COMS W4111-002/V02, Spring 22: Take Home Midterm

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 [1]:
```
from IPython.display import Image
```

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

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

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

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

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

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

In [7]:
```
sql = """
    select customerName, customerNumber, city, country from classicmodels.custom
        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
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='dvuserdvuser',
    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. Super Key is a set of attributes that uniquely identifies all attributes, and Candidate key is a subset of a Super Key. Candidate keys are super keys, but all super keys can't be candidate keys.
2. Primary Key is used to uniquely identify a row. Unique key is used to prevent duplicate values in a column.A Unique Key can have NULL values, but a Primary Key cannot.
3. They both can uniquely identify a row. However, Natural keys have real life meanings, while surrogate keys do not.

https://www.mssqltips.com/sqlservertip/5431/surrogate-key-vs-natural-key-differences-and-when-to-use-in-sql-server/

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 columns (data)" means the columns(data) are insert-only that an object alreday in the table cannot be modified.
2. Using immutable columns to define a primary key can avoid the risks of dangling references or orphan records created by the change of primary keys of another related table.

https://www.sswug.org/bentaylor/editorials/immutable-data/#:~:text=Immutable%20data%20is%20not%20a,only%20be%20set%20through%20construct

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

- Explanation: Null, different from an empty character or zero or any other thing, can also be interpreted as something unknown, something not available, or something does not fit in the attribute.
- Examples:

- Say we have a column for height and Jack's height is unknown. Instead of putting 0 in the table, we put Null. In this way, our calculation for the average height will not be affected by the unknown value of Jack's height.
- Similarily, say we have a column for address, and some people simply do not provide their addresses. We put null for these cases.

W4

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

Answer

Output from one operation can be the input of another operation, so expressions in relational model and SQL are allowed to be nested.

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.
```
CREATE VIEW no_update1 AS
 SELECT
     first_name,
     last_name
 FROM midterm_students;

CREATE VIEW no_update2 AS
    SELECT
        social_security_no,
        first_name,
        last_name
```

```
FROM midterm_students
GROUP BY social_security_no;
```

1.
```
CREATE VIEW midterm_info AS
SELECT social_security_no,
        first_name,
        last_name
FROM midterm_students;
```

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

- This section_key can be further divided into smaller key values. For example, 2021 is the year, 3 is the term, COMS is the department, 4111 is the course number, and W002 is the section number.
- Given the section key a) We are unable to access all courses of the department; b)We are unable to find all sections of the same class.

W7

Briefly explain the differences between:

- Database stored procedure
- Database function
- Database trigger

Answer

- Database stored procedure is a set of pre-compiled statements that can be explicitly called by the user.
- Database function is a set of statements that accept only input parameters and output in a single value form or a tabular form.
- Database trigger is a stored procedure that runs automatically whenever any special event occurs, such as insert, delete, update etc.

https://www.c-sharpcorner.com/blogs/about-store-proc-function-trigger-in-brif

W8

Briefly explain:

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

Answer

- Natural join returns the rows based on the common columns.
- Equi-join is a join based on equality. It only returns rows where the values matched the equality conditions.
- Theta join is a join based on a relationship other than equality. It returns the rows where the values matched the condition.
- Self-join is a join where the table joins with itself.

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. For example, orderdetails is a weak entity type.
2. ON DELETE CASCADE deletes the child table automatically when the rows from parent tables are deleted. Using ON DELETE CASCADE on weak entity types help avoid the problem of the "foreign key" deleted and the entity becomes dangling.

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 like a pointer to a specific row in the query result. It enables users to define and perform operations on a set of data rows.
2. Using a cursor can be helpful if we want to perform actions on individual rows. Users can access a row specifically, avoiding getting through the whole database.

https://www.ibm.com/docs/en/informix-servers/12.10?topic=sql-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.

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

Answer

```
CREATE TABLE bank_account
(
    branch_id TEXT NOT NULL PRIMARY KEY,
    account_id TEXT NOT NULL PRIMARY KEY,
    balance TEXT NULL
);
```

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:

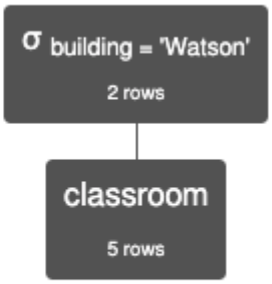
```
(ID, name)
```

- 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

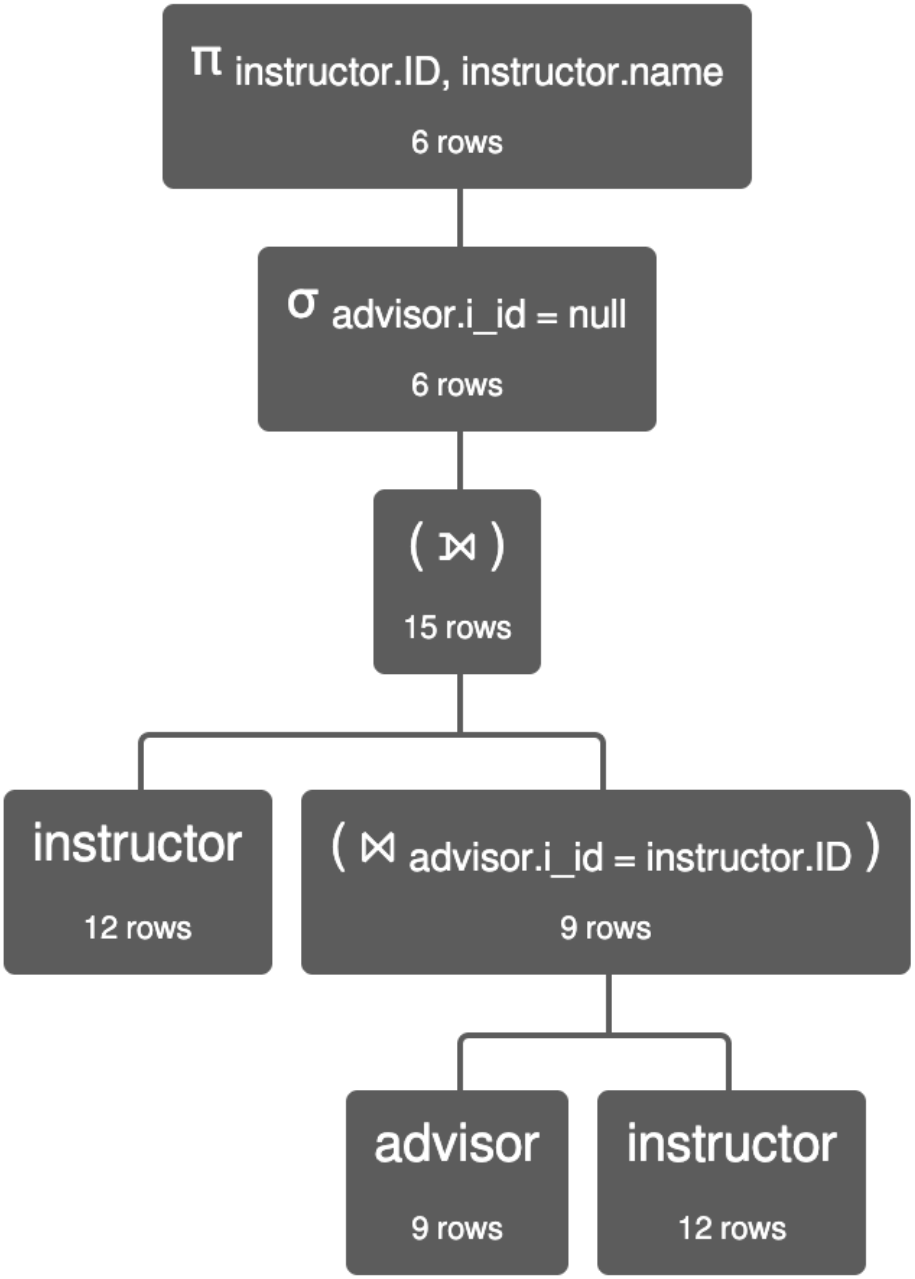
Answer

Query:

```
π instructor.ID, instructor.name
σ advisor.i_id = null
(instructor ⟕
(advisor ⋈ advisor.i_id = instructor.ID instructor)
)
```

In [13]:
```
Image(filename='R2_1.png')
```

Out[13]:



In [14]:
```
Image(filename='R2_2.png')
```

Out[14]:

$$\pi_{\text{instructor.ID, instructor.name}} \sigma_{\text{advisor.i\_id = null}} (\text{instructor} \ltimes (\text{advisor} \bowtie_{\text{advisor.i\_id = instructor.ID}} \text{instructor}))$$

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, ⋈` 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

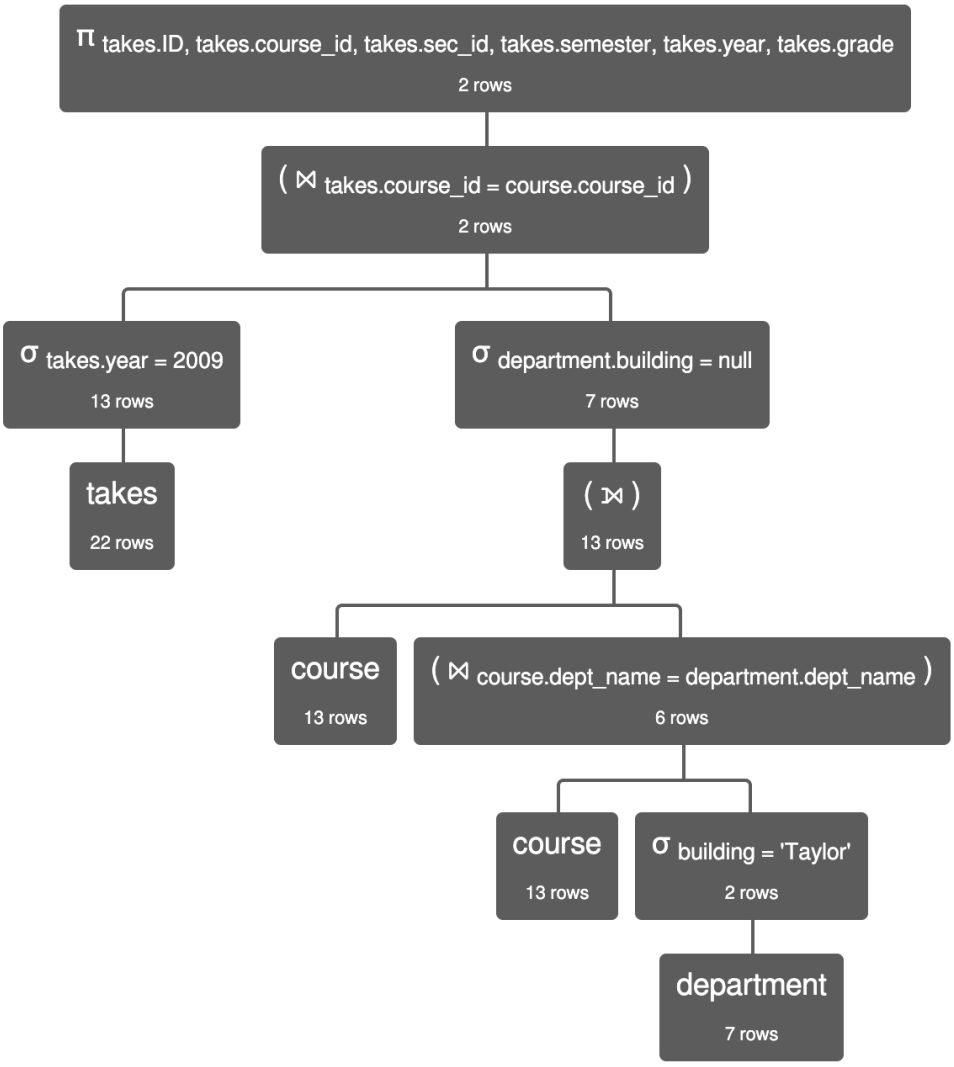
Query:

```
π
takes.ID,takes.course_id,takes.sec_id,takes.semester,takes.year,takes.gra
```

```
 (
σ takes.year = 2009
takes ⋈ takes.course_id = course.course_id
σ department.building = null
(course ⟗ (
course ⋈ course.dept_name = department.dept_name
σ building = 'Taylor' department))
)
```

In [15]:
```
Image(filename='R3_1.png')
```

Out[15]:



In [16]:
```
Image(filename='R3_2.png')
```

Out[16]:

$$\pi \text{ takes.ID, takes.course\_id, takes.sec\_id, takes.semester, takes.year, takes.grade } (\sigma \text{ takes.year = 2009 takes} \bowtie \text{takes.course\_id = course.course\_id } \sigma \text{ department.building = null } (\text{course} \rtimes (\text{course} \bowtie \text{course.dept\_name = department.dept\_name } \sigma \text{ building = 'Taylor' department })))$$

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

Entity Relationship Model

In [ ]:

- 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.

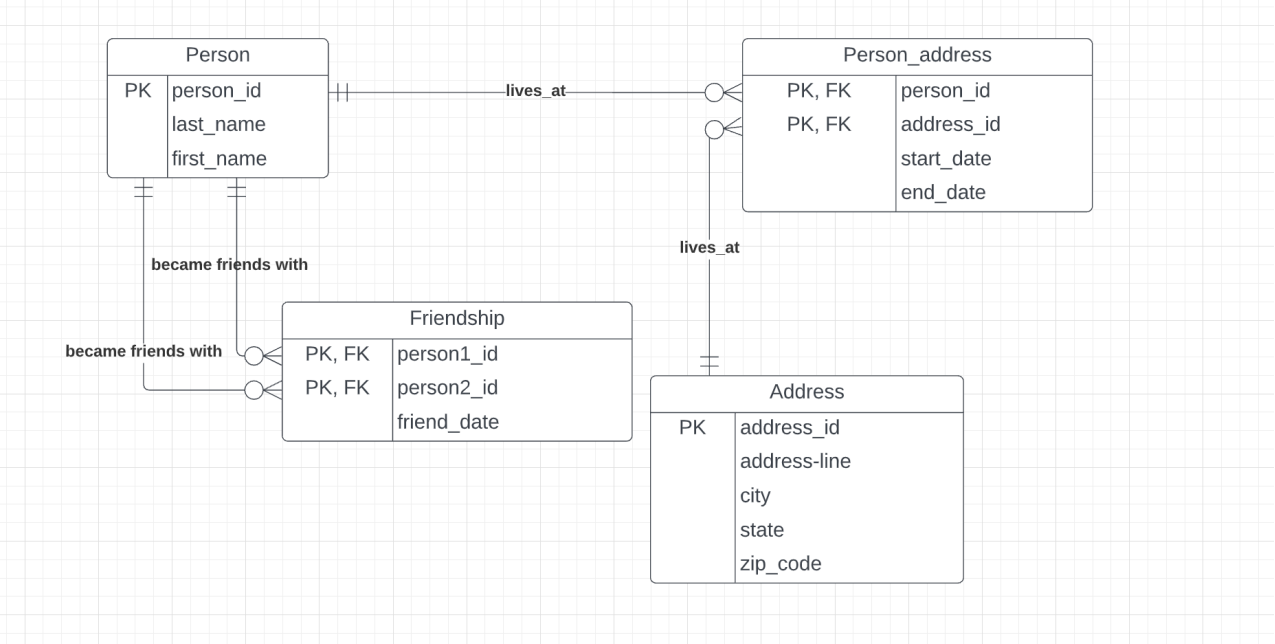
- **Notes:**

- 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

In [17]:
```
Image(filename='ER1.png')
```

Out[17]:



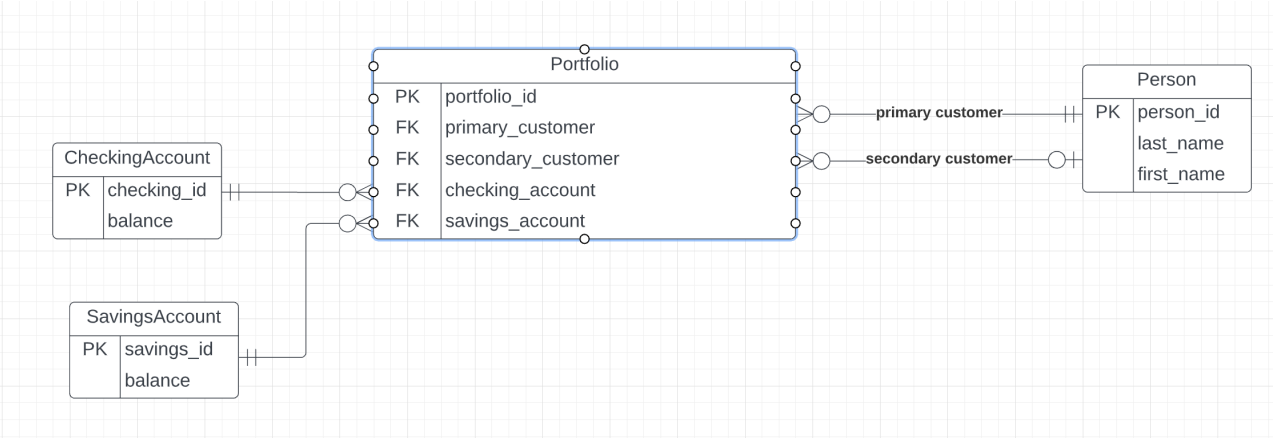
ER2

- Description:
  - There are three entity types:
    1. $$CheckingAccount(\underline{checking\_id}, balance) \quad (4)$$
    2. $$SavingsAccount(\underline{savings\_id}, balance) \quad (5)$$
    3. $$Person(\underline{person\_id}, last\_name, first\_name) \quad (6)$$
  - 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.

In [18]:
```
Image(filename='ER2.png')
```

Out[18]:



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

I created each table on their own. The tables that contain PK as FKs for other tables were created first. For the tables in the diagram that have two PK values (i.e. SchoolDepartment, Student Section), I created two separate tables to better identify the relationship.

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 [19]:
```
# DDL in cells below.
# Use you UNI for the schema.
%sql drop schema lh3119_s22_midterm;
%sql create schema if not exists lh3119_s22_midterm;
%sql select 1;
```

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

Out[19]:

1
1

In [20]:
```
# SQL tests.
# Test your DDL with sample inserts/updates showing that you correctly implement
#
```

In [21]:
```
%%sql
USE lh3119_s22_midterm;
CREATE TABLE Student
(
    UNI        varchar(16)  NOT NULL,
    last_name  varchar(32)  NOT NULL,
    first_name varchar(32)  NOT NULL,
    email      varchar(320) NULL,
    CONSTRAINT Student_pk PRIMARY KEY (UNI),
    CHECK (email REGEXP '.*@.*.edu$')
);

CREATE UNIQUE INDEX Student_UNI_uindex ON Student (UNI);
CREATE UNIQUE INDEX Student_email_uindex ON Student (email);
```

```
 * mysql+pymysql://root:***@localhost
```

```
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
```

Out[21]:
```
[]
```

In [22]:
```
%%sql
CREATE TABLE Course
(
    course_no   int          NOT NULL,
    title       varchar(64)  NOT NULL,
    description varchar(255) DEFAULT 'n/a',
    CONSTRAINT Course_pk PRIMARY KEY (course_no)
);


CREATE TABLE School
(
    school_code   varchar(64)  NOT NULL,
    school_name   varchar(64)  NOT NULL,
    CONSTRAINT School_pk PRIMARY KEY (school_code)
);

CREATE TABLE Department
(
    department_code   varchar(64)  NOT NULL,
    department_name   varchar(64)  NOT NULL,
    CONSTRAINT Department_pk PRIMARY KEY (department_code)
);
```

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

Out[22]:
```
[]
```

In [23]:
```
%%sql

CREATE TABLE Faculty
(
    UNI             varchar(64)  NOT NULL,
    last_name       varchar(64)  NOT NULL,
    first_name      varchar(64)  NOT NULL,
    department_code varchar(64)  NOT NULL,
    email           varchar(320) NULL,
    CHECK (email REGEXP '.*@.*.edu$'),
    CONSTRAINT Faculty_pk PRIMARY KEY (UNI),
    CONSTRAINT Faculty_fk FOREIGN KEY (department_code) REFERENCES Department (d
);

CREATE UNIQUE INDEX Faculty_email_uindex ON Faculty (email);
```

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

Out[23]:
```
[]
```

In [24]:
```
%%sql
CREATE TABLE Section
(
    call_no       varchar(255) NOT NULL,
    course_no     int NOT NULL,
    section_no    int NOT NULL,
    semester      varchar(32) NOT NULL,
    year          int NOT NULL,
    CHECK (year BETWEEN 2021 AND 2023),
    instructor_id varchar(64) NOT NULL,
    CONSTRAINT Section_ck CHECK (semester IN ('fall', 'summer', 'spring')),
    CONSTRAINT Section_pk PRIMARY KEY (call_no),
    CONSTRAINT Section_Course_course_no_fk FOREIGN KEY (course_no) REFERENCES Co
    CONSTRAINT Section_Faculty_UNI_fk FOREIGN KEY (instructor_id) REFERENCES Fac
    CONSTRAINT Section_uindex UNIQUE (course_no , section_no, semester, year)
);
```

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

Out[24]:
```
[]
```

In [25]:
```
%%sql
CREATE TABLE Student_section
(
    UNI           varchar(64)  NOT NULL,
    call_no       varchar(64) NOT NULL,
    role          varchar(64) NULL,
    CONSTRAINT Stusection_pk PRIMARY KEY (UNI),
    CONSTRAINT Stusection_fk1 FOREIGN KEY (UNI) REFERENCES Student (UNI),
    CONSTRAINT Stusection_fk2 FOREIGN KEY (call_no) REFERENCES Section (call_no)
);

CREATE TABLE Section_student
(
    UNI           varchar(64) NOT NULL,
    call_no       varchar(64) NOT NULL,
    role          varchar(64) NULL,
    CONSTRAINT Secstudent_pk PRIMARY KEY (call_no),
    CONSTRAINT Secstudent_fk1 FOREIGN KEY (UNI) REFERENCES Student (UNI),
    CONSTRAINT Secstudent_fk2 FOREIGN KEY (call_no) REFERENCES Section (call_no)
);
```

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

Out[25]:
```
[]
```

In [26]:
```
%%sql
CREATE TABLE School_department
(
    school_code varchar(64) NOT NULL,
    department_code varchar(64) NOT NULL,
    CONSTRAINT Schooldepart_pk PRIMARY KEY (school_code),
    CONSTRAINT Schooldepart_fk1 FOREIGN KEY (school_code) REFERENCES School (sch
    CONSTRAINT Schooldepart_fk2 FOREIGN KEY (department_code) REFERENCES Departm
);
```

