



Master 1 MIASHS - MQME 2025-2026

Projet SQL

Gestion d'un festival de musique

Réalisé par :

Amadou Garanke SOW
Pacifique NIYOKWIZERA

PROJET DU COURS DE SQL

January 5, 2026

Contents

1	Introduction	2
2	Modélisation conceptuelle	2
2.1	Présentation du modèle conceptuel	2
2.2	Justification des cardinalités	4
3	Transformation en modèle logique relationnel	4
3.1	Passage du MCD au MLD	4
3.2	Règles de transformation	6
4	Qualité du modèle et normalisation	6
4.1	Objectif de la normalisation	6
4.2	Dépendances fonctionnelles (DF)	7
4.3	Vérification des formes normales	7
4.3.1	Première forme normale (1FN)	7
4.3.2	Deuxième forme normale (2FN)	8
4.3.3	Troisième forme normale (3FN)	8
4.4	Cohérence des données et contraintes	9
5	Implémentation SQL	9
6	Analyse et exploitation des données	10
6.1	Cohérence et qualité des données	10
6.2	Exemples de requêtes complexes	10
6.2.1	Vue SQL : programmation lisible	10
6.2.2	Sous-requête : concerts sur la scène de capacité maximale	11
6.2.3	Requête analytique avancée : CTE + fenêtrage (RANK)	12
7	Améliorations possibles	13

1 Introduction

Ce projet s'inscrit dans le cadre du module de bases de données du Master 1 MIASHS. L'objectif est de concevoir, implémenter et exploiter une base de données relationnelle permettant la gestion d'un festival de musique, depuis la programmation des concerts jusqu'à la participation du public.

Le travail couvre l'ensemble du processus de conception :

- modélisation conceptuelle (MCD/UML),
- transformation en modèle logique relationnel (MLD),
- implémentation SQL,
- analyse des données à l'aide de requêtes simples et avancées.

2 Modélisation conceptuelle

2.1 Présentation du modèle conceptuel

La modélisation conceptuelle a été réalisée selon une approche UML/Merise. Elle permet d'identifier les entités principales du système ainsi que leurs relations et cardinalités.

Les entités principales sont :

- **Organisateur**
- **Scène**
- **Artiste**
- **Concert**
- **Participant**
- **Billet**

