Overview

NOTE:

- This is a draft of the midterm exam. We are releasing the draft to enable students to start thinking about the exam and potentially identifying material that needs clarification.
- Despite being a draft, the only changes will be clarifications. So, students can begin thinking about and working on answers.
- The TAs will provide guidance for submission formats and annotations.

Instructions

- **TA TODO:** TAs to finalize instructions, especially submission instructions.

Due Date, Completing the Exam and Rules

1. The midterm exam is due at 11:59 PM on Monday, 28-MAR-2022. **You are not allowed to use late days.**

2. You may use on-line information and sources to answer questions. But,
   A. You cannot simply cut and paste answers or code. You answer must demonstrate that you understood the material and are capable of producing an answer from your understanding.
   B. You must cite any only sources of information that you used. This can simply be a comment in a text/markdown cell in your answer. For example, `(Note: I used https://www.w3schools.com/sql/sql_check.asp to help me with the syntax for adding a check constraint).`
   C. You do NOT need to cite lecture notes, recordings, slides, ... You do not need to cite information from the recommended textbook or textbook slides.

3. You MUST NOT collaborate with ANYONE, including other students. You MAY speak with the professor or a TA to discuss the exam.

4. If you have questions, post them as PRIVATE question on Ed dicussion and use the Category `Exams->Midterm.`

5. There is a pinned Ed discussion topic Midterm Clarifications that the professor and TA will use to communicate updates and clarifications. **Students are responsible for checking this post.**

Submission Format and Instructions

- **TA TODO:** TAs to finalize instructions, especially submission instructions.

Environment Setup

Notes:

1. This section tests your environment.

2. You will need to change the MySQL userID and password in some of the cells below to match your configuration.

3. You may need to load data and copy databases. The relevant questions provide information.

In [2]:
```
%load_ext sql
```

In [3]:
```
%sql mysql+pymysql://root:Xcz990208!@localhost
```

Out[3]:
```
'Connected: root@None'
```

In [4]:
```
from sqlalchemy import create_engine
```

In [5]:
```
sql_engine = create_engine("mysql+pymysql://root:Xcz990208!@localhost")
```

In [6]:
```
import pandas as pd
```

In [7]:
```
sql = """
    select customerName, customerNumber, city, country from classicmodels.customers
        where country = 'France'
"""

res = pd.read_sql(sql, con=sql_engine)
```

In [8]:
```
res
```

Out[8]:

	customerName	customerNumber	city	country
0	Atelier graphique	103	Nantes	France
1	La Rochelle Gifts	119	Nantes	France
2	Saveley & Henriot, Co.	146	Lyon	France
3	Daedalus Designs Imports	171	Lille	France
4	La Corne D'abondance, Co.	172	Paris	France

	customerName	customerNumber	city	country
5	Mini Caravy	209	Strasbourg	France
6	Alpha Cognac	242	Toulouse	France
7	Lyon Souveniers	250	Paris	France
8	Auto Associés & Cie.	256	Versailles	France
9	Marseille Mini Autos	350	Marseille	France
10	Reims Collectables	353	Reims	France
11	Auto Canal+ Petit	406	Paris	France

In [9]:
```
import pymysql
```

In [10]:
```
sql_conn = pymysql.connect(
    user="root",
    password='Xcz990208!',
    host="localhost",
    port=3306,
    cursorclass=pymysql.cursors.DictCursor,
    autocommit=True)
```

In [11]:
```
cur = sql_conn.cursor()

res = cur.execute(sql)
res = cur.fetchall()
res
```

Out[11]:
```
[{'customerName': 'Atelier graphique',
  'customerNumber': 103,
  'city': 'Nantes',
  'country': 'France'},
 {'customerName': 'La Rochelle Gifts',
  'customerNumber': 119,
  'city': 'Nantes',
  'country': 'France'},
 {'customerName': 'Saveley & Henriot, Co.',
  'customerNumber': 146,
  'city': 'Lyon',
  'country': 'France'},
 {'customerName': 'Daedalus Designs Imports',
  'customerNumber': 171,
  'city': 'Lille',
  'country': 'France'},
 {'customerName': "La Corne D'abondance, Co.",
  'customerNumber': 172,
  'city': 'Paris',
  'country': 'France'},
 {'customerName': 'Mini Caravy',
  'customerNumber': 209,
  'city': 'Strasbourg',
  'country': 'France'},
 {'customerName': 'Alpha Cognac',
  'customerNumber': 242,
  'city': 'Toulouse',
  'country': 'France'},
 {'customerName': 'Lyon Souveniers',
```

```
 'customerNumber': 250,
 'city': 'Paris',
 'country': 'France'},
{'customerName': 'Auto Associés & Cie.',
 'customerNumber': 256,
 'city': 'Versailles',
 'country': 'France'},
{'customerName': 'Marseille Mini Autos',
 'customerNumber': 350,
 'city': 'Marseille',
 'country': 'France'},
{'customerName': 'Reims Collectables',
 'customerNumber': 353,
 'city': 'Reims',
 'country': 'France'},
{'customerName': 'Auto Canal+ Petit',
 'customerNumber': 406,
 'city': 'Paris',
 'country': 'France'}]
```

In [12]:
```
cur.close()
```

Written Questions

Note:

"If you can't explain something in a few words, try fewer." – Robert Brault

"Professor Ferguson has the patience of a ferret that just drank a double espresso. If your answer is long, he gets bored and cranky, and deducts points." - Anonymous TA advising students in a previous semester.

- We expect brief, succinct answers.
- We deduct points for bloviating.

W1

Briefly explain the differences between:

1. *Candidate Key* and *Super Key.*
2. *Primary Key* and _Unique*Key.*
3. *Natural Key* and *Surrogate Key.*

Answer

1. (1) Candidate Key is a subset of a super key, so number of super keys are more than number of candidate keys. All super keys can't be candidate keys, but all candidate keys are super keys.\ (2) Super key's attributes can contain NULL values while candidate key's attributes can also contain NULL values.
2. (1) Primary key will not accept NULL values whereas Unique key can.\ (2) A table can have only one primary key whereas there can be multiple unique key.\ (3) Index: Primary key creates clustered index while Primary key creates non-clustered index.

3. (1) A natural key is a column already in your data that can uniquely identify the row while a surrogate key is a key which you create for the purpose of being a primary key.\ (2) The attributes of a natural key always exist in real world while Surrogate keys have no "business" meaning and their only purpose is to identify a record in the table.

W2

1. Define the concept of *immutable* columns (data).
2. What is a benefit of using immutable columns to define a primary key.

Answer

1. Immutable data is a piece of information in a database that cannot be (or shouldn't be) deleted or modified.
2. This avoids the problem of dangling references or orphan records created by other relations referring to a tuple whose primary key has changed. If the primary key is immutable, this can never happen, we can prevent from mistakes like accidently changed the primary key.

`(Note: I used https://www.tibco.com/reference-center/what-is-immutable-data to help me with the definition for immutable columns).`

W3

Codd's Third Rule states, "Null values (distinct from the empty character string or a string of blank characters and distinct from zero or any other number) are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of data type."

Briefly explain the value of this rule, and provide two examples.

Answer

- Value:\ (1) Null value make it possible to represent data that was missing or 0 value, which is a general way in DBMS.\ (2) Because it's independent with the data type, so it's more error tolerant.It will help us distinguish the missing value and 0 value. and it makes sure all database system has a unified representation of missing value and inapplicable value.
- Example:\ (1) Death date for a living person in the table of people inforation because it's not applicable.\ (2) If we update a row with null to null+1, the result will still be null.

W4

The relational model and SQL are *closed* under their operations. Briefly explain what this concept means.

Answer

- Closure property: the relational algebra consists operator that takes one or more relations as input and produces another relation as output. It means after each operation, we always get a subset of previous set, where we can still use some other operations.Relations are closed under the algebra, just as numbers are closed under arithmetic operations.

(Note: I used https://www.encyclopedia.com/science/encyclopedias-almanacs-transcripts-and-maps/closure-property#:~:text=The%20closure%20property%20means%20that,respect%20to%20a%20give to help me with the definition for Closure property).

W5

Codd's 6th rule states, "All views that are theoretically updatable are also updatable by the system."

Using the following table definition, use SQL (`CREATE VIEW`) to define:

1. Two views of the table that are theoretically not possible to update.
2. One view that is theoretically possible to update.

- You do not need to execute the create statement. We are focusing on your understanding.

```
create table S22_W4111_Midterm.midterm_students
(
    social_security_no char(9) not null
        primary key,
    last_name varchar(64) null,
    first_name varchar(64) null,
    enrollment_year year null,
    total_credits int null
);
```

Answer

1. Can not Update\ (1) With "distinct":
```
CREATE VIEW student_status AS
 SELECT Distinct first_name, last_name, enrollment_year, total_credits
     FROM S22_W4111_Midterm.midterm_students
         WHERE total_credits is NOT Null;
```
(2)With "subquery":
```
CREATE VIEW student_status AS
 SELECT Distinct first_name, last_name
     FROM S22_W4111_Midterm.midterm_students
         WHERE (FROM S22_W4111_Midterm.midterm_students
                 SELECT first_name, last_name, enrollment_year,
total_credits
                     WHERE total_credits >= 5);
```
2. Can Update\ (1) Containing primary key:
```
CREATE VIEW student_midterm AS
 SELECT social_security_no, first_name, last_name
     FROM S22_W4111_Midterm.midterm_students
         WHERE total_credits is Null;
```

W6

In the Columbia University directory of classes, the "Section Key" for this course is `20213COMS4111W002` .

- Explain why having a column `section_key varchar(17)` that holds section key values is not-atomic.
- Give two explanations for why using the section key (not atomic data) for a column causes problems.

Answer

1. Have a complex, non-atomic attribute is really a bad idea. The value 20213COMS4111W002 is a combination of different representations of properties of the attribute. It could be seperated into 4 parts of value: year, dept, course id, section, which can be edited by changing different parts of information, so it's non-atomic.
2. (1) If a column containing multiple values, it's hard to sort or select columns by any of these 4 values, etc dept.\ (2) It's hard to find problems regarding to the table structure, the indice of values will become complicated.

W7

Briefly explain the differences between:

- Database stored procedure
- Database function
- Database trigger

Answer

Property	Functions	Stored Procedures	Trigger
Can update data	No	Yes	Yes
Can be part of a query	Yes	Yes	No
Called explicitly	Yes	Yes	No
Can return values	Yes	Yes	No
Use Transactions	No	Yes	Yes
Input/output parameter	input	input/output	No

W8

Briefly explain:

- Natural join
- Equi-join
- Theta join
- Self-join

Answer

- Natural join: Natural Join joins two tables based on same attribute name and datatypes. The resulting table will contain all the attributes of both the table but keep only one copy of each common column.
- Equi-join: EQUI JOIN creates a JOIN for equality or matching column(s) values of the relative tables. It also create JOIN by using ON and then providing the names of the columns with

their relative tables to check equality using equal sign (=).

- Theta join: Theta Join allows you to merge two tables based on the condition represented by theta. Theta joins work for all comparison operators.
- Self-join: A self join is a join in which a table is joined with itself (which is also called Unary relationships), especially when the table has a FOREIGN KEY which references its own PRIMARY KEY.

`(Note: I used https://www.w3resource.com/sql/joins/perform-a-self-join.php#:~:text=A%20self%20join%20is%20a,other%20row%20of%20the%20table. to help me with the definition for Self-join).`

W9

Consider the schema for Classic Models, which we have used in the class.

1. Is any entity type in the schema a *weak entity?* If yes, list one of the weak entity types.
2. In database design, using `ON DELETE CASCADE` may not be a desired behavior/design. Why is it more likely that `ON DELETE CASCADE` is the correct behavior for weak entities when the referencing row is deleted?

Answer

1. Yes. The productlines is weak entity because it has no meaning independent of the product.
2. Use the ON DELETE CASCADE option to specify whether you want rows deleted in a child table when corresponding rows are deleted in the parent table. If you do not specify cascading deletes, the default behavior of the database server prevents you from deleting data in a table if other tables reference it. Because the weak entity is usually composed by other entity's foreign key, so mostly the weak entity is a child table of some parent's. ON DELETE CASCADE is the right choice.

