Tom Hills
CS565: Databases and Scripting Languages
Homework 3

1) 5.8:

a. $\prod_{\text{hotelNo}} (\sigma_{price > 50}\ (Room))$:

This projection will produce a **result relation** with a single column, hotelNo, based on the predicate relation, $\sigma_{price > 50}\ (Room)$. This predicate relation contains the tuples from Room relation whose price attribute is greater than $50.00. Since projections do not show duplicates, the **result relation** shows all hotels (once!) who have rooms that cost more than $50.00.

c. $\prod_{\text{hotelName}}$ (Hotel $\bowtie_{\text{Hotel.hotelNo = Room.hotelNo}} (\sigma_{price > 50}\ (Room))$):

This projection will produce a **result relation** with a single column, hotelName, based on the join relation: Hotel $\bowtie_{\text{Hotel.hotelNo = Room.hotelNo}} (\sigma_{price > 50}\ (Room))$. This join relation contains tuples from the cartesian product, $Hotel\ X\ (\sigma_{price > 50}\ (Room))$, where the hotelNo from each operand relation is equal. The **result relation** is the same as in part a, except the result relation from part a has been joined with the Hotel relation which allows the single column to be the hotelName instead of the hotelNo.

f. $\prod_{\text{guestName, hotelNo}}$ (Booking $\bowtie_{\text{Booking.guestNo = Guest.guestNo}}$ Guest) ÷ $\prod_{\text{hotelNo}} (\sigma_{city="London"}\ (Hotel))$

In this problem, there is one projection (i) relation being divided by another projection relation (ii) to produce the final **result relation**. It is better to discuss each projection relation independently before discussing the division result:

i. $\prod_{\text{guestName, hotelNo}}$ (Booking $\bowtie_{\text{Booking.guestNo = Guest.guestNo}}$ Guest)

This projection produces a relation with two columns, guestName and hotelNo, based on the join relation: Booking $\bowtie_{\text{Booking.guestNo = Guest.guestNo}}$ Guest. This join relation contains tuples from the cartesian product, $Booking\ X\ Guest$, where the guestNo from each operand relation is equal. The resulting relation shows each guest that is in the Booking relation by name and the hotel they are staying by hotelNo.

ii. $\prod_{\text{hotelNo}} (\sigma_{city="London"}\ (Hotel))$

This projection produces a relation with one column, hotelNo, based on the predicate relation: $\sigma_{city="London"}\ (Hotel)$. This predicate relation contains the tuples from Hotel relation whose city attribute is “London”. Since projections do not show duplicates, the resulting relation shows all hotels located in London by hotelNo.

iii. Based on the resulting relations from (i) and (ii), the final **result relation** has a single column, guestName, which shows Guests who have stayed at <u>all</u> hotels located in London. This is because division only selects a tuple from $\prod_{\text{guestName, hotelNo}}$ (Booking $\bowtie_{\text{Booking.guestNo = Guest.guestNo}}$ Guest) when a guestName is found to be associated (i.e., in the same row) with all hotelNo values from $\prod_{\text{hotelNo}} (\sigma_{city="London"}\ (Hotel))$.

2) 5.12

b. List all single rooms with a price below 20 per night.

Relational algebra:

$\sigma_{type=\text{"single"} \wedge \ price < 20} \ (Room)$

Relational calculus:

$$\{ \ R \mid Room(R) \ \wedge (R.type = \text{"single"}) \wedge \ (R.price \ > \ 20) \ \}$$

c. List the names and addresses of all guests.

Relational algebra:

$\prod_{\text{guestName, guestAddress}} (Guest)$

Relational calculus:

$$\{ \ G.guestName, G.guestAddress \mid Guest(G) \ \}$$

d. List the price and type of all rooms at the Grosvenor Hotel.

