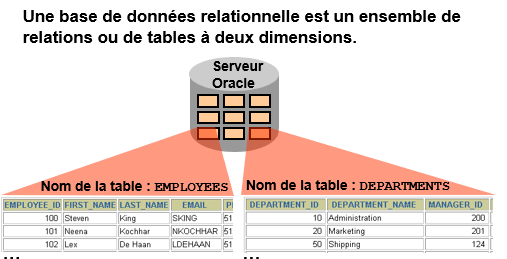
# Base de donnée

Pour gérer des bases de données, vous devez disposer d'un système de gestion de base de données (SGBD). Un SGBD est un programme permettant le stockage, l'extraction et la modification de données à la demande. Il existe quatre types principaux de base de données : *hiérarchique*, *réseau*, *relationnel* et (plus récemment) *relationnel objet*.

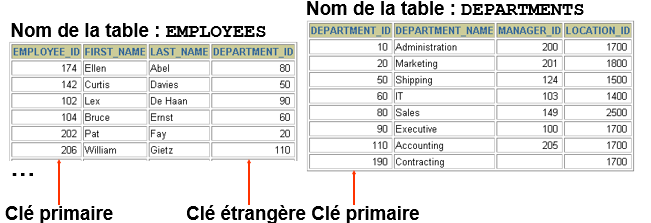
**Définition d'une base de données relationnelle**

Une base de données relationnelle utilise des relations ou des tables à deux dimensions pour le stockage des informations.

Par exemple, vous pouvez stocker des informations concernant tous les employés de votre entreprise. Dans une base de données relationnelle, vous pouvez créer plusieurs tables afin de stocker différentes informations sur les employés, telles qu'une table employees, une table department et une table salary.

**Mettre en relation plusieurs tables**

Chaque table contient des données qui décrivent exactement une entité. Par exemple, la table EMPLOYEES contient des informations sur les employés. Les catégories de données sont répertoriées en haut de chaque table et les éléments individuels sont répertoriés en dessous. Grâce à un format de tableau, vous pouvez visualiser, comprendre et utiliser les informations.

Etant donné que les données sur différentes entités sont stockées dans des tables différentes, vous pouvez être amené à combiner plusieurs tables afin de répondre à une question particulière. Par exemple, vous pouvez souhaiter connaître le lieu du département où un employé travaille. Dans ce scénario, vous avez besoin d'informations provenant de la table EMPLOYEES (qui contient des données sur les employés) et de la table DEPARTMENTS (qui contient des informations sur les départements). Avec un SGBDR, vous pouvez lier les données d'une table aux données d'une autre par l'intermédiaire des clés étrangères. Une clé étrangère est une colonne (ou un ensemble de colonnes) qui fait référence à une clé primaire de la même table ou d'une autre table.

Vous pouvez utiliser la possibilité de lier les données d'une table aux données d'une autre afin d'organiser les informations en unités distinctes pouvant être gérées. Les données des employés peuvent être conservées dans un emplacement logiquement distinct des données des départements, via leur stockage dans une table distincte.

* Les clés primaires ne peuvent généralement pas être modifiées.
* Les clés étrangères sont basées sur les valeurs des données et sont purement des pointeurs logiques (non physiques).
* La valeur d'une clé étrangère doit correspondre à une valeur de clé primaire ou de clé unique existante ; à défaut, elle doit être NULL.
* Une clé étrangère doit référencer une colonne de clé primaire ou unique.
* Une colonne contenant le numéro de département, qui est également une *clé étrangère*. Une clé étrangère est une colonne qui définit la relation entre les tables. Une clé étrangère fait référence à une clé primaire ou une clé unique dans la même table ou dans une autre table. Dans l'exemple, DEPARTMENT\_ID identifie *de manière unique* un département dans la table DEPARTMENTS.

Fonctionnalités des instructions SQL select

Projection : choisissez les colonnes d'une table qui sont renvoyées par une interrogation. Choisissez le nombre de colonnes nécessaires (peu ou beaucoup).

Sélection : choisissez les lignes d'une table qui sont renvoyées par une interrogation. Divers critères peuvent être utilisés pour limiter les lignes extraites.

Jointure : combinez des données stockées dans différentes tables en indiquant le lien entre elles. Les jointures SQL seront étudiées plus en détail dans un prochain chapitre.

Select = indique les colonnes çà afficher  
from = identifie la table contenant les colonnes répertoriées dans la clause select

Select \*  
from emloyees  
Affiche tous les colonnes de la table employees

**Valeurs NULL**

Si une valeur est absente d'une ligne pour une colonne particulière, cette valeur est dite *NULL*, ou contenant une valeur NULL.

Une valeur NULL est une valeur qui n'est pas disponible, pas affectée, inconnue ou inapplicable. Une valeur NULL est différente d'un zéro ou d'un espace. Zéro est un chiffre et un espace est un caractère.

Toutes les colonnes, quel que soit leur type de données, peuvent contenir des valeurs NULL. Cependant, certaines contraintes (NOT NULL et PRIMARY KEY) empêchent l'utilisation de valeurs NULL dans la colonne.

**Alias de colonne**

Lors de l'affichage du résultat d'une interrogation, *i*SQL\*Plus utilise généralement le nom de la colonne sélectionnée comme en-tête de la colonne. Cet en-tête n'est pas toujours évocateur et peut donc être difficile à comprendre. Vous pouvez modifier l'en-tête d'une colonne en utilisant un alias de colonne.

Indiquez l'alias après la colonne dans la liste SELECT en utilisant un espace comme séparateur. Par défaut, les en-têtes d'alias s'affichent en majuscules. Si l'alias contient des espaces ou des caractères spéciaux (tels que # ou $), ou s'il distingue les majuscules des minuscules, incluez l'alias entre guillemets (" ").

**Opérateur de concaténation**

Vous pouvez lier des colonnes à d'autres colonnes, expressions arithmétiques ou valeurs constantes afin de créer une expression de type caractère à l'aide de l'*opérateur de concaténation* (||). Les colonnes de chaque côté de l'opérateur sont combinées afin de créer une colonne de sortie unique.

**Chaînes de caractères littérales (suite)**

L'exemple de la diapositive ci-dessus affiche le nom et le code de poste de tous les employés. La colonne présente l'en-tête Employee Details. Notez les espaces entre les apostrophes dans l'instruction SELECT. Les espaces améliorent la lisibilité de la sortie.

Dans l'exemple qui suit, le nom et le salaire de chaque employé sont concaténés avec une valeur littérale afin de rendre plus significatives les lignes renvoyées :

SELECT last\_name ||': 1 Month salary = '||salary Monthly

FROM employees;

Affiche King is a AD\_PRES

**Lignes en double**

|  |  |
| --- | --- |
| **Opérateur** | **Signification** |
| **=** | **Egal à** |
| **>** | **Supérieur à** |
| **>=** | **Supérieur ou égal à** |
| **<** | **Inférieur à** |
| **<=** | **Inférieur ou égal à** |
| **<>** | **Non égal à** |
| **BETWEEN ...AND...** | **Entre deux valeurs (incluses)** |
| **IN(set)** | **Correspond à une valeur quelconque d'une liste** |
| **LIKE** | **Correspond à un modèle de caractère** |
| **IS NULL** | **Est une valeur NULL** |

Sauf indication contraire, *i*SQL\*Plus affiche les résultats d'une interrogation sans supprimer les lignes en double. Le premier exemple de la diapositive ci-dessus affiche tous les numéros de département de la table EMPLOYEES. Notez que les numéros de département sont répétés.

Pour éliminer les lignes en double dans le résultat, incluez le mot-clé DISTINCT dans la clause SELECT, immédiatement après le mot-clé SELECT. Dans le deuxième exemple de la diapositive, la table EMPLOYEES contient en fait 20 lignes, mais elle contient seulement sept numéros de département uniques.