Modèle Conceptuel de Données (UML) - Festival de Musique

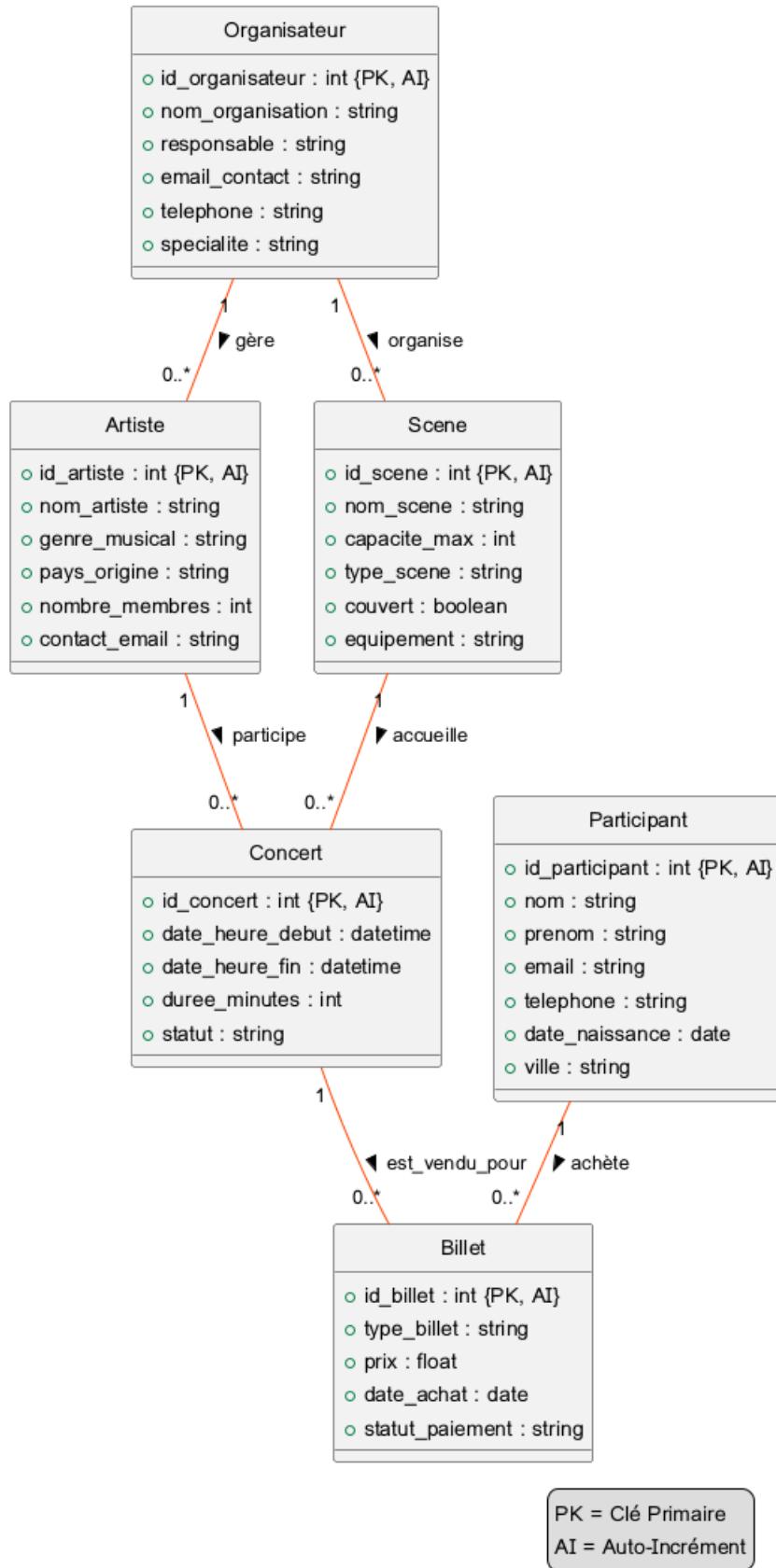


Figure 1: Modèle Conceptuel (MCD) — Classes, attributs et associations

2.2 Justification des cardinalités

- Un organisateur peut gérer plusieurs scènes (1,N), une scène dépend d'un seul organisateur.
- Une scène peut accueillir plusieurs concerts (1,N), un concert se déroule sur une seule scène.
- Un artiste peut participer à plusieurs concerts (1,N), un concert concerne un seul artiste.
- Un participant peut acheter plusieurs billets (1,N).
- Un concert peut être associé à plusieurs billets (1,N).

Ces cardinalités traduisent une logique métier réaliste du fonctionnement d'un festival.

3 Transformation en modèle logique relationnel

3.1 Passage du MCD au MLD

Le passage du modèle conceptuel au modèle logique s'effectue selon des règles de transformation classiques :

- chaque entité devient une table ;
- chaque identifiant devient une clé primaire ;
- les associations 1,N sont traduites par des clés étrangères ;
- les associations N,N sont matérialisées par une table associative enrichie.

