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

Information and Instructions

- The midterm exam is due on 04-NOV at 11:59 PM. **You may not use late days.**
- See the Ed post #403 for submission instructions.
- Students should periodically check Ed post #404 for clarifications.
- You may use lecture notes, lecture slides, ... to help answer questions. You may also use online sources of information. If you use an online source,
  1. You must provide a link to the source.
  2. You are still responsible for ensuring the answer is correct. Not everything on the web is correct.
  3. You **MUST NOT** simply cut and paste, copy verbatim, ... You can use the information for guidance but must provide the answer in your own words and own code.
- You **MUST NOT** collaborate with other students or other people in any way. You may discuss the exam with TAs and instructors.

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.
4. You will need to:
   A. Install the Classic Models database if you have not already done so.
   B. Install the sample database that comes with the recommended textbook if you have not already done so.

In [1]:
```
%load_ext sql
```

In [2]:
```
%sql mysql+pymysql://root:dbuserbdbuser@localhost
```

In [3]:
```
%sql select * from classicmodels.customers where country='Spain'
```

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

Out[3]:

customerNumber	customerName	contactLastName	contactFirstName	phone	addressLine1	addressLine2	city	state	postalCode	country	salesRepEmployee
141	Euro+ Shopping Channel	Freyre	Diego	(91) 555 94 44	C/ Moralzarzal, 86	None	Madrid	None	28034	Spain	
216	Enaco Distributors	Saavedra	Eduardo	(93) 203 4555	Rambla de Cataluña, 23	None	Barcelona	None	08022	Spain	
237	ANG Resellers	Camino	Alejandra	(91) 745 6555	Gran Vía, 1	None	Madrid	None	28001	Spain	
344	CAF Imports	Fernandez	Jesus	+34 913 728 555	Merchants House	27-30 Merchant's Quay	Madrid	None	28023	Spain	
458	Corrida Auto Replicas, Ltd	Sommer	Martín	(91) 555 22 82	C/ Araquil, 67	None	Madrid	None	28023	Spain	
465	Anton Designs, Ltd.	Anton	Carmen	+34 913 728555	c/ Gobelas, 19-1 Urb. La Florida	None	Madrid	None	28023	Spain	
484	Iberia Gift Imports, Corp.	Roel	José Pedro	(95) 555 82 82	C/ Romero, 33	None	Sevilla	None	41101	Spain	

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

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

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

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

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

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

Out[8]:

	customerName	customerNumber	city	country
0	Euro+ Shopping Channel	141	Madrid	Spain
1	Enaco Distributors	216	Barcelona	Spain
2	ANG Resellers	237	Madrid	Spain
3	CAF Imports	344	Madrid	Spain
4	Corrida Auto Replicas, Ltd	458	Madrid	Spain
5	Anton Designs, Ltd.	465	Madrid	Spain
6	Iberia Gift Imports, Corp.	484	Sevilla	Spain

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

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

In [11]:
```
try:
    cur = sql_conn.cursor()
    res = cur.execute(sql)
    res = cur.fetchall()
except Exception as e:
    print("Exception ", e, "is probably NOT good.")
```

In [12]:
```
res
```

Out[12]:
```
[{'customerName': 'Euro+ Shopping Channel',
  'customerNumber': 141,
  'city': 'Madrid',
  'country': 'Spain'},
 {'customerName': 'Enaco Distributors',
  'customerNumber': 216,
  'city': 'Barcelona',
  'country': 'Spain'},
 {'customerName': 'ANG Resellers',
  'customerNumber': 237,
  'city': 'Madrid',
  'country': 'Spain'},
 {'customerName': 'CAF Imports',
  'customerNumber': 344,
  'city': 'Madrid',
  'country': 'Spain'},
 {'customerName': 'Corrida Auto Replicas, Ltd',
  'customerNumber': 458,
  'city': 'Madrid',
  'country': 'Spain'},
 {'customerName': 'Anton Designs, Ltd.',
  'customerNumber': 465,
  'city': 'Madrid',
  'country': 'Spain'},
 {'customerName': 'Iberia Gift Imports, Corp.',
  'customerNumber': 484,
  'city': 'Sevilla',
  'country': 'Spain'}]
```