`(Note: I used https://www.ibm.com/docs/en/informix-servers/14.10?topic=clause-using-delete-cascade-option to help me with the definition for ON DELETE CASCADE).`

W10

1. Briefly explain the concept of a *database cursor.*
2. Why is using a cursor sometimes helpful for applications processing database information.

Answer

1. A database cursor is an identifier associated with a group of rows. It is, in a sense, a pointer to the current row in a buffer.
2. In my opinion, cursor is like streaming a video, we don't have to download whole video if we want to watch it, instead, we can stream at any time of a video. For large result set of DB, cursor could help to save processing time because you don't need to wait for the processing and download of your complete recordset and it will save memory, both on the server and on the client because they don't have to dedicate a big chunk of memory to resultsets. These are why using a cursor is sometimes helpful.

(Note: I used https://www.ibm.com/docs/en/informix-servers/12.10?topic=sql-database-cursor to help me with the definition for database cursor).

Relational Algebra

R1

- You can assume that the type for the columns in this question are `TEXT`.
- Translate the following relational schema definition into an equivalent SQL `CREATE TABLE` statement.
- You do not need to execute the statement. We are focusing on understanding.

$$(\underline{branch\_id}, \underline{account\_id}, balance) \tag{1}$$

Answer

```
create table table_name
(
    branch_id TEXT not null,
    account_id TEXT not null,
    balance TEXT not null,
   primary key (branch_id,account_id)
);
```

R2

- Use the RelaX online calculator with the Silberschatz - UniversityDB for this query.
- You may use **only** the `ρ (pi), σ, ⋈, ⟕, ⋈` for this question. You can use predicates/conditions for the query, and column list for the project.
- Write a relational algebra statement that produces a table containing the `ID` and `name` of instructors who do not advise any students. Your result should be of the form:

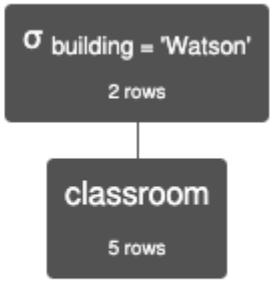
```
(ID, name)
```

- Your answer must contain the query statement (in text, not in an image) and a screen capture of the query execution and result. An example of the structure of the answer is:

Query:

```
σ building='Watson' (classroom)
```

Screen capture:



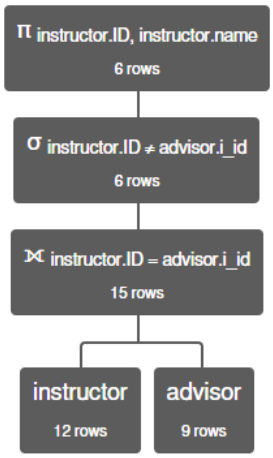
$$\sigma_{\text{building = 'Watson'}}(\text{classroom})$$

classroom.building	classroom.room_number	classroom.capacity
'Watson'	100	30
'Watson'	120	50

Answer

Query: π instructor.ID, instructor.name σ instructor.ID $\neq$ advisor.i_id (instructor $\bowtie$ instructor.ID = advisor.i_id advisor)

Screen Capture:



$$\pi_{\text{instructor.ID, instructor.name}} \sigma_{\text{instructor.ID} \neq \text{advisor.i\_id}} (\text{instructor} \leftouterjoin_{\text{instructor.ID} = \text{advisor.i\_id}} \text{advisor})$$

instructor.ID	instructor.name
12121	'Wu'
15151	'Mozart'
32343	'El Said'
33456	'Gold'
58583	'Califieri'
83821	'Brandt'

R3

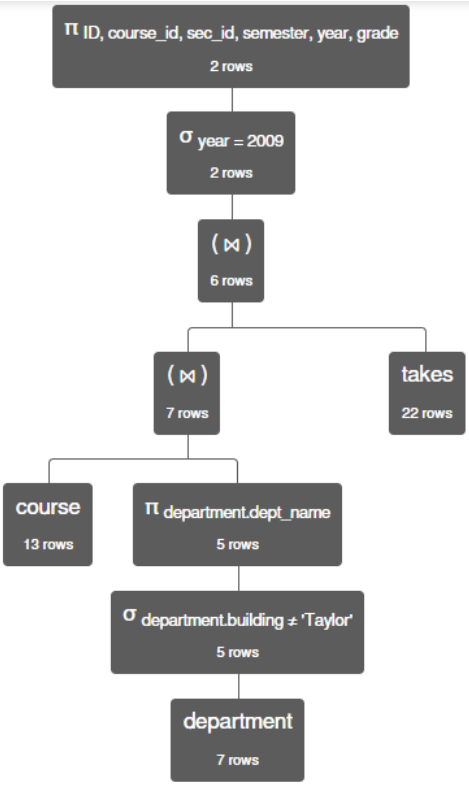
- Use the RelaX online calculator with the Silberschatz - UniversityDB for this query.
- You may only use the relational operators π, σ, $\bowtie$ for the solution. You may use column lists and conditions/predicates.
- Write a relational algebra expression equivalent to the following SQL statement.

```
select * from takes
    where course_id in
          (select course_id from course where dept_name not in
              (select dept_name from department where
building='Taylor'))
    and
    year='2009';
```

- **Note:**
  - The book's sample data you loaded into MySQL is different from the data in the RelaX calculator. RelaX has data from a prior version of the book.
  - If you want to test the SQL statement in MySQL, you can load the data from version 6 of the book into a separate schema in MySQL.

Answer

Query: `π ID,course_id,sec_id,semester,year,grade σ year = 2009(((course)⋈(π department.dept_name σ department.building ≠ 'Taylor' (department)))⋈ (takes))`



$$\pi_{\text{ID, course\_id, sec\_id, semester, year, grade}} \sigma_{\text{year = 2009}} (((\text{course}) \bowtie (\pi_{\text{department.dept\_name}} \sigma_{\text{department.building} \neq \text{'Taylor'}} (\text{department}))) \bowtie (\text{takes}))$$

takes.ID	course.course_id	takes.sec_id	takes.semester	takes.year	takes.grade
98988	'BIO-101'	1	'Summer'	2009	'A'
44553	'PHY-101'	1	'Fall'	2009	'B-'

Entity Relationship Model

- Setup:
  - Being able to translate a written description of a schema into an ER diagram is an important skill.
  - This question tests that skill by describing a data model that you have to define using Crow's Foot Notation.
  - You ER model must be implementable in SQL DDL.
  - You do not need to choose data types for columns.
  - You should add notes to your diagram for clarification if you think necessary.

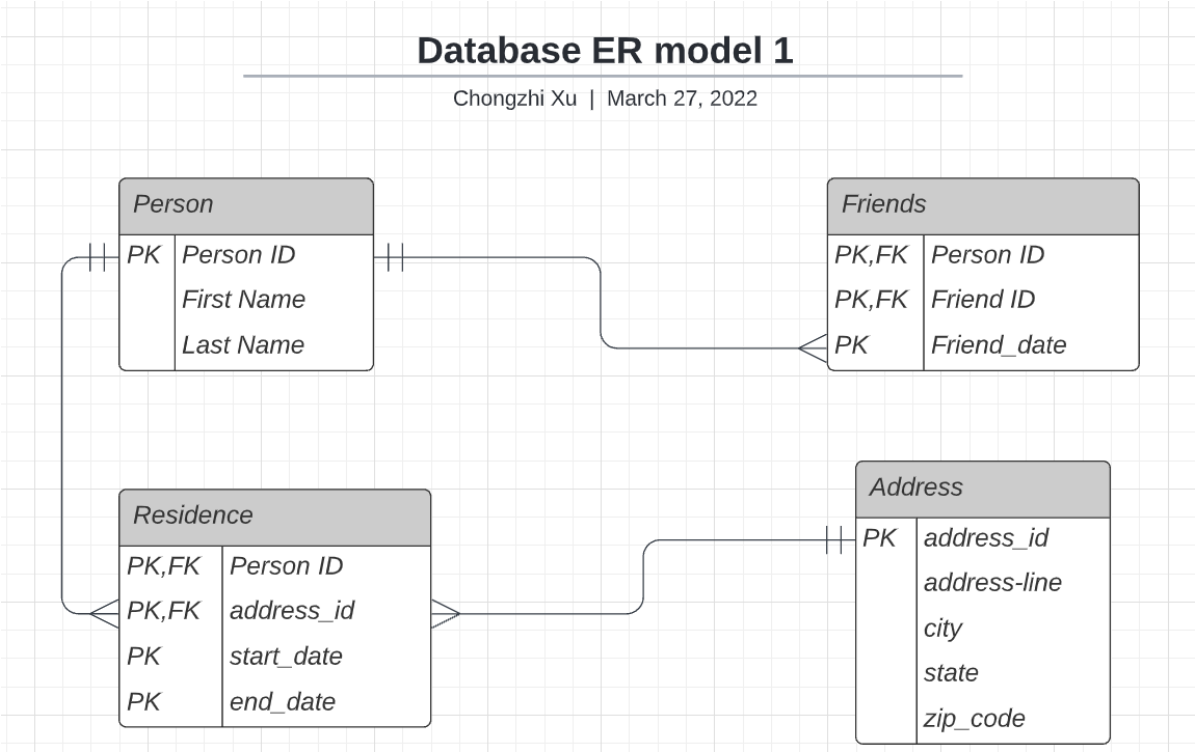
ER1

- Description:

- There are two entity types:
  1. $$Person(\underline{person\_id}, last\_name, first\_name) \quad (2)$$
  2. $$Address(\underline{address\_id}, address\_line, city, state, zip\_code) \quad (3)$$
- There are two many-to-many relationships:
  1. $Person - Address$: A $Person$ $lives\_at$ an $Address$ from a $start\_date$ to an end_date.
  2. $Person - Person$: A $Person$ $became\_friends\_with$ a $Person$ on a $friend\_date$
- You do not need to worry about semantics constraints, e.g. it is OK if $lives\_at$ terms overlap.

- **Notes:**
  - Use LucidChart to draw your ER diagram.
  - You do not need to worry about data types.
  - You must show primary and foreign keys.
  - You do not need to worry about semantics constraints, e.g. it is OK if $lives\_at$ terms overlap.
  - Put your diagram in the directory that contains your notebook and include following instructions provided for previous homework assignments.

Answer



ER2

- Description:
  - There are three entity types:
    1. $$CheckingAccount(\underline{checking\_id}, balance) \quad (4)$$

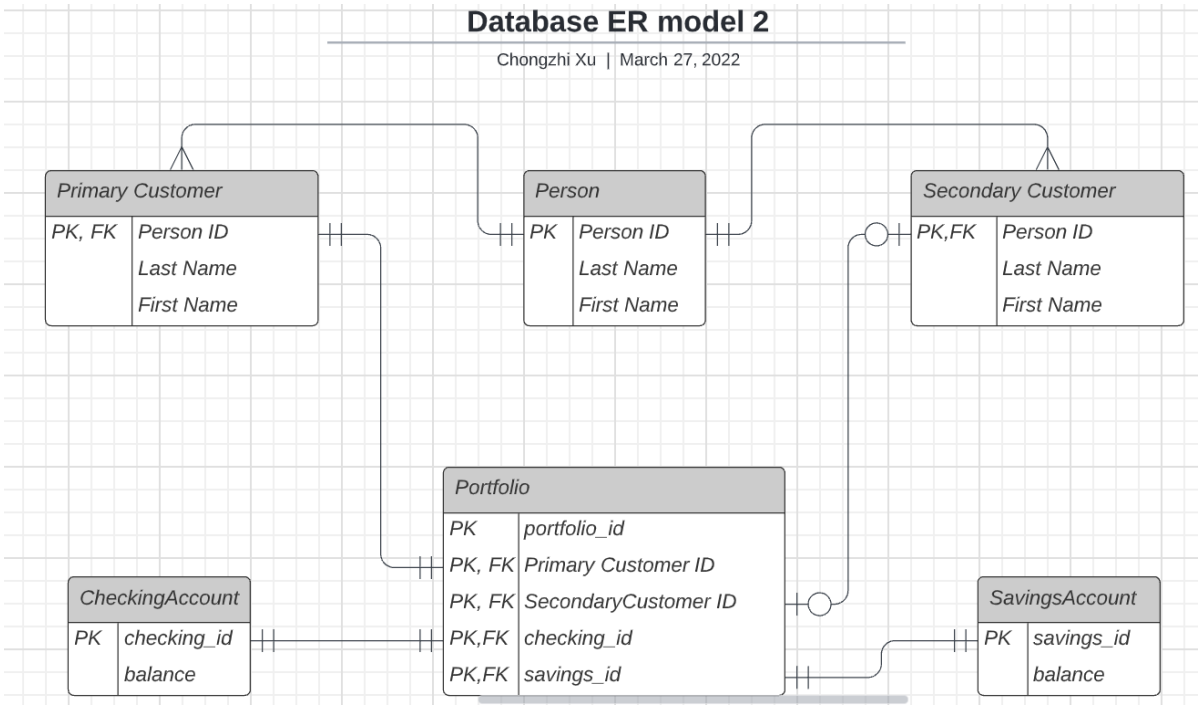
2. $$SavingsAccount(\underline{savings\_id}, balance) \tag{5}$$

3. $$Person(\underline{person\_id}, last\_name, first\_name) \tag{6}$$

   - A $Portfolio$ is an entity type that:
     1. Has a primary key $portfolio\_id$.
     2. *Aggregates* other entity types:
        - Exactly one $primary\_customer$.
        - At most one $secondary\_customer$.
        - Exactly one $checking\_account$.
        - Exactly one $savings\_account$.

