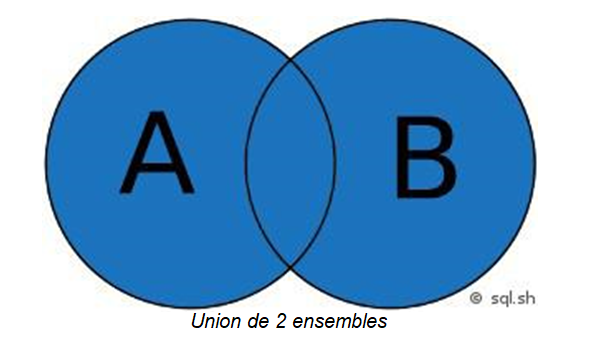
1. **SQL UNION**



**Syntaxe**

SELECT \* FROM table1

UNION

SELECT \* FROM table2

* **SQL UNION ALL**

La commande UNION ALL permet d’inclure tous les enregistrements, même les doublons.

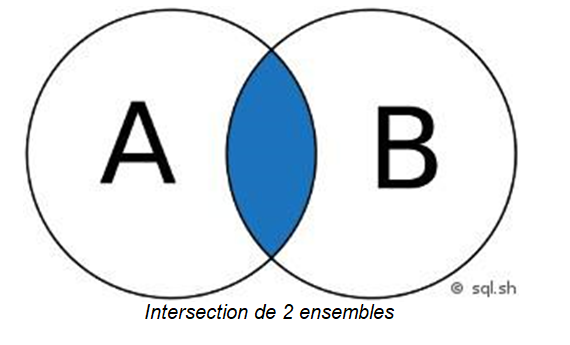
**Syntaxe**

SELECT \* FROM table1

UNION ALL

SELECT \* FROM table2

1. **SQL INTERSECT**



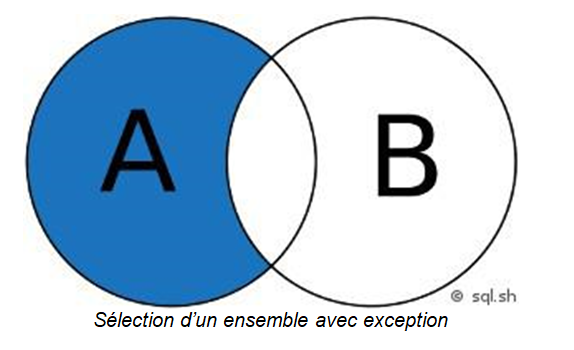
**Syntaxe**

SELECT \* FROM table1

INTERSECT

SELECT \* FROM table2

1. **SQL EXCEPT / MINUS**



**Syntaxe**

SELECT \* FROM table1

EXCEPT

SELECT \* FROM table2

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

1. **Jointure SQL**

Les jointures en SQL permettent d’associer plusieurs tables dans une même requête. Cela permet d’exploiter la puissance des bases de données relationnelles pour obtenir des résultats qui combinent les données de plusieurs tables de manière efficace.

**Types de jointures**

* **INNER JOIN :** jointure interne pour retourner les enregistrements quand la condition est vraiedans les 2 tables. C’est l’une des jointures les plus communes.

**Syntaxe**

SELECT \*

FROM table1

INNER JOIN table2 ON table1.id = table2.fk\_id

La syntaxe ci-dessus stipule qu’il faut sélectionner les enregistrements des tables table1 et table2 lorsque les données de la colonne « id » de table1 est égal aux données de la colonne fk\_id de table2.

**Exemple**

***Table utilisateur :***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Prenom | nom | email | ville |
| 1 | Aimée | Marechal | aime.marechal@example.com | Paris |
|  |  |  |  |  |
| 2 | Esmée | Lefort | esmee.lefort@example.com | Lyon |
|  |  |  |  |  |
| 3 | Marine | Prevost | m.prevost@example.com | Lille |
|  |  |  |  |  |
| 4 | Luc | Rolland | lucrolland@example.com | Marseille |
|  |  |  |  |  |

***Table commande :***

|  |  |  |  |
| --- | --- | --- | --- |
| utilisateur\_id | date\_achat | num\_facture | prix\_total |
| 1 | 2013­01­23 | A00103 | 203.14 |
|  |  |  |  |
| 1 | 2013­02­14 | A00104 | 124.00 |
|  |  |  |  |
| 2 | 2013­02­17 | A00105 | 149.45 |
|  |  |  |  |
| 2 | 2013­02­21 | A00106 | 235.35 |
|  |  |  |  |
| 5 | 2013­03­02 | A00107 | 47.58 |
|  |  |  |  |

Pour afficher toutes les commandes associées aux utilisateurs, il est possible d’utiliser la requête suivante :

SELECT id, prenom, nom, date\_achat, num\_facture, prix\_total

FROM utilisateur

INNER JOIN commande ON utilisateur.id = commande.utilisateur\_id

***Résultats***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Id | prenom | nom | date\_achat | num\_facture | prix\_total |
| 1 | Aimée | Marechal | 2013­01­23 | A00103 | 203.14 |
|  |  |  |  |  |  |
| 1 | Aimée | Marechal | 2013­02­14 | A00104 | 124.00 |
|  |  |  |  |  |  |
| 2 | Esmée | Lefort | 2013­02­17 | A00105 | 149.45 |
|  |  |  |  |  |  |
| 2 | Esmée | Lefort | 2013­02­21 | A00106 | 235.35 |
|  |  |  |  |  |  |

Le résultat de la requête montre la jointure entre les 2 tables. Les utilisateurs 3 et 4 ne sont pas affichés puisqu’il n’y a pas de commandes associés à ces utilisateurs.

