2020 – 2021 Databases

**Question 1**

**a)**

**D**

1. **This is probs wrong**

Select employee-name

FROM employee JOIN manages on employee.employee-name = manages.employee-name

WHERE manger-name = employee-name

city = (SELECT city-

FROM employee JOIN manages on employee.employee-name = manages.employee-name

WHERE employee-name IN manager-name)

AND

street = (SELECT street

FROM employee JOIN manages on employee.employee-name = manages.employee-name

WHERE employee-name IN manager-name)

Select emp.employee\_name

From manages as M join Employee as emp on M.employee-name = emp.employee-name

join Employee as manager on M.manager-name = manager.employee-name

where emp.city = manager.city and emp.street = manager.street

SELECT e.employee-name  
FROM employee AS e,  
 (SELECT m.manager - name INNER JOIN employee ON m.manager-name = employee.employee- name) AS mgrs,  
 WHERE e.city = mgrs.city AND e.street = mgrs.street

**ii.**

SELECT employee-name

FROM works

WHERE salary > all (SELECT salary FROM works WHERE company-name = ‘Bank’)

Select employee-name

From Works

Where salary >(select max(salary) from Works where company-name = ‘Bank’)

**iii.**

SELECT company-name

FROM company

EXCEPT

SELECT company-name

FROM company

WHERE city NOT IN (SELECT city FROM company WHERE company-name = ‘Bank’)

\*\*double negative -> just use IN?

Alternative, maybe? :

SELECT DISTINCT company-name

FROM Company c1

WHERE NOT EXIST (

SELECT \* FROM Company c2

WHERE c2.company-name = ‘BANK’

AND c2.city NOT IN (

SELECT c3.city FROM Company c3

WHERE c3.company-name = c1.company-name

)

My thinking is:

* In the inner query, we look for any city that Bank is in that is not in the cities that our outer (c1) company is in, and then make sure we don’t include any results (NOT EXISTS) where this is the case
* Also, the question doesn’t make sense because Company table has a PK of company-name, so couldn’t possibly be in multiple cities (the company-name column wouldn’t be unique then) ... but we’ll ignore that... in which case we’re going to have to use DISTINCT, otherwise we’re going to return multiples of company-names

Alternative:

Select company-name from (

Select count(city) as [CityCount],

Company-name

from Company

where city in (select distinct city from Company where company-name = ‘Bank’)

)

group by company-name

having CityCount = max(CityCount)

Reasoning: the inner query generates a table of all companies that operate in the same cities as ‘Bank’ alongside the count of these cities. The highest count in this will be that of Bank and any other company that operates in all of Bank’s cities. The outer query selects those companies who’s counts are the same as the max of the counts from the inner query. (doesn’t use a set operation though).

**iv.**

SELECT employee-name

FROM works

GROUP BY company-name

HAVING salary > AVG(salary)

My take:

SELECT employee-name FROM works w1

JOIN

(SELECT w2.company-name, AVG(salary) AS average\_pay

FROM works w2

GROUP BY w2.company-name)

ON w2.company-name = w1.company-name

where w1.salary > average\_pay;

Another Take:

SELECT employee\_name FROM

Works w

JOIN (SELECT company\_name, AVG(salary) AS company\_average FROM Works GROUP BY company\_name) avg

ON (w.company\_name=avg.company\_name)

WHERE w.salary>avg.company\_average

// Calculates the average salary for each company and joins it to the company name in the Works table (turns out this is just the same as above rip)

Yet another, which I think is simpler:

SELECT employee-name

FROM Works W

WHERE salary > (SELECT avg(salary) FROM Works WHERE company-name = W.company-name)

**v.**

SELECT company-name

FROM works

GROUP BY company-name

HAVING SUM(salary) <= all (SELECT SUM(salary)

FROM works

WHERE salary >0

GROUP BY company-name)

SELECT company-name

FROM Works

GROUP BY company-name

ORDER BY SUM(salary) ASC

LIMIT 1

**b)**

*Possible alternative, pls pick holes in it*

*driver(PK:ID, PhoneNo, Name)*

*truck(PK:LicNo, maxVol, maxWt)*

*trip(PK:tripNo)*

*StopPoint(PK:Address, port, openHrs) //is.a relationship does not include derrived attributes right? StopPouint can either be a port or shopnpay...*