```
CREATE TABLE Department_school
(
    school_code varchar(64) NOT NULL,
    department_code varchar(64) NOT NULL,
    CONSTRAINT Schooldepart_pk PRIMARY KEY (department_code),
    CONSTRAINT DepartSchool_fk1 FOREIGN KEY (school_code) REFERENCES School (sch
    CONSTRAINT DepartSchool_fk2 FOREIGN KEY (department_code) REFERENCES Departm
);
```

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

Out[26]:
```
[]
```

In [27]:
```
%%sql

INSERT INTO Student VALUES ('lh3119', 'Huang', 'Liting', 'lh3119@columbia.edu');

INSERT INTO Course VALUES (4111, 'Database', 'introduction to database');

INSERT INTO School VALUES('CC', 'Columbia College');

INSERT INTO Department VALUES('COMS', 'Computer Science');

INSERT INTO Faculty VALUES ('dff9', 'Ferguson', 'Donald', 'COMS', 'dff9@columbia

INSERT INTO Section VALUES ('11111', 4111, 001, 'spring', 2022, 'dff9');

INSERT INTO Student_section VALUES ('lh3119', '11111', NULL);
INSERT INTO Section_student VALUES ('lh3119', '11111', NULL);

INSERT INTO School_department VALUES ('CC', 'COMS');
INSERT INTO Department_school VALUES ('CC', 'COMS');
```

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

Out[27]:
```
[]
```

In [28]:
```
try:
    %sql INSERT INTO Student VALUES ('ab223', 'abby', 'bill', 'abbill')
    print("Email constraint")
except Exception as e:
  print(e)
```

```
 * mysql+pymysql://root:***@localhost
(pymysql.err.OperationalError) (3819, "Check constraint 'student_chk_1' is viola
ted.")
[SQL: INSERT INTO Student VALUES ('ab223', 'abby', 'bill', 'abbill')]
```

```
(Background on this error at: https://sqlalche.me/e/14/e3q8)
Email constraint
```

In [29]:

```
try:
    %sql USE lh3119_s22_midterm
    %sql INSERT INTO Student VALUES ('lh3119', 'hiii', 'liting', 'lh3119@columbi
    print("student unique uni constraint")
except Exception as e:
  print(e)
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
 * mysql+pymysql://root:***@localhost
(pymysql.err.IntegrityError) (1062, "Duplicate entry 'lh3119' for key 'student.P
RIMARY'")
[SQL: INSERT INTO Student VALUES ('lh3119', 'hiii', 'liting', 'lh3119@columbia.e
du')]
(Background on this error at: https://sqlalche.me/e/14/gkpj)
```

In [30]:

```
try:
    %sql USE lh3119_s22_midterm
    %sql INSERT INTO Student VALUES ('lh3220', 'hiii', 'liting', 'lh3119@columbi
    print("student unique email constraint")
except Exception as e:
  print(e)
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
 * mysql+pymysql://root:***@localhost
(pymysql.err.IntegrityError) (1062, "Duplicate entry 'lh3119@columbia.edu' for k
ey 'student.Student_email_uindex'")
[SQL: INSERT INTO Student VALUES ('lh3220', 'hiii', 'liting', 'lh3119@columbia.e
du')]
(Background on this error at: https://sqlalche.me/e/14/gkpj)
```

In [31]:

```
try:
    %sql USE lh3119_s22_midterm
    %sql INSERT INTO Faculty VALUES ('dfff10', 'Ferguson', 'Donald', 'COMS', 'df
    print("faculty unique email constraint")
except Exception as e:
  print(e)
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
 * mysql+pymysql://root:***@localhost
(pymysql.err.IntegrityError) (1062, "Duplicate entry 'dff9@columbia.edu' for key
'faculty.Faculty_email_uindex'")
[SQL: INSERT INTO Faculty VALUES ('dfff10', 'Ferguson', 'Donald', 'COMS', 'dff9@
columbia.edu')]
(Background on this error at: https://sqlalche.me/e/14/gkpj)
```

In [32]:

```
try:
    %sql USE lh3119_s22_midterm
    %sql INSERT INTO Section VALUES ('1200', 4332, 001, 'spring', 2022, 'dff9');
    print("Section FK constraint call_no failed")
except Exception as e:
  print(e)
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
 * mysql+pymysql://root:***@localhost
(pymysql.err.IntegrityError) (1452, 'Cannot add or update a child row: a foreign
key constraint fails (`lh3119_s22_midterm`.`section`, CONSTRAINT `Section_Course
_course_no_fk` FOREIGN KEY (`course_no`) REFERENCES `course` (`course_no`))')
[SQL: INSERT INTO Section VALUES ('1200', 4332, 001, 'spring', 2022, 'dff9');]
(Background on this error at: https://sqlalche.me/e/14/gkpj)
```

In [33]:

```
try:
    %sql USE lh3119_s22_midterm
    %sql INSERT INTO Section VALUES ('11111', 4332, 001, 'spring', 2022, 'dff9')
    print("Section UQ constraint failed")
except Exception as e:
  print(e)
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
 * mysql+pymysql://root:***@localhost
(pymysql.err.IntegrityError) (1062, "Duplicate entry '11111' for key 'section.PR
IMARY'")
[SQL: INSERT INTO Section VALUES ('11111', 4332, 001, 'spring', 2022, 'dff9');]
(Background on this error at: https://sqlalche.me/e/14/gkpj)
```

In [34]:

```
try:
    %sql USE lh3119_s22_midterm
    %sql INSERT INTO Section VALUES ('1300', 4111, 003, 'spring', 2022, 'dff10')
    print("Section FK instructor_id constraint")
except Exception as e:
  print(e)
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
 * mysql+pymysql://root:***@localhost
(pymysql.err.IntegrityError) (1452, 'Cannot add or update a child row: a foreign
key constraint fails (`lh3119_s22_midterm`.`section`, CONSTRAINT `Section_Facult
y_UNI_fk` FOREIGN KEY (`instructor_id`) REFERENCES `faculty` (`UNI`))')
[SQL: INSERT INTO Section VALUES ('1300', 4111, 003, 'spring', 2022, 'dff10');]
(Background on this error at: https://sqlalche.me/e/14/gkpj)
```

In [35]:

```
try:
    %sql USE lh3119_s22_midterm
    %sql INSERT INTO Faculty VALUES ('dff20', 'Ferguson', 'Donald', 'COMSM', 'df
    print("Faculty FK department_code constraint")
except Exception as e:
  print(e)
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
 * mysql+pymysql://root:***@localhost
(pymysql.err.IntegrityError) (1452, 'Cannot add or update a child row: a foreign
key constraint fails (`lh3119_s22_midterm`.`faculty`, CONSTRAINT `Faculty_fk` FO
REIGN KEY (`department_code`) REFERENCES `department` (`department_code`))')
[SQL: INSERT INTO Faculty VALUES ('dff20', 'Ferguson', 'Donald', 'COMSM', 'dff20
@columbia.edu');]
(Background on this error at: https://sqlalche.me/e/14/gkpj)
```

In [36]:

```
try:
    %sql USE lh3119_s22_midterm
    %sql INSERT INTO Student_section VALUES ('lh3220', '2121', NULL);
    print("Student_section FK constraint")
except Exception as e:
  print(e)
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
 * mysql+pymysql://root:***@localhost
(pymysql.err.IntegrityError) (1452, 'Cannot add or update a child row: a foreign
key constraint fails (`lh3119_s22_midterm`.`student_section`, CONSTRAINT `Stusec
tion_fk1` FOREIGN KEY (`UNI`) REFERENCES `student` (`UNI`))')
[SQL: INSERT INTO Student_section VALUES ('lh3220', '2121', NULL);]
(Background on this error at: https://sqlalche.me/e/14/gkpj)
```

In [37]:

```
try:
    %sql USE lh3119_s22_midterm
    %sql INSERT INTO Section_student VALUES ('lh3117', '212121', NULL);
    print("Section_student FK constraint")
except Exception as e:
  print(e)
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
 * mysql+pymysql://root:***@localhost
(pymysql.err.IntegrityError) (1452, 'Cannot add or update a child row: a foreign
key constraint fails (`lh3119_s22_midterm`.`section_student`, CONSTRAINT `Secstu
dent_fk2` FOREIGN KEY (`call_no`) REFERENCES `section` (`call_no`))')
[SQL: INSERT INTO Section_student VALUES ('lh3117', '212121', NULL);]
(Background on this error at: https://sqlalche.me/e/14/gkpj)
```

In [38]:

```
try:
    %sql USE lh3119_s22_midterm
    %sql INSERT INTO School_department VALUES ('SEAS', 'COMS');
    print("School_department FK constraint")
except Exception as e:
  print(e)
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
 * mysql+pymysql://root:***@localhost
(pymysql.err.IntegrityError) (1452, 'Cannot add or update a child row: a foreign
key constraint fails (`lh3119_s22_midterm`.`school_department`, CONSTRAINT `Scho
oldepart_fk1` FOREIGN KEY (`school_code`) REFERENCES `school` (`school_code`))')
[SQL: INSERT INTO School_department VALUES ('SEAS', 'COMS');]
(Background on this error at: https://sqlalche.me/e/14/gkpj)
```

In [39]:

```
try:
    %sql USE lh3119_s22_midterm
    %sql INSERT INTO Department_school VALUES ('CC', 'COMSA');
    print("Department_school FK constraint")
except Exception as e:
  print(e)
```

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

```
(pymysql.err.IntegrityError) (1452, 'Cannot add or update a child row: a foreign
key constraint fails (`lh3119_s22_midterm`.`department_school`, CONSTRAINT `Depa
rtSchool_fk2` FOREIGN KEY (`department_code`) REFERENCES `department` (`departme
nt_code`))')
[SQL: INSERT INTO Department_school VALUES ('CC', 'COMSA');]
(Background on this error at: https://sqlalche.me/e/14/gkpj)
```

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 [40]:
```
%sql select * from classicmodels;
```

```
 * mysql+pymysql://root:***@localhost
(pymysql.err.ProgrammingError) (1146, "Table 'lh3119_s22_midterm.classicmodels'
doesn't exist")
[SQL: select * from classicmodels;]
(Background on this error at: https://sqlalche.me/e/14/f405)
```

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

In [41]:
```
%%sql

USE classicmodels;

WITH product_info AS
(
    SELECT
        productLine,
        productName,
        productVendor,
        productCode
    FROM products
),

product_val AS
```

```
(
    SELECT
        productCode,
        (quantityOrdered * priceEach) AS order_value
    FROM orderDetails
),
total_revenue AS
(
    SELECT
        productCode,
        sum(order_value) AS total_value
    FROM product_val
    GROUP BY productCode
),
product_summary AS
(
    SELECT
        *
    FROM product_info
    INNER JOIN
        total_revenue
    USING (productCode)
)

SELECT
    productLine,
    productName,
    productVendor,
    total_value
FROM product_summary
WHERE
    total_value >= 15000
ORDER BY total_value DESC
LIMIT 6;
```

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

Out[41]:

productLine	productName	productVendor	total_value
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 [42]:

```
%%sql
```

```
USE classicmodels;