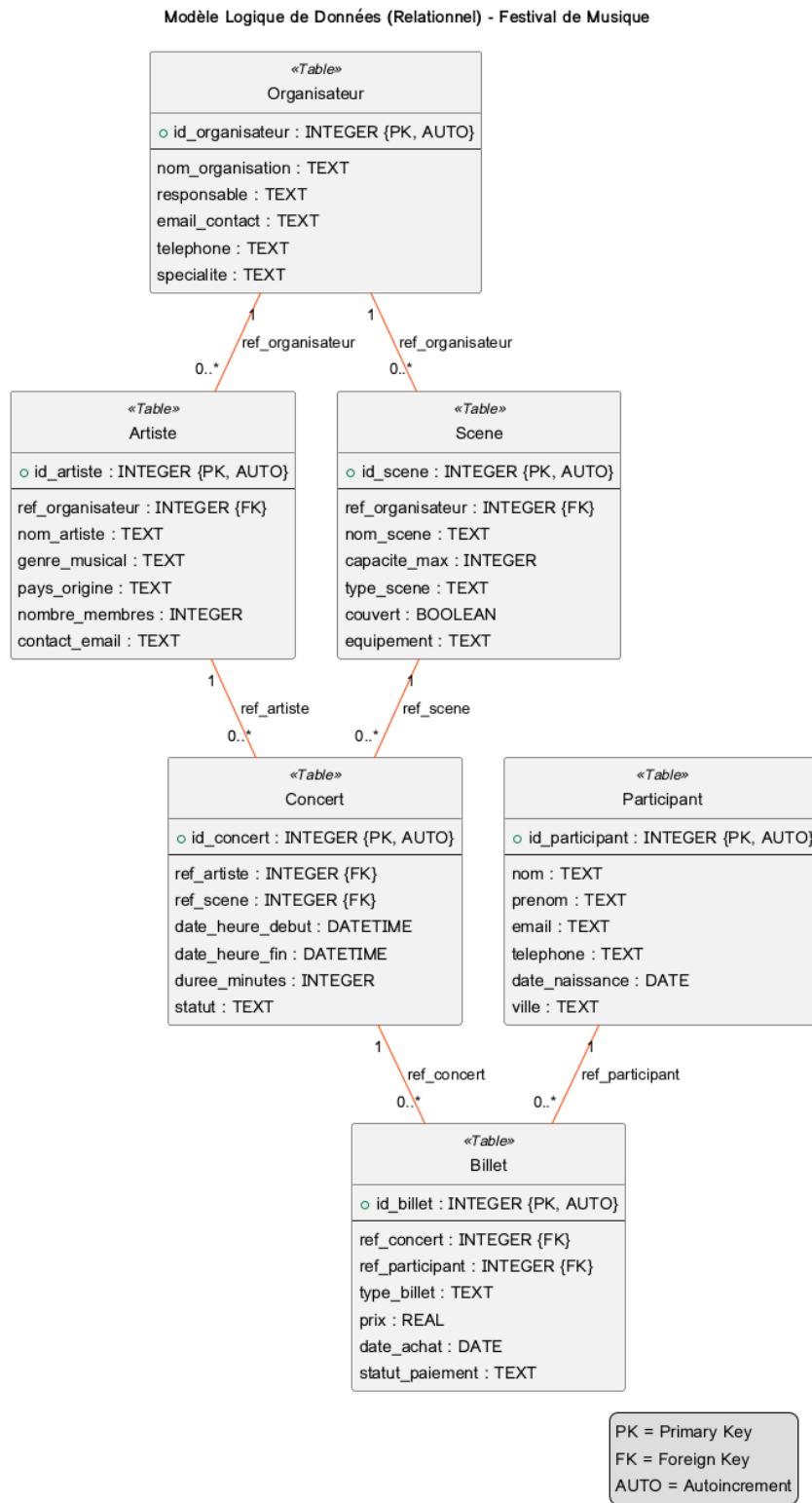


Figure 2: Modèle Logique Relationnel (MLD)

