**`Final Exam**

**CS425 - Database Organization Results**

Instructions •

Try to answer all the questions using what you have learned in class. Keep hard questions until the end.

When writing a query, write the query in a way that it would work over all possible database instances and not

just for the given example instance!

• The exam is closed book!

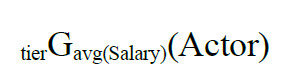
C-Id

|  |  |  |
| --- | --- | --- |
| **Customer** | **Account** | |
| |  |  |  |  | | --- | --- | --- | --- | | **cId** | **Name** | **creditcard** | **DOB** | | 152 | Pam | 4546436 | 1924 | | 26 | John | 5325626 | 1932 | | 352 | Alice | 2662366 | 1940 | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | **a-id** | **C\_id** | **Type** | **fee** | **joined** | | 64 | 152 | Premium | 100 | 2001 | | 75 | 26 | Regular | 50 | 2019 | | 4 | 352 | Regular | 60 | 2015 | | |
| **userPlay** | **actor** | |
| |  |  |  |  | | --- | --- | --- | --- | | **a\_id** | **m\_id** | **Date** | **MinutesWatch** | | 152 | 1 | 1/4/2019 | 50 | | 2 | 1 | 1/5/2019 | 60 | | 3 | 3 | 1/6/2020 | 60 | | |  |  |  |  | | --- | --- | --- | --- | | **s\_id** | **Name** | **Salary** | **tier** | | 152 | Sam | 5000000 | B | | 2 | Paul | 2000000 | C | | 3 | Tim | 8000000 | A | | |
|  |  | |
| **Movie** |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **m\_id** | **sId** | **Name** | **type** | **available** | **rating** | **Minutes** | **ReleaseDate** |
| 1 | 152 | the mask | comedy | Regular | 5 | 60 | 1/4/2019 |
| 2 | 26 | Fall | drama | Premium | 4 | 75 | 1/5/2019 |
| 3 | 352 | Outrage | fiction | Regular | 2 | 60 | 1/6/2020 |

1. Relational Algebra (total 22 points)

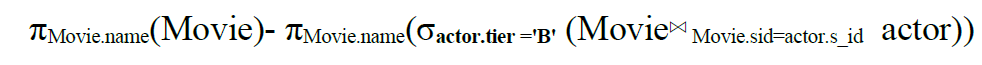
Question I.1 (5 Points) Write a relational algebra expression that returns the average salary for the actors in each tier



Question I. 2 (5 Points) Write a relational algebra expression that returns the name of the movie type customer John watches.

Π name (Π cid (σ name = john (customer))∞account∞userPlay∞movie)

Question I.3 (6 Points) Write a relational algebra expression that returns the name movies that tier B actors never acted in any of the movies



Question I.4 (6 point) Write a relational algebra expression that returns the name of the customers that watch all movies rated 5customer ÷σ rate = 5 (movie)

# SQL (40 points)

Question II.1 (7 points) Write an SQL statement that creates a new relation **payments** that records information about all payments received by each account. The payments are uniquely identified by confimationNumber, the relation includes the a\_id, m\_id, dueDate, paymentDate, amount. Amount is a positive and required field. If an account is removed from the account table, a\_ id is set to null.

**II.1**

create table payments (

    confirmationNumber  integer  primary key,

    a\_id                integer,

    m\_id                integer,

    dueDate             date,

    paymentDate         date,

    amount              integer  not null,

    foreign key (a\_id) references Account on delete cascade,

    foreign key (m\_id) references Movie,

    check (amount > 0)

);

Question II-2 (7 Points) Write an SQL statement that returns the name of the customer, the movie they watched

select Customer.Name, Movie.name

from (Customer join userPlay on (Customer.cID = userPlay.a\_id)) join Movie using (m\_id);

Question II-3 (7 Points) Write the SQL statement that returns the name of the customer who never watched the same movie twice

select Customer.Name

from (

    select cID, Customer.Name, count(m\_id), count(distinct m\_id)

    from (Customer join userPlay on (Customer.cID = userPlay.a\_id)) join Movie using (m\_id)

    group by cID, Customer.Name

    having count(m\_id) = count(distinct m\_id)

);

Question II-4 (8 Points) Write the SQL statement that returns the name of the customer name that missed one or more payments due date (payment table dueDate before paymentDate) and account type is premium.

lect distinct Customer.Name

from (Customer join payments on (Customer.cID = payments.a\_id)) join Account on (Customer.cID = Account.a-id)

where payments.paymentDate > payments.dueDate and Account.Type = 'Premium';

Question II-5 (4 Points) Write the SQL statement that returns the name of the actor’s whose movies in not included in the movie table

select distinct actor.Name

from actor left outer join Movie on (actor.a\_id = Movie.aId)

where Movie.m\_id is null

Question III -5 (7 points)

Write one SQL statement that increase member fee by 10% for all premium member register after the 2015 and 5 all regular member

Update accont s set fee =case

When category = ‘electronics’ then s.price\*1.1

Else when category = ‘baby’ then 0.95\*s.price

Else salary end

From product where product.model = store.model;

# III . ER\_diagram (22 points)

**Restaurant Application**

Create an ER diagram for a restaurant application. Where the application allows the following:

* The customer can register online with a name, address, credit card, point and get a unique id
* The menu information name, cuisine, price, category
* The staff information name, salary, speciality



* The user store user’s information the userName, password, type
* Customer and staff logs to the app using their user information
* The customer can order food from a menu and a quantity for each order and type (delivery, pickup)
* The staff fulfill an order

Question III.1 (12 points) Create the ER-diagram for the restaurant system



Question III-2 ( 10 points) Use the reduction rules to get the schema for the restaurant system.