Relational algebra:

$\prod_{\text{type, price}} (\ Room \ \bowtie_{\text{Hotel.hotelNo = Room.hotelNo}} (\sigma_{hotelName = \text{"}Grosvenor\ Hotel\text{"}} \ (Hotel)) \ )$

Relational calculus:

$$\{ \ R.price, R.type \mid Room(R) \wedge (\exists H)(\ Hotel(H) \wedge [H.hotelNo = R.hotelNo] \ \wedge [H.hotelName = \text{"Grosvenor Hotel"}] \ ) \ \}$$

e. List all guests currently staying at the Grosvenor Hotel.

Relational algebra:

$\prod_{\text{guestNo, guestName, guestAddress}}$
$(\ Guest \ \bowtie_{\text{Guest.guestNo = Booking.guestNo}}$
[
$Booking$
$\bowtie_{\text{Booking.hotelNo = Hotel.hotelNo}}$
$(\ \sigma_{hotelName = \text{"}Grosvenor\ Hotel\text{"}} \ (Hotel) \ )$
]
)

Relational calculus:

$$\{ \ G.guestName, G.guestAddress, B.guestNo \mid Guest(G) \wedge Booking(B) \wedge B.guestNo = G.guestNo \ \wedge \ (\exists H)(\ Hotel(H) \wedge [B.hotelNo = H.hotelNo] \ \wedge [H.hotelName = \text{"Grosvenor Hotel"}] \ ) \ \}$$

f. List the details of all rooms at the Grosvenor Hotel, including the name of the guest staying in the room, if the room is occupied.

Relational algebra:

NOTE: ⟕ == **left outer join**, and I'm assuming ⋈ (i.e., join without specified predicates) == **natural join**.

$$
\begin{aligned}
&\prod_{\text{roomNo, hotelNo, type, price, guestName}} \\
&(\\
&\quad [\\
&\quad\quad Room \\
&\quad\quad \bowtie_{\text{Room.hotelNo = Hotel.hotelNo}} \\
&\quad\quad (\sigma_{hotelName = \text{"}Grosvenor\ Hotel\text{"}} (Hotel)) \\
&\quad] \\
&\quad\quad \text{⟕} \\
&\quad [\\
&\quad\quad Guest \\
&\quad\quad \bowtie \\
&\quad\quad (\\
&\quad\quad\quad \sigma_{dateFrom \leq CURDATE \wedge dateTo > CURDATE} (Booking) \\
&\quad\quad\quad \bowtie \\
&\quad\quad\quad \sigma_{hotelName = \text{"}Grosvenor\ Hotel\text{"}} (Hotel) \\
&\quad\quad) \\
&\quad] \\
&)
\end{aligned}
$$

Relational calculus:

NOTE: Writing the ***selection*** part so that everything is a free variable. I'm assuming the question is asking for all Room attributes plus the guestName from Guest. The wording is somewhat vague so this should be a valid assumption.

$$
\begin{aligned}
&\{ \\
&\quad B.roomNo, H.hotelNo, R.type, R.price, G.guestName \mid \\
&\quad (\\
&\quad\quad (Room(R) \wedge Hotel(H) \\
&\quad\quad \wedge [R.hotelNo = H.hotelNo] \\
&\quad\quad \wedge [H.hotelName = \text{"Grosvenor Hotel"}]) \\
&\quad\quad \vee \\
&\quad\quad (Guest(G) \wedge Booking(B) \wedge Hotel(H) \\
&\quad\quad \wedge [H.hotelNo = B.hotelNo] \\
&\quad\quad \wedge [B.guestNo = G.guestNo] \\
&\quad\quad \wedge [H.hotelName = \text{"Grosvenor Hotel"}] \\
&\quad\quad \wedge [B.dateFrom \leq CURDATE] \\
&\quad\quad \wedge [B.dateTo > CURDATE]) \\
&\quad) \\
&\}
\end{aligned}
$$

3) 5.10

a. Creates a relation with a single column, hotelName, from Hotel relation where the city in which the hotel resides is London.

b. Creates a relation with a single column, hotelName, from Hotel relation where the hotel has rooms with a price that is greater than $50.00.

