Databases 2019-2020 (the other file wasn’t complete so Ive added this one)

https://exams.doc.ic.ac.uk/feedback/2019-2020/C526\_Examination%20Feedback.pdf

Assumes order by is ascending by default:

1. A)

Select distinct title

From movie join dvd\_rental on(movie.movieID = dvd\_rental.movieID)

having count(dateOut) > 3 AND count (dateOut) < 5

Group by title

order by title

*To satisfy requirement of using subquery:*

*SELECT Title*

*FROM MOVIE*

*WHERE Title IN (*

*SELECT DISTINCT Title*

*FROM MOVIE*

*JOIN DVD\_RENTAL ON MOVIE.MovieID = DVD\_RENTAL.MovieID*

*HAVING COUNT(MOVIE.MovieID) BETWEEN 3 AND 5*

*)*

*ORDER BY Title ASC*

Alternative:

SELECT title

FROM MOVIE

WHERE MovieID IN

(SELECT DISTINCT MovieID

FROM DVD\_RENTAL

GROUP BY MovieID

HAVING count(\*)>=3 AND count(\*)<=5)

ORDER BY title ASC

b)

assumption: dateout and date in in date format year-month-day

select title,

extract(MONTH from dateOut),

count(\*)

from dvd\_rental join movie (dvd\_rental.movieID = movie.movieID)

group by title

order by title, dateOut

except

select title

from dvd join movie (dvd\_rental.movieID = movie.movieID)

having extract(MONTH from dateOut) in (‘March’, ‘April’, ‘May’) and extract(YEAR from dateOut) = ‘2018’

**Alternative (not sure if write so do correct if not):**

SELECT m1.title, MONTH(DateOut) monthrent, COUNT(\*) monthcount FROM DVD\_RENTAL d1

JOIN MOVIE m1 ON m1.MovieID = d1.MovieID

GROUP BY d1.MovieID, monthrent

ORDER BY m1.title, YEAR(DateOut), monthrent

EXCEPT

SELECT m1.title, MONTH(DateOut) monthrent, COUNT(\*) monthcount from DVD\_RENTAL d2

JOIN MOVIE m2 ON m2.MovieID = d2.MovieID

GROUP BY d2.MovieID, monthrent

WHERE d2.monthrent != 3 AND d2.monthrent != 5 AND YEAR(DateOut) != 2018

Alternative:

SELECT MOVIE.title, MONTH(DVD\_RENTAL.DateOut) as Month, COUNT(\*) as count

FROM MOVIE JOIN DVD\_RENTAL on MOVIE.MovieID=DVD\_RENTAL.MovieID

WHERE (MONTH(DVD\_RENTAL.DateOut) = 3

OR MONTH(DVD\_RENTAL.DateOut) = 5)

AND YEAR(DVD\_RENTAL.DateOut) = 2018

GROUP BY MOVIE.MovieID, month

ORDER BY (MOVIE.title,

YEAR(DVD\_RENTAL.DateOut),

MONTH(DVD\_RENTAL.DateOut)) ASC

c)

select (think you need distinct here) s.address, r.address

from DVD\_Store as s join dvd\_rental as sd on (s.storeID = sd.storeID) join renter as r on (r.memberNo = sd.memberNo)

order by s.address

d)

select name, nof\_DVDs

from producer join movie on (producerName = name) join DVDs on (movie.movieID = DVDs.movieID)

group by name

order on nof\_DVDs

Alternative:

SELECT MOVIE.ProducerName AS Name, SUM(DVDs.nof\_DVDs) AS Number\_of\_DVDs

FROM MOVIE JOIN DVDs ON MOVIE.MovieID = DVDs.MovieID

GROUP BY Name

ORDER BY Number\_of\_DVDs ASC

e)

select memberNo

from dvdRental

where dateDue < todayDate

group by memberNo

having count(storeID) = 2 --> count(distinct storeID)?

Alternative with subquery:

SELECT MemberNo FROM

(SELECT DISTINCT MemberNo, StoreID FROM DVD\_RENTAL

WHERE DateDue < GETDATE() )

GROUP BY MemberNo

HAVING COUNT(\*) = 2

ORDER BY MemberNo ASC

Alternative:

SELECT D1.MemberNo

FROM DVD\_RENTAL D1

WHERE D1.DateDue > 2022-05-10 AND EXISTS

(SELECT \*

FROM DVD\_RENTAL D2

WHERE D2.DateDue > 2022-05-10

AND D1.MemberNo = D2.MemberNo

AND D1.StoreID <> D2. StoreID)

ORDER BY D1.MemberNo ASC

Alternative (Let me know if you think theres a mistake )

Select distinct D1.MemberNo  
 From DVD\_Rental as D1, DVD\_Rental as D2  
 Where D1.MemberNo = D2.MemberNo and D1.StoreID <> D2.StoreID   
 and D1.DateDue < current\_date and D2.DateDue < current\_Date  
 order by D1.MemberNo asc

2)

Assumption: for multivalued phone attributes, assumes that no phones are shared

Car(PK: license, manufacturer, model, year)

Client(PK: ID, address, name)

Client\_phone(PK: ID, PK: phone)

ID references client.ID on delete cascade

RepairJob(PK: Number, PK: carLicense, FK: mNumber, description, parts, work)

carLicense references car.License on delete cascade

mNumber rederences mechanic.number

mechanic(pk: number)

number references employee.number

salesman(pk: number)

number references employy.number

employee(pk: number, name)

Buys(pk: number, pk:license, pk:ID, price, date, value)

Number references salesman.number

License references car.license

ID references client.ID

Sells(pk: ID, pk: number, PK: CarLic , date, commission, value)

ID references client.ID

Number references salesman.number

CarLic references Car.license on delete cascade

Is there any reason only two of the references are delete cascade?

3.

|  |  |  |
| --- | --- | --- |
| FD | LHS extraneous | RHS extraneous |
| BD->AC  B->C | Is B?  No  Is D?  yes | is a?  {BCA  yes |
| C->E | None | None |
| B->CE  B->E | None | Is C?  {BEAC yes |
| D->C | None | None |
| B->A | none | None |
| AB->E  B->E | Is a?  {BCE yes | None |
| ABD->E  D->E | Is a?  {BDCE yes  BD->E  Is b?  {DCE yes | None |
| C->AB  C->B | none | Is a?  {CBA yes |

FDs: B->C, C->E, B->E, D->C, B->A, B->E, D->E, C->B

Canonical cover:

BàACE

DàC

CàB

ii) Key = AD

R1(BACE) R2(CD) R3(CB) R4(AD)

iii)

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

R1(CE) R2(ABCD) A+{} B+{BCEA} C+{CEAB} D+{DCEAB}

R2a(CAB) R2b(DC)

R3a(BA) R3b(CB)

If we use the minimal cover (which is equivalent to the original FD set), and the given keys B and D, the only violating FD is C->B so a simple decomp would be:

R1(CB) R2(ACDE). For every FD in the minimal cover, each LHS is a superkey.

B)

i)

|  |  |  |
| --- | --- | --- |
| D->E |  |  |
| CDE->B |  |  |
| BCD->E  D->E | Is B?  {CDE yes  CD->E  Is C?  {DE yes |  |
| E->D |  |  |
| D->C |  |  |
| B->A |  |  |
| AC->D |  |  |
| AB->CE  AB->E |  | C |

D->E, CDE->B, E->D, D->C, B->A, AC->D, AB->E

ii)

R(ABCDE) A is superkey

R1(DE) Using D->E

R2(ABCD) A+{} B+{BACED} c+{} D+{DCEBA} AC+{ACDEB} therefore B, D, and AC are superkeys

**Why there is E in the closure of B, D, AC?**

iii)

R(ABCDE)

Canonical cover

D->EC

CDE->B

E->D

B->A

AC->D

AB->E

~~R1(DEC)~~ R2(CDEB) ~~R3(ED) R4(BA)~~ R5(ACD) R6(ABE) R7(ABC)

c)

i)

|  |  |  |
| --- | --- | --- |
| F->B |  |  |
| D->A |  |  |
| D->E |  |  |
| EF->D |  |  |
| C->E |  |  |
| D->B |  |  |
| B->AE |  | Is a?  {BE no  Is E?  {BA no |
| BF->A  F->A | Is b?  {FBAE yes |  |
| AE->F | Is a?  {E no  Is E?  {A no |  |
| CF->ADE  F->ADE  F->E | Is c?  {FABEF yes | Is A?  {FDEA yes  F->DE  Is D?  {FED yes  F->E |

ii)

R(ABCDEF) superkeys C,D,F

R1(BAE) R2(BCDF)

R1a(AEF) R1b(AEB) ?

R2 superkeys: CF+{CFADEB}

R2a(FB) R2b(FDC)

iii)

canonical cover:

F->BAE

D->AEB

EF->D

C->E

B->AE

AE->F

R1(FBAE) R2(DAEB) R3(EFD) R4(CE) ~~R5(BAE) R6(AEF)~~ R7(DC)

4a.

Read committed means that only data committed by other concurrent transactions can be read. Since both transactions do not commit until the end, none of the information changed by one transaction is read by the other.

1. A deadlock occurs when two transactions block each other by holding locks on resources that each of the transaction needs.

Here, a deadlock can occur:

T1 does its first select statement and table R is locked

T2 does its first select statement and table S is locked

T1 requests table S but it is locked from T2

T2 requests table R but it is locked from T2

Transaction 1 cannot complete until transaction 2 does, and vice versa, thus this is in deadlock.

1. Do not transact concurrently, this avoids deadlocks because they are not running at the same time.

Alternatively, another solution would be to have T1 use select update on R commit then select update on S commit, and have T2 select update on R commit then select update on S commit. This way they both free the resource S or R after using it.

b. LOW CONFIDENCE but something along these lines???

102

4 \* 2 million or 6 \* 2 million (depending on how you read the question)

>100M total capacity

(4096 – 4) / 40 = 102 children per node

Each block = 4096 bytes, 40 = 32+8, -4 is for the pointer

102\*102 \* 102\* 102 > 2mil == 102^4

Therefore 4 levels, nee to bring in 3 to access each

102^4 total storage

*Very low confidence too...*  
*A block can have between m and ceil(m/2) child nodes… so each internal node will have at most m pointers to child nodes and m – 1 keys to separate the pointers.*

*Hence we need that*

*- m \* 8 + (m-1) \* 32 + 4 <= 4096 (where 8 is the child pointer size, 32 is the key size, 4 is the next node pointer)*

*- 40m <= 4124*

*- m <= 103.1*

*so we choose the largest integer that satisfies this, which is m = 103. The maximum number of child nodes should be 103.*

*(Very probably wrong) Since a pointer is 8 bytes, we can store 2^64 items in the tree.*