Brandon Vo

Problem 1: In this problem, you have to write SQL and RA queries for an online book store database.

Here are the tables:

Customers: (Customer_id, Customer_Last_Name, Customer_First_Name, City, State, Zip)

Publisher: (Publisher_id, Pub_Name, Headquarter, Phone)

Author: (Author_id, Last_Name, First_Name)

Category: (Category_id, Category_name)

Books: (ISBN, Author_id, Title, PublishDate, Publisher_id, Price, Category_id)
Publisher_id references Publisher(Publisher_id)
Category_id references Category(Category_id)
Author_id references Author(Author_id)

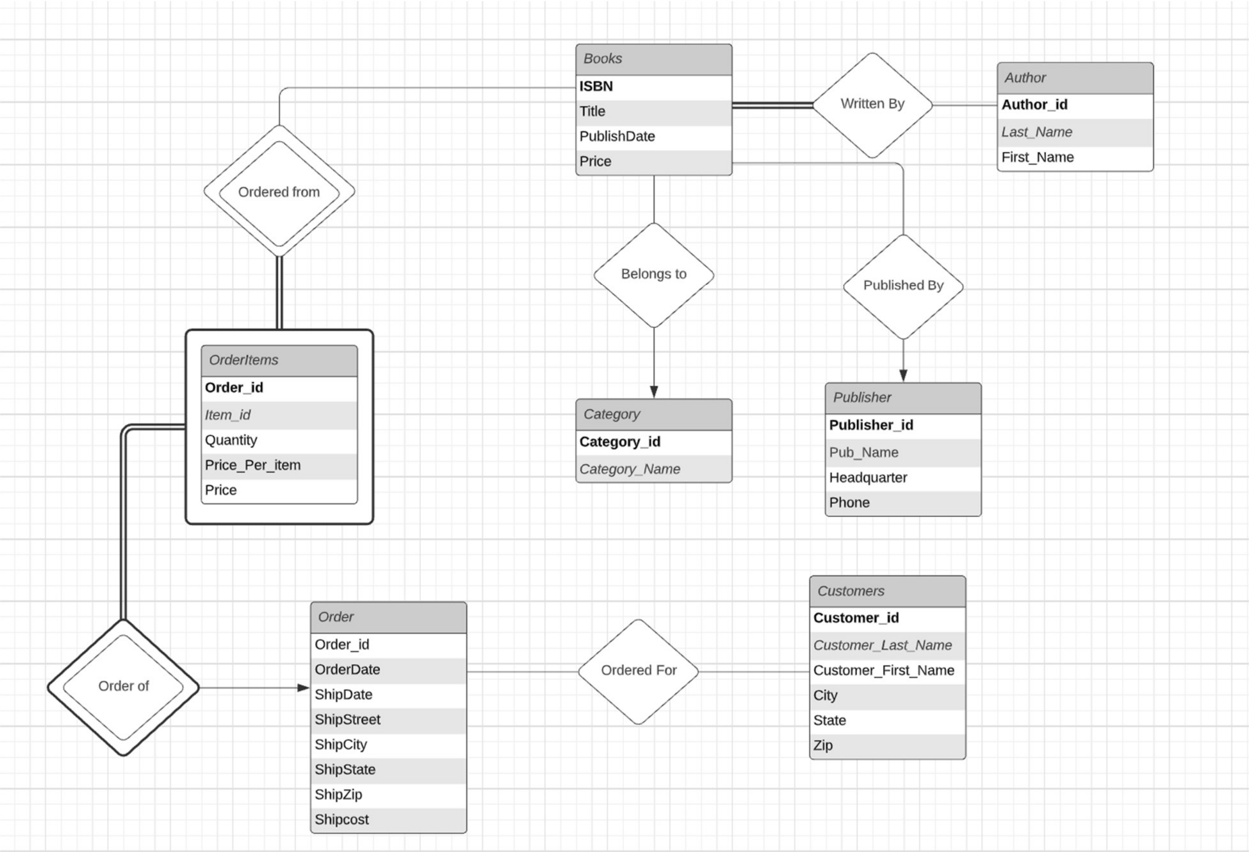
OrderItems: (Order_id, Item_id, ISBN, Quantity, Price_Per_Item)
ISBN referencesBooks(ISBN)

Orders: (Order_id, Customer_id, OrderDate, ShipDate, ShipStreet, ShipCity, ShipState, ShipZip, ShipCost)
Customer_id references Customers(Customer_id)

This schema provides data on an online book store's sales. There are customers, each with a unique id, who can buy books from this store. Each book uses an ISBN as a unique identifier and we may assume each book only has one author. The customers can buy multiple books within one order, and if an order has five distinct books, we would use Item_id values of 1 to 5 for these. A customer may also buy multiple copies of the same book, which is stored in the Quantity attribute in OrderItems. The entity Category models different types or genres of books, say Comics, Horror, Sports etc.

(a) Draw an ER diagram that models this relational schema. Identify any weak entities and the cardinalities of all the relationships.(8 marks)

Brandon Vo



Weak Entity set: Order

- Category wasn't considered a weak entity set because it's assumed that a category can exist and not have any books

