The exam is divided in four sections. All sub-questions are worth approximately the same amount of points. However, some of these will only take you a minute, some require a bit more thinking, and a couple would require weeks to answer perfectly. Make the best use of your time.

1 Active Databases (3 pt)

A university uses for its data warehouse the relational database shown in Fig. 1.

![Relational schema of the university application](image-url)

Figure 1: Relational schema of the university application

Questions

Write the code ensuring the following integrity constraints. Whenever multiple triggers are needed to enforce a single integrity constraint, list all of them, but write the code in full for only one of them. Throughout the entire question you should provide at least one example for each of ON INSERT, ON UPDATE, and ON DELETE triggers.

1. The intervals defining the lifespan of a professor are disjoint.

    CREATE TRIGGER ProfessorLifespan_OverlappingIntervals
    ON ProfessorLifespan AFTER INSERT, UPDATE AS
    IF EXISTS (SELECT * FROM inserted P1

WHERE 1 < ( 
    SELECT COUNT(*)
    FROM ProfessorLifespan P2
    WHERE P1.ProfessorKey = P2.ProfessorKey AND
    P1.FromDate < P2.ToDate AND
    P2.FromDate < P1.ToDate ) )
BEGIN
RAISERROR ('Overlapping intervals in lifespan of professor', 1, 1)
ROLLBACK TRANSACTION
END

2. The time frame of the temporal attribute Status must be included in the lifespan of Professor.
(Note that we still assume that a professor's lifespans are disjoint.)

CREATE TRIGGER ProfessorStatus_IntervalInLifespan
ON ProfessorStatus AFTER INSERT, UPDATE AS
IF EXISTS ( 
    SELECT *
    FROM INSERTED P1
    WHERE NOT EXISTS ( 
        SELECT *
        FROM ProfessorLifespan P2
        WHERE P1.ProfessorKey = P2.ProfessorKey AND
        P2.FromDate <= P1.FromDate AND
        P1.ToDate <= P2.ToDate ) )
BEGIN
RAISERROR ('Time frame of status is not contained in lifespan of professor', 1, 1)
ROLLBACK TRANSACTION
END

3. The lifespan of the relationship between a professor and a department must be covered by the respective lifespans of the involved professor and department.

CREATE TRIGGER ProfessorDepartment_LifespanInProfessor_1
ON ProfessorDepartment AFTER INSERT, UPDATE AS
IF EXISTS ( 
    SELECT *
    FROM INSERTED PD
    WHERE NOT EXISTS ( 
        SELECT *
        FROM ProfessorLifespan P
        WHERE PD.ProfessorKey = P.ProfessorKey AND
        P.FromDate <= PD.FromDate AND
        PD.ToDate <= P.ToDate ) )
BEGIN
RAISERROR ('Lifespan of relationship is not contained in lifespan of professor', 1, 1)
ROLLBACK TRANSACTION
END

CREATE TRIGGER ProfessorDepartment_LifespanInProfessor_2
ON ProfessorLifespan AFTER UPDATE, DELETE AS
IF EXISTS ( 
    SELECT *
    FROM ProfessorDepartment PD
    WHERE PD.ProfessorKey IN
    ( SELECT ProfessorKey FROM DELETED )
    AND NOT EXISTS ( 
        SELECT *
        FROM ProfessorLifespan P
        WHERE PD.ProfessorKey = P.ProfessorKey AND
        P.FromDate <= PD.FromDate AND
        PD.ToDate <= P.ToDate ) )
BEGIN
RAISERROR ('Lifespan of relationship is not contained in lifespan of professor', 1, 1)
ROLLBACK TRANSACTION
END
WHERE PD.ProfessorKey = P.ProfessorKey AND 
P.FromDate <= PD.FromDate AND 
PD.ToDate <= P.ToDate 
)
BEGIN
    RAISERROR ( 
        'Lifespan of relationship is not contained in lifespan of professor', 1, 1) 
    ROLLBACK TRANSACTION 
END