c. Creates a relation with a single column, hotelName, from Hotel relation where the hotel, at some point, will have or had a guest named “John Smith”. This assumption is based on the idea that the Booking relation doesn’t just contain current guests, but also guests from the past and future guests. From previous homework, this should be a valid assumption.

d. Creates a relation with columns hotelName, guestName, dateFrom, dateFrom from Hotel, Guest, Booking, and Booking relations. Since the two “dateFrom” columns are based on the cartesian product between two versions of the Booking relation AND the “dateFrom” values can’t be equal, you get a **result relation** that represents guests who have stayed at a particular hotel more than once. I say more than once because if the guest only stayed one time at a particular hotel, then the “B2.dateFrom != B1.dateFrom” condition would result in not showing that single instance.

Tom Hills
CS565: Databases and Scripting Languages
Homework 3

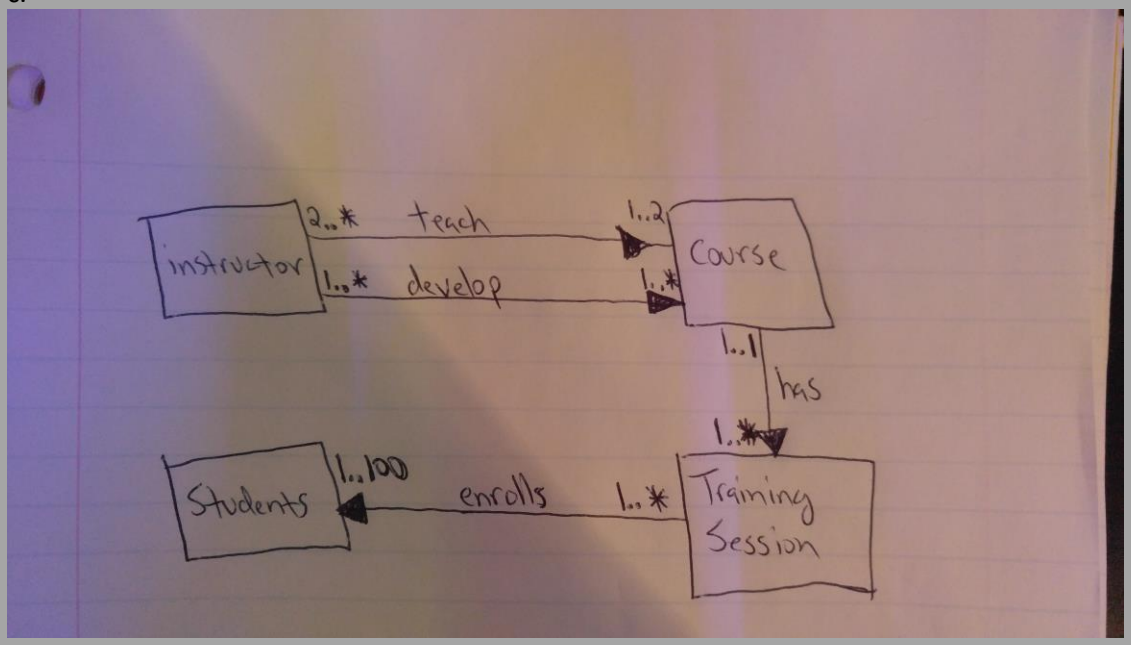
4)

a. Instructor, Course, TrainingSession, Students

b. Assumptions:
- A course could be spread across multiple training sessions.
- If an instructor is assigned to teach, then they teach either 1 or 2 classes.
- A new course in development can have more than 1 instructor assigned to help develop it.

