looks like in JSON format in the MongoDB storage system.

Note that for different tests there can be a different amount of answers provided by the student for their submissions.

### Graphical user interface, text, application, email Description automatically generated

### (ii.v) Implementation of business queries

##### (ii.v.i) SQL/AzureSQLDatabase:

To begin partitioning for Test and Assignment, we need to create filegroups for each Module.

So assuming we have a database called UniversityDatabase:

ALTER DATABASE UniversityDatabase

ADD FILEGROUP TestAdvancedDatabases

GO

ALTER DATABASE UniversityDatabase

ADD FILEGROUP TestMachineLearning

GO

ALTER DATABASE UniversityDatabase

ADD FILEGROUP TestDiscreteMathematics

GO

… and so on, for all the modules.

Then we add an .ndf file to each filegroup created. An .ndf file contains the data for the records entered based on which filegroup the .ndf file is in.

ALTER DATABASE [UniversityDatabase]

    ADD FILE

    (

    NAME = [AdvancedDatabasesRecords],

    FILENAME = 'C:\Program Files\Microsoft SQL Server\...\MSSQL\DATA\ UniversityDatabase.ndf',

        SIZE = 3072 KB,

        MAXSIZE = UNLIMITED,

        FILEGROWTH = 1024 KB

) TO FILEGROUP [TestAdvancedDatabases]

…repeated for each module.

Next we create a function to map the records in the Test table to partitions based on the values in the ModuleID column, so one column for each unique ModuleID.

CREATE PARTITION FUNCTION [PartitioningByModuleID] (int)

AS RANGE RIGHT FOR VALUES ('1', '2', '3',

               '4', ‘5’, ...);

Then we map each partition to a filegroup using a partition scheme.

CREATE PARTITION SCHEME PartitionByModuleID

AS PARTITION PartitioningByModuleID

TO (TestAdvancedDatabases, TestMachineLearning, TestDiscreteMathematics, ...);

The main Test table can now be created, with its partitions, and some test data.

CREATE TABLE Test(

TestID int IDENTITY (1,1),

TestName varchar (40) NOT NULL,

TestDetails varchar (200) NOT NULL,

Percentage tinyint NOT NULL,

TestTimeLimit datetime NOT NULL,

ModuleID int NOT NULL,

CONSTRAINT PK\_Test PRIMARY KEY (TestID),

CONSTRAINT FK\_Test\_ModuleID FOREIGN KEY (ModuleID) REFERENCES

Module (ModuleID))

ON PartitionByModule (ModuleID);

GO

INSERT INTO Reports (TestID, TestName, TestDetails, Percentage,

TestTimeLimit, ModuleID)

SELECT 1, 'Test 1', ‘Details for Test 1’, 20, ‘2021-11-11 13:00’, 1

UNION ALL

SELECT 2, 'Test 2', ‘Details for Test 2’, 10, ‘2021-12-11 10:00’, 2

UNION ALL

SELECT 3, 'Test 3', ‘Details for Test 3’, 15, ‘2021-11-11 11:00’, 3

UNION ALL

## (iii) Chen Han

**(iii.i) Denormalisation Application**

**Tutor** (TutorID, TutorName, Password, Gender, TutorContact, TutorEmailAddress, AreaOfProfession, AcademicQualification)

**Module**(ModuleID, ModuleName, CreditUnit,<CourseID>)

**LearningMaterial**(MaterialID, <ModuleID>, ModuleName, <TutorID>, TutorName, MaterialType, Description, FileName, MaterialName, DateTimeFileUploaded, DateTimeShown, IsShown)

Denormalisation has also done between LearningMaterial and Module entites as well as LearningMatieral and Tutor entities, by including non-key attributes of ModuleName in Module entity and TutorName in Tutor entity.   
This is done to avoid unnecessary data table entries in these entities while ensuring less use of join statements because the ModuleName and TutorName are often being used.

##### (iii.i.i)Denormalisation examples from previous queries done in Assignment checkpoint 2:

1. A student would like to find all the learning materials that are “Weekly Tutorial”.

SELECT lm.\*, lmt.MaterialType

FROM LearningMaterial lm

INNER JOIN LearningMaterialType lmt

ON lm.TypeID = lmt.TypeID

WHERE lmt.MaterialType = ‘Weekly Tutorial’

AND lm.IsShown = 1

**AFTER**

SELECT \*

FROM LearningMaterial lm

WHERE MaterialType = “Weekly Tutorial”

AND isShown = 1

As shown from the query above, the Inner join required for LearningMaterial and LearningMaterialType entities is removed as the MaterialType attribute is denormalised into the LearningMaterial entity. Thus improving the speed of retreiving and reduce the number of JOIN operations.

1. Tutor Ben would like to find out how many learning materials are uploaded by him in Module "Mobile Application Development".