Vous pouvez indiquer plusieurs colonnes après le qualificatif DISTINCT. Le qualificatif DISTINCT affecte toutes les colonnes sélectionnées et le résultat est constitué de chaque combinaison distincte des colonnes.

SELECT DISTINCT department\_id, job\_id

FROM employees;

**Limiter les lignes sélectionnées**

Vous pouvez restreindre les lignes renvoyées par l'interrogation en utilisant la clause WHERE. Une clause WHERE contient une condition qui doit être satisfaite ; elle suit directement la clause FROM. Si la condition est vraie, la ligne qui satisfait à cette condition est renvoyée.

**Utiliser la condition IN**

Pour tester des valeurs d'un ensemble particulier, utilisez la condition IN. La condition IN est également appelée condition d'appartenance.

L'exemple de la diapositive ci-dessus affiche le numéro, le nom, le salaire et le numéro de manager de tous les employés dont le numéro du manager est 100, 101 ou 201.

**SELECT employee\_id, last\_name, salary, manager\_id**

**FROM employees**

**WHERE manager\_id IN (100, 101, 201) ;**

**Utiliser la condition LIKE**

**SELECT first\_name**

**FROM employees**

**WHERE first\_name LIKE 'S%' ;**

Vous ne connaissez pas toujours la valeur exacte à rechercher. Vous pouvez sélectionner des lignes correspondant à un modèle de caractère à l'aide de la condition LIKE. L'opération de mise en correspondance de modèles de caractère est appelée recherche par *caractères génériques*. Deux symboles peuvent être utilisés pour construire la chaîne de recherche.

**Combiner des caractères génériques**

Les symboles % et \_ peuvent être utilisés dans n'importe quelle combinaison avec des caractères littéraux. L'exemple de la diapositive ci-dessus affiche le nom de tous les employés dont le deuxième caractère du nom est la lettre *o*.

**Utiliser la condition NULL**

Les conditions NULL incluent la condition IS NULL et la condition IS NOT NULL.

**Conditions logiques**

Une condition logique combine le résultat de deux conditions afin de générer un résultat unique basé sur ces conditions, ou inverse le résultat d'une condition unique. Une ligne est renvoyée uniquement si le résultat global de la condition est vrai.

Trois opérateurs logiques sont disponibles en langage SQL : not, or, and

Jusqu'à présent, la clause WHERE de tous les exemples présentés ne comportait qu'une seule condition. Vous pouvez utiliser plusieurs conditions dans une même clause WHERE à l'aide des opérateurs AND et OR.

**Utiliser la clause ORDER BY**

L'ordre des lignes renvoyées dans le résultat d'une interrogation n'est pas défini. La clause ORDER BY peut être utilisée pour trier les lignes. Si vous utilisez la clause ORDER BY, il doit s'agir de la dernière clause de l'instruction SQL. Vous pouvez indiquer une expression, un alias ou la position d'une colonne comme condition de tri. ASC DESC

**Fonctions SQL**

Les fonctions sont une fonctionnalité très puissante du langage SQL. Elles peuvent être utilisées pour effectuer les opérations suivantes :

* + - Effectuer des calculs sur les données
    - Modifier des données individuelles
    - Manipuler la sortie de groupes de lignes
    - Mettre en forme des dates et des nombres en vue de leur affichage
    - Convertir des types de données de colonne

Les fonctions SQL acceptent parfois des arguments et renvoient toujours une valeur.

**Remarque :** La plupart des fonctions décrites dans ce chapitre sont propres à la version Oracle du langage SQL.

**Fonctions SQL (suite)**

Il existe deux types de fonction :

* + - Fonctions monoligne

|  |  |
| --- | --- |
| **Fonction** | **Résultat** |
| **LOWER('SQL Course')** | **sql course** |
| **UPPER('SQL Course')** | **SQL COURSE** |
| **INITCAP('SQL Course')** | **Sql Course** |

* + - Fonctions multiligne

**Fonctions monoligne**

Ces fonctions opèrent sur des lignes uniques et renvoient un seul résultat par ligne. Il existe différents types de fonction monoligne

**Fonctions multiligne**

Les fonctions peuvent manipuler des groupes de lignes afin de renvoyer un seul résultat par groupe de lignes. Ces fonctions sont également appelées *fonctions de groupe* (étudiées dans un prochain chapitre). **LOWER :** convertit des chaînes de caractères (casse mixte ou majuscules)   
en minuscules.

**UPPER :** convertit des chaînes de caractères (casse mixte ou minuscules)   
en majuscules.

**SELECT employee\_id, last\_name, department\_id**

**FROM employees**

**WHERE LOWER(last\_name) = 'higgins';**

**INITCAP :** convertit la première lettre de chaque mot en majuscules et les autres lettres en minuscules

**Fonctions de manipulation de caractères**

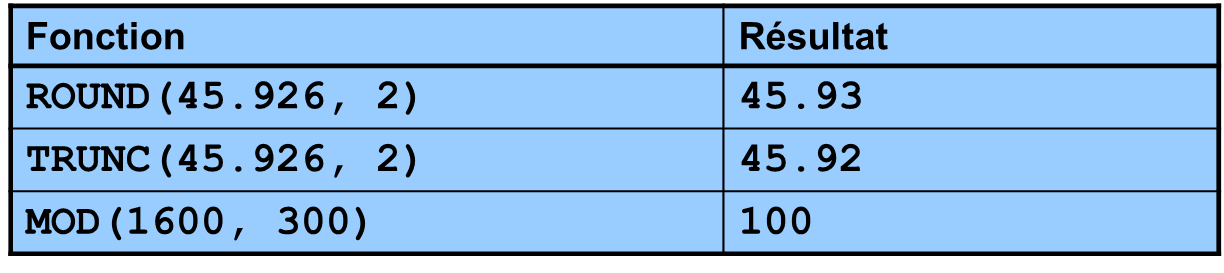
Les fonctions de manipulation de caractères étudiées dans ce chapitre sont CONCAT, SUBSTR, LENGTH, INSTR, LPAD, RPAD et TRIM.

* + - **CONCAT :** joint des valeurs (la fonction CONCAT ne permet d'utiliser que   
      deux arguments).
    - **SUBSTR :** extrait une chaîne d'une longueur déterminée.
    - **LENGTH :** renvoie la longueur d'une chaîne sous forme de valeur numérique.
    - **INSTR :** détermine la position numérique d'un caractère nommé.
    - **LPAD :** complète la valeur avec des caractères afin de la justifier à droite.
    - **RPAD :** complète la valeur avec des caractères afin de la justifier à gauche.
    - **TRIM :** supprime des caractères de début ou de fin (ou les deux) d'une chaîne de caractères (si *trim\_character* ou *trim\_source* est un littéral de type caractère, vous devez l'inclure entre apostrophes).

**Remarque :** Vous pouvez utiliser des fonctions telles que UPPER et LOWER avec l'esperluette d'interprétation. Par exemple, utilisez UPPER('&job\_title') pour que l'utilisateur n'ait pas à saisir le poste dans une casse particulière

**Exemple**

Modifiez l'instruction SQL de la diapositive afin d'afficher les mêmes informations pour les employés dont le nom se termine par la lettre *n*.

SELECT employee\_id, CONCAT(first\_name, last\_name) NAME,  
LENGTH (last\_name), INSTR(last\_name, 'a') "Contains 'a'?"  
FROM employees   
WHERE SUBSTR(last\_name, -1, 1) = 'n';

ROUND : arrondit la valeur à une décimale donnée  
TRUNC : tronque la valeur à une décimale donnée  
MOD : renvoie le reste d'une division

Les calculs se font from dual

**Format de date Oracle**

**SELECT last\_name, hire\_date**

**FROM employees**

**WHERE hire\_date < '01-FEB-88';**

Dans la base de données Oracle, les dates sont stockées dans un format numérique interne, représentant le siècle, l'année, le mois, le jour, les heures, les minutes et les secondes.

