EX1 - Seminar Sample Practical

Sunday, December 29, 2024 1:20 PM

I Let R be a table in a SQL Server database with schema R[FK1, FK2, C1, C2, C3, C4, C5]. The primary key is {FK1, FK2}. Answer questions 1-3 using the legal instance below (each question has at least one correct answer).

FK1	FK2	C1	C2	C3	C4	C5
1	1	Pisica pe acoperisul fierbinte	Tennessee Williams	100	20	AB
1	2	Conul Leonida fata cu reactiunea	Ion Luca Caragiale	50	50	CQ
1	3	Concert din muzica de Bach	Hortensia Papadat-Bengescu	50	10	QC
2	1	Fata babei si fata mosneagului	Ion Creanga	100	100	QM
2	2	Frumosii nebuni ai marilor orase	Fanus Neagu	10	10	BA
2	3	Frumoasa calatorie a ursilor panda povestita de un saxofonist care avea o iubita la Frankfurt	Matei Visniec	100	20	MQ
3	1	Mansarda la Paris cu vedere spre moarte	Matei Visniec	200	10	PQ
3	2	Richard al III-lea se interzice sau Scene din viata lui Meyerhold	Matei Visniec	100	50	PQ
3	3	Masinaria Cehov. Nina sau despre fragilitatea pescarusilor impaiati	Matei Visniec	100	100	AZ
4	1	Omul de zapada care voia sa intalneasca soarele	Matei Visniec	100	100	CP
4	2	Extraterestrul care isi dorea ca amintire o pijama	Matei Visniec	50	10	CQ
4	3	O femeie draguta cu o floare si ferestre spre nord	Edvard Radzinski	10	100	CP
4	4	Trenul din zori nu mai opreste aici	Tennessee Williams	200	200	MA

1. Consider query Q below:

```
SELECT C2, SUM(C3) TotalC3, AVG(C3) AvgC3
FROM R
WHERE C3 >= 100 OR C1 LIKE '%Pisica%'
GROUP BY C2
HAVING SUM(C3) > 100
```

a. Q returns 3 records and value *Matei Visniec* is in its result set.
b. Q returns 3 records and value *Matei Visniec* is not in its result set.
c. Q returns 2 records and value *Ion Creanga* is not in its result set.
d. Q returns 2 records and value *Ion Creanga* is in its result set.
e. None of the above answers is correct.

GROUP BY C2 ⇒ 1. Tennessee 2. Ion Creanga 3. Matei Visniec

SUM(C3): 300 100 650

HAVING SUM(C3) > 100 ⇒ 1. Tennessee 2. Matei

2. How many records does the following query return?

```
SELECT *
FROM
  (SELECT FK1, FK2, C3+C4 TotalC3C4
  FROM R
  WHERE FK1 = FK2) r1
        INNER JOIN
  (SELECT FK1, FK2, C5
  FROM R
  WHERE C5 LIKE '%Q%') r2 ON r1.FK1 = r2.FK1 AND r1.FK2 = r2.FK2
```

a. 2
b. 8
c. 0
d. 1
e. None of the above answers is correct.

r_1:

FK1	FK2	TotalC3C4
1	1	120
2	2	20
3	3	200
4	4	400

r_2

FK1	FK2	C5
1	2	CQ
1	3	QC
2	1	QM
2	3	MQ
3	1	CQ
3	2	CQ
4	2	CQ

3. Table R has a single trigger defined on it:

```
CREATE OR ALTER TRIGGER TrOnUpdate
  ON R
  FOR UPDATE
AS
  DECLARE @total INT = 0
  SELECT @total = SUM(i.C3 - d.C3)
  FROM deleted d INNER JOIN inserted i ON d.FK1 = i.FK1 AND d.FK2 = i.FK2
  WHERE d.C3 < i.C3
  PRINT @total
```

What's the value returned by the PRINT statement in the trigger when the UPDATE below is executed?

```
UPDATE R
SET C3 = 300
WHERE FK1 < FK2
```

FK1	FK2	C1	C2	C3	C4	C5
1	2	Conul Leonida fata cu reactiunea	Ion Luca Caragiale	50	50	CQ
1	3	Concert din muzica de Bach	Hortensia Papadat-Bengescu	50	10	QC
2	3	Frumoasa calatorie a ursilor panda povestita de un saxofonist care avea o iubita la Frankfurt	Matei Visniec	100	20	MQ

D ⇒ deleted

300
300
300

i ⇒ inserted

a. 550
b. 700
c. 650
d. 600
e. None of the above answers is correct.

b) total = 250 + 250 + 200 = 700

EX2 - Seminar Sample Practical

Sunday, December 29, 2024 1:30 PM

II

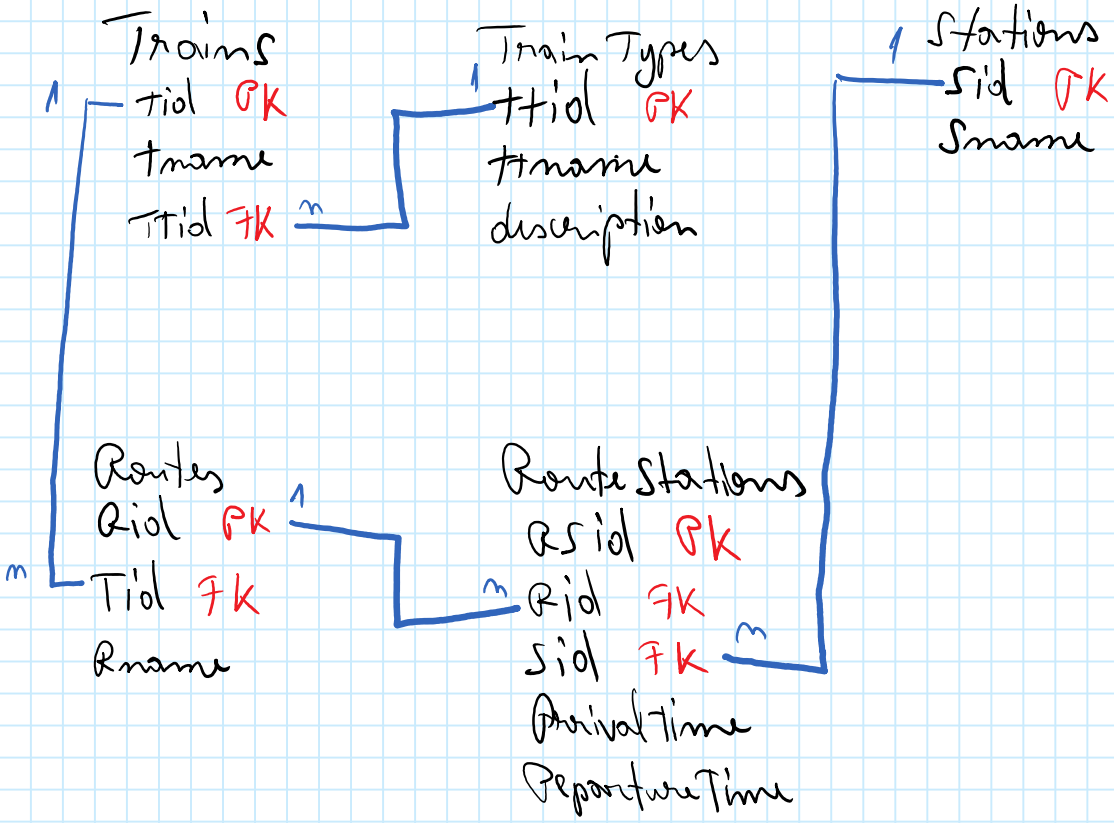
Create a database to manage train schedules. The database will store data about the routes of all the trains. The entities of interest to the problem domain are: *Trains*, *Train Types*, *Stations, and Routes*. Each train has a name and belongs to a type. A train type has a name and a description. Each station has a name. Station names are unique. Each route has a name, an associated train, and a list of stations with arrival and departure times in each station. Route names are unique. The arrival and departure times are represented as hour:minute pairs, e.g., train arrives at 5 pm and leaves at 5:10 pm.

