

Project 3 Tracking Document

T/Th 9:15am - Group 1

DUE DATE: Sunday, December 10, 2023 11:59 PM EST

Team Lead: Aleksandra Georgievskia

Project Lead Backup: Aryeh Richman

Team Members: Nicholas Kong, Aleksandra Georgievskia, Aryeh Richman, Edwin Wray, Sigalita Yakubova, Ahnaf Ahmed

Quicklinks:

[Project 2.5 Group Google Drive](#)

Specifications

Project Requirements:

- Use of the swing app
 - One of the requirements is that once we finish creating the db and putting the data in he gives us 3 or 4 queries that we should be able to run on on the database (propositions) and then asks us to come up with 10 or more as a group (think the medium queries project 1):
 - 1. Show all instructors who are teaching in classes in multiple departments
 - 2. How many instructors are in each department?
 - 3. How many classes are being taught that semester grouped by course and aggregating the total enrollment, total class limit and the percentage of enrollment.
 - 4. 10 more queries of your choice and their proposition.
 - Load all 10 queries into the JDBC app
- Upload the final design your database as backup file (.bak)
- Use the schema Udt & create User Defined Datatypes where possible

Mp4 Presentation Requirements:

- Database:
- Project Lifecycle:

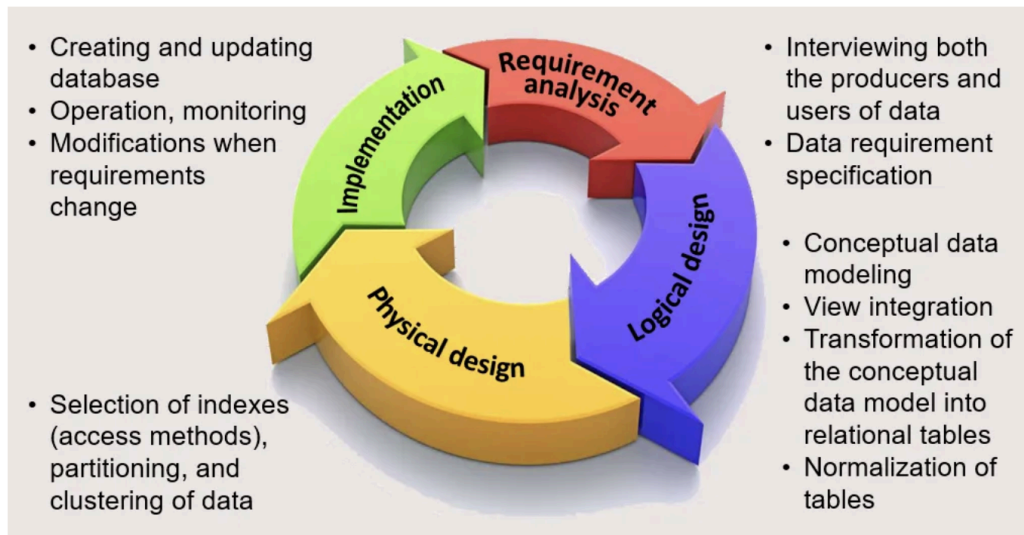
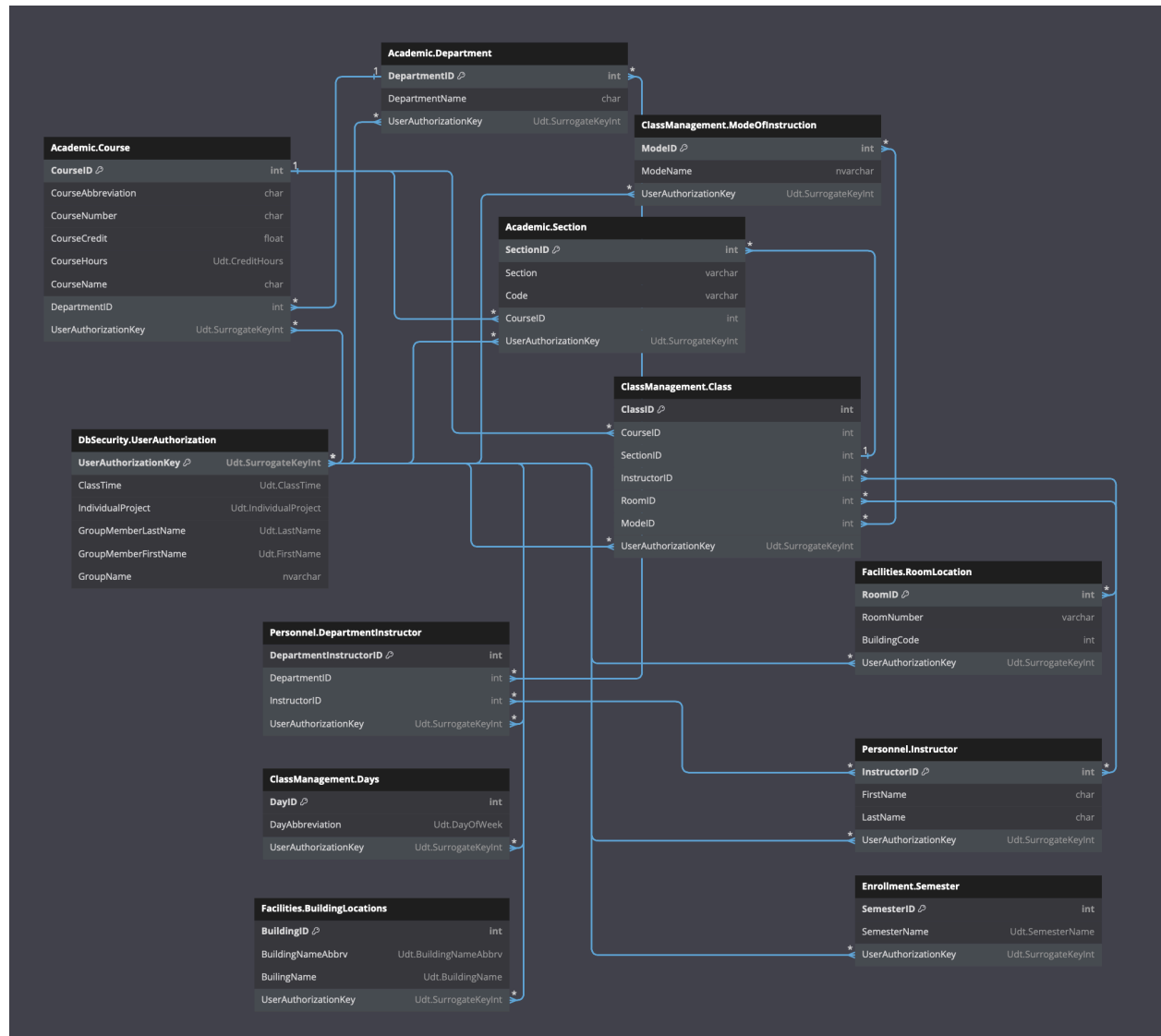
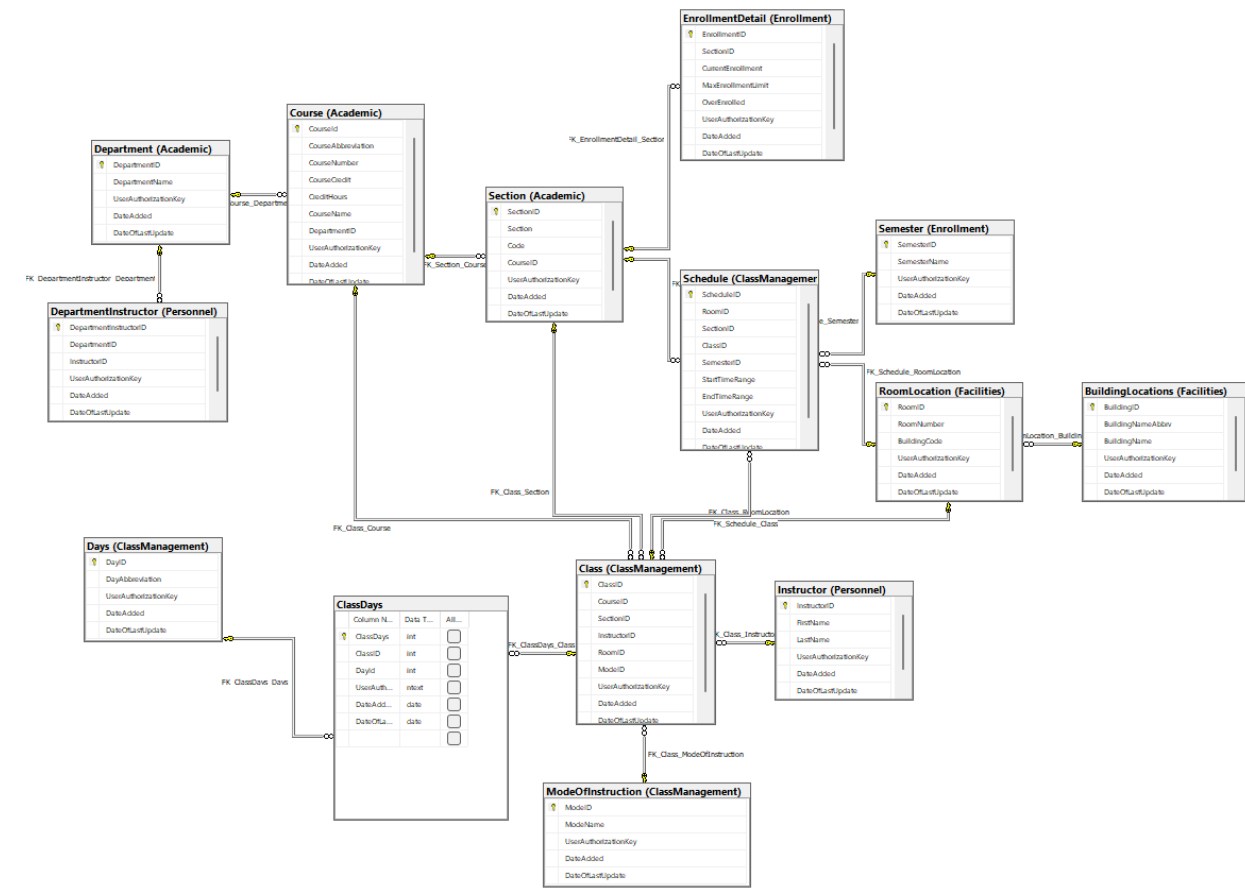