CREATE TRIGGER ProfessorDepartment_LifespanInDepartment_1 
ON ProfessorDepartment AFTER INSERT, UPDATE AS 
IF EXISTS ( 
    SELECT * 
    FROM INSERTED PD 
    WHERE NOT EXISTS ( 
        SELECT * 
        FROM DepartmentLifespan D 
        WHERE PD.DepartmentKey = P.DepartmentKey AND 
        D.FromDate <= PD.FromDate AND 
        PD.ToDate <= D.ToDate ) 
)
BEGIN
    RAISERROR ( 
        'Lifespan of relationship is not contained in lifespan of department', 1, 1) 
    ROLLBACK TRANSACTION 
END

CREATE TRIGGER ProfessorDepartment_LifespanInDepartment_2 
ON DepartmentLifespan AFTER UPDATE, DELETE AS 
IF EXISTS ( 
    SELECT * 
    FROM ProfessorDepartment PD 
    WHERE PD.DepartmentKey IN 
        ( SELECT DepartmentKey FROM DELETED ) 
    AND NOT EXISTS ( 
        SELECT * 
        FROM DepartmentLifespan D 
        WHERE PD.DepartmentKey = D.DepartmentKey AND 
        P.FromDate <= PD.FromDate AND 
        PD.ToDate <= P.ToDate ) 
)
BEGIN
    RAISERROR ( 
        'Lifespan of relationship is not contained in lifespan of department', 1, 1) 
    ROLLBACK TRANSACTION 
END

4. Professors participate in teaching only during their lifespan.

CREATE TRIGGER Teaching_StartDate_EndDate_In_ProfessorLifespan 
ON Teaching AFTER INSERT, UPDATE AS 
IF EXISTS ( 
    SELECT * 
    FROM INSERTED S 
    WHERE NOT EXISTS ( 
        SELECT * 
        FROM ProfessorLifespan P , Time T1, Time T2 
        WHERE S.ProfessorKey = P.ProfessorKey AND 
        S.StartDateKey = T1.TimeKey AND 
        S.EndDateKey = T2.TimeKey AND 
        P.FromDate <= T1.Date AND 
        T2.Date < P.ToDate ) 
)
BEGIN
    RAISERROR ( 
        'Lifespan of relationship is not contained in lifespan of professor', 1, 1) 
    ROLLBACK TRANSACTION 
END
'Interval of teaching is not contained in lifespan of professor', 1, 1)
ROLLBACK TRANSACTION
END
2 Temporal Databases (7 pt)

Consider the relational schema given in Fig. 2, which is used for analyzing car insurance policies.

![Relational schema for analyzing car insurance policies](image)

**Questions**

Write the following SQL queries.

1. Give the total policy amount per coverage and month (considering the situation on each month’s first day).

```sql
SELECT C.CoverageName, T.Year, T.MonthNumber, 
       FORMAT(SUM(P.Amount), '$##,##0.00') AS TotalAmount 
FROM Policy P, Time T1, Time T2, Time T, 
    PolicyTypeCoverage PTC, Coverage C 
WHERE T1.TimeKey = P.FromDateKey AND 
    T2.TimeKey = P.ToDateKey AND 
    PTC.PolicyTypeKey = P.PolicyTypeKey AND 
    PTC.CoverageKey = C.CoverageKey AND 
    T.Date >= T1.Date AND T.Date < T2.Date AND 
    T.DayNbMonth = 1 
GROUP BY C.CoverageName, T.Year, T.MonthNumber 
ORDER BY C.CoverageName, T.Year, T.MonthNumber
```

