Data Bases II
Trigger

Michele Beretta

michele.beretta@unibg.it

Exercise T.1

```
ClientOrder(OrderId, ProductId, Qty, ClientId, TotalSubItems)
ProductionProcess(ProdProcId, ObtainedProdId, StartingProdId, Qty, ProcessDuration, ProductionCost)
ProductionPlan(BatchId, ProdProcId, Qty, OrderId)
PurchaseOrder(PurchaseId, ProdId, OrderId)
```

The relational database above supports the production systems of a factory. Table `ProductionProcess` describes how a product can be obtained by (possibly several) other products, which can be themselves obtained from other products from outside.

Build a trigger system that reacts to the insertion of orders from clients and creates new items in `ProductionPlan` or in `PurchaseOrder`, depending on the ordered product, so as to manage the client's order (for the generation of the identifiers, use a function `GenerateId()`).

The triggers should also update the value of `TotalSubItems` (initially always set to 0) to describe the number of sub-products (internally produced or outsourced) that are used overall in the production plan deriving from the order.

Also briefly discuss the termination of the trigger system.

Exercise T.1 – Solution

```
CREATE TRIGGER NewClientOrder
AFTER INSERT ON ClientOrder
FOR EACH ROW
DECLARE
  _ProcId := NULL;
  _Qty    := 0;
BEGIN
  SELECT ProdProcId INTO _ProcId, Qty INTO _Qty
  FROM ProductionProcess
  WHERE ObtainedProdId = NEW.ProductId;

  IF _ProcId IS NOT NULL THEN
    INSERT INTO ProductionPlan
    VALUES (GenerateId(), _ProcId, _Qty * NEW.Qty, NEW.OrderId)
  ELSE
    INSERT INTO PurchaseOrder
    VALUES (GenerateId(), NEW.ProductId, NEW.Qty, NEW.OrderId)
  END IF;
END;
```

Exercise T.1 – Solution

```
CREATE TRIGGER UpdateSubItemsAfterPurchase
AFTER INSERT ON PurchaseOrder
FOR EACH ROW
BEGIN
  UPDATE ClientOrder
  SET TotalSubItems = TotalSubItems + NEW.Qty
  WHERE OrderId = NEW.OrderId;
END;
```

```
CREATE TRIGGER UpdateSubItemsAfterProduction
AFTER INSERT ON PurchaseOrder
FOR EACH ROW
BEGIN
  UPDATE ClientOrder
  SET TotalSubItems = TotalSubItems + NEW.Qty
  WHERE OrderId = NEW.OrderId;
END;
```

Exercise T.1 – Solution

```
CREATE TRIGGER NewProductionPlan
AFTER INSERT ON ProductionPlan
FOR EACH ROW
DECLARE
  _WantedProd := NULL;
  _Qty        := 0;
BEGIN
  SELECT StartingProdId INTO _WantedProd, Qty INTO _Qty
  FROM ProductionProcess
  WHERE ProdProcId = NEW.ProdProcId;

  IF _WantedProd IS NOT NULL THEN
    INSERT INTO ProductionPlan
    VALUES (GenerateId(), _WantedProd, _Qty * NEW.Qty, NEW.OrderId)
  ELSE
    INSERT INTO PurchaseOrder
    VALUES (GenerateId(), _WantedProd, _Qty, NEW.OrderId)
  END IF;
END;
```

Exercise T.2

Si consideri il seguente schema relativo a un sistema di noleggio di sale prove per gruppi musicali

```
Cliente(CodiceFiscale, Nome, Cognome, Tipo)
Prenotazione(CodFisCliente, CodiceSala, Giorno, OraInizio, OraFine)
UsoEffettivo(CodFisCliente, CodiceSala, Giorno, OraInizio, OraFine, Costo)
Sala(Codice, CostoOrario)
```

L'uso effettivo può avvenire solo in corrispondenza di una prenotazione, e al limite iniziare dopo o terminare prima degli orari della prenotazione

1. Si scriva un trigger che impedisca di prenotare una sala già prenotata
2. Si supponga che i dati sull'uso effettivo siano inseriti solo al termine dell'uso della sala. Si propongano un arricchimento dello schema per regole che assegni il valore `"Inaffidabile"` al campo `tipo` dei clienti all'accumulo di 50 ore prenotate e non usate.