(b) Create the above schema in a database system, choose appropriate attributes types, and define primary keys, foreign keys and other constraints. You may use any relational database system, as long as it supports basic SQL, views, and some sort of triggers. Write the create table statements.(5 marks)

```
    CREATE TABLE customers (
      Customer_id INT NOT NULL,
      Customer_Last_Name VARCHAR(40) NOT NULL,
      Customer_First_Name VARCHAR(40) NOT NULL,
      City VARCHAR(30),
      State VARCHAR(15),
      Zip VARCHAR(15),
      PRIMARY KEY (Customer_id));

CREATE TABLE publisher (
  Publisher_id INT NOT NULL,
  Pub_Name VARCHAR(45) NOT NULL,
  Headquarter VARCHAR(45),
  Phone VARCHAR(15),
  PRIMARY KEY (`Publisher_id`));

CREATE TABLE author (
  Author_id INT NOT NULL,
  Last_Name VARCHAR(40),
  First_Name VARCHAR(40),
  PRIMARY KEY (Author_id));

CREATE TABLE category (
  Category_id INT NOT NULL,
  Category_name VARCHAR(20),
  PRIMARY KEY (Category_id));

CREATE TABLE books (
 ISBN varchar(17),
  Author_id int references author(author_id),
  Title VARCHAR(45),
  PublishDate date,
  Publisher_Id int references publisher(publisher_id),
  Price int NULL,
  Category_id int references category(category_id),
  PRIMARY KEY (ISBN));

CREATE TABLE orderitems (
order_id INT NOT NULL,
item_id VARCHAR(45) NOT NULL,
ISBN VARCHAR(17) references books(ISBN),
quantity INT NOT NULL,
price_per_item INT NOT NULL,
PRIMARY KEY (order_id, item_id));

CREATE TABLE orders (
  Order_id INT AUTO_INCREMENT NOT NULL references orderitems(order_id),
  customer_id INT references customers(customer_id),
  orderdate date NOT NULL,
  shipdate date,
  shipstreet VARCHAR(45) NOT NULL,
  shipcity VARCHAR(45) NOT NULL,
  shipstate VARCHAR(45) NOT NULL,
  shipzip INT NOT NULL,
  shipcost INT NOT NULL,
  PRIMARY KEY (Order_id));
```

Brandon Vo

(c) Write the following SQL queries and execute them on your database. Show the queries and the screenshot of the results: (22 marks)

I. For each book category, find the book with the longest name.

- For this problem, a subquery had to be used because group by would take precedence otherwise.

```
select title, category_name, max
from (select title, category_name, max(length(title)) as max
        from books, category
  where books.category_id = category.category_id
  group by title
        order by length(title) desc
  ) test
group by category_name;
```

II. Output the 10 books that had the most copies purchased in 2021, along with the number of copies purchased.

```
select title, quantity
from books, orders, orderitems
where orderitems.order_id = orders.order_id
and orderitems.ISBN = books.ISBN
and Year(orderdate) = 2021
group by books.ISBN
order by quantity desc
limit 10
```

III. Find those orders that have not been shipped yet, and the name of the city the books are being shipped to. Display the name of the customers as well.

```
select distinct orders.order_id, customer_last_name, customer_first_name, shipcity, shipdate
from orders, customers
where (shipdate is Null
or shipdate > current_date())
and orders.customer_id = customers.customer_id;
```

IV. Show a list of authors who have written a book in the “Science Fiction” category that has never sold even a single copy in the store.

```
Select title, last_name, first_name
from author, books, category, orderitems
where books.author_id = author.author_id
and category.category_id = books.category_id
and category_name = "Science Fiction"
and books.isbn not in (select orderitems.isbn
                                        from orderitems)
```

```
group by author.author_id;
```

V. Display all distinct books purchased by a customer named “Robert Lane”.

```
select distinct title
from books, customers, orders, orderitems
where customers.Customer_Last_Name = "Lane"
and customers.Customer_First_name = "Robert"
and customers.customer_id = orders.customer_id
and orderitems.order_id = orders.order_id
and books.ISBN = orderitems.isbn;
```

VI. Show all books in the “Entertainment” category that were written by an author whose last name is “Kardashian”

```
select distinct title
from books, author, category
where last_name = "Kardashian"
and author.author_id = books.Author_id
and category.category_id = books.category_id
and category_name = "Entertainment";
```

VII. For each year from 2010 to 2020, output the category with the highest total sales (sum of purchase prices) in that year.

```
select year(orderdate), category_name, sum(price_per_item * quantity) as sum_purchase
from category, books, orderitems, orders
where orders.order_id = orderitems.order_id
and category.category_id = books.category_id
and orderitems.isbn = books.isbn
and year(orderdate) > 2010
and year(orderdate) < 2020
group by year(orders.orderdate)
order by orders.orderdate asc;
```

VIII. Delete any publishers from the database that have never published a book.

```
DELETE FROM publisher

where publisher_id not in (select publisher_id

                                        from books);
```

IX. The company decides that any customer who bought books for more than $200 in 2021 should get a free copy of the new Harry Potter book, “Harry Potter and the Castle of Doom”. (You should assume that the book has already been inserted into the database.)

Write an update query that creates records in Orders and OrderItems for this book for each such customer, with price $0.

- Orders and OrderItems have auto_increment specified for their primary keys
- Because an order is being placed, the shipdate will be null to indicate it hasn’t been shipped yet.
- Because a query can’t be inserted into two tables at once, two insert queries are used

```
insert into orders(Order_id, customer_id, OrderDate, shipdate, shipstreet, shipcity, shipstate, shipzip,
shipcost)

(select
CASE orders.order_id
   WHEN orders.order_id < (select max(order_id) from orders) then orders.order_id = (select
max(order_id from orders) + 1
END
, customer_id, current_date(), null, shipstreet, shipcity, shipstate, shipzip, shipcost
from orders, orderitems
where orderitems.order_id = orders.order_id
and year(orderdate) = 2021
group by customer_id
having sum(quantity*price_per_item) > 200
);
```

- ‘HARRY POTTER’ is a placeholder ISBN for what the actual ISBN for the corresponding ISBN for Harry Potter and the Castle of Doom book.
- Because the item_id is labelled 1-5 for the set of books we do have, 6 will be used to identify the free harry potter book

```
insert into orderitems(order_id, item_id, isbn, quantity, price_per_item)
(select orderitems.order_id, 6, 'HARRY POTTER', 1, 1
from orderitems, orders
where orderitems.order_id = orders.order_id
group by customer_id
having sum(quantity*price_per_item) > 200
)
;
```