*shipment(PK:ShipNo, FK:tripno, Vol, Weight)*

*tripno references trip.tripNo*

*journey(PK:driverid, PK:tripno, trucklicno)*

*driverid references driver.ID on delete cascade*

*tripno references trip.tripNo on delete cascade*

*trucklicno references truck.LicNo*

*between(PK:address, PK: shipno, pickup\_time, dropoff\_time)*

*address references StopPoint.Address on delete cascade*

*shipno references shipment.ShipNo on delete cascade*

1b (Alternative):

Assume: Trip records persist for tracking activity even if drivers leave or shipment is deleted

Driver(PK:ID, PhoneNo, Name)

Truck(PK:LicNo, maxVol, maxWt)

StopPoint(PK:Address)

Warehouse(PK:Address, port)

Address references StopPoint.Address on delete cascade

shopNpay(PK:Address, openHrs)

Address references StopPoint.Address on delete cascade

shipment(PK:shipNo, Vol, Weight)

Trip(PK:tripNo, PK:driver, PK:shipment, truck)

Driver references Driver.ID

Shipment references shipment.shipNo

Truck references Truck.LicNo

between(PK:Address, PK:shipment, pickup\_time, drop\_off time)

Address references StopPoint.Address on delete cascade

Shipment references shipment.shipNo on delete cascade

//requires relationships.

**Question 2**

**a)**

**i.**

|  |  |  |
| --- | --- | --- |
| **FD** | **LHS extraneous** | **RHS extraneous** |
| **1**  **A-> BCD**  **Final A->D** | none | Is B?  {ABE yes  A->CD  Is C?  {ABEC yes  A->D |
| **2**  **BCD->A**  **Final D->A** | Is B?  {CDAB yes  CD->A  Is C?  {DCA yes | none |
| **3**  **B -> CE**  **Final B->E** | none | Is C?  {BADC yes |
| **4**  **D->C** | none | none |
| **5**  **B ->A** | none | none |
| **6**  **A->BE**  **Final A->E** | none | Is B?  {ADCAB yes |
| **7**  **ABD->E**  **Final D->E** | Is A?  {BDAE yes  BD->E  Is B?  {DAE yes | none |
| **8**  **C->AB**  **C->B** | none | Is A?  {CBA yes |

**Ii.**

FDs:

A->D,

D->A,

B->E, //is this supposed to be B->E? C is extraneous when you test it first as above? This still works just doesn’t line up with the working

D->C,

B->A,

A->E,

D->E,

C->B

Canonical cover:

A->D

B->A

C->B

D->CE

Another option:

A->DBE

B->C

D->C

C->A,

Relation: R(A,B,C,D,E)

KEY: AE

3NF: R1(AD) R2(BA) R3(CB) R4(DCE) R5(AE)

**Iii.+**

BCNF

R(ABCDE) superkeys: A, BCD, B, C, D therefore in bcnf

**b)**

**i.**

|  |  |  |
| --- | --- | --- |
| **FD** | **LHS extraneous** | **RHS extranous** |
| **D-->E** | none | none |
| **CDE-->B** | Is C?  {DE no  Is D?  {CED no  Is E?  {CDE no | none |
| **BCD-->E**  **D-->E** | Is B?  {CDE yes  CD--> E  Is C?  {DE yes | none |
| **E-->D** | none | none |
| **C-->D** | none | none |
| **B-->A** | none | none |
| **AC-->D**  **C-->D** | Is A?  {CD yes | none |
| **AB-->CE**  **AB->C** | Is A?  {BA no  Is B?  no | Is C?  {ABED no  Is E?  {ABCDE yes |

**ii.**

FDs:

D-->E

CDE-->B

E-->D

C-->D

B-->A

AB-->C

Alternative: This is following the tutorial solution method (as Jonah mentioned):

**LHS:**

Fd2:

* Remove C? DE = DE. No
* Remove D? CE = CDEB Therefore Yes.
* Remove E? C = CDEB Therefore Yes

Fd3:

* Remove B? CD = CDBE therefore yes
* Remove C? D = DE therefore no
* Remove D? C = CBDE therefore yes

Fd7:

* Remove A? C = CD therefore yes

Fd8:

* Remove A? B = BACE therefore yes

**RHS:**

Fd8:

* Remove C? B = BEAD therefore no
* Remove E? B = BCE therefore yes

Final set:

**FD1: D->E**

**FD2: C->B**

**FD3: C->E**

**FD4: E->D**

**FD5: C->D**

**FD6: B->A**

**FD7: C->D**

**FD8: B->C**

**// I think we can go further with the above,**

**FD5 and FD7 are both extraneous because they are implied by FD3 and FD4, so final minimal cover is:**

**D->E**

**C->BE**

**E->D**

**B->AC**

**BCNF:**

**Using the above minimal cover, we can do a dependency preserving decomp into (since B and C are keys):**

**R1(DE) R2(ABCD) DO YOU NOT SPLIT R2? B->AC IS VIOLATING B -> AC is NOT violating R2(A,B,C,D) since closure of {B}+ is {B, A, C, D, E} and thus, B is a candidate key for R2 (A,B,C,D)**