SELECT
    *
FROM products
WHERE
    productCode NOT IN
    (
    SELECT
         productCode
    FROM orderDetails
    );
```

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

Out[42]:

productCode	productName	productLine	productScale	productVendor	productDescription	quan
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 [43]:

```
%sql select * from classicmodels.midterm_s3;
```

```
 * mysql+pymysql://root:***@localhost
(pymysql.err.ProgrammingError) (1146, "Table 'classicmodels.midterm_s3' doesn't
exist")
[SQL: select * from classicmodels.midterm_s3;]
(Background on this error at: https://sqlalche.me/e/14/f405)
```

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

In [44]:
```
%%sql

USE classicmodels;

WITH customers AS
(
    SELECT
        *
    FROM customers
    WHERE
    country = 'USA'
    OR country = 'Canada'
),

orders AS
(
    SELECT
        customerNumber,
        orderNumber,
        status,
        DATEDIFF(shippedDate,orderDate) AS order_time
    FROM orders
    WHERE
        status!='Cancelled'
)

SELECT
    customerNumber,
    customerName,
    orderNumber,
    order_time
FROM customers
INNER JOIN
    orders
    USING (customerNumber)
    WHERE
        status != 'Shipped'
        AND status != 'Resolved'
        OR order_time >= 5

    ORDER BY
        order_time IS NULL DESC,
        order_time DESC,
        orderNumber ASC;
```

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

Out[44]:

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

customerNumber	customerName	orderNumber	order_time
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

customerNumber	customerName	orderNumber	order_time
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

S4

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

In [45]:
```
%sql select productCode, productScale from classicmodels.products limit 5;
```

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

Out[45]:

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 [46]:
```
%%sql
select * from classicmodels.midterm_s4;
```

```
 * mysql+pymysql://root:***@localhost
(pymysql.err.ProgrammingError) (1146, "Table 'classicmodels.midterm_s4' doesn't
exist")
[SQL: select * from classicmodels.midterm_s4;]
(Background on this error at: https://sqlalche.me/e/14/f405)
```

In [47]:
```
%%sql

WITH unmatched AS
(
    SELECT
        SUBSTRING(productScale, 3, 4) AS scale_in_product_scale,
        SUBSTRING((SUBSTRING_INDEX(productCode, '_', 1)), 2, 3) AS scale_in_prod
        productCode,
        productScale
    FROM
        classicmodels.products
)
```

```
SELECT
    *
FROM unmatched
WHERE
    scale_in_product_scale != scale_in_product_Number;
```

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

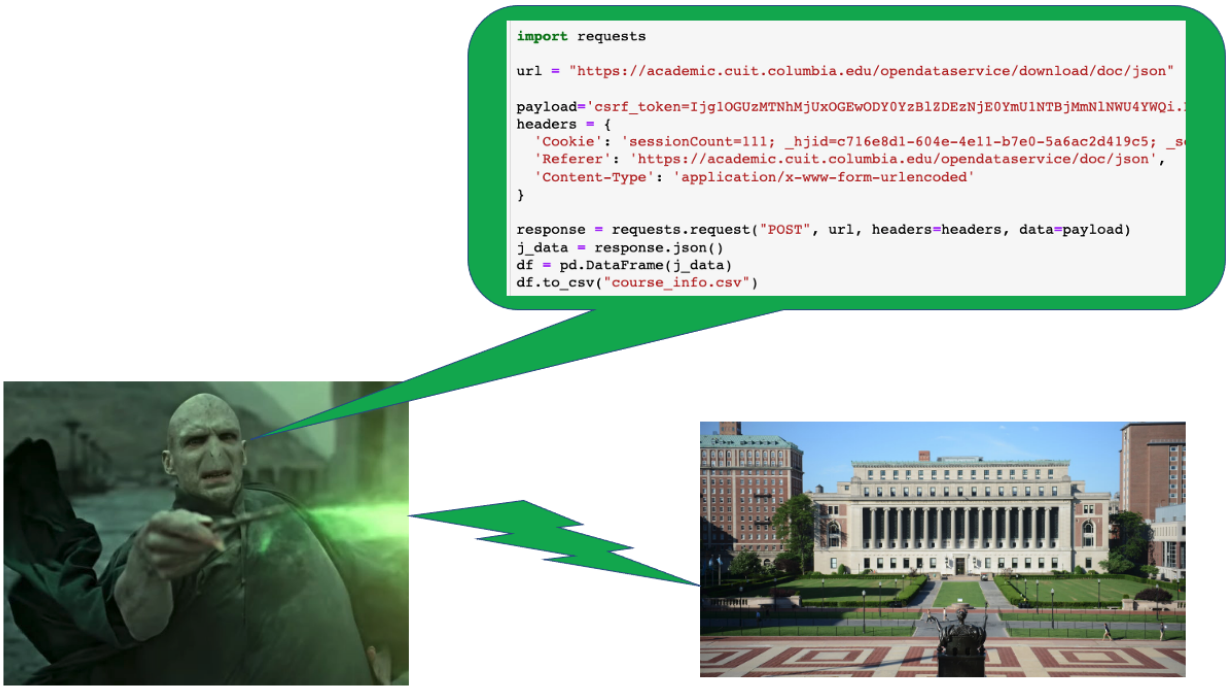
Out[47]:

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 [48]:

```
df = pd.read_csv("course_info.csv")
```

```
df
```

Out[48]:

	Unnamed: 0	ExamDate	Term	ChargeAmt1	Meets4	TypeCode	SubtermCode	CampusCo
0	0	NaN	20212	NaN	NaN	LC	NaN	MOF
1	1	NaN	20212	NaN	NaN	LC	NaN	MOF
2	2	NaN	20212	NaN	NaN	LC	NaN	MOF
3	3	NaN	20212	NaN	NaN	LC	NaN	MOF
4	4	NaN	20212	NaN	NaN	LC	NaN	MOF
...	...	...	...	...	...	...	...	
21430	21430	NaN	20222	NaN	NaN	RG	L	MOF
21431	21431	NaN	20222	NaN	NaN	RG	K	MOF
21432	21432	NaN	20222	NaN	NaN	RG	L	MOF
21433	21433	NaN	20222	100.00	NaN	LC	K	MOF
21434	21434	NaN	20222	100.00	NaN	LC	L	MOF

21435 rows × 45 columns

- The list of columns in the data is:

In [49]:

```
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 [50]:
```
%%sql
drop schema dff9_s22_midterm;
CREATE SCHEMA if not exists dff9_s22_midterm;
```

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

Out[50]:
```
[]
```

In [51]:
```
%%sql
USE dff9_s22_midterm;
CREATE TABLE student
(
    first_name  varchar(32)  NOT NULL,
    middle_name varchar(32)  NULL,
    last_name   varchar(32)  NOT NULL,
    email       varchar(255) NOT NULL,
    enrollment_year YEAR      NOT NULL,
    uni         varchar(16)  NOT NULL,
    CHECK (enrollment_year BETWEEN '2016' AND '2022'),
    GUID char(64) NOT NULL,
    CONSTRAINT students_pk1 PRIMARY KEY (uni)
);

GRANT DELETE, INSERT, UPDATE ON TABLE student TO root@localhost;
```

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

Out[51]:
```
[]
```

In [52]:
```
%%sql
CREATE TABLE employee
(
    first_name  varchar(32)  NOT NULL,
    middle_name varchar(32)  NULL,
    last_name   varchar(32)  NOT NULL,
    email       varchar(255) NOT NULL,
    employee_type varchar(32) NOT NULL,
    uni         varchar(16)  NOT NULL,
    GUID        char(64) NOT NULL,

    CONSTRAINT employees_pk PRIMARY KEY (uni),
    CONSTRAINT employees_ck CHECK (employee_type IN ('Professor', 'Lecturer', 'S
);

GRANT DELETE, INSERT, UPDATE ON TABLE employee TO root@localhost;
```

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

Out[52]:
```
[]
```

In [53]:
```
%%sql

CREATE VIEW people(uni, guid, last_name, first_name, middle_name, email, employe
SELECT uni, guid, last_name, first_name, middle_name, email, NULL, enrollment_ye
UNION ALL
SELECT uni, guid, last_name, first_name, middle_name, email, employee_type, NULL
```

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

Out[53]:
```
[]
```

In [54]:
```
%%sql
CREATE FUNCTION compute_uni(first_name  varchar(32), middle_name varchar(32), la
RETURNS varchar(16) DETERMINISTIC

BEGIN
    DECLARE result_uni varchar(16);
    DECLARE f_initial, m_initial, l_initial char(1);
    DECLARE uni_match_count int;
    DECLARE uni_prefix char(4);

    SET f_initial = lower(substr(first_name, 1, 1));
    SET l_initial = lower(substr(last_name, 1, 1));

    IF middle_name IS NULL THEN
        SET uni_prefix = concat(f_initial, l_initial, "%");
        SELECT count(*) INTO uni_match_count FROM people WHERE uni LIKE uni_pref
        SET uni_match_count = uni_match_count + 1;
        SET result_uni = concat(f_initial, l_initial, uni_match_count);


    ELSE
        SET m_initial = lower(substr(middle_name, 1, 1));
        SET uni_prefix = concat(f_initial, m_initial, l_initial, "%");
        SELECT count(*) INTO uni_match_count FROM people WHERE uni LIKE uni_pref
        SET uni_match_count = uni_match_count + 1;
        SET result_uni = concat(f_initial, m_initial, l_initial, uni_match_count
    END IF;
RETURN result_uni;
END
```

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

Out[54]:
```
[]
```

In [55]:
```
%%sql
CREATE TRIGGER student_before_insert BEFORE INSERT ON student FOR EACH ROW
BEGIN
    IF new.email IN (SELECT email FROM people) THEN
        SIGNAL SQLSTATE '45000';
    END IF;
    SET new.uni = compute_uni(new.first_name, new.middle_name, new.last_name);
    SET new.GUID = uuid();
END
```

```
 * mysql+pymysql://root:***@localhost
```

```
0 rows affected.
```

Out[55]:
```
[]
```

In [56]:
```
%%sql
CREATE TRIGGER employee_before_insert BEFORE INSERT ON employee FOR EACH ROW
BEGIN
    IF new.email IN (SELECT email FROM people) THEN
        SIGNAL SQLSTATE '45000';
    END IF;
    SET new.uni = compute_uni(new.first_name, new.middle_name, new.last_name);
    SET new.GUID = uuid();
END
```

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

Out[56]:
```
[]
```

In [57]:
```
%%sql
CREATE TRIGGER update_student BEFORE UPDATE ON student FOR each row
BEGIN
    IF new.email IN (SELECT email FROM people) THEN
        SIGNAL SQLSTATE '45000';
    END IF;

    IF new.uni != old.uni THEN
        SET new.uni = old.uni;
    END IF;
END;
```

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

Out[57]:
```
[]
```

In [58]:
```
%%sql
CREATE TRIGGER update_employee BEFORE UPDATE ON employee FOR each row
BEGIN
    IF new.email IN (SELECT email FROM people) THEN
        SIGNAL SQLSTATE '45000';
    END IF;

    IF new.uni != old.uni THEN
        SET new.uni = old.uni;
    END IF;
END;
```