Le format de date par defaut est DD-MON-RR

SYSDATE est une fonction qui renvoie : La date et L'heure

MONTHS\_BETWEEN(date1, date2) : calcule le nombre de mois entre date1 et date2. Le résultat peut être positif ou négatif. Si date1 est postérieure à date2, le résultat est positif ; si date1 est antérieure à date2, le résultat est négatif. La partie non entière du résultat représente une partie du mois.

ADD\_MONTHS(date, n) : ajoute n mois calendaires à date. La valeur de n doit être un entier et peut être négative.

NEXT\_DAY(date, 'char') : détermine la date du jour suivant de la semaine ('char') après date. La valeur de char peut être un nombre représentant un jour, ou une chaîne de caractères.

LAST\_DAY(date) : détermine la date du dernier jour du mois contenant date.

ROUND(date[,'fmt']) : renvoie date arrondie à l'unité désignée par le modèle de format fmt. Si le modèle de format fmt est omis, date est arrondie au jour le   
plus proche.

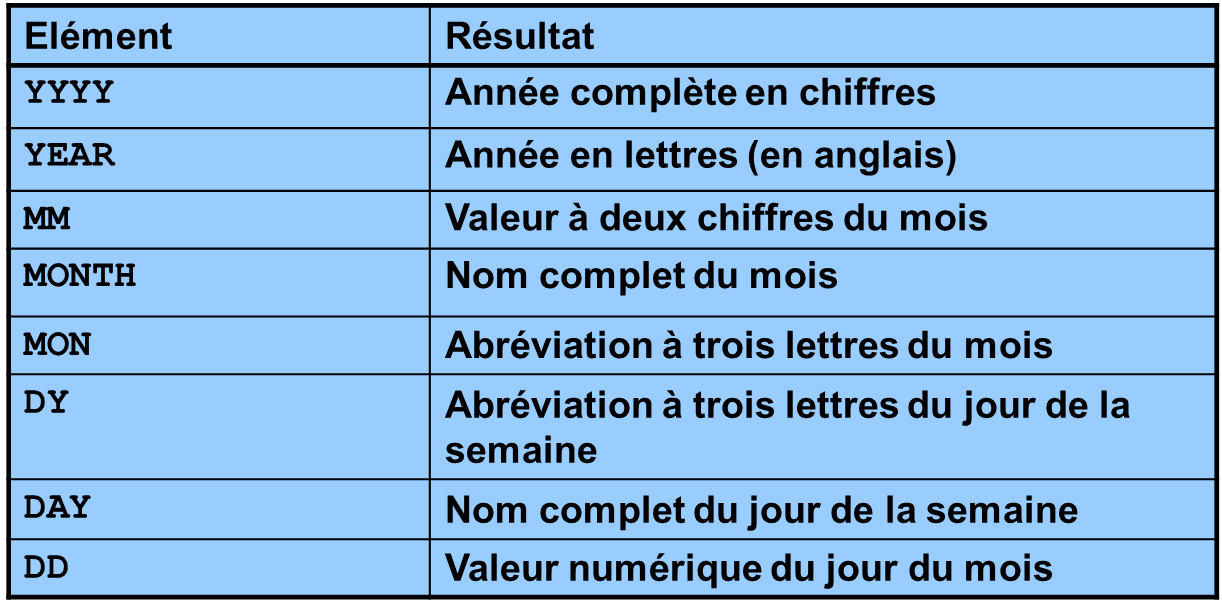
TRUNC(date[, 'fmt']) : renvoie date avec la partie heure du jour tronquée à l'unité désignée par le modèle de format fmt. Si le modèle de format fmt est omis, date est tronquée au jour le plus proche.

**Afficher une date dans un format spécifique**

Auparavant, toutes les valeurs de date Oracle étaient affichées au format DD-MON-YY. Vous pouvez utiliser la fonction TO\_CHAR pour convertir une date de ce format par défaut vers le format souhaité.

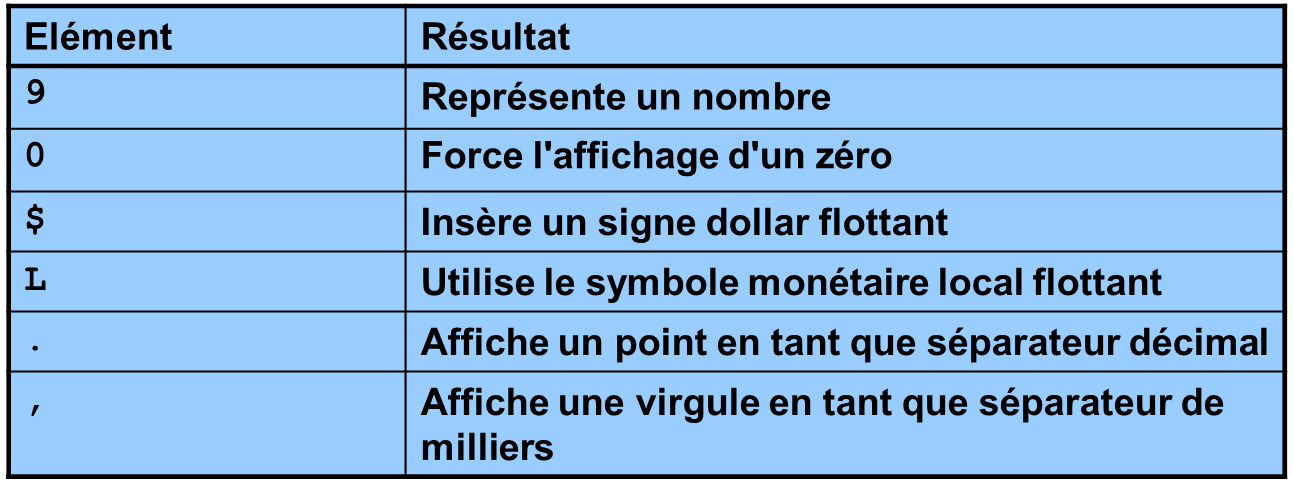
**Règles**

* + - Le modèle de format doit être inclus entre apostrophes et distingue les majuscules   
      des minuscules.
    - Il peut inclure n'importe quel élément de format de date valide. Prenez soin de séparer la valeur de date du modèle de format par une virgule.
    - Dans le résultat, les noms des jours et des mois sont complétés automatiquement par   
      des espaces.
    - Pour supprimer les espaces de remplissage ou les zéros de début, utilisez l'élément *fm* de mode de remplissage.
    - Vous pouvez formater le champ de caractère résultant à l'aide de la commande *i*SQL\*Plus COLUMN (étudiée dans un prochain chapitre).

SELECT employee\_id, TO\_CHAR(hire\_date, 'MM/YY') Month\_Hired

FROM employees

WHERE last\_name = 'higgins';

affiche 17 june 1987

**SELECT last\_name,**

**TO\_CHAR(hire\_date, 'fmDD Month YYYY')**

**AS HIREDATE**

**FROM employees;**

**TO\_CHAR(*number,* '*format\_model*') ddspth**

**Fonctions générales**

Ces fonctions fonctionnent avec n'importe quel type de données et acceptent les valeurs NULL dans la liste des expressions.

