# SQL

## Data Query Language DQL

Langage d'interrogation de données

### SELECT... FROM...

Permet d'obtenir des données de table(s) en spécifiant le(s) champ(s) à afficher dans la réponse.

```
SELECT champ1,champ2...
FROM Table1,Table2...
```

**Exemple :**

Obtenir l'affichage des noms et prénoms des enregistrements de la table questionnaire :

```
SELECT nom, prenom
FROM questionnaire;
```

**Exemple :**

Obtenir uniquement les noms :

```
SELECT nom
FROM questionnaire;
```

**Exemple :**

Obtenir toutes les informations de la table questionnaire :

```
SELECT *
FROM questionnaire;
```

### ...WHERE...

Le WHERE Permet de poser une condition pour sélectionner les enregistrements à afficher :

```
SELECT nomChamp(s)
FROM table(s)
```

```
WHERE condition(s);
```

où **condition** est une expression booléenne

**Exemple :** Affichage de toutes les informations relatives aux questionnaires de Caen

```
SELECT *
FROM questionnaire
WHERE ville='CAEN';
```

**Exemple :** Affichage de toutes les informations relatives aux questionnaires de Caen dont le nom commence par un B

```
SELECT *
FROM questionnaire
WHERE ville='CAEN' AND nom LIKE 'B%';
```

#### Opérateur logiques et arithmétiques

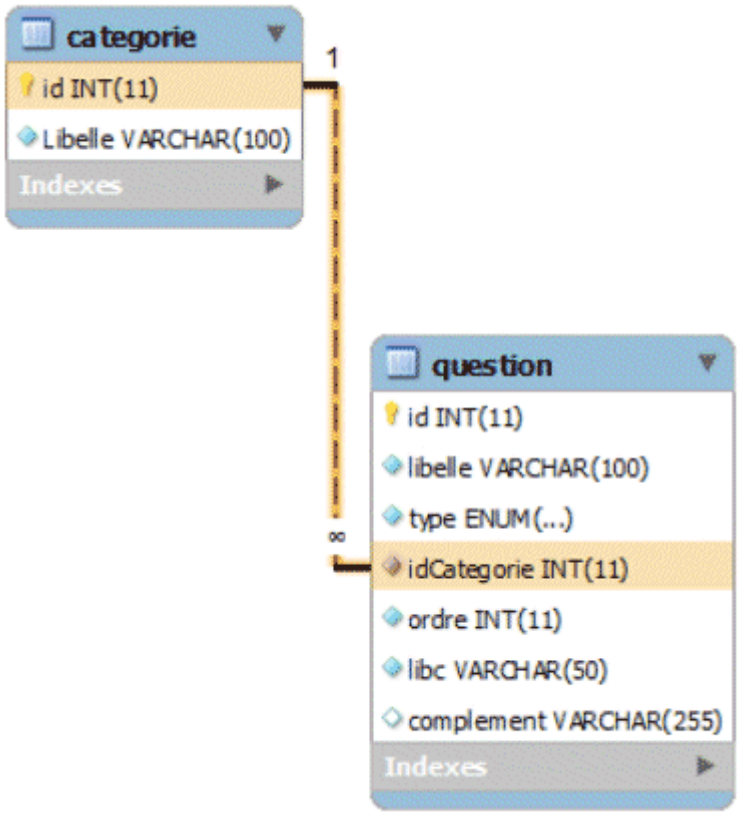
| Opérateur | Rôle | Exemple |
|---|---|---|
| = | Comparaison (égalité) | Ville='CAEN' |
| != ou <> | Comparaison (différence) | Ville!='CAEN' |
| < | Inférieur à | age<10 |
| > | Supérieur à | age>10 |
| ⇐ | Inférieur ou égal à | age<=10 |
| >= | Supérieur ou égal à | age>=10 |
| IN | Dans un ensemble de valeurs | age IN 10,11,12 |
| NOT IN | Nom compris dans un ensemble | NOT IN 20,21,22,23 |
| BETWEEN..AND | Entre une valeur et une autre | age BETWEEN 10 AND 20 |
| NOT BETWEEN..AND | Non compris entre une valeur et une autre | age NOT BETWEEN 10 AND 20 |
| LIKE | Comme... permet d'utiliser le caractère % pour simuler n'importe quelle suite de caractères, et _ pour un seul caractère | Ville LIKE 'C%' |
| NOT LIKE | Pas comme... permet d'utiliser le caractère % pour simuler n'importe quelle suite de caractères, et _ pour un seul caractère | Ville NOT LIKE 'C%' |
| IS | Est... Utilisable avec False/True/Null | Password IS Null |
| Is NOT | N'est pas... | Password IS NOT NULL |
| AND | Et logique | Ville='Caen' AND age>20 |
| OR | Ou logique | Ville!='Caen' OR age<=20 |

## SELECT FROM multi-tables

Il est possible d'extraire les informations à partir de plusieurs tables, à la condition que les tables mentionnées aient un champ commun permettant de les relier (on parle de jointure dans ce cas).

#### -- Jointure exprimée dans le WHERE (Ancienne norme)

**Exemple :**




Affichage des questions et de leur catégorie :

```
SELECT Categorie.libelle, question.libelle
FROM question, Categorie
WHERE question.idCategorie=categorie.id;
```

Le champ **idCategorie** de la **Question** correspond au champ **id** de la **Categorie**

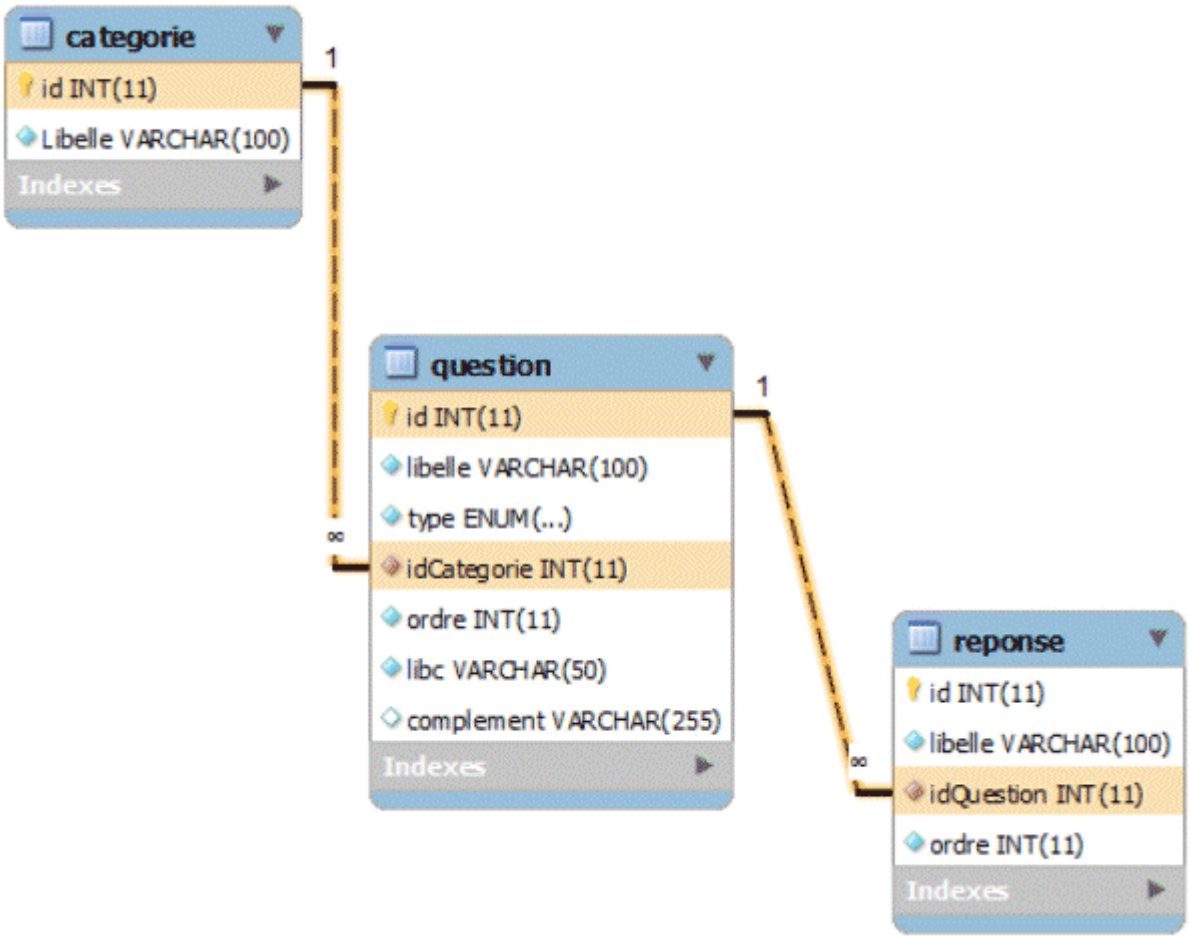
**Jointure exprimée par un JOIN (Nouvelle norme)**

**Exemple :**
Affichage des questions et de leur catégorie :

```
SELECT Categorie.libelle, question.libelle
FROM question JOIN Categorie
ON question.idCategorie=categorie.id;
```

Le champ **idCategorie** de la **Question** correspond au champ **id** de la **Categorie**

**Exemple :**
Affichage des questions, de leur catégorie et des réponses par question :




```
SELECT Categorie.libelle, question.libelle, reponse.libelle
FROM (question JOIN Categorie ON question.idCategorie=categorie.id)
JOIN Reponse
ON reponse.idQuestion=Question.id;
```

Le champ **idCategorie** de la **Question** correspond au champ **id** de la **Categorie**
Le champ **idQuestion** de la **Reponse** correspond au champ **id** de la **Question**

## GROUP BY

La clause **GROUP BY** permet de regrouper les résultats sur la valeur commune d'un champ, pour effectuer des calculs,
en utilisant les fonctions de regroupement ci-dessous :

| **Opérateur** | **Rôle** | **Exemple** |
|---|---|---|
| COUNT(*) | Calcul le nombre d'enregistrements retournés | SELECT COUNT(*) FROM Questionnaire |
| COUNT(DISTINCT nomChamp) | Calcul le nombre d'enregistrements différents retournés | SELECT COUNT(DISTINCT Ville) FROM Questionnaire |
| AVG(nomChamp) | Calcul de la moyenne des valeurs de nomChamp | SELECT Ville, AVG(age) FROM Etudiant GROUP BY Ville |
| SUM(nomChamp) | Calcul de la somme des valeurs de nomChamp | SELECT YEAR(DateFacture), SUM(montant) FROM Facture GROUP BY YEAR(DateFacture) |
| MIN(nomChamp) | Calcul du minimum des valeurs de nomChamp | SELECT MIN(age) FROM Etudiant |

| Opérateur | Rôle | Exemple |
|---|---|---|
| MAX(nomChamp) | Calcul du maximum des valeurs de nomChamp | SELECT MAX(age) FROM Etudiant |

Tous les champs intégrés dans le **SELECT** et ne faisant pas partie d'une fonction de regroupement doivent être mentionnés dans le **GROUP BY**.

**Exemples :**

Calcul du nombre de questionnaires :

```
SELECT COUNT(*)
FROM questionnaire;
```

Calcul du nombre de questionnaires par Ville :

```
SELECT Ville, COUNT(*)
FROM questionnaire
GROUP BY Ville;
```

## HAVING

La clause **HAVING** permet de poser une condition (comme le WHERE), mais en utilisant l'une des fonctions de regroupement.

**Exemples :**

Calcul du nombre de questionnaires pour les villes ayant plus de 5 questionnaires :

```
SELECT Ville, COUNT(*)
FROM questionnaire
GROUP BY Ville
HAVING COUNT(*)>5;
```

Si une partie de la condition à poser n'intègre pas de fonction de regroupement, elle doit être spécifiée dans un WHERE.

Calcul du nombre de questionnaires pour les villes commençant par un C et ayant plus de 5 questionnaires :

```
SELECT Ville, COUNT(*)
FROM questionnaire
WHERE Ville LIKE 'C%'
GROUP BY Ville
HAVING COUNT(*)>5;
```

## ORDER BY

La clause **ORDER BY** permet d'ordonner les réponses dans le résultat suivant la valeur d'1 ou de plusieurs champs :

**Exemples :**
Affichage des questionnaires classés par ordre de date de naissance du candidat :

```
SELECT *
FROM questionnaire
ORDER BY dNaiss;
```

Il est possible de préciser l'ordre : ASC⇒Ascendant par défaut ou DESC⇒Descendant Affichage des questionnaires classés par ordre décroissant de date de naissance du candidat :

```
SELECT *
FROM questionnaire
ORDER BY dNaiss DESC;
```

Sur plusieurs champs : par ordre alphabétique croissant des villes, et par ordre de date de naissance décroissant :

```
SELECT *
FROM questionnaire
ORDER BY Ville ASC,dNaiss DESC;
```

En utilisant les numéro d'ordre des champs, plutôt que leurs noms :

```
SELECT Ville, dNaiss, nom
FROM questionnaire
ORDER BY 1 ASC,2 DESC;
```

## LIMIT

La clause **LIMIT** permet de spécifier le nombre d'enregistrements à retourner, et à partir duquel.

**Exemples :**

Elle peut s'utiliser avec 1 argument (permettant de préciser le nombre d'enregistrements à retourner :
Affichage des 10 premiers questionnaires

```
SELECT *
FROM questionnaire
LIMIT 10;
```

Utilisée avec 2 arguments, le premier désigne l'enregistrement à partir duquel on affiche les résultats (0 pour le premier), et le second le nombre à extraire :
Affichage des 10 premiers questionnaires

```
SELECT *
FROM questionnaire
LIMIT 0,10;
```

Affichage des 10 suivants :

```
SELECT *
FROM questionnaire
LIMIT 10,10;
```

## UNION

Le mot clé **UNION** permet de combiner plusieurs résultats de SELECT pour en obtenir 1 seul :
L'union n peut fonctionner que si les résultats contiennent le même nombre de champs.

Affichage des 10 premiers questionnaires et des questionnaires dont le nom commence par un 'C' :

```
SELECT * FROM questionnaire LIMIT 10
UNION
SELECT * FROM questionnaire WHERE nom LIKE 'C%';
```

Les enregistrements ne proviennent pas forcément des mêmes tables :

```
SELECT Categorie.libelle from Categorie
UNION
SELECT Question.libelle From Question;
```

## VIEW

Une vue permet de stocker dans la BDD une instruction SQL correspondant à un résultat :

**Création :**

```
CREATE VIEW `Q_Caen` AS
SELECT * from Questionnaire
WHERE ville='CAEN';
```

La vue produit un résultat dynamique, mis à jour en fonction des modifications effectuées dans la base.

**Suppression :**

```
DROP VIEW `Q_Caen`;
```

**Mise à jour :**

```
ALTER VIEW `Q_Caen` AS
SELECT * from Questionnaire
```

```
WHERE ville='CAEN' AND ...;
```

# Opération CRUD sur les enregistrements d'une base

- C : Create
- R : Read
- U : Update
- D : Delete

## Create : insertion de données

INSERT INTO...

#### Insertion complète

Le nombre de valeurs à insérer est égal au nombre de champs

```
INSERT INTO parution
VALUES (1,"Programmez","Bob","2013-02-01");
```

#### Insertion incomplète

Le nombre de valeurs à insérer est inférieur au nombre de champs.
Vous devez dans ce cas préciser les champs concernés par l'insertion après le nom de la table

```
INSERT INTO parution(numP,titre)
VALUES(2,"Linux mag");
```

#### Causes possibles d'erreurs sur INSERT INTO

- Insertion provoquant des doublons sur un champ à index unique (ex : clé primaire)
- Non respect du type de données (ex : ajout d'une chaîne dans un champ de type numérique)
- Non respect de l'ordre ou du nombre de champs
- Non respect de l'intégrité référentielle (valeur de la clé étrangère non présente dans la clé référencée)

## Read : Lecture de données

SELECT...

L'accès à un enregistrement spécifique ne doit se faire que par condition sur la clé primaire, pour être certain en réponse de n'obtenir qu'un seul enregistrement.

```
SELECT * FROM parution
WHERE numP=1;
```

Compter le nombre d'enregistrements :

```
SELECT COUNT(*) as nb FROM Parution;
```

## Update : Modification de données

UPDATE...

La mise à jour d'un enregistrement spécifique ne doit se faire que par condition sur la clé primaire, pour être certain en réponse de n'obtenir qu'un seul enregistrement.

```
UPDATE parution SET titre="Programmezzzzzzzzzzzzz"
WHERE numP=1;
```

**Mise à jour de plusieurs champs :**

```
UPDATE parution SET titre="Programmezzzzzzzzzzzzzz", redacteur="moi"
WHERE numP=1;
```

**Les Causes possibles d'erreurs sur UPDATE sont les mêmes que sur le INSERT INTO**

## Delete : Suppression de données

DELETE...

La suppression d'un enregistrement spécifique ne doit se faire que par condition sur la clé primaire, pour être certain en réponse de ne supprimer qu'un seul enregistrement.

**Suppression d'un enregistrement :**

```
DELETE FROM parution
WHERE numP=1;
```

**Les Causes possibles d'erreurs sur DELETE**
Suppression d'un enregistrement père associé à des enregistrements fils (En l'absence de Suppression en cascade)

**Suppression de plusieurs enregistrements :**

```
DELETE FROM parution
WHERE titre like "b%";
```

**Suppression de tous les enregistrements :**

```
```

```
DELETE FROM parution;
```

ou

```
DELETE FROM parution
WHERE 1=1;
```

From:
http://slamwiki2.kobject.net/ - **SlamWiki 2.1**

Permanent link:
**http://slamwiki2.kobject.net/sio/bloc1/3/sql?rev=1701340113**

Last update: **2023/11/30 11:28**