HQ_2_1

The figure shows an instance of a given relation RR. Which of the given schemas is valid for this instance of RR?

✓ Game (name, publisher, releaseDate, developer, platform, rating, genre)

- Only this option’s keys can be satisfied by the given data. All other options will violate one or more key constraint because there will be two different rows with the same values for the values associated with the attributes of the key for those options.

✗ Game (name, publisher, releaseDate, platform)

- The key constraint suggested by the selected keys cannot be satisfied by the data.

✗ Game (name, publisher, releaseDate, platform,)

- The key constraint suggested by the selected keys cannot be satisfied by the data.

✗ Game (name, publisher, releaseDate,)

- The key constraint suggested by the selected keys cannot be satisfied by the data.

✗ Game (name, publisher, releaseDate, developer, platform, rating, genre)

- The key constraint suggested by the selected keys cannot be satisfied by the data.

✗ Game (name, publisher, releaseDate, developer, platform, rating, genre)

- The key constraint suggested by the selected keys cannot be satisfied by the data.

HQ_2_2

You need to store the following information in a object relational database: In an **Author** relation, you need to store his unique ID, name, books that he has written, and publishers he has worked with. The publishers in **Author** is a nested relation with country and publisher as its attributes. Each row in this nested relation corresponds to the unique publisher for the author's book in that country. The publisher attribute has a type that references a row in the **Publisher** relation and the attribute books has a type that is a set of references to rows in the **Book** relation. The **Publisher** relation stores a unique pID and a name (not necessarily unique) for each publisher.
The **Book** relation stores the unique ISBN for the book, name, year published and publisher. Note that an ISBN is associated with only one publisher. Select the **object relational schema** corresponding to the above description.

✓

Author (ID, name, books({*Book}), publisher(country, publisher(*Publisher).Publisher)), Publisher(pID, name), Book(ISBN, name, yearPublished, publisher(*Publisher).Publisher)

- From the textbook, "If an attribute has a type that is a reference to a single tuple with a relation schema named R, we show the attribute A in a schema as A(*R).R; … Similarly, if an attribute A has a type that is a set of references to tuples of schema R, then A will be shown in a schema as A({*R})." The books attribute in Author has type that is a set references to tuples of Book. The publisher attribute in Author is a nested relation and it in turn has an attribute publisher which has type that is a reference to a tuple in relation Publisher. Book's publisher attribute has type that is a reference to a tuple in relation Publisher. All other answers are incorrect because their syntax is incorrect.

✗

Author (ID, name, books(*Book), publisher(country, publisher(*Publisher).Publisher)), Publisher(pID, name), Book(ISBN, name, yearPublished, publisher(*Publisher).Publisher)

✗

Author (ID, name, books({*Book}), publisher(country, publisher(*Publisher).Publisher)), Publisher (pID, name), Book (ISBN, name, yearPublished, publisher({*Publisher}))

✗

Author (ID, name, books({*Book}), publisher(country, publisher({*Publisher}))), Publisher (pID, name), Book (ISBN, name, yearPublished, publisher({*Publisher}))

✗

Author (ID, name, books, country, publisher), Publisher(pID, name), Book(ISBN, name, yearPublished, publisher)

HQ_2_3

Since a relation is a **set** of tuples, how these tuples are ordered does not matter. Furthermore, the attributes can be reordered without causing any loss of information. An instance of a relation is equivalent to another instance if it contains the permuted rows and/or columns of the other. How many equivalent representations are there for the relation in the given figure?

✓ 16! 7!

- There are 16 rows and 7 columns. Each permutation of row and/or column in a table is equivalent to another. Total 16!7! ways to permute rows and cols.

✗ 16!

- This does not include cases when also permuting columns of table.

✗ 17!

- Only one possible answer. See explanation for correct answer above.

✗ 17!8!

- There are only 16 rows and 7 columns of data. We do not include the header row.

✗ 2

- Only one possible answer. See explanation for correct answer above.