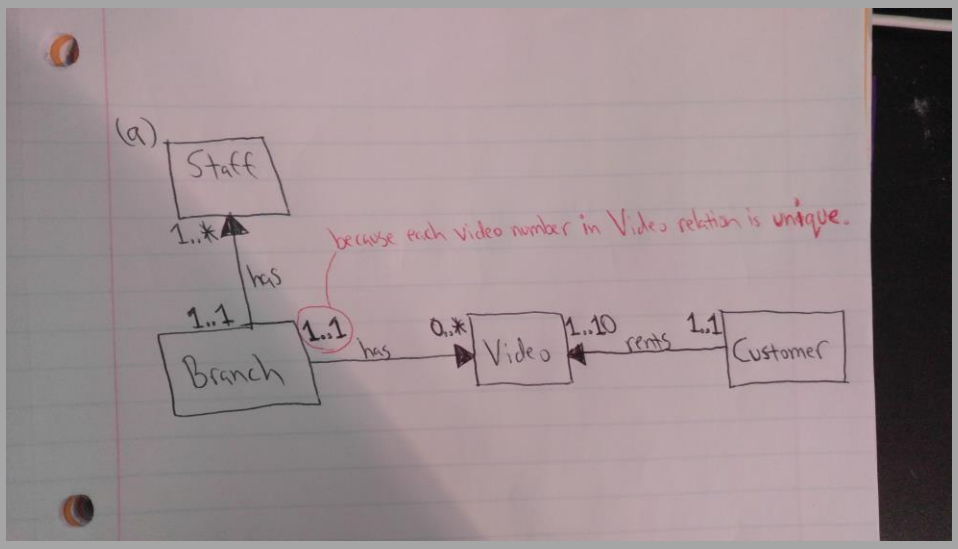
Relationships:
- Training session (1..*) -> enrolls -> (1..100)student
- Instructors(2..*) -> teach -> (1..2)Courses
- Instructors(1..*) -> develop -> (1..*)Courses
- Course(1..1) -> has -> (1..*)Training sessions

c.



5)

a. Assumptions:
- A member of Staff is only assigned to 1 Branch (i.e., they can't work at multiple branches simultaneously)
- A Branch could possibly have only 1 member of Staff: the manager.
- The "names of the main actors" for Video is contained in one attribute that jams their names into one long string, comma separated.
- The statement, "each branch **may** have one or more copies of a video title" means that a branch could potentially have zero copies of a video title.
- "video number" of an individual copy of a video title (i.e., videoCopyNo) are unique throughout all branches. This makes sense if you think about a video having a single number stamped on the outside. Nothing in the description prevents this assumption. All it says is: "Each branch may have one or more copies of a video title. Each copy has a rental status, and the individual copies are identified using the video number." Thus, all copies, regardless of title, have a unique video number.



b. Please see attributes under Entity types: **Branch, Staff, Video, Customer**

yellow == foreign keys, red == primary keys, orange == both primary AND foreign key, green == entity/base relation, blue == relationship relation											
BASE RELATIONS	**Branch**	**Staff**	**Video**	**Customer**				~~Actor~~			
ATTRIBUTES	branchNo	staffNo	videoCopyNo	customerNo				~~catalogNo~~			
	manager (staffNo)	name	catalogNo	branchNo				~~actorName~~			
	Street	position	title	firstName				decided all actor names			
	City	salary	category	lastName				could be put into one string :)			
	State		daily rental fee	address							
	Zip Code		purchase cost	registerDate							
	phone		actors								
			director								
RELATIONSHIPS	Branch -> Has -> 1 or more Video copies		Customer -> Rents -> 1 to 10 Videos simulataneously		Branch -> Has -> 1 or more Staff			"1 or more" assumes the manager can be the only Staff member at a branch			
ATTRIBUTES	videoCopyNo		rentalNo								
	branchNo		customerNo								
	rental status		videoCopyNo								
			firstName								
			lastName								
			title								
			daily rental fee								
			dateRented								
			dateReturned								

c.

i. Branch
   - Primary key: branchNo
   - Candidate keys:
     1) branchNo
     2) street, city, zip code
     3) phone

ii. Staff
   - Primary key: staffNo
   - Candidate keys:
     1) staffNo

iii. Video
   - Primary key: videoCopyNo
   - Candidate keys:
     1) videoCopyNo

iv. Customer
   - Primary key: customNo
   - Candidate keys:
     1) customNo
     2) firstName, lastName, address