next up previous contents index

Next: Table des matières

Bases de Données Relationnelles

et

Systèmes de Gestion de Bases de Données Relationnels
Le Langage sql

Yolaine Bourda

Yolaine.Bourda@supelec.fr

next up previous index

Next: Bases de Données et **Up:** Bases de Données Relationnelles **Previous:** Bases de Données Relationnelles

Table des matières

Yolaine.Bourda@supelec.fr

Bases de Données et Systèmes de Gestion de Bases de Données

Yolaine.Bourda@supelec.fr

Introduction

Les bases de données sont actuellement au coeur du système d'information des entreprises. Les systèmes de gestion de bases de données, initialement disponibles uniquement sur des ``mainframes'', peuvent maintenant être installés sur tous les types d'ordinateurs y compris les ordinateurs personnels. Mais attention, souvent on désigne, par abus de langage, sous le nom ``bases de données'' des ensembles de données qui n'en sont pas.

Qu'est-ce donc qu'une base de données? Que peut-on attendre d'un système de gestion de bases de données? C'est à ces questions, entre autres, que cet ouvrage essaie d'apporter des réponses.

Dans un premier temps, et de façon informelle, on peut considérer une Base de Données (BD) comme une grande quantité de données, centralisées ou non, servant pour les besoins d'une ou plusieurs applications, interrogeables et modifiables par un groupe d'utilisateurs travaillant en parallèle. Quant au Système de Gestion de Bases de Données (SGBD), il peut être vu comme le logiciel qui prend en charge la structuration, le stockage, la mise à jour et la maintenance des données ; c'est, en fait, l'interface entre la base de données et les utilisateurs ou leurs programmes.

Yolaine.Bourda@supelec.fr

Les limites à l'utilisation des fichiers

L' utilisation de fichiers impose d'une part, à l'utilisateur de connaître l'organisation (séquentielle, indexée, ...) des fichiers qu'il utilise afin de pouvoir accéder aux informations dont il a besoin et, d'autre part, d'écrire des programmes pour pouvoir effectivement manipuler ces informations. Pour des applications nouvelles, l'utilisateur devra obligatoirement écrire de nouveaux programmes et il pourra être amené à créer de nouveaux fichiers qui contiendront peut-être des informations déjà présentes dans d'autres fichiers.

De telles applications sont :

- rigides,
- contraignantes,
- longues et coûteuses à mettre en oeuvre.

Les données associées sont :

- mal définies et mal désignées,
- redondantes,
- peu accessibles de manière ponctuelle,
- peu fiables.

La prise de décision est une part importante de la vie d'une société. Mais elle nécessite d'être bien informé sur la situation et donc d'avoir des informations à jour et disponibles immédiatement.

Les utilisateurs, quant à eux, ne veulent plus de systèmes d'information constitués d'un ensemble de programmes inflexibles et de données inaccessibles à tout non spécialiste ; ils souhaitent des systèmes d'informations globaux, cohérents, directement accessibles (sans qu'ils aient besoin soit d'écrire des programmes soit de demander à un programmeur de les écrire pour eux) et des réponses immédiates aux questions qu'ils posent. On a donc recherché des solutions tenant compte à la fois des désirs des utilisateurs et des progrès techniques. Cette recherche a abouti au concept de base de données.
Définition (base de données) : Une base de données est un ensemble d'informations sur un sujet qui est :

- exhaustif,
- non redondant,
- structuré,
- persistant.

Définition (système de gestion de base de données) : Un système de gestion de base de données est un logiciel qui permet de :

- décrire,
- modifier,
- interroger,
- administrer,

les données d'une base de données.

---

next | up | previous | contents | index

Next: Objectifs des systèmes de **Up:** Introduction **Previous:** Introduction Yolaine.Bourda@supelec.fr

Objectifs des systèmes de gestion de bases de données

Les bases de données et les systèmes de gestion de bases de données ont été créés pour répondre à un certain nombre de besoins et pour résoudre un certain nombre de problèmes.

Des objectifs principaux ont été fixés aux systèmes de gestion de bases de données dès l'origine de ceux-ci et ce, afin de résoudre les problèmes causés par la démarche classique.

Ces objectifs sont les suivants :

- [Indépendance physique]
  La façon dont les donnnées sont définies doit être indépendante des structures de stockages utilisées.
- [Indépendance logique]
  Un même ensemble de données peut être vu différemment par des utilisateurs différents. Toutes ces visions personnelles des données doivent être intégrés dans une vision globale.
- [Manipulations des données par des non informaticiens]
  Il faut pouvoir accéder aux données sans savoir programmer ce qui signifie des langages ``quasi naturels".
- [Efficacité des accès aux données]
  Ces langages doivent permettre d'obtenir des réponses aux interrogations en un temps ``raisonnable". Ils doivent donc être optimisés et, entre autres, il faut un mécanisme permettant de minimiser le nombre d'accès disques. Tout ceci, bien sur, de façon complètement transparente pour l'utilisateur.
- [Administration centralisée des données]
  Des visions différentes des données (entre autres) se résolvent plus facilement si les données sont administrées de façon centralisée.
- [Non redondance des données]
  Afin d'éviter les problèmes lors des mises à jour, chaque donnée ne doit être présente qu'une seule fois dans la base.
- [Cohérence des données]
  Les données sont soumises à un certain nombre de contraintes d'intégrité qui définissent un état cohérent de la base. Elles doivent pouvoir être exprimées simplement et vérifiées automatiquement à chaque insertion, modification ou suppression des données.
- [Partageabilité des données]
  Il s'agit de permettre à plusieurs utilisateurs d'accéder aux mêmes données au même moment. Si ce problème est simple à résoudre quand il s'agit uniquement d'interrogations et quand on est dans un contexte mono-utilisateur, cela n'est plus le cas quand il s'agit de modifications dans un contexte muti-utilisateurs. Il s'agit alors de pouvoir :
  - permettre à deux (ou plus) utilisateurs de modifier la même donnée ``en même temps" ;

    - assurer un résultat d'interrogation cohérent pour un utilisateur consultant une table pendant qu'un autre la modifie.

- [Sécurité des données]
  Les données doivent pouvoir être protégées contre les accès non autorisés. Pour cela, il faut pouvoir associer à chaque utilisateur des droits d'accès aux données.
- [Résistance aux pannes]
  Que se passe-t-il si une panne survient au milieu d'une modification, si certains fichiers contenant les données deviennent illisibles? Les pannes, bien qu'étant assez rares, se produisent quand même de temps en temps. Il faut pouvoir, lorsque l'une d'elles arrive, récupérer une base dans un état ``sain''. Ainsi, après une panne intervenant au milieu d'une modification deux solutions sont possibles : soit récupérer les données dans l'état dans lequel elles étaient avant la modification, soit terminer l'opération interrompue.

Malheureusement, ces objectifs ne sont pas toujours atteints.

---

next | up | previous | contents | index

Next: Concepts de base **Up:** Introduction **Previous:** Les limites à l'utilisation

Yolaine.Bourda@supelec.fr

Concepts de base

Pour assurer ces objectifs (surtout les deux premiers), trois niveaux de description des données ont été définis par la norme ANSI/SPARC.

Niveau interne

Description du stockage des données au niveau des unités de stockage, des fichiers, ... On appelle cette description le schéma interne.

Niveau conceptuel

Description de la structure de toutes les données qui existent dans la base, description de leurs propriétés (relations qui existent entre elles) c'est-à-dire de leur sémantique inhérente, sans soucis d'implémentation physique ni de la façon dont chaque groupe de travail voudra s'en servir. On appelle cette description le schéma conceptuel.

Niveau externe

Description pour chaque utilisateur de sa perception des données. On appelle cette description le schéma externe ou vue.

Le résultat de la conception d'une base de données sera une description des données. Par description on entend définir les propriétés d'ensembles d'objets modélisés dans la base de données et non pas d'objets particuliers. Les objets particuliers sont définis par les programmes d'applications lors des insertions et des mises à jour des données. Ils doivent alors vérifier les propriétés des ensembles auxquels ils appartiennent.

Cette description des données sera effectuée en utilisant un modèle de données. Ce dernier est un outil intellectuel utilisé pour comprendre l'organisation logique des données. C'est un ensemble de concepts et de règles pour les utiliser, permettant de construire avec des types de données une représentation de la réalité.

Un système de gestion de bases de données est caractérisé par le modèle de description des données qu'il supporte. Les données sont décrites sous la forme de ce modèle, grâce à un langage de description des données. Cette description est appelée schéma. Les modèles utilisés sont : réseau, relationnel, objet, ...

Une fois la base de données spécifiée, on peut y insérer des données, les récupérer, les modifier et les détruire. C'est ce qu'on appelle manipuler les données. Les données peuvent être manipulées non seulement par un langage spécifique de manipulation des données mais aussi par des langages de programmation ``classiques".

Next: Composants des systèmes de **Up:** Introduction **Previous:** Objectifs des systèmes de
Yolaine.Bourda@supelec.fr

next up previous contents index

Next: Un peu d'histoire **Up:** Introduction **Previous:** Concepts de base

Composants des systèmes de gestion de bases de données

Un système de gestion de bases de données va donc posséder un certain nombre de composants logiciels et ce, quelque soit le modèle de données qu'il supporte. On trouve donc des composants chargés de :

La description des données

Cette partie sera constituée des outils (en gros des langages) permettant de décrire la vision des données de chaque utilisateur et l'intégration dans une vision globale. On y trouve aussi les outils permettant de décrire le stockage physique des données.

La récupération des données

Cette partie prend en charge l'interrogation et la modification des données et ce, de façon optimisée. Elle est composée de langages de manipulation de données spécifiques et d'extensions de langages ``classiques''. Elle gère aussi les problèmes de sécurité.

La sauvegarde et la récupération après pannes

Cette partie comporte des outils permettant de sauvegarder et de restaurer de façon explicite une base de données. Elle comporte aussi des mécanismes permettant, tant qu'une modification n'est pas finie, de pouvoir revenir à l'état de la base avant le début de cette modification.

Les accès concurrents aux données

C'est la partie chargée du contrôle de la concurrence des accès aux données. Elle doit être telle que chaque utilisateur attende le moins possible ses données tout en étant certain d'obtenir des données cohérentes en cas de mises à jour simultanées de la base.

Chacun de ces composants est décrit de façon plus détaillée par la suite.

---

next up previous contents index

Next: Un peu d'histoire **Up:** Introduction **Previous:** Concepts de base Yolaine.Bourda@supelec.fr

Un peu d'histoire

[1960]

- [1960] Uniquement des systèmes de gestion de fichiers plus ou moins sophistiqués.
- [1970] Début des systèmes de gestion de bases de données réseaux et hiérarchiques proches des systèmes de gestion de fichiers. Ces systèmes de gestion de bases de données avaient remplis certains des objectifs précédents mais on ne pouvait pas interroger une base sans savoir où était l'information recherchée (on ``naviguait'') et sans écrire de programmes.

  Sortie du papier de CODD sur la théorie des relations, fondement de la théorie des bases de données relationnelles.
- [1980] Les systèmes de gestion de bases de données relationnels apparaissent sur le marché.
- [1990] Les systèmes de gestion de bases de données relationnels dominent le marché.

  Début des systèmes de gestion de bases de données orientés objet.

Yolaine.Bourda@supelec.fr

Le modèle relationnel

Yolaine.Bourda@supelec.fr

next up previous contents index

Next: Définitions **Up:** Le modèle relationnel **Previous:** Le modèle relationnel

Introduction

Le modèle relationnel a été formalisé par CODD en 1970. Quelques exemples de réalisation en sont :

DB2(IBM), INFORMIX, INGRES, ORACLE.

Dans ce modèle, les données sont stockées dans des tables, sans préjuger de la façon dont les informations sont stockées dans la machine. Un ensemble de données sera donc modélisé par un ensemble de tables.

Le succès du modèle relationnel auprès des chercheurs, concepteurs et utilisateurs est dû à la puissance et à la simplicité de ses concepts. En outre, contrairement à certains autres modèles, il repose sur des bases théoriques solides, notamment la théorie des ensembles et la logique mathématique(théorie des prédicats d'ordre 1).

Les objectifs du modèle relationnel :

- proposer des schémas de données faciles à utiliser,
- améliorer l'indépendance logique et physique,
- mettre à la disposition des utilisateurs des langages de haut niveau pouvant éventuellement être utilisés par des non informaticiens,
- optimiser les accès à la base de données,
- améliorer l'intégrité et la confidentialité,
- fournir une approche méthodologique dans la construction des schémas.

De façon informelle, on peut définir le modèle relationnel de la manière suivante :

- Les données sont organisées sous forme de tables à deux dimensions, encore appelées relations et chaque ligne n-uplet ou tuple,
- les données sont manipulées par des opérateurs de l'algèbre relationnelle,
- l'état cohérent de la base est défini par un ensemble de contraintes d'intégrité.

Au modèle relationnel est associée la théorie de la normalisation des relations qui permet de se débarrasser des incohérences au moment de la conception d'une base de données.

---

next up previous contents index

Next: Définitions **Up:** Le modèle relationnel **Previous:** Le modèle relationnel

Yolaine.Bourda@supelec.fr

Définitions

Définition (Domaine) : Ensemble de valeurs. **Définition (Relation) :** Sous-ensemble du produit cartésien d'une liste de domaines caractérisé par un nom.

En d'autres termes, une relation n'est ni plus ni moins qu'une table dans laquelle chaque colonne correspond à un domaine et porte un nom ce qui rend leur ordre sans aucune importance. **Définition (Attribut) :** Colonne d'une relation caractérisée par un nom. **Définition (Schéma de relation) :** Nom de la relation, suivi de la liste des attributs avec leurs domaines. **Définition (Base de données relationnelles) :** Base de données dont le schéma est un ensemble de schémas de relations et dont les occurrences sont les tuples de ces relations. **Définition (Système de gestion de bases de données relationnel) :** C'est un logiciel supportant le modèle relationnel, et qui peut manipuler les données avec des opérateurs relationnels.

Yolaine.Bourda@supelec.fr

Opérateurs relationnels

Définition (Projection) : Opération qui consiste à supprimer des attributs d'une relation et à éliminer les tuples en double apparaissant dans la nouvelle relation. Cette opération est notée Π.**Définition (Restriction) :** Opération qui consiste à supprimer les tuples d'une relation ne satisfaisant pas la condition précisée. **Définition (Jointure) :** Opération qui consiste à faire le produit cartésien de deux relations, puis à supprimer les tuples ne satisfaisant pas une condition portant sur un attribut de la première relation et sur un attribut de la seconde. **Définition (Union) :** Opération portant sur deux relations ayant le même schéma et construisant une troisième relation constituée des tuples appartenant à chaque relation. Les tuples en double sont éliminés. **Définition (Différence relationnelle) :** Opération portant sur deux relations ayant le même schéma et construisant une troisième relation dont les tuples sont constitués de ceux ne se trouvant que dans une seule relation. **Définition (Intersection) :** Opération portant sur deux relations ayant le même schéma et construisant une troisième relation dont les tuples sont constitués de ceux appartenant aux deux relations.

Yolaine.Bourda@supelec.fr

Formes normales

Yolaine.Bourda@supelec.fr

Dépendance fonctionnelle

Définition (dépendance fonctionnelle) : Soit $R(A_1,A_2,...A_n)$ un schéma de relation, et X et Y des sous-ensembles de $\{A_1, A_2, ...A_n\}$. On dit que X détermine Y ou que Y dépend fonctionnellement de X si, et seulement si, des valeurs identiques de X impliquent des valeurs identiques de Y. On le note : X -> Y

Définition (dépendance fonctionnelle élémentaire) : C'est une dépendance fonctionnelle de la forme X -> Y, où A est un attribut unique n'appartenant pas à X et où il n'existe pas X' inclus dans X tel que X -> Y.

Yolaine.Bourda@supelec.fr

Notion de clé

Définition (clé de relation) : Soit $R(A_1, A_2, ..., A_n)$ un schéma de relation, et X un sous-ensemble de $(A_1, A_2, ..., A_n)$, X est une clé si, et seulement si, :

- X -> $(A_1, A_2, ..., A_n)$
- X est minimal : il n'existe pas de sous-ensemble Y de X tel que $\underline{Y \rightarrow (A_1, A_2, ..., A_n)}$

Yolaine.Bourda@supelec.fr

Formes normales

Définition (Première forme normale) : Une relation est en première forme normale si et seulement si tout attribut contient une valeur atomique. **Définition (Deuxième forme normale) :** Une relation est en deuxième forme normale si et seulement si :

- elle est en première forme normale ;
- tout attribut n'appartenant pas à une clé ne dépend pas que d'une partie de cette clé.

Définition (Troisième forme normale) : Une relation est en troisième forme normale si et seulement si :

- elle est en deuxième forme normale ;
- tout attribut n'appartenant pas à une clé ne dépend pas d'un attribut non clé.

Définition (Forme normale de BOYCE-CODD) : Une relation est en Forme normale de BOYCE-CODD (BCNF) si, et seulement si, les seules dépendances fonctionnelles élémentaires sont celles dans lesquelles une clé détermine un attribut.

Yolaine.Bourda@supelec.fr

Langages de manipulation de données relationnelles

Ces langages, dits assertionnels, sont basés sur la logique des prédicats d'ordre 1 et permettent de spécifier les données que l'on souhaite obtenir, sans dire comment y accéder. On doit y trouver des opérations permettant de :

[la modification]

- [la recherche] retrouver des tuples vérifiant certains critères,
- [l'insertion] ajouter des tuples,
- [la suppression] enlever des tuples vérifiant certains critères,
- [la modification] modifier des tuples vérifiant certains critères.

Un langage de manipulation de données n'est pas utilisable à lui seul, il doit aussi pouvoir être incorporable dans un langage de programmation classique.

On peut distinguer trois grandes classes de langages :

- Les langages algébriques basés sur l'algèbre relationnelle de CODD dans lesquels les requêtes sont exprimées comme l'application des opérateurs relationnels sur des relations. C'est dans cette catégorie que l'on trouve le langage sql(structured query language), standard pour l'interrogation de bases de données.
- Les langages basés sur le calcul relationnel de tuples construits à partir de la logique des prédicats dans lesquels les variables manipulées sont des tuples.
- Les langages basés sur le calcul relationnel de domaines, construit aussi à partir de la logique des prédicats mais en faisant varier les variables sur les domaines des relations.

Le langage sql (Structured Query Language) comprend à lui seul l'ensemble des instructions nécessaires à la spécification et à l'utilisation d'une base de données relationnelle. C'est un langage de type déclaratif c'est-à-dire que l'on spécifie les propriétés des données que l'on recherche et pas, comme dans un langage impératif, comment les retrouver.

Le langage sql est un langage normalisé, la dernière version de la norme date de 92 et, souvent, on y fait référence en parlant de sql-92. La prochaine version de la norme est en cours de rédaction afin d'intégrer, entre autres, la notion de types abstraits algébriques, on la désigne sous le nom de sql3.

C'est à la fois :

- un langage d'interrogation de données (LID) : select ;
- un langage de manipulation de données (LMD) : update, `INSERT`, `DELETE` ;
- un langage de definition des données (LDD) : `ALTER`, `CREATE`, `DROP`;
- un langage de contrôle des données et des utilisateurs (LCD) : `GRANT`, `REVOKE`.