**Fonction NVL**

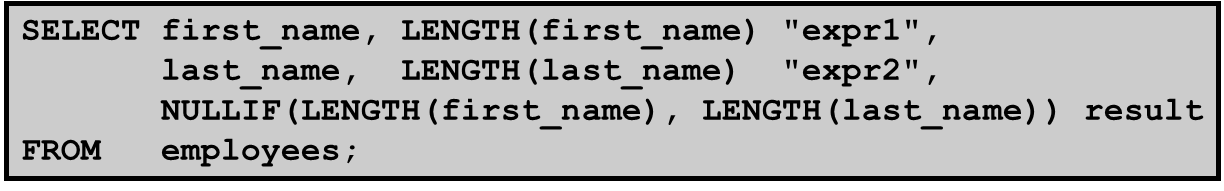
Pour convertir une valeur NULL en valeur réelle, utilisez la fonction NVL.

**Syntaxe**

NVL (*expr1*, *expr2*)

Dans la syntaxe :

* + - *expr1* est la valeur ou l'expression source pouvant contenir une valeur NULL.
    - *expr2* est la valeur cible pour la conversion de la valeur NULL.



**Fonctions de groupe**

Contrairement aux fonctions monoligne, les fonctions de groupe opèrent sur des ensembles de lignes afin de renvoyer un seul résultat par groupe. Ces ensembles peuvent être constitués de la table entière ou de la table répartie en groupes.

Ex AVG, COUNT, MAX, MIN, STDDEV, SUM, VARIANCE

**SELECT AVG(salary), MAX(salary),**

**MIN(salary), SUM(salary)**

**FROM employees**

**WHERE job\_id LIKE '%REP%';**

Vous pouvez utiliser les fonctions MAX et MIN pour les valeurs numériques, les valeurs de type caractère et les valeurs de type date. L'exemple de la diapositive ci-dessus affiche les employés dont l'ancienneté est la plus faible et ceux dont l'ancienneté est la plus élevée.

L'exemple suivant affiche le nom de l'employé en première position et celui de l'employé en dernière position dans une liste d'employés classés par ordre alphabétique :

SELECT MIN(last\_name), MAX(last\_name)

FROM employees;

La fonction COUNT présente trois formats :

* + - COUNT(\*)
    - COUNT(*expr*)
    - COUNT(DISTINCT *expr*)

La fonction COUNT(\*) renvoie le nombre de lignes d'une table qui satisfont aux critères de l'instruction SELECT, y compris les lignes en double et celles dont certaines des colonnes contiennent des valeurs NULL. Si une clause WHERE est incluse dans l'instruction SELECT, la fonction COUNT(\*) renvoie le nombre de lignes qui satisfont à la condition de la   
clause WHERE.

A l'inverse, la fonction COUNT(*expr*) renvoie le nombre de valeurs non NULL présentes dans la colonne identifiée par *expr*.

La fonction COUNT(DISTINCT *expr*) renvoie le nombre de valeurs non NULL uniques présentes dans la colonne identifiée par *expr*.

**Mot-clé DISTINCT**

Utilisez le mot-clé DISTINCT pour éviter le comptage des valeurs en double d'une colonne.

L'exemple de la diapositive ci-dessus affiche le nombre de départements distincts de la   
table EMPLOYEES.

**SELECT COUNT(DISTINCT department\_id)**

**FROM employees;**

**Fonctions de groupe et valeurs NULL**

Toutes les fonctions de groupe ignorent les valeurs NULL de la colonne.

La fonction NVL force les fonctions de groupe à inclure les valeurs NULL.

**Clause GROUP BY**

Vous pouvez utiliser la clause GROUP BY afin de diviser les lignes d'une table en groupes. Vous pouvez ensuite utiliser les fonctions de groupe afin de renvoyer des informations récapitulatives pour chaque groupe.

Dans la syntaxe :

*group\_by\_expression* indique les colonnes dont les valeurs déterminent la base de regroupement des lignes

**Règles**

* + - Si vous incluez une fonction de groupe dans une clause SELECT, vous ne pouvez pas sélectionner également des résultats individuels, *sauf* si la colonne individuelle apparaît dans la clause GROUP BY. Vous recevez un message d'erreur si vous ne parvenez pas à inclure la liste des colonnes dans la clause GROUP BY.
    - Vous pouvez utiliser une clause WHERE pour exclure des lignes avant de les répartir   
      en groupes.
    - Vous devez inclure les *colonnes* dans la clause GROUP BY.

Vous ne pouvez pas utiliser d'alias de colonne dans la clause GROUP BY.

**Utiliser la clause GROUP BY**

Lorsque vous utilisez la clause GROUP BY, assurez-vous que toutes les colonnes de la liste SELECT qui ne sont pas incluses dans des fonctions de groupe sont incluses dans la clause GROUP BY. L'exemple de la diapositive ci-dessus affiche le numéro et le salaire moyen de chaque département. Voici comment cette instruction SELECT, contenant une clause GROUP BY, est évaluée :

* + - La clause SELECT indique les colonnes à extraire, de la façon suivante :
      * Numéro de département dans la table EMPLOYEES
      * Moyenne de tous les salaires du groupe indiqué dans la clause GROUP BY
    - La clause FROM indique les tables auxquelles la base de données doit accéder, à savoir la table EMPLOYEES dans le cas présent.
    - La clause WHERE désigne les lignes à extraire. Etant donné qu'il n'y a pas de clause WHERE, toutes les lignes sont extraites par défaut.

**SELECT department\_id, AVG(salary)**

**FROM employees**

**GROUP BY department\_id ;**

La clause GROUP BY indique la façon dont les lignes doivent être regroupées. Les lignes sont regroupées par numéro de département, de sorte que la fonction AVG appliquée à la colonne de salaire calcule le salaire moyen de chaque département.

**SELECT department\_id dept\_id, job\_id, SUM(salary)**

**FROM employees**

**GROUP BY department\_id, job\_id ;**

Vous pouvez renvoyer des résultats récapitulatifs pour des groupes et des sous-groupes en répertoriant plusieurs colonnes GROUP BY. Vous pouvez déterminer l'ordre de tri par défaut des résultats en fonction de l'ordre des colonnes dans la clause GROUP BY. Dans l'exemple de la diapositive, l'instruction SELECT contenant une clause GROUP BY est évaluée de la façon suivante :

* + - La clause SELECT désigne la colonne à extraire :
      * Numéro de département dans la table EMPLOYEES
      * ID de poste dans la table EMPLOYEES
      * Somme de tous les salaires du groupe indiqué dans la clause GROUP BY
    - La clause FROM indique les tables auxquelles la base de données doit accéder, à savoir la table EMPLOYEES dans le cas présent.
    - La clause GROUP BY indique comment regrouper les lignes :
      * Les lignes sont d'abord regroupées par numéro de département.
      * Ensuite, elles sont regroupées par ID de poste dans les groupes de numéro   
        de département.

La fonction SUM est donc appliquée à la colonne de salaire pour tous les ID de poste de chaque groupe de numéros de département.

**Interrogations illégales avec des fonctions de groupe (suite)**

La clause WHERE ne peut pas être utilisée pour restreindre des groupes. L'instruction SELECT de la diapositive ci-dessus génère une erreur, car elle utilise la clause WHERE pour limiter l'affichage des salaires moyens des départements dont le salaire moyen est supérieur   
à 8 000 $.

