Documentation
Base de données seulement

Documentation pour différent éléments

James Benone

v. 2025.08.21 - 15:54

Thursday 11 June 2026

Table des matières

1. Récupérer des données

1.1 Commande de base

SQL

```sql
SELECT * FROM `Charts`
```

Cette commande permet de récupérer toutes les données d'une table, en l'occurence, `Charts` ici.

Voici le résultat que vous devriez obtenir avec les données de test :

id	title	content	created_at	id_user
1	test	content-test	2025-10-09	1
2	mermaid_db	content-mermaid_db	2025-10-09	1
3	mermaid_api	content-mermaid_api	2025-10-09	1
4	mermaid_infra	content-mermaid_infra	2025-10-09	1
5	mermaid_call	content-mermaid_call	2025-10-09	1
6	reseau maison	content-reseau_maison	2025-10-13	2
7	nas-infra	content-nas-infra	2025-10-13	2
8	planif-travaux	content-planif-travaux	2025-10-13	2
9	projet-git-commit	content-projet-git-commit	2025-10-26	4
10	projet-gantt	content-projet-gantt	2025-10-26	4
11	projet-diagram-classe	content-projet-diagram-classe	2025-10-26	4
12	projet-mcd	content-projet-mcd	2025-10-26	4
13	infra_projet	content-infra_projet	2025-10-30	5
14	merge	content-merge	2025-11-01	7

1.2 Récupérer une partie de la table

On veut récupérer tout de la table `Users` faut le mot de passe pour des raisons de sécurité. Pour se faire, on va éxecuter la commande suivante :

SQL

```sql
SELECT `id`, `email`, `code`, `created_at` FROM `Users`;
```

Voici les données que vous devriez avoir :

id	email	code	created_at
1	james@benone.ch		2025-10-01 15:04:56.000
2	martin.lavalais@ikmail.com		2025-10-03 19:45:12.000
3	michel@example.com	194012	2025-10-06 11:50:03.000
4	elton-john@yahoo.dev		2025-10-17 09:03:12.000
5	sasukedu92@gmail.com		2025-10-23 19:45:54.000
6	johnwick@gmail.com	847102	2025-10-30 05:23:23.000
7	sogo.sogot@gmail.com		2025-11-02 17:30:34.000

1.3 Récupérer une donnée spécifique

Supposons qu'on veut récupérer la liste des charts de l'utilisateur `1`.

On va faire la commande suivante :

SQL

```sql
SELECT * FROM `Charts` WHERE `id_user` = 1;
```

Voici les données que vous devriez avoir :

id	title	content	created_at	id_user
1	test	content-test	2025-10-09	1
2	mermaid_db	content-mermaid_db	2025-10-09	1
3	mermaid_api	content-mermaid_api	2025-10-09	1
4	mermaid_infra	content-mermaid_infra	2025-10-09	1
5	mermaid_call	content-mermaid_call	2025-10-09	1

1.4 Récupérer avant une date

Supposons que l'on veut récupérer tous les utilisateurs qui ont créer un compte avant novembre.

On peut récupérer cette donnée via la commande suivante :

SQL

```sql
SELECT `id`, `email`, `code`, `created_at` FROM `Users` WHERE `created_at` < '2025-11-01';
```

Voici les données récupérés :

id	email	code	created_at
1	james@benone.ch		2025-10-01 15:04:56.000
2	martin.lavalais@ikmail.com		2025-10-03 19:45:12.000
3	michel@example.com	194012	2025-10-06 11:50:03.000
4	elton-john@yahoo.dev		2025-10-17 09:03:12.000
5	sasukedu92@gmail.com		2025-10-23 19:45:54.000
6	johnwick@gmail.com	847102	2025-10-30 05:23:23.000

1.5 Récupérer des champs vides

On veut récupérer tous les utilisateurs qui on un code.

La commande est la suivante :

SQL

```sql
SELECT `id`, `email`, `code`, `created_at` FROM `Users` WHERE `code` IS NOT NULL;
```

Voici les données récupérés

id	email	code	created_at
3	michel@example.com	194012	2025-10-06 11:50:03.000
6	johnwick@gmail.com	847102	2025-10-30 05:23:23.000

1.6 Récuperer le nombre de colonne

On veut récupérer le nombre de `Charts` par `Users`. Pour cela, on va utiliser la fonction `count()`.

