Consider the following schemas:
students(matric, sname)
workings(pid, matric, since)
projects(pid, pname)
category(pid, cname)

The underlined attributes are the primary key of the schema.
You may assume that there can never be any `NULL` values in the instance.
No other constraints can be assumed.
NOTE: We will be using this schema for Questions 1 to 5.

Refer back to schema in Page 1.

Question 1

Complete the relational algebra expression below to find the pair of different students' matric number (m1,m2) working on the same project, excluding students not working on any project.

$$\pi([\_A\_], \sigma(\_B\_ \wedge \_C\_, (\rho(w1(p1, m1, s1), workings \times \rho(w2(p2, m2, s2), workings))))$$

Answer:

$$A : m1, m2$$
$$B : p1 = p2$$
$$C : m1! = m2$$

Question 2

Complete the relational algebra expression below to find the oldest projects pid in the database. The oldest project is the project with the smallest value of since attribute. You may compare the since attributes with standard relational operators (e.g., <, >, <=, >=, ==, !=).
$\pi([\_A\_], \pi([\_B\_, \_C\_], workings) - \pi([p2, s2], \sigma(\_D\_ > \_E\_, (\rho(w1(p1, m1, s1), workings \times \rho(w2(p2, m2, s2), workings))))))$
Answer:

$$A : pid$$
$$B : pid$$
$$C : since$$
$$D : s2$$
$$E : s1$$

Explanation We can split the RA queries into the followering to understand the thought process behind the entire RA expression:

1. $Q_1 = \sigma(s2 > s1, (\rho(w1(p1, m1, s1), workings \times \rho(w2(p2, m2, s2), workings))))$ retrieves the pairs of workings where $s2$ is of a later timing than $s1$, as $s2$'s value is greater than $s1$.

2. $Q_2 = \pi([p2, s2], Q_1)$ retrieves the workings tuples which are not the smallest value, aka not the oldest value.

3. $Q_3 = \pi([pid, since], workings) - Q_2$ retrieves the workings tuples which have the smallest value

4. $Q_4 = \pi([pid], Q_3)$ retrieves the pid with the smallest value of the since attribute

Refer back to schema in Page 1.

Question 3

Complete the SQL create table statement below to create the schema above. The first table (i.e., students table) has been filled in for you.

```
CREATE TABLE students (
  matric VARCHAR(9) PRIMARY KEY,
  sname VARCHAR(50)
);
CREATE TABLE projects (
  _1_ VARCHAR(9) _2_ ,
  _3_ VARCHAR(50)
);
CREATE TABLE workings (
  _4_ VARCHAR(9) REFERENCES _5_ ,
  _6_ VARCHAR(9) REFERENCES _7_ ,
  _8_ DATE,
  PRIMARY KEY(pid, matric)
);
CREATE TABLE category (
  _9_ VARCHAR(9) REFERENCES _10_ ,
  _11_ VARCHAR(9),
  PRIMARY KEY(pid, cname)
);
```

Answer:

1. pid
2. PRIMARY KEY
3. pname
4. pid
5. projects
6. matric
7. students
8. since
9. pid
10. projects
11. cname

Refer back to schema in Page 1.

Question 4

Complete the SQL query below to find all pair of distinct projects' pid (p1, p2) such that the two projects have exactly the same set of categories.

```
SELECT P1.pid, P2.pid
 FROM projects P1, projects P2
 WHERE _D_
   AND ( SELECT COUNT(*)
     FROM ( SELECT _A_ FROM category C0 WHERE _B_
       _E_
       SELECT _A_  FROM category C0 WHERE _C_ ) AS T1 )
 =
 ( SELECT COUNT(*)
   FROM ( SELECT _A_ FROM category C0 WHERE _B_
     __F__
     SELECT _A_ FROM category C0 WHERE _C_ ) AS T2 )
```

Answer:

```
A: C0.cname
B: C0.pid = P1.pid
C: C0.pid = P2.pid
D: P1.pid <> P2.pid
E: INTERSECT
F: UNION
```

Explanation: We know that two sets are equivalent by set theory **iff**

$$|p1 \cup p2| = |p1 \cap p2|$$

From this, let's try to analyze what the subquery does:

1. ```
   SELECT COUNT(*)
     FROM ( SELECT C0.cname FROM category C0 WHERE C0.pid = P1.pid
       UNION
       SELECT C0.cname FROM category C0 WHERE C0.pid = P2.pid) AS T2
   ```
   retrieves the count of $|p1.pid \cup p2.pid|$

2. ```
   SELECT COUNT(*)
     FROM ( SELECT C0.cname FROM category C0 WHERE C0.pid = P1.pid
       INTERSECT
       SELECT C0.cname FROM category C0 WHERE C0.pid = P2.pid) AS T2
   ```
   retrieves the count of $|p1.pid \cap p2.pid|$

3. As a result, by checking if the counts are equivalent, we can determine whether these two projects have the same set of categories using their intersection and union counts.

Refer back to schema in Page 1.

Question 5

Consider the following result for the following SQL query

```
SELECT *
 FROM students NATURAL JOIN workings
 NATURAL JOIN projects
 NATURAL JOIN category;
```

matric	sname	pid	since	cname
A0001	AA	P01	2002	CA
A0001	AA	P01	2002	CB
A0001	AA	P02	2004	CB
A0002	BB	P01	2003	CA
A0002	BB	P01	2003	CB
A0003	CC	P03	2004	CA
A0003	CC	P03	2004	CC
A0003	CC	P03	2004	CD
A0004	AA	P03	2004	CA
A0004	AA	P03	2004	CC
A0004	AA	P03	2004	CD

You are further told that the result of running the SQL query on Question 4 is:

pid	pid
P01	P04
P04	P01
P03	P05
P05	P03

What is the result of the following SQL query?

```
SELECT PID FROM category
EXCEPT ALL
SELECT DISTINCT pid FROM category WHERE cname != 'CA';
```

If you have fewer than 6 rows, write the answer as -. Leaving your answer as blank constitutes not answering the question.
Exclude quote symbols in your answer. For instance, write CS2102 insted of ’CS2102’ if that is your answer.
Answers in next page...

Answer:

pid
P01
P03
P03
P04
P05
P05

Explanation:
From the above, we can that `P01, P02, P03` all are present without dangling tuples.
However, from the query result in Q4, we can see an additional two pid, `P04, P05`.
From the join result in Q5, we can the distinct tuples wrt `(pid, cname)`
However, from the result of Q4, we can also tell that P01 has the same set of categories as P04 and P03 has the same set of categories as P05.

pid	cname
P01	CA
P01	CB
P02	CB
P03	CA
P03	CC
P03	CD
P04	CA
P04	CB
P05	CA
P05	CC
P05	CD

As a result, by retrieving `DISTINCT pid` tuples where `cname != CA`, we have that each and every pid has a category **NOT** `CA` and as a result, the final result is what we have above
And as a result, our final result is
1. 1 `P01` tuple
2. 0 `P02` tuples
3. 2 `P03` tuples
4. 1 `P04` tuple
5. 2 `P05` tuples

Consider the schema $R(A, B, C, D, E)$ with $F = \{BDE \rightarrow AE, AD \rightarrow C, CD \rightarrow E, BE \rightarrow AD\}$.

Question 6

Write four completely non-trivial functional dependencies that are implied by F but not in F. If you have fewer than 4 functional dependencies, write the answer as -. Leaving your answer as blank constitutes not answering the question.
Answer:

1. $AD \rightarrow E$
2. $BE \rightarrow C$
3. $BDE \rightarrow C$
4. $BDE \rightarrow A$

Explanation:
Using attribute closure, one can simply find the missing completely non-trivial functional dependencies that are implied by F but not in F.

1. $AD^+ = ACDE$
2. $CD^+ = CDE$
3. $BE^+ = ABCDE$ (superkey)
4. $BDE^+ = ABCDE$ (superkey)

Note that there are other possible completely non-trivial FDs. They are listed as follows:
1. $BE \rightarrow A$ (Decomposition)
2. $BE \rightarrow D$ (Decomposition)

Question 7

Write at most four superkeys of R with at most 3 attributes. If you have fewer than 4 superkeys, write the answer as -. Leaving your answer as blank constitutes not answering the question.
Answer:

1. BE
2. ABE
3. BCE
4. BDE

Explanation:
Given from the explanation in Question 6, we can tell that BE is already a superkey.
Hence, by adding any other attribute to BE, you should be able to get a superkey with at most 3 attributes.
Note that there are two other superkeys not mentioned, ABD and BCD, which I will discuss in Question 8.

Consider the schema $R(A, B, C, D, E)$ with $F = \{BDE \rightarrow AE, AD \rightarrow C, CD \rightarrow E, BE \rightarrow AD\}$.

Question 8

Write all keys of R with at most 3 attributes. If you have fewer than 4 keys, write the answer as -.
Answer:

1. BE
2. ABD
3. BCD
4. $-$

Explanation:
If you would like to determine other minimal superkeys, B must be definitely a prime attribute as B does not appear in the RHS of the FDs in F.

$$
\begin{gathered}
AB^{=} = AB \\
BC^{+} = BC \\
BD^{+} = BD \\
BE^{+} = ABCDE \\
ABC^{+} = ABC \\
ABD^{+} = ABCDE \\
ABE^{+} = ABCDE \\
BCD^{+} = ABCDE \\
BCE^{+} = ABCDE \\
BDE^{+} = ABCDE
\end{gathered}
$$

Consider the schema $R(A, B, C, D, E)$ with $F = \{BDE \rightarrow AE, AD \rightarrow C, CD \rightarrow E, BE \rightarrow AD\}$.

Question 9

Write one possible lossless-join valid BCNF decomposition of R into exactly three fragments such that each fragment has exactly three attributes

Answer:
$R1(C, D, E)$
$R2(A, C, D)$
$R3(A, B, D)$

Explanation:
Using the more but not all property, $AD \rightarrow C$ violates BCNF as $AD^{=} = ACDE$
Split R into $R0(A, C, D, E)$ and $R3(A, B, D)$ without attributes CE
$R3$ is in BCNF as $F3 = \varnothing$
$R0$ violates BCNF as $F0 = \{AD \rightarrow C, CD \rightarrow E\}$ as $CD \rightarrow E$ violates BCNF as $CD^{+} = CDE$
Split $R0$ into $R1(C, D, E)$ and $R2(A, C, D)$
$R1$ is in BCNF as $F1 = \{CD \rightarrow E\}$
$R2$ is in BCNF as $F2 = \{AD \rightarrow C\}$
Hence, final answer is: $R1(C, D, E), R2(A, C, D), R3(A, B, D)$

Question 10

Is your decomposition in Question 9 a dependency-preserving decomposition?

Answer: NO
Explanation: we can see that $BE \rightarrow AD$ is not preserved. And in fac t, it is impossible to produce a dependency-preserving BCNF decomposition such that there must be 3 relations with 3 attributes each. More will be discussed in the 3NF synthesis questions.

Consider the schema $R(A, B, C, D, E)$ with $F = \{BDE \rightarrow AE, AD \rightarrow C, CD \rightarrow E, BE \rightarrow AD\}$.

Question 11

Write one possible minimal cover of F. Write your answer in the same commaseparated format as above, but without the curly brackets.

Answer: $\{AD \rightarrow C, CD \rightarrow E, BE \rightarrow AD\}$
Explanation:
Seperating into Completely Non-Trivial FD:
$F = \{BDE \rightarrow A, AD \rightarrow C, CD \rightarrow E, BE \rightarrow A, BE \rightarrow D\}$.

Attribute Redundancy:
$D \in BDE \rightarrow A$ is redundant as $D^+ = D \wedge BE^+ = ABCDE$
Can remove $BE \rightarrow A$ as duplicate FD. Hence, $F^- = \{AD \rightarrow C, CD \rightarrow E, BE \rightarrow A, BE \rightarrow D\}$.

FD Redundancy:
None of the FDs are acutally redundant, this is because
By removing $AD \rightarrow C$, $AD^+ = AD$ instead of $AD^+ = ACDE$
By removing $CD \rightarrow E$, $CD^+ = CD$ instead of $CD^+ = CDE$
By removing $BE \rightarrow A$, $BE^+ = BDE$ instead of $BD^+ = ABCDE$
By removing $BE \rightarrow D$, $BE^+ = ABE$ instead of $BE^+ = ABCDE$

Question 12

Write one possible lossless-join dependency-preserving valid decomposition of R that has exactly three fragments. Write the attribute of each fragment as comma-separated value as above. Each blank is a comma-separated single attribute in uppercase.

Answer:
$R1(A, C, D)$
$R2(C, D, E)$
$R3(A, B, D, E)$
Explanation:
Given R3 contains keys BE and ABD, there is no need to add an additional table containing a key.