Vous pouvez corriger l'erreur de l'exemple en utilisant la clause HAVING pour restreindre les  
groupes :

SELECT department\_id, AVG(salary)

FROM employees

HAVING AVG(salary) > 8000

GROUP BY department\_id;

**Restreindre les résultats des groupes**

De la même façon que vous utilisez la clause WHERE pour restreindre les lignes sélectionnées, vous pouvez utiliser la clause HAVING pour restreindre des groupes. Pour déterminer le salaire maximum de chaque département dont le salaire maximum est supérieur à 10 000 $, vous devez procéder de la façon suivante :

**SELECT department\_id, MAX(salary)**

**FROM employees**

**GROUP BY department\_id**

**HAVING MAX(salary)>10000 ;**

1. Déterminez le salaire moyen de chaque département en regroupant les salaires par numéro de département.

2. Limitez les groupes aux départements dont le salaire maximum est supérieur à 10 000 $.

count(\*) renvoie le nombre de lignes d’une table

**SELECT COUNT(\*)**

**FROM employees**

**WHERE department\_id = 50;**

**Mot-clé DISTINCT**

Utilisez le mot-clé DISTINCT pour éviter le comptage des valeurs en double d'une colonne.

L'exemple de la diapositive ci-dessus affiche le nombre de départements distincts de la   
table EMPLOYEES.

**SELECT AVG(commission\_pct)**

**FROM employees;**

**Fonctions de groupe et valeurs NULL**

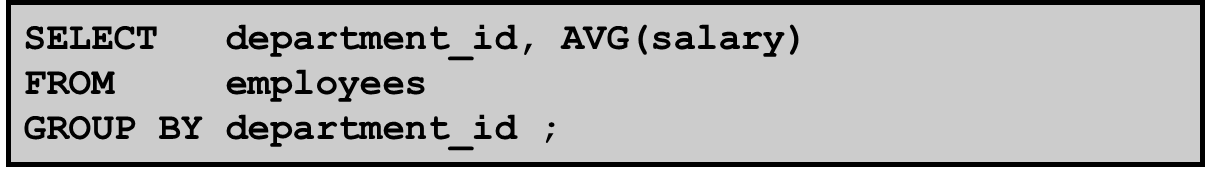
Toutes les fonctions de groupe ignorent les valeurs NULL de la colonne. La fonction NVL force les fonctions de groupe à inclure les valeurs NULL.

**SELECT AVG(NVL(commission\_pct, 0))**

**FROM employees;**

**Créer des groupes de données**

Jusqu'à présent, toutes les fonctions de groupe ont traité la table comme un groupe unique d'informations.

Il arrive cependant que vous soyez amené à diviser la table en groupes d'informations plus petits. Cette opération peut être effectuée à l'aide de la clause GROUP BY.

**Utiliser la clause GROUP BY (suite)**

La colonne GROUP BY ne doit pas nécessairement figurer dans la clause SELECT. Par exemple, l'instruction SELECT de la diapositive ci-dessus affiche le salaire moyen de chaque département, sans afficher les numéros des départements. Le résultat n'est cependant pas très significatif sans les numéros des départements.

Vous pouvez utiliser la fonction de groupe dans la clause ORDER BY :

SELECT department\_id, AVG(salary)

FROM employees

GROUP BY department\_id

ORDER BY AVG(salary);

**Groupes dans des groupes (suite)**

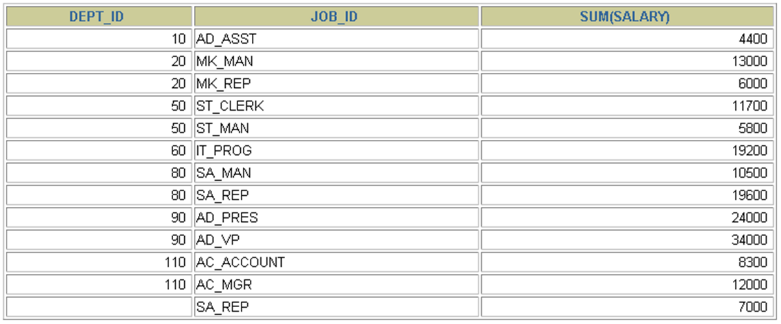
Vous pouvez renvoyer des résultats récapitulatifs pour des groupes et des sous-groupes en répertoriant plusieurs colonnes GROUP BY. Vous pouvez déterminer l'ordre de tri par défaut des résultats en fonction de l'ordre des colonnes dans la clause GROUP BY. Dans l'exemple de la diapositive, l'instruction SELECT contenant une clause GROUP BY est évaluée de la façon suivante :

La clause SELECT désigne la colonne à extraire :

Numéro de département dans la table EMPLOYEES

ID de poste dans la table EMPLOYEES

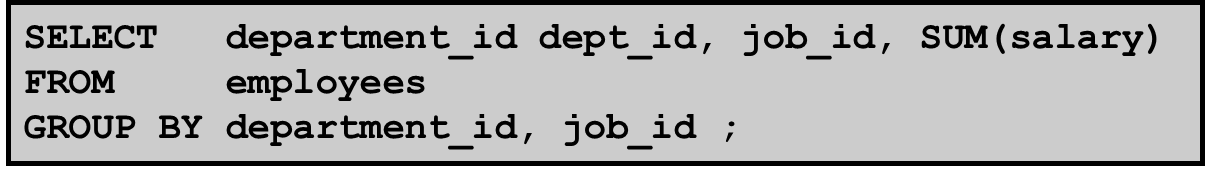
Somme de tous les salaires du groupe indiqué dans la clause GROUP BY

La clause FROM indique les tables auxquelles la base de données doit accéder, à savoir la table EMPLOYEES dans le cas présent.

la clause GROUP BY indique comment regrouper les lignes :

Les lignes sont d'abord regroupées par numéro de département.

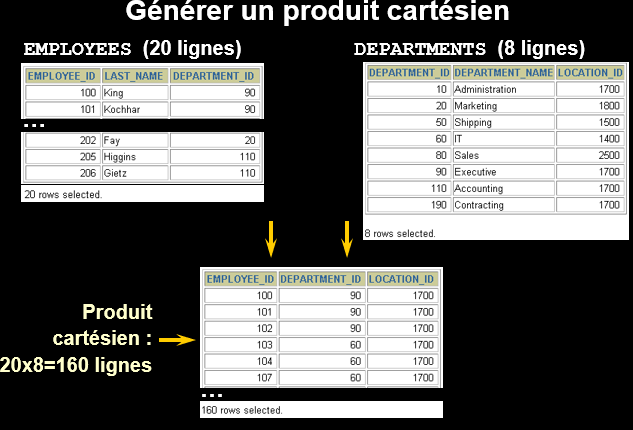
Ensuite, elles sont regroupées par ID de poste dans les groupes de numéro   
de département.

La fonction SUM est donc appliquée à la colonne de salaire pour tous les ID de poste de chaque groupe de numéros de département.

**Produits cartésiens**

Lorsqu'une condition de jointure est incorrecte ou tout simplement omise, un *produit* *cartésien* dans lequel sont affichées toutes les combinaisons de lignes est généré. Toutes les lignes de la première table sont jointes à toutes les lignes de la seconde.

Un produit cartésien tend à produire un nombre important de lignes, donnant un résultat rarement exploitable. C'est pourquoi vous devez toujours inclure une condition de jointure correcte dans une clause WHERE, à moins que vous n'ayez réellement besoin de combiner toutes les lignes de toutes les tables.

Les produits cartésiens sont utiles pour certains tests dans lesquels vous avez besoin de générer un grand nombre de lignes afin de simuler l'existence d'une quantité raisonnable de données.

**Produits cartésiens (suite)**

L'omission d'une condition de jointure génère un produit cartésien. L'exemple de la diapositive affiche le nom de l'employé et celui de son service, issus des tables EMPLOYEES et DEPARTMENTS. Comme aucune clause WHERE n'a été indiquée, toutes les lignes (20) de la table EMPLOYEES ont été jointes à l'ensemble des lignes (8) de la table DEPARTMENTS, donnant ainsi un résultat de 160 lignes.