Exercise T.2 – Solution

Part 1.

```
CREATE TRIGGER CantBook
BEFORE INSERT ON Prenotazione
FOR EACH ROW
WHEN EXISTS (
  SELECT *
  FROM Prenotazione
  WHERE CodiceSala = NEW.CodiceSala
    AND Giorno = NEW.Giorno
    AND (
         NEW.OraInizio BETWEEN OraInizio AND OraFine
      OR NEW.OraFine   BETWEEN OraInizio AND OraFine
    )
)
ROLLBACK;
```

Exercise T.2 – Solution

Part 2. Aggiungiamo `OreSprecate` a `Cliente`.

```
CREATE TRIGGER AggiornaOreSprecate
AFTER INSERT INTO UsoEffettivo
FOR EACH ROW
BEGIN
  UPDATE Cliente
  SET OreSprecate = OreSprecate + (
    SELECT (OraFine - OraInizio) - (NEW.OraFine - NEW.OraInizio)
    FROM Prenotazione
    WHERE CodFisCliente = NEW.CodFisCliente
      AND CodiceSala    = NEW.CodiceSala
      AND Giorno        = NEW.Giorno
  )
  WHERE CodiceFiscale = NEW.CodiceFisCliente
END;
```

Exercise T.2 – Solution

Gestiamo i clienti che non si sono mai presentati (supponiamo ci sia un evento a fine giornata).

```
CREATE TRIGGER AggiornaOreSprecate2
AFTER End_Day
BEGIN
  UPDATE Cliente C
  SET OreSprecate = OreSprecate + (
    SELECT SUM(OraFine - OraInizio)
    FROM Prenotazione P
    WHERE P.CodFisCliente = C.CodiceFiscale
      AND (P.CodFiscaleCliente, CodiceSala, Giorno) NOT IN (
        SELECT CodFiscaleCliente, CodiceSala, Giorno
        FROM UsoEffettivo U
        WHERE U.OraInizio BETWEEN P.OraInizio AND P.OraFine
          AND U.CodFisCliente = C.CodiceFiscale
          AND U.CodiceSala = P.CodiceSala
          AND U.Giorno = P.Giorno
      )
  )
END;
```

Exercise T.2 – Solution

Aggiorniamo il `Tipo` del cliente.

```
CREATE TRIGGER AggiornaTipo
AFTER UPDATE OF OreSprecate ON Cliente
FOR EACH ROW
WHEN OLD.OreSprecate < 50 AND NEW.OreSprecate >= 50
BEGIN
  UPDATE Cliente
  SET Tipo = "Inaffidabile"
  WHERE CodiceFiscale = NEW.CodiceFiscale
END;
```

Exercise T.3

```
Possiede(Societa1, Societa2, Percentuale)
```

Si consideri, per semplicità, che le tuple della tabella `Possiede` siano inserite a partire da una tabella vuota, che poi non viene più modificata.

Si costruisca tramite trigger la relazione `Controlla` (una società A controlla una società B se A possiede direttamente o indirettamente più del 50% di B).

Si assuma che le percentuali di controllo siano numeri decimali (tra 0 e 1), e si tenga presente che la situazione di controllo avviene in modo diretto quando una società possiede più del 50% della “controllata” o indiretto quando una società controlla altre società che possiedono percentuali di B e la somma delle percentuali possedute e controllate da A supera il 50%.

Exercise T.3 – Solution

Vogliamo creare:

```
Controlla(SocControllante, SocControllata)
```

A può controllare B in due modi distinti:

1. Direttamente: ne possiede più del 50%
2. Indirettamente: la somma della percentuale di possesso diretto e del possesso mediato (tramite altre società controllate) è superiore al 50%, anche se nessuna delle singole percentuali è sopra al 50%

Exercise T.3 – Solution

```
CREATE TRIGGER InsPossiede
AFTER INSERT INTO Possiede
FOR EACH ROW
BEGIN
  IF NEW.Percentuale > 0.5 THEN
    INSERT INTO Controlla
    VALUES (NEW.Societa1, NEW.Societa2)
  END IF;
END;
```

Exercise T.3 – Solution

```
CREATE VIEW ControlloIndiretto(Societa1, Societa2, Percentuale) AS
SELECT C.SocControllante, P.Societa2, SUM(P.Percentuale)
FROM Possiede P
    JOIN Controlla C ON P.Societa1 = C.SocControllata
WHERE (C.SocControllante, P.Societa2) NOT IN (SELECT * FROM Controlla)
GROUP BY C.SocControllante, P.Societa2
```

```
CREATE TRIGGER UpdControlla
AFTER INSERT INTO Controlla
FOR EACH ROW
BEGIN
  INSERT INTO Controlla
  SELECT C.Societa1, C.Societa2
  FROM ControlloIndiretto C
    LEFT JOIN Possiede P ON C.Societa1 = P.Societa2 AND C.Societa2 = P.Societa2
  WHERE C.Societa1 = NEW.SocControllante
    AND ((P.Percentuale IS NULL AND C.Percentuale > 0.5) OR P.Percentuale + C.Percentuale > 0.5)
END;
```