X. Increase the price of each book in the category “Physics” by 10%. (Only increase the current price of the books, not the amount paid in past or current orders.)

```
update books, category
	set price = price * 1.1
where category.category_id = books.category_id
and category_name = "Physics"
```

Brandon Vo

d) Write expressions in Relational Algebra for queries I - VII.(14 marks)

I. For each book category, find the book with the longest name.

$S_1 \text{<-} \; _{(\text{category\_name, len(title)})}\gamma_{\text{len}}(\text{title}) \text{ as total\_length}(\text{Books} \bowtie_{\text{books.category\_id=category.category\_id}} \text{Category})$

$S_2 \text{ <-} \; _{(\text{category\_name, max(total\_length)})}\gamma_{\text{max(total\_length)}} \text{ as max\_length}(S_1)$

$\pi_{\text{title,category\_name,max\_length}}(\text{Category} \bowtie_{\text{category.category\_name=S2.category\_name}} S_2)$

II. Output the 10 books that had the most copies purchased in 2021, along with the number of copies purchased.

$S_1 \text{<-} \; (_{orders.order\_id,\ sum(quantity)})\, \gamma_{\text{sum}(quantity)} \text{ as total\_sold } (\sigma_{\text{Year(OrderDate)=2021}} (\text{Order} \bowtie_{\text{orders.order\_id=orderitems.order\_id}} (\text{Books} \bowtie_{\text{orderitems.isbn=books.isbn}} \text{Orderitems}))$

$S_2 \text{<-} (_{orders.order\_id,\ sum(quantity)})\, \gamma_{\text{max}(quantity)} \text{ as max\_sold } (S_1$

$\pi_{\text{title, sum(quantity), total\_sold}} (\sigma_{\text{orders.quantity} \geq \text{S2.max\_sold}} (\text{Orders} \bowtie_{\text{orders.order\_id=S2.order\_id}} S_2)))$

III. Find those orders that have not been shipped yet, and the name of the city the books are being shipped to. Display the name of the customers as well.

$\Pi_{\text{customer\_last\_name,customer\_first\_name,shipcity}} (\sigma_{\text{shipdate=null} \vee \text{shipdate>Date(Current\_Date)}} (\text{Customers} \bowtie_{\text{customer.customer\_id=orders.customer\_id}} \text{orders}))$

IV. Show a list of authors who have written a book in the “Science Fiction” category that has never sold even a single copy in the store.

$\Pi_{\text{title,last\_name, first\_name}} (\sigma_{\text{category\_name="Science Fiction"}} (\text{Author} \bowtie_{\text{books.author\_id = author.author\_id}} (\text{Category} \bowtie_{\text{category.category\_id=books.category\_id}} \text{books})))$

$-\Pi_{\text{title,last\_name, first\_name}} (\text{orderitems} \bowtie_{\text{books.isbn =orderitems.isbn}} (\sigma_{\text{category\_name="Science Fiction"}} (\text{Author} \bowtie_{\text{books.author\_id = author.author\_id}} (\text{Category} \bowtie_{\text{category.category\_id=books.category\_id}} \text{books})))$

V. Display all distinct books purchased by a customer named “Robert Lane”

$\Pi_{\text{books.isbn,title}} (\sigma_{\text{Customer\_First\_name="Robert"} \wedge \text{Customer\_Last\_Name="Lane"}} (\text{customer} \bowtie_{\text{customer.customer\_id=orders.customer\_id}} (\text{orders} \bowtie_{\text{orders.order\_id=orderitems.order\_id}} (\text{orderitems} \bowtie_{\text{books.isbn=orderitems.isbn}} \text{Books}))))$

VI. Show all books in the “Entertainment” category that were written by an author whose last name is “Kardashian”

$\Pi_{\text{isbn,title, last\_name, first\_name}} (\sigma_{\text{last\_name="Kardashian"}} (\text{author} \bowtie_{\text{author.author\_id=books.author\_id}} (\sigma_{\text{category\_name="Entertainment"}} (\text{books} \bowtie_{\text{books.category\_id=category.category\_id}} \text{category}))))$

VII. For each year from 2010 to 2020, output the category with the highest total sales (sum of purchase prices) in that year.

$_{(\text{category\_name, len(title)})}\gamma_{\text{len}}(\text{title}) \text{ as total\_length}$

Brandon Vo

$Y_1$ <-$_{\text{(orderitems.order\_id, category\_name,(price\_per\_item*quantity)}}$ $\gamma_{sum}$ $as\ sum\_purchase$$(\sigma_{\text{orderdate}\geq 2010\ \vee\ \text{orderdate}\ \leq\ 2020}$ $(\text{Category}\bowtie_{\text{category.category\_id=Books.category\_id}}$ $(\text{Books} \bowtie_{\text{books.isbn=orderitems.isbn}}$ $(\text{Orderitems}$ $\bowtie_{\text{order.order\_id=orderitems.order\_id}}$ orders)))))

$(\Pi_{year(orderdate),category\_name,sum\_purchase}$ $(\sigma_{\text{Y1.order\_id=orderitems.order\_id}}$ $(Y_1$ x orderitems))

Brandon Vo

Problem 2: In this problem, you need to create views and triggers given the following relational schema about a chocolate-tasting experiment. Based on the rating from users, you should help the producer decide if the production of a particular chocolate needs to be increased or decreased, and which city has an effect on demand and supply.