SELECT last\_name, department\_name dept\_name

FROM employees, departments;

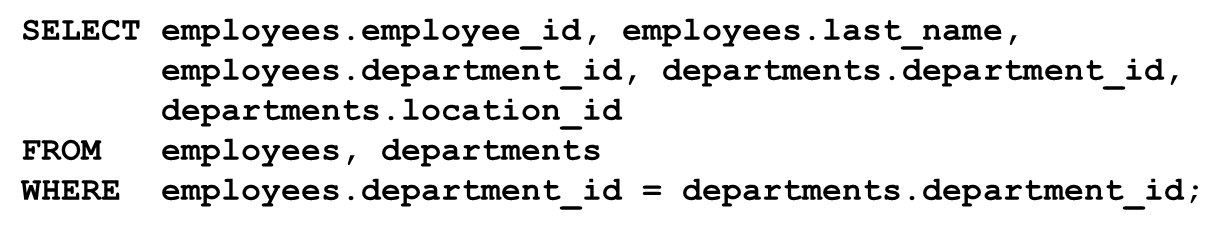
**Définir des jointures**

Pour obtenir des données appartenant à différentes tables de la base de données, vous devez utiliser une condition de *jointure*. Les lignes d'une table peuvent être reliées aux lignes d'une autre table en fonction de valeurs communes existant dans des colonnes correspondantes, désignées en général en tant que colonnes de clé primaire et de clé étrangère.

Pour afficher des données issues d'au moins deux tables, écrivez une condition de jointure simple dans la clause WHERE.

**Equijointures**

Pour déterminer le service d'un employé, vous devez comparer la valeur de la colonne DEPARTMENT\_ID de la table EMPLOYEES à celles de la colonne DEPARTMENT\_ID de la table DEPARTMENTS. La relation entre les tables EMPLOYEES et DEPARTMENTS est appelée *équijointure* puisque les valeurs de la colonne DEPARTMENT\_ID des deux tables sont égales. Ce type de jointure implique fréquemment l'utilisation de clés primaires et de clés étrangères.

**Remarque :** Les équijointures sont également appelées *jointures simples* ou *jointures internes*.

**Autres conditions de recherche**

Outre la jointure, vous pouvez utiliser des critères dans la clause WHERE pour réduire le nombre de lignes d'une ou de plusieurs tables à prendre en considération. Par exemple, pour afficher le nom et le numéro du service de l'employé Matos, vous devez inclure une condition supplémentaire dans la clause WHERE.

SELECT last\_name, employees.department\_id,

department\_name

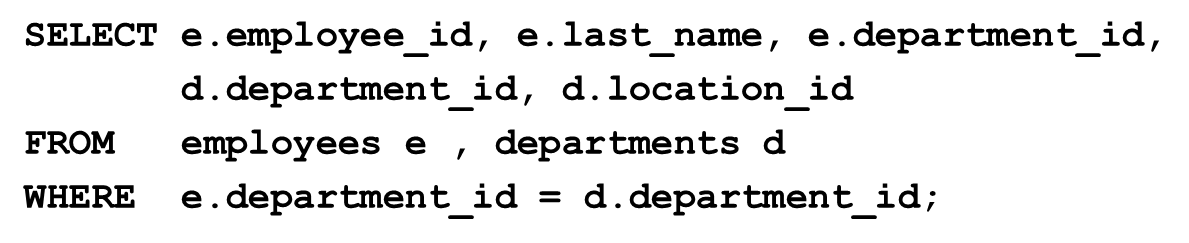
FROM employees, departments

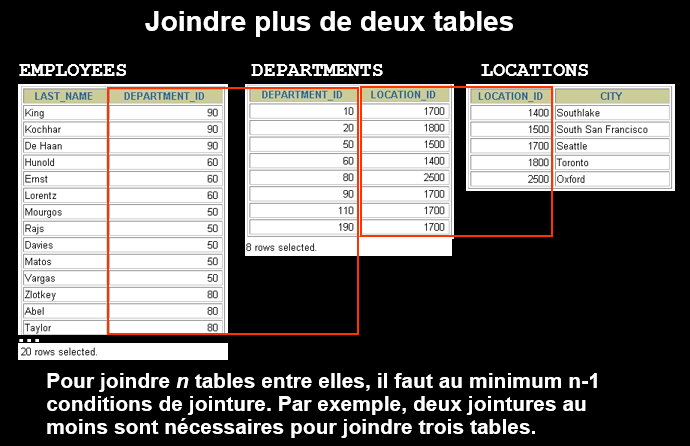
WHERE employees.department\_id = departments.department\_id

AND last\_name = 'Matos';

**Alias de table**

La qualification des noms de colonne à l'aide des noms de table peut prendre beaucoup de temps, en particulier si les noms de table sont longs. Vous pouvez remplacer les noms de table par des *alias de table*. De la même manière qu'un alias de colonne permet de désigner une colonne par un autre nom, un alias de table peut être utilisé comme nom de substitution pour une table. Les alias de table permettent ainsi de réduire le volume du code SQL et donc, de gagner de la place en mémoire.

Observez la manière dont les alias de table sont identifiés dans la clause FROM de l'exemple. Le nom complet de la table est suivi d'un espace, puis de l'alias de table. e est l'alias de la table EMPLOYEES et d celui de la table DEPARTMENTS.

**Autres conditions de recherche**

Vous avez parfois besoin de joindre plus de deux tables. Par exemple, pour afficher le nom, le service et la ville de chaque employé, vous devez joindre les tables EMPLOYEES, DEPARTMENTS et LOCATIONS.

SELECT e.last\_name, d.department\_name, l.city

FROM employees e, departments d, locations l

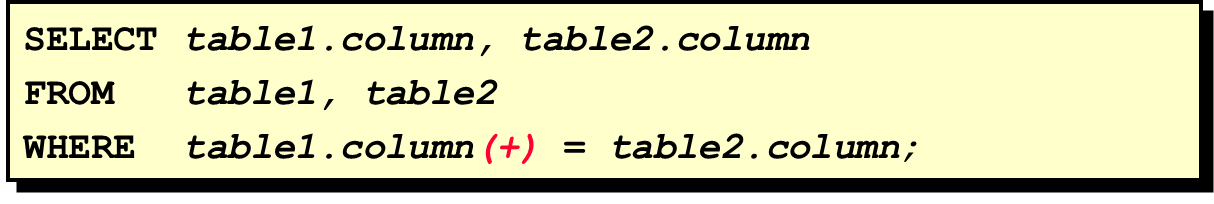
WHERE e.department\_id = d.department\_id

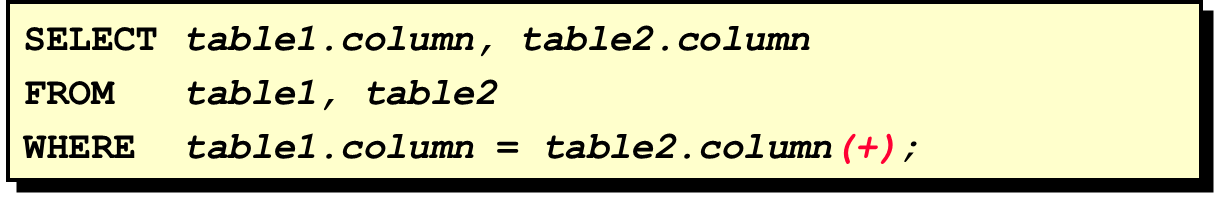
AND d.location\_id = l.location\_id;

**Afficher des enregistrements sans correspondance directe au moyen de jointures externes**

Lorsqu'une ligne ne satisfait pas à une condition de jointure, elle n'apparaît pas dans le résultat de l'interrogation. Par exemple, lorsqu'une équijointure est effectuée entre les tables EMPLOYEES et DEPARTMENTS, l'employé Grant n'apparaît pas car aucun ID de service n'est enregistré à son nom dans la table EMPLOYEES. Ainsi, vous obtenez 19 enregistrements au lieu de 20.