1. Write an SQL script that creates the corresponding relational data model.

2. Implement a stored procedure that receives a route, a station, arrival and departure times, and adds the station to the route. If the station is already on the route, the departure and arrival times are updated.

3. Create a view that shows the names of the routes that pass through all the stations.

4. Implement a function that lists the names of the stations with more than ***R*** routes, where ***R*** is a function parameter.



1.

```
CREATE TABLE TrainTypes (
    TTid INT PRIMARY KEY,
    TTname VARCHAR(50) NOT NULL,
```

```
    description VARCHAR(255)
);

CREATE TABLE TRAINS (
    Tid INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    TTid INT NOT NULL,
    FOREIGN KEY (TTid) REFERENCES TrainTypes(TTid)
);

CREATE TABLE Stations (
    Sid INT PRIMARY KEY,
    Sname VARCHAR(50) NOT NULL UNIQUE
);

CREATE TABLE Routes (
    Rid INT PRIMARY KEY,
    Rname VARCHAR(50) NOT NULL UNIQUE
    Tid INT NOT NULL
    FOREIGN KEY(Tid) REFERENCES Trains(Tid)
);

CREATE TABLE RouteStations (
    RSid INT PRIMARY KEY,
    Rid INT NOT NULL,
    Sid INT NOT NULL,
    ArrivalTime TIME,
    DepartureTime TIME,
    FOREIGN KEY(Rid) REFERENCES Routes(Rid),
    FOREIGN KEY(Sid) REFERENCES Stations(Sid)
);
```

2. 2. Implement a stored procedure that receives a route, a station, arrival and departure times, and adds the station to the route. If the station is already on the route, the departure and arrival times are updated.

```
CREATE PROCEDURE AddOrUpdateStation(
    IN RouteName VARCHAR(50),
    IN StationName VARCHAR(50),
    IN Arrival TIME,
    IN Departure TIME
)
BEGIN
    DECLARE Routeid INT;
    DECLARE Stationid INT;

    SELECT Rid INTO Routeid
    FROM Routes
    WHERE RName = RouteName;

    SELECT Sid INTO Stationid
    FROM Routes
    WHERE SName = StationName;

INSERT INTO RouteStations(Rid, Sid, Arrivaltime, DepartureTime)
VALUES(Routeid, Stationid, Arrival, Departure)
ON DUPLICATE KEY UPDATE
    ArrivalTime = VALUES(Arrivaltime),
    DepartureTime = VALUES(DepartureTime);
```

OR

```
IF EXISTS (
    SELECT 1
    FROM RouteStations
    WHERE Rid = Routeid and Sid = Stationid
```

```
) THEN
    UPDATE RouteStations
    SET ArrivalTime = Arrival,
        DepartureTime = Departure
    WHERE Rid=RouteId AND Sid=StationId
ELSE
    INSERT INTO RouteStations (Rid, Sid, ArrivalTime, DepartureTime)
    VALUES (RouteId, StationId, Arrival, Departure);
END IF;
END;
```

3. Create a view that shows the names of the routes that pass through all the stations.

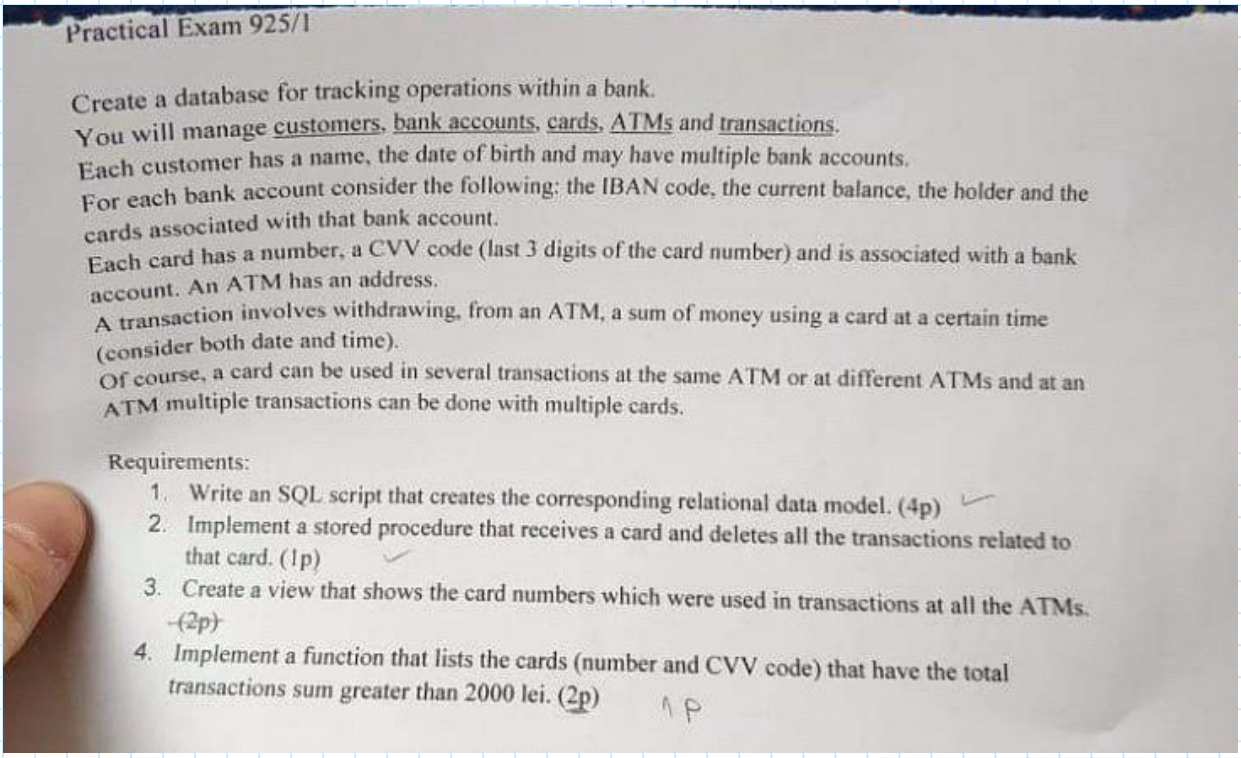
```
CREATE VIEW RoutesThroughAllStations AS
SELECT RName
FROM Routes r
INNER JOIN RouteStations rs ON r.Rid = rs.Rid
GROUP BY RName
HAVING COUNT(*) = (SELECT COUNT(*) FROM Stations);
```

4. Implement a function that lists the names of the stations with more than ***R*** routes, where ***R*** is a function parameter.

```
CREATE FUNCTION StationsWithMoreThanRRoutes(R INT)
RETURNS TABLE
RETURN
    SELECT S.Sname AS StationName
    FROM Stations S
    JOIN RouteStations rs on S.Sid = rs.Sid
    GROUP BY S.Sname
    HAVING COUNT(DISTINCT rs.RID) > R;
```