Figure 1 Database life cycle
according to Teorey et al. (2011: 4)

PDM



LDM



Project Package Requirements:

- Submit package as a VHDX file with folder directory
- Mp4 presentation
- Consolidated Powerpoint Slides
- Redgate SQL Doc
- To-Do & Gantt .xlsx files
- Project Tracking Document .docx file
-

DB Tables Plan

Table names:

1. Department
2. Class
3. Course
4. Instructor
5. BuildingLocation

6. RoomLocation
7. ModeOfInstruction
8. DepartmentInstructor
9. Schedule
10. Semester
11. Section
12. EnrollmentDetail
13. ClassDay
14. Day

Table Tiers:

- 1st
 - Aryeh: Department - DONE
 - Aleks: Instructor - DONE
 - Sigi: Semester - DONE
 - Edwin: BuildingLocation
 - Nicholas: ModeOfInstruction - DONE
 - Ahnaf: Day- DONE
- 2nd
 - Aleks: Course - DONE
 - Aryeh: DepartmentInstructor
 - Nicholas: RoomLocation
- 3rd
 - Sigi: Section - DONE
 - Ahnaf: EnrollmentDetail
 - Edwin: Class - DONE
- 4th
 - Nicholas: Schedule
 - Aryeh: ClassDay

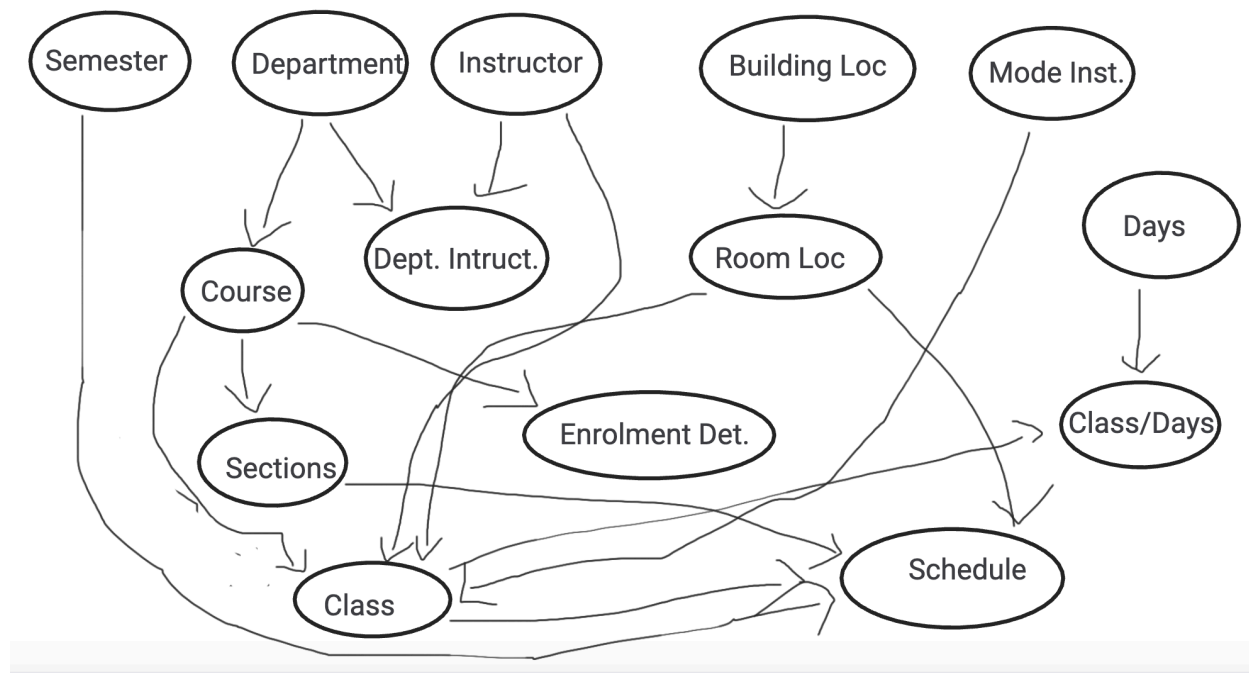


Table #1

[Academic].[Department] Table

DepartmentID PRIMARY KEY

DepartmentName

Table #2

[ClassManagement].[Class] Table

ClassID PRIMARY KEY

FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)

FOREIGN KEY (InstructorID) REFERENCES Instructor(InstructorID)

FOREIGN KEY (RoomLocationID) REFERENCES RoomLocation(RoomLocationID)

FOREIGN KEY (ModeOfInstructionID) REFERENCES ModeOfInstruction(ModeOfInstructionID)

SUGGESTION: Redundant Foreign Key in Class Table: The Class table includes a foreign key to the Courses table, but this relationship is also indirectly established through the Sections table. Each class is linked to a specific section, and each section is linked to a course. This redundancy might be unnecessary unless there is a specific requirement for it.

To address the redundancy in your Class table, you can remove the direct foreign key relationship to the Course table. Since each class is associated with a specific section, and each section is already linked to a course, the course information can be accessed through the Section-Course relationship. Here's how you can modify the tables:

1. Modify the Class Table: Remove the FOREIGN KEY (CourseID) from the Class table. This will eliminate the direct link from a class to a course.

2. Rely on Section-Course Relationship: To find out which course a class belongs to, you would first identify the section associated with that class, and then find the course linked to that section.

Revised Class Table:

ClassID (PK)

FOREIGN KEY (InstructorID) REFERENCES Instructor(InstructorID)

FOREIGN KEY (RoomLocationID) REFERENCES RoomLocation(RoomLocationID)

FOREIGN KEY (ModeOfInstructionID) REFERENCES ModeOfInstruction(ModeOfInstructionID)

FOREIGN KEY (SectionID) REFERENCES Sections(SectionID)

Workflow to Determine Course for a Class:

1. Identify the SectionID associated with a given ClassID.

2. Look up the corresponding CourseID in the Sections table using the identified SectionID.

3. Retrieve the course details from the Course table using the found CourseID.

This approach maintains the integrity of the database while avoiding unnecessary redundancy. It also aligns with normalization principles, ensuring that each piece of information is stored only once and can be accessed through clearly defined relationships.

Table #3

[Academic].[Course] Table

CourseID (PK)

CourseCode

CourseName

CourseCredit (Needs Check Constraint)

CreditHours (Needs Check Constraint) -> should be positive, possibly a UDT

CourseDescription

FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID)

UDT Suggestions

- CreditHours Type: If credit hours in your courses have certain constraints (like being a positive number, possibly with a maximum limit), a UDT can enforce these rules wherever credit hours are used in the database.

Table #4 - DONE

[Personnel].[Instructor] Table

InstructorID (PK)

FirstName

LastName

Table #5

[Facilities].[BuildingLocation] Table

LocationID (PK) -> Should be renamed "BuildingCode" -> MAKE BuildingCode a UDT

BuildingName

UDT Ideas:

- BuildingCode Type: Similar to room codes, if building codes have a unique format (like a specific number of characters or a combination of letters and digits), a UDT would enforce this consistency across the database.
 - Make UDT constraint that the abbrevs are 2 characters
 - Could add the full names of the building in the UDT definition using cases ex: if "KY" fill BuildingName with "Kiely Hall"

Table #6

[Facilities].[RoomLocation] Table

RoomID (PK)

RoomNumber (Needs Check Constraint)

FOREIGN KEY (LocationID) REFERENCES BuildingLocation(LocationID)

SUGGESTION: Foreign key should reference "BuildingCode" and we should change "LocationID" to Building Code so it's more clear what type of location it is (Self documenting)

Table #7

[ClassManagement].[ModeOfInstruction] Table

ModeID (PK)

ModeName

Table #8

[Personnel].[DepartmentInstructor] (Bridge Table)

DepartmentInstructorID (PK)

FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID)

FOREIGN KEY (InstructorID) REFERENCES Instructor(InstructorID)

Table #9

[ClassManagement].[Schedule] Table

ScheduleID (PK)

DayOfWeek -> Perhaps break this out into its own table since column contains list

StartTime -> Perhaps consolidate into a TimeSlot UDT

EndTime -> Perhaps consolidate into a TimeSlot UDT

FOREIGN KEY (RoomLocationID) REFERENCES RoomLocation(RoomLocationID)

FOREIGN KEY (SectionID) REFERENCES Sections(SectionID)
FOREIGN KEY (ClassID) REFERENCES Class(ClassID)
FOREIGN KEY (SemesterID) REFERENCES Semester(SemesterID)

UDT Suggestion:

- **ClassTime Type:** For representing class times, instead of using separate fields for start and end times, you could create a UDT that encapsulates a time slot as a single entity. This can be useful for ensuring that time slots are always defined consistently (e.g., with a start and end time).

