SQL

LE LANGAGE SQL ET LES BASES DE DONNÉES

SQL SELECT

L'utilisation la plus courante de SQL consiste à lire des données issues de la base de données. Cela s'effectue grâce à la commande **SELECT**, qui retourne des enregistrements dans un tableau de résultat. Cette commande peut sélectionner une ou plusieurs colonnes d'une table.

identifiant	prenom	nom	ville
1	Pierre	Dupond	Paris
2	Sabrina	Durand	Nantes
3	Julien	Martin	Lyon
4	David	Bernard	Marseille
5	Marie	Leroy	Grenoble

```
SELECT ville
FROM client
```

ville
Paris
Nantes
Lyon
Marseille
Grenoble

SQL SELECT

Avec la même table client il est possible de lire **plusieurs colonnes** à la fois. Il suffit tout simplement de séparer les noms des champs souhaités par une virgule.

identifiant	prenom	nom	ville
1	Pierre	Dupond	Paris
2	Sabrina	Durand	Nantes
3	Julien	Martin	Lyon
4	David	Bernard	Marseille
5	Marie	Leroy	Grenoble

```
SELECT prenom, nom
FROM client
```

prenom	nom
Pierre	Dupond
Sabrina	Durand
Julien	Martin
David	Bernard
Marie	Leroy

SQL SELECT

Il est possible de retourner automatiquement **toutes les colonnes** d'un tableau sans avoir à connaître le nom de toutes les colonnes. Il faut simplement utiliser le caractère « ***** » (étoile). C'est un joker qui permet de sélectionner toutes les colonnes :

```
SELECT * FROM client
```

identifiant	prenom	nom	ville
1	Pierre	Dupond	Paris
2	Sabrina	Durand	Nantes
3	Julien	Martin	Lyon
4	David	Bernard	Marseille
5	Marie	Leroy	Grenoble

SQL DISTINCT

L'utilisation de la commande SELECT en SQL permet de lire toutes les données d'une ou plusieurs colonnes. Cette commande peut potentiellement afficher des lignes en doubles. Pour éviter des redondances dans les résultats il faut simplement ajouter **DISTINCT** après le mot SELECT.

identifiant	prenom	nom
1	Pierre	Dupond
2	Sabrina	Bernard
3	David	Durand
4	Pierre	Leroy
5	Marie	Leroy

```
SELECT DISTINCT prenom
FROM client
```

prenom
Pierre
Sabrina
David
Marie

SQL AS (Alias)

Dans le langage SQL il est possible d'utiliser des **alias** pour renommer temporairement une **colonne** ou une **table** dans une requête. Cette astuce est particulièrement utile pour faciliter la lecture des requêtes.

Sur une colonne

```
SELECT colonne1 AS c1, colonne2
FROM `table`
```

Sur une table

```
SELECT *
FROM `nom_table` AS t1
```

SQL AS (Alias) …

Les **alias** sont utiles pour simplifier le nom de certaines **colonnes** ou **tables**.

```
SELECT p_id, p_nom_fr_fr AS nom, p_description_fr
AS description,  p_prix_euro AS prix

FROM `produit`
```

id	nom	description	prix
1	Ecran	Ecran de grandes tailles.	399.99
2	Clavier	Clavier sans fil.	27
3	Souris	Souris sans fil.	24
4	Ordinateur portable	Grande autonomie et et sacoche offerte.	700

SQL WHERE

La commande **WHERE** dans une requête SQL permet **d'extraire les lignes** d'une base de données qui respectent une **condition**. Cela permet d'obtenir uniquement les informations désirées.

Syntaxe Générale

```
SELECT nom_colonnes
FROM nom_table  WHERE condition
```

id	nom	nbr_commande	ville
1	Paul	3	paris
2	Maurice	0	rennes
3	Joséphine	1	toulouse
4	Gérard	7	paris

```
SELECT *
FROM client
WHERE ville = 'paris'
```

id	nom	nbr_commande	ville
1	Paul	3	paris
4	Gérard	7	paris

SQL WHERE ...

Les Opérateurs

Il existe plusieurs **opérateurs** de comparaisons. La liste ci-jointe présente quelques uns des opérateurs les plus couramment utilisés.

