IUT - Maubeuge
BUT – Informatique

Année universitaire 2024-2025
Groupe BUT 1 TPB1
Le 20 Janvier 2024

Compte Rendu - SAÉ 104
Création d'une base de données

CHOURAIH Baptiste
COLLOT Grégoire
MARECAILLE-HENAUT Mattieu
SIMSEK Dilara

Sommaire

Introduction

Nous sommes étudiants en première année de BUT Informatique à l'IUT de Maubeuge.

Dans le cadre de la ressource SAÉ 104, nous avons réalisé la mise en place d'une BDD (base de données) en suivant les restrictions imposées et le diagramme de classe fournit, les données utilisées durant cette SAÉ proviennent d'un projet antérieur sur la réalisation d'un recueil de besoin afin de réaliser un festival.

Lors de notre première session, nous avons mis en place les tâches nécessaires pour réaliser ce projet suivi de l'attribution de chaque tâche en fonction des qualités et envies des membres du groupe.

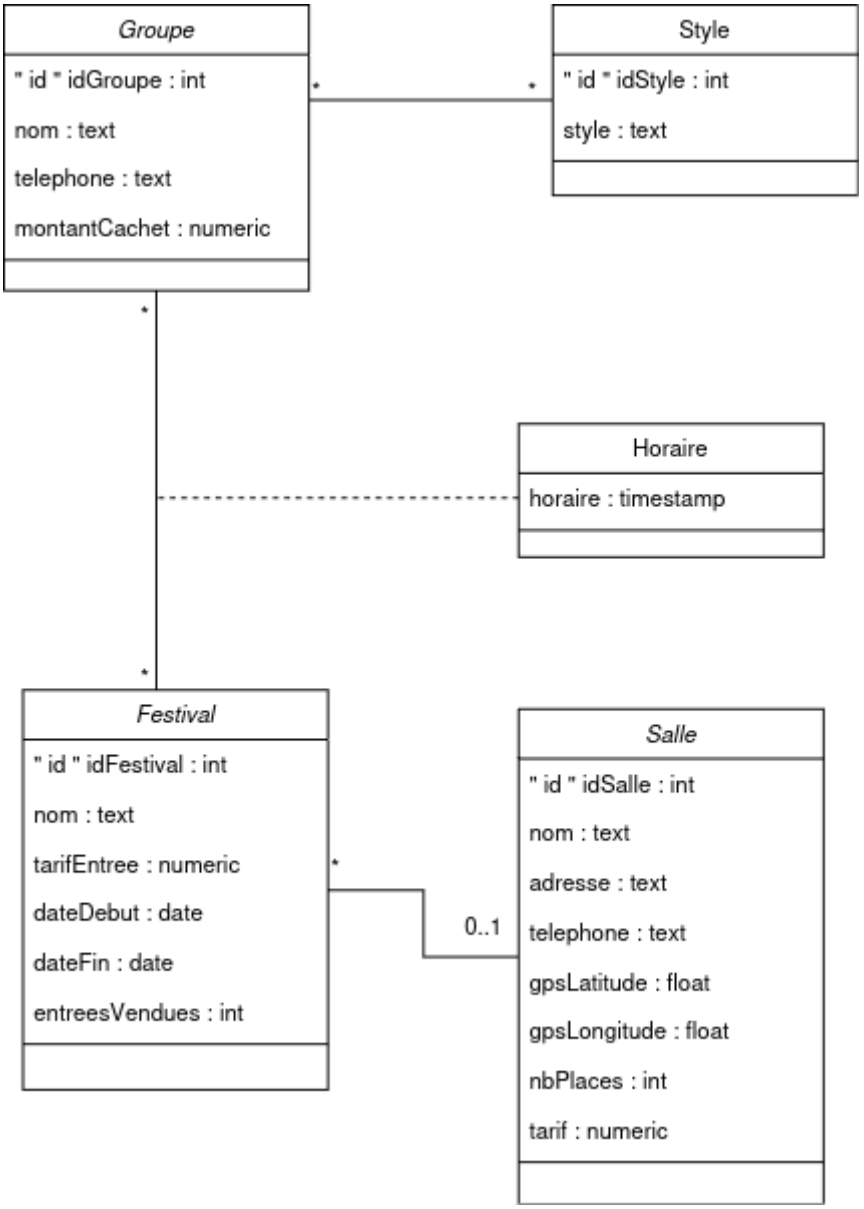
Ainsi, nous avons d'abord mis en place le script pour créer les tables de la BDD, suivis de l'insertion des données et fini par une série de tests.

Enfin nous avons finis par étendre notre première base de données afin d'y incorporer plus d'informations sur les festivals, il nous a fallu par la suite mettre à jour notre série de test.

Nous vous remercions d'avance, pour le temps que vous nous accordez en lisant ce compte rendu.

