Quiz Submissions - Quizzterm # 2 Objective

Top of Form

**Submission View**

Your quiz has been submitted successfully.

|  |  |  |
| --- | --- | --- |
| **Question 1** |  | 1 / 1 point |

We can say the following about the library loan many to many example that we did in class (check all that apply):

|  |  |  |
| --- | --- | --- |
|  | It is a many to many **with**history because a given customer might borrow the same book again. | |
|  | We **could** have made it into a many to many with**out** history by creating a separate class, called Transaction, that had the customer's ID and the date/time of the loan, and then made a many to many between Transaction and Book on Shelf instead of Customer and Book on Shelf. | |
|  | There is no way to recreate this into a many to many with**out** history without introducing redundancy into the model. | |
|  | This had to be a many to many **with** history because we used the call number rather than the ISBN in the primary key of the loans class (the junction table for this many to many). | |
| **Question 2** |  | 1 / 1 point | |

The full outer join and the cross product return the same results.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | True | |
|  |  | False | |
| **Question 3** | |  | 1 / 1 point | |

In the Relational Algebra expressions that we have done so far, we generally (when reading from left to right) start with the π{"version":"1.1","math":"<math xmlns="http://www.w3.org/1998/Math/MathML"><mi>&#x3C0;</mi></math>"} operator, then the δ{"version":"1.1","math":"<math xmlns="http://www.w3.org/1998/Math/MathML"><mi>&#x3B4;</mi></math>"} operator because:

|  |  |  |
| --- | --- | --- |
|  | It is impossible for the pi operator to perform its work until after the sigma has filtered the tuples in the relation. | |
|  | The sigma often needs to have access to one or more columns that the pi operator **ex**cludes from its resulting relation. | |
|  | It's strictly a convenient convention. | |
|  | We actually do not use these operators in this order.  Putting the pi operator first is never a good idea. | |
| **Question 4** |  | 1 / 1 point | |

A given book can have several authors participate in the book.  A given author can participate in several books.  A single author could participate in a single book in several different roles.  Therefore, this would have to be a many to many with history.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | True | |
|  |  | False | |
| **Question 5** | |  | 1 / 1 point | |

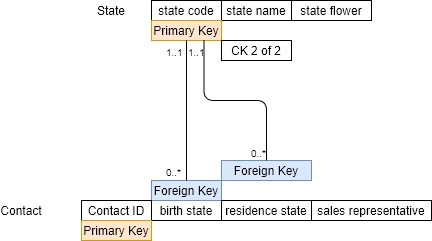
You want to model Hollywood marriages.  The gender of the two people involved is not at issue.  The same pair can get married, then divorced, then married again.  To model such an association, you would need a many to many with history.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | True | |
|  |  | False | |
| **Question 6** | |  | 1 / 1 point | |

If someone were to show you just the relation scheme diagram of a many to many association, there would be no way for you to tell whether that was a many to many **with** history or a many to many **without** history.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | True | |
|  |  | False | |
| **Question 7** | |  | 1 / 1 point | |

Consider the following relation scheme diagram:



what can you say about this model?

|  |  |  |
| --- | --- | --- |
|  | This is a many to many without history.  The State class is both parents. | |
|  | The state class performs two different roles for the contact: one as the state in which they were born, and the other is the state in which they currently live. | |
|  | This model is clearly wrong because of the key structure of the contact class. | |
|  | This is a many to many**with** history with the State class fulfilling the role of both parents. | |
| **Question 8** |  | 1 / 1 point | |

You can**not** role name the foreign keys that migrate into the junction table of a many to many without history because that would change the primary key.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | True | |
|  |  | False | |
| **Question 9** | |  | 1 / 1 point | |

The SQL equivalent of the following Relational Algebra statement:

πvOwnerLastName, vOwnerFirstNameδvColor='blue'Automobile{"version":"1.1","math":"<math xmlns="http://www.w3.org/1998/Math/MathML"><msub><mi>&#x3C0;</mi><mrow><mi>v</mi><mi>O</mi><mi>w</mi><mi>n</mi><mi>e</mi><mi>r</mi><mi>L</mi><mi>a</mi><mi>s</mi><mi>t</mi><mi>N</mi><mi>a</mi><mi>m</mi><mi>e</mi><mo>,</mo><mo>&#xA0;</mo><mi>v</mi><mi>O</mi><mi>w</mi><mi>n</mi><mi>e</mi><mi>r</mi><mi>F</mi><mi>i</mi><mi>r</mi><mi>s</mi><mi>t</mi><mi>N</mi><mi>a</mi><mi>m</mi><mi>e</mi></mrow></msub><msub><mi>&#x3B4;</mi><mrow><mi>v</mi><mi>C</mi><mi>o</mi><mi>l</mi><mi>o</mi><mi>r</mi><mo>=</mo><mo>'</mo><mi>b</mi><mi>l</mi><mi>u</mi><mi>e</mi><mo>'</mo></mrow></msub><mi>A</mi><mi>u</mi><mi>t</mi><mi>o</mi><mi>m</mi><mi>o</mi><mi>b</mi><mi>i</mi><mi>l</mi><mi>e</mi></math>"}

is:

|  |  |
| --- | --- |
|  | select \* from Automobile where vColor = 'blue'; |
|  | select vOwnerFirstName from Automobile where vColor = 'blue'; |
|  | select vOwnerLastName, vOwnerFirstName from Automobile where vColor = 'blue' order by vOwnerLastName, vOwnerFirstName; |
|  | select vOwnerLastName, vOwnerFirstName from Automobile where vColor = 'blue'; |
| **Question 10** | |  | 1 / 1 point |

Consider the following DDL:

CREATE TABLE quizzes(  
name VARCHAR(20) NOT NULL,  
description VARCHAR(200) NOT NULL,  
max\_points integer NOT NULL,  
CONSTRAINT quizzes\_pk PRIMARY KEY (name)  
);  
  
CREATE TABLE students(  
student\_ID INT NOT NULL,  
first\_name VARCHAR(80) NOT NULL,  
last\_name VARCHAR(80) NOT NULL,  
CONSTRAINT students\_pk PRIMARY KEY (student\_ID)  
);  
  
CREATE TABLE scores (  
quiz\_name VARCHAR(20) NOT NULL,  
id\_of\_student\_taking\_quiz INT NOT NULL,  
score int NOT NULL,  
CONSTRAINT scores\_pk PRIMARY KEY (quiz\_name, id\_of\_student\_taking\_quiz, score),  
CONSTRAINT scores\_quizzes FOREIGN KEY (quiz\_name)  
 REFERENCES quizzes(name),  
CONSTRAINT scores\_students FOREIGN KEY (id\_of\_student\_taking\_quiz)  
 REFERENCES students (student\_ID)  
);

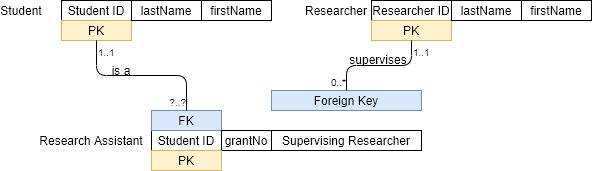
Which of the following is true?

|  |  |
| --- | --- |
|  | This is a good example of a many to many without history between Students and Quizzes. |
|  | This is a many to many with history between Quizzes and Students because the primary key of scores has the score column in it. |
|  | This is a botched attempt at a many to many without history because there is no reason to put the score of the quiz into the primary key of scores.  That would allow the same student to take the same quiz over and over again, as long as they earned a different score each time. |
|  | The CREATE TABLE statement for scores will fail because of a syntax error. |
| **Question 11** | |  | 1 / 1 point |

Normally, in a many to many association, the association class has exactly two parents: the participants in the many to many association.  But the association class can also be involved in other associations.  Select all that apply:

|  |  |
| --- | --- |
|  | The association class can, itself, be one of the two parents in another many to many association. |
|  | The association class can, itself, be a parent in a one to many association. |
|  | The association class can be the child to yet a third class as long as that association is not identifying (that is, the primary key of the parent does **not** migrate into the primary key of the child). |
|  | The association class can be the child in an identifying relationship besides the two parents in the many to many. |
| **Question 12** | |  | 1 / 1 point |