UPDATE: prof heller had this in the specs already so definitely the ClassTime should be a UDT

```
90 CREATE TYPE [Udt] . [ClassTime] FROM nchar(5) NOT NULL
91 GO
```

SUGGESTIONS for normalizing DayOfWeek:



ChatGPT

The Day column from the original flat file shows class days using abbreviations for the days of the week, with some entries listing multiple days. Here's how you can normalize this data and create a User-Defined Data Type (UDT) to improve this design:

Normalizing the Day Information

To normalize this data, you would create a separate table to store individual day entries. This would involve:

1. **Creating a Days Table:** A table that includes a unique identifier for each day of the week and the corresponding abbreviation.

Days Table:

- DayID (PK)
- DayAbbreviation (e.g., M, T, W, TH, F, S, SU)

1. **Creating a ClassDays Bridge Table:** Since classes can meet on multiple days, you'd create a many-to-many relationship between the classes and days.

ClassDays Table (Bridge Table):

- ClassID (FK)
- DayID (FK)

Each row in this table would represent a class meeting on a particular day of the week.

To incorporate the normalization of the DayOfWeek information and the bridge table concept into your existing Schedule table, you would remove the DayOfWeek column from the Schedule table and instead reference the bridge table that connects class sessions to days of the week.

Here's what you would change:

1. **Remove the DayOfWeek Column:** The Schedule table no longer needs the DayOfWeek column since this information will be stored in the bridge table.
2. **Create a ClassDays Table:** This bridge table will hold the relationship between the ClassID and the DayID.

ClassDays Table (Bridge Table):

- ClassDaysID (PK)
- ClassID (FK) REFERENCES Class(ClassID)
- DayID (FK) REFERENCES Days(DayID)

Each row in this table represents an association of a class with a day of the week on which it meets.

1. **Use the Days Table:** The Days table will have a list of days with unique identifiers.

Days Table:

- DayID (PK)
- DayAbbreviation CHAR(2) CHECK (DayAbbreviation IN ('M', 'T', 'W', 'TH', 'F', 'S', 'SU'))

So, the normalized Schedule table would look like this:

Schedule Table:

- ScheduleID (PK)
- StartTime
- EndTime
- FOREIGN KEY (RoomLocationID) REFERENCES RoomLocation(RoomLocationID)
- FOREIGN KEY (SectionID) REFERENCES Sections(SectionID)
- FOREIGN KEY (ClassID) REFERENCES Class(ClassID)
- FOREIGN KEY (SemesterID) REFERENCES Semester(SemesterID)

With these changes, the UDT `DayAbbreviation` would not directly become a column name in the Schedule table. Instead, the `DayAbbreviation` would be the data type of the Day column in the Days table, ensuring that only valid day abbreviations can be entered.

To reconstruct the schedule with the days of the week, you would perform a query joining the Schedule table, the ClassDays bridge table, and the Days table. For instance:

```
sql Copy code

SELECT
    s.ScheduleID,
    d.DayAbbreviation,
    s.StartTime,
    s.EndTime
FROM
    Schedule s
JOIN
    ClassDays cd ON s.ClassID = cd.ClassID
JOIN
    Days d ON cd.DayID = d.DayID
WHERE
    s.ClassID = @ClassID; -- assuming @ClassID is a parameter for the specif
```

This query would give you the days and times for a given class. It might require multiple rows for each ScheduleID, one for each day of the week the class meets.

UDT Suggestions:

- Create a UDT for class day abbreviations

Creating a UDT for Day Abbreviations

You could create a UDT for the day abbreviations that would only allow the specific abbreviations used in your dataset. For example, if you're using SQL Server, you could create a UDT like this:

```
sql Copy code

CREATE TYPE dbo.DayOfWeek AS TABLE
(
    Day CHAR(2) CHECK (Day IN ('M', 'T', 'W', 'TH', 'F', 'S', 'SU'))
);
```

This UDT could then be used to ensure that only valid day abbreviations are stored in your Days table or when accepting input for class schedules.

Implementing the UDT in the Normalization

When you insert or update records in the ClassDays bridge table, you would ensure that each day abbreviation matches one of the entries in your `DayOfWeek` UDT. This can be done by defining the Day column in the Days table to be of the type `DayOfWeek`.

Here's a potential implementation:

```
sql Copy code  
  
CREATE TABLE Days (  
    DayID INT PRIMARY KEY,  
    DayAbbreviation DayOfWeek -- Using the UDT here  
);
```

Using a UDT in this way ensures that every entry for a day in your database is valid according to the definitions you've set out. It also simplifies data validation and integrity checks.

This approach allows for flexibility in scheduling (since classes can meet on any combination of days) and maintains the normalization of the database. It would require a bit more work when querying the database—you would need to join the classes with the ClassDays and Days tables to get the full schedule—but it ensures that your data remains consistent and well-structured.

Table #10

[Academic].[Semester] Table

SemesterID (PK)

SemesterName

Year

QUESTION -> There is no year column in the original flat file? Will this field be empty?

Table #11

[Academic].[Section] Table

SectionID (PK)

SectionCode

FOREIGN KEY (CourseID) REFERENCES Course(CourseID)

Table #12

[Enrollment].[EnrollmentDetail] Table

EnrollmentID (PK)

CurrentEnrollmentLimit (Needs Check Constraint) -> should not exceed MaxEnrollmentLimit

MaxEnrollmentLimit (Needs Check Constraint)

FOREIGN KEY (CourseID) REFERENCES Course(CourseID)

-

Table #13

[ClassManagement].[ClassDay] - Bridge Table

FOREIGN KEY (CourseID) REFERENCES Course(CourseID)

See the suggestions under table 9 for how to implement

Table #14

[ClassManagement].[Day]

See the suggestions under table 9 for how to implement

USER Defined Datatypes

About:

User-Defined Data Types (UDTs) in database systems are custom data types that you can define to suit specific needs in your database. UDTs are particularly useful when you have a specific type of data that appears repeatedly across various tables but doesn't fit neatly into standard data types like integers, strings, or dates. Implementing UDTs can enhance the readability, maintainability, and integrity of your database.