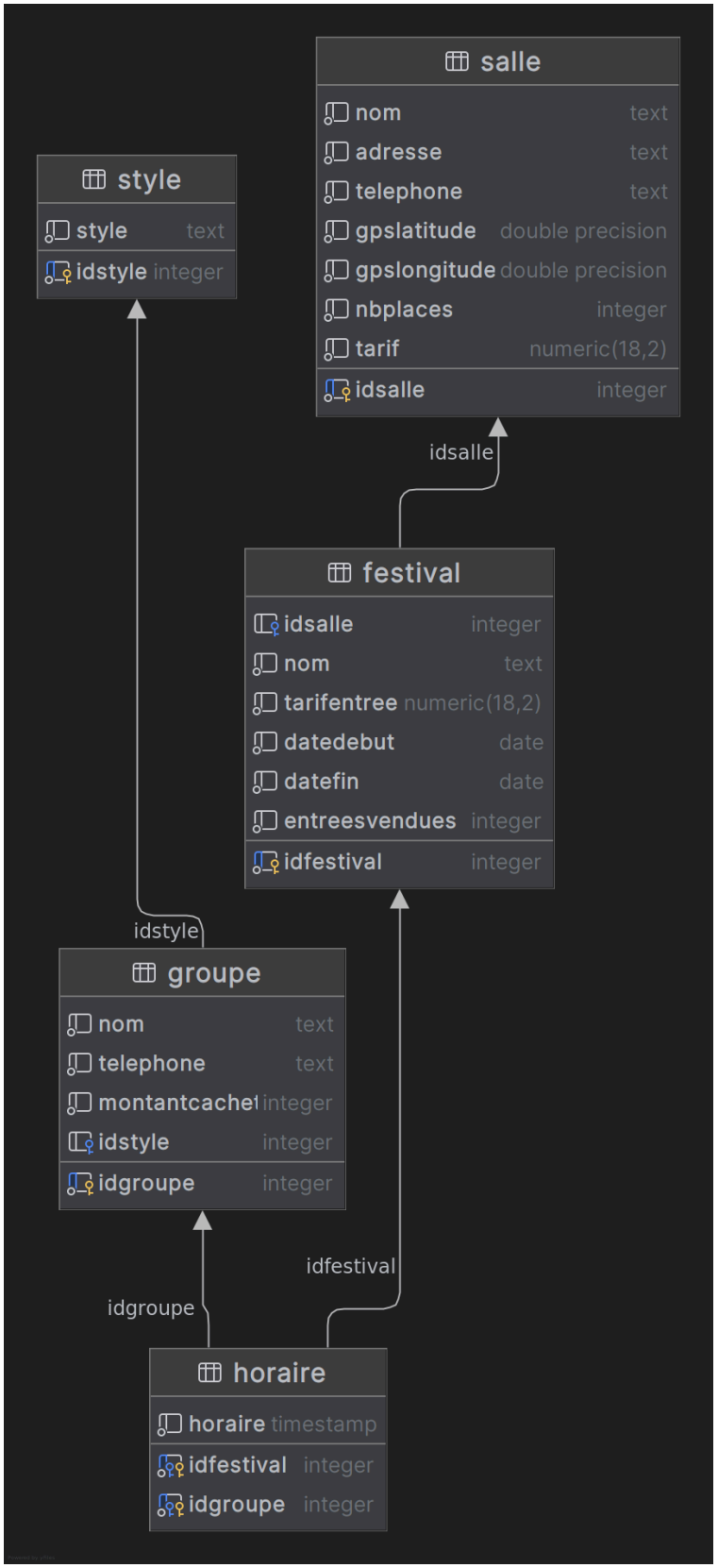
Diagramme de classe

Diagramme de classe fourni dans la SAÉ 104:



Grâce à ce diagramme de classe nous pouvons très rapidement comprendre la communication entre les différentes tables. Ceci nous permet de créer chacune des tables avec leurs colonnes correspondantes, mais aussi de savoir dans quel ordre les créer.

Modèle relationnel



Requête SQL

Pour vérifier que notre BDD fonctionne correctement nous avons dû réaliser différentes requêtes SQL, tous les résultats attendus correspondent aux résultats obtenus, hormis pour la question “ i ”.

a) Le programme d’un festival dans l’ordre chronologique

Résultat obtenu :

	festival.nom	groupe.nom
1	Les Ephemeres	Can You
2	Les Ephemeres	Gonasax
3	Les Ephemeres	Jb Sax

Requête SQL :

```
SELECT festival.nom, groupe.nom
FROM festival
   JOIN horaire using ( idfestival )
   JOIN groupe using ( idgroupe )
WHERE festival.nom = 'Les Ephemeres'
ORDER BY groupe.nom;
```

b) Les groupes pour un style donné (avec les montants cachet)

Résultat obtenu :

	nom	montantcachet
1	Can You	460
2	Gonasax	280
3	Jb Sax	460

Requête SQL :

```
SELECT groupe.nom, groupe.montantcachet
FROM groupe
   JOIN style using ( idstyle )
WHERE style.style = 'Jazz';
```

c) Pour un groupe donné, quand ont-ils été programmés

Résultat obtenu :

nom	horaire
Perlapapin	2025-02-23 22:00:00.000000

Requête SQL:

```
SELECT groupe.nom, horaire.horaire
FROM groupe
   JOIN horaire using ( idgroupe )
WHERE groupe.nom = 'Perlapapin'
```

d) Pour chaque groupe, le nombre de fois où ils ont été programmés.

Résultat obtenu:

nom	nbprogrammés
Jb Sax	1
Franklin	1
Perlapapin	1
Gonasax	1
Can You	2

Requête SQL:

```
SELECT groupe.nom, count(*) as NbProgrammés
FROM groupe
        JOIN horaire using (idgroupe)
GROUP BY groupe.nom;
```

e) Les styles proposés pour un festival donné.