PracticalEx1

Sunday, December 29, 2024 6:19 PM

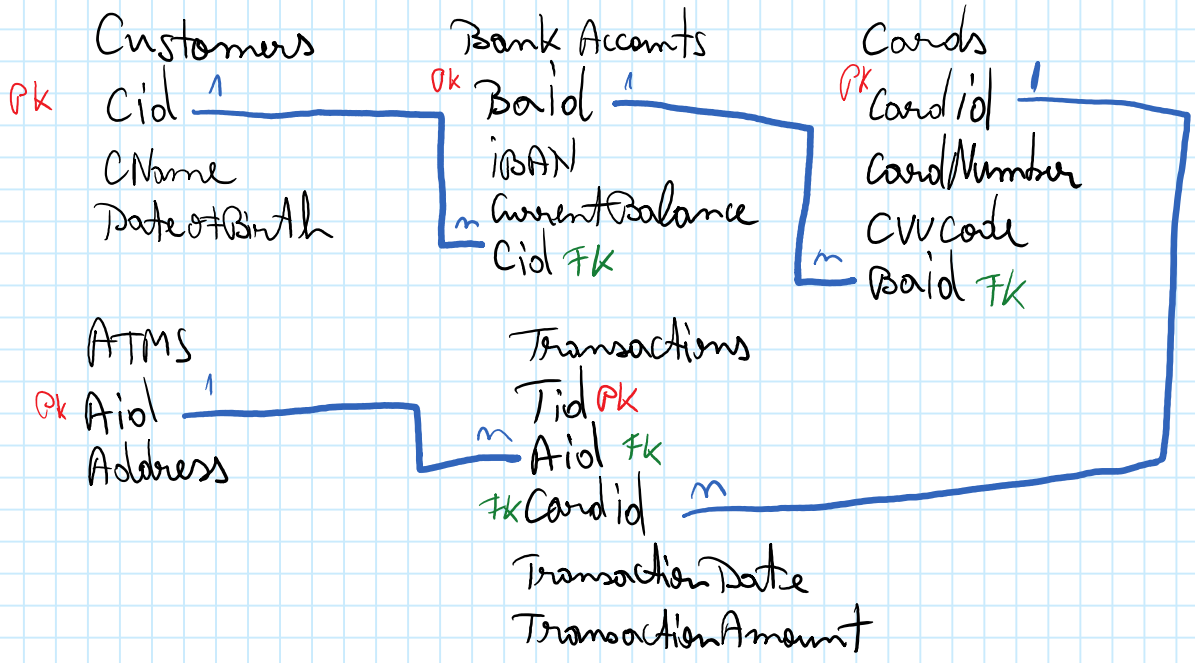


Practical Exam 925/1

Create a database for tracking operations within a bank.
You will manage customers, bank accounts, cards, ATMs and transactions.
Each customer has a name, the date of birth and may have multiple bank accounts.
For each bank account consider the following: the IBAN code, the current balance, the holder and the cards associated with that bank account.
Each card has a number, a CVV code (last 3 digits of the card number) and is associated with a bank account. An ATM has an address.
A transaction involves withdrawing, from an ATM, a sum of money using a card at a certain time (consider both date and time).
Of course, a card can be used in several transactions at the same ATM or at different ATMs and at an ATM multiple transactions can be done with multiple cards.

Requirements:

1. Write an SQL script that creates the corresponding relational data model. (4p)
2. Implement a stored procedure that receives a card and deletes all the transactions related to that card. (1p)
3. Create a view that shows the card numbers which were used in transactions at all the ATMs. (2p)
4. Implement a function that lists the cards (number and CVV code) that have the total transactions sum greater than 2000 lei. (2p)



1. CREATE TABLE Customers (
   Cid INT PRIMARY KEY,

```
    CName VCHAR(50),
    DateOfBirth Date
);

CREATE TABLE BankAccounts
    Baid INT PRIMARY KEY,
    IBAN VCHAR(34),
    CurrentBalance DECIMAL(15,2),
    Cid INT NOT NULL,
    FOREIGN KEY(Cid) REFERENCES Customers(Cid)
);

CREATE TABLE Cards (
    Cardid INT PRIMARY KEY,
    CardNumber VARCHAR(16) NOT NULL UNIQUE
    CVVcode VarCHAR(3),
    Baid INT NOT NULL,
    FOREIGN KEY (Baid) REFERENCES BankAccounts (Baid)
);

CREATE TABLE ATMs (
 Aid INT PRIMARY KEY,
 Address VARCHAR(20)
);

CREATE TABLE Transactions (
Tid INT PRIMARY KEY,
TransactionDate Date,
TransactionAmount DECIMAL(15,2)
```

```
    Aid INT NOT NULL,
    cardid INT NOT NULL,
    FOREIGN KEY (Aid) REFERENCES ATMs(Aid),
    FOREIGN KEY(cardid) REFERENCES Cards(cardid)
);
```

2. Implement a stored procedure that receives a card and deletes all the transactions related to that card. (1p)

```
CREATE PROCEDURE DeleteCardTransactions (
    IN CardNumber Varchar(16)
)
BEGIN
    DECLARE cardid INT;
    SELECT cardid INTO cardid
    FROM cards
    WHERE CardNumber = cardNumber;

    DELETE FROM Transactions
    WHERE cardid = cardid;
END
```

3. Create a view that shows the card numbers which were used in transactions at all the ATMs. (2p)

```
CREATE VIEW CardsUsed AS
SELECT C.CardNumber
FROM Cards C
JOIN Transactions t on C.cardid = t.cardid
GROUP BY C.cardNumber
HAVING COUNT(DISTINCT t.Aid) = (SELECT COUNT(*) FROM ATMs);
```

4. *Implement a function that lists the cards (number and CVV code) that have the total transactions sum greater than 2000 lei. (2p)* 1p

```
CREATE FUNCTION CardsWithLargeTransactions()
RETURNS TABLE
RETURN
    SELECT C.cardNumber, C.CVVCode
    FROM cards C
    JOIN TRANSACTIONS t on C.cardId = t.cardId
    GROUP BY C.CardNumber, C.CVVCode
    HAVING sum(t.TransactionsAmount) > 2000;
```

