ADVANCED DATABASES REPORT

Theater Organization

17 JANVIER 2021

EFREI PARIS
VARATHALINGAM Mayouran – DELPHIN Gilbert

Table of Contents

I. Introduction / M&G

Theater is the art of writing and performing plays but it's also the structure which have rows seats, from where people can watch a performance. In fact, theaters are less and less watched by people due to all other activities deemed more interesting.

So how can all theatrical companies still survive in these years of digital dominance? To face this problem some theatrical companies had the idea of making a common management of all their shows, so their theaters can welcome many other theatrical companies and their staff can play in other places and not only on their theater respectively.

M&G our team is composed of VARATHLINGAM Mayouran and DELPHIN Gilbert, we are main partners for these theaters to create all the needed databases. Our job is to help them to manage correctly this network and optimize it to see all the advantages.

II. Short description of this network

How can these theatrical companies manage this new network?

If a theater want a company to play their show on their place, the theater has to pay the company some costs (comedians' fees , staging costs and travel costs). It's not the case if the theater want it's company to perform (there will be no travel costs)**.**

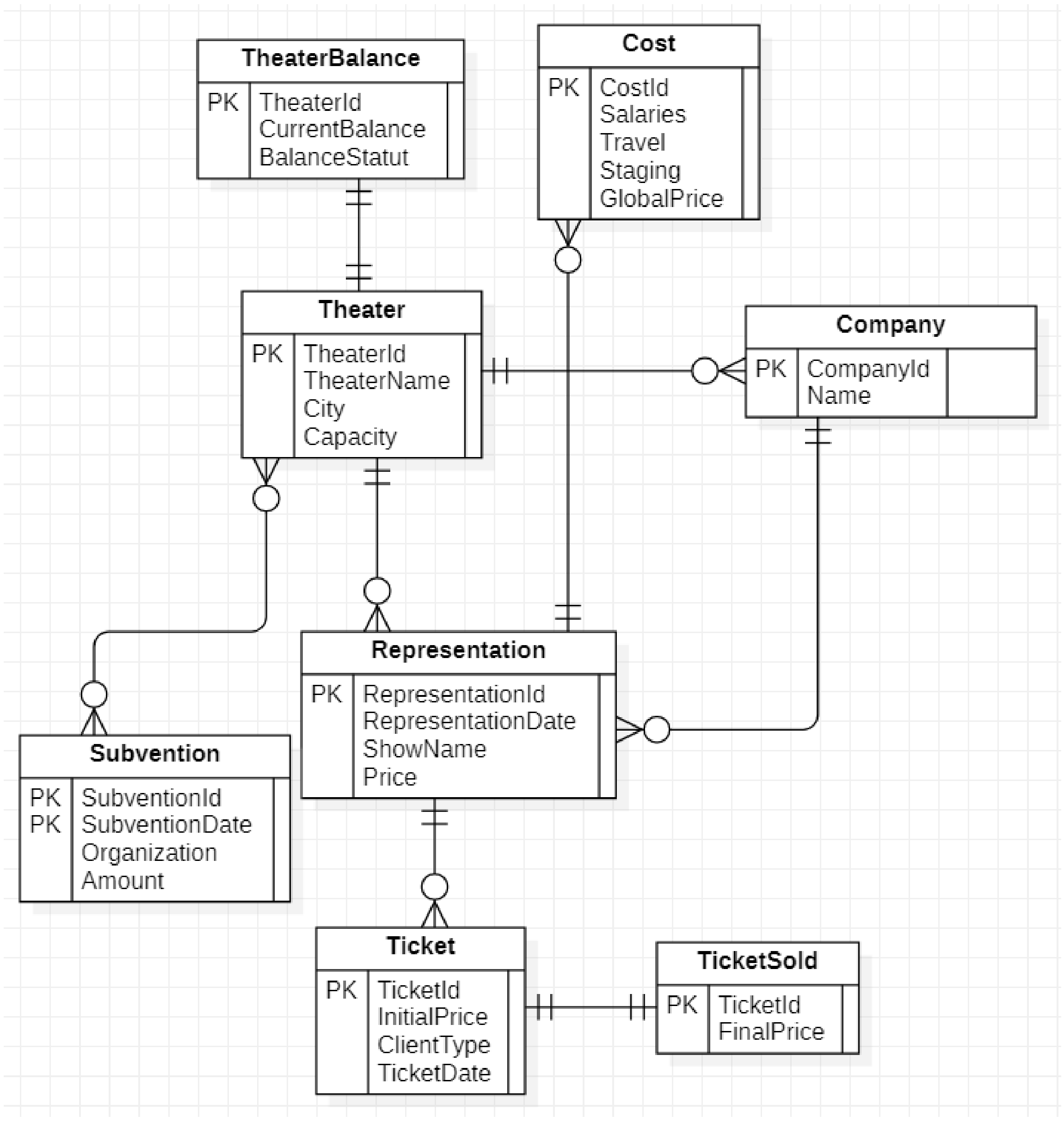
Concerning the ticketing all the revenues from any representation go to the theater where the show took place. For every representation there is an Initial Price and some promotions can be applied depending on the situation of the client (Student , Children..) , days separating the representation date and the ticket sell date or the load factor of the theater for the particular representation.

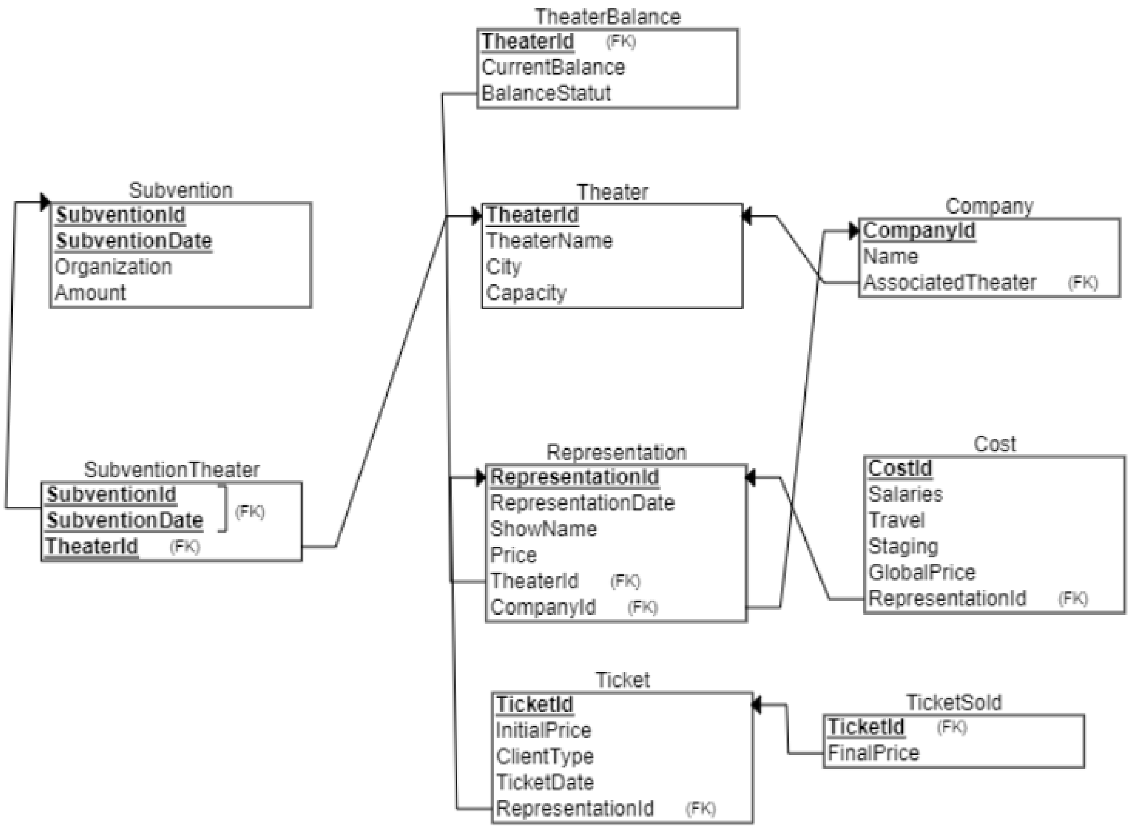
Moreover, some grants are accorded to theaters depending of their cities or supporter. For the case of Municipalities or Ministry of Culture there are negotiated with the theater and have fixed durations.

So, we should consider all these functionalities and add some to correctly manages this network.