Résultat obtenu:

	style
1	Jazz

Requête SQL:

```
SELECT distinct festival.nom ,style.style
FROM style
   JOIN groupe using ( idstyle )
   JOIN horaire using ( idgroupe )
   JOIN festival using ( idfestival )
WHERE festival.nom = 'Les Ephemeres';
```

f) Les styles proposés par chaque festival de la base.

Résultat obtenu:

	nom	styles
1	Les Ephemeres	Jazz
2	Les Ephemeres II	Jazz, Pipe à bois

Requête SQL:

```
SELECT  festival.nom,  string_agg(distinct  style.style,  ',  ')
as styles
FROM style
   JOIN groupe using ( idstyle )
   JOIN horaire using ( idgroupe )
   JOIN festival using ( idfestival )
GROUP BY festival.nom;
```

g) Pour chaque festival : le cout total, ce qu'il peut rapporter si la salle est complète ,ce qu'il rapporte selon le nombre d'entrées vendues.

Résultat obtenu :

	nom	couttotal	benefmax	benefreel
1	Les Ephemeres	2500	10000	8000
2	Les Ephemeres II	2500	11500	11500
3	Les Ephemeres III	3500	60000	50000

Requête SQL:

```
SELECT festival.nom, salle.tarif as coutTotal,
festival.tarifentree * salle.nbplaces as benefMax,
festival.tarifentree * festival.entreesvendues as benefReel
FROM festival
   JOIN salle using ( idsalle );
```

h) Pour chaque festival, calculez la marge en euros et en pourcentage dans les deux cas (salle complète, ou entrées vendues).

Résultat obtenu:

	nom	marge€_entrees_vendues	marge€_nbplaces	margepc_entrees_vendues	margepc_nbplaces
1	es Ephemeres	5500	7500	220	300
2	es Ephemeres II	9000	9000	360	360
3	es Ephemeres III	46500	56500	1328.57142857142857	1614.28571428571429

Requête SQL:

```
SELECT festival.nom, festival.tarifentree *
festival.entreesvendues - salle.tarif as
marge€_entrees_vendues,
      festival.tarifentree * salle.nbplaces - salle.tarif as
marge€_nbPlaces,
      ( ( festival.tarifentree * festival.entreesvendues -
salle.tarif ) / salle.tarif ) * 100 as
margePc_Entrees_vendues,
      ( ( festival.tarifentree * salle.nbplaces - salle.tarif
) / salle.tarif ) * 100 as margePc_nbPlaces
FROM festival
   JOIN salle using ( idsalle );
```

i) Les salles dans un rayon de 50 km autour de Maubeuge (coordonnées GPS de Maubeuge : latitude = 50,28◦, longitude = 3,97◦)

Résultat obtenu:

	nom	st_distancesphere
1	Le Manège	946.98593929
2	La Luna	1271.25569322
3	Le Phénix	49947.15657202

Remarque: Nous ne trouvons pas le résultat attendu, nous devrions trouver une distance de 33.16km (https://www.sunearthtools.com/fr/tools/distance.php) entre le Phénix et Maubeuge, mais nous obtenons une distance de 49.95 km

Nous avons testé trois manières différentes pour calculer la distance avec PostGIS.

Requête SQL:

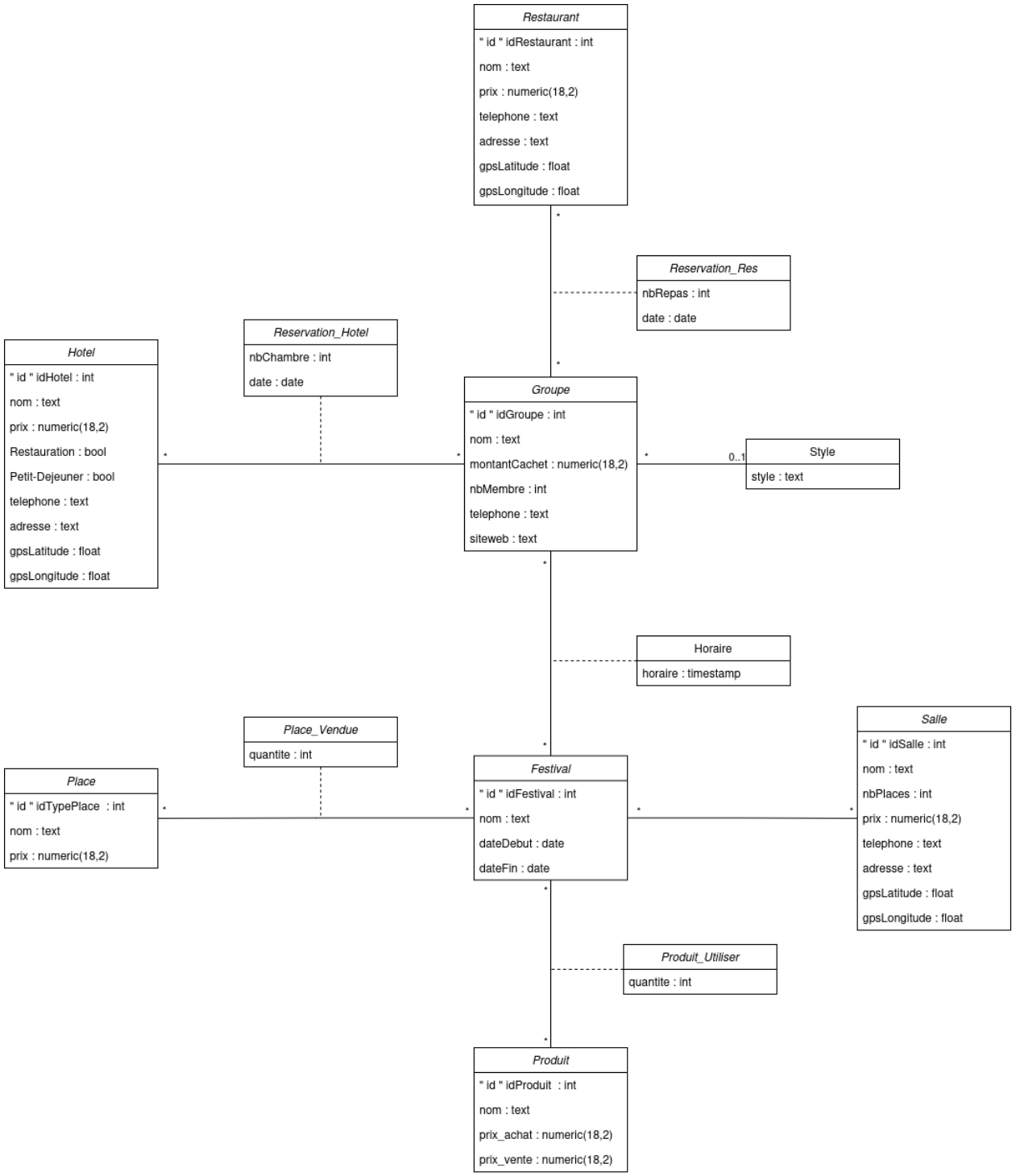
```
SELECT ST_DistanceSphere(ST_MakePoint(salle.gpslatitude,
salle.gpslongitude),ST_MakePoint(50.28, 3.97))
FROM salle;

SELECT ST_DISTANCE(
ST_GeomFromText('SRID=4326;POINT(' || gpslatitude || ' ' ||
gpslongitude || ')',3857),
ST_GeomFromText('SRID=4326;POINT(50.28 3.97)',3857))
FROM salle;

SELECT st_distancespheroid(ST_GeomFromText('POINT(' ||
salle.gpslatitude || ' ' || salle.gpslongitude || ')',
4326),ST_GeomFromText('POINT(50.28 3.97)', 4326),
'SPHEROID["WGS84",6378137,298.257223563]')
FROM salle;
```

Diagramme de classe - étendu

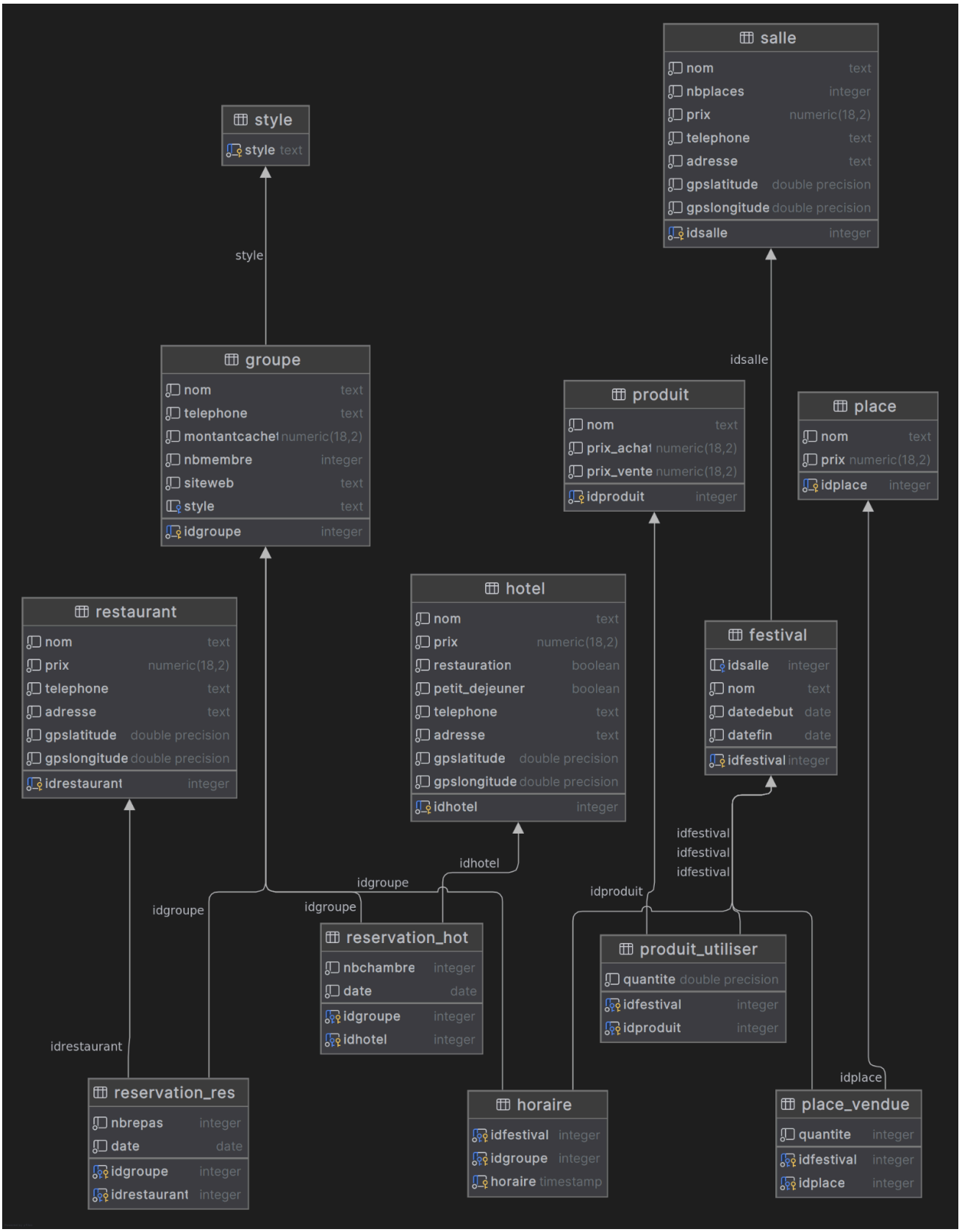
Pour notre diagramme de classe étendu, nous avons réfléchi sur comment implémenter nos données supplémentaires. C'est ainsi que nous avons décidé de rajouter quatre tables majeures: Restaurant, Hotel, Place et Produit.