In [13]:
```
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. *Candidate Key* and *Super Key*: A super key is any set of columns that identifies a row in a table in a unique way. A candidate key is a super key that cannot have any of its columns removed without losing its ability to uniquely identify data.
2. *Primary Key* and *Unique Key*: Primary key cannot accept NULL values but Unique key can accept NULL values. A table can have only one primary key but can have multiple unique key.
3. *Natural Key* and *Surrogate Key*: A surrogate key is a system-generated value (often an integer sequence) that is used to identify a record in a table in a unique way. The value of the key has no business meaning. Whereas, a natural key is a column or set of columns that already exist in the table that uniquely identify a record in the table. The key is made up of real data which has business meaning.

W2

SQL supports the modifier *ON UPDATE* and *ON DELETE* for foreign key definitions. The database engines do not support *ON INSERT*. Why would implementing *ON INSERT* be impossible in most scenarios?

Answer
ON INSERT means if a parent with an id is instert, a corresponding id will be automatically instert into a child. But in most cases, "INSERT an id in parent" has NO influence on its child. Hence, implementing *ON INSERT* is impossible in most scenarios.

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."

Consider a table of the form:

```
create table if not exists orders
(
    uni           varchar(12) not null
        primary key,
    last_name     varchar(64) not null,
    first_name    varchar(64) not null,
    age           int         null
)
```

If we do not know the value of `age,` a poor design would use a convention like setting age to `-1` instead of using `NULL`. Give an example of a query for which not following Codd's Thurd Rule would result in an incorrect answer.

Answer
If we use `SELECT COUNT(age IS NOT NULL) FROM orders` to detect the total numer of age value that we do not know, we cannot get the correct answer since we set age to -1 instead of using NULL.

W4

The relational model and SQL are *closed* under their operations. Briefly explain why this concept is critical joining three tables?

Answer
JOIN is a binary operation, it is use to join two tables. When we want to join three tables, we should join two tables first(since relational model and SQL are closed under their operations, the result we get is a table as well) and then join the result we get with the third table. In this situation, we always join two tables every time, but the total number of tables we can join is far more than two.

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 for which it is impossible to update the base table through the view.
2. One view for which it is possible to update through the view.

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

```
create table S22_W4111_Midterm.midterm_employees
(
    social_security_no char(9) not null
        primary key,
    last_name varchar(64) not null,
    first_name varchar(64) not null,
```

```
    dept_no char(4) not null,
    salary double not null
);
```

Answer

1. impossible to update:
```
CREATE VIEW departments_highest_salary(dept_no, highest_salary) as
 SELECT dept_no, MAX(salary)
 FROM S22_W4111_Midterm.midterm_employees
 GROUP BY dept_no
```
2. impossible to update:
```
CREATE VIEW departments_total_salary(dept_no, total_salary) as
 SELECT dept_no, SUM(salary)
 FROM S22_W4111_Midterm.midterm_employees
 GROUP BY dept_no
```
3. possible to update:
```
CREATE VIEW faculty as
 SELECT social_security_no, last_name, first_name, dept_no
 FROM S22_W4111_Midterm.midterm_employees
```

W6

Consider the following table:

```
create table S22_W4111_Midterm.midterm_employees
(
    phone_number varchar(64) not null primary key,
    last_name varchar(64) not null,
    first_name varchar(64) not null,
);
```

Telephone numbers are of the form `country code` followed by the phone number. Some examples are:

- `01 212-555-1212`
- `44 038 717 980 01`

Why is storing the number as a single `varchar` a poor design? What problems could that cause? How would you change the table definition.

Answer
When we store the phone number as a single `varchar`, the users are allowed to enter a string which contains letters, numbers, and special characters. However, we only hope the user enter a valid phone number(which only allow numerical number, '-', space, and '()') in the phone_number column. Hence, we can add a CHECK constraint in the table definition `CHECK(ISNUMERIC(REPLACE(TRANSLATE(phone_number, '-()', '   '),' ','')))`

W7

Briefly explain the differences between:

- Database stored procedure
- Database function
- Database trigger

Answer

1. Database stored procedure: We can execute the stored procedures when required. Stored procedures might or might not return a value or table. Stored procedures can be called by Trigger but cannot by Function.
2. Database function: We can call a function whenever required but function cannot be executed. Function must return a value or table. Function can be called by Store procedure or Trigger.
3. Database trigger: Trigger can be executed automatically only when some triggering event such as INSERT, UPDATE, or DELETE operations occur in a table. Trigger never return any values. Trigger cannot be called by Store Procedure or Function.

W8

Briefly explain:

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

Give a scenario in which a Natural Join would produce an incorrect answer.

Answer

1. Natural join: A natural join join two tables based on the common columns(columns with the same name and domain) in the tables.
2. Equi-join: An equi-join join two tables based on matching('=') values in specified columns which columns' name can be different.
3. Theta join: A theta join join two tables based on the condition represented by theta. Theta joins work for all comparison operators.
4. Self-join: A self join join a table with itself as if the table were two tables.

If no column names are found in common between the two tables, we use Natural Join, then a Cross Join is performed.

W9

We have seen examples in SQL of implementing relationships between two tables using an *associative entity* table instead of foreign keys. Give two reasons for using the associative entity design pattern.

Answer

1. Associative entity table can handle both one-to-one, one-to-many and many-to-many relationships
2. Association entity table can be added to the database without modifying tables already exists

W10

Professor Ferguson often adds a `LIMIT` to his example queries in Jupyter Notebooks. Assume the table `customers` is very large. Why would the query

```
select * from customers
```

cause problems for the notebook? How does addining `limit 20` solve this problem for an example?

Answer
If try to load numerous rows into the Jupiter notebook, the notebook will at best hang, and will most likely crash the Browser because the Jupiter notebook environment cannot handle this huge size data.
When adding `limit 20`, we can process the data a little bit at a time, so we run the query, and it computed the answer.

Relational Algebra

R1

- You can assume that the type for the columns in this question are `varchar(32).`
- 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.

$$(\underline{policy\_type}, \underline{policy\_no}, policy\_date) \tag{1}$$

Answer

CREATE TABLE policy (
policy_type varchar(32) NOT NULL PRIMARY KEY,
policy_no varchar(32) NOT NULL PRIMARY KEY,
policy_date varchar(32)
);

R2

Use the RelaX calculator with the textbook's sample data for this question.

Answer Format: Your answer to the relational algebra query should contain three secions:

1. A Markdown cell with the relational algebra statement.
2. An image capture of the query exection tree.
3. An image capture of the result table.

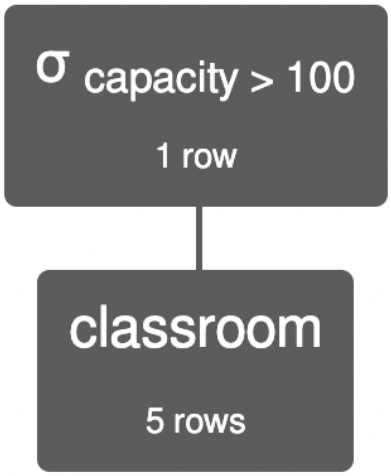
For example, a query returning all classrooms with `capacity > 100` would have the following cells:

Relational Algebra Statement

```
σ capacity > 100 (classroom)
```

Query Execution Tree

This must show the execution tree and relational algebra statement.



$$\sigma_{\text{capacity} > 100}(\text{classroom})$$

Execution time: 1 ms

Query Result

This must show the relational algebra statement and the result table.

$$\sigma_{\text{capacity} > 100}(\text{classroom})$$

Execution time: 1 ms

classroom.building	classroom.room_number	classroom.capacity
'Packard'	101	500

The Question

In the sample data,

- The relation `advisor` represents the advisor-student relationship between a `student` and an `instructor.`
- Write a relational algebra expression that produces the following information:
  - The `ID` and `name` of `student`, and the `ID` and `name` on `instructor`
  - For students and instructors in the CS department.
  - The information should be `null` if the instructor does not advise a student and vice-versa.
- To help, you are trying to produce the following information.
- **Note:**
  - You **may not** use full outer join.
  - You will have to use the column rename operation for project.

student_id	student_name	instructor_id	instructor_name

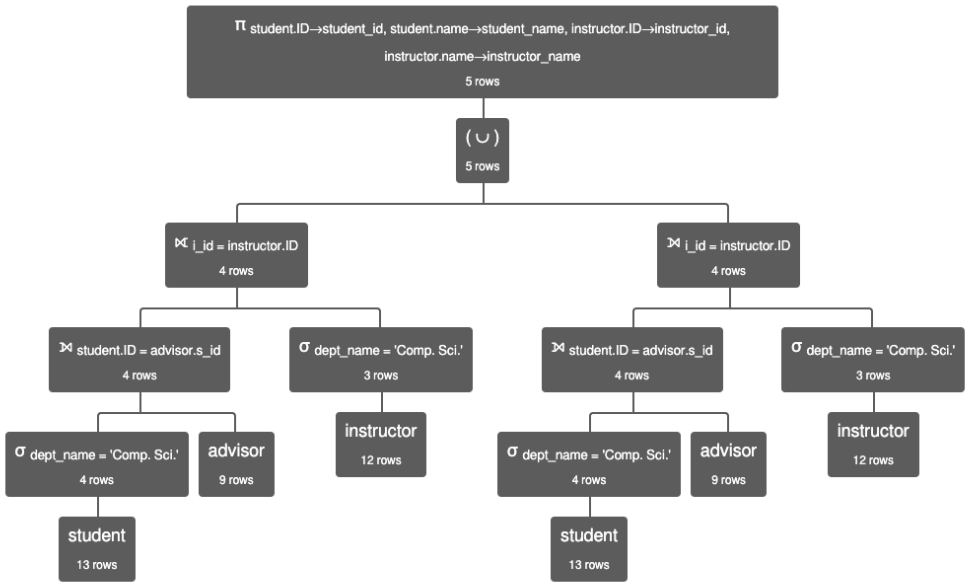
student_id	student_name	instructor_id	instructor_name
12345	'Shankar'	10101	'Srinivasan'
128	'Zhang'	45565	'Katz'
76543	'Brown'	45565	'Katz'
null	null	83821	'Brandt'
54321	'Williams'	null	null

Answer

Query

```
π student_id ← student.ID, student_name ← student.name, instructor_id ← instructor.ID, instructor_name ←
instructor.name
((σ dept_name='Comp. Sci.'(student)) ⋈ student.ID = advisor.s_id (advisor) ⋉ i_id=instructor.ID (σ dept_name='Comp.
Sci.'(instructor))
∪
(σ dept_name='Comp. Sci.'(student)) ⋈ student.ID = advisor.s_id (advisor) ⋈ i_id=instructor.ID (σ dept_name='Comp.
Sci.'(instructor)))
```

Query Tree



π student.ID→student_id, student.name→student_name, instructor.ID→instructor_id, instructor.name→instructor_name ((((σ dept_name = 'Comp. Sci.' (student)) ⋈ student.ID = advisor.s_id (advisor)) ⋉ i_id = instructor.ID (σ dept_name = 'Comp. Sci.' (instructor))) ∪ (((σ dept_name = 'Comp. Sci.' (student)) ⋈ student.ID = advisor.s_id (advisor)) ⋈ i_id = instructor.ID (σ dept_name = 'Comp. Sci.' (instructor))))

Execution time: 3 ms

Query Result

π student.ID→student_id, student.name→student_name, instructor.ID→instructor_id, instructor.name→instructor_name ((((σ dept_name = 'Comp. Sci.' (student)) ⋈ student.ID = advisor.s_id (advisor)) ⋉ i_id = instructor.ID (σ dept_name = 'Comp. Sci.' (instructor))) ∪ (((σ dept_name = 'Comp. Sci.' (student)) ⋈ student.ID = advisor.s_id (advisor)) ⋈ i_id = instructor.ID (σ dept_name = 'Comp. Sci.' (instructor))))

Execution time: 3 ms

student_id	student_name	instructor_id	instructor_name
12345	'Shankar'	10101	'Srinivasan'
128	'Zhang'	45565	'Katz'
76543	'Brown'	45565	'Katz'
null	*null*	83821	'Brandt'
54321	'Williams'	*null*	*null*

Entity Relationship Model and Implementation

Explanation

For this problem,

- There is a written description of a data model.
- You must draw (using Lucidchart) a Crow's Foot notation ER diagram for the *logical model* implementing the written description. Note that not all concepts in the data model description can be modeled in the ER diagram.
- You must then write SQL DDL statements and execute the statements to create tables and constraints realizing the written data model description.

Written Description

There are the following entity types:

- `employee`:
  - `employee_id` is a 4 digit number that may begin with 0, e.g. `0201`. An employee must have a unique `employee_id`.
  - `last_name` is a string with maximum length 64. An employee must have a last name.
  - `first_name` is a string with maximum length 64. An employee must have a first name.
  - `employee_type` must be one of the following values, `regular`, `manager`, `executive.`
  - `employee_email` may be unknown, but if known it must be unique.
- `project`:
  - `project_code` is a two character code that must contain two uppercase English letters (A, B, ..., Z) and is unique.
  - `project_name` is a text string of maximum length 32.
- `project_team` is an associative entity of the form (none of the values may be NULL):
  - `project_code`
  - `sponsor_id` is `employee_id` of an employee who is an `executive.`
  - `manager_id` is the `employee_id` of an employee who is a `manager`.
  - `employee_id` is the `employee_id` of an employee working on the project.
- Constraints on `project_team`:
  - `project_code` is unique in the table.
  - An `employee_id` can appear at most three times.
  - The combination of `(sponsor_id, manager_id)` can appear at most once.

Note:

Being able to make sense out of a written description of a data model and producing a reasonably accurate diagram and DDL is an important skill. Most of the time, you will have to make assumptions or modify/extend constraints. The business statekholder/partner specifying the data model is not a database expert. There description may be incomplete or confused.

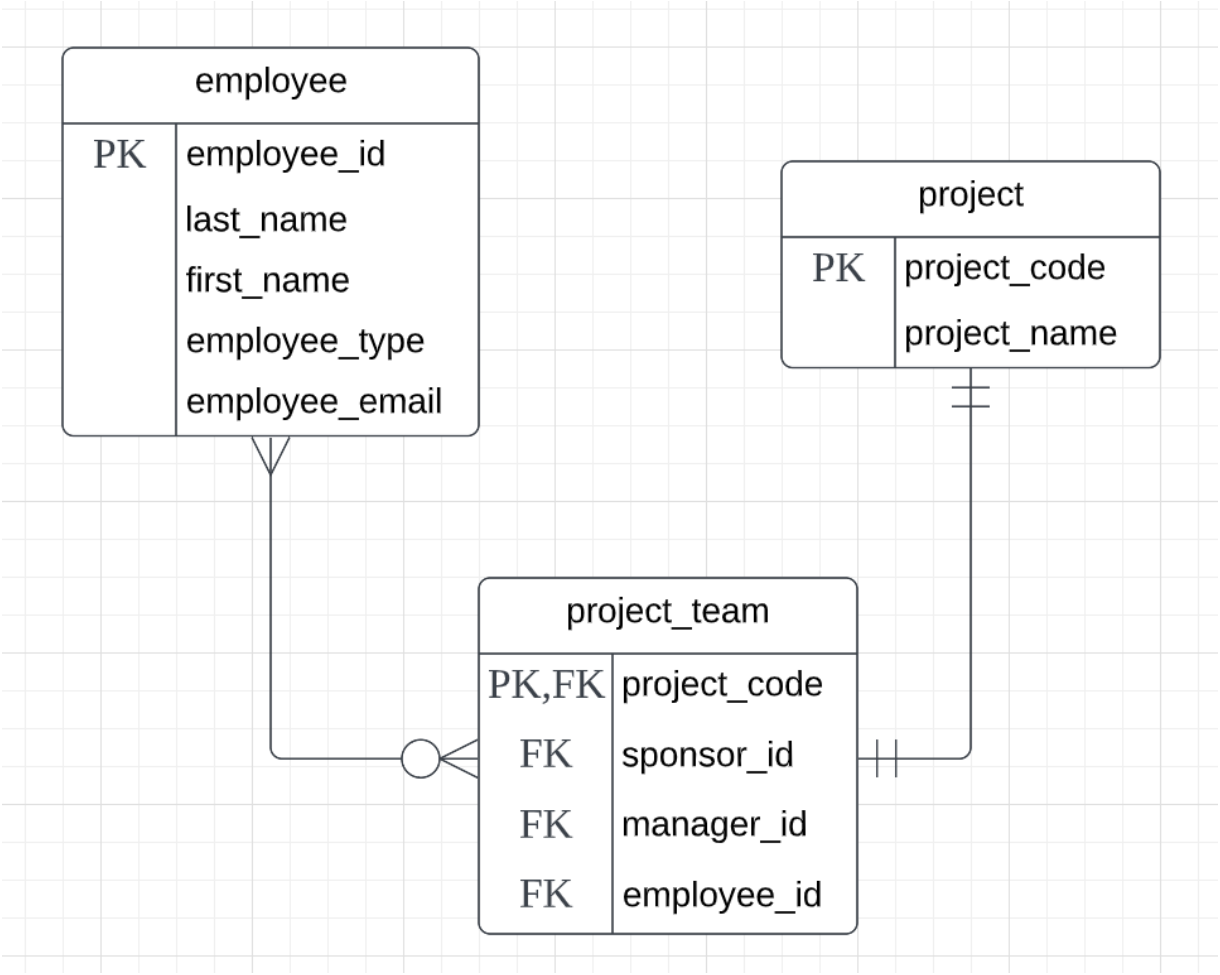
We are looking for your ability to apply what you have learned to a complex problem. If you have to make assumptions, note them. We will not deduct points for reasonable assumptions.

You may have to use check constraints, triggers, foreign keys, ... in your DDL.

Answer

Crow's Foot ER Diagram Assumption:

- For each `project_team`, we can find one and only one `project`. For each `project`, we can find 0 or 1 `project_team` because maybe some projects are new and not decide which employees to deal with yet.
- For each `project_team`, we can find three `employee` (one manager, one executive, one regular employee)
- For each `employee`, we can find 0 or more `project_team` because there may be new employees who not assign to a team yet.



DDL Statements and Execution

In [14]:
```
%%sql

drop database if exists f22_midterm;

create database f22_midterm;
```

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

Out[14]:
```
[]
```

In [15]:
```
%%sql

USE f22_midterm;

CREATE TABLE employee
(
    employee_id int(4) NOT NULL PRIMARY KEY,
    last_name varchar(64) NOT NULL,
    first_name varchar(64) NOT NULL,
    employee_type ENUM('regular', 'manager', 'executive'),
    employee_email varchar(255) UNIQUE
);

CREATE TABLE project
(
    project_code varchar(2) NOT NULL PRIMARY KEY,
    project_name varchar(32) NOT NULL,
    CHECK ( REGEXP_LIKE(project_code, '^[A-Z][A-Z]$') )
);

CREATE TABLE project_team
(
    project_code varchar(2) NOT NULL PRIMARY KEY,
    sponsor_id int NOT NULL,
    manager_id int NOT NULL,
    employee_id int NOT NULL,
    CONSTRAINT FK_project_code
        FOREIGN KEY (project_code) REFERENCES project(project_code),
    CONSTRAINT FK_sponsor_id
        FOREIGN KEY (sponsor_id) REFERENCES employee(employee_id),
    CONSTRAINT FK_manager_id
        FOREIGN KEY (manager_id) REFERENCES employee(employee_id),
```

```
    CONSTRAINT FK_employee_id
        FOREIGN KEY (employee_id) REFERENCES employee(employee_id)
);

ALTER TABLE project_team
  ADD CONSTRAINT UQ_sm UNIQUE(sponsor_id, manager_id);

CREATE TRIGGER employee_id_count
BEFORE INSERT
ON project_team FOR EACH ROW
BEGIN
    IF NEW.employee_id IN (SELECT employee_id, COUNT(*) AS total
                           FROM project_team
                           GROUP BY employee_id
                           HAVING total=3)
        THEN SIGNAL SQLSTATE '45002' SET MESSAGE_TEXT = 'employee_id appears more than three times';
    END IF;
END;

CREATE TRIGGER check_sponsor_id
BEFORE INSERT
ON project_team FOR EACH ROW
BEGIN
    IF NEW.sponsor_id NOT IN (SELECT employee_id
                              FROM employee
                              WHERE employee_type = 'executive')
        THEN SIGNAL SQLSTATE '45002' SET MESSAGE_TEXT = 'This is not a executive id';
    END IF;
END;

CREATE TRIGGER check_manager_id
BEFORE INSERT
ON project_team FOR EACH ROW
BEGIN
    IF NEW.manager_id NOT IN (SELECT employee_id
                              FROM employee
                              WHERE employee_type = 'manager')
        THEN SIGNAL SQLSTATE '45002' SET MESSAGE_TEXT = 'This is not a manager id';
    END IF;
END;

CREATE TRIGGER check_employee_id
BEFORE INSERT
ON project_team FOR EACH ROW
BEGIN
    IF NEW.employee_id NOT IN (SELECT employee_id
                              FROM employee
                              WHERE employee_type = 'regular')
        THEN SIGNAL SQLSTATE '45002' SET MESSAGE_TEXT = 'This is not a regular employee id';
    END IF;
END;
```

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

Out[15]:
```
[]
```

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: $(country, total_country_revenue)$.
- Each entry in `orderdetails` produces revenue `quantityOrdered*priceEach`.
- The revenue an `order` produces is the sum of the revenue from the `orderdetails` in the `order`, *but only if the order's status is* `shipped.`
- An `order` has a `customer` and the `customer` is in a country. The `total_country_revenue` is the sum over all shipped orders for customers in a country.
- The result table should have `total_country_revenue` nicely formated, sorted descending and have only countries with `total_country_revenue` >= 200,000.
- **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. You must match the output.

In [16]:
```
%%sql

USE classicmodels;

WITH order_revenue AS (
    SELECT orderdetails.orderNumber,
           sum(quantityOrdered*orderdetails.priceEach) AS order_revenue
    FROM orderdetails
    INNER JOIN orders ON orderdetails.orderNumber = orders.orderNumber
    WHERE orders.status = 'Shipped'
    GROUP BY orderNumber
),
    order_country AS (
    SELECT orders.orderNumber,
           customers.country
    FROM customers
    INNER JOIN orders ON customers.customerNumber = orders.customerNumber
),
    total AS (
    SELECT country,
           sum(order_revenue) AS total_country_revenue
    FROM order_revenue
    INNER JOIN order_country ON order_revenue.orderNumber = order_country.orderNumber
    GROUP BY country
    HAVING total_country_revenue >= 200000
    ORDER BY total_country_revenue DESC
    )

SELECT country,
       CONCAT('$', FORMAT(total_country_revenue, 2)) AS total_country_revenue
FROM total
```

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

Out[16]:

country	total_country_revenue
USA	$3,032,204.26
France	$965,750.58

Spain	$947,470.01
Australia	$509,385.82
New Zealand	$416,114.03
UK	$391,503.90
Italy	$360,616.81
Finland	$295,149.35
Norway	$270,846.30
Singapore	$263,997.78
Canada	$205,911.86

Out[17]:

country	total_country_revenue
USA	$3,032,204.26
France	$965,750.58
Spain	$947,470.01
Australia	$509,385.82
New Zealand	$416,114.03
UK	$391,503.90
Italy	$360,616.81
Finland	$295,149.35
Norway	$270,846.30
Singapore	$263,997.78
Canada	$205,911.86

S2

Return the product information for products not ordered by any French customer (Customer's `country` is France).

I did not want to get hit by Batman again. So, here is a sample answer.

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

Out[18]:

productCode	productName	productLine	productScale	productVendor	productDescription	quantityInStock	buyPrice	MSRP
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	7733	57.01	107.57
S18_4027	1970 Triumph Spitfire	Classic Cars	1:18	Min Lin Diecast	Features include opening and closing doors. Color: White.	5545	91.92	143.62

In [17]:

```
%%sql

WITH code_not_France AS (
    SELECT products.productCode AS productCode,
           GROUP_CONCAT(DISTINCT country SEPARATOR ';') AS countries
    FROM products
    LEFT JOIN orderdetails ON orderdetails.productCode = products.productCode
    LEFT JOIN orders ON orderdetails.orderNumber = orders.orderNumber
    LEFT JOIN customers ON orders.customerNumber = customers.customerNumber
    GROUP BY products.productCode
    HAVING countries NOT LIKE '%France%' OR countries IS NULL
)
```

```
SELECT *
FROM products
WHERE productCode = ANY (SELECT productCode FROM code_not_France)
```

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

Out[17]:

productCode	productName	productLine	productScale	productVendor	productDescription	quantityInStock	buyPrice	MSRP
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	7733	57.01	107.57
S18_4027	1970 Triumph Spitfire	Classic Cars	1:18	Min Lin Diecast	Features include opening and closing doors. Color: White.	5545	91.92	143.62