HQ_2_4

Select the relational database schema corresponding to the ER design in the figure. If possible, combine the relation corresponding to the relationship with the relation of one of the entities.

✓ E1 (A1, A2, A3, A4) , E2 (A1, A2, A3, A5, A6, A7, A8, A9, A10, A13) , E3 (A9, A10, A11, A12)

- Each entity has its own relation. The relation for the relationship R contains the keys of all three entity sets and its own attribute A13. This relation can be merged into the relation for E2 as E2 is on the "many" side of the relationship. Merging relation for R with the other relations will introduce redundancy.

✗ E1 (A1, A2, A3, A4) , E2 (A5, A6, A7, A8) , E3 (A9, A10, A11, A12) , R (A1, A2, A3, A5, A6, A9, A10)

- Relation for R can be merged with that of E2 and its missing attribute A13.

✗ E1 (A1, A2, A3, A4) , E2 (A5, A6, A7, A8, A13) , E3 (A9, A10, A11, A12)

- The relation for R should have the keys of all three entity sets and A13. After merging it into E2 the relation is missing the keys of E1 and E3.

✗ E1 (A1, A2, A3, A4) , E2 (A5, A6, A7, A8) , E3 (A1, A2, A3, A5, A6, A9, A10, A11, A12, A13)

- R can only be merged into E2 as it is the only one on the "many" side.

HQ_2_5

Select the relational database schema corresponding to the ER design in the figure. If possible, combine the relation for the relationship with the relation of one of the entities.

✓ E1 (A1, A2, A3, role1_A4, role2_A4, A7), E2 (A4, A5, A6)

- Each role introduces an attribute corresponding to the key of E2 suitably renamed to the relation R which also has attribute A7. Relation R can be merged into E1 because E1 is on the “many” side of the relationship.

✗ E1 (A1, A2, A3), E2 (A4, A5, A6)

✗ E1 (A1, A2, A3), E2 (A4, A5, A6), R (A1, A2, A4, A7)

✗ E1 (A1, A2, A3, A4, A7), E2 (A4, A5, A6)

HQ_2_6

Select the relational database schema corresponding to the ER design in the figure **using the ER approach**. Note that an employee must either be a permanent staff or a temp and cannot be both. That is, the **subclasses Permanent and Temp are disjoint**.

Simplify the database schema by merging appropriate relations if it will not result in any loss of information.

✓ PermanentEmployee (sID, name, salary, accessRights, insurance, stock), TempEmployee (sID, name, salary, accessRights, contractPeriod)

- Start off with one relation for each entity set. The relationship between Employee and Permanent is 1-1. Same for Employee and Temp. Since all employees belong either to Permanent of Temp, every row in Employee is matched to exactly one row in Permanent or Temp. The information in those rows can be moved into Permanent or Temp and Employee is no longer needed. Note that the key of relation corresponding to a subclass is always the key of the root entity set. In this exercise we asked for merging explicitly. This is possible because ISA relationship is 1-1 and because of the added condition that all employees are either Permanent or Temp and cannot be both.

✗ Employee (sID, name, salary, accessRights), Permanent (sID, insurance, stock), Temp (sID, contractPeriod)

✗ Employee (sID, name, salary, accessRights), Permanent (sID, insurance, stock), Temp (sID, contractPeriod)

✗ Employee (sID, name, salary, accessRights), Permanent (sID, insurance, stock), Temp (sID, contractPeriod), IsPermanentEmployee(sID, insurance, stock), isTempEmployee (sID, contractPeriod)

HQ_2_7

Select the relational database schema corresponding to the ER design in the figure **using the object-oriented approach**.

It is possible for an entity to belong to multiple subclasses. In this case, **someone who is both a permanent and temp employee is a "probationary employee".** Probationary employees are hired as a permanent staff but has a probation period (contractPeriod from the temp subclass) during which their contracts can be terminated for poor performance or if they are ill-suited for the job. Note that an employee must belong to exactly one of the three categories: permanent employee, temp employee, or a "probationary employee".