Pour interconnecter ses différentes tables, nous utilisons les tables associatives suivantes: Reservation_Res, Reservation_Hot, Place_Vendue et Produit_Utiliser.

Ainsi, nous obtenons le diagramme de classe suivante:



Modèle relationnel - étendue



Requête SQL - étendue

Après avoir mis à jour notre BDD, nous devons de nouveau réaliser des requêtes SQL pour s'assurer du bon fonctionnement de notre base de données. La première moitié de nos requêtes ne vont pas fondamentalement changer. Par contre, un changement majeur est imposé dès nos requêtes " g ".

Toutes nos requêtes ont eu le résultat attendu sauf pour la " i ", même problème cité précédemment.

a) Le programme d'un festival dans l'ordre chronologique

Résultat obtenu:

	festival.nom	groupe.nom
1	Les Ephemeres	Can You
2	Les Ephemeres	Gonasax
3	Les Ephemeres	Jb Sax

Requête SQL:

```
SELECT festival.nom, groupe.nom
FROM festival
        JOIN horaire using (idfestival)
        JOIN groupe using (idgroupe)
WHERE festival.nom = 'Les Ephemeres'
ORDER BY groupe.nom;
```

b) Les groupes pour un style donné avec le montant cachet

Résultat obtenu:

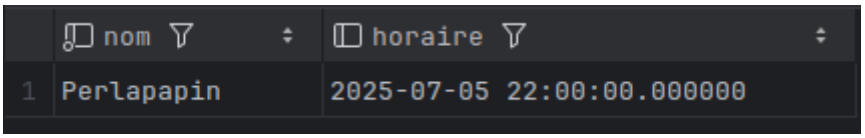
	nom	montantcachet
1	Can You	460.00
2	Gonasax	280.00
3	Jb Sax	460.00

Requête SQL:

```
SELECT groupe.nom, groupe.montantcachet
FROM groupe
        JOIN style using (style)
WHERE style.style = 'Jazz';
```

c) Pour un groupe donné, quand ont-ils été programmés

Résultat obtenu:



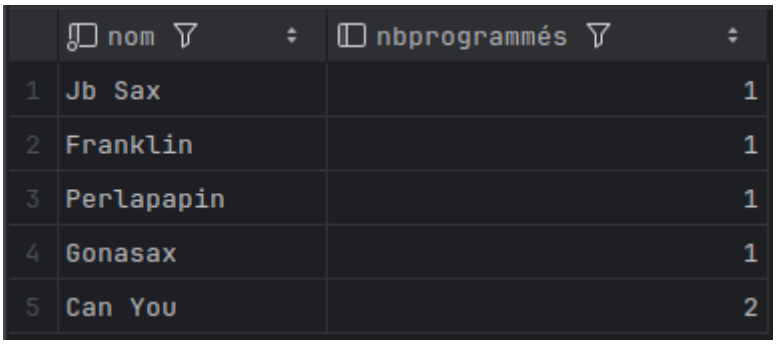
	nom	horaire
1	Perlapapin	2025-07-05 22:00:00.000000

Requête SQL:

```
SELECT groupe.nom, horaire.horaire
FROM groupe
        JOIN horaire using (idgroupe)
WHERE groupe.nom = 'Perlapapin';
```

d) Pour chaque groupe, le nombre de fois où ils ont été programmés

Résultat obtenu:



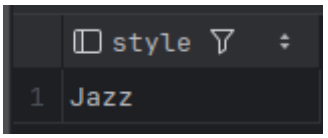
	nom	nbprogrammés
1	Jb Sax	1
2	Franklin	1
3	Perlapapin	1
4	Gonasax	1
5	Can You	2

Requête SQL:

```
SELECT groupe.nom, count(*) as NbProgrammés
FROM groupe
        JOIN horaire using (idgroupe)
GROUP BY groupe.nom;
```

e) Les styles proposés pour un festival donné

Résultat obtenu:



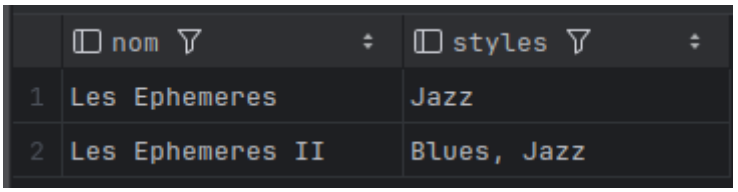
	style
1	Jazz

Requête SQL:

```
SELECT DISTINCT style
FROM style
        JOIN groupe using (style)
        JOIN horaire using (idgroupe)
        JOIN festival using (idfestival)
WHERE festival.nom = 'Les Ephemeres';
```

f) Les styles proposés par chaque festival de la base

Résultat obtenu:



	nom	styles
1	Les Ephemeres	Jazz
2	Les Ephemeres II	Blues, Jazz

Requête SQL:

```
SELECT festival.nom, string_agg(distinct style, ', ') as styles
FROM style
        JOIN groupe using (style)
        JOIN horaire using (idgroupe)
        JOIN festival using (idfestival)
```

g) Pour chaque festival : le coût total, ce qu'il peut rapporter si la salle est complète, ce qu'il rapporte selon le nombre d'entrées vendues

Une fois arrivé à cette question, nous avons rencontré notre première difficulté de récupération d'information.
En effet, nous devons utiliser un "GROUP BY" pour récupérer la somme des places vendues. En utilisant cette commande nous nous retrouvons à ne plus pouvoir récupérer les informations de la salle.
Nous devons utiliser deux "SELECT" différents afin de récupérer toutes les informations que nous souhaitons.

C'est pour cela que nous avons dû utiliser la commande "WITH", qui permet de créer deux tables temporaires contenant les informations de nos deux "SELECT" précédent et de joindre ensuite ses deux tables. Ainsi nous pouvons récupérer les informations nécessaires pour calculer le coût total en fonction des places de la salle et des tickets vendus.

Résultat obtenu:

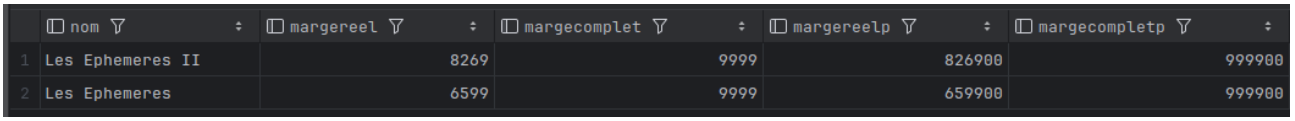
	nom	benefcomplet	benefreel
1	Les Ephemeres II	10000	55819
2	Les Ephemeres	10000	74499
3	Les Ephemeres III	<null>	<null>

Requête SQL:

```
SELECT festival.nom, sum(place.prix * place_vendue.quantite) as
BenefReel
FROM festival
        JOIN salle using (idsalle)
        JOIN place_vendue using (idfestival)
        JOIN place using (idplace)
GROUP BY festival.nom;

SELECT festival.nom, sum(place.prix * salle.nbplaces) as
BenefComplet
FROM festival
        JOIN salle using (idsalle)
        JOIN place_vendue using (idFestival)
        JOIN place using (idPlace)
GROUP BY festival.nom;

WITH totalTicketReelW as (SELECT festival.idfestival,
                                 sum(place.prix *
place_vendue.quantite +
                                     produit_utiliser.quantite *
produit.prix_vente) as BenefReel
                          FROM festival
                                   JOIN salle using (idsalle)
                                   JOIN place_vendue using
(idfestival)
                                   JOIN place using (idPlace)
                                   JOIN produit_utiliser using
(idfestival)
                                   JOIN produit using (idproduit)
                          GROUP BY festival.idfestival),
    totalTicketPlaceW as (SELECT festival.idfestival,
sum(place.prix * salle.nbplaces) as BenefComplet
                           FROM festival
                                    JOIN salle using (idsalle)
                                    JOIN place_vendue using
(idFestival)
                                    JOIN place using (idPlace)
                           WHERE place.nom ilike '%adulte%'
                           GROUP BY festival.idfestival)
SELECT nom, BenefComplet, BenefReel
FROM public.festival
        LEFT JOIN totalTicketReelW using (idfestival)
        LEFT JOIN totalTicketPlaceW using (idfestival);
```

h) **Pour chaque festival, calculez la marge en euros et en pourcentage dans les deux cas (salle complète / entrées vendues)**

Résultat obtenu:



	nom	margereel	margecomplet	margereelp	margecompletp
1	Les Ephemeres II	8269	9999	826900	999900
2	Les Ephemeres	6599	9999	659900	999900

Requête SQL:

```
WITH totalTicketReel as (SELECT festival.idfestival,
sum(place.prix * place_vendue.quantite) as BenefReel
                         FROM festival
                                  JOIN salle using (idsalle)
                                  JOIN place_vendue using
(idfestival)
                                  JOIN place using (idPlace)
                         GROUP BY festival.idfestival),
    totalTicketPlace as (SELECT festival.idfestival,
sum(place.prix * salle.nbplaces) as BenefComplet
                          FROM festival
                                   JOIN salle using (idsalle)
                                   JOIN place_vendue using
(idFestival)
                                   JOIN place using (idPlace)
                          WHERE place.nom ilike '%adulte%'
                          GROUP BY festival.idfestival)
SELECT festival.nom,
       (BenefReel - salle.prix)                          as
MargeReel,
       (BenefComplet - salle.prix)                       as
MargeComplet,
       ((BenefReel - salle.prix) / salle.prix) * 100     as
MargeReelP,
       ((BenefComplet - salle.prix) / salle.prix) * 100 as
MargeCompletP
FROM public.festival
        JOIN totalTicketReel using (idfestival)
        JOIN totalTicketPlace using (idfestival)
        JOIN salle using (idsalle);
```

i) Les salles dans un rayon de 50km autour de Maubeuge (coordonnées GPS de Maubeuge : latitude = 50.28°, longitude = 3.97°)