Consider the following relation scheme diagram:



For every instance of Student, we can have how many instances of Research Assistant:

|  |  |
| --- | --- |
|  | One and only one. |
|  | Never fewer than one. |
|  | It is impossible to tell, given this model. |
|  | No fewer than zero, no greater than one. |
| **Question 13** | |  | 1 / 1 point |

The join types are 1) natural join, 2) join using, 3) join on, 4) left outer join, 5) right outer join, 6) full outer join, 7) cross product.  Please order the following Relational Algebra join symbols in the proper order to match these join types.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | \_\_3\_\_ |  | r ⋈a=b s{"version":"1.1","math":"<math xmlns="http://www.w3.org/1998/Math/MathML"><mi>r</mi><mo>&#xA0;</mo><msub><mo>&#x22C8;</mo><mrow><mi>a</mi><mo>=</mo><mi>b</mi></mrow></msub><mo>&#xA0;</mo><mi>s</mi></math>"} | |
|  | \_\_4\_\_ |  | r =⋈ s{"version":"1.1","math":"<math xmlns="http://www.w3.org/1998/Math/MathML"><mi>r</mi><mo>&#xA0;</mo><mo>=</mo><mo>&#x22C8;</mo><mo>&#xA0;</mo><mi>s</mi></math>"} | |
|  | \_\_6\_\_ |  | r =⋈= s{"version":"1.1","math":"<math xmlns="http://www.w3.org/1998/Math/MathML"><mi>r</mi><mo>&#xA0;</mo><mo>=</mo><mo>&#x22C8;</mo><mo>=</mo><mo>&#xA0;</mo><mi>s</mi></math>"} | |
|  | \_\_7\_\_ |  | r X s{"version":"1.1","math":"<math xmlns="http://www.w3.org/1998/Math/MathML"><mi>r</mi><mo>&#xA0;</mo><mi>X</mi><mo>&#xA0;</mo><mi>s</mi></math>"} | |
|  | \_\_1\_\_ |  | r ⋈s{"version":"1.1","math":"<math xmlns="http://www.w3.org/1998/Math/MathML"><mi>r</mi><mo>&#xA0;</mo><mo>&#x22C8;</mo><mi>s</mi></math>"} | |
|  | \_\_5\_\_ |  | r ⋈= s{"version":"1.1","math":"<math xmlns="http://www.w3.org/1998/Math/MathML"><mi>r</mi><mo>&#xA0;</mo><mo>&#x22C8;</mo><mo>=</mo><mo>&#xA0;</mo><mi>s</mi></math>"} | |
|  | \_\_2\_\_ |  | r ⋈a,b s{"version":"1.1","math":"<math xmlns="http://www.w3.org/1998/Math/MathML"><mi>r</mi><mo>&#xA0;</mo><msub><mo>&#x22C8;</mo><mrow><mi>a</mi><mo>,</mo><mi>b</mi></mrow></msub><mo>&#xA0;</mo><mi>s</mi></math>"} | |
| **Question 14** | | |  | 0 / 1 point |

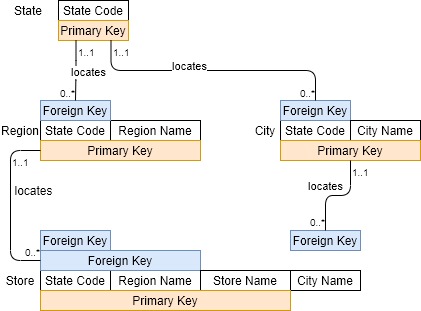
The order entry database that we looked at when we first started talking about many to many was a many to many with**out** history because (check all that apply):

|  |  |
| --- | --- |
| Correct Answer | The primary key of OrderLine has all of the columns that migrated in from Order and Product, and nothing else. |
| Incorrect Response | We could easily have done it as a many to many **with** history by removing the Order class, put the date of the order into OrderLine.  But we had not discussed many to many with history by then. |
| Correct Answer | There is never a case in which we would record a given Product in a given Order more than once.  If the customer orders, say, 3 16 oz. framing hammers, we simply put a quantity of 3 into OrderLine for that particular product in that particular order. |
| Incorrect Response | We don't care about the timing of the orders, so it's really **not** a many to many with history. |
| Correct AnswerIncorrect Response | No matter how many products the customer ordered in a given order, there would be just one sales person, and we wanted to record who that sales person was just once for the entire order. |
| **Question 15** | |  | 1 / 1 point |