Next: Remarques **Up:** Le modèle relationnel **Previous:** Formes normales Yolaine.Bourda@supelec.fr

next | up | previous | contents | index

Next: L'optimiseur de requêtes **Up:** Le modèle relationnel **Previous:** Langages de manipulation de

Remarques

Beaucoup de systèmes de gestion de données (et non pas de gestion de bases de données) sont vendus comme étant relationnels, souvent parce qu'ils présentent les données sous forme de tables.

Un système est dit minimalement relationnel s'il satisfait aux conditions suivantes :

- toute information dans la base est représentée par des valeurs dans des tables,
- il n'y a pas de pointeurs visibles par l'utilisateur entre les tables,
- le système doit supporter au moins les opérateurs relationnels de restriction, projection, jointure naturelle.

Un système est dit complètement relationnel s'il satisfait, en plus, aux conditions suivantes :

- il supporte tous les opérateurs de l'algèbre relationnelle,
- il supporte la contrainte d'unicité de clé d'une relation,
- il supporte les contraintes référentielles qui permettent de s'assurer que la valeur d'une donnée d'une relation existe dans une autre relation (notion de foreign key).

En dépit de sa simplicité et de son élégance le modèle relationnel n'apporte pas une réponse satisfaisante à tous les problèmes des applications. Il faut :

- Pouvoir prendre en compte des "objets" structurés ainsi que les opérations qui leur sont associées (bases de données orientées objet).
- Prendre en compte des données peu structurées : textes, sons, images, graphiques (bases de données multi-média).
- Faire le pont avec l'intelligence artificielle afin de pouvoir déduire de nouvelles données à partir de celles existant déjà (bases de données déductives)

---

next | up | previous | contents | index

Next: L'optimiseur de requêtes **Up:** Le modèle relationnel **Previous:** Langages de manipulation de

Yolaine.Bourda@supelec.fr

L'optimiseur de requêtes

Yolaine.Bourda@supelec.fr

Introduction

Avec des langages tels que sql, l'utilisateur précise les propriétés des données qui l'intéressent sans fournir d'algorithme d'accès. Les optimiseurs de requête sont là pour cela, leurs objectifs sont de :

- vérifier la correction syntaxique de la question,
- rechercher des questions équivalentes plus simples,
- déterminer l'ordre des opérations élémentaires d'accès en vue de :
  - réduire le nombre d'entrées/sorties,
  - réduire le temps cpu,
  - réduire la taille des ressources mémoires nécessaires,
  - optimiser en premier les questions les plus fréquentes.

Chaque sgbd contient un optimiseur de requêtes qui peut être légèrement différent d'un sgbd à l'autre. Les algorithmes utilisés sont rarement connus en détail mais les grandes étapes de l'optimisation sont en général :

1. Représenter la requête sous une forme interne et la décomposer en une séquence d'opérations élémentaires.
2. Transformer la requête par :
   - simplification : remplacement d'une question par une question équivalente plus simple,
   - ordonnancement des opérations élémentaires.
3. Construire un ensemble de plans d'exécution candidats.
4. Calculer le coût de chaque plan et choisir le meilleur.
5. Executer la requête.

Yolaine.Bourda@supelec.fr

Réecriture des requêtes

La reformulation de la requête par l'optimiseur a pour but de minimiser le nombre de calculs effectués. Voici quelques exemples de transformations possibles :

- Les expressions et conditions faisant intervenir des constantes sont, si c'est possible, évaluées. Ainsi, `age > 60 - 10` sera transformé en `age > 50` mais `age + 10 > 60` ne sera pas réécrit en `age > 50`.
- un `BETWEEN` sera remplacé par une expression contenant les symboles >= et <= séparés par `AND`.
- Une expression logique complexe préfacée par `NOT` sera réécrite afin que celui-ci soit mis devant chacune de ses sous-expressions. Le `NOT` devant une expression simple sera, si c'est possible, supprimé. Ainsi `NOT age > 50` deviendra `age <= 50`.
- Les requêtes comportant des `OR` sont réécrites en utilisant des `UNION`.
- Les sous-requêtes sont remplacées par des jointures.

Yolaine.Bourda@supelec.fr

Choix des chemins d'accès

L'optimiseur d'interrogation d'oracle est chargé de déterminer le chemin d'accès optimal pour exécuter un select .

Le choix porte essentiellement sur l'utilisation des index portant sur les colonnes mentionnées dans la clause `WHERE`, sachant qu'en l'absence d'index utilisable une interrogation portant sur une table nécessitera le balayage total de la table.

Yolaine.Bourda@supelec.fr

Requête portant sur une seule table

L'optimiseur d'interrogation classe les différents types de prédicats en fonction de leur syntaxe dans l'ordre suivant ; du plus sélectif au moins sélectif :

- `WHERE rowid` = constante
- `WHERE` clé de cluster = constante
- `WHERE` colonne de `CONNECT BY` indexée = constante
- `WHERE` colonne indexée (unique) = constante
- `WHERE` colonne indexée (non unique) = constante
- `WHERE` colonne indexée = constante
- `WHERE` colonne indexée `BETWEEN` val1 `AND` val2 ou `WHERE` colonne indexée `LIKE` 'c%'
- `WHERE` colonne indexée > ou < constante
- `WHERE MAX` ou `MIN` de colonne indexée
- `WHERE` colonne non indexée = constante
- `WHERE` colonne is null ou is not null
- `WHERE` colonne ! = constante
- `WHERE` colonne indexée `LIKE '%c'` ou `WHERE` colonne indexée `LIKE '_c'`

oracle peut dans certains cas utiliser plusieurs index sur une même table (5 au max), y compris pour évaluer des prédicats lies par `OR`.

Un select avec plusieurs prédicats indexés liés par `AND` sera exécuté comme une intersection du résultat de plusieurs select bénéficiant chacun d'un index, si les prédicats sont des égalités.

SELECT * FROM emp WHERE nom = 'MARTIN' AND n_dept = 20;De même un select avec plusieurs prédicats liés par `OR` sera exécuté comme une union du résultat de plusieurs select bénéficiant chacun d'un index. Dans ce cas, il faut que tous les prédicats bénéficient d'un index.

Exemple 10153

SELECT * FROM emp WHERE nom = 'MARTIN' OR n_dept = 20 ;

Dans ces cas il se peut que l'utilisation de l'un des index soit pénalisante. Ce peut être le cas d'un index peu sélectif, alors que les autres index utilisables sont très sélectifs. Dans ce cas l'on peut empêcher oracle d'utiliser les index inadéquats, en formulant les conditions de telle façon que le critère de recherche soit une fonction de la colonne indexée et non pas la colonne indexée elle même (dans ce cas oracle n'utilise pas l'index).

Exemple 10154

SELECT * FROM emp WHERE nom = 'MARTIN' AND n_dept + 0 = 20;où les champs `nom` et `n_dept` sont indexés. Le fait d'ajouter 0 à la colonne `n_dept` empêche d'utiliser l'index sur `n_dept`,

qui ne ferait que ralentir la requête car il est peu sélectif. L'index sur nom, qui est très sélectif, suffit.

De la même façon, l'on peut concaténer une chaîne vide à une colonne de type caractère pour empêcher oracle d'utiliser l'index sur cette colonne :

SELECT * FROM emp WHERE nom = 'MARTIN' AND fonction ||''= 'directeur' ;

---

next up previous contents index

Next: Jointures sans index **Up:** L'optimiseur de requêtes **Previous:** Choix des chemins d'accès

Yolaine.Bourda@supelec.fr

next up previous contents index

Next: Jointures avec index **Up:** L'optimiseur de requêtes **Previous:** Requête portant sur une

Jointures sans index

Dans le cas de jointure sans index, oracle classe au préalable chaque table selon le critère de jointure. L'algorithme est symétrique et l'ordre dans lequel les tables sont mentionnées n'a donc pas d'influence sur les performances.

Yolaine.Bourda@supelec.fr

Jointures avec index

Dans le cas où le critère de jointure est indexé au moins dans l'une des tables, oracle choisit une table directrice : c'est cette table qui sera lue en premier et qui pilotera la jointure. Si le critère de jointure est indexé dans une seule des tables, oracle choisit l'autre table comme table directrice. Si le critère de jointure est indexé dans les deux tables, la table directrice est celle sur laquelle portent les prédicats les moins sélectifs(compte tenu de la classification des prédicats ci-dessus). Si les prédicats portant sur chaque table sont équivalents, la table directrice sera la dernière table mentionnée dans la clause `FROM`.

Ainsi, dans l'exemple suivant, l'optimiseur considère que les restrictions portant sur chacune des tables sont équivalentes, et la table directrice sera la table `dept` qui est citée en dernier dans la clause `FROM`.

SELECT * FROM emp, dept WHERE emp.n_dept = dept.n_dept AND dept.nom = 'vente' AND fonction ='directeur' ;

Yolaine.Bourda@supelec.fr

ORDER BY

[order by]`ORDER BY`

L'optimiseur d'interrogation d'oracle décidera dans certains cas de passer par l'index pour sélectionner les lignes d'une table selon un certain ordre. Pour cela il faut que :

- le critère de classement soit le contenu d'une colonne indexée
- cette colonne ait l'attribut not null (obligatoire)
- la sélection porte sur toute la table (pas de `WHERE`)

Exemple 10155

Le select suivant utilisera l'index sur la colonne `salaire` si cette colonne ne contient pas de valeurs `NULL`.

SELECT nom, salaire FROM emp ORDER BY salaire ;

Yolaine.Bourda@supelec.fr

Cohérence des interrogations et accès concurrents

Yolaine.Bourda@supelec.fr

introduction

Ce chapitre étudie les conditions de bon fonctionnement d'un système de gestion de bases de données travaillant dans un contexte multi-utilisateurs multi-tâches. Comme un système d'exploitation classique gère, partage les ressources et synchronize les processus, un système de gestion de bases de données doit assurer (entre autres) le partage des données. On retrouve, ici, les problèmes classiques de gestion de ressources partagées.

Les problèmes à éviter se classent, en gros, en deux catégories : les pertes d'opérations et la récupération d'une bases de données incohérente. De façon plus détaillée on peut avoir des :

- lectures inconsistantes
- lectures non reproductibles
- des lectures de données fantomes
- des modifications perdues
- des données incohérentes
- des accès à des données inexistantes

Il est facile de réaliser un système de gestion de bases de données avec un contrôle de concurrence simple et efficace consistant, par exemple, à verrouiller la base entière pendant l'exécution de chaque transaction. Les performances d'un tel système se dégraderaient rapidement avec l'augmentation du nombre d'usagers et il deviendrait très rapidement inutilisable.

Un système réaliste doit assurer que :

- lectures et écritures ne mettant pas en jeu les mêmes données peuvent s'exécuter en parallèle ;
- les lectures ne sont pas en attente de fin d'écriture ou de fin de lecture des mêmes données ;
- les écritures ne sont pas en attente de fin de lecture des mêmes données ;
- les écritures attendent uniquement la fin d'écriture des mêmes données.

Il faut trouver une solution qui préserve l'intégrité des données sans pénaliser les performances. Cette solution repose sur les notions de :

- fichier image avant, ``before image'' ou ``rollback segment'', tout ces mots désignant un, ou plusieurs, fichiers contenant la valeur des données avant la modification ainsi que celle-ci.
- base cohérente et de transaction (ces deux notions sont définies ci-dessous).

Définition (contrainte d'intégrité) : assertion(propriété, prédicat) qui doit être vérifiée par certaines données à des instants déterminés. **Définition (bases de données cohérente) :** bases de données dont toutes les contraintes d'intégrité sont respectées.

Les contraintes d'intégrité peuvent non seulement porter sur une donnée unique : contrainte de domaine de variation, contrainte de plage de valeurs, mais aussi sur des données référençant d'autres données . **Définition (transaction) :** unité de traitement séquentiel executée pour le compte d'un usager qui, appliquée à une base cohérente restitue une base cohérente.

Next: Interrogation **Up:** Cohérence des interrogations et **Previous:** Cohérence des interrogations et

Yolaine.Bourda@supelec.fr

Interrogation

Yolaine.Bourda@supelec.fr

Cohérence d'une interrogation

Un utilisateur qui interroge une table (même très grande) est garanti de voir toutes les données telles qu'elles étaient au moment du début de l'interrogation, même si d'autres utilisateurs modifient la table et valident leurs modifications pendant ce temps.

Les sgbd dont oracle utilisent alors le fichier image avant pour assurer cette cohérence. Le `COMMIT` des utilisateurs modifiant la table n'est pas différé à la fin de l'interrogation. **Remarque :** Dès que l'on interroge une table, un verrou est placé sur la définition de la table, c'est à dire qu'un autre utilisateur ne peut pas détruire la table, l'indexer, la mettre en cluster ou modifier sa définition, jusqu'à ce que l'interrogation soit terminée.

Yolaine.Bourda@supelec.fr

Cohérence de plusieurs interrogations successives

Si l'utilisateur désire que l'on ne modifie pas une table pendant une session de travail, celui-ci peut vérouiller la table en mode partagé au moyen de l'ordre sql suivant :

LOCK TABLE nom_table IN SHARE MODE NOWAIT;où l'option `NOWAIT`, qui peut s'adjoindre à toutes les commandes de verrouillage, spécifie que le process qui demande le verrou n'est pas mis en attente si celui-ci n'est pas disponible.

La table n'est alors accessible aux autres utilisateurs qu'en lecture jusqu'à la fin de la transaction de celui qui l'a verrouillée (les autres utilisateurs peuvent aussi verrouiller la table en share mode).

Les modifications des autres utilisateurs seront suspendues.

Yolaine.Bourda@supelec.fr

Mise à jour

Pour s'assurer l'accès exclusif en modification à une table, l'on peut verrouiller cette table en mode exclusif par la commande :

LOCK TABLE nom_table IN EXCLUSIVE MODE NOWAIT; La table n'est alors accessible aux autres utilisateurs qu'en lecture et ils ne peuvent plus la verrouiller en mode exclusif, ni en mode mise à jour partagée, ni en mode partagé jusqu'à la fin de la transaction. Ce verrou est également obtenu automatiquement dès que l'on effectue un update, `INSERT` ou `DELETE` sur une table. C'est le mode par défaut de mise à jour d'une table.

Yolaine.Bourda@supelec.fr

Contrôle des accès à la base et sécurité des données

Yolaine.Bourda@supelec.fr

Introduction

Les systèmes de gestion de bases de données permettent à plusieurs utilisateurs de travailler en toute sécurité sur la même base ou sur des bases différentes.

Chaque donnée peut être définie, soit comme confidentielle et accessible à un seul utilisateur, soit comme étant partageable entre plusieurs utilisateurs.

Les ordres `GRANT` et `REVOKE` du langage sql permettent de définir les droits de chaque utilisateur sur les objets de la base.

Tout utilisateur doit posséder un nom d'utilisateur et un mot de passe pour pouvoir accéder à la base. C'est ce nom d'utilisateur qui déterminera les droits d'accès aux objets de la base.

Yolaine.Bourda@supelec.fr

Droits d'accès aux tables

La protection des objets d'une base de données est décentralisée : c'est le créateur d'un objet qui possède tous les droits de lecture, de modification et de suppression de cet objet. Les autres utilisateurs n'ont aucun droit d'accès à cet objet, à moins que le créateur ne les leur accorde explicitement par une commande `GRANT` :

GRANT privilege ON {nom_table | nom_vue} TO nom_utilisateur WITH GRANT OPTION ;Les privilèges pouvant être accordés sont entre autres les suivants :

[update(col,...)]

- [select] consultation
- [`INSERT`] rajout des lignes
- [`UPDATE(col,...)`] modification des lignes (peut-être restreint à certaines colonnes)
- [`DELETE`] suppression des lignes
- [`ALTER`] modification de la définition de la table
- [`ALL`] tous les droits ci-dessus

Un utilisateur ayant reçu un privilège avec l'option `GRANT` peut le transmettre à son tour (`WITH GRANT OPTION`).

Exemple 10156

L'utilisateur `scott` peut autoriser l'utilisateur `douglas` à lire sa table `emp`.

GRANT SELECT ON emp TO douglas ;Les droits peuvent être accordés à tous les utilisateurs en employant le mot réservé `PUBLIC` à la place du nom d'utilisateur.

GRANT SELECT, UPDATE ON emp TO PUBLIC;Un utilisateur ayant accordé un privilège peut le reprendre à l'aide de l'ordre `REVOKE`.

REVOKE PRIVILèGE ON {nom_table | nom_vue} FROM nom_utilisateur;Le propriétaire d'une table peut donner des droits de lecture non pas sur la table entière, mais sur une vue basée sur la table. Ainsi, seules les informations faisant partie de la vue seront accessibles.

CREATE VIEW emp1 AS SELECT nom, fonction, embauche FROM scott.emp WHERE n_dept != 10 ;

GRANT SELECT ON emp1 TO PUBLIC; Tous les utilisateurs pourront sélectionner les employés à travers la vue `emp1` : ils ne verront que les colonnes `nom`, `fonction`, `embauche` de la table `emp` et n'auront pas accès aux employés du département 10. Avec l'option de contrôle (`CHECK OPTION`), les vues peuvent servir à réaliser les contrôles lors des mises à jour.

Exemple 10157

La vue suivante ne permettrait pas d'insérer dans la table des employés `emp` un employé dont le

numéro de département ne figurerait pas dans la table des départements `dept` :

CREATE VIEW majemp AS SELECT * FROM emp WHERE n_dept IN (SELECT n_dept FROM dept) WITH CHECK OPTION ;Comme il est impossible de faire une mise à jour sur une vue comportant une jointure il faut transformer le critère de jointure en une sous-interrogation.

Pour oracle , le nom complet d'une table ou d'une vue est le nom donné par le créateur, préfixé par le nom du créateur. Par exemple `scott.emp` est le nom complet de la table `emp` créée par l'utilisateur `scott` . Ceci permet à plusieurs utilisateurs de créer des objets de même nom sans qu'il y ait confusion. En revanche, pour accéder à un objet dont on n'est pas le créateur, il faut le désigner par son nom complet incluant le nom du créateur.

---

next | up | previous | contents | index

Next: Stockage des données **Up:** Contrôle des accès à **Previous:** Introduction

Yolaine.Bourda@supelec.fr

Stockage des données

Yolaine.Bourda@supelec.fr

Introduction

La taille des données stockées dans une bases de données va de l'ordre de plusieurs centaines de méga-octets pour des bases moyennes à des dizaines ou des centaines de giga-octets pour des bases importantes. La plus grosse base connue à ce jour atteint le téra-octet.

Ces donnés, bien évidemment, vont être stockées sur un ou plusieurs disques de façon complètement transparente pour l'utilisateur final. Les temps d'accès disques étant très pénalisants, il va falloir essayer de les minimiser.

Il existe deux possibilités liées au stockage des données de minimiser les accès disques :

- les index,
- les clusters.

Yolaine.Bourda@supelec.fr

Les index

Selon le modèle relationnel les sélections peuvent être faites en utilisant le contenu de n'importe quelle colonne et les lignes sont stockées dans n'importe quel ordre.

Considérons le select suivant :

SELECT * FROM emp WHERE nom = 'MARTIN'Un moyen de retrouver la ou les lignes pour lesquelles `nom` est égal à `MARTIN` est de balayer toute la table.

Un tel moyen d'accès conduit à des temps de réponse prohibitifs pour des tables dépassant quelques centaines de lignes.