Chocolate (choc_id, choc_name, brand, category, price, weight, calorie)

User(uid, uname, preferred_category, city, country)

Rating (uid, choc_id, timestamp, rating, comment)

For each chocolate, we have a unique choc_id, the choc_name, the brand of the chocolate (e.g., Reese's, Hershey, Lindt ...), category of the chocolate product (e.g. truffles, bar, candy, ...), the price, the net weight, and the calorie count. Whenever a user tastes a chocolate and adds his/her rating for a chocolate, its timestamp is recorded. Ratings are from 1 to 5, and the user can also add a comment with the rating. For each user, we have a unique uid, a user name, the user's preferred category of chocolate, and the city and country.

(a) Define a view that stores, for each user, the choc_id, choc_name, brand, weight, calories, and rating, of all chocolates in the user's preferred category that the user has rated.(5 marks)

- Because user is a keyword in MySQL, the table user was renamed to users

```
Create view v as (select chocolate.choc_id, choc_name, brand, weight, calorie, rating
                  from chocolate, users, rating
                  where chocolate.choc_id = rating.choc_id
                  and rating.uid = users.uid
                  and users.preferred_category = chocolate.category
                  and rating is not NULL)
```

(b) Using this view, output the choc_name and brand that has the lowest calories per weight, among those that the user gave a rating of 5. (5 marks)

- Assuming this question was asking for the lowest calories per weight for every choc_name and brand found

```
select choc_name, brand, rating, (calorie/weight) as calories_per_weight
from v
where rating = 5
group by choc_name, brand
order by calories_per_weight asc
```

(c) Users can rate the same chocolate several times, but we want to limit how often they can rate it. Write a trigger that rejects any insertion of a new rating of a particular chocolate (identified by choc_id) in a specific month by a user(uid) who had already rated the same chocolate three times in that month.(8 marks)

- Delimiter was needed for these SQL queries to work for me.

Brandon Vo

```
DELIMITER $$

create trigger rate_limit before insert on rating
for each row
begin
        declare test INT;
   set test = (select count(choc_Id)
                                    from rating
                                    where month(rating.`timestamp`) = month(new.`timestamp`)
         and new.choc_id = rating.choc_id
                                    group by choc_id
   );
        if (test > 3)
        then signal sqlstate '45000';
        end if
end;
DELIMITER;
```

Brandon Vo

Problem 3: Suppose you are helping company Max.Inc build up a new database for tracking job candidates. The database should record the information about job candidates, interviewers, and recruiters.

The database has to store basic personal information such as first name, last name, resume (a piece of text), phone number, email address, position applied for, and application date for each candidate. Each candidate can only have one current application for a job, and can only apply for three positions within 12 months. Open positions for which candidates may apply are identified by a position ID, a short description, a department in which the position is located, and a set of tags describing the expertise needed for the position (see below for more on tags).

After a candidate applied for a certain position, the status of this candidate should be marked as “applied”. If a recruiter decides to move the candidate to the next step, the status should be changed to “in progress”, or otherwise the status can be changed to “rejected”. We also need to store some basic information about the recruiters, say their name, phone number, and maybe an ID. For each application, there is one recruiter who is responsible for it.

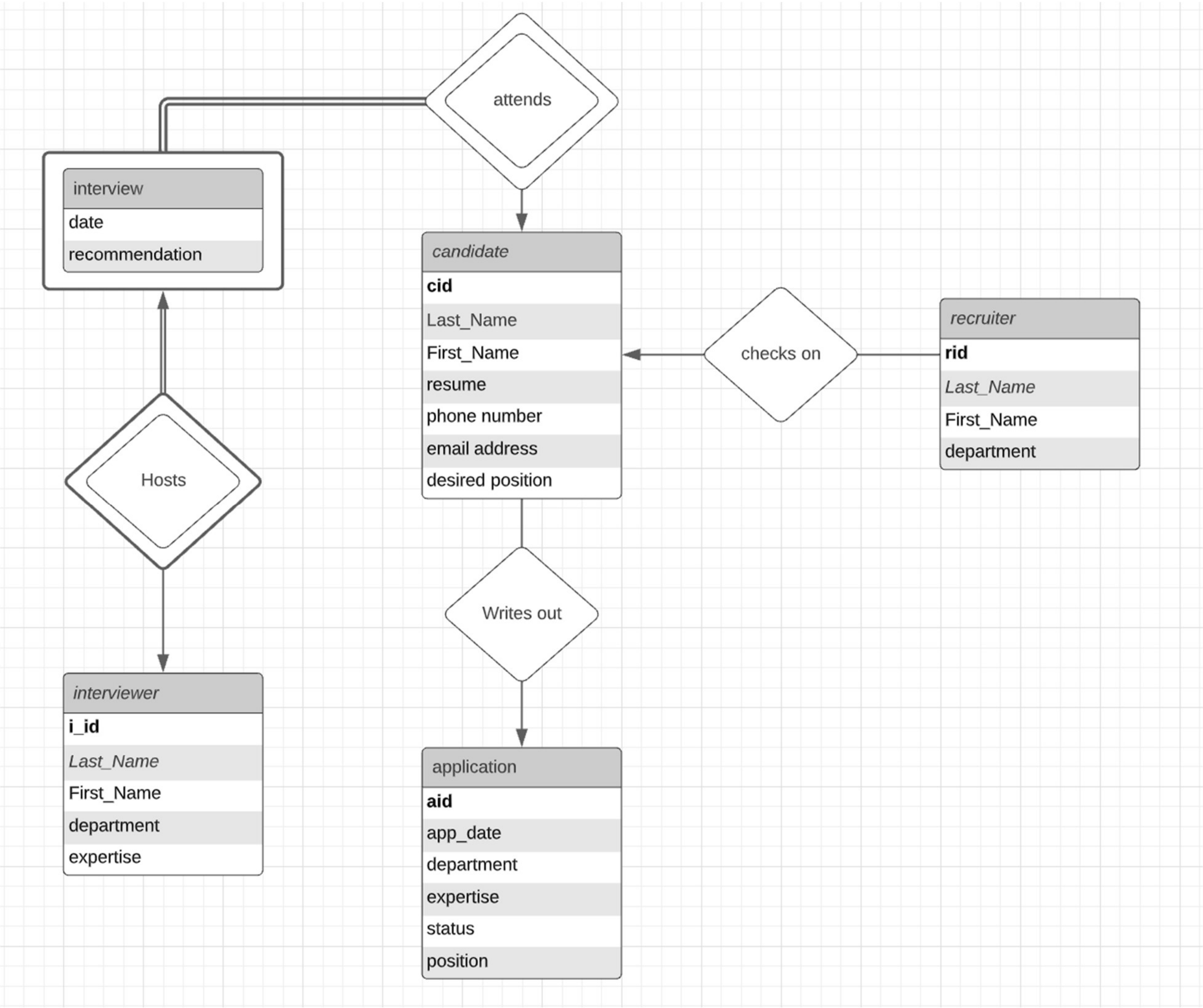
In addition to recruiters, there are interviewers, who are company employees that are available to interview applicants. Basic information about interviewers is stored in the database, including their names, phone numbers, department, and expertise. An interviewer’s expertise is stored as a set of tags chosen from a fixed dictionary of tags (e.g., “production”, “databases”, “sales”). The same dictionary of tags is also used to describe the expertise needed for a position (see above). After an application is moved to “in progress”, the recruiter will select a set of interviewers, and schedule a phone interview between the applicant and each of the chosen interviewers. The actual communication to find a convenient time for each interview is done outside the database, say via email, so you do not have to model that. However, once an interview has been scheduled, the information about the interview (who takes part in the interview and when it takes place) has to be stored in the database. After the interview completes, each interviewer can upload a short evaluation of the candidate and a hiring recommendation from 0 to 3 (3=“definite hire”, 2=“possible hire”, 1=“probably not”, or 0=“definitely do not hire”).