SELECT e.last\_name, e.department\_id, d.department\_name

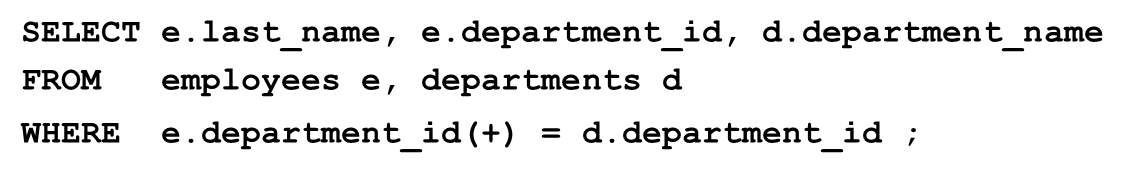
 FROM employees e, departments d

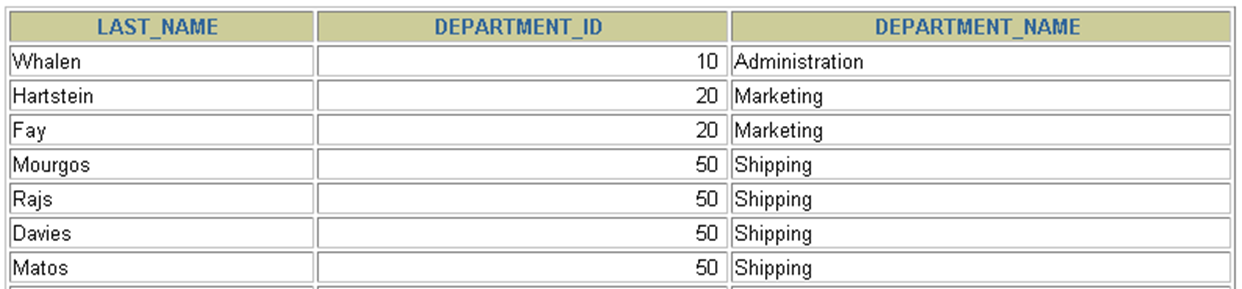
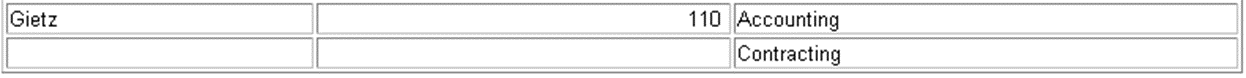
 WHERE e.department\_id = d.department\_id;

Les jointures externes permettent de visualiser également des lignes qui ne répondent pas à la condition de jointure.

L'opérateur de jointure externe est le signe (+). Cet opérateur se présente sous la forme d'un signe plus entre parenthèses (+) et se place du côté de la jointure où l'information est absente. Il crée une ou plusieurs lignes NULL, auxquelles une ou plusieurs lignes de la table complète peuvent être liées.

**Utiliser des jointures externes pour renvoyer des enregistrements sans correspondance directe (suite)**

L'exemple de la diapositive affiche le nom des employés ainsi que le nom et l'ID du service. Le service Contracting ne comporte aucun employé. La ligne correspondante apparaît donc avec un champ vide.

**Restrictions applicables aux jointures externes**

* + - L'opérateur de jointure externe ne peut être placé que *d'un seul* côté de l'expression, à savoir le côté où l'information manque. Il permet de renvoyer les lignes d'une table qui n'ont pas de correspondance directe dans l'autre table.
    - Une condition comportant une jointure externe ne peut pas utiliser l'opérateur IN, ni être liée à une autre condition par l'opérateur OR.

**Syntaxe d'une sous-interrogation**

Une sous-interrogation est une instruction SELECT incluse dans une clause d'une autre instruction SELECT. Vous pouvez créer des instructions puissantes à partir d'instructions simples, à l'aide de sous-interrogations. Les sous-interrogations peuvent s'avérer très utiles lorsque vous devez sélectionner des lignes dans une table, avec une condition qui dépend des données de la table proprement dite.

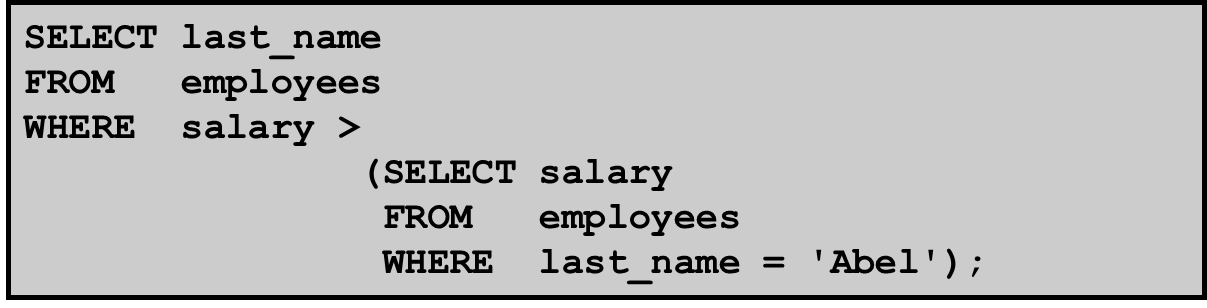
Vous pouvez placer la sous-interrogation dans un certain nombre de clauses SQL, telles que les suivantes :

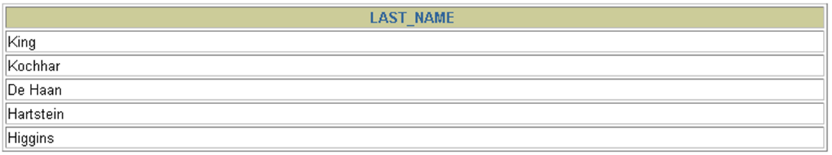
* + - Clause WHERE
    - Clause HAVING
    - Clause FROM

Dans la syntaxe :

*operator* inclut une condition de comparaison telle que >, = ou IN

**Remarque :** Les conditions de comparaison se répartissent en deux catégories : les opérateurs monoligne (>, =, >=, <, <>, <=) et les opérateurs multiligne (IN, ANY, ALL).

La sous-interrogation est souvent appelée instruction SELECT imbriquée, sous-instruction SELECT ou instruction SELECT interne. La sous-interrogation s'exécute généralement en premier et sa sortie est utilisée pour compléter la condition de l'interrogation principale   
(ou externe).

**Utiliser une sous-interrogation**

Dans la diapositive ci-dessus, l'interrogation interne détermine le salaire de l'employé Abel. L'interrogation externe utilise le résultat de l'interrogation interne et l'utilise pour afficher tous les employés qui gagnent plus que ce montant.

**Sous-interrogations monoligne**

Une sous-interrogation monoligne est une interrogation qui renvoie une seule ligne pour l'instruction SELECT interne. Ce type de sous-interrogation utilise un opérateur monoligne. La diapositive ci-dessus répertorie des opérateurs monoligne.