Une solution offerte par tous les systèmes de gestion de bases de données est la création d'index, qui permettra de satisfaire aux requêtes les plus fréquentes avec des temps de réponse acceptables.

Un index sera matérialisé par la création de blocs disque contenant des couples (valeurs d'index, numéro de bloc) donnant le numéro de bloc disque dans lequel se trouvent les lignes correspondant à chaque valeur d'index.

Yolaine.Bourda@supelec.fr

Utilisation des index

L'adjonction d'un index à une table ralentit les mises à jour (insertion, suppression, modification de la clé) mais accélère beaucoup la recherche d'une ligne dans la table.

L'index accélère la recherche d'une ligne à partir d'une valeur donnée de clé, mais aussi la recherche des lignes ayant une valeur d'index supérieure ou inférieure à une valeur donnée, car les valeurs de clés sont triées dans l'index.

Exemple 10158

Les requêtes suivantes bénéficieront d'un index sur le champ `n_dept`.

SELECT * FROM emp WHERE num = 16034 ; SELECT * FROM emp WHERE num >= 27234 ; SELECT * FROM emp WHERE num BETWEEN 16034 AND 27234;Un index est utilisable même si le critère de recherche est constitué seulement du début de la clé.

Exemple 10159

La requête suivante bénéficiera d'un index sur la colonne `nom`.

SELECT * FROM emp WHERE nom LIKE 'MPar contre si le début de la clé n'est pas connu, l'index est inutilisable.

Exemple 10160

La requête suivante ne bénéficiera pas d'un index sur le champ `nom`.

SELECT * FROM emp WHERE ename LIKE '

Yolaine.Bourda@supelec.fr

Valeurs `NULL`

Elles ne sont pas représentées dans l'index, ceci afin de minimiser le volume nécessaire pour stocker l'index. En contrepartie, l'index ne sera d'aucune utilité pour retrouver les valeurs `NULL` lorsque le critère de recherche est du type `IS NULL`.

Yolaine.Bourda@supelec.fr

Conversions

L'index n'est utilisable que si le critère de sélection est le contenu de la colonne indexée, sans aucune transformation. Par exemple un index sur `salaire` ne sera pas utilisé pour la requête suivante :

SELECT * FROM emp WHERE salaire * 12 > 300000 ;Attention en particulier aux conversions de type qui peuvent empêcher l'utilisation de l'index.

sql est un langage typé, chaque type de données (numérique, caractère, date) ayant ses propres opérateurs, ses propres fonctions et sa propre relation d'ordre. En conséquence, si dans une expression, figurent à la fois un nombre et une chaîne de caractères, sql convertira la chaîne de caractères en nombre. De même si dans une expression, figurent à la fois une chaîne de caractères et une date, sql convertira la chaîne de caractères en date.

Or, dans un prédicat du type :

WHERE fonction(col_indexée) = constantesql ne peut pas utiliser l'index.

Ceci peut se produire , de façon insidieuse, lorsque sql est obligé d'ajouter un appel à une fonction de conversion à cause d'une discordance de type.

Exemple 10161

Le prédicat suivant ne bénéficiera pas d'un index sur le champ `embauche`.

SELECT * FROM emp WHERE embauche LIKE 'En effet, sql est obligé d'effectuer une conversion, et le prédicat qui sera évalué est :

WHERE TO_CHAR(embauche) LIKE 'Le critère de recherche est une fonction de `embauche`, et non le champ `embauche` lui-même, dans ce cas l'index est inutilisable.

Choix des index

Indexer en priorité :

1. les clés primaires
2. les colonnes servant de critère de jointure
3. les colonnes servant souvent de critère de recherche

Ne pas indexer :

1. les colonnes contenant peu de valeurs distinctes (index alors peu efficace)
2. les colonnes fréquemment modifiées

Yolaine.Bourda@supelec.fr

Index comprimé et non comprimé

Les clés dans les index peuvent être comprimées ou non. La compression est une technique permettant de réduire dans des proportions très importantes (d'autant plus que la clé est longue) le volume de l'index.

En contrepartie, il faut parfois un traitement supplémentaire pour recomposer la clé lors des mises à jour de l'index.

Par défaut, les index sont comprimés, les avantages de réduction de taille l'emportant sur les inconvénients dans la plupart des cas.

sql sait exécuter certaines requêtes directement au niveau de l'index sans passer par le segment de données, si l'index est non comprimé et si tous les champs résultats de la requête sont dans l'index.

Exemple 10162

L'index crée par :

CREATE INDEX x ON emp (num, nom) nocompress ;permettra de répondre à la question :

SELECT nom FROM emp WHERE num > 17217 ;sans lire la table puisque toutes les informations se trouvent dans l'index et que l'index est non concaténé.

Yolaine.Bourda@supelec.fr

Index concatene

Un index concaténé est un index portant sur plusieurs colonnes.

Exemple 10163

CREATE INDEX xemp ON (n_dept,num) ;Les index concaténés peuvent être utilisés pour matérialiser une clé composée de plusieurs colonnes.

sql sait utiliser un index concaténé même si le critère de recherche ne porte pas sur toutes les colonnes présentes dans l'index.

Exemple 10164

L'index ci-dessus est utilisable si l'on ne connait que le numéro de département.
SELECT nom FROM emp WHERE n_dept = 20 ;

Yolaine.Bourda@supelec.fr

Les clusters

Yolaine.Bourda@supelec.fr

Buts

Le cluster est une organisation physique des données qui consiste à regrouper physiquement (dans un même bloc disque) les lignes d'une ou plusieurs tables ayant une caractéristique commune (une même valeur dans une ou plusieurs colonnes) constituant la clé du cluster.

La mise en cluster a trois objectifs :

- accélérer la jointure selon la clé de cluster des tables mises en cluster,
- accélérer la sélection des lignes d'une table ayant même valeur de clé, par le fait que ces lignes sont regroupées physiquement,
- économiser de la place, du fait que chaque valeur de la clé du cluster ne sera stockée qu'une seule fois.

Le regroupement en cluster est totalement transparent à l'utilisateur : des tables mises en cluster sont toujours vues comme des tables indépendantes.

Par exemple on pourrait mettre en cluster les tables `emp` et `dept` selon `n_dept`. Ces tables seraient réorganisées de la façon suivante : un bloc de cluster serait créé pour chaque numéro de département, ce bloc contenant à la fois les lignes de la table `emp` et de la table `dept` correspondant à ce numéro de département. La jointure entre les tables `emp` et `dept` selon `n_dept` deviendrait alors beaucoup plus rapide, puisqu'elle serait déjà réalisée dans l'organisation physique des tables.

Pour que l'on puisse mettre une table en cluster il faut que l'une au moins des colonnes faisant partie du cluster soit définie comme obligatoire (`NOT NULL`).

On peut indexer les colonnes d'une table en cluster, y compris les colonnes correspondant à la clé ou à une partie de la clé du cluster. La clé elle-même est automatiquement indexée, on peut éventuellement la réindexer pour créer un index unique servant à contrôler son unicité.

 Yolaine.Bourda@supelec.fr

Le langage sql

Yolaine.Bourda@supelec.fr

SQL : interroger une base

Yolaine.Bourda@supelec.fr

introduction

Ce chapitre expose la partie du langage sql permettant de retrouver des informations stockées dans une base de données. Il s'agit, comme cela a déjà été dit, d'un langage déclaratif dont la syntaxe est très simple (comme beaucoup de langages de ce type) ce qui permet de se concentrer sur le problème à résoudre.

Les exemples cités dans ce chapitre ont tous été testés sous oracle , un des systèmes de gestion de bases de données relationnels les plus répandus sur le marché.

Ces exemples sont bâtis sur une base de données composée des deux relations suivantes :

- emp (nom, num, fonction, n_sup, embauche, salaire, comm, n_dept)

  NOM NUM FONCTION N_SUP EMBAUCHE SALAIRE COMM N_DEPT ---- --- ------- --- ----- ---- --- --- MARTIN 16712 directeur 25717 23-MAY-90 40000 30 DUPONT 17574 administratif 16712 03-MAY-95 9000 30 DUPOND 26691 commercial 27047 04-APR-88 25000 2500 20 LAMBERT 25012 administratif 27047 14-APR-91 12000 20 JOUBERT 25717 president 10-OCT-82 50000 30 LEBRETON 16034 commercial 27047 01-JUN-91 15000 20 MARTIN 17147 commercial 27047 10-DEC-93 20000 500 20 PAQUEL 27546 commercial 27047 03-SEP-83 22000 2000 20 LEFEBVRE 25935 commercial 27047 11-JAN-84 23500 1500 20 GARDARIN 15155 ingenieur 24533 22-MAR-85 24000 10 SIMON 26834 ingenieur 24533 04-OCT-88 20000 10 DELOBEL 16278 ingenieur 24533 16-NOV-94 21000 10 ADIBA 25067 ingenieur 24533 05-OCT-87 30000 10 CODD 24533 directeur 25717 12-SEP-75 55000 10 LAMERE 27047 directeur 25717 07-SEP-89 45000 20 BALIN 17232 administratif 24533 03-OCT-87 13500 10 BARA 24831 administratif 16712 10-SEP-88 15000 30

- dept(n_dept, nom,lieu)

  N_DEPT NOM LIEU --- ------- ------- 10 recherche Rennes 20 vente Metz 30 direction Gif 40 fabrication Toulon

La commande select constitue, à elle seuls, le langage permettant d'interroger une base de données. Elle permet :

- de sélectionner certaines colonnes d'une table : c'est l'opération de *projection* ;
- de sélectionner certaines lignes d'une table en fonction de leur contenu : c'est l'opération de *restriction* ;
- de combiner des informations venant de plusieurs tables : ce sont les opérations de *jointure* , *union* , *intersection* , *différence relationnelle* ;
- de combiner entre elles ces différentes opérations.

Une interrogation, on parle plutôt de requête, est une combinaison d'opérations portant sur des tables (relations) et dont le résultat est lui-même une table dont l'existence est éphémère (le temps de la requête).

On peut introduire un commentaire à l'intérieur d'une commande sql en l'encadrant par /* */ .

Next: Interroger simplement une base **Up:** SQL : interroger une **Previous:** SQL : interroger une

Yolaine.Bourda@supelec.fr

Index

Yolaine.Bourda@supelec.fr

Interroger simplement une base

Yolaine.Bourda@supelec.fr

Sélection de colonnes ou projection

La commande select le plus simple a la syntaxe suivante :

SELECT * FROM nom_table ;Dans laquelle :

- `nom_table` : est le nom de la table sur laquelle porte la sélection.
- `*` : signifie que toutes les colonnes de la table sont sélectionnées.

Par défaut toutes les lignes sont sélectionnées. On peut limiter la sélection à certaines colonnes, en indiquant une liste de noms de colonnes à la place de l'astérisque.

SELECT nom_col1, nom_col2, ... FROM nom_table ;

Exemple 10165

Donner le nom et la fonction de chaque employé.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=2 COLS=50 SELECT nom, fonction FROM emp; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM La clause `DISTINCT` ajoutée derrière la commande select permet d'éliminer les duplications.

Exemple 10166

Quelles sont toutes les fonctions différentes.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=2 COLS=50 SELECT DISTINCT fonction FROM emp; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Yolaine.Bourda@supelec.fr

Sélection de lignes ou restriction

La clause `WHERE` permet de spécifier quelles sont les lignes à sélectionner. Elle est suivie d'un prédicat qui sera évalué pour chaque ligne de la table. Les lignes pour lesquelles le prédicat est vrai seront sélectionnées.

La syntaxe est la suivante :

SELECT * FROM nom_table WHERE predicat ;Un prédicat n'est ni plus ni moins que la façon dont on exprime une propriété. Les prédicats, qu'ils soient simples ou composés, sont constitués à partir d'expressions que l'on compare entre elles.

Yolaine.Bourda@supelec.fr

Expression simple

Une expression simple peut être :

- une variable désignée par un nom de colonne,
- une constante.

Les expressions peuvent être de trois types : numérique, chaîne de caractères ou date. A chacun de ces types correspond un format de constante :

- [Constante numérique] nombre contenant éventuellement un signe, un point décimal et une puissance de dix. Ex : -10, 2.5, 1.2 E-10
- [Constante chaîne de caractères] une chaîne de caractères entre apostrophes. Ex :'MARTIN' (Attention, une lettre en majuscules n'est pas considérée comme égale à la même lettre en minuscule).
- [Constante date] une chaîne de caractères entre apostrophes au format suivant : jour-mois-année où le jour est sur deux chiffres, le mois est désigné par les trois premières lettres de son nom en anglais, l'année est sur deux chiffres. Ex : '01-FEB-85'

On peut, en sql, exprimer des expressions plus complexes en utilisant des opérateurs et des fonctions étudiés ici.

Yolaine.Bourda@supelec.fr

Prédicat simple

Un prédicat simple est le résultat de la comparaison de deux expressions au moyen d'un opérateur de comparaison qui peut être :

[>=]

- [=] égal
- [!=] différent
- [<] inférieur
- [<=] inférieur ou égal
- [>] supérieur
- [>=] supérieur ou égal

Les trois types d'expressions peuvent être comparés au moyen de ces opérateurs :

- Pour les types date, la relation d'ordre est l'ordre chronologique.
- Pour les types caractère, la relation d'ordre est l'ordre alphabétique.

Il faut ajouter à ces opérateurs arithmétiques classiques les opérateurs suivants :

- [`expr1 BERTWEEN expr2 AND expr3`]
  vrai si `expr1` est compris entre `expr2` et `expr3`, bornes incluses
- [`expr1 IN (expr2, expr3, ...)`]
  vrai si `expr1` est égale à l'une des expressions de la liste entre parenthèses
- [`expr LIKE chaine`]
  où `chaine` est une chaîne de caractères pouvant contenir l'un des caractères jokers :

  ['%']

  - [_] remplace exactement 1 caractère
  - [%] remplace une chaîne de caractères de longueur quelconque, y compris de longueur nulle.

Exemple 10174

Quels sont les employés dont la commission est supérieure au salaire?

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=3 COLS=50 SELECT nom, salaire, comm FROM emp WHERE comm > salaire; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Exemple 10175

Quels sont les employés gagnant entre 20000 et 25000?

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=3 COLS=50 SELECT nom, salaire FROM emp WHERE salaire BETWEEN 20000 AND 25000; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Exemple 10176

Quels sont les employés commerciaux ou ingénieurs?

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=3 COLS=50 SELECT num, nom, fonction, salaire FROM emp WHERE fonction IN ('commercial','ingenieur'); /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Exemple 10177

Quels sont les employés dont le nom commence par M?

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=3 COLS=50 SELECT nom FROM emp WHERE nom LIKE 'M/TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

---

next up previous contents index

Next: Prédicats composés **Up:** Sélection de lignes ou **Previous:** Expression simple

Yolaine.Bourda@supelec.fr

Prédicats composés

Les opérateurs logiques `AND` (et) et `OR` (ou inclusif) peuvent être utilisés pour combiner entre eux plusieurs prédicats. L'opérateur `NOT` placé devant un prédicat en inverse le sens.

L'opérateur `AND` est prioritaire par rapport à l'opérateur `OR`. Des parenthèses peuvent être utilisées pour imposer une priorité dans l'évaluation du prédicat, ou simplement pour rendre plus claire l'expression logique.

Exemple 10178

Quels sont les employés du département 30 ayant un salaire supérieur à 25000?

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=4 COLS=50 SELECT nom FROM emp WHERE n_dept = 30 AND salaire > 25000; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Exemple 10179

Quels sont les employés directeurs, ou commerciaux et travaillant dans le département 10?

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=3 COLS=50 SELECT nom, fonction, salaire, n_dept FROM emp WHERE fonction = 'directeur' OR (fonction = 'commercial' AND n_dept = 10); /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM La requête précédente donnerait le même résultat sans les parenthèses, résultat différent de celui du select suivant.

Exemple 10180

Quels sont les employés directeurs ou commerciaux, et travaillant dans le département 10?

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=3 COLS=50 SELECT num, nom, fonction, n_dept FROM emp WHERE (fonction='directeur' OR fonction = 'commercial') AND n_dept = 10; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Yolaine.Bourda@supelec.fr

next up previous contents index

Next: Nom de colonne **Up:** Interroger simplement une base **Previous:** Prédicats composés

Valeurs `NULL`

Pour sql, une valeur `NULL` est une valeur non définie. Il est possible d'ajouter une ligne à une table sans spécifier de valeur pour les colonnes non obligatoires : ces colonnes absentes auront la valeur `NULL`.

Par exemple les employés dont la rémunération ne prend pas en compte de commission auront une valeur `NULL`, c'est-à-dire indéfinie, comme commission.

L'opérateur `IS NULL` permet de tester la valeur `NULL` : le prédicat `expr IS NULL` est vrai si l'expression a la valeur `NULL` (c'est-à-dire si elle est indéfinie).

Exemple 10181

Quels sont les employés dont la commission a la valeur `NULL`?

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=3 COLS=50 SELECT nom FROM emp WHERE comm IS NULL; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM L'opérateur `IS NOT NULL` permet de construire un prédicat vrai si la valeur n'est pas `NULL` (et donc le prédicat `expr IS NOT NULL` est vrai si `expr` est définie) **Remarques**

- Le prédicat `expr = NULL` est toujours faux, et ne permet donc pas de tester si l'expression a la valeur `NULL`.
- La valeur `NULL` est différente de la valeur zéro qui, elle, qui est une valeur bien définie.
- Une expression de la forme `NULL + val` donne `NULL` comme résultat quelle que puisse être la valeur de `val`.

---

next up previous contents index

Next: Nom de colonne **Up:** Interroger simplement une base **Previous:** Prédicats composés

Yolaine.Bourda@supelec.fr

Nom de colonne

Les colonnes constituant le résultat d'un select peuvent être renommées dans le select , ceci est utile en particulier lorsque la colonne résultat est une expression. Pour cela, il suffit de faire suivre l'expression définissant la colonne d'un nom, selon les règles suivantes :

- le nom (30 charactères maximum) est inséré derrière l'expression définissant la colonne, séparé de cette dernière par un espace.
- si le nom contient des séparateurs (espace, caractère spécial), ou s'il est identique à un mot clé de sql (ex : `DATE`), il doit être mis entre guillemets "".

Ce nom est celui sous lequel la colonne sera connue des interfaces externes. Sous `SQLPLUS`, par exemple, il constituera le titre par défaut de la colonne, et servira de référence pour définir un format pour la colonne.

Exemple 10182

Salaire de chaque employé.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=2 COLS=50 SELECT nom, salaire "SALAIRE MENSUEL" FROM emp; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM**Remarque :** Attention, ce nom n'est pas connu à l'intérieur du select .

Yolaine.Bourda@supelec.fr

Classer le résultat d'une interrogation

Les lignes constituant le résultat d'un select sont obtenues dans un ordre indéterminé. On peut, dans un select , demander que le résultat soit classé dans un ordre ascendant ou descendant, en fonction du contenu d'une ou plusieurs colonnes (jusqu'à 16 critères de classement possibles). Les critères de classement sont spécifiés dans une clause `ORDER BY` dont la syntaxe est la suivante :

ORDER BY {nom_col1 | num_col1 [DESC] [, nom_col2 | num_col2 [DESC],...]}Le classement se fait d'abord selon la première colonne spécifiée dans l'`ORDER BY` puis les lignes ayant la même valeur dans la première colonne sont classées selon la deuxième colonne de l'`ORDER BY`, etc... Pour chaque colonne, le classement peut être ascendant (par défaut) ou descendant (`DESC`). L'`ORDER BY` peut faire référence à une colonne par son nom ou par sa position dans la liste des colonnes présentes derrière le select (la première colonne sélectionnée a le numéro 1, la deuxième a le numéro 2, ...).