3.2 Règles de transformation

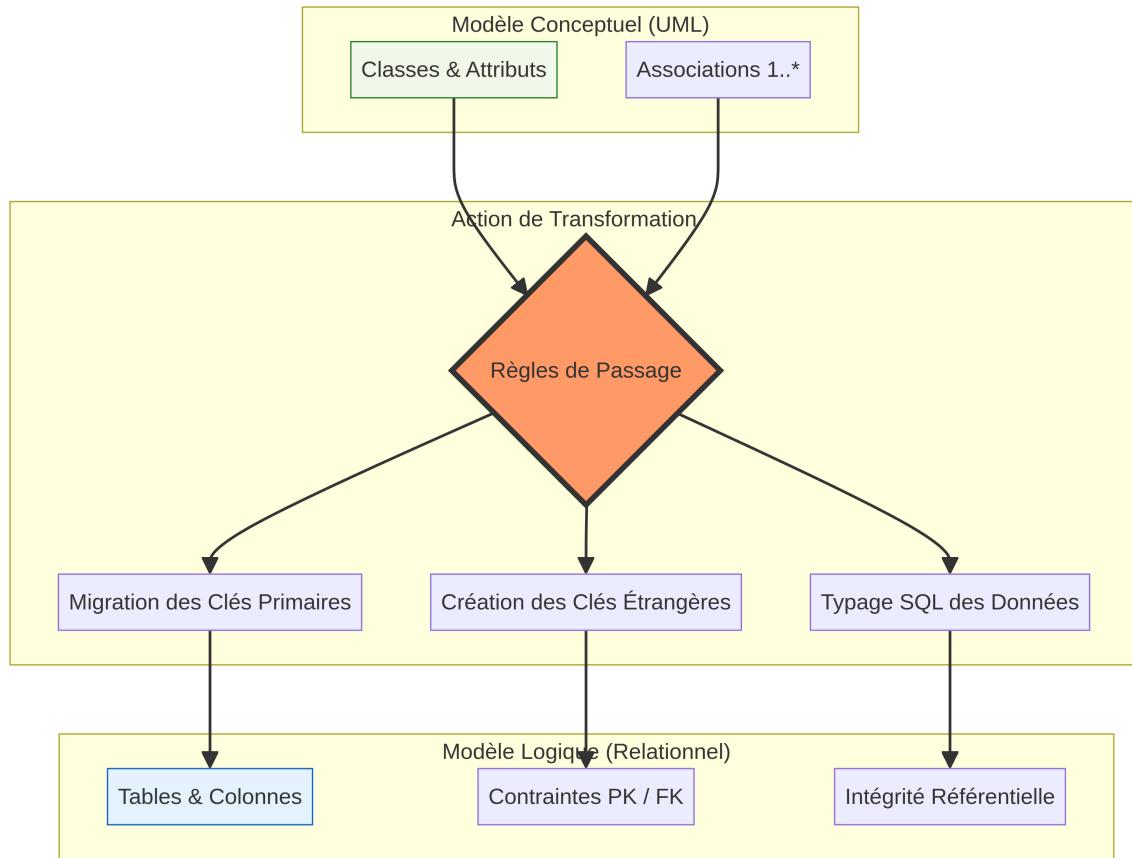


Figure 3: Transformation du MCD vers le MLD puis vers SQL

Les clés primaires sont conservées et les clés étrangères assurent l'intégrité référentielle.

4 Qualité du modèle et normalisation

4.1 Objectif de la normalisation

La normalisation vise à améliorer la qualité du schéma relationnel en :

- réduisant les redondances (éviter de stocker plusieurs fois la même information) ;
- limitant les anomalies de mise à jour (insertion, suppression, modification) ;
- garantissant une structure cohérente et maintenable.

Dans notre cas, l'objectif est de disposer d'un schéma relationnel robuste pour analyser la programmation et la billetterie du festival.

4.2 Dépendances fonctionnelles (DF)

Les dépendances fonctionnelles principales (au sens relationnel) sont :

- **Organisateur** :

$$id_organisateur \rightarrow nom_organisation, email_contact$$

- **Scène** :

$$id_scene \rightarrow id_organisateur, nom_scene, capacite_max$$

- **Artiste** :

$$id_artiste \rightarrow nom_artiste, genre_musical, pays_origine$$

- **Concert** :

$$id_concert \rightarrow ref_artiste, ref_scene, date_heure_debut$$

- **Participant** :

$$id_participant \rightarrow nom, email$$

- **Billet** :

$$id_billet \rightarrow ref_concert, ref_participant, prix$$

Ces dépendances fonctionnelles sont directement implémentées dans la base de données à l'aide de clés primaires et de clés étrangères, garantissant ainsi la cohérence entre le modèle logique et l'implémentation SQL.

4.3 Vérification des formes normales

4.3.1 Première forme normale (1FN)

La 1FN impose que :

- chaque attribut soit atomique (pas de listes, pas de valeurs multiples dans une même cellule) ;
- chaque table représente un ensemble de tuples.

Dans notre schéma, les attributs (`nom_artiste`, `capacite_max`, `date_heure`, etc.) sont atomiques et ne contiennent pas de répétitions ou de listes. **Le modèle respecte donc la 1FN.**

4.3.2 Deuxième forme normale (2FN)

La 2FN impose que :

- la table soit en 1FN ;
- chaque attribut non-clé dépende de **toute** la clé primaire.

Ici, toutes les tables utilisent des clés primaires **simples** (`id_...`). Il n'existe donc pas de dépendance partielle (cas typique des clés composées). **Le modèle respecte donc la 2FN.**

4.3.3 Troisième forme normale (3FN)

La 3FN impose que :

- la table soit en 2FN ;
- aucun attribut non-clé ne dépende transitivement de la clé primaire.

Exemple de risque de dépendance transitive Si l'on stockait dans la table `Concert` l'attribut `ref_organisateur`, on obtiendrait la dépendance fonctionnelle suivante :

$$id_{concert} \rightarrow ref_scene \rightarrow ref_organisateur$$

L'attribut `ref_organisateur` dépendrait alors transitivement de la clé primaire `id_concert`. Cela introduirait une redondance ainsi qu'un risque d'incohérence (par exemple : un concert associé à une scène donnée mais à un organisateur différent).