Finally, the recruiter will change the status of the application to either “make offer” or “reject”.

(a) Design a database for the above scenario using the ER model. Draw the ER diagram, show the cardinalities of all relationships, and identify primary keys and any weak entities. List any assumptions you have as well. (10 marks)

This assumes that every interview meeting is a one-on-one interview.

Brandon Vo



Weak Entity set: interview

(b) Convert your ER diagram into a relational schema. Identify all tables, attributes, primary keys, and foreign keys. (5 marks)

**Interviewers**: (**i_id** (Primary Key), Last_Name, First_Name, Department, expertise)

- I_id: int, primary key
- Last_Name: varchar(20)
- First_Name: varchar(20)
- Department: varchar(20)
- Expertise: varchar(15)

**Candidate: (cid** (Primary Key), Last_Name, First_Name, resume, phone_number, email_address, desired_position)

Assuming a resume can be optional for submission.

- cid: Int, primary key
- Last_name: varchar (20)
- First_Name: varchar(20)
- Resume: varchar(50)
- Phone_number: varchar(15), not null
- Email_address: varchar(30), not null
- Desired_position: varchar(20), not null

**Recruiter: (rid** (Primary Key), Last_Name, First_Name, department)

- Rid: int, primary key
- Last_Name: varchar(20), not null
- First_Name: varchar(20), not null
- Department: varchar(20), not null

**Interview: (**cid, i_id, date, recommendation)
cid references Candidate(cid)
i_id references Interviewers(i_id)cid: foreign key

- i_id: foreign key, int
- cid: foreign key, int
- date: dateandtime, not null
- recommendation: int, not null

**Application:** (aid (Primary Key), cid, rid, app_date, department, expertise, status, position)
cid references candidate(cid)
rid references recruiter(rid)

- aid: int, primary key
- cid: int, foreign key
- rid: int, foreign key
- app_date: dateandtime

Brandon Vo

- department:  varchar(20), not null
- expertise:  varchar(15), not null
- status:  varchar(20), not null
- position:  varchar(20), not null

Brandon Vo

(c) Write SQL statements for the following questions or updates. If you cannot answer a query using your schema, then you have to modify your solutions in (a) and (b) appropriately.(8 marks)

(i) For a particular “in progress” application, say identified by an applicant’s name or application ID, list all interviewers that have at least one expertise that is required for the position.

```
select distinct interviewer.last_name, interviewer.first_name
from application, interviewer
where status='in progress'
and interviewer.expertise = application.expertise
```

(ii) List the three positions posted during 2021 that received the most applications.

```
select position, count(distinct application.aid) as app_count
from application
where year(appdate) = 2021
group by application.position
order by app_count desc
limit 3
```

(iii) For each reviewer, output their name, the number of interviews they did in 2021, and the average rating they gave in 2021.

- Assuming that reviewer actually means interviewer

```
select interviewer.last_name, interviewer.first_name, count(interview.i_id), avg(recommendation)
from interviewer, interview
where interview.i_id = interviewer.i_id
and year(interview.`date`) = 2021
group by interview.i_id
```

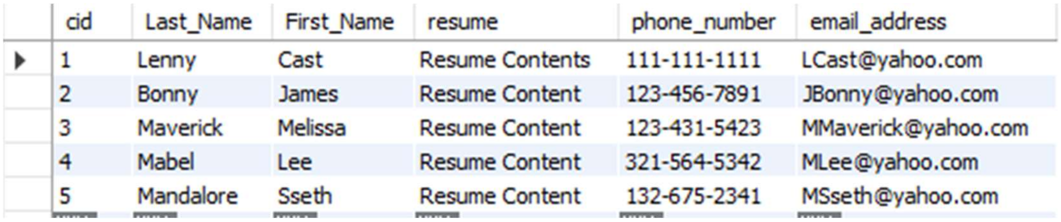
(iv) Output the names of any applicants, and the job they applied for, where an application was “in progress” for more than 100 days before being decided. (This includes both cases where current applications are still undecided after 100 days, and past applications where it took more than 100 days to decide an application.)

```
select application.aid, last_name, first_name, position
from candidate, application
where application.cid = candidate.cid
and status='in progress'
and datediff(now(), appdate) > 100
group by application.aid
```