Exemple 10183

Donner tous les employés classés par fonction, et pour chaque fonction classés par salaire décroissant

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=3 COLS=50 SELECT nom, fonction, salaire FROM emp ORDER BY fonction, salaire DESC; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM**Remarque :** Dans un classement les valeurs `NULL` sont toujours en tête quel que soit l'ordre du classement (ascendant ou descendant).

 Yolaine.Bourda@supelec.fr

les jointures

La jointure est une opération permettant de combiner des informations venant de plusieurs tables. Les exemples suivants se limiteront à deux tables, mais on peut joindre jusqu'à 256 tables. Une jointure se formule simplement en spécifiant plusieurs tables derrière le `FROM` de la façon suivante :

SELECT ... FROM nom_table1, nom_table2... WHERE predicat;Si on ne précise pas de condition de sélection, le résultat obtenu sera le produit cartésien des tables présentes derrière le `FROM` (résultat non souhaité en général).

Il n'existe pas d'associations implicites ou explicites entre les tables dans sql. Les associations entre les tables sont définies dynamiquement lors des interrogations, ce qui contribue à la grande souplesse du langage sql et rend possible toute association même si elle n'a pas été prévue lors de la définition et du chargement de la base.

Yolaine.Bourda@supelec.fr

Equi-jointure

Le rapprochement de chaque ligne de la table `emp` avec la ligne de la table `dept` ayant même numéro de département permet d'obtenir la liste des employés avec la localité dans laquelle ils travaillent. Ce rapprochement entre deux colonnes appartenant à deux tables différentes mais ayant le même sens (ici le numéro de département) et venant vraisemblablement d'une relation 1-n lors de la conception (ici 1 entité département pour n entités employés) est assez naturel. C'est pourquoi ce type de jointure porte le nom de jointure naturelle ou d'équi-jointure.

Exemple 10184

Donner pour chaque employé son nom et son lieu de travail.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=3 COLS=50 SELECT emp.nom, lieu FROM emp, dept WHERE emp.n_dept = dept.n_dept; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM Le fait que la colonne contenant le numéro de département ait le même nom dans les deux tables a rendu nécessaire le préfixage par le nom de table dans le critère de jointure (clause `WHERE`). Le nom de colonne `nom` a lui aussi besoin d'être préfixé car il appartient aux deux tables (nom de la personne dans l'une et nom du département dans l'autre). Par contre le nom de colonne `loc` n'a pas besoin d'être préfixé car il n'y a pas d'ambiguïté sur la table à laquelle cette colonne appartient.

Jointure d'une table à elle-même

Il peut être utile de rassembler des informations venant d'une ligne d'une table avec des informations venant d'une autre ligne de la même table.

Exemple 10185

Donner pour chaque employé le nom de son supérieur hiérarchique.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=3 COLS=50 SELECT emp.nom, mgr.nom FROM emp, emp mgr WHERE emp.n_sup= mgr.num; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM**Remarque :** Dans ce cas, il faut impérativement renommer au moins l'une des deux occurences de la table (ici `emp`) en lui donnant un synonyme, afin de pouvoir préfixer sans ambiguité chaque nom de colonne.

Yolaine.Bourda@supelec.fr

Autres jointures

Le critère d'égalité est le critère de jointure le plus naturel. Mais on peut utiliser d'autres types de comparaisons comme critères de jointures.

Exemple 10186

Quels sont les employés gagnant plus que `SIMON`?

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=4 COLS=50 SELECT emp.nom, emp.salaire, emp.fonction FROM emp, emp j WHERE emp.salaire > j.salaire AND J.nom = 'SIMON'; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Yolaine.Bourda@supelec.fr

Jointure externe

Lorsqu'une ligne d'une table figurant dans une jointure n'a pas de correspondant dans les autres tables, elle ne satisfait pas au critère d'équi-jointure et donc ne figure pas dans le résultat de la jointure.

Une option permet de faire figurer dans le résultat les lignes satisfaisant la condition d'équi-jointure plus celles n'ayant pas de correspondant. Cette option s'obtient en accolant (+) au nom de colonne de la table dans laquelle manquent des éléments, dans la condition d'équi-jointure.

Exemple 10187

Le département 40 ne figurait pas dans le résultat du select précédent. Par contre, il figurera dans le résultat du select suivant.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=3 COLS=50 SELECT emp.nom, lieu FROM emp, dept WHERE emp.n_dept(+) = dept.n_dept; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM Le (+) peut s'interpréter comme l'ajout d'une ligne fictive dont toutes les colonnes ont la valeur `NULL`, et qui réalise la correspondance avec les lignes de l'autre table qui n'ont pas de correspondant réel. Dans l'exemple ci-dessus, la valeur de `nom` associée au département 40 est la valeur `NULL`.

Exemple 10188

Retrouver les départements n'ayant aucun employé.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=4 COLS=50 SELECT dept.n_dept, emp.nom FROM emp,dept WHERE dept.n_dept = emp.n_dept (+) AND emp.nom IS NULL; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Yolaine.Bourda@supelec.fr

Les opérateurs ensemblistes

Les opérateurs ensemblistes permettent de ``joindre'' des tables verticalement c'est-à-dire de combiner dans un résultat unique des lignes provenant de deux interrogations. Les lignes peuvent venir de tables différentes mais après projection on doit obtenir des tables ayant même schéma de relation.

Les opérateurs ensemblistes sont les suivants :

- l'union : `UNION` [union]`UNION`
- l'intersection : `INTERSECT` [intersect]`INTERSECT`
- la différence relationnelle : `MINUS` [minus]`MINUS`

La syntaxe d'utilisation est la même pour ces trois opérateurs :

SELECT ... {UNION | INTERSECT | MINUS } SELECT ...Dans une requête utilisant des opérateurs ensemblistes :

- Tous les select doivent avoir le même nombre de colonnes sélectionnées, et leur types doivent être un à un identiques. Les conversions éventuelles doivent être faites à l'intérieur du select à l'aide des fonctions de conversion.
- Les doubles sont éliminés (`DISTINCT` implicite).
- Les noms de colonnes (titres) sont ceux du premier select .
- la largeur des colonnes est la plus grande parmi tous les select .
- Dans une requête on ne peut trouver qu'un seul `ORDER BY`. S'il est présent, il doit être mis dans le dernier select et il ne peut faire référence qu'aux numéros des colonnes et non pas à leurs noms (car les noms peuvent être différents dans chacune des interrogations).

L'on peut combiner le résultat de plus de deux select au moyen des opérateurs `UNION`, `INTERSECT`, `MINUS`.

SELECT ... UNION SELECT ... MINUS SELECT ...Dans ce cas l'expresion est évaluée de gauche à droite, mais on peut modifier l'ordre d'évaluation en utilisant des parenthèses.

SELECT ... UNION (SELECT ... MINUS SELECT ...)

Yolaine.Bourda@supelec.fr

next | up | previous | contents | index

Next: Sous-interrogation ramenant une seule **Up:** SQL : interroger une **Previous:** Les opérateurs ensemblistes

Les sous-interrogations

Une caractéristique puissante de sql est la possibilité qu'un critère de recherche employé dans une clause `WHERE` (expression à droite d'un opérateur de comparaison) soit lui-même le résultat d'un select ; c'est ce qu'on appelle une sous-interrogation.

Yolaine.Bourda@supelec.fr

Sous-interrogation ramenant une seule valeur

Exemple 10189

Quels sont les employés ayant la même fonction que codd?

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=5 COLS=50 SELECT nom FROM emp WHERE fonction = (SELECT fonction FROM emp WHERE nom ='CODD'); /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM**Remarques**

- une sous-interrogation qui ne ramène aucune ligne se termine avec un code d'erreur.
- une sous-interrogation ramenant plusieurs lignes provoquera aussi, dans ce cas, une erreur (pour traiter correctement ce cas, voir paragraphe ci-dessous)

Yolaine.Bourda@supelec.fr

Sous-interrogation ramenant plusieurs lignes

Une sous-interrogation peut ramener plusieurs lignes à condition que l'opérateur de comparaison admette à sa droite un ensemble de valeurs. Les opérateurs permettant de comparer une valeur à un ensemble de valeurs sont :

- l'opérateur `IN`
- les opérateurs obtenus en ajoutant `ANY` ou `ALL` à la suite d'un opérateur de comparaison classique (=, !=, >, >=, <, <=)
  - `ANY`: la comparaison est vraie si elle est vraie pour au moins un des éléments de l'ensemble.
  - `ALL`: la comparaison sera vraie si elle est vraie pour tous les éléments de l'ensemble.

Exemple 10196

Quels sont les employés gagnant plus que tous les employés du département 30.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=5 COLS=50 SELECT nom, salaire FROM emp WHERE salaire > ALL (SELECT salaire FROM emp WHERE n_dept = 20); /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Yolaine.Bourda@supelec.fr

Sous-interrogation ramenant plusieurs colonnes

Il est possible de comparer le résultat d'un select ramenant plusieurs colonnes à une liste de colonnes. La liste de colonnes figurera entre parenthèses à gauche de l'opérateur de comparaison.

Exemple 10197

Quels sont les employés ayant même fonction et même supérieur que `CODD`?

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=5 COLS=50 SELECT nom, fonction, n_sup FROM emp WHERE (fonction, n_sup) = (SELECT fonction, n_sup FROM emp WHERE nom = 'CODD'); /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Yolaine.Bourda@supelec.fr

Sous-interrogation synchronisée avec l'interrogation principale

Dans les exemples précédents, la sous-interrogation était évaluée d'abord, puis le résultat pouvait être utilisé pour exécuter l'interrogation principale. sql sait également traiter une sous-interrogation faisant référence à une colonne de la table de l'interrogation principale. Le traitement dans ce cas est plus complexe car il faut évaluer la sous-interrogation pour chaque ligne de l'interrogation principale.

Exemple 10198

Quels sont les employés ne travaillant pas dans le même département que leur supérieur hiérarchique.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=6 COLS=50 SELECT nom FROM emp e WHERE n_dept != (SELECT n_dept FROM emp WHERE e.n_sup = num) AND n_sup IS NOT NULL; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM Il a fallu ici renommer la table `emp` de l'interrogation principale pour pouvoir la référencer dans la sous-interrogation. `n_sup IS NOT NULL` est nécessaire car dans le cas de `JOUBERT` la colonne `n_sup` est `NULL` et la sous-requête ne ramène alors aucune valeur.

Yolaine.Bourda@supelec.fr

Sous-interrogation ramenant au moins une ligne

L'opérateur `EXISTS` permet de construire un prédicat vrai si la sous-interrogation qui suit ramène au moins une ligne.

Exemple 10199

Quels sont les employés travaillant dans un département qui a procédé à des embauches depuis le début de l'année 94.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=6 COLS=50 SELECT * FROM emp e WHERE EXISTS (SELECT * FROM emp WHERE embauche >= '01-jan-94' AND n_dept = e.n_dept); /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM**Remarque :** On peut inverser le sens de l'opérateur `EXISTS` en le faisant précéder de `NOT`.

Yolaine.Bourda@supelec.fr

Sous-interrogations multiples

Un select peut comporter plusieurs sous-interrogations, soit imbriquées, soit au même niveau dans différents prédicats combinés par des `AND` ou des `OR`.

Exemple 10200

Liste des employés du département 10 ayant même fonction que quelqu'un du département de `DUPONT`.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=8 COLS=50 SELECT nom, fonction FROM emp WHERE n_dept = 10 AND fonction IN (SELECT fonction FROM emp WHERE n_dept = (SELECT n_dept FROM emp WHERE nom = 'DUPONT')); /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Yolaine.Bourda@supelec.fr

Les expressions et fonctions

Une expression est un ensemble de variables (contenu d'une colonne), de constantes et de fonctions combinées au moyen d'opérateurs. Les fonctions prennent une valeur dépendant de leurs arguments qui peuvent être eux-mêmes des expressions.

Les expressions peuvent figurer :

- en tant que colonne résultat d'un select ,
- dans une clause `WHERE`,
- dans une clause `ORDER BY`.

Il existe trois types d'expressions correspondant chacun à un type de données de sql : arithmétique, chaîne de caractère, date. A chaque type correspondent des opérateurs et des fonctions spécifiques.

sql autorise les mélanges de types dans les expressions et effectuera les conversions nécessaires : dans une expression mélangeant dates et chaînes de caractères, les chaînes de caractères seront converties en dates, dans une expression mélangeant nombres et chaînes de caractères, les chaînes de caractères seront converties en nombre.

Yolaine.Bourda@supelec.fr

next up previous contents index

Next: Opérateurs arithmétiques **Up:** Les expressions et fonctions **Previous:** Les expressions et fonctions

Expressions et fonctions arithmétiques

Une expression arithmétique peut contenir :

- des noms de colonnes
- des constantes
- des fonctions arithmétiques

combinés au moyen des opérateurs arithmétiques.

Yolaine.Bourda@supelec.fr

Opérateurs arithmétiques

Les opérateurs arithmétiques présents dans sql sont les suivants :

- + addition ou + unaire
- - soustraction ou - unaire
- * multiplication
- / division

Remarque : la division par 0 provoque une fin avec code d'erreur.

Yolaine.Bourda@supelec.fr

Priorité des opérateurs

Une expression arithmétique peut comporter plusieurs opérateurs. Dans ce cas, le résultat de l'expression peut varier selon l'ordre dans lequel sont effectuées les opérations. Les opérateurs de multiplication et de division sont prioritaires par rapport aux opérateurs d'addition et de soustraction. Des parenthèses peuvent être utilisées pour forcer l'évaluation de l'expression dans un ordre différent de celui découlant de la priorité des opérateurs.

Exemple 10201

Donner pour chaque commercial son revenu (salaire + commission).

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=3 COLS=50 SELECT nom, salaire+comm FROM emp WHERE fonction = 'commercial'; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Exemple 10202

Donner la liste des commerciaux classée par commission sur salaire décroissant.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=4 COLS=50 SELECT nom, comm/salaire, comm, salaire FROM emp WHERE fonction = 'commercial' ORDER BY comm/salaire DESC; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Exemple 10203

Donner la liste des employés dont la commission est inférieure à 5% du salaire.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=3 COLS=50 SELECT nom, salaire, comm FROM emp WHERE comm <= salaire *.05; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Yolaine.Bourda@supelec.fr

Fonctions arithmétiques

Dans ce paragraphe, ont été regroupées les fonctions ayant un ou plusieurs nombres comme arguments, et renvoyant une valeur numérique.

[ROUND(n ,m)]

- [ABS(nb)] [abs]ABS fct Renvoie la valeur absolue de nb.
- [CEIL(nb)] [ceil]CEIL fct Renvoie le plus petit entier supérieur ou égal à nb.
- [COS(n)] [cos]COS fct Renvoie le cosinus de n, n étant un angle exprimé en radians.
- [COSH(n)] [cosh]COSH fct Renvoie le cosinus hyperbolique de n.
- [EXP(n)] [exp]EXP fct Renvoie e puissance n.
- [FLOOR(nb)] [floor]FLOOR fct Renvoie le plus grand entier inférieur ou égal à nb.
- [LN(n)] [ln]LN fct Renvoie le logarithme népérien de n qui doit être un entier strictement positif.
- [LOG(m,n)] [log]LOG fct Renvoie le logarithme en base m de n. m doit être un entier strictement supérieur à 1, et n un entier strictement positif.
- [MOD(m,n)] [mod]MOD fct Renvoie le reste de la division entière de m par n, si n vaut 0 alors renvoie m. Attention, utilisée avec au moins un de ses arguments négatifs, cette fonction donne des résultats qui peuvent être différents d'un modulo classique. Cette fonction ne donne pas toujours un résultat dont le signe du diviseur.
- [POWER(m,n)] [power]POWER fct Renvoie m puissance n, m et n peuvent être des nombres quelconques entiers ou réels mais si m est négatif n doit être un entier.
- [ROUND(n[,m])] [round]ROUND fct Si m est positif, renvoie n arrondi (et non pas tronqué) à m chiffres après la virgule. Si m est négatif, renvoie n arrondi à m chiffres avant la virgule. m doit être un entier et il vaut 0 par défaut.
- [SIGN(nb)] [sign]SIGN fct renvoie -1 si nb est négatif, 0 si nb est nul, 1 si nb est positif.
- [SIN(n)] [sin]SIN fct Renvoie le sinus de n, n étant un angle exprimé en radians.
- [SINH(n)] [sinh]SINH fct Renvoie le sinus hyperbolique de n.
- [SQRT(nb)] [sqrt]SQRT fct Renvoie la racine carrée de nb qui doit être un entier positif ou nul.
- [TAN(n)] [tan]TAN fct Renvoie la tangente de n, n étant un angle exprimé en radians.
- [TANH(n)] [tanh]tanh fct Renvoie la tangente hyperbolique de n.
- [TRUNC(n[,m])] [trunc]TRUNC fct Si m est positif, renvoie n arrondi tronqué à m chiffres après la virgule. Si m est négatif, renvoie ntronqué à m chiffres avant la virgule. m doit être un entier et il vaut 0 par défaut.

Exemple 10204

Donner pour chaque employé son salaire journalier.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=2 COLS=50 SELECT nom, ROUND(salaire/22,2) FROM emp; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

---

next up previous contents index

Next: Expressions et fonctions sur **Up:** Expressions et fonctions arithmétiques **Previous:** Priorité des opérateurs Yolaine.Bourda@supelec.fr

next up previous contents index

Next: Opérateur sur les chaînes **Up:** Les expressions et fonctions **Previous:** Fonctions arithmétiques

Expressions et fonctions sur les chaînes de caractères

Yolaine.Bourda@supelec.fr

next up previous contents index

Next: Fonctions sur les chaînes **Up:** Expressions et fonctions sur **Previous:** Expressions et fonctions sur

Opérateur sur les chaînes de caractères

Il existe un seul opérateur sur les chaînes de caractères : la concaténation. Cet opérateur se note au moyen de deux caractères (barre verticale) accolés. Le résultat d'une concaténation est une chaîne de caractères obtenue en écrivant d'abord la chaîne à gauche de puis celle à droite de . FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=2 COLS=50 SELECT nom || '/' || fonction FROM emp; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Yolaine.Bourda@supelec.fr

Fonctions sur les chaînes de caractères

Le paragraphe suivant contient les fonctions travaillant sur les chaînes de caractères et renvoyant des chaînes de caractères.