PracticalEx2

Monday, December 30, 2024 5:09 PM

10.01.2019 – 921/1

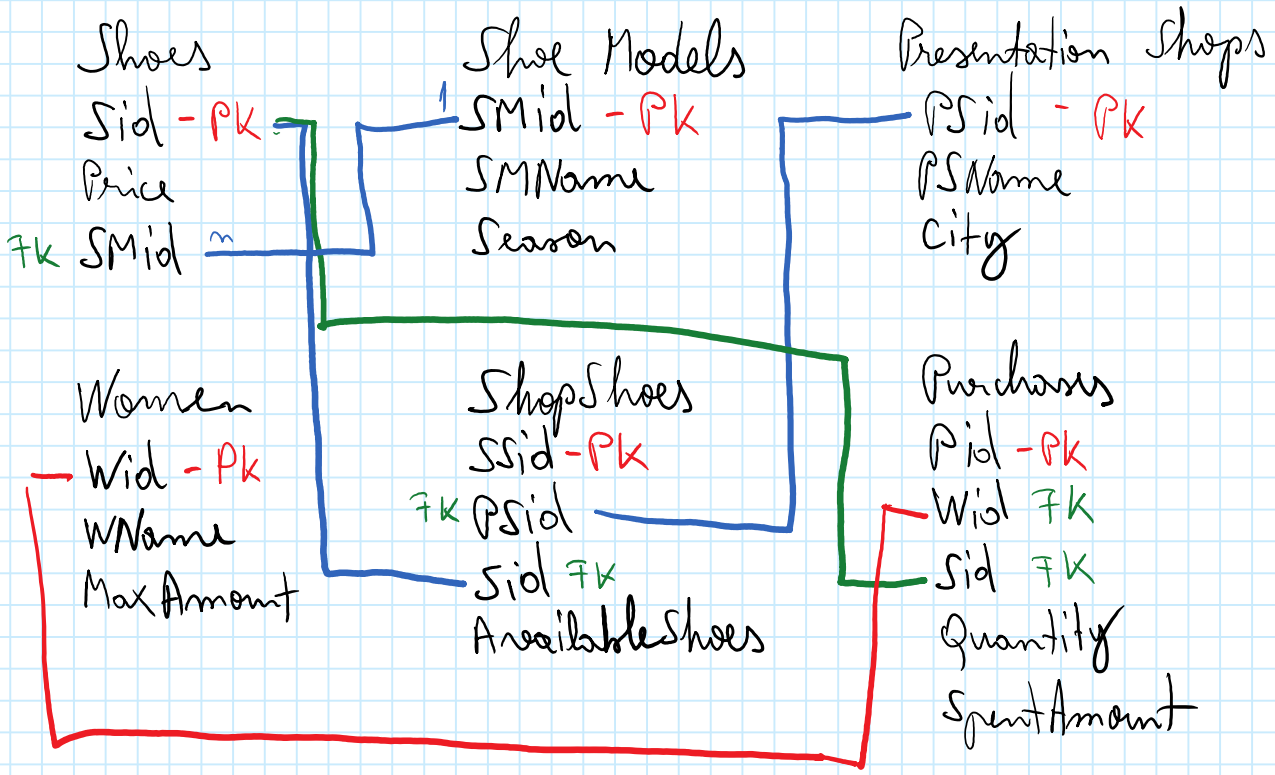
Create a database for women shoes.

a) The entities of the problem domain are: *Shoes*, *Shoe Models*, *Presentation Shops* and *Women*.
b) Each presentation shop has a name and a city.
c) Each woman has a name and a maximum amount to spend.
d) Each shoe has a price and it is part of a shoe model. Each shoe model is characterized by a name and a season. A shoe model contains one or more shoes.
e) A shoe can be found in one or more presentation shops and in a presentation shop can be one or more shoes, characterized also by the number of available shoes.
f) A woman will buy one or more shoes and a shoe will be bought by one or more women, knowing also the number of shoes bought and the spent amount.

TO DOs:

1) Write an SQL script that creates the corresponding relational data model. (4p)
2) Create a stored procedure that receives a shoe, a presentation shop and the number of shoes and adds the shoe to the presentation shop. (1p)
3) Create a view that shows the women that bought at least 2 shoes from a given shoe model. (2p)
4) Create a function that lists the shoes that can be found in at least *T* presentation shops, where *T>=1* is a function parameter. (2p)

(1p of)



1. CREATE TABLE ShoeModels (

```
    SMid INT PRIMARY KEY,
    SMName VARCHAR(50),
    Season VARCHAR(50)
);

    CREATE TABLE Shoes (
    Sid INT PRIMARY KEY,
    Price DECIMAL (15,2),
    SMid INT NO NULL,
    FOREIGN KEY (SMid) REFERENCES ShoeModels (SMid)
);

    CREATE TABLE Women (
    Wid INT PRIMARY KEY,
    WName VARCHAR(50),
    MaxAmount DECIMAL(15,2)
);

    CREATE TABLE PresentationShops (
    PSid INT PRIMARY KEY,
    PSName VARCHAR(50),
    City VARCHAR(50),
);

    CREATE TABLE shopShoes (
    SSid INT PRIMARY KEY,
    PSid INT NOT NULL,
    Sid INT NOT NULL,
    Availableshoes INT,
```

```
    FOREIGN KEY (PSid) REFERENCES PresentationShops (PSid),
    FOREIGN KEY (Sid) REFERENCES Shoes (Sid)
);

    CREATE TABLE Purchases (
    Pid INT PRIMARY KEY,
    Wid INT NOT NULL,
    Sid INT NOT NULL,
    Quantity INT NOT NULL,
    SumAmount DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (Wid) REFERENCES Women (Wid),
    FOREIGN KEY (Sid) REFERENCES Shoes (Sid)
);
```

2) Create a stored procedure that receives a shoe, a presentation shop and the number of shoes and adds the shoe to the presentation shop. (1p)

```
CREATE PROCEDURE AddShoeToShop (
    IN shoeID INT,
    IN shopID INT,
    IN AvailableShoes INT
)
BEGIN
    INSERT INTO ShopShoes (SSid, PSid, Sid, AvailableShoes)
    VALUES (NULL, shopid, shoeid, AvailableShoes)
    ON DUPLICATE KEY UPDATE
        AvailableShoes = AvailableShoes + VALUES (AvailableShoes);
END
```

3) Create a view that shows the women that bought at least 2 shoes from a given shoe model. (2p)

```
CREATE VIEW ShowWomen AS
SELECT W.WName AS WomanName, SM.SMName as ShoeModelName
```

```
FROM Women W
JOIN Purchases P ON W.WId = P.WId
JOIN Shoes S ON P.SId = S.SId
JOIN ShoeModels SM ON S.SMId = SM.SMId
GROUP BY W.WName, SM.SMName
HAVING SUM(P.Quantity) >= 2;
```

4) Create a function that lists the shoes that can be found in at least *T* presentation shops, where *T>=1* is a function parameter. (2p)

```
CREATE FUNCTION ShoesInMultipleShops(T INT)
RETURNS TABLE
RETURN
    SELECT S.SId, SM.SMName AS ShoeModelName,
    COUNT(DISTINCT SS.PSId) AS NumberOfShops
    FROM Shoes S
    JOIN ShopShoes SS on S.SId = SS.SId
    JOIN ShoeModels SM on S.SMId = SM.SMId
    GROUP BY S.Sid, SM.SMName
    HAVING COUNT(DISTINCT SS.PSId) >= T;
```