- **Notes:**
  - Use LucidChart to draw your ER diagram.
  - You do not need to worry about data types.
  - You must show primary and foreign keys.
  - You do not need to worry about semantics constraints, e.g. it is OK of the same person is the primary and secondary customer.

Answer



SQL Schema and DDL

DDL1

- You have a logical datamodel ER-diagram (see below).
- You need to use DDL to define a schema that realizes the model.
- Logical models are not specific enough for direct implementation. This means that:
  - You will have to assign concrete types to columns, and choose things like `GENERATED`, `DEFAULT`, etc.

- You have to make assumptions about things like `NOT NULL`.
- You may have to decompose a table into two tables, or extract common attributes from multiple tables into a single, referenced table.
- Implementing the relationships may require adding columns and foreign keys, associative entities, etc.
- You may have to make other design and implementation choices. **This means that there is no single correct answer.**

- In addition to the key constraints, you must also implement the following constraints:
  - `email` values must contain a `@` and end in `.edu`
  - `semester` must be `fall`, `spring` or `summer`
  - `year` must be greater than or equal to 2021 and less than or equal to 2023.

ER Diagram

Answer

In [ ]:

```
# _Design Decisions, Notes, etc._ Document any assumption or decisions.

# 1. The Mysql query should be like below to create the table.
'''
drop schema cx2273_s22_midterm;
create schema if not exists cx2273_s22_midterm;

use cx2273_s22_midterm;

CREATE SCHEMA IF NOT EXISTS midterm_ddl

CREATE TABLE `School` (
  `school_code` varchar(64) not null,
  `school_name` varchar(64),
  PRIMARY KEY (`school_code`)
);

CREATE TABLE `Department` (
  `department_code` varchar(64) not null,
  `department_name` varchar(64),
  PRIMARY KEY (`department_code`)
);

CREATE TABLE `Faculty` (
  `UNI` varchar(64) not null,
  `lasts_name` varchar(64),
  `first_name` varchar(64),
  `department_code` varchar(64) not null,
  `email` varchar(64),
  PRIMARY KEY (`UNI`),
  FOREIGN KEY (`department_code`) REFERENCES `Department`(`department_code`),
  KEY `UQ` (`email`)
);

CREATE TABLE `SchoolDepartment` (
  `school_code` varchar(64) not null,
  `department_code` varchar(64) not null,
  PRIMARY KEY (`school_code`, `department_code`),
  FOREIGN KEY (`department_code`) REFERENCES `Department`(`department_code`),
  FOREIGN KEY (`school_code`) REFERENCES `School`(`school_code`)
```

```
);

CREATE TABLE `Course` (
  `course_no` varchar(64) not null,
  `title` varchar(64),
  `descripion` text,
  PRIMARY KEY (`course_no`)
);

CREATE TABLE `Section` (
  `call_no` varchar(64) not null,
  `course_no` varchar(64) not null,
  `section_no` varchar(64),
  `semester` varchar(64),
  `year` float,
  `instructor_id` varchar(64),
  PRIMARY KEY (`call_no`),
  FOREIGN KEY (`course_no`) REFERENCES `Course`(`course_no`),
  KEY `UQ` (`course_no`, `section_no`, `semester`, `year`)
);

CREATE TABLE `Student` (
  `UNI` varchar(64) not null,
  `last_name` varchar(64),
  `first_name` varchar(64),
  `email` varchar(64),
  PRIMARY KEY (`UNI`),
  KEY `UQ` (`email`)
);

CREATE TABLE `StudentSection` (
  `UNI` varchar(64) not null,
  `call_no` varchar(64) not null,
  `role` varchar(64),
  PRIMARY KEY (`UNI`, `call_no`),
  FOREIGN KEY (`UNI`) REFERENCES `Student`(`UNI`),
  FOREIGN KEY (`call_no`) REFERENCES `Section`(`call_no`)
);

'''
#2. Execute the query below to add constraint.
'''
Alter table Student add constraint chk_email check (email like '%_@_%.edu%');
Alter table Faculty add constraint chk_email_1 check (email like '%_@_%.edu%');
Alter table Section add constraint chk_year check (year >= 2021 and year <= 2023);
Alter table Section add constraint check_semester check (semester in ('fall', 'spring'
'''
```

DDL

- Execute your DDL in the cell below. You may use DataGrip or other tools to help build the schema and statements.
- You can copy and paste the `SQL CREAT TABLE` below, but you MUST execute the statements.

In [28]:

```
# DDL in cells below.
# Use you UNI for the schema.

%sql create schema if not exists cx2273_s22_midterm;
%sql select 1;
```

```
 * mysql+pymysql://root:***@localhost
1 rows affected.
 * mysql+pymysql://root:***@localhost
1 rows affected.
```

Out[28]:

1
1

In [ ]:
```
########################################Set-up#########################################
```

In [93]:
```
%%sql
use cx2273_s22_midterm;
INSERT INTO Course (course_no, title, description)
VALUES ('4111', 'database','Great');
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
1 rows affected.
```

Out[93]:
```
[]
```

In [94]:
```
%%sql
use cx2273_s22_midterm;
INSERT INTO department (department_code, department_name)
VALUES ('coms', 'computer science');
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
1 rows affected.
```

Out[94]:
```
[]
```

In [95]:
```
%%sql
use cx2273_s22_midterm;
INSERT INTO faculty (instructor_id, last_name, first_name, department_code, email)
VALUES ('cx2273', 'Xu', 'Chongzhi', 'coms', 'cx2273@columbia.edu');
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
1 rows affected.
```

Out[95]:
```
[]
```

In [96]:
```
%%sql
use cx2273_s22_midterm;
INSERT INTO Section (call_no, course_no, section_no, semester, year, instructor_id)
VALUES ('w', '4111','001','spring', 2021, 'cx2273' );
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
1 rows affected.
```

Out[96]:
```
[]
```

In [ ]:
```
########################################################################################
```

In [32]:
```
# SQL tests.
# Test your DDL with sample inserts/updates showing that you correctly implemented cor
```

```
#
```

In [97]:
```
%%sql
use cx2273_s22_midterm;
UPDATE section
SET year = 2025
WHERE year = 2021;
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
(pymysql.err.OperationalError) (3819, "Check constraint 'chk_year' is violated.")
[SQL: UPDATE section
SET year = 2025
WHERE year = 2021;]
(Background on this error at: https://sqlalche.me/e/14/e3q8)
```

In [101…
```
%%sql
use cx2273_s22_midterm;
UPDATE section
SET semester = 'winter'
WHERE semester = 'spring';
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
(pymysql.err.OperationalError) (3819, "Check constraint 'check_semester' is violate
d.")
[SQL: UPDATE section
SET semester = 'winter'
WHERE semester = 'spring';]
(Background on this error at: https://sqlalche.me/e/14/e3q8)
```

In [105…
```
%%sql
use cx2273_s22_midterm;
UPDATE faculty
SET email = 'cx2273columbia'
WHERE email = 'cx2273@columbia.edu';
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
(pymysql.err.OperationalError) (3819, "Check constraint 'chk_email_1' is violated.")
[SQL: UPDATE faculty
SET email = 'cx2273columbia'
WHERE email = 'cx2273@columbia.edu';]
(Background on this error at: https://sqlalche.me/e/14/e3q8)
```

SQL Queries

- You will use the Classic Models data for these questions.
- You loaded this database in a previous HW and tested that you have the database in the setup section.

S1

- Produce a table of the form:
  $(productLine, productName, productVendor, total\_revenue)$

- The contribution to total_value for an $orderLine$ is $quantityOrdered * priceEach$
- Total value is the sum of the $orderLine$ contributions for all $productCodes$ in a $productLine$.
- Only include results with $total\_value$ greater or equal to `$150000` and sorted by $total\_value$ descending.
- **NOTE:** You should be able to produce the answer without my providing the correct query output. I was giggling diabolically like the Riddler from Batman when writing the question. Then something like the following happened.

- So the output is below.

In [16]:
```
%sql select * from classicmodels.midterm_s1;
```

```
 * mysql+pymysql://dbuser:***@localhost
6 rows affected.
```

Out[16]:

productLine	productName	productVendor	total_revenue
Classic Cars	1992 Ferrari 360 Spider red	Unimax Art Galleries	276839.98
Classic Cars	2001 Ferrari Enzo	Second Gear Diecast	190755.86
Classic Cars	1952 Alpine Renault 1300	Classic Metal Creations	190017.96
Motorcycles	2003 Harley-Davidson Eagle Drag Bike	Red Start Diecast	170686.00
Classic Cars	1968 Ford Mustang	Autoart Studio Design	161531.48
Classic Cars	1969 Ford Falcon	Second Gear Diecast	152543.02

- Put your query below. You do not need to create a view.

In [106…

```
%%sql
use classicmodels;
with
    customer_orders_details as
        (select productLine, productName, productVendor, productCode, quantityOrdered
           from products natural join orderdetails),
    customer_orders_totals as
         (select productLine, productName, productVendor, sum(priceEach * quantityOr
               from customer_orders_details
                   group by productName),
    product_revenue as
         (select * from customer_orders_totals
             where total_revenue > 150000.00
                 order by total_revenue DESC)
select * from product_revenue;
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
6 rows affected.
```

Out[106…

productLine	productName	productVendor	total_revenue
Classic Cars	1992 Ferrari 360 Spider red	Unimax Art Galleries	276839.98
Classic Cars	2001 Ferrari Enzo	Second Gear Diecast	190755.86
Classic Cars	1952 Alpine Renault 1300	Classic Metal Creations	190017.96
Motorcycles	2003 Harley-Davidson Eagle Drag Bike	Red Start Diecast	170686.00
Classic Cars	1968 Ford Mustang	Autoart Studio Design	161531.48
Classic Cars	1969 Ford Falcon	Second Gear Diecast	152543.02

S2

- Produce a table containing the rows from `products` for any products not in any `orderDetails`.

In [107…

```
%%sql
use classicmodels;
with product_order_not_in as
    (select * from products where productCode not in (
        select productCode from orderdetails))
    select * from product_order_not_in
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
1 rows affected.
```

Out[107…

productCode	productName	productLine	productScale	productVendor	productDescription	quantit

productCode	productName	productLine	productScale	productVendor	productDescription	quantit
S18_3233	1985 Toyota Supra	Classic Cars	1:18	Highway 66 Mini Classics	This model features soft rubber tires, working steering, rubber mud guards, authentic Ford logos, detailed undercarriage, opening doors and hood, removable split rear gate, full size spare mounted in bed, detailed interior with opening glove box	

S3

- Define `order_time` to be the number of days between `orders.orderDate` and `orders.shippedDate`.
- The following tables has the form `(customerNumber, customerName, orderNumber, order_time)` for
  - Customers with `customers.country` in USA or Canada.
  - And the `orders.status` is not `Cancelled`.
- Your answer must match the sample below, including row order.