Exercise T.4 – Transazioni

Dato il seguente schema:

```
Titolo(CodTitolo, Nome, Tipologia)
Transazione(CodTrans, CodVenditore, CodAcquirente, CodTitolo, Qta, Valore, Data, Istante)
Operatore(Codice, Nome, Indirizzo, Disponibilità)
```

Costruire un sistema di trigger che mantenga aggiornato il valore di `Disponibilità` di `Operatore` in seguito all'inserimento di tuple in `Transazione`, tenendo conto che per ogni transazione in cui l'operatore vende l'ammontare della transazione deve far crescere la disponibilità e per ogni acquisto deve invece diminuire.

Inserire inoltre gli operatori la cui disponibilità scende sotto lo zero in una tabella che elenca gli operatori "scoperti".

```
Scoperto(Codice, Nome, Indirizzo)
```

Exercise T.4 – Solution

```
CREATE TRIGGER InsTransazione
AFTER INSERT ON Transazione
FOR EACH ROW
BEGIN
  UPDATE Operatore
  SET Disponibilita = Disponibilita - NEW.Qta
  WHERE Codice = NEW.CodVenditore;

  UPDATE Operatore
  SET Disponibilita = Disponibilita + NEW.Qta
  WHERE Codice = NEW.CodAcquirente;
END;
```

Exercise T.4 – Solution

```
CREATE TRIGGER InsScoperto
AFTER UPDATE OF Disponibilita on Operatore
FOR EACH ROW
WHEN NEW.Disponibilita < 0 AND OLD.Disponibilita >= 0
INSERT INTO Scoperto
VALUES (NEW.Codice, NEW.Nome, NEW.Indirizzo);
```

```
CREATE TRIGGER DelScoperto
AFTER UPDATE OF Disponibilita ON Operatore
FOR EACH ROW
WHEN NEW.Disponibilita >= 0 and OLD.Disponibilita < 0
DELETE FROM Scoperto
WHERE Codice = NEW.Codice;
```

Exercise T.5 – Viaggi

Facendo riferimento alla base dati:

```
Dipendente(Codice, Nome, Qualifica, Settore)
Viaggio(Codice, Dip, Destinazione, Data, Km, TargaAuto)
Auto(Targa, Modello, CostoKm)
Destinazione(Nome, Stato)
```

Si consideri la vista:

```
Viaggi(Dipendente, KmTot, CostoTotale)
```

Scrivere due regole attive che calcolano il valore della vista a seguito di inserzioni di nuovi viaggi, una in modo incrementale e una ricalcolando l'intera vista.

Exercise T.5 – Viaggi

```
CREATE TRIGGER CalcolaVistaIncrementale
AFTER INSERT INTO Viaggio
FOR EACH ROW
BEGIN
  UPDATE Viaggi
  SET KmTot = KmTot + NEW.Km,
      CostoTotale = CostoTotale + (SELECT NEW.Km * CostoKm FROM Auto WHERE Targa = NEW.TargaAuto)
  WHERE Dipendente = NEW.Dip
END;
```

```
CREATE TRIGGER CalcolaVistaOneShot
AFTER INSERT INTO Viaggio
FOR EACH ROW
BEGIN
  DELETE FROM Viaggi;
  INSERT INTO Viaggi
  SELECT Dip, SUM(Km), SUM(Km * CostoKm)
  FROM Viaggio JOIN Auto ON TargaAuto = Targa
  GROUP BY Dip
END;
```

Exercise T.6 – Campus

Dato il seguente schema relazionale:

```
Studenti(Matr, Nome, Residenza, Telefono, CorsoDiLaurea, AnnoCorso, Sede, TotCrediti)
Iscrizioni(MatrStud, Corso, Anno, Data)
CorsiAnni(CodCorso, Anno, Docente, Semestre, NroStudenti, NroFuoriSede)
Abbinamenti(CodCorso, CorsoLaurea)
Corsi(CodCorso, Titolo, Crediti, Sede)
```