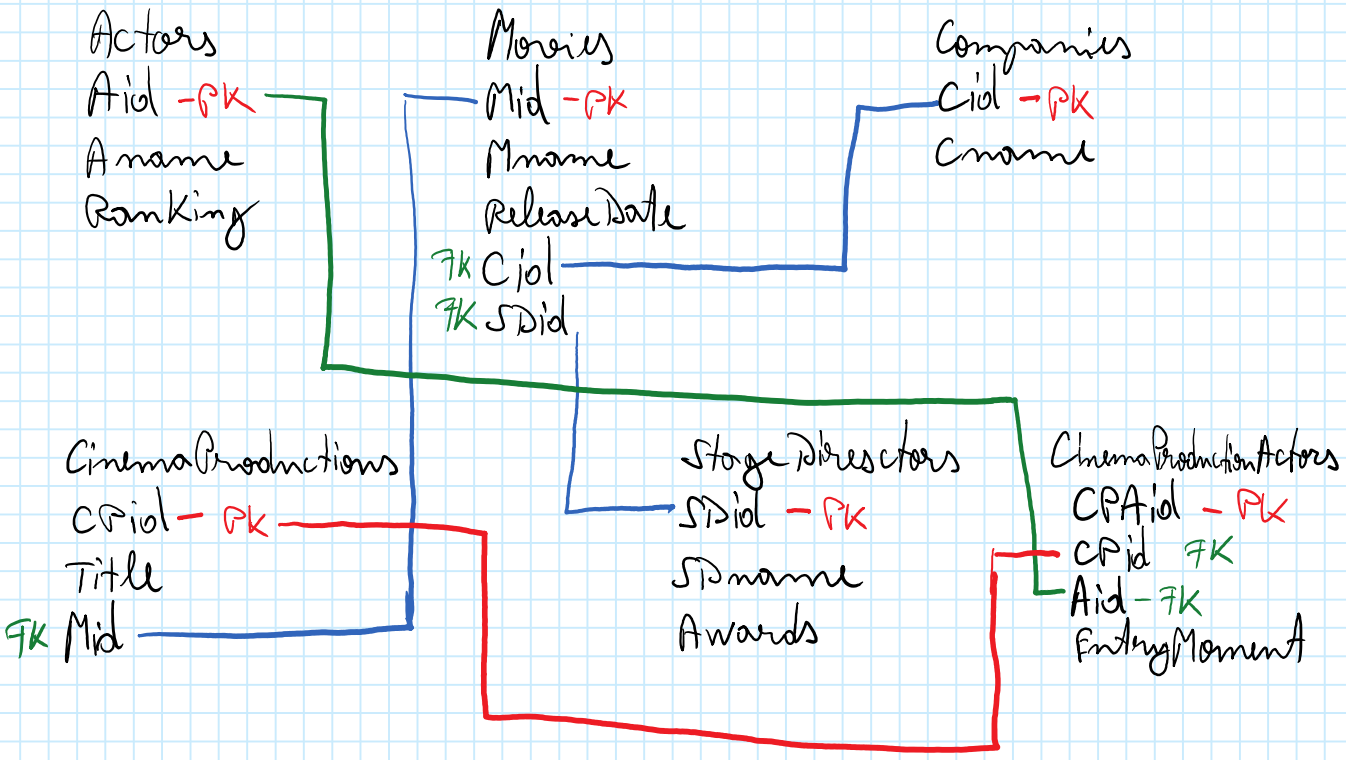
PracticalEx3

Monday, December 30, 2024 7:45 PM

Your assignment is to design a database that manages 'Cinema productions'. The entities of interest for the problem domain are: Actors, Movies, Companies, Cinema Productions and Stage Directors.

Each 'Movie' has a name, a release date, belongs to a production 'Company' and has a stage director. The 'Company' has a name and an ID. A stage director can direct multiple movies, has a name and a number of awards. Each 'Cinema Production' has a title, an associated movie and a list of actors with an entry moment for each actor. Every actor has a name and a ranking.

1) Write an SQL script that creates the corresponding relational data model. (4p)
2) Create a stored procedure that receives an actor, an entry moment and a cinema production and adds the new actor to the cinema production. (1p)
3) Create a view that shows the name of the actors that appear in all cinema productions. (2p)
4) Create a function that returns all movies that have the relase date after '2018-01-01' and have at least p productions, where p is a function parameter. (2p)
(1p of)



1. CREATE TABLE Companies (
Cid INT PRIMARY KEY,
CName VARCHAR(50)
);

```
CREATE TABLE StageDirectors (
 SDid INT PRIMARY KEY,
 SDName VARCHAR(50),
 AWARDS INT
);

CREATE TABLE Movies (
Mid INT PRIMARY KEY,
Mname VARCHAR(50),
ReleaseDate DATE,
Cid INT NOT NULL,
SDid INT NOT NULL,
FOREIGN KEY (Cid) REFERENCES Companies(Cid),
 FOREIGN KEY (SDid) REFERENCES StageDirectors(SDid)
);

CREATE TABLE CinemaProduction (
CPid INT PRIMARY KEY,
Title VARCHAR(50),
Mid INT NOT NULL,
FOREIGN KEY(Mid) REFERENCES Movies(Mid)
);

CREATE TABLE Actors (
  Aid INT PRIMARY KEY,
  Aname VARCHAR(50),
  Ranking DECIMAL(3,2)
);

CREATE TABLE CinemaProductionActors (
  CPAid INT PRIMARY KEY,
  CPid INT NOT NULL,
  Aid INT NOT NULL,
  EntryMoment DATETIME,
  FOREIGN KEY(CPid) REFERENCES CinemaProductions(CPid),
```

```
    FOREIGN KEY(Aid) REFERENCES Actors(Aid)
);
```

1) Write an SQL script that creates the corresponding relational data model.
2) Create a stored procedure that receives an actor, an entry moment and a cinema production and adds the new actor to the cinema production. (1p)
(2p)

```
CREATE PROCEDURE AddActorToProduction (
    IN ActorId INT,
    IN EntryMoment DATETIME,
    IN CinemaProductionID INT
);
BEGIN
    INSERT INTO CinemaProductionActors(CPAId, CPId, Aid, EntryMoment)
    VALUES(NULL, CinemaProductionID, ActorId, EntryMoment)
    ON DUPLICATE KEY UPDATE
        EntryMoment = VALUES(EntryMoment);
END
```

3) Create a view that shows the name of the actors that appear in all cinema productions.

```
CREATE VIEW ShowActors AS
SELECT a.Aname
FROM Actors a
JOIN CinemaProductionActors cpa on a.Aid = cpa.Aid
GROUP BY a.Aname
HAVING COUNT(DISTINCT cpa.cpid) = (SELECT COUNT(*) FROM CinemaProduction);
```

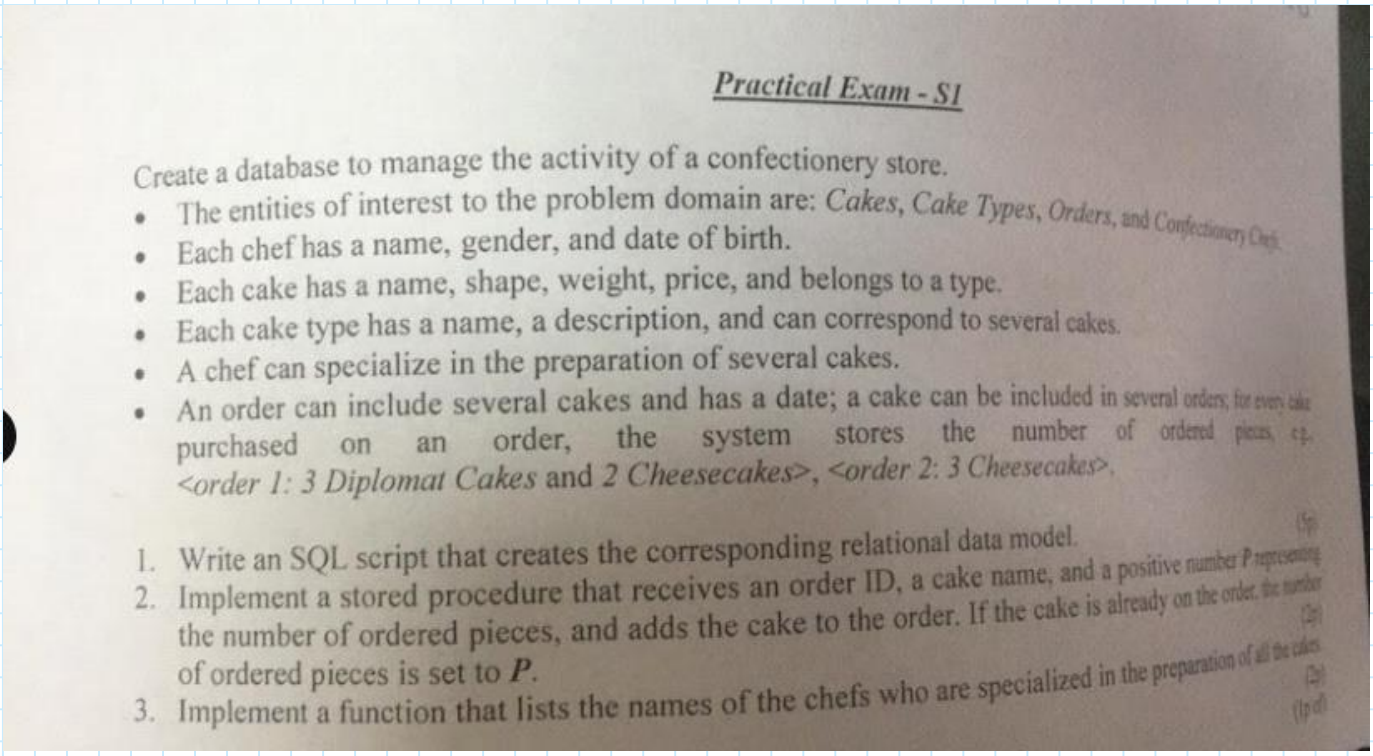
3) Create a view that shows the name of the actors that appear in all cinema productions.
4) Create a function that returns all movies that have the relase date after '2018-01-01' and have at least p productions, where p is a function parameter. (2p)
(1p of)

```
CREATE FUNCTION AllMovies(p INT)
RETURN TABLE
RETURN
    SELECT m.Mname, m.ReleaseDate
    FROM m
    JOIN CinemaProductions cp on m.Mid = cp.Mid
```

```
GROUP BY m.Mid, m.Mname, m.ReleaseDate
HAVING m.ReleaseDate > '2018-01-01' AND COUNT(cp.cpid) >= 5;
```

