Question 1. [6 marks]

Consider this schema:

Brillig(tove, gyre, gimble)
Slithy(mimsy, rath)
TumTum(vorpal, jubjub)

Is mimsy a foreign key? Circle one: Yes  No.

Explain:
It is merely a referential integrity constraint. To be more
special a f.k. constant, it must reference a key, but
tove alone is not a key.

Is tove a foreign key? Circle one: Yes  No.

Explain:
The relationship goes the wrong way. Every mimsy in Slithy is
a tove in Brillig. For tove to be a f.k., every tove would have to occur in
some other table.

Suppose relation Brillig has 4 tuples. Could Slithy have 3 tuples? Circle one: Yes  No.

If yes, give an instance of the database that demonstrates this. If no, explain why not.

<table>
<thead>
<tr>
<th>Brillig</th>
<th>Slithy</th>
</tr>
</thead>
<tbody>
<tr>
<td>tove</td>
<td>mimsy</td>
</tr>
<tr>
<td>gyre</td>
<td>rath</td>
</tr>
<tr>
<td>gimble</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

Suppose relation Slithy has 4 tuples. Could Brillig have 3 tuples? Circle one: Yes  No.

If yes, give an instance of the database that demonstrates this. If no, explain why not.

Every mimsy must be different, since it is a key.
So there are 4 different mimsy's.
Each one must appear as a tove, so there must be
at least 4 tuples in Brillig to hold the 4 tove values.
Question 2.  [8 marks]

Recall this schema from Assignment 1. (Attributes card and expiry have been removed from relation User.)

Relations

User(username, lastName, firstName, email)
A tuple in this relation represents a user.

Venue(vID, name, city, address, owner)
A venue, such as Massey Hall.

Seat(sID, name, venue, accessible)
A seat for which a ticket can be sold.

Event(eID, name, type, when, venue)
An event for which tickets can be sold, such as a concert, or play.

Ticket(eID, sID, price)
A single ticket for an event.

Purchase(eID, sID, username, when)
A ticket was purchased for a seat at an event.

Integrity constraints

Seat[venue] ⊆ Venue[vID]
Event[venue] ⊆ Venue[vID]
Ticket[eID] ⊆ Event[eID]
Ticket[sID] ⊆ Seat[sID]
Purchase[eID, sID] ⊆ Ticket[eID, sID]
Purchas[username] ⊆ User[username]
Π_{type \in \{"concert", "play", "musical", "talk"\}} (\exists Ticket.sID = Seat.sID \land Event.venue = Seat.venue \land (Event \bowtie Ticket \times Seat) = \emptyset)

Write a query in relational algebra to find the second-last event at a venue called “The El Macombo”. Report the event name and type. Use only the basic operators Π, σ, ×, ⋈, ∪, −, ρ, and assignment.

Solution:

Elmo (eid, name, type, when):= Π_{eid, Event.name, \sigma_{venue = vID \land type = \emptyset \land venue, name = "The El Macombo"}} (Event \times Venue)

NotLast (eID):= Π_{E1, E1, when \leq E2, when} (P_{E1, Elmo} \times P_{E2, Elmo})

NotSecondLast (eID):= Π_{N1, eID, \sigma_{N1, when \leq N2, when}} (P_{N1, NotLast} \times P_{N2, NotLast})

SecondLast (eID) := NotLast - NotSecondLast

Answer (name, type):= Π_{name, type} (SecondLast \times Elmo)
Question 3. [6 marks]

Consider the following definitions, and the integrity constraint that they lead up to.

\[
One(\text{who, eID, sID, when}) := \\
\Pi_{\text{username, purchase.eID, sID, Event.when}}\sigma_{\text{Purchase.eID = Event.eID}} \land \text{name = 'Adele'}(\text{Purchase} \times \text{Event})
\]

\[
Two(\text{who, eID, when}) := \\
\Pi_{T1.\text{who, T1.eID, T1.when}}\sigma_{\text{T1.eID = T2.eID}} \land \text{T1.sID} \neq \text{T2.sID} \land \text{T1.who = T2.who}(\rho_{T1.\text{One}} \times \rho_{T2.\text{One}})
\]

\[
Three(\text{who}) := \\
\Pi_{\text{One.who}}\sigma_{\text{One.who = Two.who}} \land \text{One.when < Two.when}(\text{One} \times \text{Two})
\]

\[Three = \emptyset\]

Part (a) [4 marks]

Below, make an instance of the database that violates this constraint. Draw only the relevant relations: Purchase and Event. Draw each as a table with at most 5 rows. You can leave columns empty if their values are irrelevant.