**Exemple**

Afficher les employés dont l'ID de poste est le même que celui de l'employé 141 :

SELECT last\_name, job\_id

FROM employees

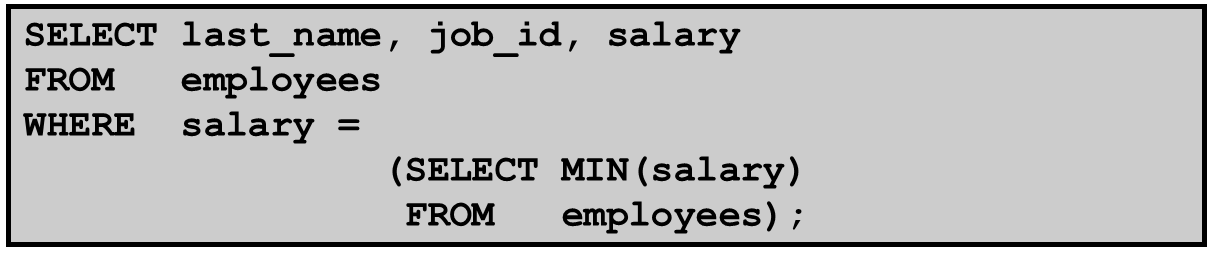
WHERE job\_id =

(SELECT job\_id

FROM employees

WHERE employee\_id = 141);

**Utiliser des fonctions de groupe dans une sous-interrogation**

Vous pouvez afficher des données d'une interrogation principale en utilisant une fonction de groupe dans une sous-interrogation afin de renvoyer une ligne unique. La sous-interrogation est placée entre parenthèses, après la condition de comparaison.

L'exemple de la diapositive ci-dessus affiche le nom de famille, l'ID de poste et le salaire de tous les employés dont le salaire est égal au salaire minimum. La fonction de groupe MIN renvoie une valeur unique (2500) à l'interrogation externe.

**Sous-interrogations multiligne**

Les sous-interrogations qui renvoient plusieurs lignes sont appelées sous-interrogations multiligne. Avec une sous-interrogation multiligne, vous utilisez un opérateur multiligne plutôt qu'un opérateur monoligne. L'opérateur multiligne accepte une ou plusieurs valeurs :

SELECT last\_name, salary, department\_id

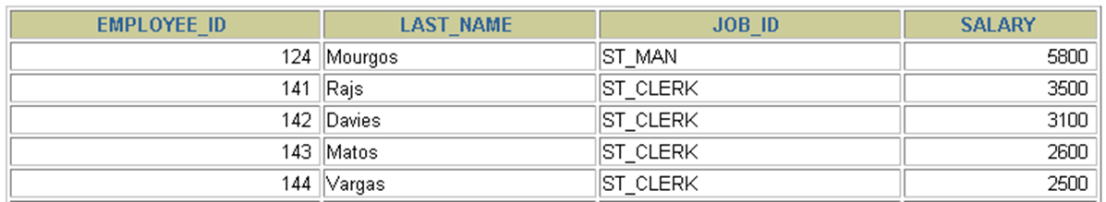
FROM employees

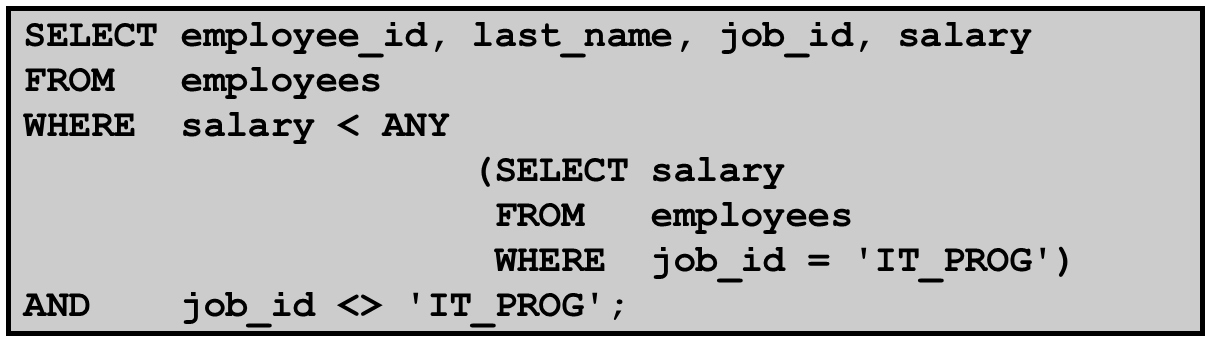
WHERE salary IN (SELECT MIN(salary)

FROM employees

GROUP BY department\_id);

**Sous-interrogations multiligne (suite)**

L'opérateur ANY (et son synonyme, à savoir l'opérateur SOME) compare une valeur à *chaque*valeur renvoyée par une sous-interrogation. L'exemple de la diapositive ci-dessus affiche les employés qui ne sont pas programmeurs et dont le salaire est inférieur à celui de n'importe quel programmeur. Le salaire maximum perçu par un programmeur est de 9 000 $.

<ANY signifie moins que le maximum. >ANY signifie plus que le maximum. =ANY est équivalent à IN.

**L'opérateur UNION renvoie les résultats des deux interrogations après avoir éliminé les doublons. Affichez les informations relatives au poste actuel et au poste précédent de tous les employés. Affichez chaque employé une seule fois.**

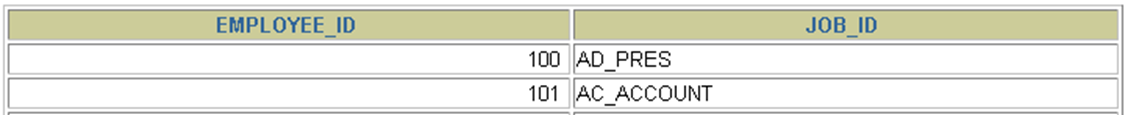
**SELECT employee\_id, job\_id**

**FROM employees**

**UNION**

**SELECT employee\_id, job\_id**

**FROM job\_history;**



**L'opérateur UNION ALL renvoie les résultats des deux interrogations, en incluant tous les doublons. Affichez le département actuel et le département précédent de tous les employés.**

**SELECT employee\_id, job\_id, department\_id**

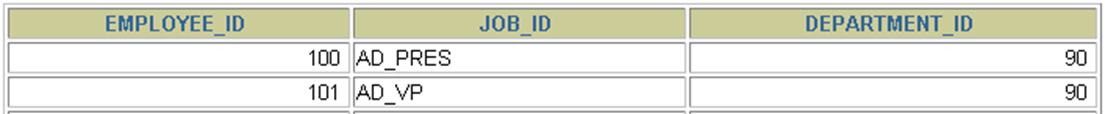
**FROM employees**

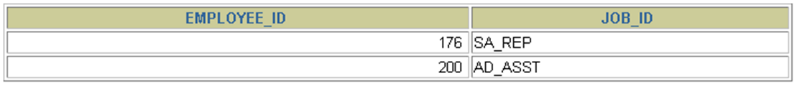
**UNION ALL**

**SELECT employee\_id, job\_id, department\_id**

**FROM job\_history**

**ORDER BY employee\_id;**



**L'opérateur INTERSECT renvoie les lignes qui sont communes aux deux interrogations. Affichez l'ID d'employé et l'ID de poste des employés dont le poste actuel est le même que celui qu'ils occupaient lors de leur embauche (c'est-à-dire ceux qui ont changé de poste, mais qui occupent aujourd'hui le même poste qu'à l'origine).**

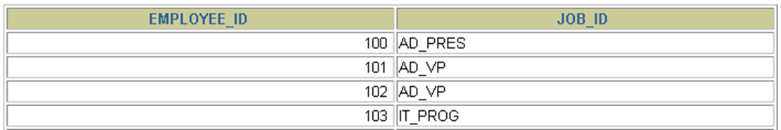
**SELECT employee\_id, job\_id**

**FROM employees**

**INTERSECT**

**SELECT employee\_id, job\_id**

**FROM job\_history;**

**L'opérateur MINUS renvoie les lignes de la première interrogation qui ne sont pas présentes dans la deuxième interrogation. Affichez l'ID des employés qui n'ont jamais changé de poste.**

**SELECT employee\_id,job\_id**

**FROM employees**

**MINUS**

**SELECT employee\_id,job\_id**

**FROM job\_history;**