III. Modelization/ Functions/ Queries / Triggers / Procedures

I. ERD & Relational Schema




TheaterBalance
TheaterId (FK)
CurrentBalance
BalanceStatut
Subvention
SubventionId
SubventionDate
Organization
Amount
Theater
TheaterId
TheaterName
City
Capacity
Company
CompanyId
Name
AssociatedTheater (FK)
SubventionTheater
SubventionId
SubventionDate
(FK)
TheaterId (FK)
Representation
RepresentationId
RepresentationDate
ShowName
Price
TheaterId (FK)
CompanyId (FK)
Cost
CostId
Salaries
Travel
Staging
GlobalPrice
RepresentationId (FK)
Ticket
TicketId
InitialPrice
ClientType
TicketDate
RepresentationId (FK)
TicketSold
TicketId (FK)
FinalPrice

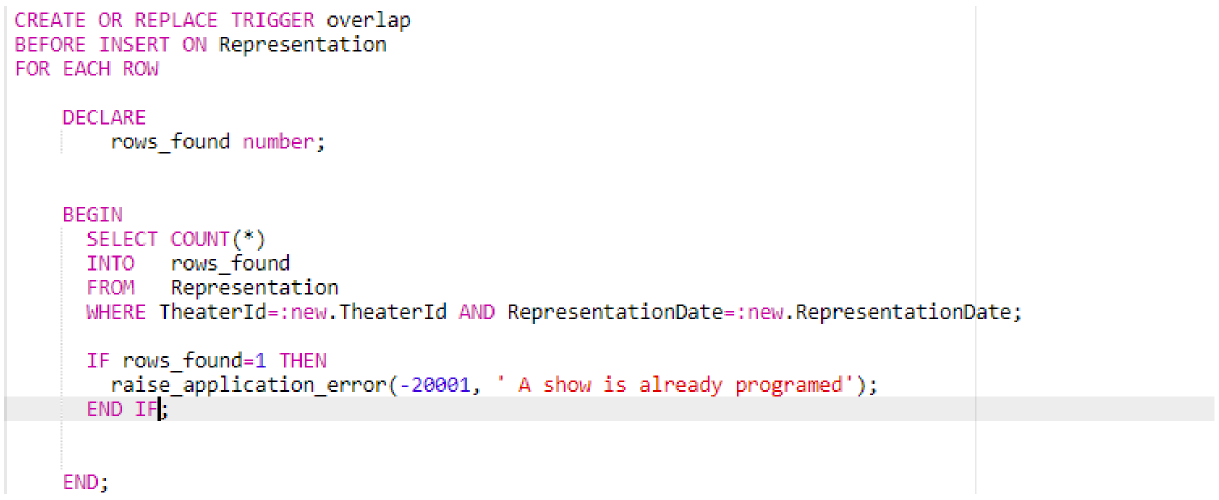
II. Functions / Queries / Triggers / Procedures

We model our network to answers some questions.

1.Organization

Is no representation of a company does not overlap anothor one?

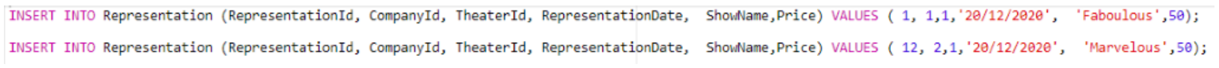
For answering this question, we created a trigger to not allow another show to happened on the same theater at the same date.



```
CREATE OR REPLACE TRIGGER overlap
BEFORE INSERT ON Representation
FOR EACH ROW

    DECLARE
        rows_found number;


    BEGIN
      SELECT COUNT(*)
      INTO   rows_found
      FROM   Representation
      WHERE TheaterId=:new.TheaterId AND RepresentationDate=:new.RepresentationDate;

      IF rows_found=1 THEN
        raise_application_error(-20001, ' A show is already programed');
      END IF;


    END;
```

So, this trigger count if there is any representation which has the same date that the one which we want to enter, if it’s not the case it’ll let it insert but if it’s we will have an error message.

So, let’s try an example to see if this trigger works.
We put the second representation at the same date and theater like the first one.



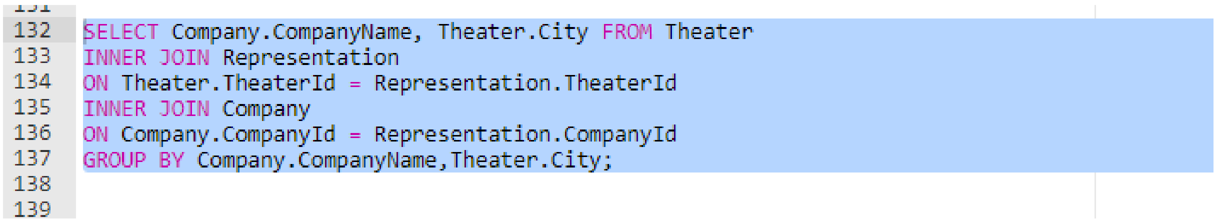
```
INSERT INTO Representation (RepresentationId, CompanyId, TheaterId, RepresentationDate,  ShowName,Price) VALUES ( 1, 1,1,'20/12/2020',  'Faboulous',50);
INSERT INTO Representation (RepresentationId, CompanyId, TheaterId, RepresentationDate,  ShowName,Price) VALUES ( 12, 2,1,'20/12/2020',  'Marvelous',50);
```

We tried to run it, but this message appeared.

```
ORA-20001:  A show is already programed ORA-06512: at "SQL_CHWXJSCKHKBENYCAEZVWKZGQY.OVERLAP", line 12
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

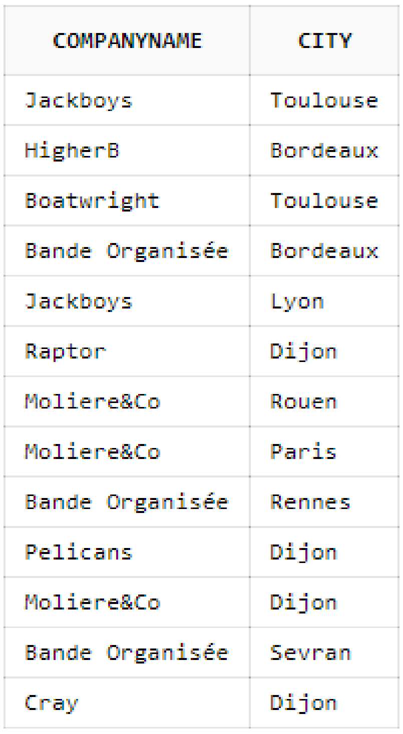
It shows that the trigger works, and we answered that question correctly.

What is the set of cities in which a company plays for a certain time period?