We want to report out all of the employees, the location of their office and the customer that they support as sales rep even if they are not sales representative for **any** customer.   Select the query that will not return all of the rows needed.  Note, they all run.

|  |  |  |
| --- | --- | --- |
|  | select  offices.COUNTRY, offices.TERRITORY, offices."STATE",          employees.LASTNAME, employees.FIRSTNAME, customers.CUSTOMERNAME from    customers right outer join employees on          customers.SALESREPEMPLOYEENUMBER = employees.EMPLOYEENUMBER         inner join offices on employees.OFFICECODE = offices.OFFICECODE order by employees.LASTNAME, employees.FIRSTNAME, customers.CUSTOMERNAME; | |
|  | select  offices.COUNTRY, offices.TERRITORY, offices."STATE",          employees.LASTNAME, employees.FIRSTNAME, customers.CUSTOMERNAME from    offices inner join          (employees inner join customers on          employees.EMPLOYEENUMBER = customers.SALESREPEMPLOYEENUMBER)         using (officeCode) order by employees.LASTNAME, employees.FIRSTNAME, customers.CUSTOMERNAME; | |
|  | select  offices.COUNTRY, offices.TERRITORY, offices."STATE",          employees.LASTNAME, employees.FIRSTNAME, customers.CUSTOMERNAME from    offices inner join employees on offices.OFFICECODE = employees.OFFICECODE         left outer join customers on          employees.EMPLOYEENUMBER = customers.SALESREPEMPLOYEENUMBER order by employees.LASTNAME, employees.FIRSTNAME, customers.CUSTOMERNAME; | |
|  | select  offices.COUNTRY, offices.TERRITORY, offices."STATE",          employees.LASTNAME, employees.FIRSTNAME, customers.CUSTOMERNAME from    (customers right outer join employees on          customers.SALESREPEMPLOYEENUMBER = employees.EMPLOYEENUMBER)         inner join offices on employees.OFFICECODE = offices.OFFICECODE order by employees.LASTNAME, employees.FIRSTNAME, customers.CUSTOMERNAME; | |
|  | select  offices.COUNTRY, offices.TERRITORY, offices."STATE",          employees.LASTNAME, employees.FIRSTNAME, customers.CUSTOMERNAME from    customers right outer join (offices inner join employees on          offices.OFFICECODE = employees.OFFICECODE)         on employees.EMPLOYEENUMBER = customers.SALESREPEMPLOYEENUMBER order by employees.LASTNAME, employees.FIRSTNAME, customers.CUSTOMERNAME; | |
| **Question 16** |  | 0.5 / 1 point | |

The following model is a simplification of the *Cascading Keys* homework assignment:



The State Code migrates into Store from two different classes: Region and City.  Since there is only one column for State Code in Store, that means that those two values of State Code must agree.  That, in turn, requires that the city that the store is located in must be in the same state as the Region that the store is located in.

Imagine that we give Region a surrogate ID.  Which of the following (select all that apply) is true?

|  |  |
| --- | --- |
|  | We can just make State Code part of the primary key along with the RegionID, and thus retain the colliding foreign key. |
| Correct AnswerIncorrect Response | Ideally, we would want some 3rd generation language code that would impose the constraint that the state that the city belonged to was the same state that the Region belongs to. |
| Correct Answer | The State Code will no longer migrate into Store from Region, which means that we will lose the colliding foreign key, and with it, the ability to require that the Region and the City that are the parents to Store come from the same state. |
| Correct Answer | The Region relation scheme needs a candidate key of {State Code, Region Name} because that set of attributes is still unique, even with the addition of the surrogate for Region. |
|  |  |
|  | |

|  |
| --- |
| **Attempt Score:**  14.5 / 16 - 90.63 % |
| **Overall Grade** (highest attempt)**:**  14.5 / 16 - 90.63 % |

|  |
| --- |
|  |

Done

|  |
| --- |
|  |

Bottom of Form