Brandon Vo

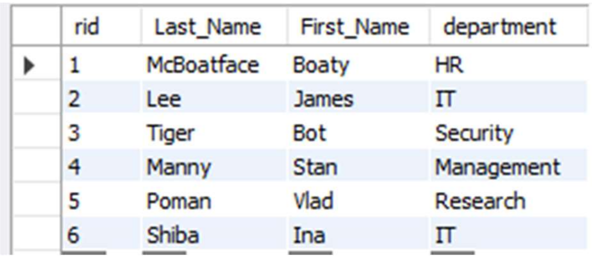
(d) Create tables in the database system, and insert some sample data (5-10 tuples per table, but choose an interesting and meaningful data set, so that queries do not output empty results). Submit screenshots of what your tables look like after you inserted the data. Then execute the queries in (c) and submit the screenshots of the queries and outputs.(5 marks)

**Candidate table**



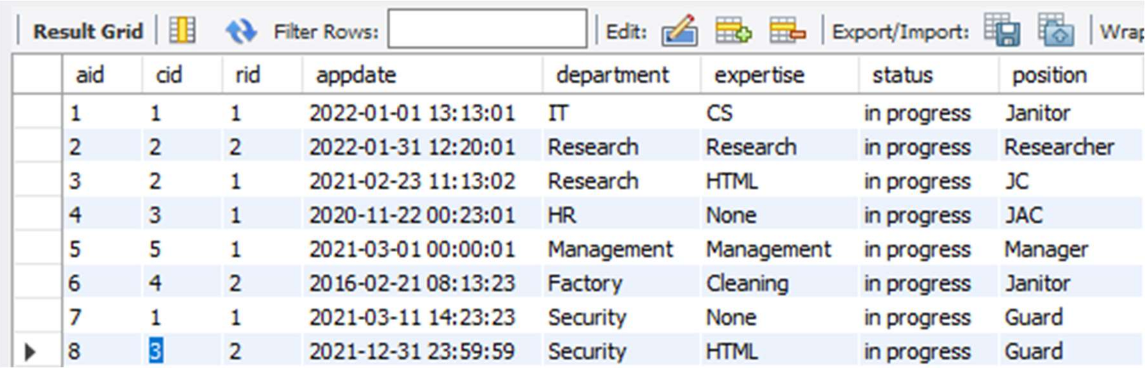
| cid | Last_Name | First_Name | resume | phone_number | email_address |
|---|---|---|---|---|---|
| 1 | Lenny | Cast | Resume Contents | 111-111-1111 | LCast@yahoo.com |
| 2 | Bonny | James | Resume Content | 123-456-7891 | JBonny@yahoo.com |
| 3 | Maverick | Melissa | Resume Content | 123-431-5423 | MMaverick@yahoo.com |
| 4 | Mabel | Lee | Resume Content | 321-564-5342 | MLee@yahoo.com |
| 5 | Mandalore | Sseth | Resume Content | 132-675-2341 | MSseth@yahoo.com |

**Recruiter table**



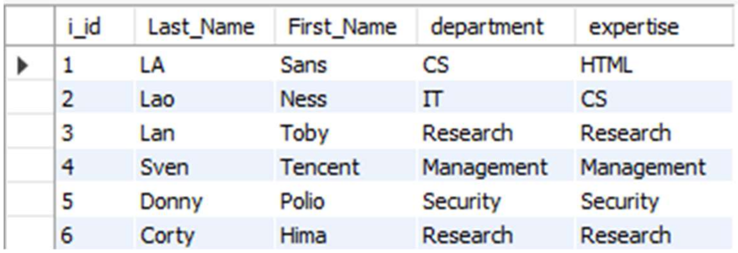
| rid | Last_Name | First_Name | department |
|---|---|---|---|
| 1 | McBoatface | Boaty | HR |
| 2 | Lee | James | IT |
| 3 | Tiger | Bot | Security |
| 4 | Manny | Stan | Management |
| 5 | Poman | Vlad | Research |
| 6 | Shiba | Ina | IT |

**Application table**



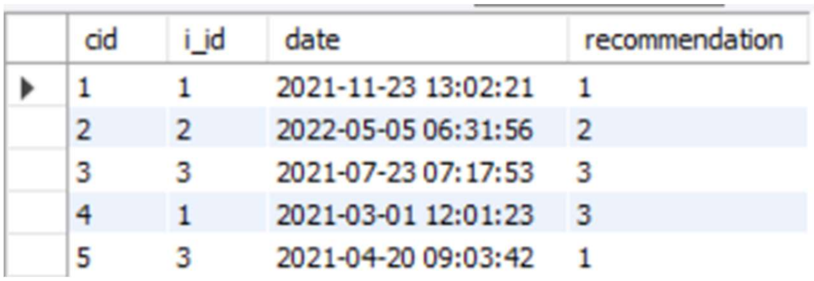
| aid | cid | rid | appdate | department | expertise | status | position |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 2022-01-01 13:13:01 | IT | CS | in progress | Janitor |
| 2 | 2 | 2 | 2022-01-31 12:20:01 | Research | Research | in progress | Researcher |
| 3 | 2 | 1 | 2021-02-23 11:13:02 | Research | HTML | in progress | JC |
| 4 | 3 | 1 | 2020-11-22 00:23:01 | HR | None | in progress | JAC |
| 5 | 5 | 1 | 2021-03-01 00:00:01 | Management | Management | in progress | Manager |
| 6 | 4 | 2 | 2016-02-21 08:13:23 | Factory | Cleaning | in progress | Janitor |
| 7 | 1 | 1 | 2021-03-11 14:23:23 | Security | None | in progress | Guard |
| 8 | 3 | 2 | 2021-12-31 23:59:59 | Security | HTML | in progress | Guard |