Voici un exemple :

SQL

```sql
SELECT `id_user`, count(`id`) FROM `Charts`;
```

Voici les données que vous devriez avoir :

id_user	count("id")
1	14

On a eu le nombre de `Charts`, mais pas comme on voulait.

Ici il y a 2 problèmes.

Dans un premier temps, on n'a pas le nombre **PAR** utilisateur et c'est marqué `count("id")`.

Mais je vous rassure, il y a moyen d'y remédier !

1.7 Récupérer des données groupés

Reprenons la commande d'avant mais rajoutons un petit truc et regardons ce que cela donne :

SQL

```sql
SELECT `id_user`, count(`id`) FROM `Charts` GROUP BY `id_user`;
```

Voici les données récupérés :

id_user	count("id")
1	5
2	3
4	4
5	1
7	1

On a déjà rêgler un problème, mais il en reste un...

1.8 Mettre des alias

Il est possible de mettre des alias pour les colonnes mais aussi pour les tables.

Voici un exemple avec les 2 :

SQL

```sql
SELECT c.id_user, count(c.id) AS nb_charts FROM `Charts` AS c GROUP BY c.id_user;
```

Voici les données récupérés :

id_user	nb_charts
1	5
2	3
4	4
5	1
7	1

1.9 Trier les données

Supposons que maintenant, on veut trier en fonction de celui qui a créé le plus de `Charts`

À savoir : Il y a 2 types de tri, `DESC` et `ASC`. Si vous ne mettez rien, par défaut, c'est `ASC`. `ASC` tri par ordre croissant alors que `DESC` tri en décroissant.

On peut via la commande suivante :

SQL

```sql
SELECT c.id_user, count(c.id) AS "nb_charts" FROM `Charts` AS c GROUP BY c.id_user ORDER BY nb_charts DE
SC;
```

Voici les données récupérés :

id_user	nb_charts
1	5
4	4
2	3
5	1
7	1

1.10 Limiter

On garde la même requête qu'avant mais on veut limiter le nombre de retour à, par exemple, 3.

On peut via la commande suivante :

SQL

```sql
SELECT c.id_user, count(c.id) AS "nb_charts" FROM `Charts` AS c GROUP BY c.id_user ORDER BY nb_charts DE
SC LIMIT 3;
```

Voici les données récupérés :

id_user	nb_charts
1	5
4	4
2	3

1.11 Récupérer 2 tables

On veut récupérer les 2 tables liés avec les champs suivants : `id, email created_at, nb_charts,`

Pour se faire, on peut le faire via la commande suivante :

SQL

```sql
SELECT u.id, u.email, u.created_at, count(c.id) AS "nb_charts" FROM `Users` AS u
JOIN `Charts` AS c ON c.id_user = u.id
GROUP BY u.id
ORDER BY nb_charts DESC;
```

Voici les données récupérés :

id	email	created_at	nb_charts
1	james@benone.ch	2025-10-01 15:04:56.000	5
4	elton-john@yahoo.dev	2025-10-17 09:03:12.000	4
2	martin.lavalais@ikmail.com	2025-10-03 19:45:12.000	3
5	sasukedu92@gmail.com	2025-10-23 19:45:54.000	1
7	sogo.sogot@gmail.com	2025-11-02 17:30:34.000	1

Mais il y a un problème, on n'a pas tous les utilisateurs.

Il est tout de même possible de les afficher avec la commande suivante :

SQL

```sql
SELECT u.id, u.email, u.created_at, count(c.id) AS "nb_charts" FROM `Users` AS u
LEFT JOIN `Charts` AS c ON c.id_user = u.id
GROUP BY u.id
ORDER BY nb_charts DESC;
```

Voici les données récupérés :

id	email	created_at	nb_charts
1	james@benone.ch	2025-10-01 15:04:56.000	5
4	elton-john@yahoo.dev	2025-10-17 09:03:12.000	4
2	martin.lavalais@ikmail.com	2025-10-03 19:45:12.000	3
5	sasukedu92@gmail.com	2025-10-23 19:45:54.000	1
7	sogo.sogot@gmail.com	2025-11-02 17:30:34.000	1
3	michel@example.com	2025-10-06 11:50:03.000	0
6	johnwick@gmail.com	2025-10-30 05:23:23.000	0

Il faut savoir qu'il existe plusieurs types de `JOIN`. `LEFT`, `RIGHT`, `INNER`, etc...

Par défaut, si vous ne mettez que `JOIN`, il l'exécutera comme un `INNER JOIN`.

5 novembre 2025

2. Gérer des données

2.1 INSERT

2.1.1 Simple

On peut insérer les données simplement.

Pour la table utilisateur, Comme ceci :

SQL

```sql
INSERT INTO `Users` VALUES (8, "pascal@gmail.com", "Super", null, now());
```

On a créer un utilisateur avec l'identifiant `8`, l'email `pascal@gmail.com`, le mot de passe `Super`, aucun code et la date de création à maintenant.

2.1.2 Colonnes précisés

Si on ajoute un utilisateur à la base de données, très souvent, on ne va pas donner l'identifiant, on laisse la base de données faire.

Pour cela, on peut via la commande suivante :

SQL

```sql
INSERT INTO `Users`(`email`, `password`, `code`, `created_at`) VALUES ("kevin@gmail.com", "Super",
null, now());
```

On a créer un utilisateur avec l'identifiant généré par la base de données, l'email `kevin@gmail.com`, le mot de passe `Super`, aucun code et la date de création à maintenant.

2.2 UPDATE

Pour modifier une donnée, on peut le faire via la commande suivante :

SQL

```sql
UPDATE `Users` SET `code` = null WHERE `id` = 3;
```

On a modifié l'utilisateur avec l'identifiant `3` et on lui a dit qu'il n'a maintenant plus de code.

2.3 DELETE

Pour supprimer une donnée, on peut le faire via la commande suivante :

SQL

```sql
DELETE FROM `Users` WHERE `id` = 8;
```

On a supprimé l'utilisateur avec l'identifiant `8`. Il n'apparaîtra donc plus dans la base de données.

5 novembre 2025

3. Gérer une base de données

3.1 Créer une base de données

SQL

```sql
CREATE DATABASE test_db;
```

Ceci créer une base de données nommée `test_db`.

Mais il y a un problème, supposons qu'elle existe, cela va nous créer une erreur. Nous pouvons toute fois y remédier avec un petit ajout :

SQL

```sql
CREATE DATABASE IF NOT EXISTS test_db;
```

Il va alors créer la base de données **UNIQUEMENT** si aucune base de données à ce nom.

3.2 Supprimer une base de données

SQL

```sql
DROP DATABASE test_db;
```

Ceci supprime une base de données nommée `test_db`.

Mais même cas, si cela n'existe pas, il y aura une erreur et comme pour avant, il y a :

SQL

```sql
DROP DATABASE IF EXISTS test_db;
```

Il va alors supprimer la base de données **UNIQUEMENT** si il y a une base de données à ce nom.

3.3 Créer une table

Supposons que nous avons créer la base de données `test_db`.

Nous voulons rajouter une table simple qui contient `id`, `email`, `password`, `code` et `created_at`. Rien de plus, rien de moins. On va l'appeler `Users`.

Il faut savoir plusieurs choses dans ce cas, il faut penser aux types des variables (int, varchar, date, datetime, etc...) et il faut toujours une clé primaire par table. La clé primaire est un identifiant unique d'une colonne. Dans notre cas, sa sera `id` et on va l'auto-incrémenté (augmente à chaque ajout d'une ligne).

Voici la commande pour créer une base de données :

SQL

```sql
CREATE TABLE test_db.Users (
    id int(11) auto_increment NOT NULL,
    email varchar(256) NOT NULL,
```

```
    password varchar(128) NOT NULL,
    code varchar(10) NULL DEFAULT NULL,
    created_at TIMESTAMP DEFAULT now() NOT NULL,
    CONSTRAINT Users_PK PRIMARY KEY (id),
    CONSTRAINT Users_email_UNIQUE UNIQUE KEY (email),
    CONSTRAINT Users_code_UNIQUE UNIQUE KEY (code)
)
ENGINE=InnoDB
DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;
```

Il y a quelques particularités que je vais expliqué ici.

Déjà, vous avez remarqué qu'à quasiment toutes les lignes, il y a `NULL` ou `NOT NULL`. Il faut généralement en mettre à chaque ligne et cela détermine si la variable peut-être null ou doit avoir un contenu.

À la ligne `created_at`, vous voyez `DEFAULT now()`. Cela veut dire que la valeur par défaut de cette variable sera la date d'ajout de la ligne. Il y a aussi `DEFAULT NULL`, cela veut juste dire que par défaut, la valeur est vide, qu'elle ne contient rien.

Il y a aussi `CONSTRAINT Users_PK PRIMARY KEY (id)` qui définit la clé primaire.