* **CROSS JOIN :** jointure croisée permettant de faire le produit cartésien de 2 tables. End’autres mots, permet de joindre chaque lignes d’une table avec chaque lignes d’une seconde table. Attention, le nombre de résultats est en général très élevé.

**Syntaxe**

SELECT \*

FROM table1

CROSS JOIN table2

**OR**

SELECT \*

FROM table1, table2

La syntaxe ci-dessus permet d’associer tous les résultats de table1 avec chacun des résultats de table2.

**Exemple**

***Table legume :***

|  |  |  |
| --- | --- | --- |
| l\_id | l\_nom\_fr | l\_nom\_en\_gb |
| 45 | Carotte | Carott |
|  |  |  |
| 46 | Oignon | Onion |
|  |  |  |
| 47 | Poireau | Leek |
|  |  |  |
| ***Table fruit :*** |  |  |
|  |  |  |
| f\_id | f\_nom\_fr | f\_nom\_en\_gb |
| 87 | Banane | Banana |
|  |  |  |
| 88 | Kiwi | Kiwi |
|  |  |  |
| 89 | Poire | Pear |

SELECT l\_id, l\_nom\_fr, f\_id, f\_nom\_fr

FROM legume

CROSS JOIN fruit

**OR**

SELECT l\_id, l\_nom\_fr, f\_id, f\_nom\_fr

FROM legume, fruit

***Résultats :***

|  |  |  |  |
| --- | --- | --- | --- |
| l\_id | l\_nom\_fr | f\_id | f\_nom\_fr |
| 45 | Carotte | 87 | Banane |
|  |  |  |  |
| 45 | Carotte | 88 | Kiwi |
|  |  |  |  |
| 45 | Carotte | 89 | Poire |
|  |  |  |  |
| 46 | Oignon | 87 | Banane |
|  |  |  |  |
| 46 | Oignon | 88 | Kiwi |
|  |  |  |  |
| 46 | Oignon | 89 | Poire |
|  |  |  |  |
| 47 | Poireau | 87 | Banane |
|  |  |  |  |
| 47 | Poireau | 88 | Kiwi |
|  |  |  |  |
| 47 | Poireau | 89 | Poire |
|  |  |  |  |

Le résultat montre bien que chaque légume est associé à chaque fruit. Avec 3 fruits et 3 légumes, il y a donc 9 lignes de résultats (3 x3 = 9).

* **LEFT JOIN (ou LEFT OUTER JOIN) :** jointure externe pour retourner tous lesenregistrements de la table de gauche (LEFT = gauche) même si la condition n’est pas vérifié dans l’autre table.

**Syntaxe**

SELECT \*

FROM table1

LEFT JOIN table2 ON table1.id = table2.fk\_id

**OR**

SELECT \*

FROM table1

LEFT OUTER JOIN table2 ON table1.id = table2.fk\_id

**Exemple**

***Table utilisateur :***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | prenom | nom | email | ville |
| 1 | Aimée | Marechal | aime.marechal@example.com | Paris |
|  |  |  |  |  |
| 2 | Esmée | Lefort | esmee.lefort@example.com | Lyon |
|  |  |  |  |  |
| 3 | Marine | Prevost | m.prevost@example.com | Lille |
|  |  |  |  |  |
| 4 | Luc | Rolland | lucrolland@example.com | Marseille |
|  |  |  |  |  |

***Table commande :***

|  |  |  |  |
| --- | --- | --- | --- |
| utilisateur\_id | date\_achat | num\_facture | prix\_total |
| 1 | 2013­01­23 | A00103 | 203.14 |
|  |  |  |  |
| 1 | 2013­02­14 | A00104 | 124.00 |
|  |  |  |  |
| 2 | 2013­02­17 | A00105 | 149.45 |
|  |  |  |  |
| 2 | 2013­02­21 | A00106 | 235.35 |
|  |  |  |  |
| 5 | 2013­03­02 | A00107 | 47.58 |
|  |  |  |  |

SELECT \*

FROM utilisateur

LEFT JOIN commande ON utilisateur.id = commande.utilisateur\_id

Pour lister tous les utilisateurs avec leurs commandes et afficher également les utilisateurs qui n’ont pas effectuées d’achats.