SELECT COUNT(\*) AS "No. of learning materials Ben uploaded in MAD"

FROM LearningMaterial lm

INNER JOIN Module m

ON m.ModuleID = lm.ModuleID

INNER JOIN Tutor t

ON t.TutorID = lm.TutorID

WHERE t.TutorName = 'Ben' AND lm.ModuleID IN

(SELECT ModuleID FROM Module

WHERE ModuleName = 'MobileApplicationDevelopment')

**AFTER**

SELECT COUNT(\*) AS “No. of learning materials Ben uploaded in MAD”  
FROM LearningMaterial

WHERE TutorName = ‘Ben’ AND

ModuleName = ‘MobileApplicationDevelopment’

As shown from the query above, the Inner join required for TutorUploadMaterial and Tutor entities is removed as the TutorName attribute is denormalised into the TutorUploadMaterial entity. Also there are no need to use a sub query to get the ModuleName from Module entity as it is already being denormalised into the LearningMaterial entity. Thus, only one Inner join operation is needed in the problem to retreive speficic learning material for a tutor in a module.

### (iii.ii) Data Placement Strategy

With the additional requirement added to the system, being the oversea university branches. A unique data placement approach should be considered to cater to the needs of students aboard.   
Firstly, we will be using **distributed database** design for this LMS, where it is partitioned into different sites, E.g. campus in Singapore will have its own database and management system containing learning materials for all the courses offered by local university, whereas for other oversea branches, they have their own courses in their campus hosting on a different site. Since all the sites are interconnected in a distributed database, students from Singapore are also able to download the learning materials from oversea sites if they are enrolled to an oversea program.   
Next, for data placement strategy used at each site, taking into accounts that the learning material that the database will be storing are mainly images, videos and document files, these are often large files and takes up to Gigabytes for one video. So replication will not be the main approach to store these data as the cost of replicating the learning materials will be doubled. I will be using a similar approach as James (see Above) which is to partition the tables for learning materials with **horizontal fragmentation** on the different modules offered in the university.

E.g. If a student taking Module ADB wants to see what are the learning material uploaded in this module, he can simply go to the Fragment where module is ADB and search for whatever learning material he wants to see. This omits the use of WHERE clause in the SELECT statement and this in term, improves the efficiency of accessing data on smaller scale.

This helps to reduce the cost for having replicated databases in each site to store the learning materials uploaded by tutor. In fact, it stores them according to whichever modules that the Tutor taught, and where they would upload their materials.

However, some learning materials that are important will be replicated and store it in the central database (In Singapore branch), this ensures the realiability and availability of the data and it can be recovered immedidately once a breakdown has happened in one of the sites.

##### 

### 

### (iii.iii) Cloud Storage Strategies

##### (iii.iii.i)SQL

**Tutor** (TutorID, TutorName, Password, Gender, TutorContact, TutorEmailAddress, AreaOfProfession, AcademicQualification)

**Module**(ModuleID, ModuleName, CreditUnit,<CourseID>)

**LearningMaterial**(MaterialID, <ModuleID>, ModuleName, <TutorID>, TutorName, MaterialType, Description, FileName, MaterialName, DateTimeFileUploaded, DateTimeShown, IsShown)

The table Tutor, Course, Module and LearningMaterialType can be stored in the SQL format as all the columns are fixed and scalabiltiy is not required for these tables.

The CourseModule and LearningMaterial tables containing foreign keys used to refer to other tables. For LearningMaterial table, there should be a file attached in each record, the file can be obtained by using a link to Blob Storage online and it can be then displayed for the Student to click and download the file online.

The LearningMaterial table will also be partitioned using Hortizontal Paritioning approach according to the ModuleID.

##### (iii.iii.ii)NoSQL: Blob Storage

Using Blob storage to store the large file objects uploaded by Tutor, once uploaded, a new record of Learning Material will be generaeted together with the URL link to download the file uploaded. This link will be then displayed for the student to download the file when they are at the module page.

### (iii.iv) Design/Schema on Cloud Storage Used

##### (iii.iv.i)SQL

A picture containing diagram

Description automatically generated

##### (iii.iv.ii)No SQL: Blob Storage

Graphical user interface, text, application, email

Description automatically generated

This is a Blog Storage created with multiple containers, each container follows a strict naming convension – learingmaterial + short form of the module. The container can be manually created on the Azure itself or using the Query to create the container while uploading the file.

### (iii.v) Implementation of business queries

##### (iii.v.i) SQL

For the purpose of demonstration, the Tutor and Module entites created only containing their names.

First by creating the Database

CREATE DATABASE [ADB\_Assignment 2]

GO

USE [ADB\_Assignment 2]

GO

Next, perform horizontal partitioning for learning materials table according to the module ID in the module table. In the example below, we will be using three Modules: Advanced Database, SDD and Machine Learning as an example to demonstrate how the partition looks like.