```sql
SELECT Company.CompanyName, Theater.City FROM Theater
INNER JOIN Representation
ON Theater.TheaterId = Representation.TheaterId
INNER JOIN Company
ON Company.CompanyId = Representation.CompanyId
GROUP BY Company.CompanyName,Theater.City;
```

So, we had to do 2 Inner Join to access all the columns we want, and we have as a result the cities where all companies play.

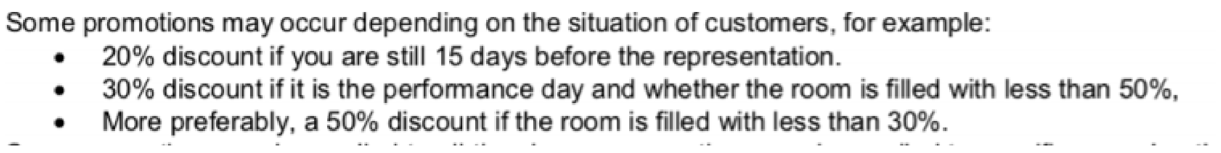


COMPANYNAME	CITY
Jackboys	Toulouse
HigherB	Bordeaux
Boatwright	Toulouse
Bande Organisée	Bordeaux
Jackboys	Lyon
Raptor	Dijon
Moliere&Co	Rouen
Moliere&Co	Paris
Bande Organisée	Rennes
Pelicans	Dijon
Moliere&Co	Dijon
Bande Organisée	Sevran
Cray	Dijon

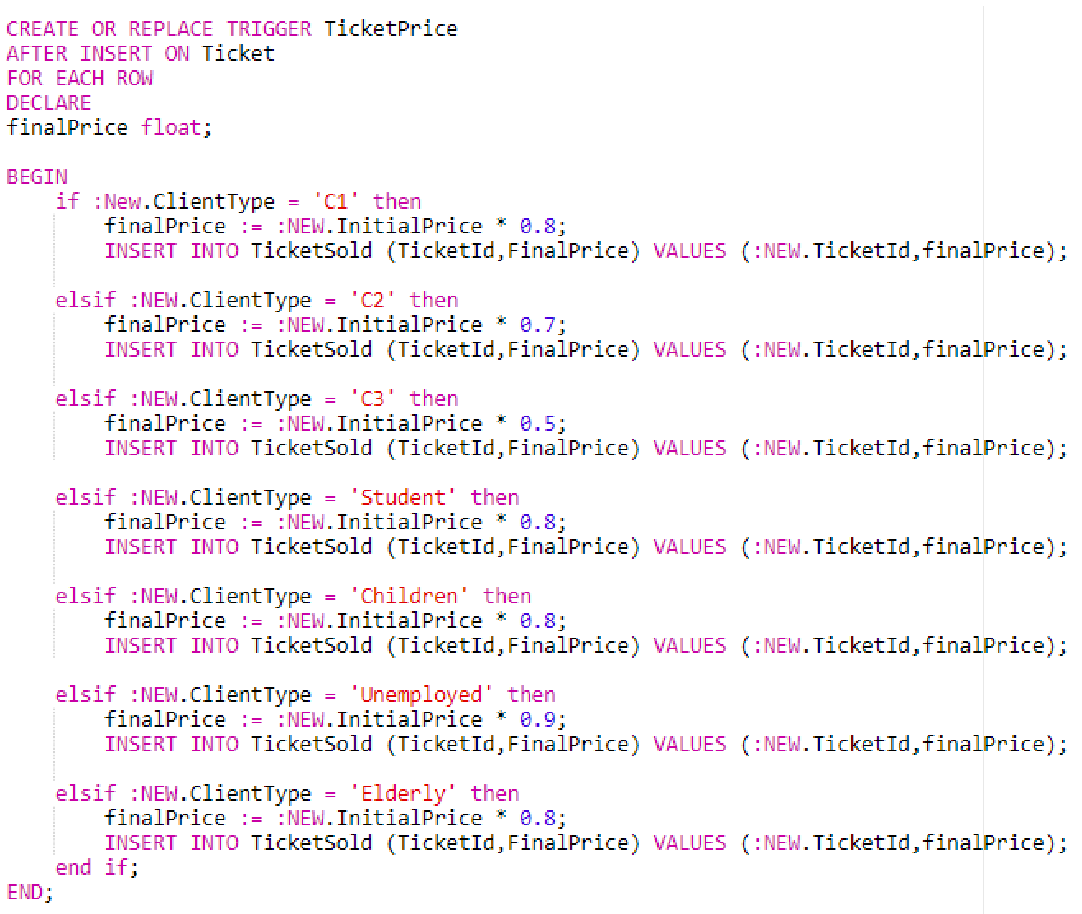
2.Ticketing

So first we will create a trigger to fill the TicketSold table, this table contains the TicketId and the FinalPrice with the promotion added.

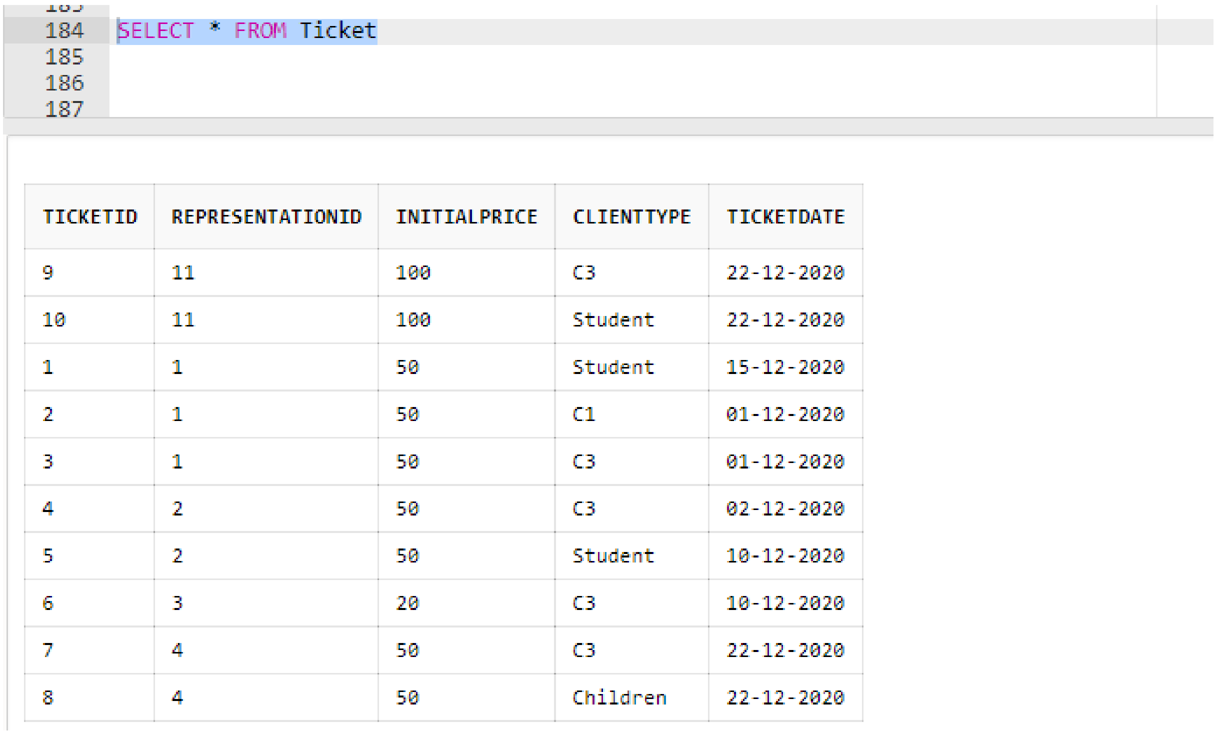
So, depending on the Status of the client TicketSold will contain the price the client paid. So, there are many Status. Children, Student etc but we also have C1, C2, C3 that are respectively those promotion.


Some promotions may occur depending on the situation of customers, for example:

- 20% discount if you are still 15 days before the representation.
- 30% discount if it is the performance day and whether the room is filled with less than 50%,
- More preferably, a 50% discount if the room is filled with less than 30%.

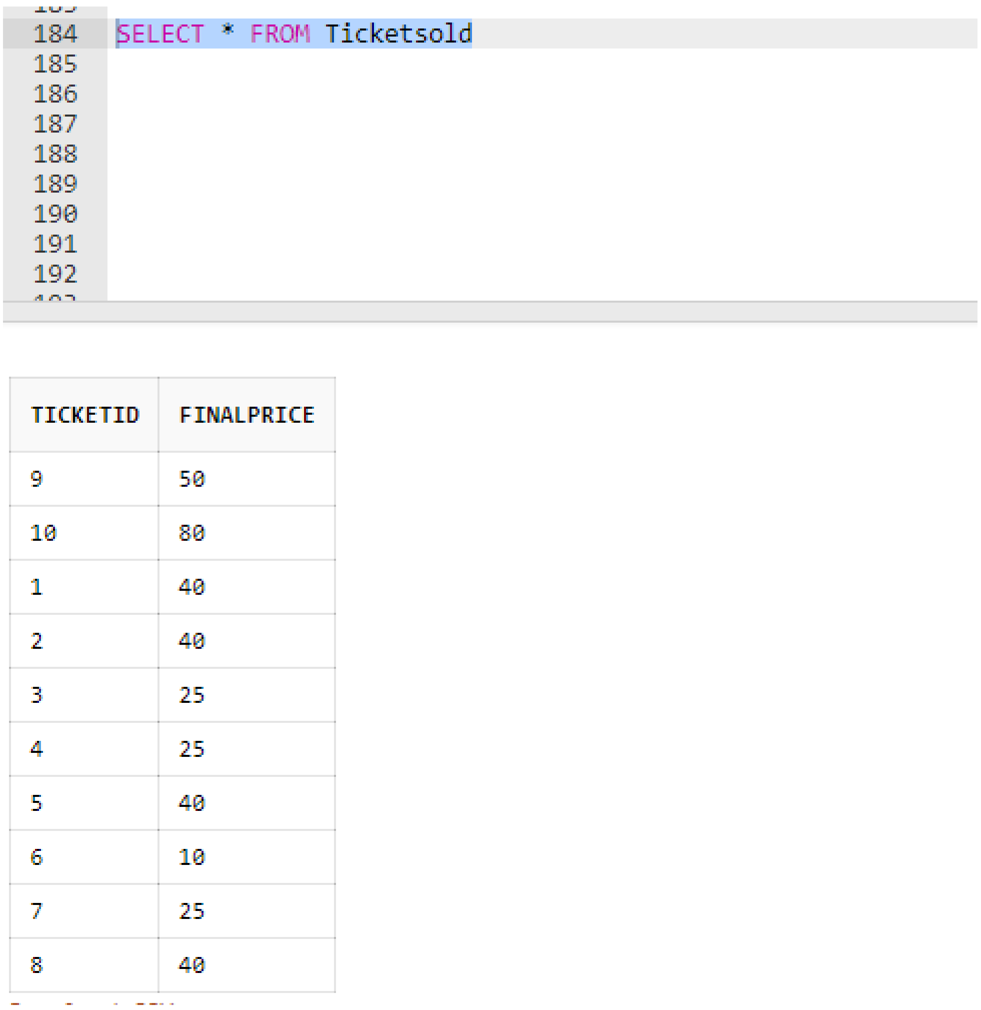

```
CREATE OR REPLACE TRIGGER TicketPrice
AFTER INSERT ON Ticket
FOR EACH ROW
DECLARE
finalPrice float;