- [`CONCAT(chaîne1,chaîne2)`] [concat]`CONCAT` fct
  Renvoie la chaîne obtenue en concaténant `chaîne1` à `chaîne2`. Cette fonction est équivalente à l'opérateur de concaténation .
- [`INITCAP(chaîne)`] [initcap]`INITCAP` fct
  Renvoie `chaîne` en ayant mis la première lettre de chaque mot en majuscule et toutes les autres en minuscule. Les séparateurs de mots sont les espaces et les caractères non alphanumériques.
- [`LOWER(chaîne)`] [lower]`LOWER` fct
  Renvoie `chaîne` en ayant mis toutes ses lettres en minuscules.
- [`LPAD(chaîne,long,[char])`] [lpad]`LPAD` fct
  Renvoie la chaîne obtenue en complétant, ou en tronquant, `chaîne` pour qu'elle ait comme longueur `long` en ajoutant éventuellement à gauche le caractère (ou la chaîne de caractères) `char`. La valeur par défaut de `char` est un espace.
- [`LTRIM(chaîne[,ens])`] [ltrim]`LTRIM` fct
  Renvoie la chaîne obtenue en parcourant à partir de la gauche `chaîne` et en supprimant tous les caractères qui sont dans `ens`. On s'arrête quand on trouve un caractère qui n'est pas dans `ens`. La valeur de defaut de `ens` est un espace.
- [`REPLACE(chaine,avant,après`] [replace]`REPLACE` fct
  Renvoie `chaine` dans laquelle toutes les occurrences de la chaîne de caractères `avant` ont été remplacés par la chaîne de caractères`après`.
- [`RPAD(chaîne,n,[char])`] [rpad]`RPAD` fct
  Renvoie la chaîne obtenue en complétant, ou en tronquant, `chaîne` pour qu'elle ait comme longueur `long` en ajoutant éventuellement à droite le caractère (ou la chaîne de caractères) `char`. La valeur par défaut de `char` est un espace.
- [`RTRIM(chaîne[,ens])`] [rtrim]`RTRIM` fct
  Renvoie la chaîne obtenue en parcourant à partir de la droite `chaîne` et en supprimant tous les caractères qui sont dans `ens`. On s'arrête quand on trouve un caractère qui n'est pas dans `ens`. La valeur de defaut de `ens` est un espace.
- [`SOUNDEX(chaîne)`] [soundex]`SOUNDEX` fct
  Renvoie la chaîne de caratères constituée de la représentation phonétique des mots de `chaîne`.
- [`SUBSTR(chaîne,m[,n])`] [substr]`SUBSTR` fct
  Renvoie la partie de `chaîne` commençant au caractère `m` et ayant une longueur de `n`.
- [`TRANSLATE(chaîne, avant, après)`] [translate]`TRANSLATE` fct
  Renvoie une chaîne de caratères en remplaçant chaque caratère de `chaîne` présent dans `avant` par le caractère situé à la même position dans `après`. Les caratères de `chaîne` non présents dans `avant` ne sont pas modifiés. `avant` peut contenir plus de caractères que `apres`, dans ce cas les caratères de `avant` sans correspondants dans `apres` seront supprimés de `chaîne` .

- [`UPPER(chaîne)`] [upper]`UPPER` fct
  Renvoie `chaîne` en ayant mis toutes ses lettres en majuscules.

Le paragraphe suivant contient les fonctions travaillant sur les chaînes de caractères et renvoyant des entiers.

- [`INSTR(chaîne, sous-chaîne, debut, occ)`] [instr]`INSTR` fct
  Renvoie la position du premier caractère de `chaîne` correspondant à l'occurence `occ` de `sous-chaîne` en commençant la recherche à la position `début`.
- [`LENGTH(chaîne)`] [length]`LENGTH` fct
  renvoie la longueur de `chaîne`, exprimée en nombre de caractères.

---

next | up | previous | contents | index

Next: Expressions et fonctions sur **Up:** Expressions et fonctions sur **Previous:** Opérateur sur les chaînes Yolaine.Bourda@supelec.fr

Expressions et fonctions sur les dates

Yolaine.Bourda@supelec.fr

next up previous contents index

Next: Fonctions sur les dates **Up:** Expressions et fonctions sur **Previous:** Expressions et fonctions sur

Opérateurs sur les dates

Au moyen des opérateurs arithmétiques + et - il est possible de construire les expressions suivantes :

- `date +/- nombre` : le resultat est une date obtenue en ajoutant le nombre de jours `nombre` à la date `date`.
- `date2 - date1` : le resultat est le nombre de jours entre les deux dates.

Yolaine.Bourda@supelec.fr

Fonctions sur les dates

- [ADD_MONTHS(date,n)] [add_months]ADD_MONTHS fct
  Renvoie la date obtenue en ajoutant n mois à date. n peut être un entier quelconque. Si le mois obtenu a moins de jours que le jour de date, le jour obtenu est le dernier du mois.
- [LAST_DAY(date)] [last_day]LAST_DAY fct
  Renvoie la date du dernier jour du mois de date.
- [MONTHS_BETWEEN(date2, date1)] [months_between]MONTHS_BETWEEN fct
  Renvoie le nombre de mois entre date2 et date1, si date2 est après date1 le résultat est positif, sinon le résultat est négatif. Si les jours date2 et date1 sont les mêmes, ou si ce sont les derniers jours du mois, le résultat est un entier. La partie fractionnaire est calculée en considérant chaque jour comme 1/31ème de mois
- [NEXT_DAY(date,nom_du_jour)] [next_day]NEXT_DAY fct
  Renvoie la date du prochain jour de la semaine dont le nom est nom_de_jour.
- [ROUND(date[,précision])] [round]ROUND fct
  Renvoie date arrondie à l'unité spécifiée dans précision. L'unité de précision est indiquée en utilisant un des masques de mise en forme de la date. On peut ainsi arrondir une date à l'année, au mois, à la minute,... Par défaut la précision est le jour.
- [SYSDATE] [sysdate]SYSDATE fct
  Renvoie la date et l'heure courantes du système d'exploitation hote.
- [TRUNC(date[,précision])] [trunc]TRUNC fct
  Renvoie date tronquée à l'unité spécifiée dans précision. Les paramètres sont analogues à ceux de la fonction ROUND.

Exemple 10205

Donner la date du lundi suivant l'embauche de chaque employé.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=2 COLS=50 SELECT NEXT_DAY (embauche,'MONDAY') FROM emp; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Exemple 10206

Donner la date date d'embauche de chaque employé arrondie à l'année.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=2 COLS=50 SELECT ROUND (embauche,'Y') FROM emp; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Exemple 10207

Donner pour chauqe employé le nombre de jours depuis son embauche.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=2 COLS=50 SELECT ROUND (SYSDATE-embauche) FROM emp; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

---

next up previous contents index

Next: Fonctions de conversion **Up:** Expressions et fonctions sur **Previous:** Opérateurs sur les dates

Yolaine.Bourda@supelec.fr

Fonctions de conversion

- [ASCII(chaine)] [ascii]ASCII fct
  Renvoie le nombre correspondant au code ascii du premier caractère de chaine.
- [CHR(nombre)] [chr]CHR fct
  Renvoie le caractère dont nombre est le code ascii.
- [TO_CHAR(nombre,format)] [to_char]TO_CHAR fct
  Renvoie la chaîne de caratères en obtenue en convertissant nombre en fonction de format. format est une chaîne de caractères pouvant contenir les caractères suivants :

  [EEEE]
  - [9] représente un chiffre (non représenté si non significatif)
  - [0] représente un chiffre (représenté même si non significatif)
  - [.] point décimal apparent
  - [V] définit la position du point décimal non apparent
  - [,] une virgule apparaitra à cet endroit
  - [$] un $ précèdera le premier chiffre significatif
  - [B] le nombre sera représenté par des blancs s'il vaut 0
  - [EEEE] le nombre sera représenté avec un exposant (le spécifier avantMI ou PR)
  - [MI] le signe négatif sera à droite
  - [PR] un nombre négatif sera entre <>
- [TO_CHAR(date,format)] [to_char]TO_CHAR fct
  Renvoie conversion d'une date en chaîne de caractères. Le format indique quelle partie de la date doit apparaître, c'est une combinaison des codes suivants :

  [hh ou hh12]
  - [scc] siècle avec signe
  - [cc] siècle
  - [sy,yyy] année (avec signe et virgule)
  - [y,yyy] année(avec virgule)
  - [yyyy] année
  - [yyy] 3 derniers chiffres de l'année
  - [yy] 2 derniers chiffres de l'année
  - [y] dernier chiffre de l'année
  - [q] numéro du trimestre dans l'année
  - [ww] numéro de la semaine dans l'année
  - [w] numéro de la semaine dans le mois
  - [mm] numéro du mois

- [ddd] numéro du jour dans l'année
- [dd] numéro du jour dans le mois
- [d] numéro du jour dans la semaine
- [hh ou hh12] heure (sur 12 heures)
- [hh24] heure sur 24 heures
- [mi] minutes
- [ss] secondes
- [sssss] secondes après minuit
- [j] jour du calendrier julien

Les formats suivants permettent d'obtenir des dates en lettres (en anglais) :

[syear ou year]

- [syear ou year] année en toutes lettres
- [month] nom du mois
- [mon] nom du mois abrégé sur 3 lettres
- [day] nom du jour
- [dy] nom du jour abrégé sur 3 lettres
- [am ou pm] indication am ou pm
- [bc ou ad] indication avant ou après jesus christ

Les suffixes suivants modifient la présentation du nombre auquel ils sont accolés :

- [th] ajout du suffixe ordinat st, nd, rd, th
- [sp] nombre en toutes lettres

Tout caractère spécial inséré dans le format sera reproduit tel quel dans la chaîne de caractères résultat.

- [TO_DATE(chaîne,format)] [to_date]TO_DATE fct
  Permet de convertir une chaîne de caracteres en donnée de type date. Le format est identique à celui de la fonction TO_CHAR.
- [TO_NUMBER(chaine)] [to_number]TO_NUMBER fct
  Convertit chaine en sa valeur numérique.

Remarque : on peut également inserer dans le format une chaîne de caractères quelconque, à condition de la placer entre guillemets("). FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=2 COLS=50 SELECT TO_CHAR (embauche,'DD/MM/YY HH24:MI:SS') FROM emp; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=2 COLS=50 SELECT INSTR(fonction,'a',1,2) FROM emp; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Exemple 10209

Donner la liste de tous les employés dont le nom ressemble à DUPONT.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=3

COLS=50 SELECT nom FROM emp WHERE SOUNDEX(nom) = SOUNDEX('DUPONT'); /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Exemple 10210

Donner la liste de tous les noms des employés en ayant supprimé tous les 'L' et les 'E' en tête des noms.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=2 COLS=50 SELECT LTRIM(nom,'LE') FROM emp; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Exemple 10211

Donner la liste de tous les noms des employés en ayant remplacé les A et les M par des * dans les noms.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=2 COLS=50 SELECT TRANSLATE (nom,'AM','**') FROM emp; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Exemple 10212

Afficher tous les salaires avec un $ en tête et au moins trois chiffres (dont deux décimales).

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=2 COLS=50 SELECT TO_CHAR (salaire,'99900.00') FROM emp; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

---

next | up | previous | contents | index

Next: Autres fonctions **Up:** Les expressions et fonctions **Previous:** Fonctions sur les dates

Yolaine.Bourda@supelec.fr

Autres fonctions

- [GREATEST(expr1, expr2,...)] [greatest]GREATEST fct
  Renvoie la plus grande des valeurs expr1, expr2,.... Toutes les expressions sont converties au format de expr1 avant comparaison.
- [LEAST] [least]LEAST fct
  Renvoie la plus petite des valeurs expr1, expr2,.... Toutes les expressions sont converties au format de expr1 avant comparaison.
- [NVL(expr_1, expr_2)] [nvl]NVLl fct
  Prend la valeur expr_1, sauf si expr_1 est NULL auquel cas NVL prend la valeur expr_2. Une valeur NULL en sql est une valeur non définie. Lorsque l'un des termes d'une expression a la valeur NULL, l'expression entière prend la valeur NULL. D'autre part, un prédicat comportant une comparaison avec une expression ayant la valeur NULL prendra toujours la valeur faux. La fonction NVL permet de remplacer une valeur NULL par une valeur significative.
- [DECODE(crit, val_1, res_1 [, val_2, res_2 ...], def)] [decode]DECODE fct

  Cette fonction permet de choisir une valeur parmi une liste d'expressions, en fonction de la valeur prise par une expression servant de critère de sélection.

  Le résultat récupéré est :

  - res_1 si l'expression crit a la valeur val_1
  - res_2 si l'expression crit a la valeur val_2
  - def (la valeur par défaut) si l'expression crit n'est égale à aucune des expressions val_1, val_2,...,.

  Les expressions résultats res_1, res_2, ..., def peuvent être de types différents : caractère et numérique, ou caractère et date (le résultat est du type de la première expression rencontré dans le DECODE).

  La fonction DECODE permet également de mélanger dans une colonne résultat des informations venant de plusieurs colonnes d'une même table.

Exemple 10213

Donner pour chaque employé ses revenus (salaire + commission).

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=2 COLS=50 SELECT nom, salaire, comm, salaire+NVL(comm,0) FROM emp; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Exemple 10214

Donner la liste des employés avec pour chacun d'eux sa catégorie (président = 1, directeur = 2, autre = 3)

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=2 COLS=50 SELECT nom, DECODE(fonction,'president',1,'directeur',2,3) FROM emp; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Exemple 10215

Donner la liste des employés en les identifiant par leur fonction dans le département 10 et par leur nom dans les autres départements.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=2 COLS=50 SELECT DECODE (n_dept,10,fonction,nom) FROM emp; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

---

next up previous contents index

Next: Les fonctions de groupe **Up:** Les expressions et fonctions **Previous:** Fonctions de conversion

Yolaine.Bourda@supelec.fr

Les fonctions de groupe

Yolaine.Bourda@supelec.fr

Les fonctions de groupe

Dans les exemples précedents, chaque ligne résultat d'un select était le résultat de calculs sur les valeurs d'une seule ligne de la table consultée. Il existe un autre type de select qui permet d'effectuer des calculs sur l'ensemble des valeurs d'une colonne. Ces calculs sur l'ensemble des valeurs d'une colonne se font au moyen de l'une des fonctions suivantes :

- [`AVG([DISTINCT | ALL] expression)`] [avg]`AVG` fct
  Renvoie la moyenne des valeurs d'`expression`.
- [`COUNT(* | [DISTINCT | ALL] expression)`] [count]`COUNT` fct
  Renvoie le nombre de lignes du résultat de la requête. Si `expression` est présent, on ne compte que les lignes pour lesquelles cette expression n'est pas `NULL`.
- [`MAX([DISTINCT | ALL] expression)`] [max]`MAX` fct
  Renvoie la plus petite des valeurs d'`expression`.
- [`MIN([DISTINCT | ALL] expression)`] [min]`MIN` fct
  Renvoie la plus grande des valeurs d'`expression`.
- [`STDDEV([DISTINCT | ALL] expression)`] [stddev]`STDDEV` fct
  Renvoie l'écart-type des valeurs d'`expression`.
- [`SUM([DISTINCT | ALL] expression)`] [sum]`SUM` fct
  somme des valeurs
- [`VARIANCE([DISTINCT | ALL] expression)`] [variance]`VARIANCE` fct
  Renvoie la variance des valeurs d'`expression`.
- [`DISTINCT`]
  Indique à la fonction de groupe de ne prendre en compte que des valeurs distinctes.
- [`ALL`]
  Indique à la fonction de groupe de prendre en compte toutes les valeurs, c'est la valeur par défaut.

Exemple 10216

Donner le total des salaires du département 10.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=3 COLS=50 SELECT SUM(salaire) FROM emp WHERE n_dept = 10; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Exemple 10217

Donner le nom, la fonction et le salaire de l'employé (ou des employés) ayant le salaire le plus élevé.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=4

COLS=50 SELECT nom, fonction, salaire FROM emp WHERE salaire = (SELECT MAX(salaire) FROM emp); /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM**Remarques**

- Ces select sont différents de ceux vus précédemment. Il est, par exemple, impossible de demander en résultat à la fois une colonne et une fonction de groupe.
- un select comportant une fonction de groupe peut être utilisé dans une sous-interrogation.

---

next up previous contents index

Next: Valeurs NULL **Up:** Les fonctions de groupe **Previous:** Les fonctions de groupe

Yolaine.Bourda@supelec.fr

Valeurs `NULL`

Aucune des fonctions de groupe ne tient compte des valeurs `NULL` à l'exception de `count(*)`. Ainsi, `SUM(col)` est la somme des valeurs non `NULL` de la colonne `col`. De même `AVG` est la somme des valeurs non `NULL` divisée par le nombre de valeurs non `NULL`.

Yolaine.Bourda@supelec.fr

Calcul sur plusieurs groupes

Il est possible de subdiviser la table en groupes, chaque groupe étant l'ensemble des lignes ayant une valeur commune. C'est la clause `GROUP BY` qui permet de découper la table en plusieurs groupes :

GROUP BY expr_1, expr_2, ...Si on a une seule expression, ceci définit les groupes comme les ensembles de lignes pour lesquelles cette expression prend la même valeur. Si plusieurs expressions sont présentes les groupes sont définis de la façon suivante : parmi toutes les lignes pour lesquelles `expr_1` prend la même valeur, on regroupe celles ayant `expr_2` identique, ... Un select de groupe avec une clause `GROUP BY` donnera une ligne résultat pour chaque groupe.

Exemple 10218

Total des salaires pour chaque département

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=3 COLS=50 SELECT SUM(salaire), n_dept FROM emp GROUP BY n_dept; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM**Remarque :** Dans la liste des colonnes résultat d'un select comportant une fonction de groupe, ne peuvent figurer que des caractéristiques de groupe, c'est-à-dire :

- soit des fonctions de groupe ;
- soit des expressions figurant dans le `GROUP BY`.

Yolaine.Bourda@supelec.fr

Sélection des groupes

[having]`HAVING`

De la même façon qu'il est possible de sélectionner certaines lignes au moyen de la clause `WHERE`, il est possible dans un select comportant une fonction de groupe de sélectionner par la clause `HAVING`, qui se place après la clause `GROUP BY`.

Le prédicat dans la clause `HAVING` suit les mêmes règles de syntaxe qu'un prédicat figurant dans une clause `WHERE`.

Cependant, il ne peut porter que sur des caractéristiques du groupe : fonction de groupe ou expression figurant dans la clause `GROUP BY`, dans ce cas la clause `HAVING` doit être placée après la clause `GROUP BY`.

Exemple 10219

Donner la liste des salaires moyens par fonction pour les groupes ayant plus de deux employés.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=4 COLS=50 SELECT fonction,COUNT(*),AVG(salaire) FROM emp GROUP BY fonction HAVING COUNT(*) > 2; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM Un select de groupe peut contenir à la fois une clause `WHERE` et une clause `HAVING`. La clause `WHERE` sera d'abord appliquée pour sélectionner les lignes, puis les groupes seront constitués à partir des lignes sélectionnées, et les fonctions de groupe seront évaluées.

Exemple 10220

Donner le nombre d'ingénieurs ou de commerciaux des départements ayant au moins deux employés de ces catégories.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=5 COLS=50 SELECT n_dept, COUNT(*) FROM emp WHERE fonction in ('ingenieur','commercial') GROUP BY n_dept HAVING COUNT(*) >= 2; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM Une clause `HAVING` peut comporter une sous-interrogation.

Exemple 10221

Quel est le département ayant le plus d'employés?

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=6 COLS=50 SELECT n_dept,COUNT(*) FROM emp GROUP BY n_dept HAVING COUNT(*) = (SELECT MAX(COUNT(*)) FROM emp GROUP BY n_dept) ; /TEXTAREA p INPUT

TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Yolaine.Bourda@supelec.fr

Fonction de groupe à deux niveaux

Il est possible d'appliquer au résultat d'un select avec `GROUP BY` un deuxième niveau de fonction de groupe.