*Partitioning was learnt from this source:* [*https://www.sqlshack.com/database-table-partitioning-sql-server/*](https://www.sqlshack.com/database-table-partitioning-sql-server/)

ALTER DATABASE [ADB\_Assignment 2]

ADD FILEGROUP LearningMaterialML

GO

ALTER DATABASE [ADB\_Assignment 2]

ADD FILEGROUP LearningMaterialADB

GO

ALTER DATABASE [ADB\_Assignment 2]

ADD FILEGROUP LearningMaterialSDD

GO

ALTER DATABASE [ADB\_Assignment 2]

ADD FILE

(

NAME = [ADBRecords],

FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL15.SQLEXPRESS\MSSQL\DATA\ADB\_Assignment 2.ndf',

SIZE = 3072 KB,

MAXSIZE = UNLIMITED,

FILEGROWTH = 1024 KB

) TO FILEGROUP [LearningMaterialADB]

GO

ALTER DATABASE [ADB\_Assignment 2]

ADD FILE

(

NAME = [MLRecords],

FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL15.SQLEXPRESS\MSSQL\DATA\ADB\_Assignment 22.ndf',

SIZE = 3072 KB,

MAXSIZE = UNLIMITED,

FILEGROWTH = 1024 KB

) TO FILEGROUP [LearningMaterialML]

GO

ALTER DATABASE [ADB\_Assignment 2]

ADD FILE

(

NAME = [SDDRecords],

FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL15.SQLEXPRESS\MSSQL\DATA\ADB\_Assignment 23.ndf',

SIZE = 3072 KB,

MAXSIZE = UNLIMITED,

FILEGROWTH = 1024 KB

) TO FILEGROUP [LearningMaterialSDD]

GO

SELECT

name as [FileName],

physical\_name as [FilePath]

FROM sys.database\_files

where type\_desc = 'ROWS'

GO

Output

ADB\_Assignment 2 C:\Users\Chen Han\ADB\_Assignment 2.mdf

ADBRecords C:\Program Files\Microsoft SQL Server\MSSQL15.SQLEXPRESS\MSSQL\DATA\ADB\_Assignment 2.ndf

MLRecords C:\Program Files\Microsoft SQL Server\MSSQL15.SQLEXPRESS\MSSQL\DATA\ADB\_Assignment 22.ndf

SDDRecords C:\Program Files\Microsoft SQL Server\MSSQL15.SQLEXPRESS\MSSQL\DATA\ADB\_Assignment 23.ndf

CREATE PARTITION FUNCTION [PartitioningByModuleID] (int)

AS RANGE RIGHT FOR VALUES ('2', '3')

GO

CREATE PARTITION SCHEME PartitionByModuleID

AS PARTITION PartitioningByModuleID

TO (LearningMaterialADB, LearningMaterialSDD, LearningMaterialML)

GO

After initializing the Partiton scheme, moving on to creating the Tutor and Module tables which is required for the LearningMaterials for foreign key references.

CREATE TABLE Tutor

(

TutorID int NOT NULL,

TutorName varchar (50),

CONSTRAINT Tutor\_PK PRIMARY KEY CLUSTERED (TutorID)

)

GO

INSERT INTO Tutor

VALUES

(1,'Mr Tan'),

(2, 'Mr Hung'),

(3, 'Mr Toh')

Go

CREATE TABLE Module

(

ModuleID int NOT NULL,

ModuleName varchar (50),

CONSTRAINT Module\_PK PRIMARY KEY CLUSTERED (ModuleID)

)

GO

INSERT INTO Module

VALUES

(1,'Advanced Database'),

(2, 'Solution Design & Development'),

(3, 'Machine Learning')

GO

Please take note that the Filename attribute in the LearningMaterial entity is referred to the downloadable link generated by Blob Storage when a Tutor has uploaded the learning material. This link can be used in the LMS for the student to download the learning material, which will be demonstrated in the next section.

CREATE TABLE LearningMaterial(

MaterialID int NOT NULL,

MaterialName varchar(50) NOT NULL,

TutorName varchar(50) NOT NULL,

ModuleName varchar(50) NOT NULL,

MaterialType varchar(50) NOT NULL,

Description varchar(200) NOT NULL,

FileName varchar(200) NOT NULL,

DateTimeUploaded datetime NOT NULL,

DateTimeShown datetime NOT NULL,

isShown varchar(3) NOT NULL,

TutorID int NOT NULL,

ModuleID int NOT NULL,

CONSTRAINT LearningMaterial\_PK PRIMARY KEY CLUSTERED (MaterialID, ModuleID, TutorID),

CONSTRAINT FK\_LearningMaterial\_Module FOREIGN KEY (ModuleID) REFERENCES Module (ModuleID),

CONSTRAINT FK\_LearningMaterial\_Tutor FOREIGN KEY (TutorID) REFERENCES Tutor (TutorID))

ON PartitionByModuleID (ModuleID);

GO

INSERT INTO LearningMaterial (MaterialID, MaterialName, TutorName, ModuleName, MaterialType, Description, FileName, DateTimeUploaded, DateTimeShown, isShown, TutorID, ModuleID)