TIP: it would be easier to do the 3NF synthesis questions first to determine if it is possible to get a BCNF dependency-preserving decomposition :)

Question 13

We know that Armstrong's axioms are both sound and complete. Consider the augmentation rule in Armstrong's axioms:

$$\text{If } A \rightarrow B, \text{ then } AC \rightarrow BC$$

Our aim in this question is to show that the "reverse" of the augmentation rule is not sound. By the "reverse", we meant the following rule:

$$\text{If } AC \rightarrow BC, \text{ then } A \rightarrow B$$

To show that it is unsound, we want to imply the following:

$$\{\} \vDash A \rightarrow B$$

In other words, any functional dependencies are implied if we allow "reverse" augmentation rule. Note that you are not allowed to use the decomposition and union rule from extended Armstrong's axioms. The following example shows the use of Armstrong's axioms in a proof:
1. A → B [Given]
2. A → C [Given]
3. AB → A [Reflexivity]
4. A → AB [Augmentation (1) with A]
5. AB → BC [Augmentation (2) with B]
6. A → BC [Transitivity (4) and (5)]

To use reverse augmentation:
1. AC → BC [Given]
2. A → B [Remove C from (1)]

Your proof should not exceed 3 lines. If you have fewer than 3 lines, write the answer as -. Leaving your answer as blank constitutes not answering the question.

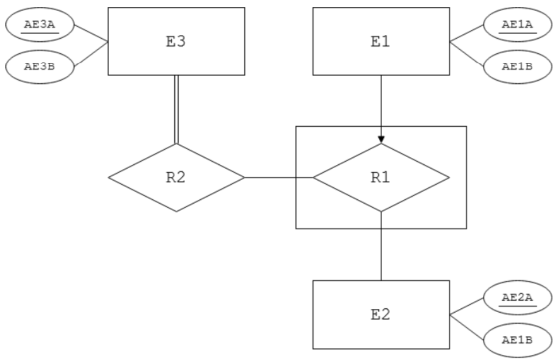
Answer:
1. AAB → AB [Reflexivity]
2. AAB → ABB [Augmentation (1) with B]
3. A → B [Remove AB from (2)]

Explanation:
Note that in order to get the final result of $A \rightarrow B$, we must have something in the form of $AX \rightarrow BX$ before hand, where X is a set of attributes.

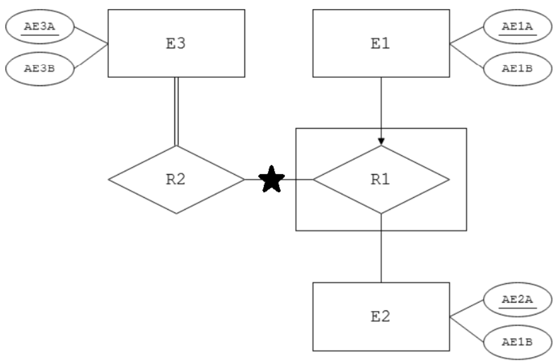
There are mutliple ways to go around this, so long as you get the state of $AX \rightarrow BX$

Question 14

Find the mistake in the ER diagram below. If the mistake is on a line, make sure your pin is on the line outside of any box/diamond/oval. If the mistake is in the arrow make sure your pin is on the arrow. If the mistake is on the box/diamond/oval, make sure your pin is inside them without touching any other lines.



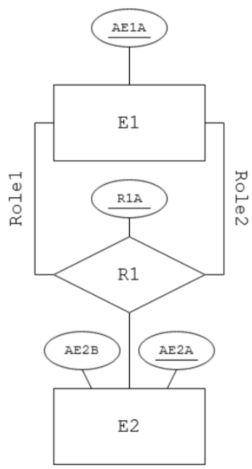
Answer:



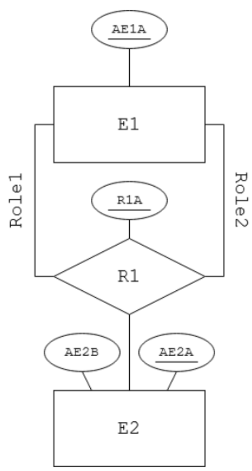
Explanation:
Can a relationship be connected to another relationship?

Question 15

Find the mistake in the ER diagram below. If the mistake is on a line, make sure your pin is on the line outside of any box/diamond/oval. If the mistake is in the arrow make sure your pin is on the arrow. If the mistake is on the box/diamond/oval, make sure your pin is inside them without touching any other lines.



Answer:



There is no error.
Explanation:
There is no error.
Note that in previous years, key attributes were not allowed on relationship sets. However, for this semester key attributes are allowed once again.