**Interviewer table**



| i_id | Last_Name | First_Name | department | expertise |
|---|---|---|---|---|
| 1 | LA | Sans | CS | HTML |
| 2 | Lao | Ness | IT | CS |
| 3 | Lan | Toby | Research | Research |
| 4 | Sven | Tencent | Management | Management |
| 5 | Donny | Polio | Security | Security |
| 6 | Corty | Hima | Research | Research |

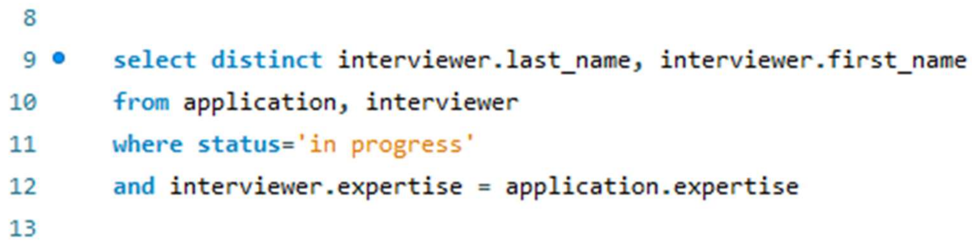
**Interview table**



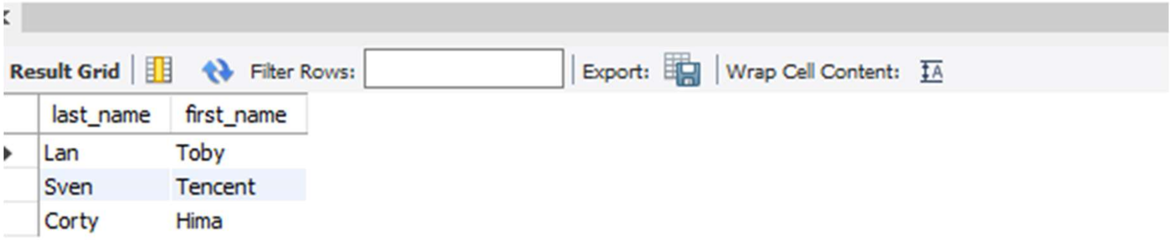
| | cid | i_id | date | recommendation |
|---|---|---|---|---|
| ▶ | 1 | 1 | 2021-11-23 13:02:21 | 1 |
| | 2 | 2 | 2022-05-05 06:31:56 | 2 |
| | 3 | 3 | 2021-07-23 07:17:53 | 3 |
| | 4 | 1 | 2021-03-01 12:01:23 | 3 |
| | 5 | 3 | 2021-04-20 09:03:42 | 1 |

**c) queries**

(i) For a particular "in progress" application, say identified by an applicant's name or application ID, list all interviewers that have at least one expertise that is required for the position.

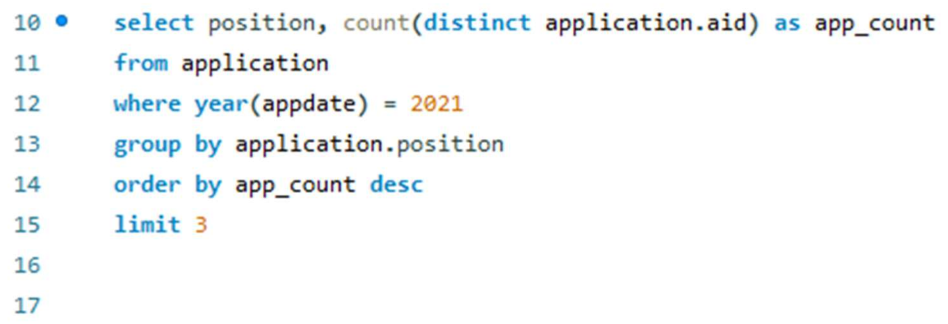


```
select distinct interviewer.last_name, interviewer.first_name
from application, interviewer
where status='in progress'
and interviewer.expertise = application.expertise
```

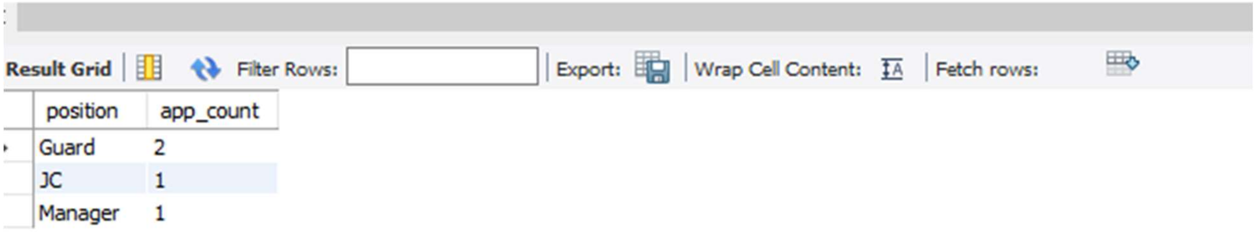


| | last_name | first_name |
|---|---|---|
| ▶ | Lan | Toby |
| | Sven | Tencent |
| | Corty | Hima |

(ii) List the three positions posted during 2021 that received the most applications.