SELECT 1, 'Lecture 1', 'Mr Tan', 'Advanced Database', 'Lecture', 'Please look through the lecture before the lesson starts',

'https://chenhanadbassignment2.blob.core.windows.net/learningmaterialadb/ADB Week 01 - Stored Procedures - Part I.pdf',

'2022/2/7 5:00:00 PM','2022/2/8 10:00:00 am','No', 1, 1 UNION ALL

SELECT 2, 'Lecture 1', 'Mr Hung', 'Solution Design & Development', 'Lecture', 'Please look through the lecture before the lesson starts',

'https://chenhanadbassignment2.blob.core.windows.net/learningmaterialsdd/01-2 - Introduction to Agile(1).pptx',

'2022/2/7 5:00:00 PM','2022/2/8 10:00:00 am','No',2, 2 UNION ALL

SELECT 3, 'Lecture 1', 'Mr Toh', 'Machine Learning', 'Lecture', 'Please look through the lecture before the lesson starts',

'https://chenhanadbassignment2.blob.core.windows.net/learningmaterialml/Lecture 0 - Intro to ML & DM\_PET.pptx',

'2022/2/7 5:00:00 PM','2022/2/8 10:00:00 am','No', 3, 3

SELECT

p.partition\_number AS PartitionNumber,

f.name AS PartitionFilegroup,

p.rows AS NumberOfRows

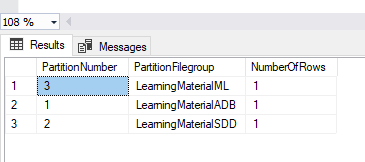
FROM sys.partitions p

JOIN sys.destination\_data\_spaces dds ON p.partition\_number = dds.destination\_id

JOIN sys.filegroups f ON dds.data\_space\_id = f.data\_space\_id

WHERE OBJECT\_NAME(OBJECT\_ID) = 'LearningMaterial

OUTPUT



Now the horizontal pertitioning has done for learning material table according to the module ID.

##### (iii.v.ii) NoSQL: Blob Storage

1. How can a student use this feature to download the file?

In the WebRole1

The function below is called to display the link for the user to click by entering the moduleName – which is the unique container name in Azure that contains all the learning materials for that module.

public void displayAllBlobs(string moduleName)

{

CloudBlobClient blobClient;

CloudStorageAccount cloudStorageAccount;

cloudStorageAccount = CloudStorageAccount.Parse(

CloudConfigurationManager.GetSetting("myConnectionString"));

blobClient = cloudStorageAccount.CreateCloudBlobClient();

CloudBlobContainer container = blobClient.GetContainerReference(moduleName);

foreach (IListBlobItem item in container.ListBlobs(null, false))

{

if (item.GetType() == typeof(CloudBlockBlob))

{

CloudBlockBlob blob = (CloudBlockBlob)item;

Response.Write("Module: " + moduleName + "<br/>");

Response.Write("Please download here: " + "<a href='" + blob.Uri + "'>" + blob.Name + "</a>" + "<br/>");

}

}

}

Next, by defining a new modules list that consist of all the module name, this is to demonstrate on the moment a Student has click into the Module of “Advanced Database”, the moduleName will set the “learningmaterialadb”, and all the learning materials will be shown for the students to download. Please ignore all other details such as Description and Learning Material Type, these can be soon added to improve the functionality of the LMS.

List<string> modules = new List<string>();