BEGIN
    if :New.ClientType = 'C1' then
        finalPrice := :NEW.InitialPrice * 0.8;
        INSERT INTO TicketSold (TicketId,FinalPrice) VALUES (:NEW.TicketId,finalPrice);

    elsif :NEW.ClientType = 'C2' then
        finalPrice := :NEW.InitialPrice * 0.7;
        INSERT INTO TicketSold (TicketId,FinalPrice) VALUES (:NEW.TicketId,finalPrice);

    elsif :NEW.ClientType = 'C3' then
        finalPrice := :NEW.InitialPrice * 0.5;
        INSERT INTO TicketSold (TicketId,FinalPrice) VALUES (:NEW.TicketId,finalPrice);

    elsif :NEW.ClientType = 'Student' then
        finalPrice := :NEW.InitialPrice * 0.8;
        INSERT INTO TicketSold (TicketId,FinalPrice) VALUES (:NEW.TicketId,finalPrice);

    elsif :NEW.ClientType = 'Children' then
        finalPrice := :NEW.InitialPrice * 0.8;
        INSERT INTO TicketSold (TicketId,FinalPrice) VALUES (:NEW.TicketId,finalPrice);

    elsif :NEW.ClientType = 'Unemployed' then
        finalPrice := :NEW.InitialPrice * 0.9;
        INSERT INTO TicketSold (TicketId,FinalPrice) VALUES (:NEW.TicketId,finalPrice);

    elsif :NEW.ClientType = 'Elderly' then
        finalPrice := :NEW.InitialPrice * 0.8;
        INSERT INTO TicketSold (TicketId,FinalPrice) VALUES (:NEW.TicketId,finalPrice);
    end if;
END;
```

You can see that Ticket have all information concerning the initial price the client and the ticket buying date but Ticket Sold has the exact amount of what the client paid to the cash desk.



```sql
SELECT * FROM Ticket
```

TICKETID	REPRESENTATIONID	INITIALPRICE	CLIENTTYPE	TICKETDATE
9	11	100	C3	22-12-2020
10	11	100	Student	22-12-2020
1	1	50	Student	15-12-2020
2	1	50	C1	01-12-2020
3	1	50	C3	01-12-2020
4	2	50	C3	02-12-2020
5	2	50	Student	10-12-2020
6	3	20	C3	10-12-2020
7	4	50	C3	22-12-2020
8	4	50	Children	22-12-2020



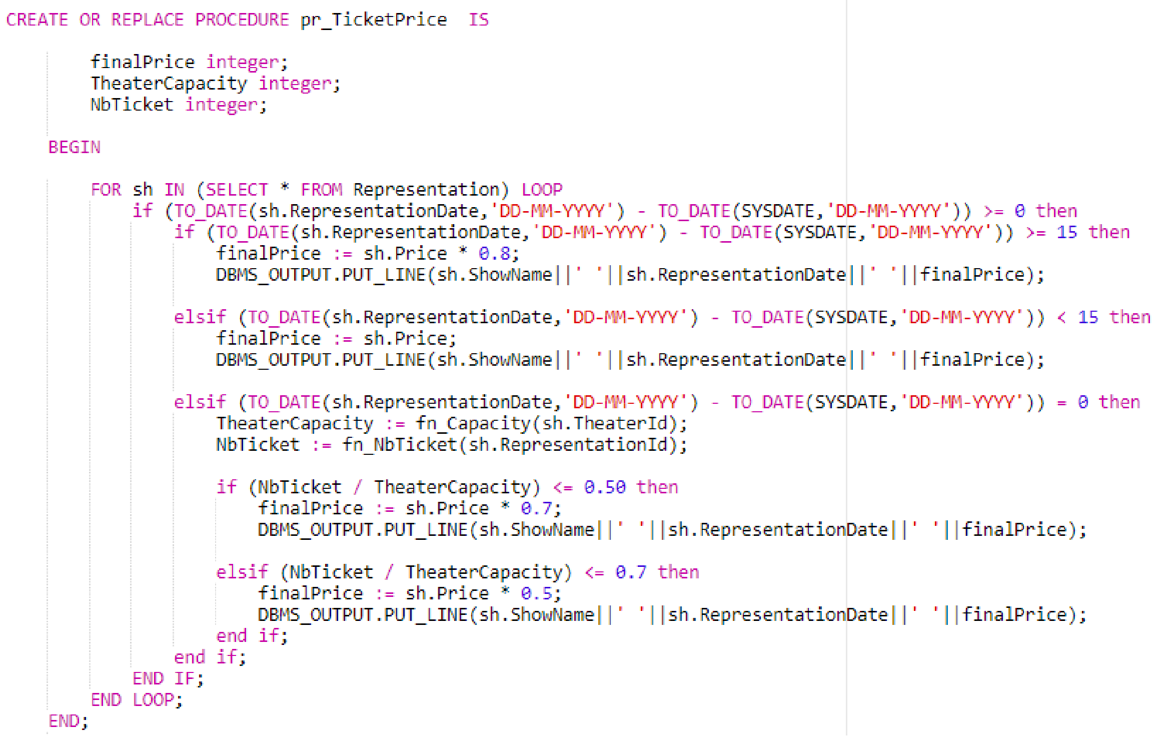
```sql
SELECT * FROM Ticketsold
```

TICKETID	FINALPRICE
9	50
10	80
1	40
2	40
3	25
4	25
5	40
6	10
7	25
8	40

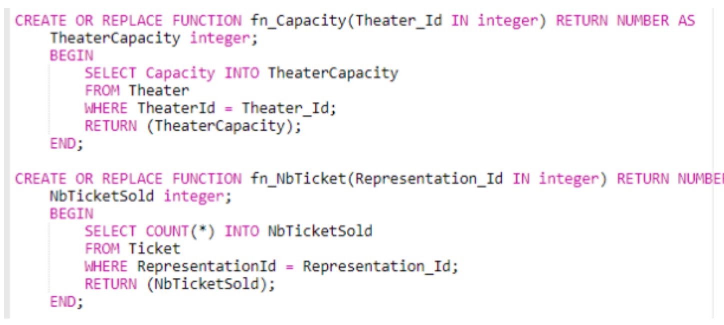
We can now answer to three questions concerning Ticketing.

What is the ticket price today?

For this example, we took the date 27/12/2020 as today date.
This trigger check the date condition,
in fact if it's 15 days before the representation the clinet can benefit 20% of reduction on the initial price and it's also the case if it's the day of representation and the room is not filled enough (less than 50% and less than 30%)
The help of function was needed to correctly make the trigger work. Those function returned the Theater capacity and the number of Ticket Sold to check promotion conditions on the trigger.



```
CREATE OR REPLACE PROCEDURE pr_TicketPrice  IS

        finalPrice integer;
        TheaterCapacity integer;
        NbTicket integer;

    BEGIN

        FOR sh IN (SELECT * FROM Representation) LOOP
            if (TO_DATE(sh.RepresentationDate,'DD-MM-YYYY') - TO_DATE(SYSDATE,'DD-MM-YYYY')) >= 0 then
                if (TO_DATE(sh.RepresentationDate,'DD-MM-YYYY') - TO_DATE(SYSDATE,'DD-MM-YYYY')) >= 15 then
                    finalPrice := sh.Price * 0.8;
                    DBMS_OUTPUT.PUT_LINE(sh.ShowName||' '||sh.RepresentationDate||' '||finalPrice);

                elsif (TO_DATE(sh.RepresentationDate,'DD-MM-YYYY') - TO_DATE(SYSDATE,'DD-MM-YYYY')) < 15 then
                    finalPrice := sh.Price;
                    DBMS_OUTPUT.PUT_LINE(sh.ShowName||' '||sh.RepresentationDate||' '||finalPrice);

                elsif (TO_DATE(sh.RepresentationDate,'DD-MM-YYYY') - TO_DATE(SYSDATE,'DD-MM-YYYY')) = 0 then
                    TheaterCapacity := fn_Capacity(sh.TheaterId);
                    NbTicket := fn_NbTicket(sh.RepresentationId);

                    if (NbTicket / TheaterCapacity) <= 0.50 then
                        finalPrice := sh.Price * 0.7;
                        DBMS_OUTPUT.PUT_LINE(sh.ShowName||' '||sh.RepresentationDate||' '||finalPrice);