In [17]:
```
%sql select * from classicmodels.midterm_s3;
```

```
 * mysql+pymysql://dbuser:***@localhost
46 rows affected.
```

Out[17]:

customerNumber	customerName	orderNumber	order_time
328	Tekni Collectables Inc.	10401	None
450	The Sharp Gifts Warehouse	10407	None
362	Gifts4AllAges.com	10414	None
124	Mini Gifts Distributors Ltd.	10421	None
157	Diecast Classics Inc.	10422	None
161	Technics Stores Inc.	10140	6
205	Toys4GrownUps.com	10145	6
219	Boards & Toys Co.	10154	6
321	Corporate Gift Ideas Co.	10159	6
347	Men 'R' US Retailers, Ltd.	10160	6
462	FunGiftIdeas.com	10166	6
175	Gift Depot Inc.	10172	6
124	Mini Gifts Distributors Ltd.	10182	6

320	Mini Creations Ltd.	10185	6
205	Toys4GrownUps.com	10189	6
328	Tekni Collectables Inc.	10251	6
204	Online Mini Collectables	10276	6
198	Auto-Moto Classics Inc.	10290	6
339	Classic Gift Ideas, Inc	10307	6
161	Technics Stores Inc.	10317	6
363	Online Diecast Creations Co.	10322	6
486	Motor Mint Distributors Inc.	10331	6
198	Auto-Moto Classics Inc.	10352	6
462	FunGiftIdeas.com	10388	6
129	Mini Wheels Co.	10111	5
198	Auto-Moto Classics Inc.	10130	5
447	Gift Ideas Corp.	10131	5
124	Mini Gifts Distributors Ltd.	10142	5
487	Signal Collectibles Ltd.	10149	5
363	Online Diecast Creations Co.	10192	5
455	Super Scale Inc.	10196	5
475	West Coast Collectables Co.	10199	5
239	Collectable Mini Designs Co.	10226	5
486	Motor Mint Distributors Inc.	10236	5
181	Vitachrome Inc.	10237	5
456	Microscale Inc.	10242	5
455	Super Scale Inc.	10245	5
233	Québec Home Shopping Network	10261	5
319	Mini Classics	10308	5
157	Diecast Classics Inc.	10318	5
424	Classic Legends Inc.	10337	5
161	Technics Stores Inc.	10362	5
124	Mini Gifts Distributors Ltd.	10368	5
219	Boards & Toys Co.	10376	5
124	Mini Gifts Distributors Ltd.	10396	5
233	Québec Home Shopping Network	10411	5

- Put your query below. You do not need to create a view.

In [22]:
```
%%sql
use classicmodels;
with
```

```
    customer_order as
    (
        select customerNumber, customerName, Country, orderNumber, shippedDate, order
            from customers join orders using(customerNumber)
    ),
    customer_from_USA_CAN as
    (
        select *
            from customer_order
                where country = 'USA' or country = 'Canada'
    ),
    order_not_cancelled as
    (
        select *
            from customer_from_USA_CAN
                where status != 'Cancelled'
    ),
    customer_order_time as
    (
        select
            customerNumber,
            customerName,
            orderNumber,
            datediff(shippedDate, orderDate) as order_time
                from order_not_cancelled
                    order by order_time
    )

select * from customer_order_time
    where order_time = 5 or order_time = 6 or order_time is null
        order by case when order_time is Null then 0 else 1 end, order_time desc
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
46 rows affected.
```

Out[22]:

customerNumber	customerName	orderNumber	order_time
124	Mini Gifts Distributors Ltd.	10421	None
157	Diecast Classics Inc.	10422	None
328	Tekni Collectables Inc.	10401	None
362	Gifts4AllAges.com	10414	None
450	The Sharp Gifts Warehouse	10407	None
219	Boards & Toys Co.	10154	6
124	Mini Gifts Distributors Ltd.	10182	6
198	Auto-Moto Classics Inc.	10290	6
198	Auto-Moto Classics Inc.	10352	6
205	Toys4GrownUps.com	10189	6
204	Online Mini Collectables	10276	6
161	Technics Stores Inc.	10140	6
161	Technics Stores Inc.	10317	6
205	Toys4GrownUps.com	10145	6
175	Gift Depot Inc.	10172	6

customerNumber	customerName	orderNumber	order_time
320	Mini Creations Ltd.	10185	6
321	Corporate Gift Ideas Co.	10159	6
328	Tekni Collectables Inc.	10251	6
339	Classic Gift Ideas, Inc	10307	6
347	Men 'R' US Retailers, Ltd.	10160	6
363	Online Diecast Creations Co.	10322	6
462	FunGiftIdeas.com	10166	6
462	FunGiftIdeas.com	10388	6
486	Motor Mint Distributors Inc.	10331	6
319	Mini Classics	10308	5
124	Mini Gifts Distributors Ltd.	10142	5
124	Mini Gifts Distributors Ltd.	10368	5
124	Mini Gifts Distributors Ltd.	10396	5
239	Collectable Mini Designs Co.	10226	5
157	Diecast Classics Inc.	10318	5
161	Technics Stores Inc.	10362	5
181	Vitachrome Inc.	10237	5
363	Online Diecast Creations Co.	10192	5
198	Auto-Moto Classics Inc.	10130	5
424	Classic Legends Inc.	10337	5
219	Boards & Toys Co.	10376	5
233	Québec Home Shopping Network	10261	5
455	Super Scale Inc.	10196	5
455	Super Scale Inc.	10245	5
456	Microscale Inc.	10242	5
447	Gift Ideas Corp.	10131	5
233	Québec Home Shopping Network	10411	5
475	West Coast Collectables Co.	10199	5
486	Motor Mint Distributors Inc.	10236	5
129	Mini Wheels Co.	10111	5
487	Signal Collectibles Ltd.	10149	5

S4

- In almost all cases, the `scale` of a `products` is encoded in the `productCode`.
- For example,

In [19]:
```
%sql select productCode, productScale from classicmodels.products limit 5;
```

```
 * mysql+pymysql://dbuser:***@localhost
5 rows affected.
```

Out[19]:

productCode	productScale
S10_1678	1:10
S10_1949	1:10
S10_2016	1:10
S10_4698	1:10
S10_4757	1:10

- There are, however, some cases where this is not true. Produce the following table.

In [20]:
```
%%sql
select * from classicmodels.midterm_s4;
```

```
 * mysql+pymysql://dbuser:***@localhost
6 rows affected.
```

Out[20]:

scale_in_product_scale	scale_in_product_Number	productCode	productScale
18	12	S12_3148	1:18
72	18	S18_2581	1:72
18	24	S24_3856	1:18
18	24	S24_4620	1:18
18	700	S700_2824	1:18
72	700	S700_3167	1:72

In [109…
```
%%sql
use classicmodels;
select
    substring_index(productScale, ':', -1) as scale_in_product_scale,
    substring_index(substring_index(productCode, 'S', -1), '_', 1) as scale_in_produc
    productCode,
    productScale
        from products
            where substring_index(productScale, ':', -1) != substring_index(substring
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
6 rows affected.
```

Out[109…

scale_in_product_scale	scale_in_product_Number	productCode	productScale
18	12	S12_3148	1:18
72	18	S18_2581	1:72
18	24	S24_3856	1:18
18	24	S24_4620	1:18
18	700	S700_2824	1:18
72	700	S700_3167	1:72

Data Loading and Cleanup

DL1

- There is a file `course_info.csv` in the directory/folder for the midterm.
- Do not ask how I got this information, but it looked something like this ... ...

- The following code reads the JSON file and displays some of the data.

In [21]:
```
df = pd.read_csv("course_info.csv")
df
```

Out[21]:

	Unnamed: 0	ExamDate	Term	ChargeAmt1	Meets4	TypeCode	SubtermCode	CampusCode
0	0	NaN	20212	NaN	NaN	LC	NaN	MORN
1	1	NaN	20212	NaN	NaN	LC	NaN	MORN
2	2	NaN	20212	NaN	NaN	LC	NaN	MORN
3	3	NaN	20212	NaN	NaN	LC	NaN	MORN
4	4	NaN	20212	NaN	NaN	LC	NaN	MORN
...	...	...	...	...	...	...	...	...

	Unnamed: 0	ExamDate	Term	ChargeAmt1	Meets4	TypeCode	SubtermCode	CampusCode
21430	21430	NaN	20222	NaN	NaN	RG	L	MORN
21431	21431	NaN	20222	NaN	NaN	RG	K	MORN
21432	21432	NaN	20222	NaN	NaN	RG	L	MORN
21433	21433	NaN	20222	100.00	NaN	LC	K	MORN
21434	21434	NaN	20222	100.00	NaN	LC	L	MORN

21435 rows × 45 columns

- The list of columns in the data is:

In [23]:
```
for c in df.columns:
    print(c)
```

```
Unnamed: 0
ExamDate
Term
ChargeAmt1
Meets4
TypeCode
SubtermCode
CampusCode
Course
Instructor3Name
ChargeAmt2
CallNumber
CourseTitle
NumFixedUnits
Instructor1Name
DivisionName
MaxSize
Instructor4Name
Meets2
SchoolCode
ChargeMsg2
SubtermName
Approval
BulletinFlags
PrefixName
PrefixLongname
EnrollmentStatus
CampusName
Meets5
```

```
ClassNotes
DivisionCode
ExamMeet
DepartmentName
NumEnrolled
Meets3
MaxUnits
SchoolName
Instructor2Name
Meets6
MinUnits
DepartmentCode
ChargeMsg1
TypeName
CourseSubtitle
Meets1
```

- And you can find out the meaning of the columns on CU's Open Data site.

```
"Term": Five digit term code, year then semester 1=Spring, 2=Summer,
3=Fall. (e.g. 20131 = Spring 2013)
    "Course": Subject, Course and Section number
    "PrefixName":
    "DivisionCode": Code of the division providing the course
    "DivisionName": Name of the division providing the course
    "CampusCode": Code of the campus the course is on
    "CampusName": Name of the campus the course is on
    "SchoolCode": Code of the school providing the course
    "SchoolName": Name of the school providing the course
    "DepartmentCode": Code of the department providing the course
    "DepartmentName": Name of the department providing the course
    "SubtermCode":
    "SubtermName":
    "CallNumber": Registration call number of the couse
    "NumEnrolled": number currently enrolled
    "MaxSize": max enrollment size
    "EnrollmentStatus": O=Open C=Closed
    "NumFixedUnits": default credits for the course
    "MinUnits":
    "MaxUnits":
    "CourseTitle": Title of the couse
    "CourseSubtitle": Subtitle of the course
    "TypeCode": "LC",
    "TypeName": "LECTURE",
    "Approval": if approval is needed
    "BulletinFlags": "B",
    "ClassNotes": "",
    "Meets1": "",
    "Meets2": "",
    "Meets3": "",
    "Meets4": "",
    "Meets5": "",
    "Meets6": "",
    "Instructor1Name": "NISSIM, DORON",
    "Instructor2Name": "",
    "Instructor3Name": "",
    "Instructor4Name": "",
    "PrefixLongname": "ACCOUNTING",
```

```
    "ExamMeet": "",
    "ExamDate": "",
    "ChargeMsg1": "",
    "ChargeAmt1": "",
    "ChargeMsg2": "",
    "ChargeAmt2": ""
```

- Now, while laughing diabolically, I had started to write a question asking you to create a schema, factor the data into multiple tables, clean up the data and set types, set keys and constraints, etc.
- Performing the work to answer this question took me two hours. I find that students typical need 10X as much time as I need. The diabolical Riddler laughter really took off when I was done.
- But, something a lot like this happened. ... ...

- So, I deleted the question and decided to make it one of the extra-credit assignments later in the semester.
- Do not worry about this question now. I will publish as extra-credit later in the semester.

Putting it All Together

A1

- Sadly for you, Batman seems to think I learned my lesson and can be trusted to define reasonable questions. We all know that is not true. So, when you think about it, this horrible question is at least partly Batman's fault.
- The midterm directory contains a file `people_info.csv`. The columns are:
  - `first_name`
  - `middle_name`

- `last_name`
- `email`
- `employee_type`, which can be one of `Professor, Lecturer, Staff.` The value is empty of the person is a student.
- `enrollment_year` which must be in the range `2016 - 2022.` The value is empty of the person is an employee.