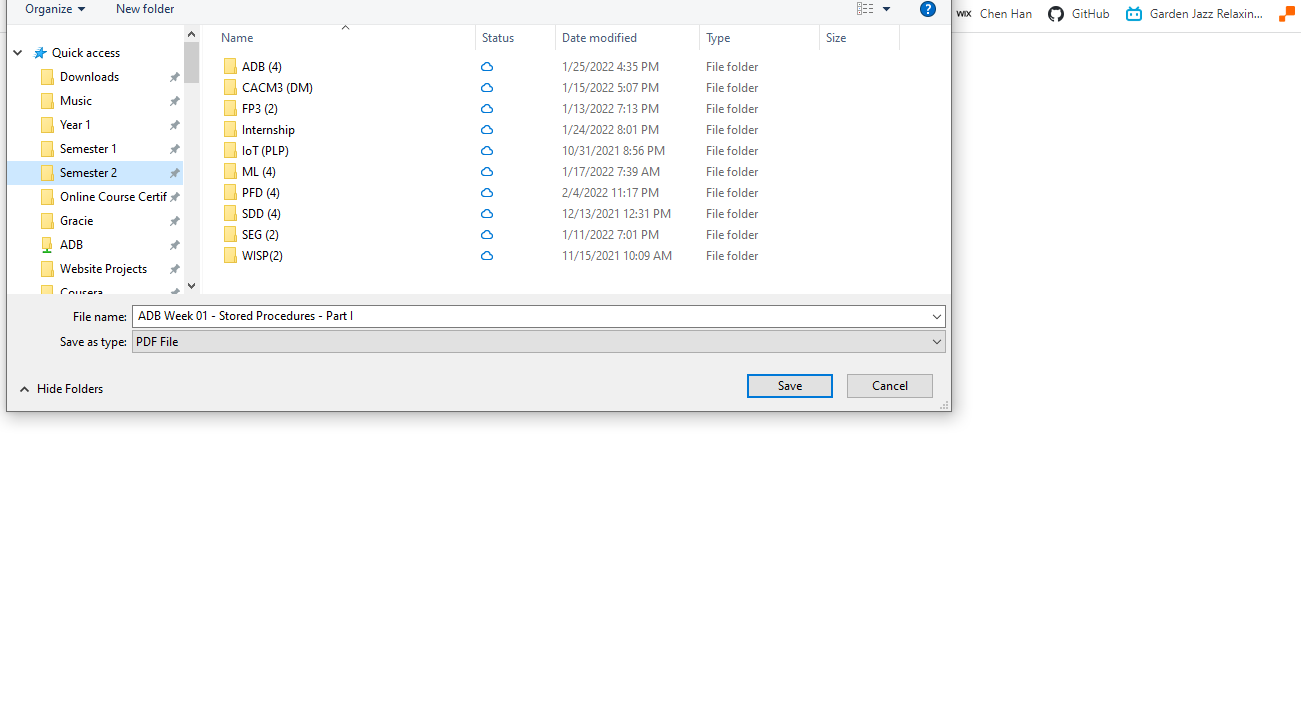
modules.Add("learningmaterialadb");

displayAllBlobs(modules[0]);

Graphical user interface, text, application, email

Description automatically generated

After clicking on the link, a download dialog has appeared.



2. How can a Tutor use this feature to upload the learning material?

In the WebRole1

This function is to be embeded to a upload button and is triggered whenever a tutor press it. It takes in two parameters, the moduleName (which can be obtained by the web application once the Tutor has enter the Module that he taught), as well as the filePath, which is the absolute path in the Tutor’s computer to the file that he wanting to upload.

public void UploadMaterial(string moduleName, string filePath)

{

CloudBlobClient blobClient;

CloudStorageAccount cloudStorageAccount;

cloudStorageAccount = CloudStorageAccount.Parse(

CloudConfigurationManager.GetSetting("myConnectionString"));

blobClient = cloudStorageAccount.CreateCloudBlobClient();

CloudBlobContainer container = blobClient.GetContainerReference(moduleName);

container.CreateIfNotExists();

container.SetPermissions(new BlobContainerPermissions

{

PublicAccess = BlobContainerPublicAccessType.Blob

});

string[] fileNameString = filePath.Split('/');

string reference = fileNameString[fileNameString.Count() - 1];

CloudBlockBlob blockBlob = container.GetBlockBlobReference(reference);

using (var fileStream = System.IO.File.OpenRead(filePath))

{

blockBlob.UploadFromStream(fileStream);

Response.Write("Successfully uploaded file: " + reference);

Response.Write("<br/>");

Response.Write("Into container: " + moduleName);

Response.Write("<br/>");

};

The following code demonstrates the idea of Tutor uploading a learning material. First by creating a list consist of a list of module name that would automatically generated when a Tutor chose which module to upload the file in. (E.g if the ModuleName is Machine Learning, the container name will be referred as “learningmaterialml”). The containername and the path to the file for uploading are then passed as parameters to the function.

//This can be specific by the tutor:

// e.g. Tutor select Course - Module - Upload

List<string> modules = new List<string>();

modules.Add("learningmaterialadb");

modules.Add("learningmaterialsdd");

modules.Add("learningmaterialml");

UploadMaterial(modules[0], "D:/ADB Blob/ADB/ADB Week 01 - Stored Procedures - Part I.pdf");

UploadMaterial(modules[1], "D:/ADB Blob/SDD/01-2 - Introduction to Agile(1).pptx");