To implement a UDT, you typically use the CREATE TYPE statement in SQL, and then use this type when defining columns in your tables. For example, if you create a RoomCode type, when defining the Room table, you would use the RoomCode type for the room code column.

Suggestions:

3 Queries Provided by Prof. for Database

1. Show all instructors who are teaching in classes in multiple departments – Edwin

```
SELECT DI.InstructorID, I.FirstName, I.LastName
FROM Personnel.DepartmentInstructor AS DI
|   INNER JOIN Personnel.Instructor AS I
|   |   ON DI.InstructorID = I.InstructorID
GROUP BY DI.InstructorID, I.FirstName, I.LastName
HAVING COUNT(DISTINCT DI.DepartmentID) > 1
ORDER BY DI.InstructorID;
```

Results	Messages		
	InstructorID	FirstName	LastName
1	10	Dorian	Abreu
2	17	David	Adler
3	28	Christina	Alaimo
4	56	Yoshio	Aomori
5	59	Maria	Argyros
6	62	Alice	Artzt
7	119	David	Berkman
8	133	Brenda	Biddle
9	139	Jeffrey	Bird
10	150	Richard	Bodnar
11	188	Joshua	Brumberg
12	201	Harvey	Burstein
13	205	Fred	Cadieu
14	209	Peter	Calandra
15	220	Nora	Carr
16	223	Clare	Carroll
17	245	Riddhi	Chauhan
18	259	Yu	Chen
19	270	Euna	Cho
20	285	Yves	Cloarec
21	287	Joseph	Cohen
22	299	Lewis	Cook
23	306	Catherine	Cordeiro
24	314	Georges	Courtadon
25	315	Sarah	Covington
26	351	David	Debora
27	361	John	Dennehy
28	386	Antonio	Donato
29	393	Abigail	Doukhan
30	416	Omri	Elisha
31	421	Robert	Engel
32	434	Jeffrey	Farber

2. How many instructors are in each department? – Edwin

```

SELECT D.DepartmentID, D.DepartmentName, COUNT(DI.InstructorID) AS NumberOfInstructors
FROM Academic.Department AS D
    LEFT JOIN Personnel.DepartmentInstructor AS DI
        ON D.DepartmentID = DI.DepartmentID
GROUP BY D.DepartmentID, D.DepartmentName
ORDER BY D.DepartmentID;

```

	DepartmentID	DepartmentName	NumberOfInstructors
1	1	ACCT	58
2	2	AFST	1
3	3	AMST	2
4	4	ANTH	32
5	5	ARAB	2
6	6	ARTH	12
7	7	ARTS	64
8	8	ASTR	3
9	9	BALA	10
10	10	BIOCH	2
11	11	BIOL	62
12	12	BUS	19
13	13	CERT	0
14	14	CESL	3
15	15	CHEM	50
16	16	CHIN	6
17	17	CLAS	6
18	18	CMAL	1
19	19	CMLIT	23
20	20	CO-OP	1
21	21	CSCI	53
22	22	CUNBA	0
23	23	DANCE	11
24	24	DRAM	22
25	25	EAST	8

3. How many classes that are being taught that semester grouped by course and aggregating the total enrollment, total class limit and the percentage of Enrollment. – Edwin

```

SELECT DISTINCT
    C.CourseID, C.CourseName
    , SUM(DISTINCT CS.ClassID) AS TotalClasses
    , E.CurrentEnrollment AS CurrentEnrollment
    , E.MaxEnrollmentLimit AS MaxEnrollmentLimit
FROM
    ClassManagement.Class AS CS
    JOIN Academic.Course AS C
        ON CS.CourseID = C.CourseID
    LEFT JOIN Academic.Section AS S
        ON C.CourseID = S.CourseId
    LEFT JOIN Enrollment.EnrollmentDetail AS E
        ON S.SectionID = E.SectionID
GROUP BY
    C.CourseID, C.CourseName, E.CurrentEnrollment, E.MaxEnrollmentLimit
ORDER BY
    C.CourseID;

```

	CourseID	CourseName	TotalClasses	CurrentEnrollment	MaxEnrollmentLimit
1	1	Fin & Mgr Acct	6	20	22
2	1	Fin & Mgr Acct	6	21	22
3	1	Fin & Mgr Acct	6	22	22
4	2	Int Theo & Prac Acct 1	114	20	30
5	2	Int Theo & Prac Acct 1	114	29	55
6	2	Int Theo & Prac Acct 1	114	30	30
7	2	Int Theo & Prac Acct 1	114	39	55
8	2	Int Theo & Prac Acct 1	114	42	55
9	2	Int Theo & Prac Acct 1	114	45	45
10	2	Int Theo & Prac Acct 1	114	47	55
11	2	Int Theo & Prac Acct 1	114	54	55
12	2	Int Theo & Prac Acct 1	114	55	55
13	3	Intro Theo & Prac Acct 2	180	16	55
14	3	Intro Theo & Prac Acct 2	180	17	55
15	3	Intro Theo & Prac Acct 2	180	19	30
16	3	Intro Theo & Prac Acct 2	180	41	55
17	3	Intro Theo & Prac Acct 2	180	44	55
18	3	Intro Theo & Prac Acct 2	180	51	55
19	3	Intro Theo & Prac Acct 2	180	55	55
20	4	Inter Acct 1	228	29	30
21	4	Inter Acct 1	228	39	42
22	4	Inter Acct 1	228	40	42
23	4	Inter Acct 1	228	42	42
24	4	Inter Acct 1	228	43	46
25	4	Inter Acct 1	228	49	51

10 Queries for Database

1. List of Classes in a Specific Building: Retrieve all classes being held in a particular building, including the class name, instructor, and room number.
2. Instructor Schedule for the Semester: Display the complete schedule of a specific instructor for the semester, showing all classes they are teaching, the times, and the locations.
3. **Courses Offered by a Department: Show a list of all courses offered by a specific department, including course names and their credit hours.** – Edwin

```
DECLARE @DepartmentID INT;
-- Set the value of the specific department
SET @DepartmentID = 1;

SELECT @DepartmentID AS DepartmentID, D.DepartmentName
      , C.CourseId, C.CourseName, C.CreditHours
FROM Academic.Department AS D
     INNER JOIN Academic.Course AS C
      ON D.DepartmentID = C.DepartmentID
WHERE D.DepartmentID = @DepartmentID
```

Results		Messages			
	DepartmentID	DepartmentName	CourseId	CourseName	CreditHours
1	1	ACCT	1	Fin & Mgr Acct	3
2	1	ACCT	2	Int Theo & Prac Acct 1	4
3	1	ACCT	3	Intro Theo & Prac Acct 2	4
4	1	ACCT	4	Inter Acct 1	4
5	1	ACCT	5	Inter Acct 2	3
6	1	ACCT	6	Business Law 1	3
7	1	ACCT	7	Cost Acct	3
8	1	ACCT	8	Quant Techniq Pln & Ctrl	4
9	1	ACCT	9	Advcd Acct	4
10	1	ACCT	10	Auditing 1	4
11	1	ACCT	11	Auditing 2	3
12	1	ACCT	12	Acct Info Syst	3
13	1	ACCT	13	Microcomp Apps Acct	3
14	1	ACCT	14	Finan Stat Analys	3
15	1	ACCT	15	Business Law 2	3
16	1	ACCT	16	Business Law 2	3
17	1	ACCT	17	Business Law 3	3
18	1	ACCT	18	Fed & Ny State Tax	4
19	1	ACCT	19	Gov & Nfp Acct & Aud	3
20	1	ACCT	20	Seminar in Accounting	3
21	1	ACCT	21	Internship	3
22	1	ACCT	22	Issues In Mgmt Acct	3
23	1	ACCT	23	Adv Fin Acct Theory	3
24	1	ACCT	24	Adv Auditing Theory	3
25	1	ACCT	25	Comm & Accountants	3
26	1	ACCT	26	Adv Acct Info Sysms	3
27	1	ACCT	27	Adv Stdy In Bus Law	3
28	1	ACCT	28	Taxation Of Bus Ents	3
29	1	ACCT	29	State & Local Taxatn	3
30	1	ACCT	30	Govt Acct & Audit	3

4. Which Instructors teach courses with over 150 students enrolled? – Aryeh


```

1 SELECT DISTINCT CONCAT(I.FirstName, ' ', I.LastName) AS Professor
2 FROM Enrollment.EnrollmentDetail AS E
3     INNER JOIN Academic.Section AS S
4         ON S.SectionID = E.SectionID
5     INNER JOIN Academic.Course AS C
6         ON C.CourseID = S.CourseID
7     INNER JOIN Personnel.DepartmentInstructor AS DI
8         ON DI.DepartmentID = C.DepartmentID
9     INNER JOIN Personnel.Instructor AS I
10        ON DI.InstructorID = I.InstructorID
11 WHERE E.CurrentEnrollment > 150

```

Results Messages

Professor	
1	Emmet Gammada
2	Maria Stalias
3	Renee Goodwin
4	Karl Path
5	Jose Anadon
6	Riddhi Chauhan
7	Erica Doran
8	Jeffrey Farber
9	Anna Budd
1	Vivuk Inathuvu

5. Which classes have over enrollment and by how many students? - Aryeh

```

1 SELECT DISTINCT CONCAT(C.CourseAbbreviation, C.CourseNumber) AS Course,
2     C.CourseName,
3     S.SectionID,
4     E.CurrentEnrollment,
5     E.MaxEnrollmentLimit,
6     (E.CurrentEnrollment - E.MaxEnrollmentLimit) AS Overflow
7 FROM Academic.Course AS C
8     INNER JOIN Academic.Section AS S
9         ON C.CourseID = S.CourseID
10     INNER JOIN Enrollment.EnrollmentDetail AS E
11        ON S.SectionID = E.SectionID
12 WHERE E.CurrentEnrollment > E.MaxEnrollmentLimit

```

Course	CourseName	SectionID	CurrentEnrollment	MaxEnrollmentLimit	Overflow
1 ACCT 282	Inter Acct 2	834	55	53	2
2 ACCT 282	Inter Acct 2	5821	55	53	2
3 ACCT 282	Inter Acct 2	9388	55	53	2
4 ACCT 261	Business Law I	2139	46	44	2
5 ACCT 261	Business Law I	6326	46	44	2
6 ACCT 261	Business Law I	18513	46	44	2
7 ACCT 386	Quant Techniq Pln & Ctrl	2249	55	53	2
8 ACCT 386	Quant Techniq Pln & Ctrl	6436	55	53	2
9 ACCT 386	Quant Techniq Pln & Ctrl	18623	55	53	2
10 ACCT 322	Auditing 2	887	51	58	1
11 ACCT 322	Auditing 2	5854	51	58	1

6. Departmental Class Size Averages: Calculate the average class size for each department, based on current enrollment in all courses offered by the department.

7. Instructors Not Teaching This Semester: Identify instructors who are not assigned to any class for the current semester. -- Sigi

8. Most Popular Courses: Rank courses based on total enrollment for the semester, showing the most to least popular. --Sigi

9. Show all the courses being taught in a specific department - Sigi

```
57
58  /* =====
59  -- Author:      Sigalita Yakubova
60  -- Create date: 12/10/23
61  -- Proposition: Show all the courses being taught in a specific department
62  -- =====*/
63  DECLARE @DeptID INT = 1;
64
65  SELECT D.DepartmentID, D.DepartmentName, C.CourseName, C.CourseAbbreviation, C.CourseNumber, C.CourseId
66  FROM [Academic].Department AS D
67  INNER JOIN [Academic].Course AS C ON C.DepartmentID = D.DepartmentID
68  WHERE D.DepartmentID = @DeptID
69
```

Results						
	DepartmentID	DepartmentName	CourseName	CourseAbbreviation	CourseNumber	CourseId
1	1	ACCT	Fin & Mgr Acct	ACCT	100	1
2	1	ACCT	Int Theo & Prac Acct 1	ACCT	101	2
3	1	ACCT	Intro Theo & Prac Acct 2	ACCT	102	3
4	1	ACCT	Inter Acct 1	ACCT	201	4
5	1	ACCT	Inter Acct 2	ACCT	202	5
6	1	ACCT	Business Law 1	ACCT	261	6
7	1	ACCT	Cost Acct	ACCT	305	7
8	1	ACCT	Quant Techniq Pln & Ctrl	ACCT	306	8

10. Show all teachers in the Accounting Department whose first name starts with A - Sigi

```
70
71  /* =====
72  -- Author:      Sigalita Yakubova
73  -- Create date: 12/10/23
74  -- Proposition: Show all teachers in the Accounting Department whose first name starts with A
75  -- =====*/
76
77  SELECT D.DepartmentID, D.DepartmentName, I.FirstName, I.LastName
78  FROM [Personnel].DepartmentInstructor AS DI
79  INNER JOIN [Personnel].Instructor AS I ON I.InstructorID = DI.InstructorID
80  INNER JOIN [Academic].Department AS D ON D.DepartmentID = DI.DepartmentID
81  WHERE D.DepartmentName = 'ACCT' AND I.FirstName LIKE 'A%'
82
83
```

Results				
	DepartmentID	DepartmentName	FirstName	LastName
1	1	ACCT	Arthur	Adelberg
2	1	ACCT	Amy	David
3	1	ACCT	Arthur	Dignam
4	1	ACCT	Anita	Feisullin
5	1	ACCT	Anique	Qureshi
6	1	ACCT	Arthur	Silverman

11. Different Classes Taught by Same Instructor : Nicholas

```
50 /* =====
51 -- Author: Nicholas Kong
52 -- Create date: 12/10/23
53 -- Proposition: Show the same instructor who teaches from different departments
54 -- =====
55 */
56
57 SELECT D.DepartmentID, D.DepartmentName, I.FirstName, I.LastName
58 FROM [Personnel].DepartmentInstructor AS DI
59 INNER JOIN [Personnel].Instructor AS I ON I.InstructorID = DI.InstructorID
70 INNER JOIN [Academic].Department AS D ON D.DepartmentID = DI.DepartmentID
71 WHERE I.FirstName = 'David' AND I.LastName = 'Lahti'
72
73
74 ----- EXEC COMMANDS TO MANAGE THE DB -----
75
```

Results Messages

DepartmentID	DepartmentName	FirstName	LastName
11	BIOL	David	Lahti
45	HMNS	David	Lahti

12. Departmental Teaching Load: Report the total number of credit hours being taught by each instructor in a department. - needs class table

13. Show the number of students enrolled in each course - Ahnaf

```
-- Display the course, course name, and total enrollment - Ahnaf
SELECT CONCAT(C.CourseAbbreviation, C.CourseNumber) AS [Course],
       C.CourseName,
       SUM(E.CurrentEnrollment) AS [TotalEnrollment]
FROM Academic.Course AS C
     INNER JOIN Academic.Section AS S
       ON C.CourseId = S.CourseID
     INNER JOIN Enrollment.EnrollmentDetail AS E
       ON S.SectionID = E.SectionID
GROUP BY C.CourseAbbreviation, C.CourseNumber, C.CourseName
```

	Course	CourseName	TotalEnrollment
1	ACCT 100	Fin & Mgr Acct	63
2	ACCT 101	Int Theo & Prac Acct 1	500
3	ACCT 102	Intro Theo & Prac Acct 2	342
4	ACCT 201	Inter Acct 1	311
5	ACCT 202	Inter Acct 2	289
6	ACCT 261	Business Law I	352
7	ACCT 305	Cost Acct	296
8	ACCT 306	Quant Techniq Pln & Ctrl	267
9	ACCT 311	Advcd Acct	263
10	ACCT 321	Auditing 1	193

14. -- Which department offers the most number of courses - ALEKS

```
-- Which department offers the most number of courses
GO
```

```
WITH DepartmentCourseCount AS (
    SELECT
        DepartmentID,
        COUNT(CourseID) AS NumberOfCoursesOffered
    FROM [Academic].[Course]
    GROUP BY DepartmentID
)
SELECT TOP 1
    DepartmentID,
    NumberOfCoursesOffered
FROM DepartmentCourseCount
ORDER BY NumberOfCoursesOffered DESC;
```

Results:

Results	Messages
DepartmentID	NumberOfCoursesOffered
66	146

15. -- what instructors teach the course with the most number of sections? - ALEKS

```
GO
WITH CourseMostSections (CourseName, CourseSectionCounts, DepartmentID)
AS
(
    SELECT TOP 1
        C.CourseName
        , COUNT(S.Section) AS CourseSectionCounts
        , DepartmentID
    FROM [Academic].[Course] AS C
    INNER JOIN [Academic].[Section] AS S
    ON C.CourseId = S.CourseId
    GROUP BY CourseName, DepartmentID
    ORDER BY CourseSectionCounts DESC
```

```

)
SELECT    C.CourseName
          , C.CourseSectionCounts
          , C.DepartmentId
          , D.InstructorId
          , (I.FirstName + ' ' + I.LastName) AS InstructorFullName
FROM CourseMostSections AS C
INNER JOIN [Personnel].[DepartmentInstructor] AS D
ON C.DepartmentId = D.DepartmentId
INNER JOIN [Personnel].[Instructor] AS I
ON D.InstructorId = I.InstructorID

```

Results:

Results		Messages				
	CourseName	CourseSectionCounts	DepartmentId	InstructorId	InstructorFullName	
1	Private Instruction Jazz Perf	62	51	17	David	Adler
2	Private Instruction Jazz Perf	62	51	56	Yoshio	Aomori
3	Private Instruction Jazz Perf	62	51	58	Darcy	Argue
4	Private Instruction Jazz Perf	62	51	59	Maria	Argyros
5	Private Instruction Jazz Perf	62	51	60	Timothy	Armacost
6	Private Instruction Jazz Perf	62	51	119	David	Berkman
7	Private Instruction Jazz Perf	62	51	148	Rogelio	Boccatto
8	Private Instruction Jazz Perf	62	51	154	Paul	Bollenback
9	Private Instruction Jazz Perf	62	51	158	Luis	Bonilla
10	Private Instruction Jazz Perf	62	51	209	Peter	Calandra
11	Private Instruction Jazz Perf	62	51	262	Vincent	Cherico
12	Private Instruction Jazz Perf	62	51	418	John	Ellis
13	Private Instruction Jazz Perf	62	51	488	Ian	Froman
14	Private Instruction Jazz Perf	62	51	606	Antonio	Hart
15	Private Instruction Jazz Perf	62	51	607	William	Hart
16	Private Instruction Jazz Perf	62	51	676	Aubrey	Johnson
17	Private Instruction Jazz Perf	62	51	868	Jeannette	Lovetri
18	Private Instruction Jazz Perf	62	51	886	Dennis	Mackrel
19	Private Instruction Jazz Perf	62	51	917	Joseph	Martin
20	Private Instruction Jazz Perf	62	51	995	Michael	Moreno
21	Private Instruction Jazz Perf	62	51	999	Michael	Mossman
22	Private Instruction Jazz Perf	62	51	1090	Jeb	Patton
23	Private Instruction Jazz Perf	62	51	1126	Lonnie	Plaxico
24	Private Instruction Jazz Perf	62	51	1474	Charenee	Wade
25	Private Instruction Jazz Perf	62	51	1546	David	Wong

ChatGPT Revelations

How would you design a normalized (3NF) database for a colleges classes with the following given data: Semester, Section, Code, Course, Description (what class it is), Day, Time, Instructor, Location, Enrolled, Limit, Mode of Instruction (In-person, hybrid, etc.)

1. **Semesters Table:**

- SemesterID (Primary Key)
- SemesterName
- Year

<Currently the semester column has "Current Semester" as its only value so maybe we should have a function that looks at the current date and turns it into "Fall 2023" or "Spring 2024" depending on when the data was inserted into the database? This function won't really shine through in the data but I think Heller would appreciate the thought into the future use>

2. **Courses Table:**

- CourseID (Primary Key)
- Code
- Description

3. **Instructors Table:**

- InstructorID (Primary Key)
- InstructorName
- Department <Maybe to include a department column to show all the departments the instructor teaches in>

4. **Locations Table:**

- LocationID (Primary Key)
- LocationName

5. **Classes Table:**

- ClassID (Primary Key)
- SemesterID (Foreign Key referencing Semesters)
- CourseID (Foreign Key referencing Courses)
- InstructorID (Foreign Key referencing Instructors)
- LocationID (Foreign Key referencing Locations)
- Day
- Time
- Enrolled
- Limit
- ModeOfInstruction
- Section

In this design:

- **Semesters Table:** Stores information about different semesters.
- **Courses Table:** Stores information about different courses.
- **Instructors Table:** Stores information about different instructors.
- **Locations Table:** Stores information about different class locations.
- **Classes Table:** Represents the main table that links all the information together. It includes foreign keys to reference other tables and stores specific details about each class.

Delegated Tasks

TASKS LEFT

- write 8 queries

- Create a Conceptual Data Model (CDM) that explains the entities and relationships.
- SQL Doc 5 from Redgate to fully document db
- Create a backup of your final database solution (ClassTimeLastNameFirstNameNameBIClass.bak)
- Make the VHDX submission package

Assigned:

Aleksandra Georgievska (Windows & Mac)

- [DbSecurity].[UserAuthorization] Table
- [Process].[WorkflowSteps] table
- [Process].[usp_TrackWorkFlow]
- Instructor Table
- Course Table
- 2 of 10 queries
- Gantt Chart
- TODO sheet
- Create a Physical Data Model (PDM) that explains all of the Tables and foreign keys with their cardinality. (Using <https://dbdiagram.io/d>)
 - i. Isolate table and foreign key relationships in subject areas.
 - ii. Show the User Defined Datatypes as part of the PDM.
 - TODO: finalize the relationships (many to many etc)
- Make the slides/presentations

Nicholas Kong (Windows)

- Mode Instruction Table
- RoomLocation Table
- Schedule Table
- TODO sheet
- Make the slides/presentations

Aryeh Richman (Mac)

- Department Table
- DepartmentInstructor Table
- ClassDay Table
- TODO sheet
- Make the slides/presentations

Edwin Wray (Mac)

- Building Location Table
- Class Table
- TODO sheet
- Make the slides/presentations

Sigalita Yakubova (Windows)

- Semester Table
- Section Table
- Create a Logical Data Model (LDM) that explains all of the required attributes that fully describe their entity which are required to be maintained by the application.
- TODO sheet
- Make the slides/presentations

Ahnaf Ahmed (Windows & Mac)

- Days table
- EnrollmentDetail Table
- TODO sheet
- Make the slides/presentations

Questions for Heller

○

Meeting Minutes

12.1.23

12:00 pm - 1:00 pm

Attendance: Nicholas Kong, Aleksandra Georgievska, Aryeh Richman, Edwin Wray, Sigalita Yakubova, Ahnaf Ahmed

Agenda

- ☒ ~~Go over specifications~~
- ☒ ~~Determine if we have any questions for professor Heller~~

Meeting Minutes

- Review the specs
- Note: take notice of potential many to many relationships like “Instructor who can work in one or more Departments”

- Made plan to meet monday to review ideas

Action Items

- Everyone look through the classes data over the weekend and think about ERD ideas
- Find a time that works for everyone on Monday to have second meeting to plan work delegation
- Write 10 queries
- Aleks will make template
 - Order of execution should be truncate then load then process workflow steps

12.4.23

12:00 pm - 1:00 pm

Attendance: Nicholas Kong, Aleksandra Georgievska, Aryeh Richman, Sigalita Yakubova, Ahnaf Ahmed

Agenda

- ☐ Discuss project ideas
- ☐ Delegate tasks
- ☐ Set due dates

Meeting Minutes

- Discussed tables plan
- Reviewed the UDT suggestions
 - Agreed upon **BuildingCode**, **TimeSlot**, **CreditHours**, **DayAbbreviations** Types
- ISSUE: Any table that has a foreign key the value of that foreign key depends on what the value of the primary key was, so the table of primary key needs to be defined first
- Any table that has a PK can be filled in first and then we move on to the tables with the foreign keys
 - Make a digraph to determine prereqs of the tables - Aryeh doing
- Aryeh demod the graph prereqs

Action Items

Due Tuesday night:

- Make the these tables
- make the constraints
- make the stored procedures
- let Aleks know when you're ready to merge your changes