- If `enrollment_year` is NULL, the person is a `Student` and `employee_type` must be NULL. If `employee_type` is not NULL, then the person is NOT a student and `enrollment_year` must be NULL.

- You must implement a two-table solution to the inheritance pattern. This means that your solution will have tables `students` and `employees`, and will have a view `people.`

- Your solution must implement the following tasks and meet the following criteria.
  1. You must implement the two tables and views, including reasonable data types and constraints. This must include an additional column `uni` that we explain below. You may have additional columns if that helps.
  2. You must implement four stored procedures. The implementations of the four procedures are almost identical.
     A. The procedures are:
        a. create_student(first_name, middle_name, last_name, email, enrollment_year, uni, guid)
        b. create_employee(first_name, middle_name, last_name, email, employee_type, uni, guid)
        c. update_student(uni, first_name, middle_name, last_name, email, enrollment_year)
        d. update_employee(uni, first_name, middle_name, last_name, email, employee_type)
     B. For updates, the procedures ignore input parameters that are NULL and only apply the non-NULL values. The procedure does not update the `uni.`
     C. The create procedures must generate the value for the `GUID` and `uni` and return them as out values.
  3. `employee_type` must be one of `Professor, Lecturer, Staff.`
  4. `enrollment_year` must be in the range `2016 - 2022` inclusive.
  5. Only `middle_name` can be null.
  6. uni:
     A. Must be of the form "FMLnnnn" where "F" is the first letter of the first name, "M" is the first letter of the middle_name (if not NULL), "L" is the first letter of the last name.
     B. For any combination of letters, the numbers following the letters must start at 1 and increase. That is "DFF1", "AB1", "DFF2", "CD1", "CAD1", "CD2", ...
     C. You must implement a function to generate the `uni` and a trigger on the relevant tables to automatically set the `uni` and `GUID` on INSERT. You must implement a trigger that prevents changing the `uni` or `GUID.`
  7. `email` must be unique over both `student` and `employee.`

8. You must use security to disable calling `INSERT, DELETE` and `UPDATE` on the underlying tables for any user other than `root` or `dbuser.` You must test this by creating an additional user `general_user` and connecting as that user.

- You must demonstrate correct implementation by loading the data, using select statements, attempting insert/update/delete, ...
- The following code will load the tables for you if you have properly implemented the tables and functions.
- The execution also helps you understand how to test procedures, etc.

In [14]:
```
%%sql
Drop schema cx2273_s22_midterm_A;
create schema if not exists  cx2273_s22_midterm_A;
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
1 rows affected.
[]
```

Out[14]:

In [ ]:
```
# Put your create table, procedure, functions, etc. definitions int he following cells
#
```

In [ ]:
```
# Create table using 2 table implementation
```

In [54]:
```
%%sql
use cx2273_s22_midterm_a;

DROP TABLE IF EXISTS`Student`, `Employee`;
CREATE TABLE `Student` (
  `uni` varchar(8),
  `first_name` varchar(16) not null,
  `middle_name` varchar(16),
  `last_name` varchar(16) not null,
  `email` varchar(64) not null,
  `employee_type` enum('Professor', 'Lecturer', 'Staff') null,
  `enrollment_year` year not null,
  `guid` varchar(256) not null,
  PRIMARY KEY (`uni`),
  KEY `UQ` (`email`),
  CONSTRAINT check_student_email
  CHECK (email like '%@%'),
  CONSTRAINT check_enrollment_year
  CHECK (enrollment_year>=2016 and enrollment_year<=2022)
);

CREATE TABLE `Employee` (
  `uni` varchar(8),
  `first_name` varchar(16) not null,
  `middle_name` varchar(16),
  `last_name` varchar(16) not null,
  `email` varchar(64),
  `employee_type` enum('Professor', 'Lecturer', 'Staff') not null,
  `enrollment_year` year null,
  `guid` varchar(256) not null,
  PRIMARY KEY (`uni`),
```

```
  KEY `UQ` (`email`),
  CONSTRAINT check_employee_email
  CHECK (email like '%@%')
);

DROP VIEW IF EXISTS people;
CREATE VIEW people as
    SELECT uni, first_name, middle_name, last_name, email, employee_type, enrollment_
        FROM Student
    UNION
    SELECT uni, first_name, middle_name, last_name, email, employee_type, enrollment_
        FROM Employee;
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
```

Out[54]: []

In [ ]:
```
# Constraint of these 2 tables
```

In [58]:
```
%%sql
use cx2273_s22_midterm_a;
Alter table students drop constraint chk_email_1;
Alter table employees drop constraint chk_email_2;
Alter table students drop constraint chk_year;
Alter table employees drop constraint check_semester;

Alter table students add constraint chk_email_1 check (email like '%_@_%.edu%');
Alter table employees add constraint chk_email_2 check (email like '%_@_%.edu%');
Alter table students add constraint chk_year check (enrollment_year >= 2016 and e
Alter table employees add constraint check_semester check (employee_type in ('Pro
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
```

Out[58]: []

In [ ]:
```
# Functions generating uni(derivate from Professor's lecture)
```

In [64]:
```
%%sql
use cx2273_s22_midterm_a;
DROP FUNCTION IF EXISTS compute_uni;
CREATE DEFINER=`root`@`localhost` FUNCTION `compute_uni`(f_name varchar(16),
                                                         l_name varchar(16))
    RETURNS varchar(12) CHARSET utf8mb4
        DETERMINISTIC
    BEGIN
```

```
        declare result_uni varchar(12);
        declare f_initial, l_initial char(1);
        declare uni_match_count int;
        declare uni_match_count1 int;
        declare uni_match_count2 int;
        declare uni_prefix char(3);

        set f_initial = upper(substr(f_name,1,1));
        set l_initial = upper(substr(l_name,1,1));

        set uni_prefix = concat(f_initial, l_initial,'%');

        select count(*) into uni_match_count1 from Student
            where uni like uni_prefix;

        select count(*) into uni_match_count2 from Employee
            where uni like uni_prefix;

        set uni_match_count = uni_match_count1+uni_match_count2+1;

        set result_uni = concat(f_initial, l_initial, uni_match_count);

    RETURN result_uni;
    END
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
0 rows affected.
0 rows affected.
[]
```

Out[64]:

In [ ]:

```
# Procedures create_student
```

In [59]:

```
%%sql
use cx2273_s22_midterm_a;

DROP PROCEDURE IF EXISTS create_student;
CREATE DEFINER=`root`@`localhost` PROCEDURE `create_student`(first_name varchar(1
                                                    middle_name varchar(16),
                                                    last_name varchar(16),
                                                    email varchar(64),
                                                    enrollment_year year,
                                                    uni varchar(12),
                                                    guid varchar(256))

    BEGIN

        declare computed_uni varchar(12);
        declare new_guid varchar(256);

        set computed_uni = compute_uni(first_name, last_name);
        select uuid() into new_guid;

        INSERT INTO Student (uni, first_name, middle_name, last_name, email, enrollm
            VALUES (computed_uni, first_name, middle_name, last_name, email, enrollme

    END
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
```

```
0 rows affected.
0 rows affected.
[]
```

Out[59]:

In [ ]:
```
# Procedures create_employee
```

In [60]:
```
%%sql

DROP PROCEDURE IF EXISTS create_employee;
CREATE DEFINER=`root`@`localhost` PROCEDURE `create_employee`(first_name varchar(
                                                              middle_name varchar(16),
                                                              last_name varchar(16),
                                                              email varchar(64),
                                                              employee_type enum('Prof
                                                              uni varchar(12),
                                                              guid varchar(256))
    BEGIN

        declare computed_uni varchar(12);
        declare new_guid varchar(256);

        set computed_uni = compute_uni(first_name, last_name);
        select uuid() into new_guid;

        INSERT INTO Employee (uni, first_name, middle_name, last_name, email, employ
        VALUES (computed_uni, first_name, middle_name, last_name, email, employee_typ

    END
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
0 rows affected.
[]
```

Out[60]:

In [ ]:
```
# Procedures update_student
```

In [62]:
```
%%sql
use cx2273_s22_midterm_a;

DROP PROCEDURE IF EXISTS update_student;
CREATE DEFINER=`root`@`localhost` PROCEDURE update_student( IN uni_in varchar(8),
                                                            IN first_name_in varchar(
                                                            IN middle_name_in varchar
                                                            IN last_name_in varchar(16
                                                            IN email_in varchar(64),
                                                            IN enrollment_year_in yea
                                                            )
    BEGIN

        UPDATE student
            set first_name = if(first_name_in is NOT NULL and first_name != first_n
                middle_name = if(middle_name_in is NOT NULL and middle_name != midd
                last_name = if(last_name_in is NOT NULL and last_name != last_name_
                email = if(email_in is NOT NULL and email != email_in, email_in, ema
                enrollment_year = if(enrollment_year_in is NOT NULL and enrollment_y
        WHERE uni = uni_in;

    END;
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
0 rows affected.
0 rows affected.
[]
```

Out[62]:

In [ ]:
```
# Procedures update_employee
```

In [63]:
```
%%sql
use cx2273_s22_midterm_a;

DROP PROCEDURE IF EXISTS update_employee;
CREATE DEFINER=`root`@`localhost` PROCEDURE update_employee(IN uni_in varchar(8),
                                                            IN first_name_in varchar(
                                                            IN middle_name_in varchar
                                                            IN last_name_in varchar(1
                                                            IN email_in varchar(64),
                                                            IN employee_type_in ENUM('
                                                            )
    BEGIN

        UPDATE student
            set first_name = if(first_name_in is NOT NULL and first_name != first_n
                middle_name = if(middle_name_in is NOT NULL and middle_name != midd
                last_name = if(last_name_in is NOT NULL and last_name != last_name_
                email = if(email_in is NOT NULL and email != email_in, email_in, em
                employee_type = if(employee_type_in is NOT NULL and employee_type !=
        WHERE uni = uni_in;
    END;
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
0 rows affected.
0 rows affected.
[]
```

Out[63]:

In [ ]:
```
# Trigger when insert to compte uni
```

In [67]:
```
%%sql

DROP TRIGGER IF EXISTS cx2273_s22_midterm_a.Student_BEFORE_INSERT;
CREATE DEFINER = CURRENT_USER
    TRIGGER `cx2273_s22_midterm_a`.`Student_BEFORE_INSERT` BEFORE INSERT ON `Stude
    FOR EACH ROW

BEGIN
    set new.uni = compute_uni(new.first_name, new.last_name);
    set new.guid = uuid();
END
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
0 rows affected.
[]
```

Out[67]:

In [66]:
```
%%sql
```

```
DROP TRIGGER IF EXISTS cx2273_s22_midterm_a.Employee_BEFORE_INSERT;
CREATE DEFINER = CURRENT_USER
    TRIGGER `cx2273_s22_midterm_a`.`Employee_BEFORE_INSERT` BEFORE INSERT ON `Empl
    FOR EACH ROW

BEGIN
    set new.uni = compute_uni(new.first_name, new.last_name);
    set new.guid = uuid();
END
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
0 rows affected.
[]
```

Out[66]:

In [ ]:

```
# Security setup
```

In [49]:

```
%%sql

drop user if exists root;
create user root identified by 'dbuser';

grant
    update,
    insert,
    delete
on cx2273_s22_midterm_a.*
to root
with grant option;
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
0 rows affected.
0 rows affected.
[]
```

Out[49]:

In [ ]:

```
# Triggers prevent constraint
```

In [46]:

```
%%sql
use cx2273_s22_midterm_a;
drop trigger if exists insert_students_trigger;
create trigger insert_students_trigger before insert on Students
for each row begin
    set @cc =
    (
    SELECT COUNT(*) FROM people WHERE email = NEW.email
    );
    if (@cc > 0) then
        signal sqlstate '50000' set message_text = 'Email is not Unique', mysql_er
    end if;
    insert into Students
    (uni, guid)
    values(uni_generator(NEW.first_name, NEW.middle_name, NEW.last_name), uuid());
end
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
```

```
0 rows affected.
0 rows affected.
[]
```

Out[46]:

In [50]:

```
%%sql
use cx2273_s22_midterm_a;
drop trigger if exists insert_employees_trigger;
create trigger insert_employees_trigger before insert on Employees
for each row begin
    set @cc =
    (
    SELECT COUNT(*) FROM people WHERE email = NEW.email
    );
    if (@cc > 0) then
        signal sqlstate '50000' set message_text = 'Email is not Unique', mysql_er
    end if;
    insert into Employees
    (uni, guid)
    values(uni_generator(NEW.first_name, NEW.middle_name, NEW.last_name), uuid());
end
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
0 rows affected.
0 rows affected.
[]
```

Out[50]:

In [52]:

```
%%sql
use cx2273_s22_midterm_a;
drop trigger if exists update_students_trigger;
create trigger update_students_trigger before update on Students
for each row begin
    if (NEW.uni != OLD.uni) then
        signal sqlstate '50000' set message_text = 'UNI cannot be changed', mysql_
    end if;
    if (NEW.guid is NULL or NEW.guid!= OLD.guid) then
        signal sqlstate '50000' set message_text = 'GUID cannot be changed', mysql
    end if;
    set @cc = 0;
    select COUNT(*) INTO @cc FROM people WHERE email = NEW.email;
    if (@cc > 0) then
        signal sqlstate '50000' set message_text = 'Email is not Unique', mysql_er
    end if;
end
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
0 rows affected.
0 rows affected.
[]
```

Out[52]:

In [53]:

```
%%sql
use cx2273_s22_midterm_a;
drop trigger if exists update_employees_trigger;
create trigger update_employees_trigger before update on Employees
for each row begin
    if (NEW.uni != OLD.uni) then
        signal sqlstate '50000' set message_text = 'UNI cannot be changed', mysql_
    end if;
    if (NEW.guid is NULL or NEW.guid!= OLD.guid) then
```

```
        signal sqlstate '50000' set message_text = 'GUID cannot be changed', mysql
    end if;
    set @cc = 0;
    select COUNT(*) INTO @cc FROM people WHERE email = NEW.email;
    if (@cc > 0) then
        signal sqlstate '50000' set message_text = 'Email is not Unique', mysql_er
    end if;
end
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
0 rows affected.
0 rows affected.
[]
```

Out[53]:

In [70]:

```
# The following only works if you have already created the tables, procedures, etc.
# So, you mus have answered the questions to run the load.
#
%sql use cx2273_s22_midterm_a;
%sql delete from student;
%sql delete from employee;
%sql select 1;
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
 * mysql+pymysql://root:***@localhost
0 rows affected.
 * mysql+pymysql://root:***@localhost
0 rows affected.
 * mysql+pymysql://root:***@localhost
1 rows affected.
```

Out[70]:

1
1

In [71]:

```
# Load the information from the file and show sample values.
import csv

people_info = []
with open("people_info.csv", "r") as in_file:
    d_rdr = csv.DictReader(in_file)
    for r in d_rdr:
        people_info.append(dict(r))

people_info_df = pd.DataFrame(people_info)
people_info_df
```

Out[71]:

	first_name	middle_name	last_name	email	employee_type	enrollmen
0	Sanders	Arline	Breckell	abreckell1x@fotki.com	Professor	
1	Zared		Fenelon	afenelona@themeforest.net		
2	Ethelin		Fidele	afidele12@google.ru	Lecturer	
3	Bibbye	Annabal	Guesford	aguesfordb@tumblr.com		
4	Xenia	Ardella	Kief	akieft@free.fr	Staff	
...	...	...	...	...	...	
95	Norry		Rubinchik	trubinchik16@howstuffworks.com		

	first_name	middle_name	last_name	email	employee_type	enrollmen
96	Doug		Medforth	vmedforth1o@homestead.com	Staff	
97	Gerty		O'Donegan	vodoneganf@clickbank.net		
98	Anabelle	Wallas	Quimby	wquimby1c@nba.com		
99	Sasha	Win	Ruffli	wruffli2q@wordpress.com	Lecturer	

100 rows × 6 columns

In [72]:
```
people_info[-5:]
```

Out[72]:
```
[{'first_name': 'Norry',
  'middle_name': '',
  'last_name': 'Rubinchik',
  'email': 'trubinchik16@howstuffworks.com',
  'employee_type': '',
  'enrollment_year': '2016'},
 {'first_name': 'Doug',
  'middle_name': '',
  'last_name': 'Medforth',
  'email': 'vmedforth1o@homestead.com',
  'employee_type': 'Staff',
  'enrollment_year': ''},
 {'first_name': 'Gerty',
  'middle_name': '',
  'last_name': "O'Donegan",
  'email': 'vodoneganf@clickbank.net',
  'employee_type': '',
  'enrollment_year': '2020'},
 {'first_name': 'Anabelle',
  'middle_name': 'Wallas',
  'last_name': 'Quimby',
  'email': 'wquimby1c@nba.com',
  'employee_type': '',
  'enrollment_year': '2022'},
 {'first_name': 'Sasha',
  'middle_name': 'Win',
  'last_name': 'Ruffli',
  'email': 'wruffli2q@wordpress.com',
  'employee_type': 'Lecturer',
  'enrollment_year': ''}]
```

In [73]:
```
# If you have defined your procedures, functions, tables and constraints correctly, yo
# to load your data.
#
import copy

def add_person(p):
    """
    p is a dictionary containing the column values for either a student or an employee
    """

    cur = sql_conn.cursor()

    # This function changes the data, converting '' to None.
    # So, we make a copy and change the copy.
    p_dict = copy.copy(p)
    for k,v in p_dict.items():
```

```
        if v == '':
            p_dict[k] = None

    # Is the person a student?
    #
    if p_dict['employee_type'] is None:

        # This provides a hint for what your stored procedure will look like.
        res = cur.callproc("cx2273_s22_midterm_a.create_student",
                           # The following are in parameters
                           (p_dict['first_name'],
                           p_dict['middle_name'],
                           p_dict['last_name'],
                            p_dict['email'],
                           p_dict['enrollment_year'],
                            # The following are out parameters for uni and GUID.
                           None,
                           None))

        # After the procedure executes, the following query will select the out values
        res = cur.execute("""SELECT @_cx2273_s22_midterm_a.create_student_5,
                              @_cx2273_s22_midterm_a.create_student_6""")
        result = cur.fetchall()

    # The following does the same for employee.
    elif p_dict['enrollment_year'] is None:
        res = cur.callproc("cx2273_s22_midterm_a.create_employee",
                           (p_dict['first_name'],
                           p_dict['middle_name'],
                           p_dict['last_name'],
                            p_dict['email'],
                           p_dict['employee_type'],
                           None,
                           None))
        res = cur.execute("""SELECT @_dff9_s22_midterm.create_student_5,
                              @_dff9_s22_midterm.create_student_6""")
        result = cur.fetchall()
    else:
        print("Huh")
        result = None

    sql_conn.commit()
    cur.close()
    return result
```

In [74]:

```
for p in people_info:
    add_person(p)
```

In [75]:

```
#
# Test that we loaded people.
#
%sql select * from people;
```

```
 * mysql+pymysql://root:***@localhost
100 rows affected.
```

Out[75]:

uni	first_name	middle_name	last_name	email	employee_type	enrol
AA1	Abbey	Helge	Anyene	hanyene2c@newsvine.com	None	
AB1	Avie	Else	Blissitt	eblissitti@youtu.be	None	

uni	first_name	middle_name	last_name	email	employee_type	enrol
AQ1	Anabelle	Wallas	Quimby	wquimby1c@nba.com	None	
BB1	Base	None	Baybutt	bbaybutty@tmall.com	None	
BE1	Barry	Cullin	Elias	celias1k@scribd.com	None	
BG1	Bibbye	Annabal	Guesford	aguesfordb@tumblr.com	None	
BG2	Bank	Elane	Guerola	eguerola29@blogger.com	None	
BM1	Brigida	Cameron	Maclean	cmaclean13@mac.com	None	
CG1	Courtnay	Elbert	Gillespie	egillespie2m@slideshare.net	None	
CH1	Charline	Ferd	Hartell	fhartell1@meetup.com	None	
CV1	Carroll	None	Venables	rvenablesu@friendfeed.com	None	
DB1	Dougy	None	Burchett	eburchettd@chicagotribune.com	None	
DC1	Dacey	Katuscha	Cranfield	kcranfield1s@nature.com	None	
DM1	Drona	None	McKinie	smckinie1a@springer.com	None	
DP1	Duffy	Auberon	Pounder	apounder2h@reuters.com	None	
DR1	Dyna	Felic	Rozenzweig	frozenzweig2i@rambler.ru	None	
EC1	Esra	Richard	Coghlin	rcoghlinv@qq.com	None	
EH1	Emili	Lynnet	Haddinton	lhaddintonn@over-blog.com	None	
FG1	Fredek	Raddie	Goudie	rgoudie5@blogs.com	None	
GA1	Granville	Guinevere	Adolphine	gadolphine11@instagram.com	None	
GO1	Gerty	None	O'Donegan	vodoneganf@clickbank.net	None	
IL1	Ira	None	Linacre	jlinacreo@webnode.com	None	
JH1	Jenine	Berry	Habberjam	bhabberjam2k@examiner.com	None	
KF1	Kerwin	Conrade	Foort	cfoortw@vinaora.com	None	
KG1	Kermit	None	Glanders	dglanders15@booking.com	None	
KG2	Kearney	Selie	Gino	sgino1l@bluehost.com	None	
LB2	Leann	Gabriellia	Burdas	gburdas1i@indiegogo.com	None	
LI1	Lynett	Myrwyn	Ionesco	mionesco2n@moonfruit.com	None	
LT1	Levey	Ottilie	Tomlins	otomlins27@icq.com	None	
MI1	Mal	Jermaine	Issett	jissett2e@freewebs.com	None	
MW1	Meridel	Rawley	Willatts	rwillatts1t@si.edu	None	
NG1	Nonna	None	Gillanders	tgillanders9@example.com	None	
NR1	Nickolai	Herby	Reddyhoff	hreddyhoff1g@vk.com	None	
NR2	Norry	None	Rubinchik	trubinchik16@howstuffworks.com	None	
QT1	Quent	None	Threadgall	cthreadgall1v@booking.com	None	
RC1	Reece	Corbett	Caps	ccapsz@telegraph.co.uk	None	
RV1	Rozelle	None	Vigar	mvigar8@vistaprint.com	None	
SB2	Shea	Gail	Bates	gbates1e@alibaba.com	None	

uni	first_name	middle_name	last_name	email	employee_type	enrol
SS1	Sada	None	Stidson	gstidsonp@ebay.com	None	
SS2	Sterne	Sasha	Spooner	sspooner1f@wordpress.org	None	
TC1	Travis	Melba	Colbourne	mcolbourne2j@mapquest.com	None	
TO1	Tobiah	Gerek	Offield	goffield1u@addthis.com	None	
TP1	Trix	None	Pretswell	jpretswell1h@xrea.com	None	
VM1	Vincenty	Elisabeth	Marousek	emarousek10@huffingtonpost.com	None	
VM2	Vance	None	Maffioni	fmaffioni1r@amazon.com	None	
VS1	Veronika	Cordey	Simeons	csimeons2@microsoft.com	None	
WK1	Willi	None	Kerby	ekerby1p@amazon.com	None	
WM1	Woodrow	Camile	Moughtin	cmoughtin17@illinois.edu	None	
WS1	Winslow	None	Scrauniage	jscrauniage28@aol.com	None	
ZF1	Zared	None	Fenelon	afenelona@themeforest.net	None	
AS1	Ari	Rheta	Sellek	rsellek6@oakley.com	Lecturer	
AW1	Ailbert	Danie	Warmisham	dwarmishame@soundcloud.com	Staff	
BH1	Bettina	Sonya	Higgonet	shiggonet2b@163.com	Lecturer	
BM2	Bamby	Rubetta	Mabbs	rmabbs4@xing.com	Lecturer	
BS1	Bonny	None	Scheffel	lscheffel7@taobao.com	Professor	
CG2	Clim	None	Guislin	lguislin2o@chicagotribune.com	Professor	
CL1	Cari	Andriana	Leask	aleask1n@devhub.com	Lecturer	
CS1	Christie	None	Siegertsz	hsiegertsz21@instagram.com	Professor	
CT1	Carmine	None	Tolman	gtolmanr@slideshare.net	Staff	
CW1	Carey	Maudie	Wyrall	mwyrallj@scientificamerican.com	Staff	
DA1	Doyle	None	Aslin	jaslin24@redcross.org	Lecturer	
DM2	Doug	None	Medforth	vmedforth1o@homestead.com	Staff	
DS1	Duncan	Shellie	Sillars	ssillars2l@unicef.org	Professor	
DW1	Darrin	Mario	Wynrahame	mwynrahamem@admin.ch	Professor	
EF1	Ethelin	None	Fidele	afidele12@google.ru	Lecturer	
EM1	Electra	Krystle	Morfell	kmorfell2g@istockphoto.com	Professor	
FM1	Francene	None	MacNeely	nmacneely22@cpanel.net	Lecturer	
GB1	Gisela	None	Blagden	gblagden1q@buzzfeed.com	Professor	
GP1	Genni	None	Purbrick	lpurbrick25@canalblog.com	Professor	
HC1	Hobart	Dominic	Croal	dcroalx@purevolume.com	Professor	
HH1	Holli	None	Henstridge	lhenstridgeh@sogou.com	Lecturer	
JB1	Jacky	Nydia	Bolles	nbolles23@ucoz.ru	Staff	
JJ1	Jany	Sherry	Johl	sjohlg@soundcloud.com	Professor	
KB1	Karon	None	Bree	ebree1z@creativecommons.org	Professor	