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

Out[58]:
```
[]
```

In [59]:
```
%%sql

CREATE PROCEDURE create_student (IN first_name varchar(32),
                                 IN middle_name varchar(32),
                                 IN last_name varchar(32),
```

```
                                    IN email varchar(255),
                                    IN enrollment_year YEAR,
                                   OUT uni varchar(16),
                                   OUT GUID char(64))

BEGIN
    INSERT INTO student
    VALUES (first_name, middle_name, last_name, email, enrollment_year, NULL, NU

    SELECT uni INTO uni FROM student WHERE student.email = email;
    SELECT GUID INTO GUID FROM student WHERE student.email = email;
END
```

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

Out[59]:
```
[]
```

In [60]:
```
%%sql

CREATE PROCEDURE create_employee (IN first_name varchar(32),
                                  IN middle_name varchar(32),
                                  IN last_name varchar(32),
                                  IN email varchar(255),
                                  IN employee_type varchar(32),
                                 OUT uni varchar(16),
                                 OUT GUID char(64))

BEGIN
    INSERT INTO employee
    VALUES (first_name, middle_name, last_name, email, employee_type, NULL, NULL

    SELECT uni INTO uni FROM employee WHERE employee.email = email;
    SELECT GUID INTO GUID FROM employee WHERE employee.email = email;
END
```

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

Out[60]:
```
[]
```

In [61]:
```
%%sql


CREATE PROCEDURE update_student (IN _uni varchar(16),
                                 IN _first_name varchar(32),
                                 IN _middle_name varchar(32),
                                 IN _last_name varchar(32),
                                 IN _email varchar(255),
                                 IN _enrollment_year YEAR)

BEGIN
    IF _uni IS NOT NULL THEN

        UPDATE student
            SET first_name = IFNULL(_first_name, first_name),
                middle_name= IFNULL(_middle_name, middle_name),
                last_name = IFNULL(_last_name, last_name),
                email = IFNULL(_email, email),
```

```
                enrollment_year = IFNULL(_enrollment_year, enrollment_year)
        WHERE uni=_uni;
    END IF;

END
```

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

Out[61]:
```
[]
```

In [62]:
```
%%sql


CREATE PROCEDURE update_employee (IN _uni varchar(16),
                                  IN _first_name varchar(32),
                                  IN _middle_name varchar(32),
                                  IN _last_name varchar(32),
                                  IN _email varchar(255),
                                  IN _employee_type varchar(32))

BEGIN
    IF _uni IS NOT NULL THEN

        UPDATE employee
            SET first_name = IFNULL(_first_name, first_name),
                middle_name= IFNULL(_middle_name, middle_name),
                last_name = IFNULL(_last_name, last_name),
                email = IFNULL(_email, email),
                employee_type = IFNULL(_employee_type, employee_type)
        WHERE uni=_uni;
    END IF;

END
```

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

Out[62]:
```
[]
```

In [63]:
```
# Put your create table, procedure, functions, etc. definitions in the following
#
```

In [64]:
```
#
```

In [65]:
```
#
```

In [66]:
```
# The following only works if you have already created the tables, procedures, e
# So, you mus have answered the questions to run the load.
#
%sql use dff9_s22_midterm;
%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[66]: **1**

1

In [67]:

```
# 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[67]:

	first_name	middle_name	last_name	email	employee_type	enrollm
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		
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 [68]:

```
people_info[-5:]
```

Out[68]:

```
[{'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 [ ]:

In [69]:

```
# If you have defined your procedures, functions, tables and constraints correct
# to load your data.
#
import copy


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

    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("dff9_s22_midterm.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
        res = cur.execute("""SELECT @_dff9_s22_midterm.create_student_5,
                                @_dff9_s22_midterm.create_student_6""")
        result = cur.fetchall()

    # The following does the same for employee.
    elif p_dict['enrollment_year'] is None:
        res = cur.callproc("dff9_s22_midterm.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 [70]:

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

In [71]:

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

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

Out[71]:

uni	guid	last_name	first_name	middle_name	email	em
aeb1	b678ab16-af0d-11ec-8e23-b2f5b3a00848	Blissitt	Avie	Else	eblissitti@youtu.be	
aha1	b67af088-af0d-11ec-8e23-b2f5b3a00848	Anyene	Abbey	Helge	hanyene2c@newsvine.com	
awq1	b6810ec8-af0d-11ec-8e23-b2f5b3a00848	Quimby	Anabelle	Wallas	wquimby1c@nba.com	

uni	guid	last_name	first_name	middle_name	email	em
bag1	b6753440-af0d-11ec-8e23-b2f5b3a00848	Guesford	Bibbye	Annabal	aguesfordb@tumblr.com	
bb1	b6765ba4-af0d-11ec-8e23-b2f5b3a00848	Baybutt	Base	None	bbaybutty@tmall.com	
bce1	b6779bae-af0d-11ec-8e23-b2f5b3a00848	Elias	Barry	Cullin	celias1k@scribd.com	
bcm1	b677ef0a-af0d-11ec-8e23-b2f5b3a00848	Maclean	Brigida	Cameron	cmaclean13@mac.com	
beg1	b6794792-af0d-11ec-8e23-b2f5b3a00848	Guerola	Bank	Elane	eguerola29@blogger.com	
ceg1	b6792ec4-af0d-11ec-8e23-b2f5b3a00848	Gillespie	Courtnay	Elbert	egillespie2m@slideshare.net	
cfh1	b67991a2-af0d-11ec-8e23-b2f5b3a00848	Hartell	Charline	Ferd	fhartell1@meetup.com	
cv1	b67f2b6c-af0d-11ec-8e23-b2f5b3a00848	Venables	Carroll	None	rvenablesu@friendfeed.com	
dap1	b67621fc-af0d-11ec-8e23-b2f5b3a00848	Pounder	Duffy	Auberon	apounder2h@reuters.com	
db1	b678df64-af0d-11ec-8e23-b2f5b3a00848	Burchett	Dougy	None	eburchettd@chicagotribune.com	
dfr1	b679c262-af0d-11ec-8e23-b2f5b3a00848	Rozenzweig	Dyna	Felic	frozenzweig2i@rambler.ru	
dkc1	b67c36aa-af0d-11ec-8e23-b2f5b3a00848	Cranfield	Dacey	Katuscha	kcranfield1s@nature.com	
dm2	b680206c-af0d-11ec-8e23-b2f5b3a00848	McKinie	Drona	None	smckinie1a@springer.com	

uni	guid	last_name	first_name	middle_name	email	em
elh1	b67cdcb8-af0d-11ec-8e23-b2f5b3a00848	Haddinton	Emili	Lynnet	lhaddintonn@over-blog.com	
erc1	b67eaa2a-af0d-11ec-8e23-b2f5b3a00848	Coghlin	Esra	Richard	rcoghlinv@qq.com	
frg1	b67ecadc-af0d-11ec-8e23-b2f5b3a00848	Goudie	Fredek	Raddie	rgoudie5@blogs.com	
gga1	b679db44-af0d-11ec-8e23-b2f5b3a00848	Adolphine	Granville	Guinevere	gadolphine11@instagram.com	
go1	b680ebe6-af0d-11ec-8e23-b2f5b3a00848	O'Donegan	Gerty	None	vodoneganf@clickbank.net	
il1	b67ba712-af0d-11ec-8e23-b2f5b3a00848	Linacre	Ira	None	jlinacreo@webnode.com	
jbh1	b676f8e8-af0d-11ec-8e23-b2f5b3a00848	Habberjam	Jenine	Berry	bhabberjam2k@examiner.com	
kcf1	b677d524-af0d-11ec-8e23-b2f5b3a00848	Foort	Kerwin	Conrade	cfoortw@vinaora.com	
kg1	b6787e20-af0d-11ec-8e23-b2f5b3a00848	Glanders	Kermit	None	dglanders15@booking.com	
ksg1	b67f6712-af0d-11ec-8e23-b2f5b3a00848	Gino	Kearney	Selie	sgino1l@bluehost.com	
lgb1	b67a2ad6-af0d-11ec-8e23-b2f5b3a00848	Burdas	Leann	Gabriellia	gburdas1i@indiegogo.com	
lmi1	b67dc222-af0d-11ec-8e23-b2f5b3a00848	Ionesco	Lynett	Myrwyn	mionesco2n@moonfruit.com	
lot1	b67e8dd8-af0d-11ec-8e23-b2f5b3a00848	Tomlins	Levey	Ottilie	otomlins27@icq.com	

uni	guid	last_name	first_name	middle_name	email	em
mji1	b67b8c64-af0d-11ec-8e23-b2f5b3a00848	Issett	Mal	Jermaine	jissett2e@freewebs.com	
mrw1	b67f4912-af0d-11ec-8e23-b2f5b3a00848	Willatts	Meridel	Rawley	rwillatts1t@si.edu	
ng2	b68083ae-af0d-11ec-8e23-b2f5b3a00848	Gillanders	Nonna	None	tgillanders9@example.com	
nhr1	b67b0b22-af0d-11ec-8e23-b2f5b3a00848	Reddyhoff	Nickolai	Herby	hreddyhoff1g@vk.com	
nr1	b680a73a-af0d-11ec-8e23-b2f5b3a00848	Rubinchik	Norry	None	trubinchik16@howstuffworks.com	
qt1	b6783884-af0d-11ec-8e23-b2f5b3a00848	Threadgall	Quent	None	cthreadgall1v@booking.com	
rcc1	b677668e-af0d-11ec-8e23-b2f5b3a00848	Caps	Reece	Corbett	ccapsz@telegraph.co.uk	
rv1	b67dfaee-af0d-11ec-8e23-b2f5b3a00848	Vigar	Rozelle	None	mvigar8@vistaprint.com	
sgb1	b679f46c-af0d-11ec-8e23-b2f5b3a00848	Bates	Shea	Gail	gbates1e@alibaba.com	
ss1	b67a9548-af0d-11ec-8e23-b2f5b3a00848	Stidson	Sada	None	gstidsonp@ebay.com	
sss1	b680611c-af0d-11ec-8e23-b2f5b3a00848	Spooner	Sterne	Sasha	sspooner1f@wordpress.org	
tgo1	b67a7b30-af0d-11ec-8e23-b2f5b3a00848	Offield	Tobiah	Gerek	goffield1u@addthis.com	
tmc1	b67d8910-af0d-11ec-8e23-b2f5b3a00848	Colbourne	Travis	Melba	mcolbourne2j@mapquest.com	

uni	guid	last_name	first_name	middle_name	email	em
tp1	b67be13c-af0d-11ec-8e23-b2f5b3a00848	Pretswell	Trix	None	jpretswell1h@xrea.com	
vcs1	b67821dc-af0d-11ec-8e23-b2f5b3a00848	Simeons	Veronika	Cordey	csimeons2@microsoft.com	
vem1	b679791a-af0d-11ec-8e23-b2f5b3a00848	Marousek	Vincenty	Elisabeth	emarousek10@huffingtonpost.com	
vm1	b679a98a-af0d-11ec-8e23-b2f5b3a00848	Maffioni	Vance	None	fmaffioni1r@amazon.com	
wcm1	b67808be-af0d-11ec-8e23-b2f5b3a00848	Moughtin	Woodrow	Camile	cmoughtin17@illinois.edu	
wk1	b6796038-af0d-11ec-8e23-b2f5b3a00848	Kerby	Willi	None	ekerby1p@amazon.com	
ws1	b67bfd3e-af0d-11ec-8e23-b2f5b3a00848	Scrauniage	Winslow	None	jscrauniage28@aol.com	
zf1	b674af16-af0d-11ec-8e23-b2f5b3a00848	Fenelon	Zared	None	afenelona@themeforest.net	
adw1	b6789446-af0d-11ec-8e23-b2f5b3a00848	Warmisham	Ailbert	Danie	dwarmishame@soundcloud.com	
ars1	b67f0646-af0d-11ec-8e23-b2f5b3a00848	Sellek	Ari	Rheta	rsellek6@oakley.com	
brm1	b67ee8fa-af0d-11ec-8e23-b2f5b3a00848	Mabbs	Bamby	Rubetta	rmabbs4@xing.com	
bs1	b67d5080-af0d-11ec-8e23-b2f5b3a00848	Scheffel	Bonny	None	lscheffel7@taobao.com	
bsh1	b67f85b2-af0d-11ec-8e23-b2f5b3a00848	Higgonet	Bettina	Sonya	shiggonet2b@163.com	