Opérateur	Description
=	Égale
<>	Pas égale
!=	Pas égale
>	Supérieur à
<	Inférieur à
>=	Supérieur ou égale à
<=	Inférieur ou égale à
IN	Liste de plusieurs valeurs possibles
BETWEEN	Valeur comprise dans un intervalle donnée (utile pour les nombres ou dates)
LIKE	Recherche en spécifiant le début, milieu ou fin d'un mot.
IS NULL	Valeur est nulle
IS NOT NULL	Valeur n'est pas nulle

SQL AND & OR

Une requête SQL peut être restreinte à l'aide de la condition **WHERE**. Les opérateurs logiques **AND** et **OR** peuvent être utilisées au sein de la commande **WHERE** pour combiner des conditions.

id	nom	categorie	stock	prix
1	ordinateur	informatique	5	950
2	clavier	informatique	32	35
3	souris	informatique	16	30
4	crayon	fourniture	147	2

Opérateur **AND**

```
SELECT * FROM produit
WHERE categorie = 'informatique' AND stock < 20
```

id	nom	categorie	stock	prix
1	ordinateur	informatique	5	950
3	souris	informatique	16	30

SQL AND & OR

id	nom	categorie	stock	prix
1	ordinateur	informatique	5	950
2	clavier	informatique	32	35
3	souris	informatique	16	30
4	crayon	fourniture	147	2

Opérateur **OR**

```
SELECT * FROM produit
WHERE nom = 'ordinateur' OR nom = 'clavier'
```

id	nom	categorie	stock	prix
1	ordinateur	informatique	5	950
2	clavier	informatique	32	35

SQL AND & OR

id	nom	categorie	stock	prix
1	ordinateur	informatique	5	950
2	clavier	informatique	19	35
3	souris	informatique	26	30
4	crayon	fourniture	147	2

Combiner **AND** et **OR**

```
SELECT * FROM produit
WHERE ( categorie = 'informatique' AND stock < 20 )  OR ( categorie = 'fourniture' AND stock < 200 )
```

id	nom	categorie	stock	prix
1	ordinateur	informatique	5	950
2	clavier	informatique	19	35
4	crayon	fourniture	147	2

SQL IN

L'opérateur logique **IN** dans SQL s'utilise avec la commande **WHERE** pour vérifier si une colonne est égale à une des valeurs comprise dans un jeu de valeurs déterminé.

C'est une méthode simple pour vérifier si une colonne est égale à une valeur OU une autre valeur OU une autre valeur et ainsi de suite, sans avoir à utiliser de multiple fois l'opérateur OR.

Simplicité de l'opérateur **IN**

La syntaxe utilisée avec l'opérateur est plus simple que d'utiliser une succession d'opérateur **OR**. Pour le montrer concrètement avec un exemple, voici 2 requêtes qui retournerons les mêmes résultats, l'une utilise l'opérateur **IN**, tandis que l'autre utilise plusieurs **OR**.

Requête avec plusieurs **OR**

```
SELECT prenom
FROM utilisateur
WHERE prenom = 'Maurice' OR prenom = 'Marie' OR prenom = 'Thimoté'
```

Requête équivalente avec l'opérateur IN

```
SELECT prenom
FROM utilisateur
WHERE prenom IN ( 'Maurice', 'Marie', 'Thimoté' )
```

SQL BETWEEN

L'opérateur **BETWEEN** est utilisé dans une requête SQL pour sélectionner un intervalle de données dans une requête utilisant **WHERE**. L'intervalle peut être constitué de **chaînes de caractères**, de **nombres** ou de **dates**. L'exemple le plus concret consiste par exemple à récupérer uniquement les enregistrements entre 2 dates définies.

Filtrer entre 2 dates :

id	nom	date_inscription
1	Maurice	20120302
2	Simon	20120305
3	Chloé	20120414
4	Marie	20120415
5	Clémentine	20120426

```
SELECT *
FROM utilisateur
WHERE date_inscription BETWEEN '2012-04-01' AND '2012-04-20'
```

id	nom	date_inscription
3	Chloé	20120414
4	Marie	20120415

SQL LIKE

L'opérateur **LIKE** est utilisé dans la clause WHERE des requêtes SQL.

Ce mot-clé permet d'effectuer une recherche sur un modèle particulier. Il est par exemple possible de rechercher les enregistrements dont la valeur d'une colonne commence par telle ou telle lettre. Les modèles de recherches sont multiples.

id	nom	ville
1	Léon	Lyon
2	Odette	Nice
3	Vivien	Nantes
4	Etienne	Lille

```
SELECT *
FROM client
WHERE ville LIKE 'N%'
```

id	nom	ville
2	Odette	Nice
3	Vivien	Nantes

SQL IS NULL

Dans le langage SQL, l'opérateur **IS NULL** permet de filtrer les résultats qui contiennent la valeur **NULL**.