**iii.**

FDs:

D-->E

CDE-->B

E-->D

C-->D

B-->A

AB-->C

Canonical cover: as above

3NF:

~~R1(DE)~~ R2(CDEB) ~~R3(ED)~~ ~~R4(CD)~~ ~~R5(BA)~~ R6(ABC)

**c)**

**i.**

|  |  |  |
| --- | --- | --- |
| **FD** | **LHS extraneous** | **RHS extraneous** |
| **F-->B** | none | none |
| **D-->A** | none | none |
| **D-->E** | none | none |
| **EF-->D** | none | none |
| **C-->E** | none | none |
| **D-->B** | none | none |
| **B-->AE** | none | none |
| **BF-->A**  **F-->A** | B | none |
| **AE-->F** | none | none |

**ii.**

**R(ABCDEF)**

**R1(FB) R2(ACDEF)**

**R2a(DA) R2b(CDEF)**

**R2ba(DE) R2bb(CDF)**

**Alternative solution that preserves dependency:**

**From FD9, FD1, FD4 we derive a new FD: AE -> BDF**

**Since AE is not a super key, take this as the violating FD and split into R1(A, E, B, D, F) and R2(C, E, A).**

**This is now in BCNF and is dependency preserving.**

**iii.**

**Canonical cover:**

F->BA

D->AEB

EF->D

C->E

B->AE

AE->F

Key: D

R1(FBA) R2(DAEB) R3(EFD) R4(CE) ~~R5(BAE)~~ R6(AEF)

**Alternative c) answer: Let me know if you spot a mistake!**

c)

R(A, B, C, D, E, F)

FDX:

FD1: F->B

FD2: D->A

FD3: D->E

FD4: ~~E~~ F->D

FD5: C->E

FD6: D->B

FD7: B->AE

FD8: ~~B~~F ->A

FD9: AE->F

|  |  |  |
| --- | --- | --- |
| **FD** | **LHS Extraneous?** | **RHS Extraneous?** |
| FD4: EF->D        FD4: F->D | E?  {LHS-E}+ = {FBAE…}  Yes    FD4: F->D | N/A |
| FD7: B->AE | N/A | A?  {LHS}+ = {BE} No  E?  {LHS}+ = {BA} No |
| FD8: BF->A      F->A | B?  {LHS-B}+ = {FB…} Yes | N/A |
| FD9: AE->F | A?  {AE-A}+ = {E} No  E?  {AE – E} = {A} No | N/A |
|  |  |  |

So FD without extraneous:

FD1: F->B

FD2: D->A

FD3: D->E

FD4: F->D

FD5: C->E

FD6: D->B

FD7: B->AE

FD8: F ->A

FD9: AE->F

*ii) If not already in BCNF, normalise to BCNF such that the resulting decomposition is dependency preserving. If it cannot be decomposed dependency preserving, explain why*

Cannot be decomposed dependency preserving.

*iii) Unless already in 3NF, decompose R into 3NF.*

Canonical cover:

AE->F – A not extraneous, A not

B->AE – A not, E not

C->E

D->~~A~~ B ~~E~~ – A extraneous, E extraneous

F->~~A B~~ D – A extraneous, B extraneous

So canonical cover is:

AE->F

B->AE

C->E

D->B

F->D

Key = {CF}

3NF:

**R1(AEF), R2(ABE), R3(CE), R4(BD), R5(DF), R5(CF)**

**d)**

In conflict: Y is used by T1 then T2 (fine) then t3(also fine) but then back to t2

Therefore the schedule is not conflict serialisable

Could commit after T3Wy?

Move T3Wy?

**e)**

Yes

Pretty sure its not based on the cycle in the precedence graph

**f)**

T1: Wa C Wb C

T2: Wa C Rb C

Alternative

T1: Wa Wb C

T2: Wa Rb C

T1: Wa Wa Wb C

T2: Wa Rb C

T1: Wa Wb C

T2: Rb Wa C

ALTERNATIVE ANSWER:

i)

T1| WA WB C

T2| WA RB C

ii)

T1| WA WB C

T2| WA RB C

iii)

T1| WA WB C

T2| WA RB C