Simplify the database schema by merging appropriate relations if it will not result in any loss of information.

✓ PermanentEmployee (sID, name, salary, accessRights, insurance, stock), TempEmployee (sID, name, salary, accessRights, contractPeriod), ProbationaryEmployee (sID, name, salary, accessRights, insurance, stock, contractPeriod)

- Under the OO approach, we should first end up with four relations - Employee, PermanentEmployee, TempEmployee, and ProbationaryEmployee. Employee is an empty table as all employees must be exactly one of the other three and so can be discarded.

✗ PermanentEmployee (sID, name, salary, accessRights, insurance, stock), TempEmployee (sID, name, salary, accessRights, contractPeriod)

✗ Employee (sID, name, salary, accessRights), PermanentEmployee (sID, name, salary, accessRights, insurance, stock), TempEmployee (sID, name, salary, accessRights, contractPeriod)

✗ Employee (sID, name, salary, accessRights), Permanent (sID, insurance, stock), Temp (sID, contractPeriod)

✗ Employee (sID, name, salary, accessRights), Permanent (sID, insurance, stock), Temp (sID, contractPeriod), IsPermanentEmployee(sID, insurance, stock), isTempEmployee (sID, contractPeriod)

✗ Employee (sID, name, salary, accessRights), Permanent (sID, insurance, stock), Temp (sID, contractPeriod)

✗ Employee (sID, name, salary, accessRights), Permanent (sID, insurance, stock), Temp (sID, contractPeriod), ProbationaryEmployee (sID, name, salary, accessRights, insurance, stock, contractPeriod)

✗ Employee (sID, name, salary, accessRights), PermanentEmployee (sID, name, salary, accessRights, insurance, stock), TempEmployee (sID, name, salary, accessRights, contractPeriod)

HQ_2_8

Which of the following is **NOT** valid JSON data?

✓ { “Title” : “Fifa 2017”, “Platform” : { “PlayStation”, “Windows”, “Mac OS X 10.12” }, “Genre” : “Sports” }

- Objects in JSON must be field-value pairs so the value of “Platform” is invalid. It should have been a list [“PlayStation”, “Windows”, “Mac OS X 10.12”].

✗ { “UserID” : 1235455, “Friends”: [], “Name”: “John Doe”, “Devices” : [“iPhone”, “iPad”, “Windows 8”] }

- Empty list is allowed. Everything else is pretty standard.

✗ { “UserID” : 1235455, “Friends”: [34135, 13556, 20344], “Name”: “John Doe”, “Devices” : [“iPhone”, “iPad”, “Windows”], “Albums” : {“CameraRoll” : [“378721.jpg”, “243535.jpg”], “Family” : [] } }

- Empty list is allowed. Everything else is pretty standard.

✗ []

- Empty list is allowed.

✕ [{}]

- Empty object is allowed.

✕ [“John Doe”]

✕ {}

- Empty object is allowed.

✕ {“Accounts” : 12356 }

- Just an object with one field-value pair.

✕ [false]

- false is a valid value. This is just a list of one value.

HQ_2_9

Convert the model specified by the UML diagram to a relation schema. Combine appropriate relations for the relationships if it does not result in any loss of information or introduce any redundancy.

✓ Book (<u>ISBN</u>, name, publisher, ISBNofOriginal), Writer (<u>ID</u>, name), WrittenBy (<u>ISBN</u>, <u>ID</u>)

- Each class converts to one relation containing the attributes of the class. Primary key for each class is specified in the diagram and they become primary key of relation. Associations are analogous to relationships so they convert to a relation as well containing the keys of the classes connected to it along with any attributes of its own (none here). The association of Book to itself denoting the sequel relationship becomes a table as well containing the key for the original book and the sequel (similar situation as the case for ER modelling). However, this is a many-one relationship and so can be merged (required by this question) into the Book relation so the key for the original (ISBNofOriginal) is added to the Book relation. Adding ISBNofSequel to book is wrong as an “original” book can have many sequels so the relationship from original to sequel is one-many. Trying to add ISBNofSequel will introduce one row for every original and its sequel and the information of the original will be repeated (a form of redundancy).

✕ Book (ISBN, name, publisher), Writer (ID, name), Sequels (ISBNofOriginal, ISBNofSequel), WrittenBy (ISBN,ID)

- The association that results in Sequels is a many-one from Book to itself. Sequels should be merged into Book as required by the question.

✕ Book (ISBN, ISBNofSequel, name, publisher), Writer (ID, name), WrittenBy (ISBN, ID,)

- A sequel is not unique. Merging the relation for the association between Book and itself into Book this way will result in rows containing duplicate information about the book. ISBNofOriginal should be stored in Book instead.

✕ Book (ISBN, ISBNofSequel, name, publisher), Writer (ID, ISBNofBook, name)

- Missing relation for the Written-By association. Also, the way the sequel association is merged into Book is incorrect. See correct answer for explanation.

✕ Book (ISBN, name, publisher), Writer (ID, ISBNofBook, name), Sequels (ISBNofOriginal, ISBNofSequel)

- Missing relation for the Written-By association. Sequels should be merged into Book. See correct answer for explanation.

✕ Book (ISBN, name, publisher), Writer (ID, name)

- Missing information from the two associations.

HQ_2_10

Scenario 1:
You need to store a large collection of images. Each image has tags indicating the objects in that image. The most common query is to retrieve all images associated with a given set of labels.

Scenario 2:
You run an online discussion forum BufferOverflow for programmers. Your system needs to keep track of registered users and allow them to post questions, answers and comments. The posts can be tagged with topics. Some common queries are: retrieve all posts containing a set of keywords, retrieve posts related to a given set of topics within a time period, and retrieve all posts written by a particular user.

Scenario 3:
You are tasked with building the backend of a social network. This requires you to construct a knowledge base, which is essentially a collection of interconnected data. The system must be able to discover products that are similar to those “liked” by a user or people close to her (i.e., her family, friends, or friends of friends etc.) so that the system can display relevant advertisements to the user.

Which type of database, relational, key-value, or graph, is the most appropriate for scenarios 1, 2 and 3?

✓ 1: Key-value, 2: Relational, 3: Graph

- Scenario 1: Query is simple - retrieve values (images, or more specifically, image filenames / URLs) given keys which are the labels tagged to each image. For a key-value DB such as Redis, images can be represented as URLs/filenames in a sorted set associated with a tag. Intersect sets of images to get images containing a set of tags. A relational DB is not well-suited for this, especially if the user wants to look up images that has a large set of specified tags as that might require many table joins. Graph DB is not suited for this as there are practically no significant relationships to be modelled here. Scenario 2: This is a typical discussion forum application. Each entity has a fixed set of attributes, which means the schema is fixed and can be determined right from the start. The queries are also fairly complicated and requires use of various attributes from various entities. Relational DB is good for such use cases but key-value DB are not because of the complicated queries. Scenario 3: The social network in this case can be well modelled using a graph. Entities are represented as nodes and relationships as edges. Graph DB is suitable for this as it makes it easy to discover relationships between entities by examining relationships between them. Relational DB is not well suited for this. The information for various entities will be spread across many tables. Same for the relationships. This means any query to find relationships between entities and entities not directly related to them will require several complicated operations involving many joins. Queries will also be complex and unwieldy. Key-value

DB is not good for this case as it does not model connections between entities well.

✗ 1: Relational, 2: Key-value, 3: Graph

✗ 1: Key-value, 2: Key-value, 3: Graph

✗ 1: Relational, 2: Key-value, 3: Key-value

✗ 1: Relational, 2: Graph, 3: Key-value

✗ 1: Graph, 2: Key-value, 3: relational