2. For each vehicle, give the periods during which it was covered by a policy.

```sql
WITH VehicleCovered(VehicleKey, FromDate, ToDate) AS ( 
    SELECT VehicleKey, T1.Date, T2.Date 
    FROM Vehicle V, Time T1, Time T2 
    WHERE V.VehicleKey = T1.VehicleKey AND 
    V.VehicleKey = T2.VehicleKey AND 
    T1.Date <= T2.Date AND T2.Date - T1.Date + 1 = T1.DayNbWeek 
    ORDER BY VehicleKey, T1.Date, T2.Date
)
```
FROM Policy P, Time T1, Time T2
WHERE P.FromDateKey = T1.TimeKey AND
P.ToDateKey = T1.TimeKey,
SELECT DISTINCT F.VehicleKey, F.FromDate, L.ToDate
FROM VehicleCovered F, VehicleCovered L
WHERE F.VehicleKey = L.VehicleKey AND F.FromDate < L.ToDate
AND NOT EXISTS ( SELECT *
FROM VehicleCovered C
WHERE F.VehicleKey = C.VehicleKey AND
F.FromDate < C.FromDate AND
C.FromDate <= L.ToDate AND NOT EXISTS ( SELECT *
FROM VehicleCovered C1
WHERE F.VehicleKey = C1.VehicleKey AND
C1.FromDate < C.FromDate AND
C.FromDate <= C1.ToDate ) )
AND NOT EXISTS ( SELECT *
FROM VehicleCovered E
WHERE F.VehicleKey = E.VehicleKey AND
( (E.FromDate < F.FromDate AND F.FromDate <= E.ToDate) 
OR (E.FromDate <= L.ToDate AND L.ToDate < E.ToDate) ) )

3. For each vehicle, give the total policy amount in the periods during which the vehicle was driven by only one driver.

WITH VehicleOneDriver(VehicleKey, FromDate, ToDate) AS (
-- Case 1
SELECT V1.VehicleKey, V1.FromDate, V2.FromDate
FROM VehicleDriver V1, VehicleDriver V2
WHERE V1.VehicleKey = V2.VehicleKey AND
V1.FromDate < V2.FromDate AND
V2.FromDate < V1.ToDate AND NOT EXISTS ( SELECT *
FROM VehicleDriver V3
WHERE V1.VehicleKey = V3.VehicleKey AND
V1.FromDate < V3.ToDate AND
V3.FromDate < V2.FromDate )
UNION
-- Case 2
SELECT V1.VehicleKey, V2.ToDate, V1.ToDate
FROM VehicleDriver V1, VehicleDriver V2
WHERE V1.VehicleKey = V2.VehicleKey AND
V1.FromDate < V2.ToDate AND
V2.ToDate < V1.ToDate AND NOT EXISTS ( SELECT *
FROM VehicleDriver V3
WHERE V1.VehicleKey = V3.VehicleKey AND
V2.ToDate < V3.ToDate AND
V3.FromDate < V1.ToDate )
UNION
-- Case 3
SELECT V1.VehicleKey, V2.ToDate, V3.FromDate
FROM VehicleDriver V1, VehicleDriver V2, VehicleDriver V3
WHERE V1.VehicleKey = V2.VehicleKey AND
V1.VehicleKey = V3.VehicleKey AND
V2.ToDate < V3.FromDate AND
V1.FromDate < V2.ToDate AND
V3.FromDate < V1.ToDate AND NOT EXISTS (
SELECT *
FROM VehicleDriver T4
WHERE V1.VehicleKey = T4.VehicleKey AND
V2.ToDate < T4.ToDate AND
T4.FromDate < V3.FromDate )
UNION
-- Case 4
SELECT VehicleKey, FromDate, ToDate
FROM VehicleDriver V1
WHERE NOT EXISTS (
    SELECT *
    FROM VehicleDriver V2
    WHERE V1.VehicleKey = V2.VehicleKey AND
    V1.DriverKey <> V2.DriverKey AND
    V1.FromDate < V2.ToDate AND
    V2.FromDate < V1.ToDate ) ),
VehicleOneDriverCoalesced(VehicleKey, FromDate, ToDate) AS ( 
    -- Coalescing the table VehicleOneDriver above ...
)
SELECT VehicleKey, dbo.MaxDate(O.FromDate, T1.Date) AS FromDate,
    dbo.MinDate(O.ToDate, T2.Date) AS ToDate,
    FORMAT(SUM(Amount), '$###,##0.00') AS TotalAmount
FROM Policy P, Time T1, Time T2, VehicleOneDriverCoalesced O
WHERE P.FromDateKey = T1.TimeKey AND P.ToDateKey = T2.TimeKey AND
    P.VehicleKey = O.VehicleKey AND
    dbo.MaxDate(O.FromDate, T1.Date) < dbo.MinDate(O.ToDate, T2.Date)
GROUP BY VehicleKey, dbo.MaxDate(O.FromDate, T1.Date),
    dbo.MinDate(O.ToDate, T2.Date)
ORDER BY VehicleKey, dbo.MaxDate(O.FromDate, T1.Date)