                    elsif (NbTicket / TheaterCapacity) <= 0.7 then
                        finalPrice := sh.Price * 0.5;
                        DBMS_OUTPUT.PUT_LINE(sh.ShowName||' '||sh.RepresentationDate||' '||finalPrice);
                    end if;
                end if;
            END IF;
        END LOOP;
    END;
```



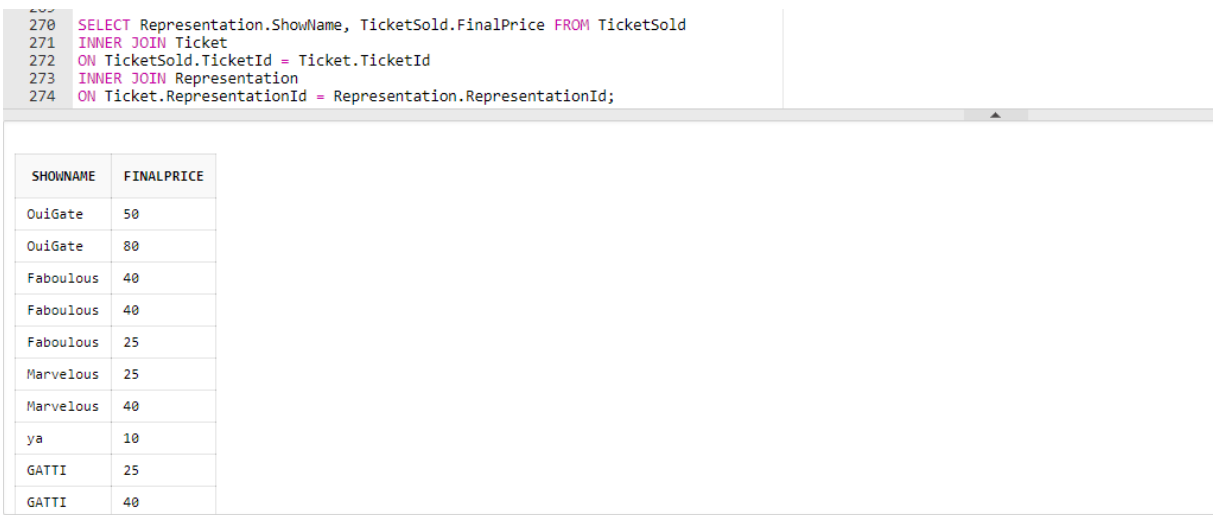
```
CREATE OR REPLACE FUNCTION fn_Capacity(Theater_Id IN integer) RETURN NUMBER AS
    TheaterCapacity integer;
    BEGIN
        SELECT Capacity INTO TheaterCapacity
        FROM Theater
        WHERE TheaterId = Theater_Id;
        RETURN (TheaterCapacity);
    END;

CREATE OR REPLACE FUNCTION fn_NbTicket(Representation_Id IN integer) RETURN NUMBER AS
    NbTicketSold integer;
    BEGIN
        SELECT COUNT(*) INTO NbTicketSold
        FROM Ticket
        WHERE RepresentationId = Representation_Id;
        RETURN (NbTicketSold);
    END;
```

```
Statement processed.
GATTI 11-01-2021 40
Don 12-02-2021 40
Dream 01-01-2021 10
Destiny 20-04-2021 32
Love 03-03-2021 40
```

So, we can see that if a client wants to know the initial prices for shows in the theater, it'll shows us those shows with prices different from the initial price and depending on the date and the filling of shows.

What is the distribution of tickets by price?

The second question is concerning the distribution of ticket price, in fact with discounts applied on tickets the same tickets shows can be sold with different prices.



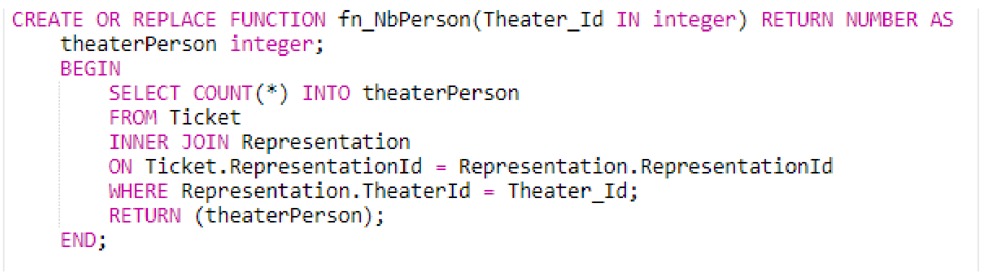
```sql
SELECT Representation.ShowName, TicketSold.FinalPrice FROM TicketSold
INNER JOIN Ticket
ON TicketSold.TicketId = Ticket.TicketId
INNER JOIN Representation
ON Ticket.RepresentationId = Representation.RepresentationId;
```

SHOWNAME	FINALPRICE
OuiGate	50
OuiGate	80
Faboulous	40
Faboulous	40
Faboulous	25
Marvelous	25
Marvelous	40
ya	10
GATTI	25
GATTI	40

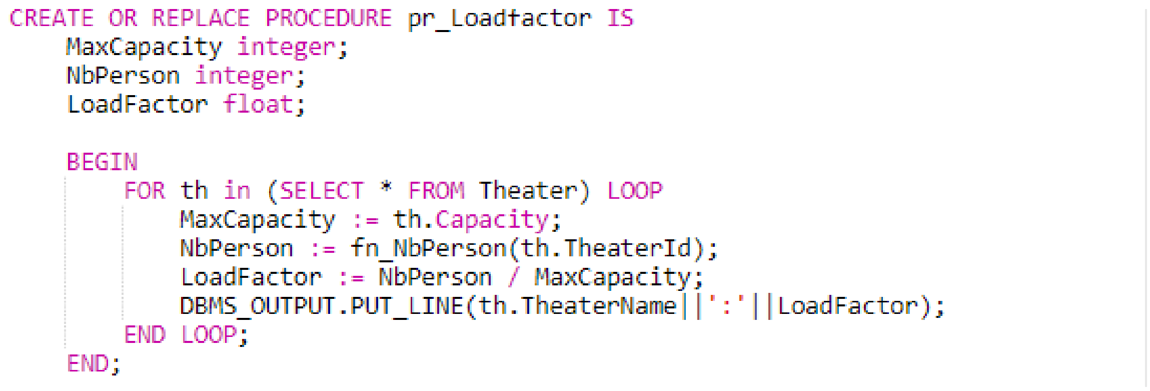
What is the average load factor?

So first what is the definition of average load?
We took the number of people who bought tickets for a specific show divide by the capacity of the theater in which the show will be played.
The help of functions was needed.

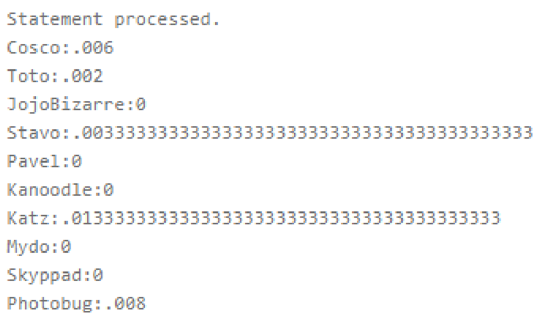
We created a procedure to shows us the average load factor of each theater.



```
CREATE OR REPLACE FUNCTION fn_NbPerson(Theater_Id IN integer) RETURN NUMBER AS
    theaterPerson integer;
    BEGIN
        SELECT COUNT(*) INTO theaterPerson
        FROM Ticket
        INNER JOIN Representation
        ON Ticket.RepresentationId = Representation.RepresentationId
        WHERE Representation.TheaterId = Theater_Id;
        RETURN (theaterPerson);
    END;
```



```
CREATE OR REPLACE PROCEDURE pr_Loadfactor IS
    MaxCapacity integer;
    NbPerson integer;
    LoadFactor float;

    BEGIN
        FOR th in (SELECT * FROM Theater) LOOP
            MaxCapacity := th.Capacity;
            NbPerson := fn_NbPerson(th.TheaterId);
            LoadFactor := NbPerson / MaxCapacity;
            DBMS_OUTPUT.PUT_LINE(th.TheaterName||':'||LoadFactor);
        END LOOP;
    END;
```



```
Statement processed.
Cosco:.006
Toto:.002
JojoBizarre:0
Stavo:.003333333333333333333333333333333333333333
Pavel:0
Kanoodle:0
Katz:.0133333333333333333333333333333333333333
Mydo:0
Skyppad:0
Photobug:.008
```

Those numbers are very low considering our database with few tickets sold but a great capacity for theaters.

3.Accounting

For this part we created a table name **TheaterBalance**, a table that will allow each theater to know their current balance. To make this table work perfectly, we created a trigger that will update the current balance of a theater for each ticket sold.

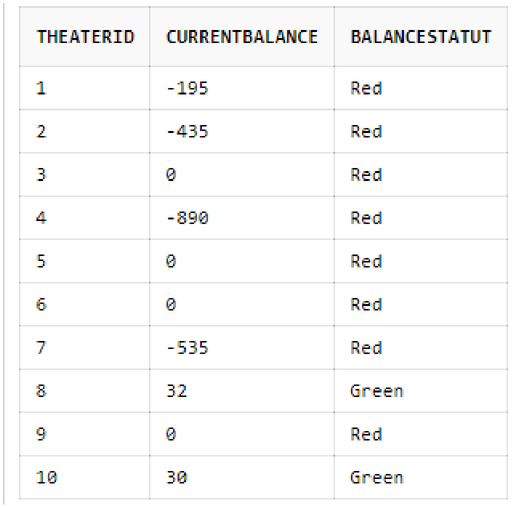
```
CREATE OR REPLACE TRIGGER CheckBalance
AFTER INSERT ON Ticket
FOR EACH ROW
    DECLARE
        theater_id integer;
        company_id integer;
        ticket_price float;
        actual_balance float;
        final_balance float;