Cet opérateur est indispensable car la valeur **NULL** est une valeur inconnue et ne peut par conséquent pas être filtrée par les opérateurs de comparaison (cf. égal, inférieur, supérieur ou différent).

id	nom	date_inscription	fk_adresse_livraison_id	fk_adresse_facturation_id
23	Grégoire	20130212	12	12
24	Sarah	20130217	NULL	NULL
25	Anne	20130221	13	14
26	Frédérique	20130302	NULL	NULL

```
SELECT *
FROM `utilisateur`
WHERE `fk_adresse_livraison_id` IS NULL
```

id	nom	date_inscription	fk_adresse_livraison_id	fk_adresse_facturation_id
24	Sarah	20130217	NULL	NULL
26	Frédérique	20130302	NULL	NULL

SQL GROUP BY

La commande **GROUP BY** est utilisée en SQL pour grouper plusieurs résultats et utiliser une fonction de totaux sur un groupe de résultat. Sur une table qui contient toutes les ventes d'un magasin, il est par exemple possible de liste regrouper les ventes par clients identiques et d'obtenir le coût total des achats pour chaque client. On utilise ici la fonction statistique **SUM**

id	client	tarif	date
1	Pierre	102	2012-10-23
2	Simon	47	2012-10-27
3	Marie	18	2012-11-05
4	Marie	20	2012-11-14
5	Pierre	160	2012-12-03

```
SELECT client, SUM(tarif)
FROM achat  GROUP BY client
```

client	SUM(tarif)
Pierre	262
Simon	47
Marie	38

SQL Fonctions Statistiques

Utilisation d'autres fonctions de statistiques :

Il existe plusieurs fonctions qui peuvent être utilisées pour manipuler plusieurs enregistrements, il s'agit des fonctions **d'agrégations statistiques**, les principales sont les suivantes :

- **AVG()** pour calculer la moyenne d'un set de valeur. Permet de connaître le prix du panier moyen pour de chaque client
- **COUNT()** pour compter le nombre de lignes concernées. Permet de savoir combien d'achats a été effectué par chaque client
- **MAX()** pour récupérer la plus haute valeur. Pratique pour savoir l'achat le plus cher
- **MIN()** pour récupérer la plus petite valeur. Utile par exemple pour connaître la date du premier achat d'un client
- **SUM()** pour calculer la somme de plusieurs lignes. Permet par exemple de connaître le total de tous les achats d'un client

Ces petites fonctions se révèlent rapidement indispensables pour travailler sur des données.

SQL HAVING

La condition **HAVING** en SQL est presque similaire à WHERE à la seule différence que HAVING permet de filtrer en utilisant des fonctions telles que SUM(), COUNT(), AVG(), MIN() ou MAX().

id	client	tarif	date_achat
1	Pierre	102	20121023
2	Simon	47	20121027
3	Marie	18	20121105
4	Marie	20	20121114
5	Pierre	160	20121203

```
SELECT client, SUM(tarif)
FROM achat  GROUP BY client
HAVING SUM(tarif) > 40
```

client	SUM(tarif)
Pierre	262
Simon	47

SQL ORDER BY

La commande **ORDER BY** permet de trier les lignes dans un résultat d'une requête SQL. Il est possible de trier les données sur une ou plusieurs colonnes, par ordre **ascendant (ASC)** ou **descendant (DESC)**.

id	nom	prenom	date_inscription	tarif_total
1	Durand	Maurice	20120205	145
2	Dupond	Fabrice	20120207	65
3	Durand	Fabienne	20120213	90
4	Dubois	Chloé	20120216	98
5	Dubois	Simon	20120223	27

```
SELECT *
FROM utilisateur  ORDER BY nom
```

id	nom	prenom	date_inscription	tarif_total
4	Dubois	Chloé	20120216	98
5	Dubois	Simon	20120223	27
2	Dupond	Fabrice	20120207	65
1	Durand	Maurice	20120205	145
3	Durand	Fabienne	20120213	90

SQL LIMIT

La clause **LIMIT** est à utiliser dans une requête SQL pour spécifier le nombre maximum de résultats que l'ont souhaite obtenir.

Cette clause est souvent associé à un **OFFSET** (qui démarre toujours à **0**), c'est-à-dire effectuer un décalage sur le jeu de résultat. Ces 2 clauses permettent par exemple d'effectuer des système de pagination (exemple : récupérer les 10 articles de la page 4).

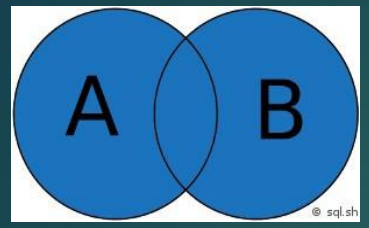
Syntaxe usuelle avec MySQL :

```
SELECT *
FROM table  LIMIT 10 OFFSET  5 ;
```

Cette requête retourne les enregistrements **6** à **15** d'une table. Le premier nombre est l'OFFSET tandis que le suivant est la limite.