4. Give the monthly number of policies by customer.

WITH Month(FromDate, ToDate) AS ( 
    SELECT MIN(Date), DateAdd(month, 1, MIN(Date))
    FROM Time
    GROUP BY Year, MonthNumber ),
PolicyChanges(CustomerKey, Day) AS ( 
    SELECT CustomerKey, T.Date
    FROM Policy F, Time T
    WHERE F.FromDateKey = T.TimeKey
UNION
    SELECT CustomerKey, T.Date
    FROM Policy F, Time T
    WHERE F.ToDateKey = T.TimeKey
UNION
    SELECT CustomerKey, FromDate
    FROM CustomerLifespan C
UNION
    SELECT CustomerKey, ToDate
    FROM CustomerLifespan C ),
PolicyPeriods(CustomerKey, FromDate, ToDate) AS ( 
    SELECT T1.CustomerKey, T1.Day, T2.Day
    FROM PolicyChanges T1, PolicyChanges T2
    WHERE T1.CustomerKey = T2.CustomerKey AND
    T1.Day < T2.Day AND NOT EXISTS ( 
    SELECT *
    FROM PolicyChanges T3
    WHERE T1.CustomerKey = T3.CustomerKey AND
PolicyCount(CustomerKey, NoPolicies, FromDate, ToDate) AS (
SELECT P.CustomerKey, COUNT(*), PFromDate, PToDate
FROM Policy F, Time T1, Time T2, PolicyPeriods P
WHERE FFromDateKey = T1.TimeKey AND
  FToDateKey = T2.TimeKey AND
  FCustomerKey = P.CustomerKey AND
  T1.Date <= PFromDate AND PToDate <= T2.Date
GROUP BY P.CustomerKey, PFromDate, PToDate
UNION
SELECT P.CustomerKey, 0, PFromDate, PToDate
FROM PolicyPeriods P
WHERE NOT EXISTS (SELECT *
    FROM Policy F, Time T1, Time T2,
    WHERE FFromDateKey = T1.TimeKey AND
      FToDateKey = T2.TimeKey AND
      FCustomerKey = P.CustomerKey AND
      T1.Date <= PFromDate AND PToDate <= T2.Date ),
PolicyCountCoalesced(CustomerKey, NoPolicies, FromDate, ToDate) AS (    -- Coalescing the table PolicyCount above
    ...
),
SELECT CustomerName, NoPolicies,
    dbo.MaxDate(MFromDate, CFromDate) AS FromDate,
    dbo.MinDate(MToDate, CToDate) AS ToDate
FROM Month M, PolicyCountCoalesced C, Customer U
WHERE C.CustomerKey = U.CustomerKey AND
    dbo.MaxDate(MFromDate, CFromDate) < dbo.MinDate(MToDate, CToDate)
ORDER BY CustomerName, dbo.MaxDate(MFromDate, SFromDate)
3 Object Databases (5 pt)

Consider the diagram of a French horse race application given in Fig. 3.