  BEGIN
      theater_id := fn_TheaterId(:NEW.RepresentationId);
      company_id := fn_CompanyId(:NEW.RepresentationId);

      IF :New.ClientType = 'C1' THEN
          ticket_price := :NEW.InitialPrice * 0.8;

      ELSIF :NEW.ClientType = 'C2' THEN
          ticket_price := :NEW.InitialPrice * 0.7;

      ELSIF :NEW.ClientType = 'C3' THEN
          ticket_price := :NEW.InitialPrice * 0.5;

      ELSIF :NEW.ClientType = 'Student' THEN
          ticket_price := :NEW.InitialPrice * 0.8;

      ELSIF :NEW.ClientType = 'Children' THEN
          ticket_price := :NEW.InitialPrice * 0.8;

      ELSIF :NEW.ClientType = 'Unemployed' THEN
          ticket_price := :NEW.InitialPrice * 0.9;

      ELSIF :NEW.ClientType = 'Elderly' THEN
          ticket_price := :NEW.InitialPrice * 0.8;

      END IF;

      IF (TO_DATE(:NEW.TicketDate,'DD-MM-YYYY') - TO_DATE(SYSDATE,'DD-MM-YYYY')) = 0 THEN

          SELECT CurrentBalance INTO actual_balance FROM TheaterBalance WHERE TheaterBalance.TheaterId = theater_id;

          final_balance := actual_balance + ticket_price ;

          IF final_balance <= 0 THEN
              UPDATE TheaterBalance SET CurrentBalance = final_balance, BalanceStatut = 'Red' WHERE TheaterId = theater_id;
          ELSE
              UPDATE TheaterBalance SET CurrentBalance = final_balance, BalanceStatut = 'Green' WHERE TheaterId = theater_id;
          END IF;

      END IF;
  END;
```

Basically, this trigger will be activated if we insert a new ticket in the table **Ticket**. So, if a new ticket is sold today (it means that we insert a new ticket) then this trigger will update the current balance of the theater concerned. And of course, this trigger checks the client type to apply an eventual discount.

Here is an example:

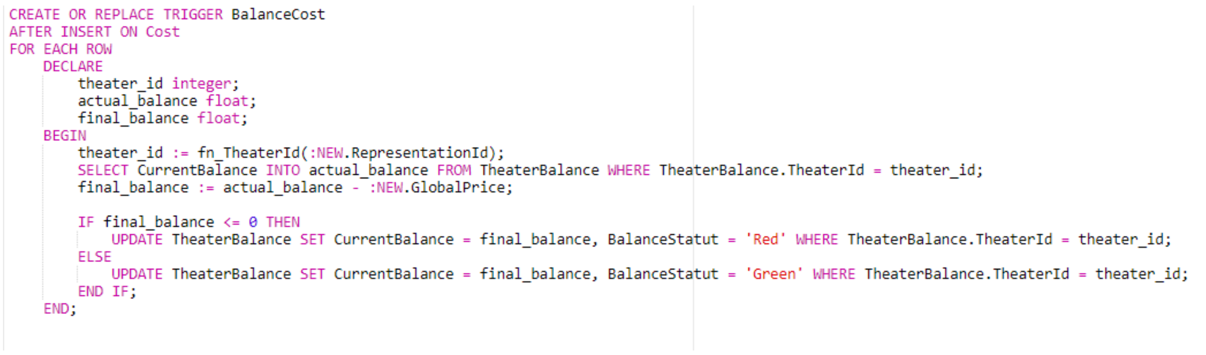
```
INSERT INTO Ticket(TicketId,RepresentationId,InitialPrice,ClientType,TicketDate) VALUES (11,7,40,'Student','27/12/2020');
```

We inserted a new ticket today, for the representation 7. This representation takes place in the 8th theater, so it means that in the table **TheaterBalance** the trigger will update the 8th theater's current balance which was in the beginning at 0.



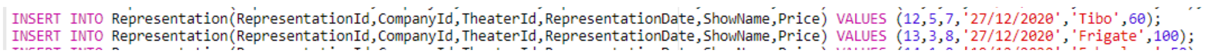
THEATERID	CURRENTBALANCE	BALANCESTATUT
1	-195	Red
2	-435	Red
3	0	Red
4	-890	Red
5	0	Red
6	0	Red
7	-535	Red
8	32	Green
9	0	Red
10	30	Green

Here we can see that the trigger updated the 8th theater's current balance. It was at zero but now it is at 32. It is the price of the ticket sold. But we have a problem here. In fact, when a show is programmed for a theater, it induces that the theater has to pay a global price as indicated in the subject. To do this we created another trigger that will update the table **TheaterBalance** each time we insert a new cost. Actually, a new cost will be inserted when a new representation is programmed.



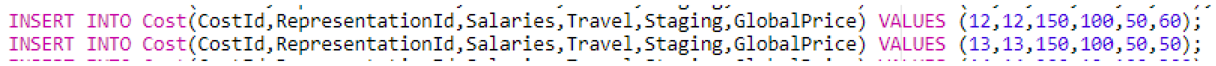
```
CREATE OR REPLACE TRIGGER BalanceCost
AFTER INSERT ON Cost
FOR EACH ROW
    DECLARE
        theater_id integer;
        actual_balance float;
        final_balance float;
    BEGIN
        theater_id := fn_TheaterId(:NEW.RepresentationId);
        SELECT CurrentBalance INTO actual_balance FROM TheaterBalance WHERE TheaterBalance.TheaterId = theater_id;
        final_balance := actual_balance - :NEW.GlobalPrice;

        IF final_balance <= 0 THEN
            UPDATE TheaterBalance SET CurrentBalance = final_balance, BalanceStatut = 'Red' WHERE TheaterBalance.TheaterId = theater_id;
        ELSE
            UPDATE TheaterBalance SET CurrentBalance = final_balance, BalanceStatut = 'Green' WHERE TheaterBalance.TheaterId = theater_id;
        END IF;
    END;
```

Here is an example:



```
INSERT INTO Representation(RepresentationId,CompanyId,TheaterId,RepresentationDate,ShowName,Price) VALUES (12,5,7,'27/12/2020','Tibo',60);
INSERT INTO Representation(RepresentationId,CompanyId,TheaterId,RepresentationDate,ShowName,Price) VALUES (13,3,8,'27/12/2020','Frigate',100);
```

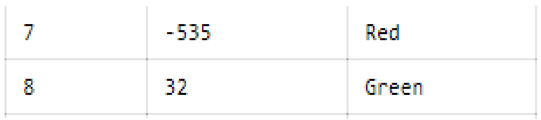
Here we inserted two new representation that are programmed for today. We can see that one take place in the 7th theater and the other one in the 8th.



```
INSERT INTO Cost(CostId,RepresentationId,Salaries,Travel,Staging,GlobalPrice) VALUES (12,12,150,100,50,60);
INSERT INTO Cost(CostId,RepresentationId,Salaries,Travel,Staging,GlobalPrice) VALUES (13,13,150,100,50,50);
```

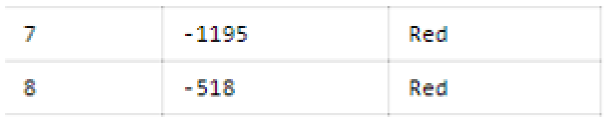
As we inserted two new representations, we will also insert their costs.

Normally, the trigger has updated the **TheaterBalance** table.



7	-535	Red
8	32	Green

Before

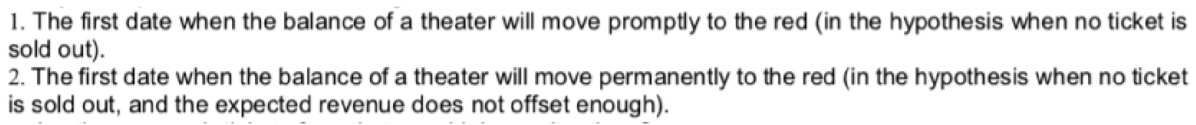


7	-1195	Red
8	-518	Red

After

We can see that the current balance of the concerned theater has been updated. Now we have a fully functional **TheaterBalance** table.

To be honest we did not really understand the following questions.

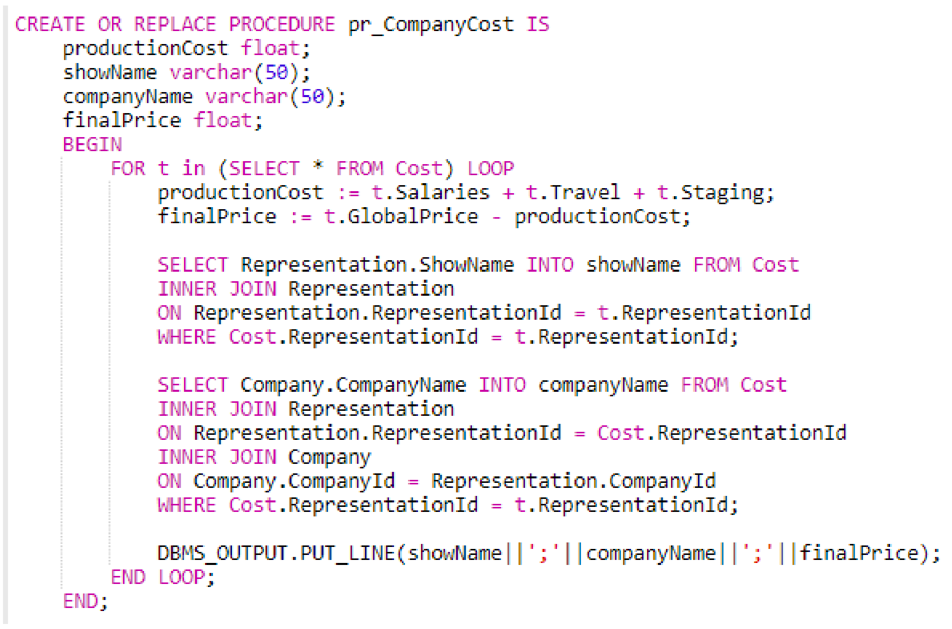


1. The first date when the balance of a theater will move promptly to the red (in the hypothesis when no ticket is sold out).
2. The first date when the balance of a theater will move permanently to the red (in the hypothesis when no ticket is sold out, and the expected revenue does not offset enough).

So, what we did is that if the current balance of a theater is less or equal to 0 then the balance will be red. Else the balance will be green. You can refer to our trigger's screenshots above to see when we update the balance status.

A show given by a company was sold out with a certain price. Is it cost effective for the company? (Compared to costs incurred (salaries, travel, staging))

To answer this question, we created a procedure that displays each companies' final cost for each show.