Exemple 10222

la fonction `MAX` peut être appliquée aux nombres d'employés de chaque département pour obtenir le nombre d'employés du département ayant le plus d'employés.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=3 COLS=50 SELECT MAX(COUNT(*)) FROM emp GROUP BY n_dept ; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Yolaine.Bourda@supelec.fr

Le traitement des structures d'arbre

Il est possible de représenter selon le modèle relationnel des listes ou des structures d'arbre. Pour cela, il suffit d'introduire dans la relation un attribut représentant un lien vers l'élément suivant ou le prédécesseur dans la liste ou l'arbre :

Relation(clé, ... , clé du prédécesseur)C'est le cas dans la table `emp` où chaque ligne contient un ``pointeur'' vers le supérieur hiérarchique : (`n_sup`). Chaque employé a un seul supérieur hiérarchique. Par contre, un employé peut être le supérieur hiérarchique de plusieurs employés. Le lien `n_sup` définit donc une structure d'arbre.

Yolaine.Bourda@supelec.fr

Parcours d'un arbre

En utilisant la clé `num` et le lien vers le supérieur hiérarchique `n_sup`, il est possible de parcourir l'arbre hiérarchique implicitement décrit par la table `emp`. sql permet d'obtenir comme résultat d'un select les lignes dans l'ordre de parcours de l'arbre (ou de la liste). Pour cela il faut :

- définir comment se fait le lien entre une ligne et la précédente, en indiquant la colonne clé et la colonne lien dans une clause `CONNECT BY` [connect]`CONNECT BY`:

  CONNECT BY nom_col_1 = PRIOR nom_col_2Le mot clé `PRIOR` [prior]`PRIOR`, accolé à l'une ou l'autre des colonnes, définit le sens du parcours.
- définir un (ou plusieurs) point de départ, par une clause `START WITH` [start with]`START WITH` placée après la clause `CONNECT BY`.

Un arbre (ou une liste) sera construit à partir de chaque ligne répondant au prédicat. En l'absence de clause `START WITH`, un arbre sera construit à partir de chaque ligne.

Exemple 10223

Quels sont toutes les personnes dont `CODD` est le supérieur hiérarchique (`CODD` compris).

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=4 COLS=50 SELECT nom, num, fonction, n_sup FROM emp CONNECT BY n_sup = PRIOR num START WITH nom = 'CODD'; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Yolaine.Bourda@supelec.fr

next up previous contents index

Next: Sélection de lignes **Up:** Le traitement des structures **Previous:** Parcours d'un arbre

Niveau : LEVEL

[level]`LEVEl`

Il est possible d'obtenir pour chaque ligne le niveau correspondant dans l'arbre ou dans la liste, la première ligne sélectionnée étant de niveau 1. Ce niveau s'obtient dans la variable `LEVEL`, que l'on peut utiliser dans le select comme résultat. Cette variable peut aussi être présente dans un prédicat.

Exemple 10224

Donner la liste des subordonnés de `JOUBERT`, avec pour chacun d'eux son niveau.

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=4 COLS=50 SELECT LEVEL, num, nom, fonction, n_sup, n_dept FROM emp CONNECT BY n_sup = PRIOR num START WITH nom = 'JOUBERT'; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM

Yolaine.Bourda@supelec.fr

next | up | previous | contents | index

Next: Clause WHERE **Up:** Le traitement des structures **Previous:** Niveau : LEVEL

Sélection de lignes

Yolaine.Bourda@supelec.fr

Clause `WHERE`

Le prédicat de la clause `WHERE` permet d'éliminer certaines lignes. Mais le parcours de l'arbre n'est pas interrompu lorsqu'une ligne ne répondant pas au prédicat est rencontrée.

Exemple 10225

Quels sont toutes les personnes dont `JOUBERT` est le supérieur hiérarchique (`JOUBERT` compris et `CODD` non compris).

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=5 COLS=50 SELECT num, nom, fonction, n_sup, n_dept FROM emp WHERE nom != 'CODD' CONNECT BY n_sup = PRIOR num START WITH nom = 'JOUBERT'; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM `SIMON` ne figure pas dans le résultat , mais tous ses subordonnés y figurent.

Yolaine.Bourda@supelec.fr

Clause `CONNECT BY`

Il est possible de rajouter des conditions dans le `CONNECT BY`. Lorsque le prédicat du `CONNECT BY` n'est plus satisfait, le parcours de l'arbre s'arrête.

Exemple 10226

Quels sont toutes les personnes dont `JOUBERT` est le supérieur hiérarchique (`JOUBERT` compris et `CODD` et ses subordonnés non compris).

FORM ACTION="/htbin/poly.com" METHOD=GET TEXTAREA NAME="query" ROWS=5 COLS=50 SELECT num, nom, fonction, n_sup, n_dept FROM emp CONNECT BY n_sup = PRIOR num AND nom !='CODD' START WITH nom = 'JOUBERT'; /TEXTAREA p INPUT TYPE="submit" VALUE="Exécuter la Requête" INPUT TYPE="reset" VALUE="Réinitialiser" /FORM Dans la requête précédente, `nom !='CODD'` n'était pas un des prédicats du `CONNECT BY` alors qu'ici il l'est.

Yolaine.Bourda@supelec.fr

Restrictions

Yolaine.Bourda@supelec.fr

Boucle

sql détecte les boucles dans les `CONNECT BY`. Le select est interrompu avec un code d'erreur.

Yolaine.Bourda@supelec.fr

Niveaux

Le parcours descendant d'un arbre nécessite, lorsqu'on a atteint l'extrémité d'une branche, de remonter jusqu'au noeud le plus proche pour parcourir la branche suivante. Le nombre de niveaux qu'il est ainsi possible de remonter est limité à 256.

Yolaine.Bourda@supelec.fr

Jointure

Un select avec clause `CONNECT BY` ne doit pas être une jointure.

Yolaine.Bourda@supelec.fr

SQL : modifier une base

Le langage de manipulation de données est le langage permettant de modifier les informations contenues dans une base de données. L'unité manipulée est la ligne. Il existe trois commandes sql permettant d'effectuer les trois types de modifications des données : ajout, modification et suppression.

Yolaine.Bourda@supelec.fr

Ajout de lignes

[insert]`INSERT` La commande `INSERT` permet d'insérer une ligne dans une table en spécifiant les valeurs à insérer. La syntaxe est la suivante :

INSERT INTO nom_table(nom_col1, nom_col2, ...) VALUES (val1, val2...) La liste des noms de colonne est optionnelle. Si elle est omise, la liste des colonnes sera par défaut la liste de l'ensemble des colonnes de la table dans l'ordre de la création de la table. Si une liste de colonnes est spécifiée, les colonnes ne figurant pas dans la liste auront la valeur `NULL`.

Il est possible d'insérer dans une table des lignes provenant d'une autre table. La syntaxe est la suivante :

INSERT INTO nom_table(nom_col1, nom_col2, ...) SELECT ...Le select peut contenir n'importe quelle clause sauf un `ORDER BY` qui impliquerait un classement des lignes contraire à l'esprit du relationnel.

Exemple 10227

Insérer dans la table bonus les noms et salaires des directeurs.

INSERT INTO bonus SELECT nom, salaire FROM emp WHERE fonction = 'directeur';

Yolaine.Bourda@supelec.fr

Modification de lignes

[update]update La commande update permet de modifier les valeurs d'une ou plusieurs colonnes, dans une ou plusieurs lignes existantes d'une table. La syntaxe est la suivante :

UPDATE nom_table SET nom_col1 = {expression1 | (SELECT ...) }, nom_col2 = {expression2 | (SELECT ...) } WHERE predicatLes valeurs des colonnes `nom_col1, nom_col2, ...` sont modifiées dans toutes les lignes satisfaisant au prédicat. En l'absence d'une clause `WHERE`, toutes les lignes sont mises à jour. Les expressions `expression1, expression2,...` peuvent faire référence aux anciennes valeurs de la ligne.

Exemple 10228

Augmenter de 10% les ingénieurs.

UPDATE emp SET salaire = salaire * 1.1 WHERE fonction = 'ingenieur' ;

Yolaine.Bourda@supelec.fr

Suppression de lignes

[delete]`DELETE` La commande `DELETE` permet de supprimer des lignes d'une table. La syntaxe est la suivante :

DELETE FROM nom_table WHERE prédicat ;Toutes les lignes pour lesquelles `prédicat` est évalué à vrai sont supprimées. En l'absence de clause `WHERE`, toutes les lignes de la table sont supprimées.

Yolaine.Bourda@supelec.fr

Gestion des transactions

Une transaction est un ensemble de modifications de la base qui forme un tout indivisible. Il faut effectuer ces modifications entièrement ou pas du tout, sous peine de laisser la base dans un état incohérent.

Les Systèmes de Gestion de Bases de Données permettent aux utilisateurs de gérer leurs transactions. Ils peuvent à tout moment :

- valider la transaction en cours par la commande `COMMIT`. Les modifications deviennent définitives et visibles à tous les utilisateurs. [commit]`COMMIT`
- annuler la transaction en cours par la commande `ROLLBACK`. Toutes les modifications depuis le début de la transaction sont alors défaites. [rollback]`ROLLBACK`

En cours de transaction, seul l'utilisateur ayant effectué les modifications les voit.

Ce mécanisme est utilisé par les systèmes de gestion de bases de données pour assurer l'intégrité de la base en cas de fin anormale d'une tâche utilisateur : il y a automatiquement `ROLLBACK` des transactions non terminées.

oracle est un système transactionnel qui assure la cohérence des données en cas de mise à jour de la base, même si plusieurs utilisateurs lisent ou modifient les mêmes données simultanément.

oracle utilise un mécanisme de verrouillage pour empêcher deux utilisateurs d'effectuer des transactions incompatibles et régler les problèmes pouvant survenir.

oracle permet le verrouillage de certaines unités (table ou ligne) automatiquement ou sur demande de l'utilisateur.

Les verrous sont libérés en fin de transaction.

Yolaine.Bourda@supelec.fr

SQL : définir une base

Le langage de définition des données est le langage permettant de créer ou de modifier le schéma d'une relation et donc d'une table. Il permet de créer, de modifier et de supprimer non seulement les tables, mais aussi les vues, les index et les clusters.

- Supprimer un cluster

Yolaine.Bourda@supelec.fr

Les tables

Yolaine.Bourda@supelec.fr

Créer une table

[create table]`CREATE TABLE`

La table est la structure de base contenant les données des utilisateurs. Quand on crée une table, on peut spécifier les informations suivantes :

- la définition des colonnes,
- les contraintes d'intégrité,
- La tablespace contenant la table,
- les caractéristiques de stockage,
- le cluster contenant la table,
- les données résultant d'une éventuelle requête.

Yolaine.Bourda@supelec.fr

Contraintes d'intégrité

A la création d'une table, les contraintes d'intégrité se déclarent de la façon suivante :

CREATE TABLE nom_table (nom_col_1 type_1,

nom_col_2 type_2, ... nom_col_n type_n CONSTRAINT [nom_contrainte_1] contrainte_1, CONSTRAINT [nom_contrainte_2] contrainte_2, ... CONSTRAINT [nom_contrainte_m] contrainte_m); Ou bien de la façon suivante :

CREATE TABLE nom_table (nom_col_1 type_1 CONSTRAINT [nom_contrainte_1_1] contrainte_1_1 CONSTRAINT [nom_contrainte_1_2] contrainte_1_2 ... CONSTRAINT [nom_contrainte_1_m] contrainte_1_m,

nom_col_2 type_2 CONSTRAINT [nom_contrainte_2_1] contrainte_2_1 CONSTRAINT [nom_contrainte_2_2] contrainte_2_2 ... CONSTRAINT [nom_contrainte_2_p] contrainte_2_p,

... nom_col_n type_n CONSTRAINT [nom_contrainte_n_1] contrainte_n_1 CONSTRAINT [nom_contraint_n_2] contrainte_n_2 ... CONSTRAINT [nom_contrainte_n_q] contrainte_n_q);Les contraintes différentes que l'on peut déclarer sont les suivantes :

- [`NOT NULL`] La colonne ne peut pas contenir de valeurs `NULL`.
- [`UNIQUE`] Chaque ligne de la table doit avoir une valeur différente ou `NULL` pour cette (ou ces) colonne.
- [`PRIMARY KEY`] Chaque ligne de la table doit avoir une valeur différente pour cette (ou ces) colonne. les valeurs `NULL` sont rejetées.
- [`FOREIGN KEY`] Cette colonne fait référence à une colonne clé d'une autre table.
- [`CHECK`] Permet de spécifier les valeurs acceptables pour une colonne.

Yolaine.Bourda@supelec.fr

Modifier d'une table

[alter table]`ALTER TABLE` On peut modifier dynamiquement la définition d'une table grace a la commande prgALTER TABLE. Deux types de modifications sont possibles : ajout d'une colonne et modification d'une colonne existante.

Il n'est pas possible de supprimer une colonne. Par contre une colonne qui n'est plus utilisée peut être mise à la valeur `NULL`, auquel cas elle n'occupe plus d'espace disque. Si on désire vraiment supprimer une colonne, il faut :

- se créer une nouvelle table sans la colonne en question
- détruire l'ancienne table,
- donner à la nouvelle table le nom de l'ancienne.

Yolaine.Bourda@supelec.fr

Supprimer une table

[drop table]`drop table` La commande `DROP TABLE` permet de supprimer une table, sa syntaxe est la suivante :

DROP TABLE nom_table ;La table `nom_table` est alors supprimée. La définition de la table ainsi que son contenu sont détruits, et l'espace occupé par la table est libéré.

Yolaine.Bourda@supelec.fr

Renommer une table

[rename]`RENAME`

On a la possibilité de changer le nom d'une table par la commande `RENAME`, la syntaxe est la suivante :
RENAME ancien_nom TO nouveau_nom ;

Yolaine.Bourda@supelec.fr

Les vues

Les vues permettent d'assurer l'objectif d'indépendance logique. Grace à elles, chaque utilisateur pourra avoir sa vision propre des données.

On a vu que le résultat d'un select est lui-même une table. Une telle table, qui n'existe pas dans la base mais est créée dynamiquement lors de l'exécution du select , peut être vue comme une table réelle par les utilisateurs. Pour cela, il suffit de cataloguer le select en tant que vue.

Les utilisateurs pourront consulter la base, ou modifier la base (avec certaines restrictions) à travers la vue, c'est-à-dire manipuler la table résultat du select comme si c'était une table réelle.

Yolaine.Bourda@supelec.fr

Créer une vue

[create view]`create view`

La commande `CREATE` view permet de créer une vue en spécifiant le select constituant la définition de la vue :

CREATE VIEW nom_vue [(nom_col1,...)] AS SELECT ... WITH CHECK OPTION ;La spécification des noms de colonnes de la vue est facultative. Par défaut, les noms des colonnes de la vue sont les mêmes que les noms des colonnes résultat du select (si certaines colonnes résultat du select sont des expressions, il faut renommer ces colonnes dans le select , ou spécifier les noms de colonne de la vue).

Une fois créée, une vue s'utilise comme une table. Il n'y a pas de duplication des informations mais stockage de la définition de la vue.

Exemple 10232

Création d'une vue constituant une restriction de la table `emp` aux employés du departement 10.

CREATE VIEW emp10 AS SELECT * FROM emp WHERE n_dept = 10 ;Le `CHECK OPTION` permet de vérifier que la mise à jour ou l'insertion faite à travers la vue ne produisent que des lignes qui font partie de la sélection de la vue.

Ainsi donc, si la vue `emp10` a été créée avec `CHECK OPTION` on ne pourra à travers cette vue ni modifier, ni insérer des employés ne faisant pas partie du département 10.

Il est possible d'effectuer des `INSERT` et des update à travers des vues, sous deux conditions :

- le select définissant la vue ne doit pas comporter de jointure,
- les colonnes résultat du select doivent être des colonnes réelles et non pas des expressions.

Exemple 10233

Modification des salaires du département 10 à travers la vue `emp10`.

UPDATE emp10 SET sal = sal *1.1;Toutes les lignes de la table `emp`, telles que le contenu de la colonne `n_dept` est égal à 10 seront modifiées.

 Yolaine.Bourda@supelec.fr

Supprimer une vue

[drop view]`drop view`

Une vue peut être détruite par la commande :
DROP VIEW nom_vue;

Yolaine.Bourda@supelec.fr

Renommer une vue

On peut renommer une vue par la commande :
RENAME ancien_nom TO nouveau_nom;

Yolaine.Bourda@supelec.fr

Les index

Yolaine.Bourda@supelec.fr

Créer d'un index

[create index]`create index`

Un index peut être créé par la commande suivante :

CREATE [UNIQUE] INDEX nom_index ON nom_table (nom_col1 , nom_col2, ...) [PCTFREE nombre] [COMPRESS | NOCOMPRESS] [ROWS = nombre_lignes] ;dans laquelle :

- L'option `UNIQUE` indique que l'on interdit que deux lignes aient la même valeur dans la colonne indexée.
- `PCTFREE` précise le pourcentage de place laissée libre dans les blocs d'index à la création de l'index. Cette place libre évitera une réorganisation de l'index des les premières insertions de nouvelles clés. La valeur par défaut est 20% .
- `NOCOMPRESS` indique que l'on ne veut pas comprimer les clés.
- `nombre_lignes` est une estimation du nombre de lignes, permettant d'optimiser l'algorithme de classement..

Un index peut être créé dynamiquement sur une table contenant déjà des lignes. Il sera ensuite tenu à jour automatiquement lors des modifications de la table.

Un index peut porter sur plusieurs colonnes, la clé d'accès sera alors la concaténation des différentes colonnes.

On peut créer plusieurs index indépendants sur une même table.

Les requêtes sql sont transparentes au fait qu'il existe un index ou non. C'est l'optimiseur du système de gestion de bases de données qui, au moment de l'exécution de chaque requête, recherche s'il peut s'aider ou non d'un index.

 Yolaine.Bourda@supelec.fr

Supprimer un index

[drop index]`drop index`

Un index peut être supprimé dynamiquement par la commande :

DROP INDEX nom_index;L'espace libéré reste attaché au segment d'index de la table : il pourra être utilisé pour un autre index sur la même table.

L'espace ne sera rendu à la partition que lors de la suppression de la table.

- Structure d'un index

Yolaine.Bourda@supelec.fr

Les clusters

Yolaine.Bourda@supelec.fr

Créer un cluster

[create cluster]`create cluster`

Avant de pouvoir mettre en cluster une ou plusieurs tables il faut créer le cluster au moyen de la commande `CREATECLUSTER` dont la syntaxe est la suivante :

CREATE CLUSTER nom_cluster (cle1 type1, cle2 type2, ...)où l'on donne un nom au cluster, et où l'on définit le nom et le type des colonnes constituant la clé du cluster.

CREATE CLUSTER nom_cluster (cle1 type1, cle2 type2, ...) [SIZE taille_du_bloc] [COMPRESS | NOCOMPRESS] [SPACE nom_de_space_definition]dans laquelle :

- [`SIZE`]
  est la taille d'un bloc de cluster. Cette taille peut varier de 1/6 de bloc oracle à 1 bloc oracle (2k octets sur vax/vms), ce paramètre doit être choisi de façon à avoir un bon remplissage des blocs.
- [`COMPRESS NOCOMPRESS`]
  est relatif à l'index qui sera crée sur la clé du cluster
- [`SPACE`]
  spécifie le `SPACE DEFINITION` qui définira les paramètres d'allocation d'espace pour le cluster.