![Entity-relationship diagram of a French horse race application](image)

Here is part of the associated types in Oracle:

```sql
CREATE TYPE TPerson;
CREATE TYPE TJockey;
CREATE TYPE THorse;
CREATE TYPE TParticipates;
CREATE TYPE THorseRace;

CREATE TYPE TSetRefHorses AS TABLE OF REF THorse;
CREATE TYPE TSetRefParticipates AS TABLE OF REF TParticipates;

CREATE OR REPLACE TPerson AS OBJECT(
    personKey INTEGER,
    firstName VARCHAR2(50),
    lastName VARCHAR2(50),
    sex ENUM('male', 'female'),
    birthDate DATE,
    address VARCHAR2(100),
    trains TSetRefHorses,
    owns TSetRefHorses);

We impose that each relationship can be traversed by queries in both directions.

Questions

1. Write the type definitions for TJockey, THorse, THorseRace, and TParticipates.

   CREATE OR REPLACE TJockey UNDER TPerson (
        height INTEGER,
        participates TSetRefParticipates);

   CREATE OR REPLACE THorse AS OBJECT (
horseKey INTEGER,
horseName VARCHAR2(50),
breed VARCHAR2(20),
sex ENUM('male', 'female'),
birthDate DATE,
geldingDate DATE,
deathDate DATE,
coatColor VARCHAR2(10),
participatesIn TSetRefParticipates,
trainedBy REF TPerson,
ownedBy REF TPerson);

CREATE OR REPLACE TRacePrize AS OBJECT (  
    placeNo INTEGER
    amount NUMBER);

CREATE OR REPLACE TRacePrizes AS TABLE OF TRacePrize;

CREATE OR REPLACE THorseRace AS OBJECT (  
    raceNumber INTEGER,
raceDate DATE,
hippodrome VARCHAR2(30),
raceName VARCHAR2(50),
distance INTEGER,
raceType ENUM('flat', 'obstacle'),
prizes TRacePrizes,
participatedBy TSetRefParticipates);

CREATE OR REPLACE TParticipates AS OBJECT (  
    jockey REF TJockey,
horse REF THorse,
weight NUMBER,
arrivalPlace INTEGER,
margin NUMBER,
participatesIn REF THorseRace);

2. Write the query that returns:

(a) the name of the owners who participated as jockey in at least one race

```
SELECT P.firstName, P.lastName
FROM Person P
WHERE VALUE(P) IS OF (TJockey)
AND EXISTS (SELECT * FROM TABLE(P.owns))
AND EXISTS (SELECT * FROM TABLE(TREAT(VALUE(P) AS TJockey).participates);
```

(b) for each jockey, the name of the races in which he/she ran during 2014, ordered by descending date, and, for each of these races, its finishing place and the resulting gain

```
SELECT P.firstName, P.lastName,  
    VALUE(HR).raceName, VALUE(HR).raceDate,  
    VALUE(RP).arrivalPlace,  
    (SELECT VALUE(HRP).amount  
        FROM TABLE(VALUE(HR).prizes) HRP  
        WHERE VALUE(HRP).placeNo = P.arrivalPlace) AS amount
FROM Person P,  
    TABLE(TREAT(VALUE(P) AS TJockey).participates) RP,  
    TABLE(VALUE(RP).participatesIn) HR  
WHERE VALUE(P) IS OF (TJockey)  
    YEAR(VALUE(HR).raceDate) = 2014  
ORDER BY P.lastName, P.firstName,
```
(c) for each horse and each race type (flat or obstacle), the number of times the horse arrives in the first position and the total gain in the first position.