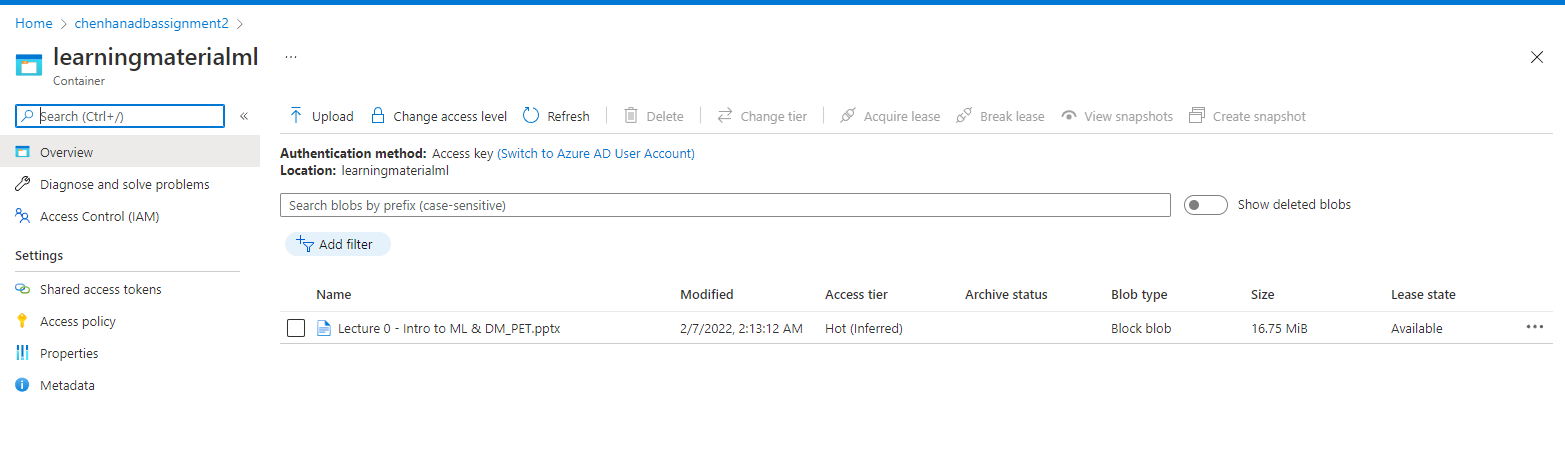
UploadMaterial(modules[2], "D:/ADB Blob/ML/Lecture 0 - Intro to ML & DM\_PET.pptx");

This is the Response page that shown up after a successful uploading process.

Graphical user interface, timeline

Description automatically generated with medium confidence

This is where the ML Lecture 0 powerpoint slides are stored in the Azure Blob Storage.



Upon uploading the file, a SQL INSERT query is also generated and run, the example of the adding a new Machine Learning learning material query is below:

SELECT 3, 'Lecture 1', 'Mr Toh', 'Machine Learning', 'Lecture', 'Please look through the lecture before the lesson starts',

'https://chenhanadbassignment2.blob.core.windows.net/learningmaterialml/Lecture 0 - Intro to ML & DM\_PET.pptx',

'2022/2/7 5:00:00 PM','2022/2/8 10:00:00 am','No', 3, 3

Where the filename will be the URL of the downloadable file on the Azure Blob Storage, as well as some other values can be filled using C# codes such as DateTimeUploaded.

## (iv) Keerthivasan

### (iv.i) Implementation of Denormalization in Database

**Quiz**(QuizID, <ModuleID>,HostID, Title, Summary, Type, Percentage, Published, CreatedAt, StartsAt, EndsAt, ModuleName)

There is denormalization between the Quiz entity and the Module entity. This denormalization is done by duplication of non-key attribute of ModuleName from module to quiz with the specific aim of reducing or removing joins as the ModuleName attribute is often used in queries to retrieve data as the quiz and module entities are normally frequently referenced together. Other non-key attributes in the module entity such as credit unit will not be needed to be duplicated as it will not have much use in the quiz entity or help with removing of joins.

**QuizQuestion**(QuestionID, <QuizID>, CategoryID, Type, Level, Score, Content, Category)

There is denormalization between the QuizQuestion entity and the QuestionCategory entity. This is a special case of a one-to-many relationship being a lookup table. A lookup table is a table that contains a code an a description and since this lookup table is used in frequent and critical queries and the description in this case the category is very unlikely to change duplication of the category attribute to quizquestion is effective as it will reduce joins if queries require the category of the question.

**StudentAnswer**(<StudentID>,<QuizID>,<QuestionID>,<ChoiceID>, Answer, Score, CorrectChoice)

There can be denormalization done between the StudentAnswer entity and the QuestionAnswer entity. This denormalization is done by duplication of non-key attribute of CorrectChoice from QuestionAnswer to quiz with the specific aim of reducing or removing joins as the CorrectChoice attribute is often used in queries when students may want to check what questions they had gotten correct and wrong this will prevent the need of a join between StudentAnswer and QuestionAnswer.

**StudentQuiz**(<StudentID>, <QuizID>, Status, TotalScore, StartedAt, FinishedAt, Remarks, <ModuleID>, ModuleName, Title)

Denormalization can be done for the StudentQuiz entity and the Module entity. This can be done by introducing a direct relationship between the Module and StudentQuiz entities. By doing so we can duplicate both foreign key and non-key attributes from module to StudenQuiz in this scenario they are ModuleID and ModuleName. These are introduced into the StudentQuiz table with the aim to reduce or remove joins as the ModuleName attribute is often used in queries to retrieve data such as finding the grade of students for quiz 1 in the machine learning module.

There is also denormalization done between the StudentQuiz and Quiz entity. This is done by by duplication of non-key attribute of Title from Quiz to StudentQuiz with the specific aim of reducing or removing joins as the Title attribute is often used in queries to retrieve data to find student answers of a specific quiz. Hence, it will help reduce joins.