Résultat obtenu:

	nom	st_distancesphere
1	Le Manège	946.98593929
2	La Luna	1271.25569322
3	Le Phénix	49947.15657202

Remarque: Comme cité précédemment, nous ne trouvons pas le résultat attendu, nous devrions trouver une distance de 33.16km (https://www.sunearthtools.com/fr/tools/distance.php) entre le Phénix et Maubeuge, mais nous obtenons une distance de 49.95 km

Nous avons testé trois manières différentes pour calculer la distance avec PostGIS.

Requête SQL:

```
SELECT ST_DistanceSphere(ST_MakePoint(salle.gpslatitude,
salle.gpslongitude), ST_MakePoint(50.28, 3.97))
FROM salle;

SELECT ST_DISTANCE(
            ST_GeomFromText('SRID=4326;POINT(' || gpslatitude ||
' ' || gpslongitude || ')', 3857),
            ST_GeomFromText('SRID=4326;POINT(50.28 3.97)',
3857))
FROM salle;

SELECT st_distancespheroid(ST_GeomFromText('POINT(' ||
salle.gpslatitude || ' ' || salle.gpslongitude || ')', 4326),
                           ST_GeomFromText('POINT(50.28 3.97)',
4326), 'SPHEROID["WGS84",6378137,298.257223563]')
FROM salle;
```

j) Intégrez le coût de ces prestations supplémentaires dans le calcul du coût et de la marge d'un festival

Cette requête va nous permettre de comparer notre budget trouvé dans notre projet de recueil de besoin avec les données entrées dans la BDD.

Résultat obtenu:

	nom	recettetotal	couttotal	beneftotal
1	Les Ephemeres	11633	4040.7	7592.3
2	Les Ephemeres II	13093	3645.09	9447.91

Requête SQL:

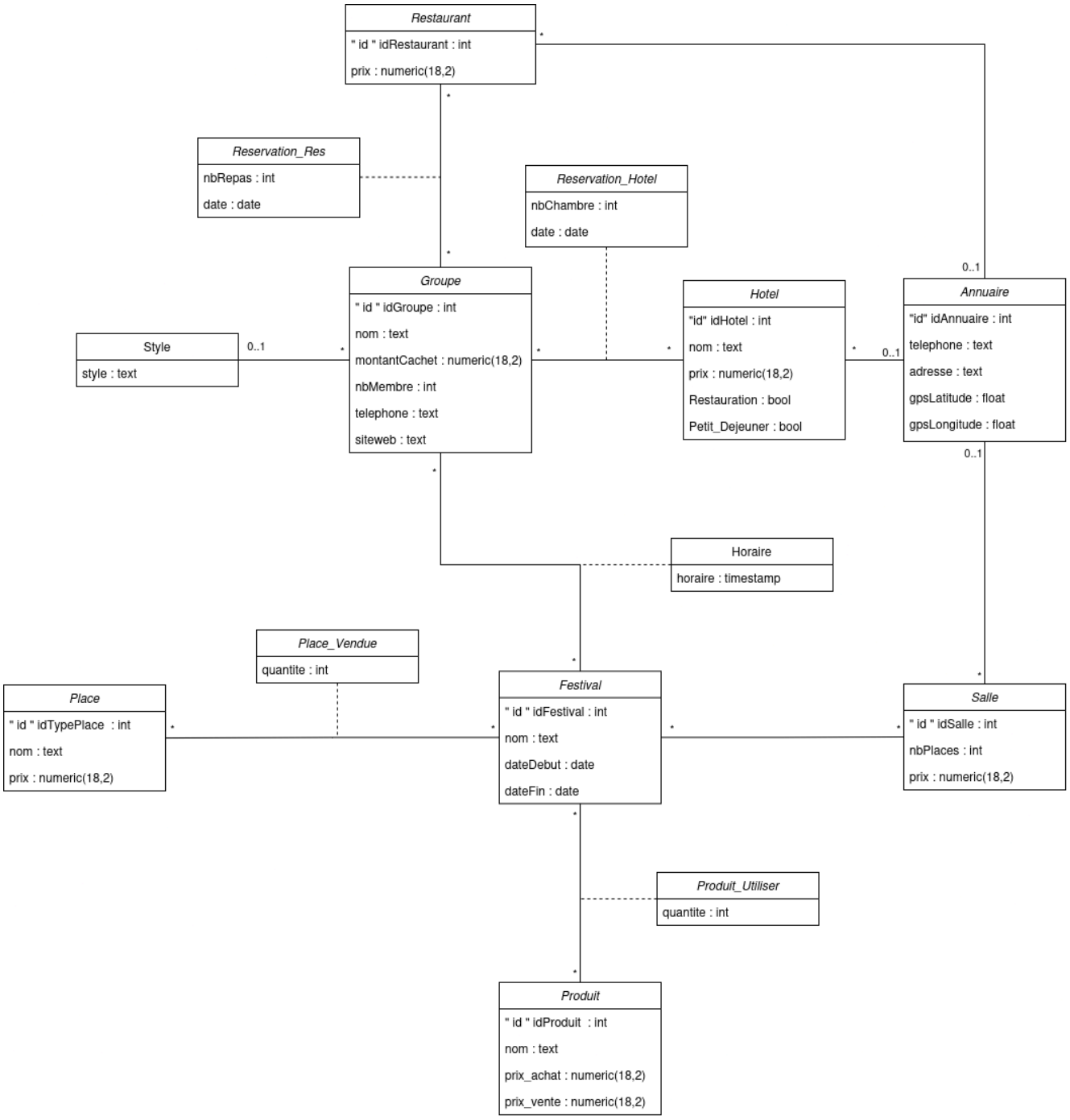
```
WITH ProduitB as (SELECT festival.idfestival,
sum(produit_utiliser.quantite * produit.prix_vente) as
BenefProduit
                  FROM festival
                           JOIN salle using (idsalle)
                           JOIN produit_utiliser using (idfestival)
                           JOIN produit using (idproduit)
                  GROUP BY festival.idfestival),
    TicketB as (SELECT festival.idfestival,
                       sum(place.prix * place_vendue.quantite) as
BenefTicket
                FROM festival
                         JOIN salle using (idsalle)
                         JOIN place_vendue using (idfestival)
                         JOIN place using (idPlace)
                GROUP BY festival.idfestival),
    ProduitC as (SELECT festival.idfestival,
sum(produit_utiliser.quantite * produit.prix_achat) as CoutProduit
                  FROM festival
                           JOIN salle using (idsalle)
                           JOIN produit_utiliser using (idfestival)
                           JOIN produit using (idproduit)
                  GROUP BY festival.idfestival),
    hotC as (SELECT festival.idfestival,
sum(reservation_hot.nbchambre) * avg(hotel.prix) as CoutHotel
             FROM festival
                      JOIN horaire using (idfestival)
                      JOIN groupe using (idgroupe)
                      JOIN reservation_hot using (idgroupe)
                      JOIN hotel using (idhotel)
             GROUP BY festival.idfestival),
    resC as (SELECT festival.idfestival,
                    sum(reservation_res.nbrepas + groupe.nbmembre)
* avg(restaurant.prix) as CoutRestaurant
             FROM festival
                      JOIN horaire using (idfestival)
                      JOIN groupe using (idgroupe)
```

```
                    JOIN reservation_res using (idgroupe)
                    JOIN restaurant using (idrestaurant)
            GROUP BY festival.idfestival),
    artC as (SELECT festival.idfestival, sum(groupe.montantcachet)
as CoutArtiste
            FROM festival
                    JOIN horaire using (idfestival)
                    JOIN groupe using (idgroupe)
            GROUP BY festival.idfestival)
SELECT festival.nom,
      BenefProduit + BenefTicket
as RecetteTotal,
      CoutHotel + CoutProduit + CoutRestaurant + CoutArtiste
as CoutTotal,
      BenefProduit + BenefTicket - (CoutHotel + CoutProduit +
CoutRestaurant + CoutArtiste) as BenefTotal
FROM public.festival
        JOIN produitB using (idfestival)
        JOIN TicketB using (idfestival)
        JOIN ProduitC using (idfestival)
        JOIN hotC using (idfestival)
        JOIN resC using (idfestival)
        JOIN artC using (idfestival)
ORDER BY nom;
```

Problème rencontré

Vers la fin de notre réalisation, nous nous sommes rendu compte que nous aurions dû mettre en commun les informations des lieux. En effet, les données comme “adresse”, “gpsLatitude”, “gpsLongitude” et “téléphone” sont communs à tous nos lieux (restaurant, hotel et salle).

Ainsi avec ce diagramme de classe suivant nous pourrions créer une table “annuaire” contenant les informations communes de nos différents lieux.



Conclusion

Nous venons de voir comment nous avons mis en place une base de données en suivant les restrictions imposées, en l'adaptant à nos besoins d'information et en la vérifiant à travers différentes requêtes SQL.

Nous avons pu nous rendre compte de l'importance du diagramme de classe, si celui-ci est erroné ou mal organisé la relation des diverses informations devient très vite compliquée. Grâce à la ressource R.105 (Introduction aux bases de données et SQL) nous avons pu la réaliser sans problème majeur.

De plus, ce projet nous a permis de comprendre la facilité de récupération d'information à travers les requêtes SQL. Qu'importe les données futures entrées dans la BDD, ses requêtes vont toujours être fonctionnelles et permettent de rapidement avoir des informations à jour.

Ce type de pratique peut être un élément majeur dans nos futures entreprises, ce projet est donc un passage crucial et important afin de pouvoir manipuler les BDD et de pouvoir répondre aux différentes demandes d'entreprise/client.

Ainsi, nous avons dû être organisés et attentifs aux détails afin de créer une base de données cohérente et fonctionnelle.

Nous vous remercions pour le temps que vous nous avez accordé en lisant ce compte rendu,

CHOURAIH Baptiste,
COLLOT Grégoire,
MARECAILLE-HENAUT Mattieu,
SIMSEK Dilara.