<table>
<thead>
<tr>
<th>Purchase</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>eID</td>
<td>sID</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>5</td>
</tr>
</tbody>
</table>

Part (b) [2 marks]

Describe this integrity constraint in English.

A user who has bought a ticket to an Adele event cannot buy >1 ticket to a later Adele event.
Question 4. [6 marks]

Recall the schema from Question 1.

Relations

User(username, lastName, firstName, email)
A tuple in this relation represents a user.

Venue(vID, name, city, address, owner)
A venue, such as Massey Hall.

Seat(sID, name, venue, accessible)
A seat for which a ticket can be sold.

Event(eID, name, type, when, venue)
An event for which tickets can be sold, such as a concert, or play.

Ticket(eID, sID, price)
A single ticket for an event.

Purchase(eID, sID, username, when)
A ticket was purchased for a seat at an event.

Integrity constraints

Seat[venue] ⊆ Venue[vID]
Event[venue] ⊆ Venue[vID]
Ticket[eID] ⊆ Event[eID]
Ticket[sID] ⊆ Seat[sID]
Purchase[eID, sID] ⊆ Ticket[eID, sID]
Purchase[username] ⊆ User[username]

\( \Pi_{\text{type Event}} \subseteq \{ \text{"concert", "play", "musical", "talk"} \} \)

\( \neg \exists \text{ Ticket}.sID = \text{Seat}.sID \land \text{Event}.venue = \text{Seat}.venue \)

\( \neg (\text{Event} \bowtie \text{Ticket}) \times \text{Seat} = \emptyset \)

Write a query in SQL to find the events for which more than 1,000 tickets have been sold. Report the event ID and the average price for tickets that were sold.

Solution:

-- A natural join would also work.

```
SELECT eID, avg(price)
FROM Purchase p
JOIN Ticket t ON p.eID = t.eID AND p.sID = t.sID
GROUP BY eID
HAVING count(*) > 1000
```
Question 5. [4 marks]

Suppose we have the following tables:

One:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>(8 rows)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Two:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>(5 rows)</td>
<td></td>
</tr>
</tbody>
</table>

Three:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>hello</td>
<td>its</td>
</tr>
<tr>
<td>me</td>
<td>I</td>
</tr>
<tr>
<td>was</td>
<td>wondering</td>
</tr>
<tr>
<td>hello</td>
<td>can</td>
</tr>
<tr>
<td>you</td>
<td>hear</td>
</tr>
<tr>
<td>the</td>
<td>world</td>
</tr>
<tr>
<td>(6 rows)</td>
<td></td>
</tr>
</tbody>
</table>

Four:

<table>
<thead>
<tr>
<th>x</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>hello</td>
<td>from</td>
</tr>
<tr>
<td>the</td>
<td>other</td>
</tr>
<tr>
<td>hello</td>
<td>from</td>
</tr>
<tr>
<td>the</td>
<td>outside</td>
</tr>
<tr>
<td>tell</td>
<td>you</td>
</tr>
<tr>
<td>(5 rows)</td>
<td></td>
</tr>
</tbody>
</table>

Show the output of each of the following queries:

Solutions

csc343h-dianeh=> (SELECT x, y FROM One)
csc343h-dianeh=> EXCEPT ALL
csc343h-dianeh=> (SELECT * FROM Two);

```
x | y
---|---
1 | 3
1 | 3
6 | 11
4 | 7
0 | 3
(5 rows)
```

csc343h-dianeh=> (SELECT x FROM One)
csc343h-dianeh=> UNION
csc343h-dianeh=> (SELECT x FROM Two);

```
x
---
0
6
5
1
4
(5 rows)
```

csc343h-dianeh=> SELECT count(*)
csc343h-dianeh=> FROM Three NATURAL JOIN Four;

```
count
-----
(10 rows)
```
csc343h-diane\texttt{h}\rightarrow \texttt{SELECT} \texttt{count(*)}
csc343h-diane\texttt{h}\rightarrow \texttt{FROM} \texttt{Three} \texttt{NATURAL LEFT OUTER JOIN} \texttt{Four};
count

9
(1 row)