### (iv.ii) Data Placement Strategy

There are four different data placement strategies to consider centralised, replicated, partitioned and hybrid. Centralised strategy would mean that all the data is at one site only as a result retrieval costs will be high, reliability will be low, availability will be poor and locality of reference will be low making it very unreliable and this strategy would make it not a true distributed database system and since our system will have to be a distributed database system which is able to ensure optimal performance and accessibility while maintaining a high available, resilient, scalable and low cost and consider support for overseas branches this strategy is out of the picture. A replicated strategy would keep a complete copy of a database at each node which allows for maximum locality of reference, reliability, data availability, and processing load distribution but as a result there are high storage costs and costs of updates is high since every site must receive the updates but since the online quiz function and tables require many updates from various users it would be extremely costly to have a replicated data placement strategy as all the databases will be constantly updated with the new records. Hence, the most efficient strategy to use would be partitioning. In specific, I will be doing horizontal fragmentation based on each module offered in each branch. As a result this will provide more operation flexibility, improve availability, performance and scalability as business queries will be much more efficient when looking at partitioned tables of each module instead of the whole table which includes all the data. This will also reduce opertational costs as the computing needed will not be as high. Furthermore, when updating records it will only need to update the records in the specific partition instead of the whole table which as a result improves operational costs aswell. It is also very important to have efficient and realiable updating and retrieval as online quizzes have time limits on them so updates need to be realialbe and fast or there might be errors/inconsistencies. An example of this strategy is the StudentQuiz table can be portioned with horizontal fragmentations based on the ModuleID. Hence, the partions will be by ModuleID allowing for easy updates and retrieval of answers for quizzes by modules as a result when doing business queries there will not be a need to specify which module the record is under making queries even more efficient and data retrieval and updates faster. These fragements can also be named after their modules as that is the main partion. Such as StudentQuizMachineLearning. The table below is an example of how it would look like. \*Assuming ModuleID 0001 = Maching Learning and 0002 = Advanced Databases.

Table

Description automatically generated

### (iv.iii) Cloud Storage Strategies

The Quiz and StudentQuiz table can be stored in the SQL format as the records in the tables will not change and will remain the same as they are strictly defined tables and as a result the data is highly structed and the structure will not change furthermore there is not much data that needs to be constantly updated hence the scale-out capability that NoSQL offers will not be needed. Hence the SQL database which follows the ACID properties Atomicity, Consistency, Isolation and Durability would be better as due to consistency the data written to the database must be valid according to the refined rules and structure. Even though SQL scales vertically, since there is not much data needed to be updated or kept sql being able to maintain the integrity of the data is enough. Furthermore, since there is a high degree of relations between the Module table and Quiz table and the Quiz table and StudentQuiz table, StudentQuiz Table and Module Table and the StudentQuiz table and Student Table it would be better to use SQL for these tables.

On the other hand, for the QuizQuestion, QuestionCategory, QuestionAnswer and StudentAnswer tables it would be much more efficient and better to use NoSQL. In particular, document databases such as MongoDB with JSON documents. This is because it supports high performance, high availability and automatic scaling. Since there is not a set number of questions and answers and the large amount of student answers there will be a lot of data and the more the number of quizzes the amount of data scales aswell because of this document database is a good choice as it scales horizontally and very efficiency across systems and locations making it possible to accommodate large storages of dsitrubuted data furthermore since NoSQL database uses dynamic schema it provides more flexibility where one collection can hold different documents.

### (iv.iii.i) SQL

**Quiz**(QuizID, <ModuleID>,HostID, Title, Summary, Type, Percentage, Published, CreatedAt, StartsAt, EndsAt, ModuleName)

**StudentQuiz**(<StudentID>, <QuizID>, Status, TotalScore, StartedAt, FinishedAt, Remarks, <ModuleID>, ModuleName, Title)

Since Quiz and StudentQuiz have a relationship with the module table and will be used for partitioning as mentioned earlier, using SQL is useful in terms of queries as it can create very complex queries using join and other query capabilities which allows for complex query capabilities after partitioning as most business queries related to quiz will come from these 2 tables. Furthermore, since the structure will not change the use of SQL for storing these tables are better SQL also provides reliability which is essential in this case as if the FinishedAt value for StudentQuiz is updated inconsistently it could lead to mistakes and false errors and SQL’s consistency and reliability prevents this.