Choix retenu Afin de respecter la troisième forme normale (3FN), la table `Concert` ne contient que les informations directement liées à l'événement :

- `ref_scene` : localisation du concert ;
- `ref_artiste` : artiste programmé ;
- `date_heure_debut` : planification temporelle du concert.

L'organisateur est ainsi déduit indirectement via la table `Scene`, ce qui permet d'éviter toute dépendance transitive et de limiter la redondance des données.

Le modèle respecte donc la 3FN.

Lien avec l'implémentation SQL Les principes de normalisation sont directement respectés dans l'implémentation SQL. Chaque table correspond à une entité du MLD, les dépendances fonctionnelles sont garanties par les clés primaires, et les relations sont matérialisées par des clés étrangères. Cela assure une cohérence stricte entre le modèle conceptuel, le modèle logique et la base de données effectivement exploitée dans le notebook.

4.4 Cohérence des données et contraintes

La cohérence est assurée par :

- **Clés primaires** : unicité des enregistrements ;
- **Clés étrangères** : intégrité référentielle (`Billet` ne peut exister sans `Concert`) ;
- **Contraintes de domaine** : par exemple `capacite_max > 0`, `prix >= 0` ;
- **Unicité** : emails uniques (organisateur, participant).

Ces contraintes limitent les incohérences et garantissent une base exploitable pour les analyses.

5 Implémentation SQL

La base de données a été implémentée en SQL (SQLite). Les tables sont créées avec des contraintes garantissant la qualité des données :

- `PRIMARY KEY` pour l'unicité,
- `FOREIGN KEY` pour l'intégrité référentielle,
- `NOT NULL`, `UNIQUE` et `CHECK` pour la validation.

Le jeu de données contient environ **180 enregistrements**, avec :

- peu de scènes,
- beaucoup de participants,
- un nombre important de billets.

afin de permettre des analyses pertinentes.

6 Analyse et exploitation des données

6.1 Cohérence et qualité des données

Les données respectent les contraintes du modèle :

- aucune référence orpheline,
- des valeurs réalistes (dates, prix, capacités),
- possibilité d'analyser des cas variés (participants avec ou sans billet).

6.2 Exemples de requêtes complexes

6.2.1 Vue SQL : programmation lisible

Objectif Créer une vue fournissant une représentation claire et exploitable de la programmation du festival, en combinant les informations relatives aux concerts, aux scènes et aux artistes.

Code SQL

```
CREATE VIEW v_programmation AS
SELECT
    c.id_concert,
    date(c.date_heure_debut) AS jour,
    time(c.date_heure_debut) AS heure,
    s.nom_scene           AS scene,
    a.nom_artiste         AS artiste,
    a.genre_musical       AS genre
FROM Concert c
JOIN Scene s  ON s.id_scene    = c.ref_scene
JOIN Artiste a ON a.id_artiste = c.ref_artiste;
```

...	id_concert	jour	heure	scene	artiste	genre	
0	1	2025-07-18	16:00:00	Grande Scene	Artiste_01	Rock	
1	2	2025-07-18	18:00:00	Electro Dome	Artiste_01	Rock	
2	3	2025-07-18	20:00:00	Indie Garden	Artiste_01	Rock	
3	4	2025-07-19	16:00:00	Grande Scene	Artiste_02	Electro	
4	5	2025-07-19	18:00:00	Electro Dome	Artiste_02	Electro	
5	6	2025-07-19	20:00:00	Indie Garden	Artiste_02	Electro	
6	7	2025-07-20	16:00:00	Grande Scene	Artiste_03	Pop	
7	8	2025-07-20	18:00:00	Electro Dome	Artiste_03	Pop	
8	9	2025-07-20	20:00:00	Indie Garden	Artiste_03	Pop	
9	10	2025-07-21	16:00:00	Grande Scene	Artiste_04	Jazz	
10	11	2025-07-21	18:00:00	Electro Dome	Artiste_04	Jazz	
11	12	2025-07-21	20:00:00	Indie Garden	Artiste_04	Jazz	

Résultat et interprétation

La vue v_programmation agit comme une table virtuelle offrant une lecture simplifiée de la programmation du festival. Elle permet d'accéder rapidement aux informations essentielles (jour, heure, scène, artiste, genre) sans avoir à réécrire les jointures complexes entre les tables.