uni	guid	last_name	first_name	middle_name	email	em
cal1	b675df9e-af0d-11ec-8e23-b2f5b3a00848	Leask	Cari	Andriana	aleask1n@devhub.com	
cg1	b67cc188-af0d-11ec-8e23-b2f5b3a00848	Guislin	Clim	None	lguislin2o@chicagotribune.com	
cmw1	b67e3554-af0d-11ec-8e23-b2f5b3a00848	Wyrall	Carey	Maudie	mwyrallj@scientificamerican.com	
cs1	b67b25a8-af0d-11ec-8e23-b2f5b3a00848	Siegertsz	Christie	None	hsiegertsz21@instagram.com	
ct1	b67ab32a-af0d-11ec-8e23-b2f5b3a00848	Tolman	Carmine	None	gtolmanr@slideshare.net	
da2	b67b4060-af0d-11ec-8e23-b2f5b3a00848	Aslin	Doyle	None	jaslin24@redcross.org	
dm3	b680c8f0-af0d-11ec-8e23-b2f5b3a00848	Medforth	Doug	None	vmedforth1o@homestead.com	
dmw1	b67e1722-af0d-11ec-8e23-b2f5b3a00848	Wynrahame	Darrin	Mario	mwynrahamem@admin.ch	
dss1	b68040ba-af0d-11ec-8e23-b2f5b3a00848	Sillars	Duncan	Shellie	ssillars2l@unicef.org	
ef1	b674ebd4-af0d-11ec-8e23-b2f5b3a00848	Fidele	Ethelin	None	afidele12@google.ru	
ekm1	b67c6f08-af0d-11ec-8e23-b2f5b3a00848	Morfell	Electra	Krystle	kmorfell2g@istockphoto.com	
fm1	b67e7168-af0d-11ec-8e23-b2f5b3a00848	MacNeely	Francene	None	nmacneely22@cpanel.net	
gb1	b67a10aa-af0d-11ec-8e23-b2f5b3a00848	Blagden	Gisela	None	gblagden1q@buzzfeed.com	

uni	guid	last_name	first_name	middle_name	email	em
gp1	b67d3474-af0d-11ec-8e23-b2f5b3a00848	Purbrick	Genni	None	lpurbrick25@canalblog.com	
hdc1	b67850b2-af0d-11ec-8e23-b2f5b3a00848	Croal	Hobart	Dominic	dcroalx@purevolume.com	
hh1	b67cfb12-af0d-11ec-8e23-b2f5b3a00848	Henstridge	Holli	None	lhenstridgeh@sogou.com	
jnb1	b67e53ea-af0d-11ec-8e23-b2f5b3a00848	Bolles	Jacky	Nydia	nbolles23@ucoz.ru	
jsj1	b67fa3a8-af0d-11ec-8e23-b2f5b3a00848	Johl	Jany	Sherry	sjohlg@soundcloud.com	
kb1	b678c52e-af0d-11ec-8e23-b2f5b3a00848	Bree	Karon	None	ebree1z@creativecommons.org	
ki1	b67da652-af0d-11ec-8e23-b2f5b3a00848	Inkin	Karole	None	minkinc@google.de	
km2	b67fdee0-af0d-11ec-8e23-b2f5b3a00848	Malacrida	Kristin	None	smalacrida1w@economist.com	
kmp1	b67ddee2-af0d-11ec-8e23-b2f5b3a00848	Penzer	Kahaleel	Meg	mpenzer14@dailymail.co.uk	
lbb1	b676a9f6-af0d-11ec-8e23-b2f5b3a00848	Bradnocke	Lemmy	Burr	bbradnockek@nifty.com	
lcf1	b677baa8-af0d-11ec-8e23-b2f5b3a00848	Flaxman	Lu	Cinnamon	cflaxman1b@cdbaby.com	
lef1	b679139e-af0d-11ec-8e23-b2f5b3a00848	Fulk	Lelah	Ellette	efulk1d@discuz.net	
ls1	b67c18c8-af0d-11ec-8e23-b2f5b3a00848	Snodin	Lilllie	None	jsnodin20@princeton.edu	

uni	guid	last_name	first_name	middle_name	email	em
mdf1	b678680e-af0d-11ec-8e23-b2f5b3a00848	Favey	Marylin	Darcy	dfavey2p@mozilla.com	
mec1	b678f9e0-af0d-11ec-8e23-b2f5b3a00848	Cella	Maybelle	Esteban	ecella26@mail.ru	
mk2	b67d1750-af0d-11ec-8e23-b2f5b3a00848	Kleinschmidt	Meghan	None	lkleinschmidtl@squarespace.com	
mks1	b67c8a2e-af0d-11ec-8e23-b2f5b3a00848	Slipper	Matthieu	Kalle	kslipper2f@nytimes.com	
mt1	b67d6c0a-af0d-11ec-8e23-b2f5b3a00848	Tennick	Mendie	None	ltennick3@aboutads.info	
ngf1	b67a45d4-af0d-11ec-8e23-b2f5b3a00848	Form	Niki	Gardiner	gform18@blogger.com	
nt1	b67ca61c-af0d-11ec-8e23-b2f5b3a00848	Trehearn	Nadia	None	ktrehearn19@tinyurl.com	
oh1	b67b731e-af0d-11ec-8e23-b2f5b3a00848	Hedley	Olwen	None	jhedley1m@disqus.com	
pl1	b67fc14e-af0d-11ec-8e23-b2f5b3a00848	Lyles	Pattie	None	slyles1j@amazon.de	
rjc1	b67b596a-af0d-11ec-8e23-b2f5b3a00848	Charte	Robinett	Jami	jcharte1y@merriam-webster.com	
rjp1	b67bc594-af0d-11ec-8e23-b2f5b3a00848	Plessing	Renae	Jaquith	jplessing0@samsung.com	
sab1	b67455e8-af0d-11ec-8e23-b2f5b3a00848	Breckell	Sanders	Arline	abreckell1x@fotki.com	
sbl1	b67728d6-af0d-11ec-8e23-b2f5b3a00848	Lalley	Sibylle	Bearnard	blalley2d@rediff.com	

uni	guid	last_name	first_name	middle_name	email	em
sh1	b67a603c-af0d-11ec-8e23-b2f5b3a00848	Hellyar	Suki	None	ghellyar2a@cornell.edu	
skm1	b67c52de-af0d-11ec-8e23-b2f5b3a00848	McKnish	Sayers	Karon	kmcknishs@reddit.com	
swr1	b68130f6-af0d-11ec-8e23-b2f5b3a00848	Ruffli	Sasha	Win	wruffli2q@wordpress.com	
wy1	b67ad53a-af0d-11ec-8e23-b2f5b3a00848	Yousef	Wells	None	gyousef2r@spotify.com	
xak1	b6757b58-af0d-11ec-8e23-b2f5b3a00848	Kief	Xenia	Ardella	akieft@free.fr	
ysm1	b6800104-af0d-11ec-8e23-b2f5b3a00848	McColley	Yehudi	Sile	smccolleyq@amazon.com	

In [72]:

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

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

Out[72]:

first_name	middle_name	last_name	email	enrollment_year	uni	
Avie	Else	Blissitt	eblissitti@youtu.be	2018	aeb1	b2
Abbey	Helge	Anyene	hanyene2c@newsvine.com	2019	aha1	b2
Anabelle	Wallas	Quimby	wquimby1c@nba.com	2022	awq1	b2
Bibbye	Annabal	Guesford	aguesfordb@tumblr.com	2018	bag1	b2

first_name	middle_name	last_name	email	enrollment_year	uni	
Base	None	Baybutt	bbaybutty@tmall.com	2021	bb1	b2
Barry	Cullin	Elias	celias1k@scribd.com	2018	bce1	b2
Brigida	Cameron	Maclean	cmaclean13@mac.com	2019	bcm1	b2
Bank	Elane	Guerola	eguerola29@blogger.com	2021	beg1	b2
Courtnay	Elbert	Gillespie	egillespie2m@slideshare.net	2020	ceg1	b2
Charline	Ferd	Hartell	fhartell1@meetup.com	2020	cfh1	b2
Carroll	None	Venables	rvenablesu@friendfeed.com	2019	cv1	b2
Duffy	Auberon	Pounder	apounder2h@reuters.com	2017	dap1	b2
Dougy	None	Burchett	eburchettd@chicagotribune.com	2022	db1	b2
Dyna	Felic	Rozenzweig	frozenzweig2i@rambler.ru	2022	dfr1	b2
Dacey	Katuscha	Cranfield	kcranfield1s@nature.com	2020	dkc1	b2
Drona	None	McKinie	smckinie1a@springer.com	2018	dm2	b2
Emili	Lynnet	Haddinton	lhaddintonn@over-blog.com	2020	elh1	b2

first_name	middle_name	last_name	email	enrollment_year	uni	
Esra	Richard	Coghlin	rcoghlinv@qq.com	2019	erc1	b2
Fredek	Raddie	Goudie	rgoudie5@blogs.com	2022	frg1	b2
Granville	Guinevere	Adolphine	gadolphine11@instagram.com	2017	gga1	b2
Gerty	None	O'Donegan	vodoneganf@clickbank.net	2020	go1	b2
Ira	None	Linacre	jlinacreo@webnode.com	2021	il1	b2
Jenine	Berry	Habberjam	bhabberjam2k@examiner.com	2021	jbh1	b2
Kerwin	Conrade	Foort	cfoortw@vinaora.com	2020	kcf1	b2
Kermit	None	Glanders	dglanders15@booking.com	2022	kg1	b2
Kearney	Selie	Gino	sgino1l@bluehost.com	2016	ksg1	b2
Leann	Gabriellia	Burdas	gburdas1i@indiegogo.com	2021	lgb1	b2
Lynett	Myrwyn	Ionesco	mionesco2n@moonfruit.com	2020	lmi1	b2
Levey	Ottilie	Tomlins	otomlins27@icq.com	2020	lot1	b2
Mal	Jermaine	Issett	jissett2e@freewebs.com	2018	mji1	b2

first_name	middle_name	last_name	email	enrollment_year	uni	
Meridel	Rawley	Willatts	rwillatts1t@si.edu	2022	mrw1	b2
Nonna	None	Gillanders	tgillanders9@example.com	2021	ng2	b2
Nickolai	Herby	Reddyhoff	hreddyhoff1g@vk.com	2016	nhr1	b2
Norry	None	Rubinchik	trubinchik16@howstuffworks.com	2016	nr1	b2
Quent	None	Threadgall	cthreadgall1v@booking.com	2022	qt1	b2
Reece	Corbett	Caps	ccapsz@telegraph.co.uk	2019	rcc1	b2
Rozelle	None	Vigar	mvigar8@vistaprint.com	2018	rv1	b2
Shea	Gail	Bates	gbates1e@alibaba.com	2018	sgb1	b2
Sada	None	Stidson	gstidsonp@ebay.com	2022	ss1	b2
Sterne	Sasha	Spooner	sspooner1f@wordpress.org	2020	sss1	b2
Tobiah	Gerek	Offield	goffield1u@addthis.com	2020	tgo1	b2
Travis	Melba	Colbourne	mcolbourne2j@mapquest.com	2017	tmc1	b2
Trix	None	Pretswell	jpretswell1h@xrea.com	2022	tp1	b2

first_name	middle_name	last_name	email	enrollment_year	uni	
Veronika	Cordey	Simeons	csimeons2@microsoft.com	2021	vcs1	b2
Vincenty	Elisabeth	Marousek	emarousek10@huffingtonpost.com	2020	vem1	b2
Vance	None	Maffioni	fmaffioni1r@amazon.com	2020	vm1	b2
Woodrow	Camile	Moughtin	cmoughtin17@illinois.edu	2022	wcm1	b2
Willi	None	Kerby	ekerby1p@amazon.com	2020	wk1	b2
Winslow	None	Scrauniage	jscrauniage28@aol.com	2017	ws1	b2
Zared	None	Fenelon	afenelona@themeforest.net	2021	zf1	b2