1. Scrivere una regola attiva che controlla ogni inserimento di una tupla nella tabella `Iscrizioni` e nel caso in cui non esistano elementi corrispondenti in `Studenti` o `CorsiAnni` (cioè l'iscrizione non sia una iscrizione significativa, perché lo studente è sconosciuto o il corso non attivo) annulli l'inserimento.
2. Supponendo che l'attributo `NroFuoriSede` di `CorsiAnni` rappresenti il numero di studenti iscritti al corso che fanno riferimento a una sede *diversa* da quella del corso, scrivere una regola attiva che reagisce alle modifiche dell'attributo `Sede` di `Studente`, aggiornando se necessario il valore dell'attributo `NroFuoriSede`.

Exercise T.6 – Part 1

```
CREATE TRIGGER InsIscrizione
BEFORE INSERT ON Iscrizioni
WHEN NOT Exists(
  SELECT *
  FROM Studenti
  WHERE Matr = NEW.MatrStud
) OR NOT EXISTS (
  SELECT *
  FROM CorsiAnni
  WHERE CodCorso = NEW.Corso AND Anno = NEW.Anno
)
ROLLBACK;
```

Exercise T.6 – Part 2

```
CREATE TRIGGER UpdSedeStudente
AFTER UPDATE OF Sede ON Studenti
FOR EACH ROW
BEGIN
  UPDATE CorsiAnni
  SET NroFuoriSede = NroFuoriSede - 1
  WHERE CodCorso IN (
    SELECT CodCorso
    FROM Corsi
    WHERE Sede = NEW.Sede
  ) AND (CodCorso, Anno) IN (
    SELECT CodCorso, Anno
    FROM Iscrizioni
    WHERE MatrStud = NEW.Matr
  )

  UPDATE CorsiAnni
  SET NroFuoriSede = NroFuoriSede + 1
  WHERE CodCorso IN (
    SELECT CodCorso
    FROM Corsi
    WHERE Sede = OLD.Sede
  ) AND (CodCorso, Anno) IN (
    SELECT CodCorso, Anno
    FROM Iscrizioni
    WHERE MatrStud = OLD.Matr
  )
END;
```

Due contatori da aggiornare: lo studente diventa “fuori sede” per tutti i suoi corsi erogati nella sede abbandonata, e diventa “in sede” per tutti i suoi corsi erogati nella nuova sede.

Attenzione all’uso di `OLD` e `NEW` per `Sede`. Inoltre, un update che non aggiorna sede lascia invariati i contatori.

Exercise T.7 – Prodotti

Lo schema seguente descrive un insieme gerarchico di prodotti, dove per i prodotti non contenuti in altri prodotti assumiamo `Level` (che descrive il livello di profondità a cui il prodotto è situato nella gerarchia) pari a zero e `SuperProdotto` pari a `NULL`:

```
Product(Code, Name, Description, SuperProduct, Level)
```

1. Scrivere una regola attiva che alla cancellazione di un prodotto cancelli tutti i sottoprodotti corrispondenti
2. Scrivere una regola attiva che, alla creazione di un nuovo prodotto (*eventualmente* figlio di un prodotto esistente), calcoli il valore dell'attributo `Level`
3. Implementare mediante regole attive un vincolo di pseudo-integrità referenziale sull'attributo `SuperProduct`, per cui gli unici valori ammessi sono `NULL` oppure il codice di un altro prodotto (non se stesso)

Exercise T.7 – Part 1

```
CREATE TRIGGER DelSubProd
BEFORE DELETE ON Product
FOR EACH ROW
BEGIN
  DELETE FROM Product
  WHERE SuperProduct = OLD.Code
END;
```

Exercise T.7 – Part 2

```
CREATE TRIGGER InsProd
AFTER INSERT ON Product
FOR EACH ROW
BEGIN
  UPDATE Product
  SET Level = 0
  WHERE Code = NEW.Code AND SuperProduct IS NULL;

  UPDATE Product
  SET Level = 1 + (
    SELECT Level
    FROM Product
    WHERE Code = NEW.SuperProduct
  )
  WHERE Code = NEW.Code AND SuperProduct IS NOT NULL;
END;
```

Exercise T.7 – Part 3

```
CREATE TRIGGER IntegrProd
AFTER INSERT ON Product
FOR EACH ROW
WHEN NEW.SuperProduct IS NOT NULL
  AND (NEW.SuperProduct = NEW.Code OR NEW.SuperProduct NOT IN (SELECT Code FROM Product))
ROLLBACK;
```