Il y a aussi `CONSTRAINT Users_email_UNIQUE UNIQUE KEY (email)` qui définit que cette valeur doit être unique et ne peut pas être en doublon dans la table. **À SAVOIR**, si une colonne est unique et nullable, null est la seule "valeur" qui peut-être attribué à plusieurs endroits car il signifie qu'il n'y a rien.

`ENGINE=InnoDB` définit le moteur qui gère la table.

Les lignes ci-dessous définissent la table de caractère utilisé pour cette table. Elle peut varié en fonction des besoins linguistiques.

SQL

```
DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci`
```

3.4 Supprimer une table

Comme pour les bases de données, on peut mettre un `IF EXISTS`.

SQL

```
DROP TABLE IF EXISTS test_db.Users;
```

Cela supprime la table `Users` ainsi que toutes ces données.

5 novembre 2025

4. Gérer des utilisateurs

4.1 Créer un utilisateur

IMPORTANT : Si vous voulez créer un utilisateur avec des paramètres IP spécifiques, je vous recommande fortement de vous renseigner sur le fonctionnement des IP et des tables IP.

4.1.1 Créer un utilisateur local

Supposons que l'on veut créer un utilisateur `admin` et qu'il ne soit accessible qu'en local. On peut via la commande suivante :

SQL

```sql
CREATE USER "admin"@"localhost" IDENTIFIED BY "Super";
```

Cela créer l'utilisateur `admin` avec le mot de passe `Super` qui ne peut-être accéder qu'en local ou avec l'IP `127.0.0.1`.

4.1.2 Créer un utilisateur public

Supposons que vous avez une équipe de sécurité qui est externe mais qui doit avec accès à une table de type `Logs`.

On peut via la commande suivante :

SQL

```sql
CREATE USER "cyber-team"@"%" IDENTIFIED BY "Super";
```

Cela créer l'utilisateur `cyber-team` avec le mot de passe `Super` qui peut-être accéder de n'importe ou.

4.1.3 Créer un utilisateur interne

Supposons que vous avez une équipe qui gère les données manuellements et qui à besoin des accès, vous pouvez via la commande suivante :

SQL

```sql
CREATE USER "bi-dep"@"10.5.40.0/24" IDENTIFIED BY "Super";
```

Cela créer l'utilisateur `bi-dep` avec le mot de passe `Super` qui peut-être accéder par tous les ordinateurs ayant une IP qui commence par `10.5.40`.

4.2 Supprimer un utilisateur

Supposons que l'équipe de sécurité ne travail plus pour vous et vous devez supprimer l'utilisateur.

Vous pouvez via la commande suivante :

SQL

```sql
DROP USER "cyber-team"@"%";
```

Cela supprime l'utilisateur `cyber-team`.

4.3 Gérer les droits

4.3.1 Ajouter des droits

Supposons qu'on veut attribuer les droits de gérer les données montrés dans Gérer les données et Récupérer les données.

On peut via la commande suivante :

SQL

```sql
GRANT SELECT, INSERT, UPDATE, DELETE ON test_db.* TO "bi-dep"@"10.5.40.0/24";
```

Cela donne les droits de récupérer, ajouter, changer ou supprimer les données de toutes les tables de la base de données `test_db`.

4.3.2 Ajouter tous les droits

ATTENTION : Il est important de ne jamais faire ceci à un utilisateur publiquement accéssible.

Supposons que l'on veut donner tous les droits à notre administrateur.

Il faut pour cela, éxécuter la commande suivante :

SQL

```sql
GRANT ALL PRIVILEGES ON *.* TO "admin"@"localhost" WITH GRANT OPTION;
```

Cela donne tous les droits sur toutes les tables de toutes les bases de données. et `WITH GRANT OPTION` donne également le droit de gérer les autres utilisateurs.

4.3.3 Supprimer des droits

Supposons que l'on veut empécher au `bi-dep` de modifier ou supprimer les données de la table `Logs`.

On peut via la commande suivante :

SQL

```sql
REVOKE UPDATE, DELETE ON test_db.Logs TO "bi-dep"@"10.5.40.0/24";
```

4.3.4 Actualiser les droits

Pour actualiser les droits et les appliqués, il faut dire au logiciel qui gère les bases de données de mettre les permissions à jour via la commande suivante :

SQL

```sql
FLUSH PRIVILEGES;
```

5 novembre 2025

Tables des figures