### (iv.iii.ii) NoSQL

**QuestionAnswer**(<QuestionID>, <QuizID>, ChoiceID, CorrectChoice, Content)

**QuizQuestion**(QuestionID, <QuizID>, CategoryID, Type, Level, Score, Content, Category)

**QuestionCategory**(CategoryID, <QuestionID>, Category)

**StudentAnswer**(<StudentID>,<QuizID>,<QuestionID>,<ChoiceID>, Answer, Score, CorrectChoice)

The StudentAnswer table is used to store all the answers that the student has chosen for a quiz, if there is 20 questions in a quiz that would be 20 records of answers for each student this amount would increase exponentially with the increasing amount of students taking the quiz as a result a large amount of records will be created that is why NoSQL such as document database is very good here as NoSQL databases scale horizontally very efficiently which will help with storage of records. Since NoSQL database uses dynamic schema it provides more flexibility where one collection can hold different documents such as the QuizQuestion will have one document with all the information including the QuestionAnswers as an array and the StudentAnswer will be another document which references to the array of QuestionAnswerID.

### (iv.iv) Design/Schema on Cloud Storage Used

##### (iv.iv.i) SQL:

Graphical user interface

Description automatically generated

##### (iv.iv.ii) NoSQL(Document Storage):

Graphical user interface, text, application, Teams

Description automatically generated

Graphical user interface, text, application

Description automatically generated

### (iv.v) Implementation of business queries after denormalization & fragmentation

1. Tutor Mr Ben would like to know how many students had gotten A for Quiz 1 uploaded for the Machine Learning module.

**BEFORE**

Select COUNT(\*) As “Number Of Students Who Got A”

FROM Module m INNER JOIN Quiz q on m.ModuleID = q.ModuleID

INNER JOIN StudentQuiz sq ON q.QuizID = sq.QuizID

WHERE m.ModuleName = “Machine Learning” AND q.Title = “Quiz 1” AND sq.TotalScore >= 80.

**AFTER**

Select COUNT(\*) As “Number Of Students Who Got A”

FROM StudentQuiz sq WHERE sq.ModuleName = “Machine Learning” AND sq.Title = “Quiz 1” and sq.TotalScore >= 80.

As seen from the SQL query above due to denormalization I was able to completely remove the needs of joins to achieve this query where as before denormalization I needed two joins to achieve this query. Hence by these denormalizations I was able to remove joins and improve query performance by speeding up retrieval and minising computing overheads.

1. A student with the ID of S10205505 wants to check the results of their Quiz 1 in the machine learning module and see his answers and the correct answers to see which questions he had gotten wrong and correct and their respective scores

**BEFORE**

SELECT sa.QuestionID As “Question Number”, sa.Answer as “Student Answer”, qa.CorrectChoice as “Correct Answer”, sa.Score as “Student Score” FROM Module m INNER JOIN Quiz q on m.ModuleID = q.ModuleID

INNER JOIN StudentQuiz sq ON q.QuizID = sq.QuizID INNER JOIN StudentAnswer sa ON sq.SQuizID = sa.SQuizID INNER JOIN QuestionAnswer qa ON qa.ChoiceID = sa.ChoiceID WHERE m.ModuleName = “Machine Learning” AND q.Title = “Quiz 1” AND sq.StudentID = "S10205505" ORDER BY sa.QuestionID.

**AFTER**

SELECT sa.QuestionID As “Question Number”, sa.Answer as “Student Answer”, sa.CorrectChoice as “Correct Answer”, sa.Score as “Student Score” FROM StudentAnswer sa INNER JOIN StudentQuiz sq WHERE sq.ModuleName = “Machine Learning” AND sq.Title = “Quiz 1” AND sq.StudentID = “S10205505” ORDER BY sa.QuestionID

**AFTER PARTITONING**

SELECT sa.QuestionID As “Question Number”, sa.Answer as “Student Answer”, sa.CorrectChoice as “Correct Answer”, sa.Score as “Student Score” FROM StudentAnswer sa INNER JOIN StudentQuiz sq WHERE sq.Title = “Quiz 1” AND sq.StudentID = “S10205505” ORDER BY sa.QuestionID

As seen from the SQL query due to denormalization I was able to reduce the amount of joins needed to carry out the query previously there was a need of 4 inner joins to complete the query now after denormalization there is only a need of 1 inner join. Hence by these denormalizations I was able to reduce joins and improve query performance by speeding up retrieval and minising computing overheads. After partitioning there is no need to specify the module names as each fragment is a module and in this scenario we are assuming that it is the machine learning fragment.

1. Tutor Mr Ben would like to know the average score of the students for Quiz 1 uploaded for the Machine Learning module.

**BEFORE**

Select AVG(sq.TotalScore) As “Average Marks”

FROM Module m INNER JOIN Quiz q on m.ModuleID = q.ModuleID

INNER JOIN StudentQuiz sq ON q.QuizID = sq.QuizID

WHERE m.ModuleName = “Machine Learning” AND q.Title = “Quiz 1”.

**AFTER**

Select AVG(sq.TotalScore) As “Average Marks”

FROM StudentQuiz sq WHERE sq.ModuleName = “Machine Learning” AND sq.Title = “Quiz 1”

As seen from the SQL query due to denormalization I was able to completely remove the need of inner joins from 2 to 0. Therefore, by these denormalizations I was able to remove joins and improve query performance by speeding up retrieval and minising computing overheads.