```
CREATE OR REPLACE PROCEDURE pr_CompanyCost IS
    productionCost float;
    showName varchar(50);
    companyName varchar(50);
    finalPrice float;
    BEGIN
        FOR t in (SELECT * FROM Cost) LOOP
            productionCost := t.Salaries + t.Travel + t.Staging;
            finalPrice := t.GlobalPrice - productionCost;

            SELECT Representation.ShowName INTO showName FROM Cost
            INNER JOIN Representation
            ON Representation.RepresentationId = t.RepresentationId
            WHERE Cost.RepresentationId = t.RepresentationId;

            SELECT Company.CompanyName INTO companyName FROM Cost
            INNER JOIN Representation
            ON Representation.RepresentationId = Cost.RepresentationId
            INNER JOIN Company
            ON Company.CompanyId = Representation.CompanyId
            WHERE Cost.RepresentationId = t.RepresentationId;

            DBMS_OUTPUT.PUT_LINE(showName||';'||companyName||';'||finalPrice);
        END LOOP;
    END;
```

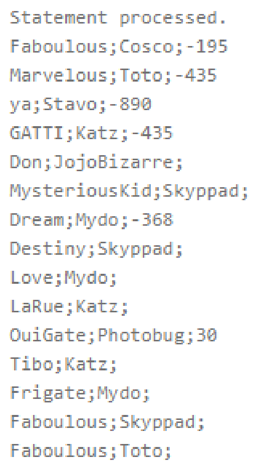
```
Statement processed.
Faboulous;Moliere&Co;0
Marvelous;Moliere&Co;0
ya;BandeOrganisée;0
GATTI;HigherB;0
Don;Jackboys;0
MysteriousKid;Raptor;0
Dream;Boatwright;0
Destiny;Cray;-500
Love;Boatwright;-30
LaRue;BandeOrganisée;-200
OuiGate;BandeOrganisée;-200
Tibo;Pelicans;-240
Frigate;Jackboys;-250
Faboulous;Moliere&Co;-10
Faboulous;Moliere&Co;-25
```

Here we can see the final cost of each companies for each show. We can clearly see if it was cost effective for the company or not.

Was it the effective cost for the theater? (Costs / ticketing)

For this question, we did the same as before. We created a procedure that displays the final cost of each theater for each show.

```
CREATE OR REPLACE PROCEDURE pr_TheaterProfitability IS
    finalPrice float;
    sumTicket float;
    globalPrice float;
    showName varchar(50);
    theaterName varchar(50);
    BEGIN
        FOR t in (SELECT * FROM Cost) LOOP

            SELECT Cost.GlobalPrice INTO globalPrice FROM Representation
            INNER JOIN Cost
            ON Cost.RepresentationId = Representation.RepresentationId
            WHERE Cost.RepresentationId = t.RepresentationId;

            SELECT SUM(TicketSold.FinalPrice) INTO sumTicket FROM Ticket
            INNER JOIN TicketSold
            ON TicketSold.TicketId = Ticket.TicketId
            INNER JOIN Representation
            ON Representation.RepresentationId = Ticket.RepresentationId
            WHERE Ticket.RepresentationId = t.RepresentationId;

            SELECT Theater.TheaterName INTO theaterName FROM Representation
            INNER JOIN Theater
            ON Theater.TheaterId = Representation.TheaterId
            WHERE Representation.RepresentationId = t.RepresentationId;

            SELECT Representation.ShowName INTO showName FROM Representation
            WHERE Representation.RepresentationId = t.RepresentationId;

            finalPrice := sumTicket - globalPrice;
            DBMS_OUTPUT.PUT_LINE(showName||';'||theaterName||';'||finalPrice);
        END LOOP;
    END;
```



```
Statement processed.
Faboulous;Cosco;-195
Marvelous;Toto;-435
ya;Stavo;-890
GATTI;Katz;-435
Don;JojoBizarre;
MysteriousKid;Skyppad;
Dream;Mydo;-368
Destiny;Skyppad;
Love;Mydo;
LaRue;Katz;
OuiGate;Photobug;30
Tibo;Katz;
Frigate;Mydo;
Faboulous;Skyppad;
Faboulous;Toto;
```

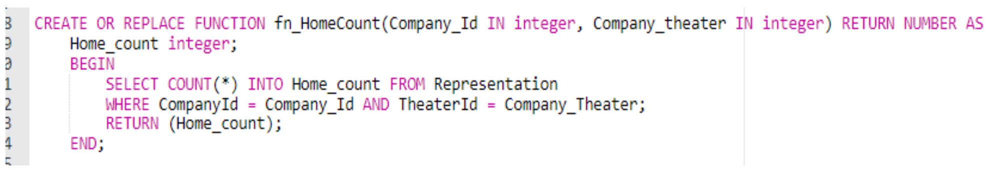
We can see that some theaters don’t have any cost, it is because no tickets were sold yet for this show.

4.Network

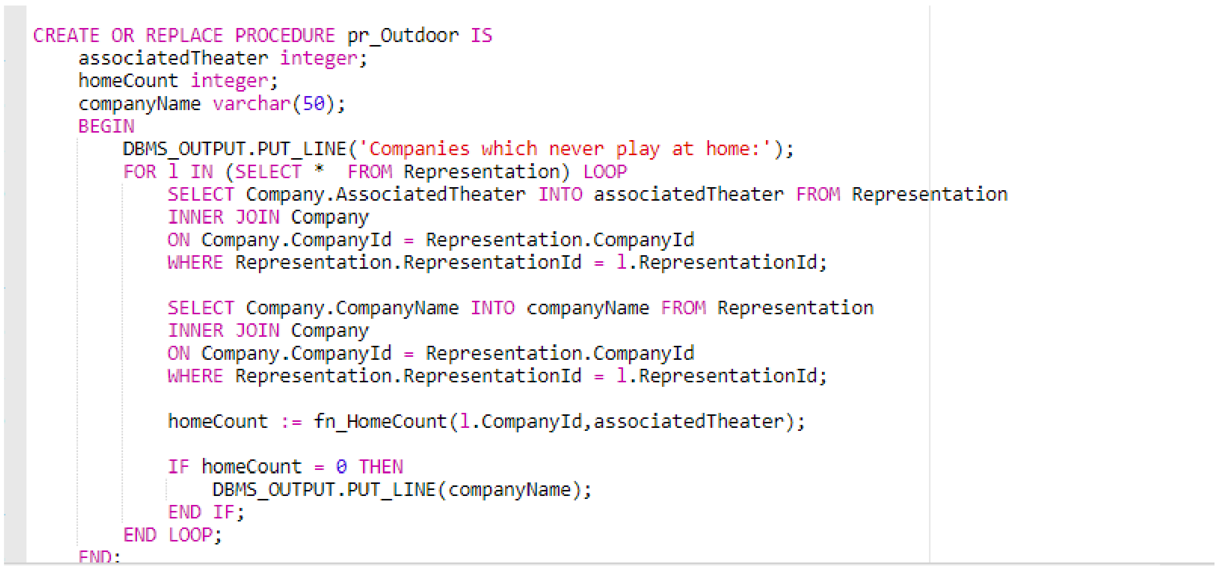
The last section is concerning the network itself.

Are there companies that will never play in their theater?

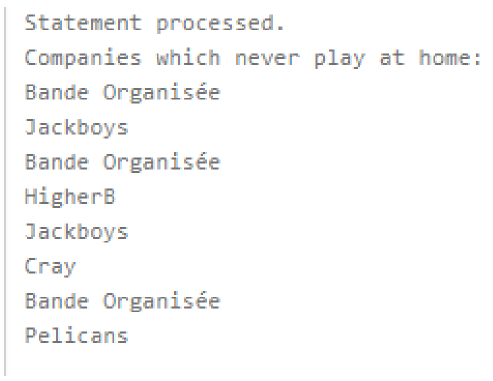
So, for answering this question we checked if some companies ever played on their theater with the help of this function that return the number of companies that play in their theater. Then with the procedure depending on companies we display companies that never played in their theater