uni	first_name	middle_name	last_name	email	employee_type	enrol
KI1	Karole	None	Inkin	minkinc@google.de	Staff	
KM1	Kristin	None	Malacrida	smalacrida1w@economist.com	Staff	
KP1	Kahaleel	Meg	Penzer	mpenzer14@dailymail.co.uk	Professor	
LB1	Lemmy	Burr	Bradnocke	bbradnockek@nifty.com	Lecturer	
LF1	Lu	Cinnamon	Flaxman	cflaxman1b@cdbaby.com	Lecturer	
LF2	Lelah	Ellette	Fulk	efulk1d@discuz.net	Staff	
LS1	Lilllie	None	Snodin	jsnodin20@princeton.edu	Lecturer	
MC1	Maybelle	Esteban	Cella	ecella26@mail.ru	Staff	
MF1	Marylin	Darcy	Favey	dfavey2p@mozilla.com	Staff	
MK1	Meghan	None	Kleinschmidt	lkleinschmidtl@squarespace.com	Lecturer	
MS1	Matthieu	Kalle	Slipper	kslipper2f@nytimes.com	Staff	
MT1	Mendie	None	Tennick	ltennick3@aboutads.info	Staff	
NF1	Niki	Gardiner	Form	gform18@blogger.com	Staff	
NT1	Nadia	None	Trehearn	ktrehearn19@tinyurl.com	Lecturer	
OH1	Olwen	None	Hedley	jhedley1m@disqus.com	Staff	
PL1	Pattie	None	Lyles	slyles1j@amazon.de	Staff	
RC2	Robinett	Jami	Charte	jcharte1y@merriam-webster.com	Staff	
RP1	Renae	Jaquith	Plessing	jplessing0@samsung.com	Staff	
SB1	Sanders	Arline	Breckell	abreckell1x@fotki.com	Professor	
SH1	Suki	None	Hellyar	ghellyar2a@cornell.edu	Staff	
SL1	Sibylle	Bearnard	Lalley	blalley2d@rediff.com	Lecturer	
SM1	Sayers	Karon	McKnish	kmcknishs@reddit.com	Lecturer	
SR1	Sasha	Win	Ruffli	wruffli2q@wordpress.com	Lecturer	
WY1	Wells	None	Yousef	gyousef2r@spotify.com	Professor	
XK1	Xenia	Ardella	Kief	akieft@free.fr	Staff	
YM1	Yehudi	Sile	McColley	smccolleyq@amazon.com	Staff	

In [76]:

```
#
# Test that we loaded students.
#
%sql select * from student;
```

```
 * mysql+pymysql://root:***@localhost
50 rows affected.
```

Out[76]:

uni	first_name	middle_name	last_name	email	employee_type	enroll
AA1	Abbey	Helge	Anyene	hanyene2c@newsvine.com	None	

uni	first_name	middle_name	last_name	email	employee_type	enroll
AB1	Avie	Else	Blissitt	eblissitti@youtu.be	None	
AQ1	Anabelle	Wallas	Quimby	wquimby1c@nba.com	None	
BB1	Base	None	Baybutt	bbaybutty@tmall.com	None	
BE1	Barry	Cullin	Elias	celias1k@scribd.com	None	
BG1	Bibbye	Annabal	Guesford	aguesfordb@tumblr.com	None	
BG2	Bank	Elane	Guerola	eguerola29@blogger.com	None	
BM1	Brigida	Cameron	Maclean	cmaclean13@mac.com	None	
CG1	Courtnay	Elbert	Gillespie	egillespie2m@slideshare.net	None	
CH1	Charline	Ferd	Hartell	fhartell1@meetup.com	None	
CV1	Carroll	None	Venables	rvenablesu@friendfeed.com	None	
DB1	Dougy	None	Burchett	eburchettd@chicagotribune.com	None	
DC1	Dacey	Katuscha	Cranfield	kcranfield1s@nature.com	None	
DM1	Drona	None	McKinie	smckinie1a@springer.com	None	

uni	first_name	middle_name	last_name	email	employee_type	enroll
DP1	Duffy	Auberon	Pounder	apounder2h@reuters.com	None	
DR1	Dyna	Felic	Rozenzweig	frozenzweig2i@rambler.ru	None	
EC1	Esra	Richard	Coghlin	rcoghlinv@qq.com	None	
EH1	Emili	Lynnet	Haddinton	lhaddintonn@over-blog.com	None	
FG1	Fredek	Raddie	Goudie	rgoudie5@blogs.com	None	
GA1	Granville	Guinevere	Adolphine	gadolphine11@instagram.com	None	
GO1	Gerty	None	O'Donegan	vodoneganf@clickbank.net	None	
IL1	Ira	None	Linacre	jlinacreo@webnode.com	None	
JH1	Jenine	Berry	Habberjam	bhabberjam2k@examiner.com	None	
KF1	Kerwin	Conrade	Foort	cfoortw@vinaora.com	None	
KG1	Kermit	None	Glanders	dglanders15@booking.com	None	
KG2	Kearney	Selie	Gino	sgino1l@bluehost.com	None	
LB2	Leann	Gabriellia	Burdas	gburdas1i@indiegogo.com	None	

uni	first_name	middle_name	last_name	email	employee_type	enroll
LI1	Lynett	Myrwyn	Ionesco	mionesco2n@moonfruit.com	None	
LT1	Levey	Ottilie	Tomlins	otomlins27@icq.com	None	
MI1	Mal	Jermaine	Issett	jissett2e@freewebs.com	None	
MW1	Meridel	Rawley	Willatts	rwillatts1t@si.edu	None	
NG1	Nonna	None	Gillanders	tgillanders9@example.com	None	
NR1	Nickolai	Herby	Reddyhoff	hreddyhoff1g@vk.com	None	
NR2	Norry	None	Rubinchik	trubinchik16@howstuffworks.com	None	
QT1	Quent	None	Threadgall	cthreadgall1v@booking.com	None	
RC1	Reece	Corbett	Caps	ccapsz@telegraph.co.uk	None	
RV1	Rozelle	None	Vigar	mvigar8@vistaprint.com	None	
SB2	Shea	Gail	Bates	gbates1e@alibaba.com	None	
SS1	Sada	None	Stidson	gstidsonp@ebay.com	None	
SS2	Sterne	Sasha	Spooner	sspooner1f@wordpress.org	None	

uni	first_name	middle_name	last_name	email	employee_type	enroll
TC1	Travis	Melba	Colbourne	mcolbourne2j@mapquest.com	None	
TO1	Tobiah	Gerek	Offield	goffield1u@addthis.com	None	
TP1	Trix	None	Pretswell	jpretswell1h@xrea.com	None	
VM1	Vincenty	Elisabeth	Marousek	emarousek10@huffingtonpost.com	None	
VM2	Vance	None	Maffioni	fmaffioni1r@amazon.com	None	
VS1	Veronika	Cordey	Simeons	csimeons2@microsoft.com	None	
WK1	Willi	None	Kerby	ekerby1p@amazon.com	None	
WM1	Woodrow	Camile	Moughtin	cmoughtin17@illinois.edu	None	
WS1	Winslow	None	Scrauniage	jscrauniage28@aol.com	None	
ZF1	Zared	None	Fenelon	afenelona@themeforest.net	None	

In [31]:
```
#
# Test that we loaded employees.
#
%sql select * from employee;
```

```
 * mysql+pymysql://dbuser:***@localhost
50 rows affected.
```

Out[31]:

id	first_name	middle_name	last_name	email	uni	emplo
103e6a74-a882-11ec-	Sanders	Arline	Breckell	abreckell1x@fotki.com	SAB1	

a3ad-edb7cbd2bb34					
103f2a54-a882-11ec-a3ad-edb7cbd2bb34	Ethelin	None	Fidele	afidele12@google.ru	EF1
103fa696-a882-11ec-a3ad-edb7cbd2bb34	Xenia	Ardella	Kief	akieft@free.fr	XAK1
103fe1ec-a882-11ec-a3ad-edb7cbd2bb34	Cari	Andriana	Leask	aleask1n@devhub.com	CAL1
10409f88-a882-11ec-a3ad-edb7cbd2bb34	Lemmy	Burr	Bradnocke	bbradnockek@nifty.com	LBB1
10411828-a882-11ec-a3ad-edb7cbd2bb34	Sibylle	Bearnard	Lalley	blalley2d@rediff.com	SBL1
1041cc50-a882-11ec-a3ad-edb7cbd2bb34	Lu	Cinnamon	Flaxman	cflaxman1b@cdbaby.com	LCF1
1043122c-a882-11ec-a3ad-edb7cbd2bb34	Hobart	Dominic	Croal	dcroalx@purevolume.com	HDC1
10434364-a882-11ec-a3ad-edb7cbd2bb34	Marylin	Darcy	Favey	dfavey2p@mozilla.com	MDF1
1043a598-a882-11ec-a3ad-edb7cbd2bb34	Ailbert	Danie	Warmisham	dwarmishame@soundcloud.com	ADW1
10440c22-a882-11ec-a3ad-edb7cbd2bb34	Karon	None	Bree	ebree1z@creativecommons.org	KB1
10448a62-a882-11ec-a3ad-edb7cbd2bb34	Maybelle	Esteban	Cella	ecella26@mail.ru	MEC1
1044d36e-a882-11ec-a3ad-edb7cbd2bb34	Lelah	Ellette	Fulk	efulk1d@discuz.net	LEF1
10471598-a882-11ec-a3ad-edb7cbd2bb34	Gisela	None	Blagden	gblagden1q@buzzfeed.com	GB1