In [73]:
```
#
# Test that we loaded employees.
#
%sql select * from employee;
```

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

Out[73]:

first_name	middle_name	last_name	email	employee_type	uni	
Ailbert	Danie	Warmisham	dwarmishame@soundcloud.com	Staff	adw1	b2f
Ari	Rheta	Sellek	rsellek6@oakley.com	Lecturer	ars1	b2f
Bamby	Rubetta	Mabbs	rmabbs4@xing.com	Lecturer	brm1	b2f

first_name	middle_name	last_name	email	employee_type	uni	
Bonny	None	Scheffel	lscheffel7@taobao.com	Professor	bs1	b2f
Bettina	Sonya	Higgonet	shiggonet2b@163.com	Lecturer	bsh1	b2f
Cari	Andriana	Leask	aleask1n@devhub.com	Lecturer	cal1	b2f
Clim	None	Guislin	lguislin2o@chicagotribune.com	Professor	cg1	b2f
Carey	Maudie	Wyrall	mwyrallj@scientificamerican.com	Staff	cmw1	b2f
Christie	None	Siegertsz	hsiegertsz21@instagram.com	Professor	cs1	b2f
Carmine	None	Tolman	gtolmanr@slideshare.net	Staff	ct1	b2f
Doyle	None	Aslin	jaslin24@redcross.org	Lecturer	da2	b2f
Doug	None	Medforth	vmedforth1o@homestead.com	Staff	dm3	b2f
Darrin	Mario	Wynrahame	mwynrahamem@admin.ch	Professor	dmw1	b2f
Duncan	Shellie	Sillars	ssillars2l@unicef.org	Professor	dss1	b2f
Ethelin	None	Fidele	afidele12@google.ru	Lecturer	ef1	b2f
Electra	Krystle	Morfell	kmorfell2g@istockphoto.com	Professor	ekm1	b2f

first_name	middle_name	last_name	email	employee_type	uni	
Francene	None	MacNeely	nmacneely22@cpanel.net	Lecturer	fm1	b2f
Gisela	None	Blagden	gblagden1q@buzzfeed.com	Professor	gb1	b2f
Genni	None	Purbrick	lpurbrick25@canalblog.com	Professor	gp1	b2f
Hobart	Dominic	Croal	dcroalx@purevolume.com	Professor	hdc1	b2f
Holli	None	Henstridge	lhenstridgeh@sogou.com	Lecturer	hh1	b2f
Jacky	Nydia	Bolles	nbolles23@ucoz.ru	Staff	jnb1	b2f
Jany	Sherry	Johl	sjohlg@soundcloud.com	Professor	jsj1	b2f
Karon	None	Bree	ebree1z@creativecommons.org	Professor	kb1	b2f
Karole	None	Inkin	minkinc@google.de	Staff	ki1	b2f
Kristin	None	Malacrida	smalacrida1w@economist.com	Staff	km2	b2f
Kahaleel	Meg	Penzer	mpenzer14@dailymail.co.uk	Professor	kmp1	b2f
Lemmy	Burr	Bradnocke	bbradnockek@nifty.com	Lecturer	lbb1	b2f
Lu	Cinnamon	Flaxman	cflaxman1b@cdbaby.com	Lecturer	lcf1	b2f

first_name	middle_name	last_name	email	employee_type	uni	
Lelah	Ellette	Fulk	efulk1d@discuz.net	Staff	lef1	b2f
Lilllie	None	Snodin	jsnodin20@princeton.edu	Lecturer	ls1	b2f
Marylin	Darcy	Favey	dfavey2p@mozilla.com	Staff	mdf1	b2f
Maybelle	Esteban	Cella	ecella26@mail.ru	Staff	mec1	b2f
Meghan	None	Kleinschmidt	lkleinschmidtl@squarespace.com	Lecturer	mk2	b2f
Matthieu	Kalle	Slipper	kslipper2f@nytimes.com	Staff	mks1	b2f
Mendie	None	Tennick	ltennick3@aboutads.info	Staff	mt1	b2f
Niki	Gardiner	Form	gform18@blogger.com	Staff	ngf1	b2f
Nadia	None	Trehearn	ktrehearn19@tinyurl.com	Lecturer	nt1	b2f
Olwen	None	Hedley	jhedley1m@disqus.com	Staff	oh1	b2f
Pattie	None	Lyles	slyles1j@amazon.de	Staff	pl1	b2f
Robinett	Jami	Charte	jcharte1y@merriam-webster.com	Staff	rjc1	b2f
Renae	Jaquith	Plessing	jplessing0@samsung.com	Staff	rjp1	b2f

first_name	middle_name	last_name	email	employee_type	uni	
Sanders	Arline	Breckell	abreckell1x@fotki.com	Professor	sab1	b2f
Sibylle	Bearnard	Lalley	blalley2d@rediff.com	Lecturer	sbl1	b2f
Suki	None	Hellyar	ghellyar2a@cornell.edu	Staff	sh1	b2f
Sayers	Karon	McKnish	kmcknishs@reddit.com	Lecturer	skm1	b2f
Sasha	Win	Ruffli	wruffli2q@wordpress.com	Lecturer	swr1	b2f
Wells	None	Yousef	gyousef2r@spotify.com	Professor	wy1	b2f
Xenia	Ardella	Kief	akieft@free.fr	Staff	xak1	b2f
Yehudi	Sile	McColley	smccolleyq@amazon.com	Staff	ysm1	b2f

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.`

successful execution of each procedure

In [74]:
```
%%sql
SET @_uni = '', @_guid = '';
```

```
CALL create_employee('abc',NULL,'green','abc@abc.edu', 'Professor', @_uni, @_gui
SELECT * FROM people WHERE email = 'abc@abc.edu';
```

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

Out[74]:

uni	guid	last_name	first_name	middle_name	email	employee_type	enrollmer
ag1	b6862142-af0d-11ec-8e23-b2f5b3a00848	green	abc	None	abc@abc.edu	Professor	

In [75]:

```
%%sql
SET @_uni = '', @_guid = '';
CALL create_student('abc',NULL,'green','abc@abg.edu', '2021', @_uni, @_guid);
SELECT * FROM people WHERE email = 'abc@abg.edu';
```

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

Out[75]:

uni	guid	last_name	first_name	middle_name	email	employee_type	enrollmer
ag2	b6872eb6-af0d-11ec-8e23-b2f5b3a00848	green	abc	None	abc@abg.edu	None	

In [76]:

```
%%sql
CALL update_employee('adw1','hello','hi','world','abc@abc.com', 'Professor');
SELECT * FROM people WHERE uni = 'adw1';
```

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

Out[76]:

uni	guid	last_name	first_name	middle_name	email	employee_type	enrollm
adw1	b6789446-af0d-11ec-8e23-b2f5b3a00848	world	hello	hi	abc@abc.com	Professor	

In [77]:

```
%%sql
CALL update_student('cv1','hello','hi','world','hhh123@cg.edu', '2021');
SELECT * FROM people WHERE uni = 'cv1';
```

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

Out[77]:

uni	guid	last_name	first_name	middle_name	email	employee_type	enrollm

uni	guid	last_name	first_name	middle_name	email	employee_type	enrollm
cv1	b67f2b6c-af0d-11ec-8e23-b2f5b3a00848	world	hello	hi	hhh123@cg.edu	None	

In [78]:
```
%%sql
select * from people where last_name = 'world'
```

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

Out[78]:

uni	guid	last_name	first_name	middle_name	email	employee_type	enroll
cv1	b67f2b6c-af0d-11ec-8e23-b2f5b3a00848	world	hello	hi	hhh123@cg.edu	None	
adw1	b6789446-af0d-11ec-8e23-b2f5b3a00848	world	hello	hi	abc@abc.com	Professor	

failures on constraints

In [79]:
```
try:
    %sql USE dff9_s22_midterm
    %sql UPDATE employee SET uni = 'hh333' WHERE uni = 'adw1';
    print("uni constraint unchangable")
except Exception as e:
  print(e)

%sql SELECT * FROM people WHERE uni = 'adw1';
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
 * mysql+pymysql://root:***@localhost
(pymysql.err.OperationalError) (1644, 'Unhandled user-defined exception conditio
n')
[SQL: UPDATE employee SET uni = 'hh333' WHERE uni = 'adw1';]
(Background on this error at: https://sqlalche.me/e/14/e3q8)
uni constraint unchangable
 * mysql+pymysql://root:***@localhost
1 rows affected.
```

Out[79]:

uni	guid	last_name	first_name	middle_name	email	employee_type	enrollm
adw1	b6789446-af0d-11ec-8e23-b2f5b3a00848	world	hello	hi	abc@abc.com	Professor	

In [80]:
```
try:
    %sql USE dff9_s22_midterm
    %sql UPDATE employee SET first_name = NULL WHERE uni = 'adw1';
    print("NULL values unchangable")
except Exception as e:
```

```
  print(e)

%sql SELECT * FROM people WHERE uni = 'adw1';
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
 * mysql+pymysql://root:***@localhost
(pymysql.err.OperationalError) (1644, 'Unhandled user-defined exception conditio
n')
[SQL: UPDATE employee SET first_name = NULL WHERE uni = 'adw1';]
(Background on this error at: https://sqlalche.me/e/14/e3q8)
NULL values unchangable
 * mysql+pymysql://root:***@localhost
1 rows affected.
```

Out[80]:

uni	guid	last_name	first_name	middle_name	email	employee_type	enrollm
adw1	b6789446-af0d-11ec-8e23-b2f5b3a00848	world	hello	hi	abc@abc.com	Professor	