PracticalEx4

Monday, December 30, 2024 8:54 PM

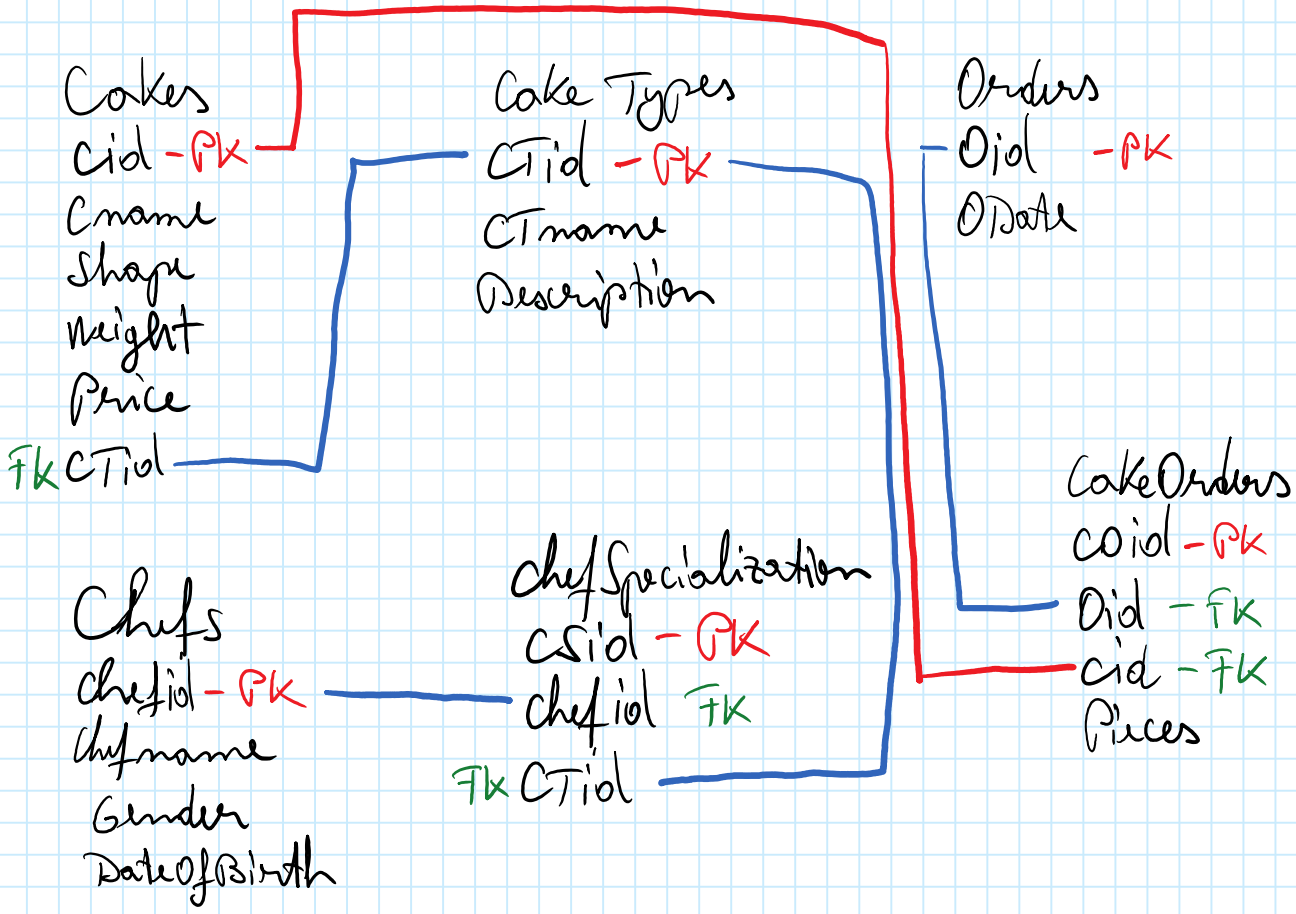


Practical Exam - S1

Create a database to manage the activity of a confectionery store.

- The entities of interest to the problem domain are: *Cakes*, *Cake Types*, *Orders*, and Confectionery Chefs.
- Each chef has a name, gender, and date of birth.
- Each cake has a name, shape, weight, price, and belongs to a type.
- Each cake type has a name, a description, and can correspond to several cakes.
- A chef can specialize in the preparation of several cakes.
- An order can include several cakes and has a date; a cake can be included in several orders; for every cake purchased on an order, the system stores the number of ordered pieces, e.g. *<order 1: 3 Diplomat Cakes* and *2 Cheesecakes>*, *<order 2: 3 Cheesecakes>*.

1. Write an SQL script that creates the corresponding relational data model.
2. Implement a stored procedure that receives an order ID, a cake name, and a positive number *P* representing the number of ordered pieces, and adds the cake to the order. If the cake is already on the order, the number of ordered pieces is set to *P*.
3. Implement a function that lists the names of the chefs who are specialized in the preparation of all the cakes.