10479086-a882-11ec-a3ad-edb7cbd2bb34	Niki	Gardiner	Form	gform18@blogger.com	NGF1
1047c452-a882-11ec-a3ad-edb7cbd2bb34	Suki	None	Hellyar	ghellyar2a@cornell.edu	SH1
10486628-a882-11ec-a3ad-edb7cbd2bb34	Carmine	None	Tolman	gtolmanr@slideshare.net	CT1
10489936-a882-11ec-a3ad-edb7cbd2bb34	Wells	None	Yousef	gyousef2r@spotify.com	WY1
1049298c-a882-11ec-a3ad-edb7cbd2bb34	Christie	None	Siegertsz	hsiegertsz21@instagram.com	CS1
10495588-a882-11ec-a3ad-edb7cbd2bb34	Doyle	None	Aslin	jaslin24@redcross.org	DA2
104989f4-a882-11ec-a3ad-edb7cbd2bb34	Robinett	Jami	Charte	jcharte1y@merriam-webster.com	RJC1
1049b9f6-a882-11ec-a3ad-edb7cbd2bb34	Olwen	None	Hedley	jhedley1m@disqus.com	OH1
104a4b96-a882-11ec-a3ad-edb7cbd2bb34	Renae	Jaquith	Plessing	jplessing0@samsung.com	RJP1
104ad9e4-a882-11ec-a3ad-edb7cbd2bb34	Lilllie	None	Snodin	jsnodin20@princeton.edu	LS1
104b39de-a882-11ec-a3ad-edb7cbd2bb34	Sayers	Karon	McKnish	kmcknishs@reddit.com	SKM1
104b6ee0-a882-11ec-a3ad-edb7cbd2bb34	Electra	Krystle	Morfell	kmorfell2g@istockphoto.com	EKM1
104b9cda-a882-11ec-a3ad-edb7cbd2bb34	Matthieu	Kalle	Slipper	kslipper2f@nytimes.com	MKS1
104bd38a-a882-11ec-	Nadia	None	Trehearn	ktrehearn19@tinyurl.com	NT1

a3ad-edb7cbd2bb34					
104c009e-a882-11ec-a3ad-edb7cbd2bb34	Clim	None	Guislin	lguislin2o@chicagotribune.com	CG1
104c5eae-a882-11ec-a3ad-edb7cbd2bb34	Holli	None	Henstridge	lhenstridgeh@sogou.com	HH1
104c902c-a882-11ec-a3ad-edb7cbd2bb34	Meghan	None	Kleinschmidt	lkleinschmidtl@squarespace.com	MK2
104cbcaa-a882-11ec-a3ad-edb7cbd2bb34	Genni	None	Purbrick	lpurbrick25@canalblog.com	GP1
104ce9c8-a882-11ec-a3ad-edb7cbd2bb34	Bonny	None	Scheffel	lscheffel7@taobao.com	BS1
104d14ca-a882-11ec-a3ad-edb7cbd2bb34	Mendie	None	Tennick	ltennick3@aboutads.info	MT1
104d6c04-a882-11ec-a3ad-edb7cbd2bb34	Karole	None	Inkin	minkinc@google.de	KI1
104dcb9a-a882-11ec-a3ad-edb7cbd2bb34	Kahaleel	Meg	Penzer	mpenzer14@dailymail.co.uk	KMP1
104e268a-a882-11ec-a3ad-edb7cbd2bb34	Darrin	Mario	Wynrahame	mwynrahamem@admin.ch	DMW1
104e64b0-a882-11ec-a3ad-edb7cbd2bb34	Carey	Maudie	Wyrall	mwyrallj@scientificamerican.com	CMW1
104e98ea-a882-11ec-a3ad-edb7cbd2bb34	Jacky	Nydia	Bolles	nbolles23@ucoz.ru	JNB1
104ed44a-a882-11ec-a3ad-edb7cbd2bb34	Francene	None	MacNeely	nmacneely22@cpanel.net	FM1
105016e8-a882-11ec-a3ad-edb7cbd2bb34	Bamby	Rubetta	Mabbs	rmabbs4@xing.com	BRM1

1050566c-a882-11ec-a3ad-edb7cbd2bb34	Ari	Rheta	Sellek	rsellek6@oakley.com	ARS1
10513686-a882-11ec-a3ad-edb7cbd2bb34	Bettina	Sonya	Higgonet	shiggonet2b@163.com	BSH1
10516af2-a882-11ec-a3ad-edb7cbd2bb34	Jany	Sherry	Johl	sjohlg@soundcloud.com	JSJ1
105199d2-a882-11ec-a3ad-edb7cbd2bb34	Pattie	None	Lyles	slyles1j@amazon.de	PL1
1051c8d0-a882-11ec-a3ad-edb7cbd2bb34	Kristin	None	Malacrida	smalacrida1w@economist.com	KM2
1052055c-a882-11ec-a3ad-edb7cbd2bb34	Yehudi	Sile	McColley	smccolleyq@amazon.com	YSM1
10525aca-a882-11ec-a3ad-edb7cbd2bb34	Duncan	Shellie	Sillars	ssillars2l@unicef.org	DSS1
10531bd6-a882-11ec-a3ad-edb7cbd2bb34	Doug	None	Medforth	vmedforth1o@homestead.com	DM3
1053a128-a882-11ec-a3ad-edb7cbd2bb34	Sasha	Win	Ruffli	wruffli2q@wordpress.com	SWR1

Your tests should be below. You should have test cells that test:

1. Successful execution of each procedure.

2. Execution of update procedures showing that your constraints prevent incorrect data entry and enforce the defined semantics and behavior.

3. Demonstrating that:
   A. You create a user `general_user.`
   B. `general_user` can query the data and call the procedures, but cannot perform `INSERT, UPDATE, DELETE.`

1. Successful execution of each procedure

In [77]:
```
# Insert student
```

```
cur=sql_conn.cursor()

cur.execute("USE cx2273_s22_midterm_a;")

res = cur.callproc("create_student",('Chongzhi', None,'Xu','cx2273@columbia.edu',2021
```

In [78]:
```
%%sql
use cx2273_s22_midterm_a;
select * from Student where first_name='Chongzhi';
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
1 rows affected.
```

Out[78]:

uni	first_name	middle_name	last_name	email	employee_type	enrollment_year
CX1	Chongzhi	None	Xu	cx2273@columbia.edu	None	2021

00

In [79]:
```
# Insert employee
cur=sql_conn.cursor()

cur.execute("USE cx2273_s22_midterm_a;")

res = cur.callproc("create_employee",('Chongzhi', None,'Xu','cx2273@columbia.edu','Pr
```

In [80]:
```
%%sql
use cx2273_s22_midterm_a;
select * from employee where first_name='Chongzhi';
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
1 rows affected.
```

Out[80]:

uni	first_name	middle_name	last_name	email	employee_type	enrollment_year
CX2	Chongzhi	None	Xu	cx2273@columbia.edu	Professor	None

00

In [81]:
```
# Update student
cur=sql_conn.cursor()

cur.execute("USE cx2273_s22_midterm_a;")

res = cur.callproc("update_student",('CX1','Chongzhi', None,'Xu','cx2273@columbia.edu
```

In [82]:
```
%%sql
use cx2273_s22_midterm_a;
select * from Student where first_name='Chongzhi';
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
```

```
1 rows affected.
```

Out[82]:

uni	first_name	middle_name	last_name	email	employee_type	enrollment_year
CX1	Chongzhi	None	Xu	cx2273@columbia.edu	None	2019

00

In [88]:

```
# Update employee
cur=sql_conn.cursor()

cur.execute("USE cx2273_s22_midterm_a;")

res = cur.callproc("update_employee",('CX2','Chongzhi', None,'Xu','cx2273@columbia.ed
```

In [89]:

```
%%sql
use cx2273_s22_midterm_a;
select * from employee where first_name='Chongzhi';
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
1 rows affected.
```

Out[89]:

uni	first_name	middle_name	last_name	email	employee_type	enrollment_year
CX2	Chongzhi	None	Xu	cx2273@columbia.edu	Professor	None

00

2. Execution of update procedures showing that your constraints prevent incorrect data entry and enforce the defined semantics and behavior.

In [ ]:

```
# check employee type
```

In [91]:

```
try:
    cur=sql_conn.cursor()
    cur.execute("USE cx2273_s22_midterm_a;")
    res=cur.execute("UPDATE Employee SET employee_type='cleanning' where uni='LS1'")
except Exception as e:
    print('Error:', e)
```

```
Error: (1265, "Data truncated for column 'employee_type' at row 1")
```

In [ ]:

```
# check_enrollment_year
```

In [90]:

```
try:
    cur=sql_conn.cursor()
    cur.execute("USE cx2273_s22_midterm_a;")
    res=cur.execute("UPDATE Student SET enrollment_year = 2035 where Student.uni='FG1'
```

```
except Exception as e:
    print('Error:', e)
```

```
Error: (3819, "Check constraint 'check_enrollment_year' is violated.")
```

In [ ]:
```
# check wrong value insertion
```

In [92]:
```
try:
    cur=sql_conn.cursor()
    cur.execute("USE cx2273_s22_midterm_a;")
    res=cur.execute("INSERT INTO Student (first_name, last_name, email, employee_type)

except Exception as e:
    print('Error:', e)
```

```
Error: (1364, "Field 'enrollment_year' doesn't have a default value")
```

3. Demonstrating that:

1. You create a user general_user.
2. general_user can query the data and call the procedures, but cannot perform INSERT, UPDATE, DELETE.

In [ ]:
```
# Check the status of user
```

In [93]:
```
%%sql

DELETE FROM mysql.user WHERE user LIKE 'general_user%';

DELETE FROM mysql.user WHERE user LIKE 'student_user%';

SELECT * FROM mysql.user;
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
0 rows affected.
6 rows affected.
```

Out[93]:

Host	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv
%	ChongzhiXu	Y	Y	Y	Y	Y	Y
%	root	N	N	N	N	N	N
localhost	mysql.infoschema	Y	N	N	N	N	N
localhost	mysql.session	N	N	N	N	N	N
localhost	mysql.sys	N	N	N	N	N	N
localhost	root	Y	Y	Y	Y	Y	Y

In [ ]:
```
# add the general user
```

In [94]:
```
%%sql

CREATE USER 'general_user'@'localhost' IDENTIFIED BY 'dbuserdbuser_dff9';

SELECT * FROM mysql.user;
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
7 rows affected.
```

Out[94]:

Host	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv
%	ChongzhiXu	Y	Y	Y	Y	Y	Y
%	root	N	N	N	N	N	N
localhost	general_user	N	N	N	N	N	N
localhost	mysql.infoschema	Y	N	N	N	N	N
localhost	mysql.session	N	N	N	N	N	N
localhost	mysql.sys	N	N	N	N	N	N
localhost	root	Y	Y	Y	Y	Y	Y

In [ ]:
```
# Grant for execution rights of the procedure
```

In [96]:
```
%%sql

GRANT SELECT ON *.* TO 'general_user'@'localhost';

GRANT EXECUTE ON PROCEDURE cx2273_s22_midterm_a.create_student TO 'general_user'@'

GRANT EXECUTE ON PROCEDURE cx2273_s22_midterm_a.create_employee TO 'general_user'@

GRANT EXECUTE ON PROCEDURE cx2273_s22_midterm_a.update_student TO 'general_user'@'

GRANT EXECUTE ON PROCEDURE cx2273_s22_midterm_a.update_employee TO 'general_user'@

GRANT EXECUTE ON FUNCTION cx2273_s22_midterm_a.compute_uni TO 'general_user'@'loca
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
```

Out[96]:
```
[]
```

In [ ]:
```
# check the status and rights of general user
```

In [97]:
```
%%sql

SELECT * FROM mysql.user;
```

```
 * mysql+pymysql://root:***@localhost
7 rows affected.
```

Out[97]:

Host	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv
%	ChongzhiXu	Y	Y	Y	Y	Y	Y
%	root	N	N	N	N	N	N
localhost	general_user	Y	N	N	N	N	N
localhost	mysql.infoschema	Y	N	N	N	N	N
localhost	mysql.session	N	N	N	N	N	N
localhost	mysql.sys	N	N	N	N	N	N
localhost	root	Y	Y	Y	Y	Y	Y

In [ ]:
```
# connect to sql as general user
```

In [98]:
```
%sql mysql+pymysql://general_user:dbuserdbuser_dff9@localhost
```

Out[98]:
```
'Connected: general_user@None'
```

In [106…
```
try:
    cur=sql_conn.cursor()

    cur.execute("USE cx2273_s22_midterm_a;")

    res = cur.callproc("update_student",('CX1','Chongzhi', None,'Xu','cx2273@columbia
    print('General user could have access to procedure update_student!')
except Exception as e:
    print('Error:', e)
    print('General user do not have access to procedure update_student.')
```

```
Error: (1048, "Column 'enrollment_year' cannot be null")
General user do not have access to procedure update_student.
```

In [108…
```
try:
    cur=sql_conn.cursor()
    cur.execute("USE cx2273_s22_midterm_a;")
    res=cur.execute("INSERT INTO Student (first_name, last_name, email, employee_type)
    print('This should have failed')
except Exception as e:
    print('Error:', e)
    print('General user do not have access to insert student.')
```

```
Error: (1364, "Field 'enrollment_year' doesn't have a default value")
General user do not have access to insert student.
```