```sql
SELECT VALUE(H).horseName, VALUE(HR).raceType,
    COUNT(*) AS NbTimesFirst,
    SUM(TMP.amount) AS TotalGain
FROM Participates P,
    TABLE(VALUE(P).horse) H,
    TABLE(VALUE(P).participatesIn) HR,
    (SELECT VALUE(HRP).amount
    FROM TABLE(VALUE(HR).prizes) HRP
    WHERE placeNo = 1) TMP
AND P.arrivalPlace = 1
GROUP BY VALUE(H).horseName, VALUE(HR).raceType
ORDER BY VALUE(H).horseName, VALUE(HR).raceType
```
4 Spatial Databases (5 pt)

Consider a spatial data warehouse whose relational schema is given in Fig. 4.

Figure 4: Relational schema of the spatial data warehouse

Write in SQL the following queries.

1. Display (by means of one single query) the sales figures (store sales, store costs, and unit sales) summarized for every store that is either:
   - located in the state of California and less than 200 km from Los Angeles, or
   - located in the state of Washington and less than 200 km from Seattle.

   ```sql
   SELECT S.store_id, SUM(S.store_sales) AS TotalStoreSales,
           SUM(S.store_cost) AS TotalStoreCost, SUM(S.unit_sales) AS TotalUnitSales
   FROM Sales S, Store O, City C, State A, City C1, City C2
   WHERE S.store_id = O.store_id AND O.store_city_id = C.city_id AND
       C.state_id = A.state_id AND C1.city_name = 'Los Angeles' AND
       C2.city_name = 'Seattle' AND
       ( ( C.state_name = 'California' AND
           ST_Distance(O.store_geom,C1.city_geom) < 200 ) OR
       ( C.state_name = 'Washington' AND
           ST_Distance(O.store_geom,C2.city_geom) < 200 ) )
   GROUP BY S.store_id
   ```

2. Display the total sales of stores that are located less than 5 km from the city center against total sales for all stores in their state.

   ```sql
   WITH SalesState(state_id, sales_state) AS (...
   ```
SELECT A.state_id, SUM(S.store_sales) AS sales_5Km, sales_state
FROM Sales S, Store O, City C, State A
WHERE S.store_id = O.store_id AND O.store_city_id = C.city_id AND C.state_id = A.state_id
GROUP BY A.state_id
)
SELECT S.store_id, SUM(store_sales) AS sales_5Km, sales_state
FROM Sales S, Store O, City C, SalesState SS
WHERE S.store_id = O.store_id AND O.store_city_id = C.city_id AND ST_Distance(O.store_geom,C.city_geom) < 5 AND C.state_id = SS.state_id
GROUP BY S.store_id

3. For each store list total store sales to customers living closer than 10 km to the store, against total sales for the store.

WITH TotalSales AS (
  SELECT S.store_id, SUM(store_sales) AS total_sales
  FROM Sales S, Store O
  WHERE S.store_id = O.store_id
  GROUP BY S.store_id
)
SELECT S.store_id, SUM(store_sales) AS sales_10Km, TS.total_sales
FROM Sales S, Customer U, Store O, TotalSales TS
WHERE S.customer_id = U.customer_id AND S.store_id = O.store_id AND S.store_id = TS.store_id AND ST_Distance(O.store_geom,U.customer_geom) < 10
GROUP BY S.store_id

4. For each city give the store closest to the city center and its the best sold brand name.

WITH ClosestStore AS (
  SELECT C.city_id, O.store_id
  FROM City C, Store O
  WHERE ST_Distance(C.city_geom,O.store_geom) <= ALL ( SELECT ST_Distance(C.city_geom, O1.store_geom) FROM Store O1 ) ),
BrandSales AS ( SELECT CS.store_id, P.brand_name, SUM(store_sales) AS brand_sales
  FROM Sales S, ClosestStore CS, Product P
  WHERE S.store_id = CS.store_id AND S.product_id = P.product_id
  GROUP BY CS.store_id, P.brand_name ),
TopBrandSales AS ( SELECT BS.store_id, BS.brand_name
  FROM BrandSales BS
  WHERE BS.brand_sales >= ALL ( SELECT BS1.brand_sales
    FROM BrandSales BS1
    WHERE BS.store_id = BS1.store_id ) ),
SELECT CS.city_id, CS.store_id, TS.brand_name
FROM ClosestStore CS, TopBrandSales TS
WHERE CS.store_id = TS.store_id

Note. You might need some of the following PostGIS functions:

- ST_Distance(geometry1,geometry2) returns the distance between the two geometries.