CREATE CLUSTER DEM (DEPNO NUMBER) SIZE 512

Yolaine.Bourda@supelec.fr

Mise en cluster d'une table

En principe c'est dès sa création qu'il faut spécifier si une table sera implantée dans un cluster.

Yolaine.Bourda@supelec.fr

Retrait d'une table d'un cluster

Pour retirer une table d'un cluster il faut :

- créer une nouvelle table en dehors du cluster et copier la table en cluster dans la nouvelle table ;
- détruire l'ancienne table ;
- la nouvelle table pourra alors être renommée pour prendre le nom de l'ancienne.

ceci ne détruit pas la table, mais la reconstruit en dehors du cluster.

Yolaine.Bourda@supelec.fr

Supprimer un cluster

[drop cluster]`drop cluster`

Un cluster ne contenant aucune table peut être supprimé par la commande :

DROP CLUSTER nom_cluster;**Remarque :** les performances du cluster ne sont valables que si on n'a pas de blocs chaînés (ex de grande table).

Yolaine.Bourda@supelec.fr

Dictionnaire de données

Tous les systèmes de gestion de bases de données relationnels contiennent un dictionnaire de données intégré. C'est un ensemble de tables et de vues dans lesquelles sont stockées les descriptions des objets de la base, et qui sont tenues à jour automatiquement par le système de gestion de bases de données.

Ces tables ou vues, comme toutes les tables ou vues, peuvent être consultées au moyen du langage sql.

Yolaine.Bourda@supelec.fr

Description du dictionnaire des données

La vue DICTIONARY contient la description des tables et vues du dictionnaire de données, le contenu de cette vue varie en fonction du type d'utilisateur qui l'interroge (dba ou non). Les tables et vues du dictonnaire de donnees ont chacune un nom prefixé par :

[user]

- [user] pour les vues contenant la description des objets appartenant à l'utilisateur qui interroge le dictionnaire.
- [DBA] pour les vues accessibles uniquement aux utilisateurs ayant le privilège dba.
- [ALL] pour les vues contenant la description de tous les objets auxquels a accès l'utilisateur qui interroge le dictionnaire. Ces vues contiennent non seulement les objets créés par l'utilisateur mais aussi ceux pour lesquels on lui a explicitement donnés des droits d'accès ainsi que ceux qui sont accessibles a tout le monde (acces PUBLIC).

Ce chapitre contient la description d'un sous-ensemble des vues du dictionnaire de données.

Yolaine.Bourda@supelec.fr

vues décrivant les objets de l'utilisateur

USER_CATALOG

Liste des objets appartenant à l'utilisateur.

USER_CLUSTERS

Description des clusters créés par l'utilisateur.

USER_COL_COMMENTS

Commentaires sur les colonnes des tables et des vues créés par l'utilisateur.

USER_INDEXES

Description des index créés par l'utilisateur.

USER_IND_COLUMNS

Liste des colonnes indexées par l'utilisateur courant ou indexant les tables de l'utilisateur courant.

USER_OBJECTS

Description des objets appartenant à l'utilisateur.

USER_SYNONYMS

Liste de synonymes créées par l'utilisateur.

USER_TABLES

Description des tables créées par l'utilisateur.

USER_TAB_COLUMNS

Description des colonnes de chaque table ou vue créée par l'utilisateur courant. Chaque ligne de la vue col décrit une colonne.

USER_TAB_COMMENTS

Commentaires sur les tables et les vues créés par l'utilisateur.

USER_TAB_GRANTS

Droits sur les objets donnés par l'utilisateur ou dont l'utilisateur est le bénéficiaire.

USER_TAB_GRANTS_MADE

Droits sur ses objets, tables ou vues, donnés par l'utilisateur.

USER_TAB_GRANTS_RECD

Droits sur des objets donnés à l'utilisateur.

USER_USERS

Informations sur l'utilisateur courant.

USER_VIEWS

Texte des vues créées par l'utilisateur.

---

next up previous contents index

Next: Vues décrivant les objets **Up:** Dictionnaire de données **Previous:** Description du dictionnaire des Yolaine.Bourda@supelec.fr

Vues décrivant les objets auxquels l'utilisateur a accès

ALL_CATALOG

Liste de tous les objets accessibles par l'utilisateur.

ALL_COL_COMMENTS

Commentaires sur les colonnes des tables et des vues accessibles par l'utilisateur.

ALL_INDEXES

Description des index accessibles par l'utilisateur.

ALL_IND_COLUMNS

Liste des colonnes indexées appartenant aux tables accessibles par l'utilisateur.

ALL_OBJECTS

Description des objets accessibles par l'utilisateur.

ALL_SYNONYMS

Liste de tous les synonymes accessibles par l'utilisateur.

ALL_TABLES

Description des tables accessibles par l'utilisateur.

ALL_TAB_COLUMNS

Description de toutes les colonnes des tables ou vues accessibles par l'utilisateur.

ALL_TAB_COMMENTS

Commentaires sur les tables et les vues accessibles par l'utilisateur.

ALL_TAB_GRANTS

Droits sur les objets donnés par l'utilisateur ou dont l'utilisateur est le bénéficiaire.

ALL_TAB_GRANTS_MADE

Droits sur ses objets, tables ou vues, donnés par l'utilisateur.

ALL_TAB_GRANTS_RECD

Droits sur des objets donnés à l'utilisateur.

ALL_USERS

Informations sur tous les utilisateurs.

ALL_VIEWS

Texte des vues accessibles par l'utilisateur.

---

next | up | previous | contents | index

Next: Vues décrivant les objets **Up:** Dictionnaire de données **Previous:** vues décrivant les objets

Yolaine.Bourda@supelec.fr

Vues décrivant les objets accessibles uniquement aux DBA

DBA_CATALOG

Tous les objets de la base.

DBA_CLUSTERS

Tous les clusters de la base.

DBA_COL_COMMENTS

Commentaires sur toutes les colonnes de toutes les tables et les vues.

DBA_DATA_FILES

Informations sur les fichiers contenant la base.

DBA_FREE_SPACE

Informations sur l place disponible dans les tablespaces.

DBA_INDEXES

Tous les index de la base.

DBA_OBJECTS

Tous les objets de la base.

DBA_SYNONYMS

Tous les synonymes de la base.

DBA_TABLES

Description de toutes les tables de la base.

DBA_TABLESPACES

Description de toutes les tablespaces de la base.

DBA_TAB_COMMENTS

Commentaires sur toutes les tables et vues de la base

DBA_TAB_GRANTS

Tous les droits accordrés sur tous les objets de la base.

DBA_TS_QUOTAS

Quotas dans les tablespaces pour tous les utilsateurs de la base.

DBA_USERS

Informations sur tous les utilisateurs de la base.

DBA_VIEWS

Textes de toutes les vues de la base.

---

next | up | previous | contents | index

Next: Synonymes **Up:** Dictionnaire de données **Previous:** Vues décrivant les objets

Yolaine.Bourda@supelec.fr

Synonymes

Les noms des vues étant assez longs, les synonymes suivants ont étés définis :

[MYPRIV]

- [CAT] Synonyme pour USER_CATALOG
- [CLU] Synonyme pour USER_CLUSTERS
- [COLS] Synonyme pour USER_TAB_COLUMNS
- [DICT] Synonyme pour DICTIONARY
- [IND] Synonyme pour USER_INDEXES
- [MYPRIV] Synonyme pour USER_USERS
- [OBJ] Synonyme pour USER_OBJECTS
- [SEQ] Synonyme pour USER_SEQUENCES
- [SYN] Synonyme pour USER_SYNONYMS
- [TABS] Synonyme pour USER_TABLES

Yolaine.Bourda@supelec.fr

SQL : manuel de référence

Yolaine.Bourda@supelec.fr

ALTER TABLE

[alter table]ALTER TABLE commande Cette commande permet de modifier la définition d'une table, les modifications permises sont les suivantes :

- ajouter une colonne,
- ajouter une contrainte d'intégrité,
- redéfinir une colonne (type de donnée, taille, valeur par défaut)
- modifier les caractéristiques de stockage ou d'autres paramètres,
- activer, désactiver une contrainte d'intégrité ou un déclencheur,
- allouer explicitement une extension de plus.

Yolaine.Bourda@supelec.fr

Syntaxe

ALTER TABLE [schema.]table [ADD column datatype [DEFAULT expr] [column_constraint] ... | table_constraint | (column datatype [DEFAULT expr] [column_constraint] ... | table_constraint [, column datatype [DEFAULT expr] [column_constraint] ... | table_constraint] ...)] [MODIFY column [datatype] [DEFAULT expr] [column_constraint] ... | (column [datatype] [DEFAULT expr] [column_constraint] ... [, column datatype [DEFAULT expr] [column_constraint] ...] ...)] [PCTFREE integer] [PCTUSED integer] [INITRANS integer] [MAXTRANS integer] [STORAGE storage_clause] [DROP drop_clause] ... [ALLOCATE EXTENT [([SIZE integer [K|M]] [DATAFILE 'filename'] [INSTANCE integer])] [ENABLE enable_clause | DISABLE disable_clause] ...

- [schema] est le nom du schéma contenant la table. Le schéma par défaut est celui de l'utilisateur qui exécute la requête.
- [table] est le nom de la table qui sera modifiée.
- [ADD] ajoute une colonne ou une contrainte d'intégrité.
- [MODIFY]
- [column]
- [datatype]
- [DEFAULT]
- [column_constraint]
- [table_constraint]
- [PCTFREE, PCTUSED, INITRANS, MAXTRANS]
- [STORAGE]
- [DROP]
- [ALLOCATE EXTENT] Alloue explicitement une nouvelle extension pour la table.
- [ENABLE] Active une contrainte d'intégrité ou tous les triggers associés à la table.
- [DISABLE] Désactive une contrainte d'intégrité ou tous les triggers associés à la table.

Les contraintes d'intégrité spécifiées dans ces clauses doivent avoir été définies auparavant. On peut aussi activer ou désactiver des contraintes d'intégrité en utilisant les paramètre ENABLE et DISABLE de la clause CONSTRAINT. Quand on définit une contrainte d'intégrité elle est activée par défaut.

 Yolaine.Bourda@supelec.fr

Prérequis

Pour pouvoir modifier la définition d'une il faut soit être propriétaire de cette table, soit avoir le privilège `ALTER TABLE` sur cette table.

le privilège `ALTER ANY TABLE` permet de modifier la définition de n'importe quelle table appartenant à n'importe quel utilisateur.

Yolaine.Bourda@supelec.fr

Voir aussi

`CONSTRAINT`, `CREATE TABLE`, `DROP TABLE`CREATE TABLE

Yolaine.Bourda@supelec.fr

CONSTRAINT

[constraint]`CONSTRAINT` clause Cette clause permet de défir des contraintes d'intégrité. Une contrainte d'intégrité est une règle restreignant les valeurs contenues dans une ou plusieurs colonnes d'une table.

Yolaine.Bourda@supelec.fr

Syntaxe

Column constraint:

[CONSTRAINT constraint] [NOT] NULL | UNIQUE | PRIMARY KEY | REFERENCES [schema.]table [(column)] [ON DELETE CASCADE] | CHECK (condition) [USING INDEX [PCTFREE integer] [INITRANS integer] [MAXTRANS integer] [TABLESPACE tablespace] [STORAGE storage_clause]] [EXCEPTIONS INTO [schema.]table | DISABLE Table constraint:

[CONSTRAINT constraint] UNIQUE | PRIMARY KEY (column [,column] ...) | FOREIGN KEY (column [,column] ...) REFERENCES [schema.]table [(column [,column] ...)] [ON DELETE CASCADE] | CHECK (condition) [USING INDEX [PCTFREE integer] [INITRANS integer] [MAXTRANS integer] [TABLESPACE tablespace] [STORAGE storage_clause]] [EXCEPTIONS INTO [schema.]table | DISABLE

- [`CONSTRAINT`]
- [`NULL`] précise que la colonne peut contenir des valeurs `NULL`.
- [`NOT NULL`] précise que la colonne ne peut pas contenir des valeurs `NULL`.
- [`UNIQUE`]
- [`PRIMARY KEY`]
- [`FOREIGN KEY`]
- [`REFERENCES`]
- [`ON DELETE CASCADE`]
- [`CHECK`]
- [`USING INDEX`]
- [`EXCEPTIONS INTO`]
- [`DISABLE`]

Yolaine.Bourda@supelec.fr

Prérequis

Les clauses `CONSTRAINT` peuvent apparaître dns les commandes `CREATE TABLE` ou prgALTER TABLE. Pour définir une contrainte d'intégrité il faut avoir les privilèges nécessaires pour effectuer l'une ou l'autre de ces commandes. Définir des contraintes d'intégrité peut nécessiter des privilèges en plus qui dépendent du type de la contrainte.

Yolaine.Bourda@supelec.fr

Voir aussi

prgALTER TABLE,CREATE TABLE

Yolaine.Bourda@supelec.fr

CREATE CLUSTER

[create cluster]`CREATE CLUSTER` commande Cette commande permet de créer un cluster. Un cluster est une structure de stockage qui contient une ou plusieurs tables ayant une ou plusieurs colonnes en commun.

Yolaine.Bourda@supelec.fr

Syntaxe

CREATE CLUSTER [schema.]cluster (column datatype [,column datatype] ...) [PCTUSED integer] [PCTFREE integer] [SIZE integer [K|M]] [INITRANS integer] [MAXTRANS integer] [TABLESPACE tablespace] [STORAGE storage_clause] [INDEX | [HASH IS column] HASHKEYS integer]where:

- [schema] est le nom du schéma contenant le cluster. Le schéma par défaut est celui de l'utilisateur qui exécute la requête.
- [cluster] est le nom du cluster à créer.
- [column] est le nom d'une colonne de la clé du cluster.
- [datatype]
- [PCTUSED]
- [PCTFREE] précise le pourcentage de place laissée libre dans les blocs du clusters à la création du cluster l'index.
- [INITRANS]
- [MAXTRANS]
- [SIZE]
- [TABLESPACE] specifies the tablespace in which the cluster is created.
- [STORAGE] specifies how data blocks are allocated to the cluster.
- [INDEX]

  .
- [HASH IS]
- [HASHKEYS]

Yolaine.Bourda@supelec.fr

Prérequis

Pour pouvoir créer un cluster dans son propre schéma, il faut avoir le privilège `CREATE CLUSTER`. Pour pouvoir créer un cluster dans n'importe quel schéma il faut avoir le privilège `CREATE ANY CLUSTER`, de plus le propriétaire du schéma doit avoir assez d'espace libre dans la tablespace devant contenir le cluster ou le privilège `UNLIMITED TABLESPACE`.

Yolaine.Bourda@supelec.fr

Voir aussi

CREATE TABLE

Yolaine.Bourda@supelec.fr

CREATE TABLE

[create table]`CREATE TABLE` commande Cette commande permet de créer une table.

Yolaine.Bourda@supelec.fr

Syntaxe

CREATE TABLE [schema.]table (column datatype [DEFAULT expr] [column_constraint] ... | table_constraint [, column datatype [DEFAULT expr] [column_constraint] ... | table_constraint]...) [[PCTFREE integer] [PCTUSED integer] [INITRANS integer] [MAXTRANS integer] [TABLESPACE tablespace] [STORAGE storage_clause] | [CLUSTER cluster (column [, column]...)]] [ENABLE enable_clause | DISABLE disable_clause] ... [AS subquery]

- [`schema`] est le nom du schéma qui contiendra la table. Le schéma par défaut est celui de l'utilisateur qui exécute la requête.
- [`table`] est le nom de la table à créer.
- [`column`] est le nom d'une colonne de la table. Le nombre de colonnes possibles dans une table est compris entre 1 et 254.
- [`datatype`] est le type de la colonne.
- [`DEFAULT`] spécifie une valeur qui sera affectée à cette colonne si, lors d'un `INSERT`, one ne lui en précise pas.
- [`column_constraint`] est une contrainte d'intégrité qui fait partie de la définition de la colonne.
- [`table_constraint`] est une contrainte d'intégrité qui fait partie de la définition de la table.
- [`TABLESPACE`] précise la tablespace dans laquelle la table sera crée.
- [`CLUSTER`] précise que la table fera partie d'un cluster.
- [`ENABLE`] active une contrainte d'intégrité.
- [`DISABLE`] désactive une contrainte d'intégrité.
- [`AS subquery`] insère les lignes renvoyées par la sous requête dans la table, à sa création.

 Yolaine.Bourda@supelec.fr

Prérequis

Pour pouvoir créer une table dans son propre schéma, il faut avoir le privilège `CREATE TABLE`. Pour pouvoir créer une table dans le schéma d'un autre utilisateur, il faut avoir le privilège `CREATE ANY TABLE`. De plus le propriétaire du schéma doit avoir assez d'espace libre dans la tablespace devant contenir le cluster ou le privilège `UNLIMITED TABLESPACE`.

Yolaine.Bourda@supelec.fr

CREATE VIEW

[create view]`CREATE VIEW` commande Cette commande permet de définir une vue, table virtuelle, basée sur une ou plusieurs tables ou vues.

Yolaine.Bourda@supelec.fr

Syntaxe

CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW [schema.]view [(alias [,alias]...)] AS subquery [WITH CHECK OPTION [CONSTRAINT constraint]]

- [OR REPLACE]

  recrée la vue si elle existe déjà.
- [FORCE] crée la vue sans s'inquiéter de l'existence de la table et des privilèges sur celle-ci.
- [NOFORCE] crée la vue uniquement si la table existe et si le propriétaire du schéma contenant la vue possède les privilèges adéquats sur celle-ci. C'est la valeur par défaut.
- [schema] est le nom du schéma qui contiendra la vue. Le schéma par défaut est celui de l'utilisateur qui exécute la requête.
- [view] est le nom de la vue crée.
- [alias]
- [AS subquery]
- [WITH CHECK OPTION]
- [CONSTRAINT]

Yolaine.Bourda@supelec.fr

Prérequis

Pour pouvoir créer une vue dans son propre schéma, il faut avoir le privilège `CREATE VIEW`. Pour pouvoir créer une vue dans le schéma d'un autre utilisateur, il faut avoir le privilège `CREATE ANY VIEW`.

le propiétaire du schéma contenant la vue doit avoir les privilèges nécessaires pour pouvoir utiliser les commandes `SELECT, INSERT, UPDATE, DELETE` sur les tables ou vues sur lesquelles la vue est basée.

Yolaine.Bourda@supelec.fr

Voir aussi

`CREATE TABLE`, `CREATE SYNONYM`

Yolaine.Bourda@supelec.fr

DELETE

[delete]`DELETE` commande Cette commande permet de supprimer des données contenues dans une table ou dans une vue.