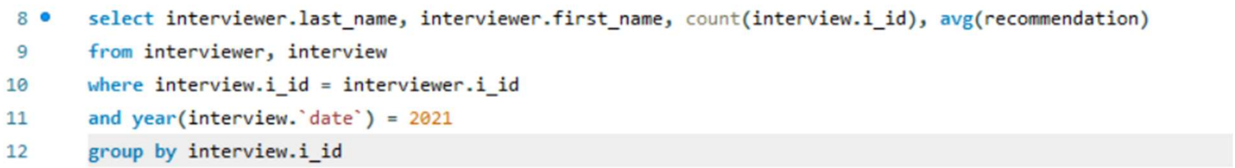


```
select position, count(distinct application.aid) as app_count
from application
where year(appdate) = 2021
group by application.position
order by app_count desc
limit 3
```

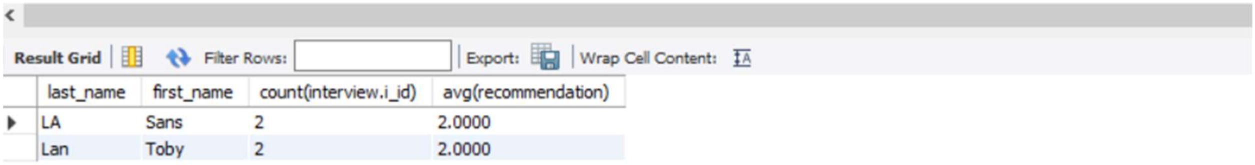


| position | app_count |
|---|---|
| Guard | 2 |
| JC | 1 |
| Manager | 1 |

(iii) For each reviewer, output their name, the number of interviews they did in 2021, and the average rating they gave in 2021.

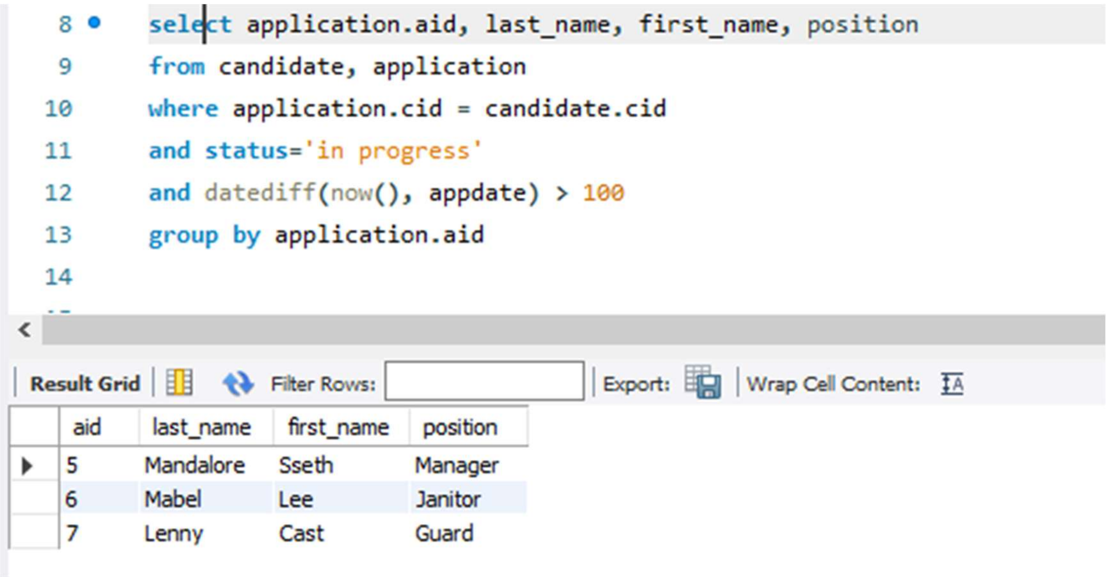


```
select interviewer.last_name, interviewer.first_name, count(interview.i_id), avg(recommendation)
from interviewer, interview
where interview.i_id = interviewer.i_id
and year(interview.`date`) = 2021
group by interview.i_id
```



| last_name | first_name | count(interview.i_id) | avg(recommendation) |
|---|---|---|---|
| LA | Sans | 2 | 2.0000 |
| Lan | Toby | 2 | 2.0000 |

(iv) Output the names of any applicants, and the job they applied for, where an application was "in progress" for more than 100 days before being decided. (This includes both cases where current applications are still undecided after 100 days, and past applications where it took more than 100 days to decide an application.)



```
select application.aid, last_name, first_name, position
from candidate, application
where application.cid = candidate.cid
and status='in progress'
and datediff(now(), appdate) > 100
group by application.aid
```

| aid | last_name | first_name | position |
|---|---|---|---|
| 5 | Mandalore | Sseth | Manager |
| 6 | Mabel | Lee | Janitor |
| 7 | Lenny | Cast | Guard |

(e) It was stated in the description of the problem that each applicant can only apply for one position at a time, and that applicants cannot apply for more than three positions in any 12-month period. Discuss how you would enforce such a requirement in your system. Would you do this via integrity constraints? Or using a trigger? Or have the recruiter manually check this and if needed reject additional applications? Justify your answer.(5 marks)

Integrity constraints only define what each attribute can or cannot be. While it is p possible to have a recruiter manually check and reject every additional application, that person cannot be reasonably expected to keep count of every application submitted and compare it to any other possible applications sent by the same candidate.

A trigger can be designed such that whenever a new application is inserted, the SQL server will initialize an SQL query that counts the number of applications sent by the candidate using the same candidate ID to see how many applications that person has sent. The SQL subquery will peruse through the list of applications sent by the matching applicant.

This subquery can use a loop that, for every tuple found, it will check for any other applications such that datediff(application_1.appdate-application_2.appdate) > 365. In addition, for every submitted application found, the trigger will check if application_1.position = application_2.position. If two positions from two different applications match, then the insert query is automatically rejected.

If this if condition is fulfilled, then both applications will be counted. If the total count > 3, then the application insert is automatically rejected. Otherwise, the application will compare the next set of applications and their dates to see if they fit within the 12 month time-span.

If the total counting of applications < 3 for all applications of the same candidate, then the application is inserted into the application table.