1.
```
CREATE TABLE Chefs (
    Chefid INT PRIMARY KEY,
    Chefname VARCHAR(50),
    Gender VARCHAR(1),
    DateOfBirth DATE
);

CREATE TABLE CakeTypes (
    CTid INT PRIMARY KEY,
    CTname VARCHAR(50),
    Description TEXT
);

CREATE TABLE Cakes (
    Cid INT PRIMARY KEY,
    Cname VARCHAR(50),
    Shape VARCHAR(50),
    Weight DECIMAL(5,2),
    Price DECIMAL(10,2),
    CTid INT NOT NULL,
    FOREIGN KEY (CTid) REFERENCES CakeTypes(CTid)
);

CREATE TABLE Chefspecialization (
    CSid INT PRIMARY KEY,
    Chefid INT NOT NULL,
    CTid INT NOT NULL,
    FOREIGN KEY (Chefid) REFERENCES Chefs(Chefid),
    FOREIGN KEY (CTid) REFERENCES CakeTypes(CTid)
);

CREATE TABLE Orders (
```

```
);
CREATE TABLE Orders (
Oid INT PRIMARY KEY,
ODate Date
);
CREATE TABLE CakeOrders (
COid INT PRIMARY KEY,
Oid INT NOT NULL,
cid INT NOT NULL,
PIECES INT,
FOREIGN KEY (Oid) REFERENCES Orders(Oid),
FOREIGN KEY (Cid) REFERENCES Cakes(Cid)
);
```

2. Implement a stored procedure that receives an order ID, a cake name, and a positive number P representing the number of ordered pieces, and adds the cake to the order. If the cake is already on the order, the number of ordered pieces is set to P.

```
CREATE PROCEDURE AddCakeToOrder (
    IN OrderID INT,
    IN CakeName VARCHAR(50),
    IN Pieces INT,
);
BEGIN
    DECLARE CakeID INT;
    SELECT Cid INTO CakeID
    FROM Cakes
    WHERE Cname = CakeName;

    IF EXISTS (
```

```
    SELECT 1
    FROM CakeOrders
    WHERE Oid = OrderID and Cid = CakeID
  ) THEN
    UPDATE CakeOrders
    SET Pieces = Pieces
    WHERE Oid = OrderID and Cid = CakeID;
  ELSE
    INSERT INTO cakeOrders (COid, Oid, Cid, Pieces)
    VALUES (NULL, OrderID, CakeID, Pieces);
  END IF;
END
```

3. Implement a function that lists the names of the chefs who are specialized in the preparation of all the cakes

```
CREATE FUNCTION DisplayChefs()
RETURN TABLE
RETURN
  SELECT C.ChefName AS ChefName
  FROM Chefs C
  JOIN ChefSpecializations CS on C.Chefid = CS.Chefid
  Join CakeTypes ct on CS.CTid = ct.CTid;
```

PracticalEx5

Wednesday, January 1, 2025 2:46 PM

Model for the practical exam

Create a database to manage train schedules. The database will store data about the routes of all the trains. The entities of interest to the problem domain are: *Trains*, *Train Types*, *Stations, and Routes*. Each train has a name and belongs to a type. The train type has only a description. Each station has a name. Station names are unique. Each route has a name, an associated train, and a list of stations with arrival and departure times in each station. Route names are unique. The arrival and departure times are represented as hour:minute pairs, e.g., train arrives at 5pm and leaves at 5:10pm.

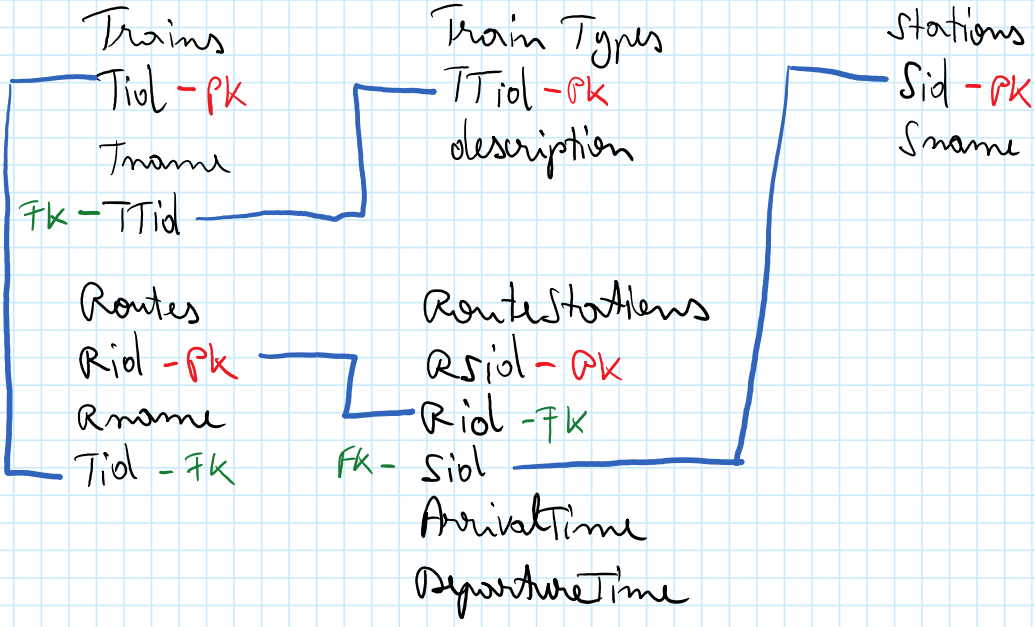
1) Write an SQL script that creates the corresponding relational data model. (4p)

2) Implement a stored procedure that receives a route, a station, arrival and departure times, and adds the station to the route. If the station is already on the route, the arrival and departure times are updated. (1p)

3) Create a view that shows the names of the routes that pass through all the stations. (2p)

4) Implement a function that lists the names of the stations with more than ***R*** routes, where ***R>=1*** is a function parameter. (2 points)

(1p of)



1.
```
CREATE TABLE TrainTypes (
    TTid INT PRIMARY KEY,
    description VARCHAR(100)
);
```

```
CREATE TABLE Stations (
Siol INT PRIMARY KEY,
Sname VARCHAR(50)
);

CREATE TABLE Trains (
Tiol INT PRIMARY KEY,
Tname VARCHAR(50),
TTiol INT NOT NULL,
FOREIGN KEY (TTiol) REFERENCES TrainTypes(TTiol)
);

CREATE TABLE Routes (
Riol INT PRIMARY KEY,
Rname VARCHAR(50) NOT NULL UNIQUE,
Tiol INT NOT NULL,
FOREIGN KEY (Tiol) REFERENCES Trains(Tiol)
);

CREATE TABLE RouteStations (
RSid INT PRIMARY KEY,
Riol INT NOT NULL,
Siol INT NOT NULL,
ArrivalTime TIME,
DepartureTime TIME,
FOREIGN KEY (Riol) REFERENCES Routes(Riol),
FOREIGN KEY (Siol) REFERENCES Stations(Siol)
);
```

2) Implement a stored procedure that receives a route, a station, arrival and departure times, and adds the station to the route. If the station is already on the route, the arrival and departure times are updated. (1p)

```
CREATE PROCEDURE AddStationToRoute(
    IN RouteName VARCHAR(50),
```

```
    IN StationName VARCHAR(50),
    IN ArrivalTime TIME,
    IN DepartureTime TIME
)
BEGIN
    DECLARE RouteID INT;
    DECLARE StationID INT;

    SELECT Rid INTO RouteID
    FROM Routes
    WHERE Rname = RouteName;

    SELECT Sid INTO StationID
    FROM Stations
    WHERE Sname = StationName

    INSERT INTO RoutesStations (Rid, Sid, ArrivalTime, DepartureTime)
    VALUES (RouteID, StationID, Arrivaltime, DepartureTime)
    ON DUPLICATE KEY UPDATE
        ArrivalTime = VALUES (ArrivalTime)
        DepartureTime = VALUES (DepartureTime);
END
```

3) Create a view that shows the names of the routes that pass through all the stations.

```
CREATE VIEW RoutesThroughAllStations AS
SELECT r.Rname
FROM Routes r
JOIN RoutesStations rs on r.Rid = rs.Rid
Group BY r.Rname
HAVING COUNT(DISTINCT rs.Rid) =(SELECT COUNT(*) FROM STATIONS);
```