In [81]:

```
try:
    %sql USE dff9_s22_midterm
    %sql UPDATE employee SET middle_name = NULL WHERE uni = 'kb1';
    print("NULL values unchangable")
except Exception as e:
  print(e)

%sql SELECT * FROM people WHERE uni = 'kb1';
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
 * mysql+pymysql://root:***@localhost
(pymysql.err.OperationalError) (1644, 'Unhandled user-defined exception conditio
n')
[SQL: UPDATE employee SET middle_name = NULL WHERE uni = 'kb1';]
(Background on this error at: https://sqlalche.me/e/14/e3q8)
NULL values unchangable
 * mysql+pymysql://root:***@localhost
1 rows affected.
```

Out[81]:

uni	guid	last_name	first_name	middle_name	email	employee_
kb1	b678c52e-af0d-11ec-8e23-b2f5b3a00848	Bree	Karon	None	ebree1z@creativecommons.org	Prof

In [82]:

```
try:
    %sql USE dff9_s22_midterm
    %sql UPDATE employee SET employee_type = 'chef' WHERE uni = 'adw1';
    print("employee_type constraint failed")
except Exception as e:
  print(e)

%sql SELECT * FROM people WHERE uni = 'adw1';
```

```
 * mysql+pymysql://root:***@localhost
```

```
0 rows affected.
 * mysql+pymysql://root:***@localhost
(pymysql.err.OperationalError) (1644, 'Unhandled user-defined exception conditio
n')
[SQL: UPDATE employee SET employee_type = 'chef' WHERE uni = 'adw1';]
(Background on this error at: https://sqlalche.me/e/14/e3q8)
employee_type constraint failed
 * mysql+pymysql://root:***@localhost
1 rows affected.
```

Out[82]:

uni	guid	last_name	first_name	middle_name	email	employee_type	enrollm
adw1	b6789446-af0d-11ec-8e23-b2f5b3a00848	world	hello	hi	abc@abc.com	Professor	

In [83]:

```
try:
    %sql USE dff9_s22_midterm
    %sql UPDATE student SET enrollment_year = '1985' WHERE uni = 'aeb1';
    print("enrollment_year constraint failed")
except Exception as e:
  print(e)

%sql SELECT * FROM people WHERE uni = 'aeb1';
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
 * mysql+pymysql://root:***@localhost
(pymysql.err.OperationalError) (1644, 'Unhandled user-defined exception conditio
n')
[SQL: UPDATE student SET enrollment_year = '1985' WHERE uni = 'aeb1';]
(Background on this error at: https://sqlalche.me/e/14/e3q8)
enrollment_year constraint failed
 * mysql+pymysql://root:***@localhost
1 rows affected.
```

Out[83]:

uni	guid	last_name	first_name	middle_name	email	employee_type	enr
aeb1	b678ab16-af0d-11ec-8e23-b2f5b3a00848	Blissitt	Avie	Else	eblissitti@youtu.be	None	

In [84]:

```
try:
    %sql USE dff9_s22_midterm
    %sql UPDATE student SET email = 'blalley2d@rediff.com' WHERE uni = 'ceg1';
    print("unique email constraint on update failed")
except Exception as e:
  print(e)

%sql SELECT * FROM people WHERE email = 'blalley2d@rediff.com';
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
 * mysql+pymysql://root:***@localhost
(pymysql.err.OperationalError) (1644, 'Unhandled user-defined exception conditio
n')
[SQL: UPDATE student SET email = 'blalley2d@rediff.com' WHERE uni = 'ceg1';]
(Background on this error at: https://sqlalche.me/e/14/e3q8)
```

```
unique email constraint on update failed
 * mysql+pymysql://root:***@localhost
1 rows affected.
```

Out[84]:

uni	guid	last_name	first_name	middle_name	email	employee_type	er
sbl1	b67728d6-af0d-11ec-8e23-b2f5b3a00848	Lalley	Sibylle	Bearnard	blalley2d@rediff.com	Lecturer	

In [85]:

```
try:
    %sql USE dff9_s22_midterm
    %sql INSERT INTO student VALUES('liting', NULL,'huang', 'blalley2d@rediff.co
    print("Unique Email constraint on insert failed")
except Exception as e:
  print(e)

%sql SELECT * FROM people WHERE first_name = 'liting';
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
 * mysql+pymysql://root:***@localhost
(pymysql.err.OperationalError) (1644, 'Unhandled user-defined exception conditio
n')
[SQL: INSERT INTO student VALUES('liting', NULL,'huang', 'blalley2d@rediff.com',
2021, NULL, NULL);]
(Background on this error at: https://sqlalche.me/e/14/e3q8)
Unique Email constraint on insert failed
 * mysql+pymysql://root:***@localhost
0 rows affected.
```

Out[85]:

uni	guid	last_name	first_name	middle_name	email	employee_type	enrollment_year

In [ ]:

In [86]:

```
%%sql
use dff9_s22_midterm;
UPDATE employee SET first_name = 'hi' where uni = 'ars1';
SELECT * FROM people WHERE uni = 'ars1';
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
(pymysql.err.OperationalError) (1644, 'Unhandled user-defined exception conditio
n')
[SQL: UPDATE employee SET first_name = 'hi' where uni = 'ars1';]
(Background on this error at: https://sqlalche.me/e/14/e3q8)
```

In [87]:

```
%%sql
use dff9_s22_midterm;
UPDATE student SET first_name = 'hi' where uni = 'ceg1';
SELECT * FROM people WHERE uni = 'ceg1';
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
(pymysql.err.OperationalError) (1644, 'Unhandled user-defined exception conditio
n')
```

```
[SQL: UPDATE student SET first_name = 'hi' where uni = 'ceg1';]
(Background on this error at: https://sqlalche.me/e/14/e3q8)
```

In [88]:
```
%%sql
use dff9_s22_midterm;
INSERT INTO student VALUES ('liting', NULL, 'huang', 'lh3119@columbia.edu', );
SELECT * FROM people WHERE uni = 'ceg1';
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
(pymysql.err.ProgrammingError) (1064, "You have an error in your SQL syntax; che
ck the manual that corresponds to your MySQL server version for the right syntax
to use near ')' at line 1")
[SQL: INSERT INTO student VALUES ('liting', NULL, 'huang', 'lh3119@columbia.ed
u', );]
(Background on this error at: https://sqlalche.me/e/14/f405)
```

In [89]:
```
%%sql
DROP USER IF EXISTS 'general_user'@'localhost';
CREATE USER 'general_user'@'localhost' IDENTIFIED BY 'password';

GRANT ALL PRIVILEGES ON *.* TO 'general_user'@'localhost' WITH GRANT OPTION;
REVOKE INSERT, DELETE, UPDATE ON *.* FROM 'general_user'@'localhost';
```

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

Out[89]:
```
[]
```

In [90]:
```
%sql mysql+pymysql://general_user:password@localhost
```

Out[90]:
```
'Connected: general_user@None'
```

In [91]:
```
sql_conn = pymysql.connect(
    user="general_user",
    password='password',
    host="localhost",
    port=3306,
    cursorclass=pymysql.cursors.DictCursor,
    autocommit=True)
```

general user cannot use update, insert, delete

In [92]:
```
try:
    %sql USE dff9_s22_midterm
    %sql INSERT INTO student VALUES('liting', NULL,'huang', 'x@rediff.com', 2021
    print("general_user cannot call insert")
except Exception as e:
  print(e)
```

```
 * mysql+pymysql://general_user:***@localhost
  mysql+pymysql://root:***@localhost
```

```
0 rows affected.
 * mysql+pymysql://general_user:***@localhost
   mysql+pymysql://root:***@localhost
(pymysql.err.OperationalError) (1142, "INSERT command denied to user 'general_us
er'@'localhost' for table 'student'")
[SQL: INSERT INTO student VALUES('liting', NULL,'huang', 'x@rediff.com', 2021, N
ULL, NULL);]
(Background on this error at: https://sqlalche.me/e/14/e3q8)
general_user cannot call insert
```

In [93]:

```
try:
    %sql USE dff9_s22_midterm
    %sql UPDATE student SET email = 'y@soundcloud.com' WHERE uni = 'ceg1';
    print("general_user cannot call update")
except Exception as e:
  print(e)
```

```
 * mysql+pymysql://general_user:***@localhost
   mysql+pymysql://root:***@localhost
0 rows affected.
 * mysql+pymysql://general_user:***@localhost
   mysql+pymysql://root:***@localhost
(pymysql.err.OperationalError) (1142, "UPDATE command denied to user 'general_us
er'@'localhost' for table 'student'")
[SQL: UPDATE student SET email = 'y@soundcloud.com' WHERE uni = 'ceg1';]
(Background on this error at: https://sqlalche.me/e/14/e3q8)
general_user cannot call update
```

In [94]:

```
try:
    %sql USE dff9_s22_midterm
    %sql DELETE FROM student WHERE uni = 'ceg1';
    print("general_user cannot call DELETE")
except Exception as e:
  print(e)
```

```
 * mysql+pymysql://general_user:***@localhost
   mysql+pymysql://root:***@localhost
0 rows affected.
 * mysql+pymysql://general_user:***@localhost
   mysql+pymysql://root:***@localhost
(pymysql.err.OperationalError) (1142, "DELETE command denied to user 'general_us
er'@'localhost' for table 'student'")
[SQL: DELETE FROM student WHERE uni = 'ceg1';]
(Background on this error at: https://sqlalche.me/e/14/e3q8)
general_user cannot call DELETE
```

In [ ]:

successful call of create_student and create_employee

In [95]:

```
%%sql
SET @_uni = '', @_guid = '';
CALL create_student('alexis',NULL,'green','pg@k.edu', '2022', @_uni, @_guid);
```

```
 * mysql+pymysql://general_user:***@localhost
```

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

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

In [96]:
```
%sql SELECT * FROM people WHERE first_name = "alexis"
```

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

Out[96]:

uni	guid	last_name	first_name	middle_name	email	employee_type	enrollment_
ag3	b6a17e24-af0d-11ec-8e23-b2f5b3a00848	green	alexis	None	pg@k.edu	None	2

In [97]:
```
%%sql
SET @_uni = '', @_guid = '';
CALL create_employee('alea',NULL,'green','pg@ly.edu', 'Professor', @_uni, @_guid
```

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

Out[97]:
```
[]
```

In [98]:
```
%sql SELECT * FROM people WHERE first_name = "alea";
```

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

Out[98]:

uni	guid	last_name	first_name	middle_name	email	employee_type	enrollment_
ag4	b6a3ba36-af0d-11ec-8e23-b2f5b3a00848	green	alea	None	pg@ly.edu	Professor	N

successful call of update_student and update_employee

In [99]:
```
%%sql
CALL update_employee('ag1', 'bill', NULL, 'black', 'acd@gmail.com', 'Professor')
SELECT * FROM people WHERE uni = "ag1";
```

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

Out[99]:

uni	guid	last_name	first_name	middle_name	email	employee_type	enrollm

uni	guid	last_name	first_name	middle_name	email	employee_type	enrollm
ag1	b6862142-af0d-11ec-8e23-b2f5b3a00848	black	bill	None	acd@gmail.com	Professor	

In [100…

```
%%sql
CALL update_student('ag2', 'brea', NULL, NULL, 'adg@cd.edu', '2020');
SELECT * FROM people WHERE uni = "ag2"
```

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

Out[100…

uni	guid	last_name	first_name	middle_name	email	employee_type	enrollmen
ag2	b6872eb6-af0d-11ec-8e23-b2f5b3a00848	green	brea	None	adg@cd.edu	None	

In [ ]: