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	ac
141	Euro+ Shopping Channel	Freyre	Diego	(91) 555 94 44	C/ Moralzarzal, 86	
216	Enaco Distributors	Saavedra	Eduardo	(93) 203 4555	Rambla de Cataluña, 23	
237	ANG Resellers	Camino	Alejandra	(91) 745 6555	Gran Vía, 1	
344	CAF Imports	Fernandez	Jesus	+34 913 728 555	Merchants House	
458	Corrida Auto Replicas, Ltd	Sommer	Martín	(91) 555 22 82	C/ Araquil, 67	
465	Anton Designs, Ltd.	Anton	Carmen	+34 913 728555	c/ Gobelas, 19-1 Urb. La Florida	
484	Iberia Gift Imports, Corp.	Roel	José Pedro	(95) 555 82 82	C/ Romero, 33	

In [5]:
```
from sqlalchemy import create_engine
```

In [6]:
```
sql_engine = create_engine("mysql+pymysql://root:dbuserbdbuser@localhost")
```

In [7]:
```
import pandas as pd
```

In [8]:
```
sql = """
    select customerName, customerNumber, city, country from classicmodels.customers
```

```
        where country = 'Spain'
"""

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

In [9]:
```
res
```

Out[9]:

	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 [10]:
```
import pymysql
```

In [11]:
```
sql_conn = pymysql.connect(
    user="root",
    password='dbuserbdbuser',
    host="localhost",
    port=3306,
    cursorclass=pymysql.cursors.DictCursor,
    autocommit=True)
```

In [12]:
```
try:
    cur = sql_conn.cursor()
    res = cur.execute(sql)
    res = cur.fetchall()
except Exception as e:
    print("Exception ", e, "is probably NOT good.")
```

In [13]:
```
res
```

Out[13]:

```
[{'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

- Both candidate key and super key are unique while candidate key is minimal.
- Unique key and PK are both unique, while unique key can be null but PK cannot be null.
- The natural key is the field that acts as the primary key, which has some meaning in itself and is part of the record ,while a surrogate key is a primary key in which the field itself has no business meaning and only serves as a primary key

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

- If the parent table inserts new data, but the child table is not updated, then the insert is meaningless. Therefore, in most scenarios, ON INSERT need not to be implementing.

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 For example

- SELECT MIN(age) FROM orders GROUP BY last_name

W4

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

Answer

- If the relational model and SQL are not closed, the result of table A join table B cannot join with table C. Therefore, the query could benested if and only if relation model and SQL are closed.

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. (1)CREATE VIEW person AS (SELECT last_name,first_name,salary FROM S22_W4111_Midterm.midterm_employees ORDER BY salary) (2) CREATE VIEW salary AS (SELECT dept_no,SUM(salary) FROM S22_W4111_Midterm.midterm_employees GROUP BY dept_no)
- 1. CREATE VIEW rich_employee AS (SELECT * 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

- Because obviously the phone numbers are not a string, it contains country code and the phone number. If the two parts have been stored together, it is difficult to exectue some queries such as select the phone number in the same country.
- Therefore, I'd like to define country code and phone number seperately and store them as two attributes.

W7

Briefly explain the differences between:

- Database stored procedure
- Database function
- Database trigger

- Triggers can change data. Triggers never return value. Triggers is called by reaction.
- Functions do not change data. Functions always return value. Functions is called in a statement.
- Stored procedures can change data. Procedures sometimes return values. Procedures is called by execution.

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

- Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column.

- Equi-join creates a join for equal or matching column values of the relative table. Reference:https://www.geeksforgeeks.org/sql-equi-join-and-non-equi-join/
- A theta join is a join that links tables based on a relationship other than equality between two columns. A theta join could use any operator other than the “equal” operator. The following example and procedure show you how to create a theta join that uses the “Between” operator. Reference:http://pfy.carpenters.org/OnlineHelp/mergedProjects/businessobjects/mergedF
- A self join is a join in which a table is joined with itself (which is also called Unary relationships), especially when the table has a FOREIGN KEY which references its own PRIMARY KEY. To join a table itself means that each row of the table is combined with itself and with every other row of the table. Reference:https://www.w3resource.com/sql/joins/perform-a-self-join.php#:~:text=A%20self%20join%20is%20a%20join%20in%20which,and%20with%20ev

For example, there is a table students which has arrtibutes ID,dept_name,grades wants to natural join with another table professors which has attributes ID,dept_name,student_id. The ID in table students represents student's ID and the ID in professors represents professor's ID. If we use natural join to join them together, the result is obviously incorrect.

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

- In some scenarios, in order to avoid redundant attributes, we need to use associative entity.
- In designing a database schema, associative entity can help to avoid incompleteness.

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 Because the table customers is too large so that it may take a long time to make a query. If add 'limit' 20 to the query, the data is not so huge so the query can execute quickly.

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 %%sql CREATE TABLE SAMPLE (policy_type varchar(32), policy_no varchar(32) not null, policy_date varchar(32) PRIMARY KEY (policy_no,policy_type));

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.

Query Result

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

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.

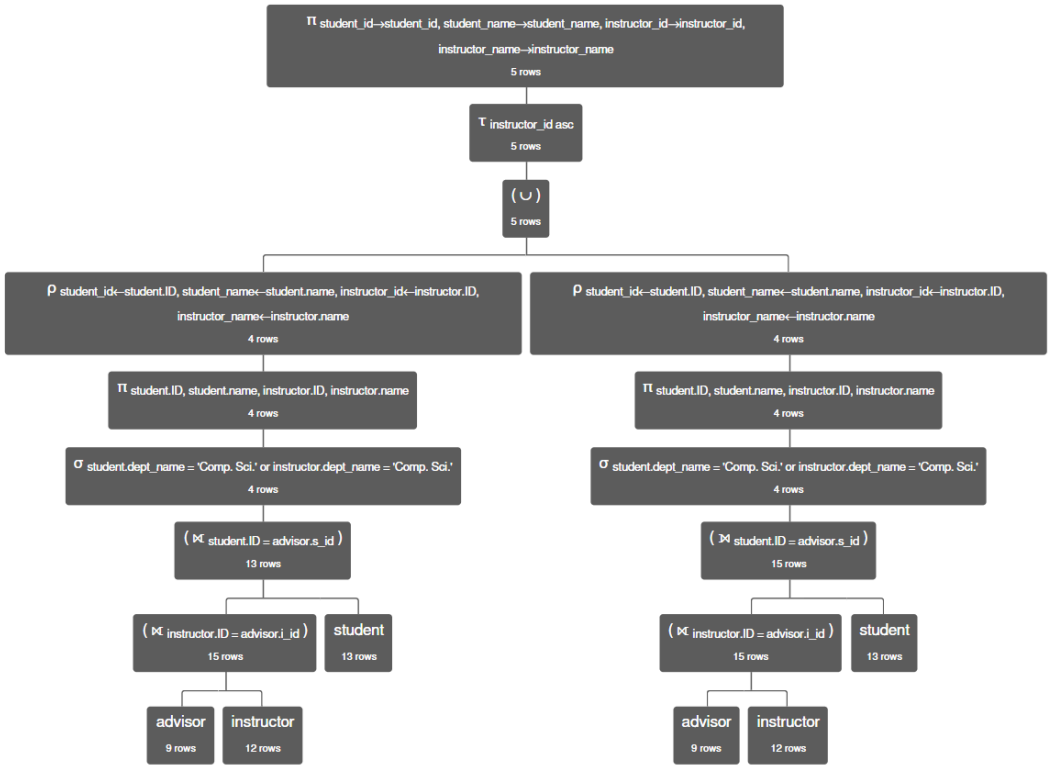
student_id	student_name	instructor_id	instructor_name
12345	'Shankar'	10101	'Srinivasan'
128	'Zhang'	45565	'Katz'
76543	'Brown'	45565	'Katz'
null	null	83821	'Brandt'
54321	'Williams'	null	null

Answer

Query

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

Query Tree



Query Result

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:

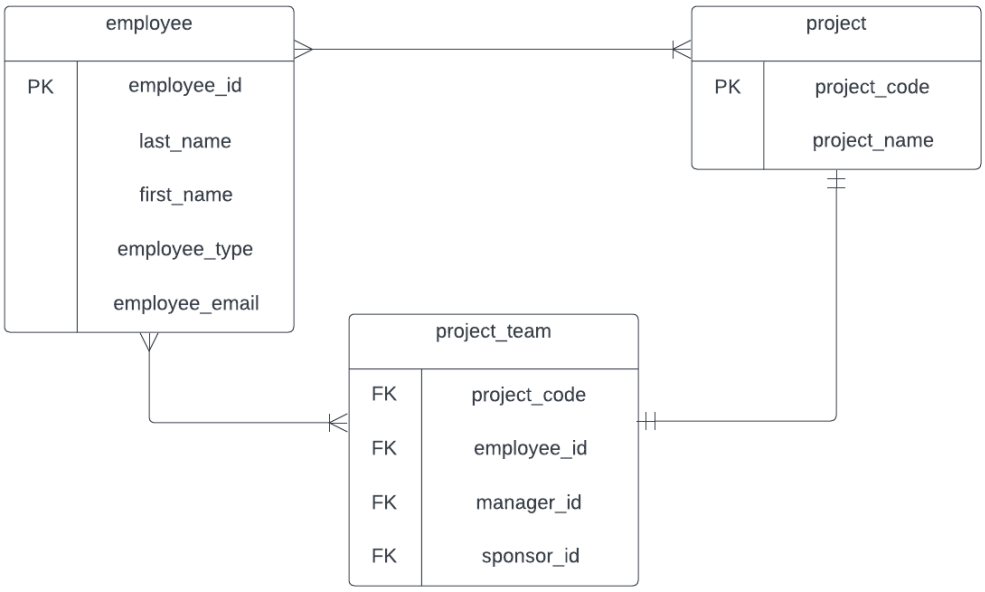
Being able to make sense out of a written description of a data model and producing a reasonably accurate diagram and DDL is an important skill. Most of the time, you will have to make assumptions or modify/extend constraints. The business statekholder/partner specifying the data model is not a database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