6.2.2 Sous-requête : concerts sur la scène de capacité maximale

Objectif Identifier les concerts organisés sur la scène disposant de la plus grande capacité d'accueil, afin d'analyser quels artistes sont programmés sur la scène principale du festival.

Principe La sous-requête permet de déterminer la capacité maximale parmi toutes les scènes. Cette valeur est ensuite utilisée dans la requête principale pour filtrer uniquement les concerts correspondant à cette scène.

Code SQL (extrait du notebook)

```

SELECT
    c.id_concert,
    s.nom_scene AS scene,
    a.nom_artiste AS artiste,
    c.date_heure_debut AS debut
FROM Concert c
JOIN Scene s ON s.id_scene = c.ref_scene
JOIN Artiste a ON a.id_artiste = c.ref_artiste
WHERE s.capacite_max = (SELECT MAX(capacite_max) FROM Scene)
ORDER BY c.date_heure_debut;

```

...	id_concert	scene	artiste	debut	
0	1	Grande Scene	Artiste_01	2025-07-18 16:00:00	
1	4	Grande Scene	Artiste_02	2025-07-19 16:00:00	
2	7	Grande Scene	Artiste_03	2025-07-20 16:00:00	
3	10	Grande Scene	Artiste_04	2025-07-21 16:00:00	

Résultat et interprétation

Le résultat montre que seuls les concerts programmés sur la *Grande Scene*, qui possède la capacité maximale du festival, sont sélectionnés. On observe que ces concerts correspondent aux créneaux les plus visibles et accueillent des artistes majeurs de la programmation.

Cette requête permet ainsi :

- d'identifier les scènes stratégiques du festival ;
- d'analyser la répartition des artistes selon la capacité d'accueil ;
- d'étudier les choix d'organisation liés à la gestion du public.

6.2.3 Requête analytique avancée : CTE + fenêtrage (RANK)

Objectif Classer les artistes selon leur nombre de concerts programmés, en utilisant une CTE (WITH) pour calculer les statistiques, puis une fonction fenêtre (RANK()) pour attribuer un rang (en gérant les ex-æquo).

Code SQL

```
WITH stats AS (
    SELECT
        a.id_artiste,
        a.nom_artiste AS artiste,
        a.genre_musical AS genre,
        COUNT(c.id_concert) AS nb_concerts
    FROM Concert c
    JOIN Artiste a ON a.id_artiste = c.ref_artiste
    GROUP BY a.id_artiste, a.nom_artiste, a.genre_musical
),
classement AS (
    SELECT
        *,
        RANK() OVER (ORDER BY nb_concerts DESC) AS rang
    FROM stats
)
```

```
SELECT artiste, genre, nb_concerts, rang  
FROM classement  
ORDER BY rang, artiste;
```

The screenshot shows a table with four columns: 'artiste', 'genre', 'nb_concerts', and 'rang'. There are four rows of data, each with a row index (0, 1, 2, 3) and a blue circular icon with a white number.

...	artiste	genre	nb_concerts	rang	grid icon
0	Artiste_01	Rock	3	1	grid icon
1	Artiste_02	Electro	3	1	grid icon
2	Artiste_03	Pop	3	1	grid icon
3	Artiste_04	Jazz	3	1	grid icon

Résultat et interprétation

Cette requête produit un classement des artistes les plus programmés. La fonction RANK() attribue le même rang aux artistes ayant un même nombre de concerts, ce qui est utile pour identifier des têtes d'affiche ou des artistes récurrents dans la programmation.

7 Améliorations possibles

Plusieurs améliorations peuvent être envisagées :

- ajout d'une table **Paiement** (date, moyen, statut),
- gestion des types de billets (standard, VIP, réduit),
- contrôle automatique des capacités via des triggers,
- ajout d'index pour améliorer les performances,
- gestion des annulations et remboursements.

Conclusion

Ce projet a permis de mettre en œuvre de manière cohérente l'ensemble des étapes de conception d'une base de données relationnelle, depuis la modélisation conceptuelle jusqu'à l'analyse avancée des données via SQL. La rigueur apportée à la normalisation, à l'intégrité des données et à l'exploitation par des requêtes complexes garantit une base fiable, évolutive et pleinement adaptée à la gestion d'un festival de musique.