Rule 1) Strong entities

user(user\_id, username, password, name, type)

menu\_item(item\_id, name, cuisine, price, category)

order(order\_id, quantity, type)

Rule 2) Weak entities

There are none

Rule 8) Specialization method 1 (I do this before so I can do the relationships after)

customer(user\_id, address, credit\_card, points)

staff(user\_id, salary, speciality)

Rule 3) Binary relationships one-to-one

There are none

Rule 4) Binary relationships one-to-many / many-to-one

order(order\_id, quantity, type, item\_id, customer\_id, staff\_id)

Rule 5) Binary relationships many-to-many

There are none

Rule 6) N-ary relationships

There are none

Rule 7) Multivalue attributes

There are none

FINAL SCHEMA

user(user\_id, username, password, name, type)

menu\_item(item\_id, name, cuisine, price, category)

order(order\_id, quantity, type, item\_id, customer\_id, staff\_id)

customer(user\_id, address, credit\_card, points)

staff(user\_id, salary, speciality)

# IV. Transaction ( 16 points)

For each of the following schedules determine which properties this schedule has. E.g., a schedule may be recoverable and cascade, view serializable, conflict serializable and justification

S1 = r3(B), r2(B), w2(C), w4(A),w1(C), w2(A), c2, r3(A), c1, c3 , c4

S2 = r2(A), r1(B), w2(A), r2(B), r3(A), w1(B), c1, w3(A), c3, w2(B), c2



|  |  |  |  |
| --- | --- | --- | --- |
| T1 | T2 | T3 | T4 |
|  |  | read(B) |  |
|  | read(B) |  |  |
|  | write(C) |  |  |
|  |  |  | write(A) |
| write(C) |  |  |  |
|  | write(A) |  |  |
|  | commit |  |  |
|  |  | read(A) |  |
| commit |  |  |  |
|  |  | commit |  |
|  |  |  | Commit |

* Conflict serializable
  + Precedence graph:
    - T2 -> T1 (both write C)
    - T4 -> T2 (both write A)
    - T4 -> T3 (T4 writes A, T3 reads A)
    - T2 -> T3 (T2 writes A, T3 reads A)
  + Precedence graph is acyclic, so S1 is conflict serializable
* View serializable
  + Since S1 is conflict serializable, it must also be view serializable
* Not recoverable
  + Note that T4 writes to A before T3 reads from A, but T4 commits after T3 commits
* Not cascadeless
  + Note that T4 writes to A before T3 reads from A, but T4 commits after T3 reads A

**S2**

|  |  |  |
| --- | --- | --- |
| T1 | T2 | T3 |
|  | read(A) |  |
| read(B) |  |  |
|  | write(A) |  |
|  | read(B) |  |
|  |  | read(A) |
| write(B) |  |  |
| commit |  |  |
|  |  | write(A) |
|  |  | Commit |
|  | write(B) |  |
|  | commit |  |

* Not conflict serializable
  + Precedence graph:
    - T2 -> T3 (T2 reads A, T3 writes A)
    - T1 -> T2 (T1 reads B, T2 writes B)
    - T2 -> T3 (T2 writes A, T3 reads A)
    - T2 -> T3 (both write A)
    - T2 -> T1 (T2 reads B, T1 writes B)
    - T1 -> T2 (both write B)
  + Precedence graph contains a cycle between T1 and T2, so S2 is not conflict serializable
* View serializable
  + Initial value reads:
    - T2 reads initial value of A
      * T3 also reads/writes to A, so T3 must come after T2 in serial equivalent
    - T1 reads initial value of B
      * T2 also reads/writes to B, so T2 must come after T1 in serial equivalent
  + Write-reads
    - T2 writes A and T3 reads that written value of A
      * Therefore T3 must come after T2 in serial equivalent
  + Final writes
    - T3 performs final write to A
      * T2 also writes to A, so T2 must come before T3 in serial equivalent
    - T2 performs final write to B
      * T1 also writes to B, so T1 must come before T2 in serial equivalent
  + Using the observations above, we can construct an equivalent serial schedule with the serial transaction order of (T1, T2, T3)
  + Thus S2 is view serializable
* Not recoverable
  + T2 writes to A before T3 reads from A, but T2 commits after T3 commits
* Not cascadeless
  + T2 writes to A before T3 reads from A, but T2 commits after T3 reads from A

## Part 4 Normalization and Functional Dependencies (24 Points)



Consider the following relation *R*(*A, B, C, D, E, F,H*) and functional dependencies *F* that hold over this relation.

A→ D

AE → H

DF → BC

E → C

H → E

## Question 4.1 (3 Points)

Determine all the possible candidate key(s) of R.

{A,E,F}; {A,H,F};

## Question 4.2 (4 Points)

Compute the attribute cover of X = {D, F} according to F.

X+ = {D, F , B, C}

## Question 4.3 (5 Points)

Compute the canonical cover of *F.* Show each step of the generation process according to the algorithm shown in class.

**Union**

AE → H

D F → C

D F → B

E → C

H → E

*Fc*

AE→ H

DF → BC

H → C

**R1 {AEH}  
R2 {DFBC}**

**R3 {HC}**

**R4{AEF}**