***Résultats :***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| id | prenom | nom | date\_achat | num\_facture | prix\_total |
| 1 | Aimée | Marechal | 2013­01­23 | A00103 | 203.14 |
|  |  |  |  |  |  |
| 1 | Aimée | Marechal | 2013­02­14 | A00104 | 124.00 |
|  |  |  |  |  |  |
| 2 | Esmée | Lefort | 2013­02­17 | A00105 | 149.45 |
|  |  |  |  |  |  |
| 2 | Esmée | Lefort | 2013­02­21 | A00106 | 235.35 |
|  |  |  |  |  |  |
| 3 | Marine | Prevost | NULL | NULL | NULL |
|  |  |  |  |  |  |
| 4 | Luc | Rolland | NULL | NULL | NULL |
|  |  |  |  |  |  |

La ligne retourne la valeur NULL pour les colonnes concernant les utilisateurs qui n’ont pas effectués des achats.

* **RIGHT JOIN (ou RIGHT OUTER JOIN) :** jointure externe pour retourner tous lesenregistrements de la table de droite (RIGHT = droite) même si la condition n’est pas vérifié dans l’autre table.
* **FULL JOIN (ou FULL OUTER JOIN) :** jointure externe pour retourner les résultats quand lacondition est vrai dans au moins une des 2 tables.
* **SELF JOIN :** permet d’effectuer une jointure d’une table avec elle-même comme si c’était uneautre table.
* **NATURAL JOIN :** jointure naturelle entre 2 tables s’il y a au moins une colonne qui porte lemême nom entre les 2 tables SQL
* **UNION JOIN :** jointure d’union

1. **SQL Sous-requête**

Dans le langage SQL une sous-requête (aussi appelé « requête imbriquée » ou « requête en cascade ») consiste à exécuter une requête à l’intérieur d’une autre requête. Une requête imbriquée est souvent utilisée au sein d’une clause WHERE ou de HAVING pou remplacer une ou plusieurs constante.

**Syntaxe**

Il y a plusieurs façons d’utiliser les sous-requêtes. De cette façon il y a plusieurs syntaxes envisageables pour utiliser des requêtes dans des requêtes.

**Exemples**

**Requête imbriquée qui retourne un seul résultat :**

une requête interne (celle sur « table2″) qui renvoi une seule valeur. La requête externe quant à elle, va chercher les résultat de « table » et filtre les résultats à partir de la valeur retournée par la requête interne.

SELECT \*

FROM `table`

WHERE `nom\_colonne` = (

SELECT `valeur`

FROM `table2`

LIMIT 1

)

***A noter : il est possible d’utiliser n’importe quel opérateur d’égalité tel que =, >, <, >=, <= ou <>.***

**Requête imbriquée qui retourne une colonne :**

Une requête imbriquée peut également retournée une colonne entière. Dès lors, la requête externe peut utiliser la commande IN pour filtrer les lignes qui possèdent une des valeurs retournées par la requête interne.

SELECT \*

FROM `table`

WHERE `nom\_colonne` IN (

SELECT `colonne`

FROM `table2`

WHERE `cle\_etrangere` = 36

)

1. **INDEX SQL**

Avec un index placé sur une ou plusieurs colonnes le système d’une base de données peut rechercher les données d’abord sur l’index et s’il trouve ce qu’il cherche il saura plus rapidement où se trouve les enregistrements concernés.

**SQL CREATE INDEX**

En SQL, la commande CREATE INDEX permet de créer un index. L’index est utile pour accélérer l’exécution d’une requête SQL qui lit des données et ainsi améliorer les performances d’une application utilisant une base de données.

**Syntaxe**

**Créer un index ordinaire**

La syntaxe basique pour créer un index est la suivante :



CREATE INDEX `index\_nom` ON `table`;

Il est également possible de créer un index sur une seule colonne en précisant la colonne sur laquelle doit s’appliquer l’index :



CREATE INDEX `index\_nom` ON `table` (`colonne1`);

L’exemple ci-dessus va donc insérer l’index intitulé « index\_nom » sur la table nommée « table » uniquement sur la colonne « colonne1″. Pour insérer un index sur plusieurs colonnes il est possible d’utiliser la syntaxe suivante :



CREATE INDEX `index\_nom` ON `table` (`colonne1`, `colonne2`);

L’exemple ci-dessus permet d’insérer un index les 2 colonnes : colonne1 et colonne2.

**Créer un index unique**

Un index unique permet de spécifier qu’une ou plusieurs colonnes doivent contenir des valeurs uniques à chaque enregistrement. Le système de base de données retournera une erreur si une requête tente d’insérer des données qui feront doublons sur la clé d’unicité. Pour insérer un tel index il suffit d’exécuter une requête SQL respectant la syntaxe suivante :



CREATE UNIQUE INDEX `index\_nom` ON `table` (`colonne1`);

Dans cet exemple un index unique sera créé sur la colonne nommée colonne1. Cela signifie qu’il ne peut pas y avoir plusieurs fois la même valeur sur 2 enregistrements distincts contenus dans cette table.

Il est également possible de créer un index d’unicité sur 2 colonnes, en respectant la syntaxe suivante :



CREATE UNIQUE INDEX `index\_nom` ON `table` (`colonne1`, `colonne2`);