Syntaxe plus courte avec MySQL (Notez l'inversion des nombres dans ce cas):

```
SELECT *
FROM table  LIMIT 5, 10;
```



SQL UNION

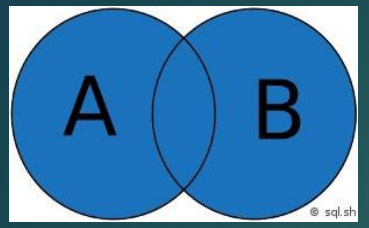
La commande **UNION** de SQL permet de mettre bout-à-bout les résultats de plusieurs requêtes utilisant elles-mêmes la commande SELECT. C'est donc une commande qui permet de concaténer les résultats de 2 requêtes ou plus. Pour l'utiliser il est nécessaire que chacune des requêtes à concaténer retournes le **même nombre de colonnes**, avec les **mêmes types de données** et dans le **même ordre**.

Liste des Clients du magasin 1 :

prenom	nom	ville	date_naissance	total_achat (€)
Léon	Dupuis	Paris	19830306	135
Marie	Bernard	Paris	19930703	75
Sophie	Dupond	Marseille	19860222	27
Marcel	Martin	Paris	19761124	39

Liste des Clients du magasin 2 :

prenom	nom	ville	date_naissance	total_achat (€)
Marion	Leroy	Lyon	19821027	285
Paul	Moreau	Lyon	19760419	133
Marie	Bernard	Paris	19930703	75
Marcel	Martin	Paris	19761124	39



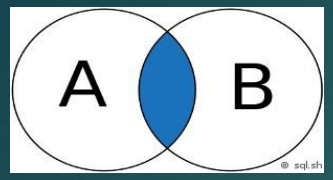
SQL UNION …

```
SELECT * FROM magasin1_client
UNION
SELECT * FROM magasin2_client
```

prenom	nom	ville	date_naissance	total_achat (€)
Léon	Dupuis	Paris	19830306	135
Marie	Bernard	Paris	19930703	75
Sophie	Dupond	Marseille	19860222	27
Marcel	Martin	Paris	19761124	39
Marion	Leroy	Lyon	19821027	285
Paul	Moreau	Lyon	19760419	133

Le résultat de cette requête montre bien que les enregistrements des 2 requêtes sont mis bout-à-bout mais **sans inclure plusieurs fois les mêmes lignes**.

Pour concaténer **tous les enregistrements** de ces tables, il est possible d'effectuer une seule requête utilisant la commande **UNION ALL**



SQL INTERSECT

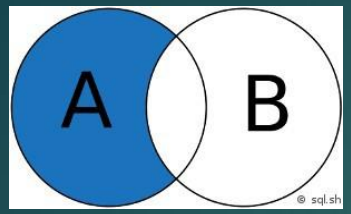
La commande SQL **INTERSECT** permet d'obtenir l'intersection des résultats de 2 requêtes. Cette commande permet donc de récupérer les enregistrements communs à 2 requêtes. Cela peut s'avérer utile lorsqu'il faut trouver s'il y a des données similaires sur 2 tables distinctes.

prenom	nom	ville	date_naissance	total_achat (€)
Léon	Dupuis	Paris	19830306	135
Marie	Bernard	Paris	19930703	75
Sophie	Dupond	Marseille	19860222	27
Marcel	Martin	Paris	19761124	39

prenom	nom	ville	date_naissance	total_achat (€)
Marion	Leroy	Lyon	19821027	285
Paul	Moreau	Lyon	19760419	133
Marie	Bernard	Paris	19930703	75
Marcel	Martin	Paris	19761124	39

```
SELECT * FROM magasin1_client
INTERSECT
SELECT * FROM magasin2_client
```

prenom	nom	ville	date_naissance	total_achat (€)
Marie	Bernard	Paris	19930703	75
Marcel	Martin	Paris	19761124	39



SQL MINUS

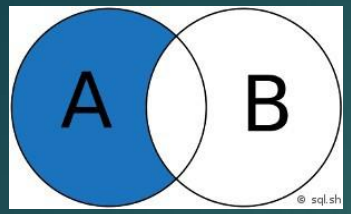
Dans le langage SQL la commande **MINUS** s'utilise entre 2 instructions pour récupérer les enregistrements de la première instruction sans inclure les résultats de la seconde requête.