Yolaine.Bourda@supelec.fr

syntaxe

DELETE [FROM] [schema.]table | view [alias] [WHERE condition]

- [`schema`] est le nom du schéma contenant la table ou la vue à détruire. Le schéma par défaut est celui de l'utilisateur qui exécute la requête.
- [`table, view`] est le nom de la table ou de la vue contenant les lignes qui seront détruites. Si c'est un nom de vue, les lignes détruites appartiennent à la table sur laquelle la vue est basée.
- [`alias`] Est un alias assigné à la table. les alias sont généralement utilisés dans des `DELETE` contenant des requêtes.
- [`WHERE`] Détruit seulement les lignes satisfaisant la condition. Cette condition peut référencer la table et peut contenir des sous-requêtes. Si cette clause est omise détruit toutes les lignes.

Yolaine.Bourda@supelec.fr

Prérequis

Pour détruire des lignes appartenant à une table, il faut soit être propriétaire de la table, soit avoir le privilège `DELETE` sur cette table.

Le privilège `DELETE ANY TABLE` permet à un utilisateur de détruire des lignes se trouvant dans n'importe quelle table, ou n'importe quelle vue basée sur une table.

Yolaine.Bourda@supelec.fr

Voir aussi

`DROP TABLE, TRUNCATE`

Yolaine.Bourda@supelec.fr

DROP CLUSTER

[drop cluster]`DROP CLUSTER` commande Cette commande permet de détuire un cluster de la base de données.

Yolaine.Bourda@supelec.fr

Syntaxe

DROP CLUSTER [schema.]cluster [INCLUDING TABLES [CASCADE CONSTRAINTS]]where:

- [`schema`] est le nom du schéma contenant le cluster à détruire. Le schéma par défaut est celui de l'utilisateur qui exécute la requête.
- [`cluster`] est le nom du cluster à détruire
- [`INCLUDING TABLES`]
- [`CASCADE CONSTRAINTS`]

Yolaine.Bourda@supelec.fr

Prérequis

Le cluster doit appartenir à l'utilisateur, ou celui-ci doit avoir le privilège `DROP ANY CLUSTER`.

Yolaine.Bourda@supelec.fr

Voir aussi

DROP TABLE

Yolaine.Bourda@supelec.fr

DROP TABLE

[drop table]`DROP TABLE` commande Cette commande permet de détruire une table et toutes ses données.

Yolaine.Bourda@supelec.fr

Syntaxe

DROP TABLE [schema.]table [CASCADE CONSTRAINTS]

- [`schema`] est le nom du schéma contenant la table ou la vue à détruire Le schéma par défaut est celui de l'utilisateur qui exécute la requête.
- [`table`] est le nom de la table à détruire.
- [`CASCADE CONSTRAINTS`] drops all referential integrity constraints that refer to primary and unique keys in the dropped table. If you omit this option, and such referential integrity constraints exist, ORACLE returns an error and does not drop the table.

Yolaine.Bourda@supelec.fr

Prérequis

Pour pouvoir détruire une table il faut soit être propriétaire de cette table, soit avoir le privilège `DROP ANY TABLE`.

Yolaine.Bourda@supelec.fr

DROP VIEW

[drop view]`DROP VIEW` commande Cette commande permet de détruire une vue.

Yolaine.Bourda@supelec.fr

Syntaxe

DROP VIEW [schema.]view

- [schema] est le nom du schéma contenant la vue à détruire. Le schéma par défaut est celui de l'utilisateur qui exécute la requête.
- [view] est le nom de la vue à détruire.

Yolaine.Bourda@supelec.fr

Prérequis

Pour pouvoir détruire une vue il faut soit être propriétaire de cette vue, soit avoir le privilège `DROP ANY VIEW`.

Yolaine.Bourda@supelec.fr

Voir aussi

`CREATE TABLE`, `CREATE VIEW`, `CREATE SYNONYM`

Yolaine.Bourda@supelec.fr

INSERT

[insert]`INSERT` commande Cette commande permet d'ajouter des lignes à une table ou à une vue basée sur une table.

Yolaine.Bourda@supelec.fr

Syntaxe

INSERT INTO [schema.]table | view [(column [, column] ...)] VALUES (expr [, expr] ...) | subqueryDans lequel :

- [schema] est le nom du schéma contenant la table ou la vue. Le schéma par défaut est celui de l'utilisateur qui exécute la requête.
- [table]
- [view] est le nom de la table dans laquelle les lignes seront insérés. Si c'est un nom de vue qui est précisé, les données seront insérés dans la table basée sur la vue.
- [column]
- [VALUES]
- [subquery]

Yolaine.Bourda@supelec.fr

Prérequis

Pour pouvoir insérer des lignes dans une table il faut soit être propriétaire de cet objet, soit avoir le privilège `INSERT` sur cette table.

le privilège `INSERT ANY TABLE` permet d'insérer des lignes dans n'importe quelle table appartenant à n'importe quel utilisateur.

Yolaine.Bourda@supelec.fr

Voir aussi

`DELETE`, update

Yolaine.Bourda@supelec.fr

RENAME

[rename]`RENAME` commande Cette commande permet de renommer une table, une vue, une séquence, ou un synonyme privé.

Yolaine.Bourda@supelec.fr

Syntaxe

RENAME old TO new

- [old] est le nom actuel de l'objet.
- [new] est le nouveau nom de l'onbet.

Yolaine.Bourda@supelec.fr

Prérequis

L'objet doit appartenir à l'utilisateur courant.

Yolaine.Bourda@supelec.fr

Voir aussi

`CREATE SEQUENCE, CREATE SYNONYM, CREATE TABLE, CREATE VIEW`

Yolaine.Bourda@supelec.fr

SELECT

[select]SELECT commande Cette commande permet de récupérer des données contenues dans une ou plusieurs tables, vues, ou clichés.

Yolaine.Bourda@supelec.fr

Syntaxe

SELECT [DISTINCT | ALL] { * | { [schema.]{table | view | snapshot}.* | expr [c_alias] [, { [schema.]{table | view | snapshot}.* | expr [c_alias] }] ... } FROM [schema.]{table | view | snapshot} [t_alias] [, [schema.]{table | view | snapshot} [t_alias]] ... [WHERE condition] [[START WITH condition] CONNECT BY condition] [GROUP BY expr [, expr] ... [HAVING condition]] [{UNION | UNION ALL | INTERSECT | MINUS} SELECT command] [ORDER BY {expr|position} [ASC | DESC] [, {expr | position} [ASC | DESC]] ...] [FOR UPDATE [OF [[schema.]{table | view}.]column [, [[schema.]{table | view}.]column] ...] [NOWAIT]]

- [`DISTINCT`] renvoie toutes les lignes sélectionnées en enlevant les doublons.
- [`ALL`] renvoie toutes les lignes sélectionnées sans enlever les doublons. C'est la valeur par défaut.
- [`*`] renvoie toutes les colonnes de toutes les tables, les vues et les clichés précisés dans le `FROM`.
- [`table.*, view.*, snapshot.*`]

  sélectionne toutes les colonnes de la table, de la vue ou du cliché précisé.
- [`expr`] sélectionne une expression habituellement calculée sur les valeurs des colonnes appartenant à l'une des tables, vues, ou clichés de la clause `FROM`.
- [`c_alias`] la chaîne de caractères qui sert d'en-tête à la colonne (par défaut `expr`)
- [`schema`] est le nom du schéma contenant les tables, vues ou clichés sélectionnés. Le schéma par défaut est celui de l'utilisateur qui exécute la requête.
- [`table, view, snapshot`] est le nom de la table, de la vue ou du cliché contenant les données sélectionnées.
- [`t_alias`] synonyme pour la table dont le nom précède, à utiliser dans le reste de la requête.
- [`WHERE`] restreint les lignes sélectionnées à celles pour lesquelles la condition est vraie. Si cette clause est omise, toutes les lignes des tables, vues ou clichés précisés derrière le `FROM` sont renvoyées.
- [`START WITH`, `CONNECT BY`] renvoie les lignes en parcourant une arborescence.
- [`GROUP BY`] groupe les lignes sélectionnées en se basant sur la valeur de `expr` pour chaque ligne et renvoie une seule ligne par groupe.
- [`HAVING`] restreint les groupes de lignes renvoyés à ceux pour lesquels la condition spécifiée est vraie. Sans cette clause, tous les groupes sont renvoyés.
- [`UNION`, `UNION ALL`, `INTERSECT`, `MINUS`] Combine les lignes retournées par deux `SELECT` en utilisant une opération ensembliste.
- [`ORDER BY`] ordonne les lignes sélectionnées :
  - [`expr`] en utilisant la valeur de `expr`. Cette expression est basée sur des colonnes précisées derrière le `SELECT` ou sur des colonnes appartenant à des tables, vues ou clichés présents derrière le `FROM`.
  - [`position`] donne le numéro de la colonne dans l'ordre du `SELECT`.
  - [`ASC, DESC`] mode ascendant ou descendant. La valeur par défaut `ASC`.

- [FOR UPDATE] ``locke'' les lignes sélectionnées.
- [NO WAIT] retourne le controle à l'utilisateur si la commande SELECT essaye de bloquer une table utilisée par un autre utilisateur.

---

next | up | previous | contents | index

Next: Prérequis **Up:** SELECT **Previous:** SELECT Yolaine.Bourda@supelec.fr

Prérequis

Pour pouvoir sélectionner des lignes d'un objet(table, vue, cliché) il faut soit être propriétaire de cet objet, soit avoir le privilège `SELECT` sur cet objet.

le privilège `SELECT ANY TABLE` permet de sélectionner des lignes de n'importe quel objet appartenant à n'importe quel utilisateur.

Yolaine.Bourda@supelec.fr

Voir aussi

`DELETE`, update

Yolaine.Bourda@supelec.fr

UPDATE

[update]`UPDATE` commande Cette commande permet de modifier des données contenues dans une table ou dans une vue.

Yolaine.Bourda@supelec.fr

Syntaxe

UPDATE [schema.]table | view [alias] SET (column [, column] ...) = (subquery) | column = expr | (subquery) [, (column [, column] ...) = (subquery) | column = expr | (subquery)] ... [WHERE condition]

- [`schema`] est le nom du schéma contenant la table ou la vue à modifier. Le schéma par défaut est celui de l'utilisateur qui exécute la requête.
- [`table, view`] est le nom de la table à mettre à jour. Si c'est un nom de vue, la table mise à jour est celle sur laquelle la vue est définie.
- [`alias`] est un alias assigné à la table. les alias sont généralement utilisés dans des `UPDATE` contenant des requêtes.
- [`column`] est le nom de la colonne qui sera modifiée.
- [`expr`] est la nouvelle valeur de la colonne.
- [`subquery`] est un `SELECT` qui renvoie les nouvelles valeurs affectées aux colonnes correspondantes.
- [`WHERE`] restreint les lignes modifiées à celles pour lesquelles la condition est vraie. Si on omet cette clause toutes les lignes sont modifiées.

Yolaine.Bourda@supelec.fr

Prérequis

Pour pouvoir modifier des lignes appartenant à une table, il faut soit être propriétaire de cet objet, soit avoir le privilège `UPDATE` sur cette table.

le privilège `UPDATE ANY TABLE` permet de sélectionner des lignes de n'importe quel objet appartenant à n'importe quel utilisateur.

Yolaine.Bourda@supelec.fr

Voir aussi

`DELETE`, `INSERT`

Yolaine.Bourda@supelec.fr

Références

1

Alfred V. Aho and Jeffrey D. Ullman.
Foundations of Computer science.
Computer Science Press, 1982.

2

M. Bouzeghoub, M. Jouve, and P. Pucheral.
Systèmes de Bases de Données : des techniquqes d'implantation à la conception de schémas.
Eyrolles, 1990.

3

G. Gardarin and P. Valduriez.
Bases de Données relationnelles : analyse et comparaison des systèmes.
Eyrolles, 1985.

4

Georges Gardarin.
Bases de Données : Les systèmes et leurs langages.
Eyrolles, 1982.

5

P. C. Kanellakis.
Elements of relational database theory.
In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science Volume B : Formal model and semantics*. Elsevier, the MIT press, 1990.

6

H. F. Korth and A. Silberschatz.
Database System Concepts.
Mac Graw-Hill, 1991.

7

Carnegie Mellon.
Sql reference page, 1995.
"http://www.contrib.andrew.cmu.edu/usr/shadow/sql.html".

8

M. Tamer Ozsu and P. Valduriez.
principles of Distributed database dystems.
prentice-Hall, 1991.

Yolaine.Bourda@supelec.fr

Àpropos de ce document...

Bases de Données Relationnelles

et

Systèmes de Gestion de Bases de Données Relationnels
Le Langage sql

This document was generated using the **LaTeX**2HTML translator Version 97.1 (release) (July 13th, 1997)

The command line arguments were:
latex2html `-dir /urisc/si/yb/public_html/poly_bd poly`.

The translation was initiated by Yolaine Bourda on 3/17/1998

Yolaine.Bourda@supelec.fr

Création simple

La commande de création de table la plus simple ne comportera que le nom et le type de chaque colonne de la table. L'on peut créer une table par la commande `CREATE TABLE` en spécifiant le nom et le type de chaque colonne. A la création, la table sera vide mais un certain espace lui sera alloué. La syntaxe est la suivante :

CREATE TABLE nom_table (nom_col1 TYPE1, nom_col2 TYPE2, ...) L'option `NOT NULL` assure qu'oracle interdit lors d'un `INSERT` ou d'un update que cette colonne contienne la valeur `NULL`, par défaut elle est autorisée.

Yolaine.Bourda@supelec.fr

next up previous contents index

Next: Les types de données **Up:** Créer une table **Previous:** Création simple

Création avec Insertion de données

On peut insérer des données dans une table lors de sa création par la commande suivante :

CREATE TABLE nom_table [(nom_col1, nom_col2, ...)] AS SELECT... On peut ainsi, en un seul ordre sql créer une table et la remplir avec des données provenant du résultat d'un select .

On n'a pas besoin alors de spécifier de type pour les colonnes : les types des données sont ceux provenant du select . Si des conversions de type sont à faire, on peut dans le select utiliser les fonctions `TO_CHAR`, `TO_DATE`, `TO_NUMBER`.

Par défaut les noms des colonnes de la nouvelle table sont les noms des colonnes du select . Si des expressions apparaissent dans le select les colonnes correspondantes doivent impérativement être renommées.

Le select peut contenir des fonctions de groupes mais pas d'`ORDER BY` car les lignes d'une table ne peuvent pas être classées.

On peut, et même on doit, quand on crée une table définir les contraintes d'intégrité que devront respecter les données que l'on mettra dans la table (voir un peu plus bas).

---

next up previous contents index

Next: Les types de données **Up:** Créer une table **Previous:** Création simple

Yolaine.Bourda@supelec.fr

Les types de données

Les types de données peuvent être :

- [`NUMBER[(longueur,[précision])]` [number]`NUMBER` type
  Ce type de données permet de stocker des données numériques à la fois entières et réelles dont la valeur est comprise entre 10^{-130} et 10^{125} avec une précision de 38 chiffres.
  - [`longueur`]
    précise le nombre maximum de chiffres significatifs stockés (par défaut 38),
  - [`précision`]
    donne le nombre maximum de chiffres après la virgule (par défaut 38), sa valeur peut être comprise entre -84 et 127. Une valeur négative signifie que le nombre est arrondi à gauche de la virgule.
- [`CHAR(longueur)`] [char]`CHAR` type
  Ce type de données permet de stocker des chaînes de caractères de longueur fixe. `longueur` doit être inférieur à 255, sa valeur par défaut est 1.
- [`VARCHAR(longueur)`] [char]`VARCHAR` type
  Ce type de données permet de stocker des chaînes de caractères de longueur variable. `longueur` doit être inférieur à 2000, il n'y a pas de valeur par défaut.
- [`DATE`] [date]`DATE` type
  Ce type de données permet de stocker des données constituées d'une date et d'une heure.
- [`RAW(longueur)`] [raw]`RAW` type
  Ce type de données permet de stocker des caractères non imprimables.
- [`LONG`] [long]`LONG` type
  Ce type de données permet des stocker des chaînes de caractères de longueur variable et inférieure à 2^{31} -1. Les colonnes de ce type sont soumises à certaines restrictions ;
  - une table ne peut pas contenir plus d'une colonne de ce type ;
  - les colonnes de ce type ne peuvent pas apparaître dans des contraintes d'intégrité ;
  - les colonnes de ce type ne peuvent pas être indexées ;
  - les colonnes de ce type ne peuvent pas apparaître dans des clauses : `WHERE`, `GROUP BY`, `ORDER BY` ou `CONNECT BY` ainsi que dans un `DISTINCT`.

Yolaine.Bourda@supelec.fr

Ajouter une colonne

La commande suivante permet d'ajouter une ou plusieurs colonnes à une table existante :

ALTER TABLE nom_table ADD (nom_col1 TYPE1, nom_col2 TYPE2, ...)Les types possibles sont les mêmes que ceux décrits avec la commande `CREATE TABLE`.

Si la table contient déjà des lignes, la nouvelle colonne aura des valeurs `NULL` pour les lignes existantes.

Yolaine.Bourda@supelec.fr

Modifier une colonne

Il est possible de modifier la définition d'une colonne, à condition que la nouvelle définition soit compatible avec le contenu de la colonne et en respectant les contraintes suivantes :

- dans tous les cas il est possible d'augmenter la taille d'une colonne ;
- il est possible de diminuer la taille, ou même de changer le type d'une colonne vide ;
- on peut spécifier `NOT NULL` si la colonne ne contient aucune valeur `NULL` ;
- on peut dans tous les cas spécifier `NULL` pour autoriser les valeurs `NULL`.

Yolaine.Bourda@supelec.fr

Structure d'un index

Les index sont des structures permettant de retrouver une ligne dans une table à partir de la valeur d'une colonne ou d'un ensemble de colonnes. Un index contient la liste triée des valeurs des colonnes indexées avec les adresses des lignes (numéro de bloc dans la partition et numéro de ligne dans le bloc) correspondantes.

Tous les index oracle sont stockés sous forme d'arbres équilibrés (btree) : une structure arborescente permet de retrouver rapidement dans l'index la valeur de clé cherchée, et donc l'adresse de la ligne correspondante dans la table.

Dans un tel arbre, toutes les feuilles sont à la même profondeur, et donc la recherche prend approximativement le même temps quelle que soit la valeur de la clé.

Lorsqu'un bloc d'index est plein, il est éclaté en deux blocs. en conséquence, tous les blocs d'index ont un taux de remplissage variant de 50% à 100%. Sans index on balaie séquentiellement toute la table quelle que soit la position de élément recherché.

Yolaine.Bourda@supelec.fr

Lors de la création de la table

L'option cluster de l'ordre `CREATE` table permet de spécifier que la table doit être mise en cluster. Le cluster doit déjà exister.

CREATE TABLE nom_table (NOM_COL1 TYPE1 NOT NULL , (NOM_COL2 TYPE2 NOT NULL , ...) CLUSTER NOM_CLUSTER (NOM_COLI, NOM_COLJ...)nom_coli, nom_colj sont des noms de colonnes de la table, elles seront identifiées une à une aux colonnes clés du cluster spécifiées à la création du cluster.

Yolaine.Bourda@supelec.fr

next | up | previous | contents | index

Next: Retrait d'une table d'un **Up:** Mise en cluster d'une **Previous:** Lors de la création

Table déjà existante

En principe cela n'est pas possible, il faut donc procéder de la façon suivante :

- créer une nouvelle table avec l'option cluster et y copier le contenu de la table initiale ;
- supprimer l'ancienne table ;
- renommer éventuellement la nouvelle table.

Yolaine.Bourda@supelec.fr