DDL Statements and Execution

In [30]:
```
%%sql

drop database if exists f22_midterm;

create database f22_midterm;
```

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

Out[30]:

In [31]:
```
%sql use f22_midterm
```

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

Out[31]:

In [32]:
```
%%sql
CREATE TABLE employee (
 employee_id decimal(4) ZEROFILL,
 last_name varchar(64),
 first_name varchar(64),
 employee_type varchar(15),
 employee_email varchar(60) UNIQUE,
 PRIMARY KEY (employee_id),
 CHECK (employee_type in ('regular','manager', 'executive'))
);
```

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

Out[32]:

In [37]:
```
%%sql
CREATE TABLE project (
 project_code char(2) UNIQUE,
 project_name text(32),
 PRIMARY KEY (project_code),
 CONSTRAINT check_upper
 CHECK (NOT(CAST(project_code AS BINARY) REGEXP BINARY '[a-z]'))
);
```

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

Out[37]:
```
[]
```

- Assume that each team have only one sponsor ,one manager and one employee.
- Assume that the employee_id in project_team is regular in table employee.

In [50]:
```
%%sql
CREATE TABLE project_team (
 project_code char(2) UNIQUE,
 sponsor_id decimal(4) ZEROFILL,
 manager_id decimal(4) ZEROFILL,
 employee_id decimal(4) ZEROFILL,
 FOREIGN KEY (project_code) REFERENCES project(project_code),
 FOREIGN KEY (sponsor_id) REFERENCES employee(employee_id),
 FOREIGN KEY (manager_id) REFERENCES employee(employee_id),
 FOREIGN KEY (employee_id) REFERENCES employee(employee_id)
);
```

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

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

In [52]:
```
%%sql
CREATE TRIGGER unique_pair
    BEFORE INSERT
    ON project_team
    FOR EACH ROW
BEGIN
    IF (SELECT COUNT(*) FROM project_team WHERE sponsor_id = new.sponsor_id ANI
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'The (manager,sponsor) pair
    END IF;
END;
```

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

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

In [51]:
```
%%sql
CREATE TRIGGER employee_type
BEFORE INSERT
ON project_team
FOR EACH ROW
BEGIN
    IF NOT(New.sponsor_id in (SELECT employee_id from employee where employee.e
        New.manager_id in (SELECT employee_id from employee where employee.emplo
        New.employee_id in (SELECT employee_id from employee where employee.empl
        SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = 'Employee type do not match'
    END IF;
END;
```

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

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

In [54]:
```
%%sql
CREATE TRIGGER three_teams_constraint
BEFORE INSERT
ON project_team
FOR EACH ROW
BEGIN
    IF (SELECT COUNT(*) from project_team where employee_id = new.employee_id )
        SIGNAL SQLSTATE '45002' SET MESSAGE_TEXT = 'An employee can join at mos
    END IF;
END;
```

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

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

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 [56]:
```
%sql use classicmodels;
```

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

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

In [58]:
```
%%sql

DROP TABLE Revenue;

CREATE TABLE Revenue AS(
 SELECT
  orderNumber,
  country,
  quantityOrdered*priceEach AS  revenue
 FROM
  orderdetails

 INNER JOIN
  orders
 USING (orderNumber)

 INNER JOIN
  customers
 USING (customerNumber)
 WHERE status = 'Shipped')
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
2771 rows affected.
```

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

In [59]:
```
%%sql

DROP TABLE Revenue_per_order;

CREATE TABLE Revenue_per_order AS(
 SELECT
  country,
  SUM(revenue) AS revenue_per_order
 FROM
  Revenue
 GROUP BY orderNumber)
```

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

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

In [64]:
```
%%sql

DROP TABLE Total_country_revenue;

CREATE TABLE Total_country_revenue AS(
 SELECT
  country,
  SUM( Revenue_per_order) AS total_country
 FROM
   Revenue_per_order
 GROUP BY country)
```

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

Out[64]:

In [65]:
```
%%sql
SELECT
 country,
 CONCAT('$', FORMAT(total_country, 2)) AS total_country_revenue
FROM
 Total_country_revenue
WHERE total_country >= 200000
ORDER BY total_country DESC
```

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

Out[65]:

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.

In [29]:
```
%%sql
CREATE TABLE France_product AS(
SELECT
 productCode
FROM
 customers

INNER JOIN
 orders
USING (customerNumber)
```

```
INNER JOIN
 orderdetails
USING (orderNumber)
WHERE
 country = 'France')
```

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

Out[29]: []

In [30]:
```
%%sql
SELECT
 *
FROM
 products

LEFT JOIN
 France_product
USING(productCode)
WHERE
 France_product.productCode IS NULL
```

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

Out[30]:

productCode	productName	productLine	productScale	productVendor	productDescription	qu
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	
S18_4027	1970 Triumph Spitfire	Classic Cars	1:18	Min Lin Diecast	Features include opening and closing doors. Color: White.	