Si un même enregistrement devait être présent dans les résultats des 2 syntaxes, ils ne seront pas présents dans le résultat final.

```
SELECT * FROM table1
MINUS
SELECT * FROM table2
```

Cette requête permet de lister les résultats de la table 1 sans inclure les enregistrements de la table 1 qui sont aussi dans la table 2.

Attention : les colonnes de la première requête doivent être similaires entre la première et la deuxième requête (**même nombre**, **même type** et **même ordre**).



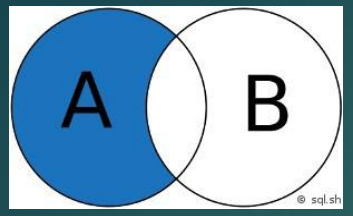
SQL MINUS …

Table **clients_inscrits** :

id	prenom	nom	date_inscription
1	Lionel	Martineau	20121114
2	Paul	Cornu	20121215
3	Sarah	Schmitt	20121217
4	Sabine	Lenoir	20121218

Table **clients_refus_email** :

id	prenom	nom	date_inscription
1	Paul	Cornu	20130127
2	Manuel	Guillot	20130127
3	Sabine	Lenoir	20130129
4	Natalie	Petitjean	20130203



SQL MINUS …

Pour pouvoir sélectionner uniquement le **prénom** et le **nom** des utilisateurs qui accepte de recevoir des emails informatifs. La requête SQL à utiliser est la suivante :

```
SELECT prenom, nom FROM clients_inscrits
MINUS
SELECT prenom, nom FROM clients_refus_email
```

prenom	nom
Lionel	**Martineau**
Sarah	**Schmitt**

Ce tableau de résultats montre bien les utilisateurs qui sont inscrits et qui ne sont pas présents dans la deuxième table. Par ailleurs, les résultats de la deuxième table ne sont pas présents sur ce résultat final.

SQL INSERT INTO

L'insertion de données dans une table s'effectue à l'aide de la commande **INSERT INTO**. Cette commande permet au choix d'inclure **une seule ligne** à la base existante ou **plusieurs lignes** d'un coup.

Insertion d'une seule ligne :

```
INSERT INTO table
(nom_colonne_1, nom_colonne_2, ... )
VALUES ('valeur 1', 'valeur 2', ...)
```

Insertion de plusieurs lignes :

```
INSERT INTO client (prenom, nom, ville, age)
VALUES
('Rébecca', 'Armand', 'Saint-Didier-des-Bois', 24),
('Aimée', 'Hebert', 'Marigny-le-Châtel', 36),
('Marielle', 'Ribeiro', 'Maillères', 27),
```

SQL UPDATE

La commande **UPDATE** permet d'effectuer des modifications sur des lignes existantes.

Très souvent cette commande est utilisée avec **WHERE** pour spécifier sur quelles lignes doivent porter la ou les modifications.

id	nom	rue	ville	code_postal	pays
1	Chantal	12 Avenue du Petit Trianon	Puteaux	92800	France
2	Pierre	18 Rue de l'Allier	Ponthion	51300	France
3	Romain	3 Chemin du Chiron	Trévérien	35190	France

```
UPDATE client
SET rue = '49 Rue Ameline',  ville = 'Saint-Eustache-la-Forêt',  code_postal = '76210'
WHERE id = 2
```

id	nom	rue	ville	code_postal	pays
1	Chantal	12 Avenue du Petit Trianon	Puteaux	92800	France
2	Pierre	49 Rue Ameline	Saint-Eustache-la-Forêt	76210	France
3	Romain	3 Chemin du Chiron	Trévérien	35190	France

SQL DELETE

La commande **DELETE** en SQL permet de supprimer des lignes dans une table. En utilisant cette commande associé à WHERE il est possible de sélectionner les lignes concernées qui seront supprimées.

Attention : Avant d'essayer de supprimer des lignes, il est recommandé d'effectuer une sauvegarde de la base de données, ou tout du moins de la table concernée par la suppression. Ainsi, s'il y a une mauvaise manipulation il est toujours possible de restaurer les données.

id	nom	prenom	date_inscription
1	Bazin	Daniel	20120213
2	Favre	Constantin	20120403
3	Clerc	Guillaume	20120412
4	Ricard	Rosemonde	20120624
5	Martin	Natalie	20120702

```
DELETE FROM utilisateur
WHERE date_inscription < '2012-04-10'
```

id	nom	prenom	date_inscription
3	Clerc	Guillaume	20120412
4	Ricard	Rosemonde	20120624
5	Martin	Natalie	20120702