```
CREATE OR REPLACE FUNCTION fn_HomeCount(Company_Id IN integer, Company_theater IN integer) RETURN NUMBER AS
    Home_count integer;
    BEGIN
        SELECT COUNT(*) INTO Home_count FROM Representation
        WHERE CompanyId = Company_Id AND TheaterId = Company_Theater;
        RETURN (Home_count);
    END;
```



```
CREATE OR REPLACE PROCEDURE pr_Outdoor IS
    associatedTheater integer;
    homeCount integer;
    companyName varchar(50);
    BEGIN
        DBMS_OUTPUT.PUT_LINE('Companies which never play at home:');
        FOR l IN (SELECT *  FROM Representation) LOOP
            SELECT Company.AssociatedTheater INTO associatedTheater FROM Representation
            INNER JOIN Company
            ON Company.CompanyId = Representation.CompanyId
            WHERE Representation.RepresentationId = l.RepresentationId;

            SELECT Company.CompanyName INTO companyName FROM Representation
            INNER JOIN Company
            ON Company.CompanyId = Representation.CompanyId
            WHERE Representation.RepresentationId = l.RepresentationId;

            homeCount := fn_HomeCount(l.CompanyId,associatedTheater);

            IF homeCount = 0 THEN
                DBMS_OUTPUT.PUT_LINE(companyName);
            END IF;
        END LOOP;
    END;
```

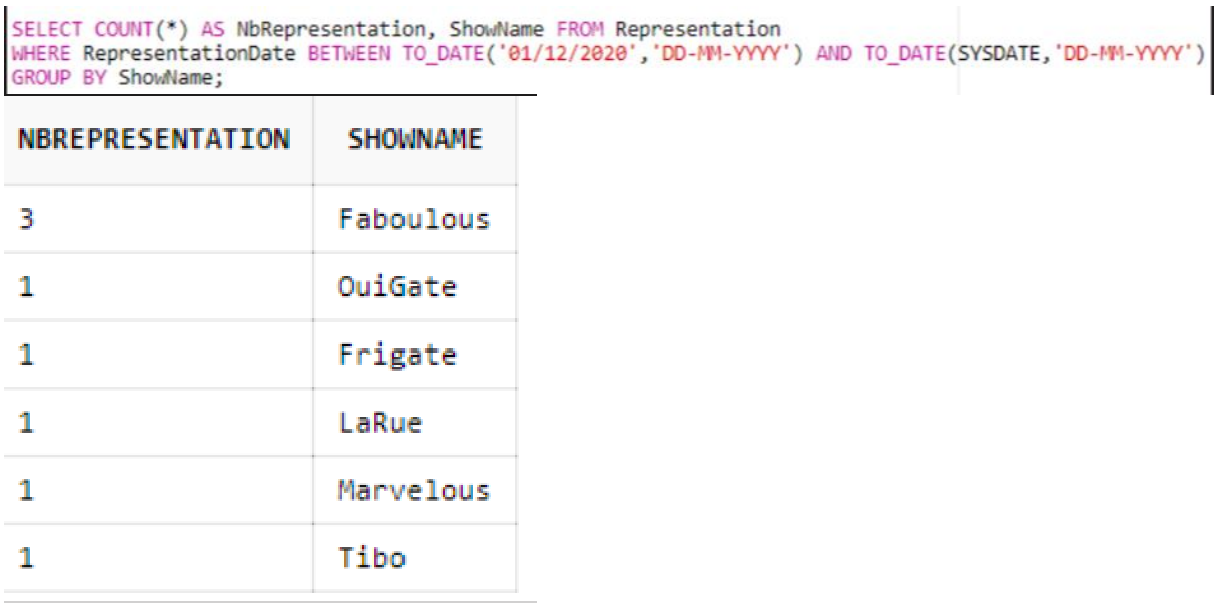


```
Statement processed.
Companies which never play at home:
Bande Organisée
Jackboys
Bande Organisée
HigherB
Jackboys
Cray
Bande Organisée
Pelicans
```

The small problem with this statement is that it can display the same company if it plays many times outside and never in their theater.

Now let's see the most popular shows in a certain period

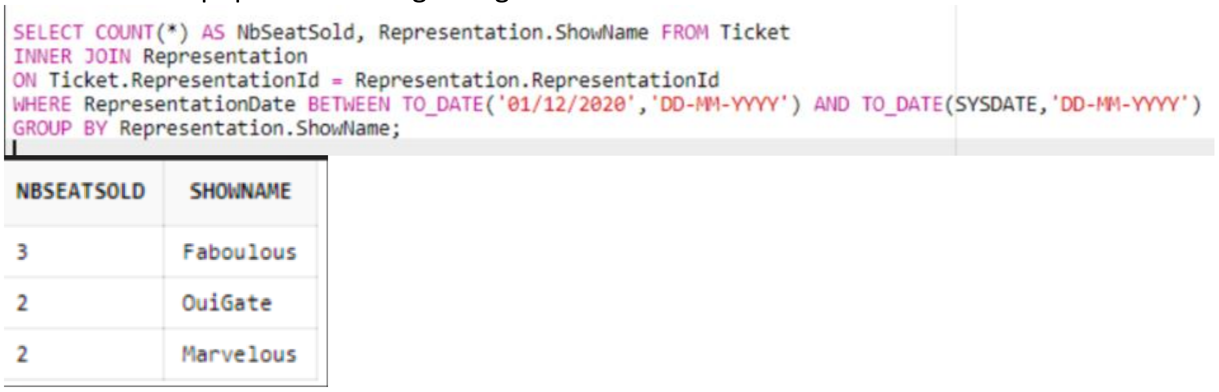
First, the most popular show in number of representations played in the month of December.



```
SELECT COUNT(*) AS NbRepresentation, ShowName FROM Representation
WHERE RepresentationDate BETWEEN TO_DATE('01/12/2020','DD-MM-YYYY') AND TO_DATE(SYSDATE,'DD-MM-YYYY')
GROUP BY ShowName;
```

NBREPRESENTATION	SHOWNAME
3	Faboulous
1	OuiGate
1	Frigate
1	LaRue
1	Marvelous
1	Tibo

Then the most popular show regarding the number of seats sold.



```
SELECT COUNT(*) AS NbSeatSold, Representation.ShowName FROM Ticket
INNER JOIN Representation
ON Ticket.RepresentationId = Representation.RepresentationId
WHERE RepresentationDate BETWEEN TO_DATE('01/12/2020','DD-MM-YYYY') AND TO_DATE(SYSDATE,'DD-MM-YYYY')
GROUP BY Representation.ShowName;
```

NBSEATSOLD	SHOWNAME
3	Faboulous
2	OuiGate
2	Marvelous

IV. Encountered Problems

The first problem we encountered was about the software itself. In fact, we were used to use **sql server management studio**. So, we naturally began our project with this software. But we quickly realized that the syntax for triggers was not the same as the one we saw in class with oracle. So, we wasted a big amount of time searching how to continue the project. We tried on the oracle VM, but we could not import our initial script from our computer. Furthermore, oracle VM was not the ideal software to use for a project.

We finally use Oracle Live Sql to do our project even if it was not the ideal software for a project too. Our session expired multiple times while we were coding and we had to re-run all the statements we did previously. Moreover, we also had many errors while coding on live sql, errors that we didn't understand the first time. The syntax in Oracle change from what we used to do in sql server, so errors due to syntax and others occurred frequently. And here again we spent a lot of time to understand and correct the errors even if it was a small one. Finally, we managed to finish the project but honestly, we think that the next time, it will be good to have a proper software in which we practiced. So that we can do our project according to what we saw in class without wasting time.

V. Conclusion

In conclusion, we must admit that we are proud of our project and our investment of time to finish correctly all queries.

This project helped us to understand how to use triggers, functions and procedures but also to improve our skills in SQL. Despite having some difficulties with the program used, SQL Oracle was a new program for us but thanks to our TP on triggers and function we understand easily how to use Oracle SQL Developer. Again, we would like to thank Mr. Issam Falih for his help concerning the understanding of the project. For further details about the code please check our commented script on triggers/functions/procedures.