4) Implement a function that lists the names of the stations with more than ***R*** routes, where ***R>=1*** is a function parameter. (2 points)

```
CREATE FUNCTION ListStations(R INT)
RETURN TABLE
RETURN
    SELECT S.Sname as StationName, COUNT(DISTINCT rs.Rid) As NumberofRoutes
    FROM Stations S
    JOIN RouteStations rs on r.Sid = rs.Sid
    GROUP BY S.Sid, S.Sname
    HAVING COUNT(DISTINCT rs.Rid) > R ;
```

PracticalEx6

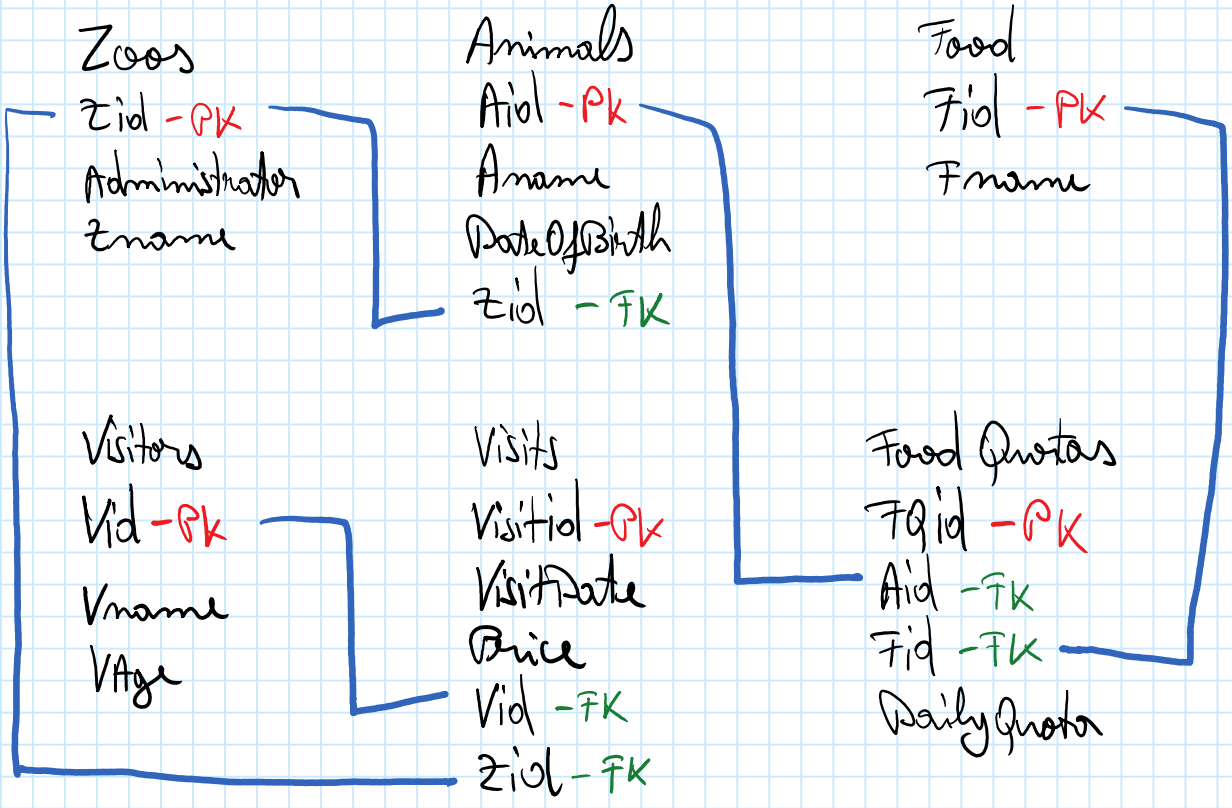
Wednesday, January 1, 2025 4:13 PM

P1
Create a database storing data about several Zoos. You will manage zoos, animals, food, visitors and visits.

Each zoo has an id, an administrator, a name and several animals. An animal has an id, a name and date of birth; it can eat various foods, the latter consisting of an id and a name. The system stores the daily quota (integer number) for each animal and food, e.g., animal A1 <food F1, 10; food F2, 5>; animal A2 <food F2, 1; food F5, 2>. A visitor is characterized by a personal number (an id), name and age. A visitor can visit several zoos. Such a visit is defined by a unique identifier, a day, the paid price, the visitor's personal number and the zoo id.

Requirements:
1. Write an SQL script that creates the corresponding relational data model. (4p)
2. Implement a stored procedure that receives an animal and deletes all the data about the food quotas for that animal.
(1p)
3. Create a view that shows the ids of the zoos with the smallest number of visits. (2p)
4. Implement a function that lists the ids of the visitors who went to zoos that have at least N animals, where N>=1 is a function parameter. (2p)

(1p of)



1. CREATE TABLE zoos (
zid INT PRIMARY KEY,

```
    Administrator VARCHAR(50),
    Ename VARCHAR(50),
);

    CREATE TABLE Animals (
    Aid INT PRIMARY KEY,
    Aname VARCHAR(50),
    DateofBirth DATE,
    Zid INT NOT NULL,
    FOREIGN KEY (Zid) REFERENCES Zoos(Zid)
);

    CREATE TABLE Food (
    Fid INT PRIMARY KEY,
    Fname VARCHAR(50)
);

    CREATE TABLE FoodQuotas (
    FQid INT PRIMARY KEY,
    Aid INT NOT NULL,
    Fid INT NOT NULL,
    DailyQuota INT,
    FOREIGN KEY (Aid) REFERENCES Animals (Aid),
    FOREIGN KEY (Fid) REFERENCES Food (Fid)
);

    CREATE TABLE Visitors (
    Vid INT PRIMARY KEY,
    Vname VARCHAR(50),
```

```
    VAge INT
);

CREATE TABLE Visits (
Visitid INT PRIMARY KEY,
VisitDate DATE,
Price DECIMAL(10,2),
Vid INT NOT NULL,
Zid INT NOT NULL,
FOREIGN KEY (Vid) REFERENCES Visitors(Vid),
FOREIGN KEY (Zid) REFERENCES Zoos(Zid)
);
```

2. Implement a stored procedure that receives an animal and deletes all the data about the food quotas for that animal.
(1p)

```
CREATE PROCEDURE DeleteAnimalData(
    IN AnimalName VARCHAR(50),
)
BEGIN
    DECLARE Animalid INT;

    SELECT Aid INTO Animalid
    FROM Animals
    WHERE Aname = AnimalName;

    DELETE FROM FoodQuotas
    WHERE Aid = Animalid;

    DELETE FROM Animals
```

```
    WHERE Aid = AnimalId;
END
```

3. Create a view that shows the ids of the zoos with the smallest number of visits. (2p)

```
CREATE VIEW ShowZoos AS
 SELECT z.zid
 FROM Zoos z
 LEFT JOIN Visits V on z.zid = V.zid
 GROUP BY z.zid
 HAVING COUNT(V.Visitid) = (
        SELECT COUNT(V2.Visitid)
        FROM Visits V2
        GROUP BY V2.zid
        ORDER BY COUNT(V2.Visitid) ASC
        LIMIT 1
);
```

4. Implement a function that lists the ids of the visitors who went to zoos that have at least **N** animals, where **N>=1** is a function parameter. (2p) 1p

(1p of)

```
CREATE FUNCTION VisitorsWithZoos (N INT)
RETURNS TABLE
RETURN
    SELECT V.Vid AS Visitorsid
    FROM Visitors V
    JOIN Visits VS on V.Vid = VS.Vid
    JOIN Zoos z ON VS.zid = z.zid
    JOIN Animals a ON z.zid = a.zid
    GROUP BY V.Vid, z.zid
    HAVING COUNT(DISTINCT a.Aid) >= N;
```