SQL – ORACLE

Par M. Tondeur H.

Oracle en Pratique

Notion de Schéma sous Oracle

C'est un ensemble comprenant des structures de données et des données, il appartient à un utilisateur et porte le nom de ce dernier.

Chaque utilisateur possède ainsi son propre schéma.

Leurs éléments ou objet sont créé et modifié par des ordres SQL.

Utiliser SQLPLUS

Les commandes dans SQLPLUS

R exécute une requête
L Liste le contenue du Buffer
L* Liste la ligne Courante
Ln liste la ligne numéro n
I insère une ligne après la ligne courante
A « texte » Ajoute texte après la ligne courante
DEL Supprime la ligne courante
CLEAR Efface le Buffer

QUIT ou EXIT quitter sqlplus

CONNECT user/password@descripteur

GET fichier Charge dans le buffer le contenu du fichier.sql qui se trouve dans le répertoire courant.

SAVE fichier Ecrit le buffer courant dans le fichier.sql

START fichier Ouvre et lance le fichier.sql

SPOOL OUT
SPOOL fichier enregistre le buffer au fils de l'eau dans un fichier.
SPOOL OFF met fin à cet enregistrement

Poser un commentaire dans un fichier où requête SQL

* --commentaire sur une seule ligne
* /* commentaires sur plusieurs lignes*/

LDD Langage de Description des Données

Création de tables

CREATE TABLE [schéma.]nomtable
(colonne1 type1 [DEFAULT valeur1] [NOT NULL]
[,colonne2 type2 [DEFAULT valeur2] [NOT NULL]]
[CONSTRAINT nomContrainte1 TypeContrainte1]…) ;

- Schéma peut être omis, il sera donc assimilé au nom de l'utilisateur.

- Type des colonnes

Caractères (CHAR,NCHAR, VARCHAR2, NVARCHAR2,CLOB,NCLOB, LONG)
Valeur Numérique (NUMBER,INTEGER)
Date/heure (DATE,DATETIME,TIME)
Données binaires(BLOB,BFILE,RAW,LONG RAW)

Utilisation :

CHAR(n) chaîne de caractères fixes.
VARCHAR2(n) chaîne de caractère variable.
NCHAR(n)
NVARCHAR2(n)
NUMBER((t,d)) valeur numérique de t chiffres dont d décimales.
INTEGER nombre entier entre -2^{31} et $2^{31} - 1$

DATE Permet de stocker des moments ponctuels, la précision est composée du siècle, de l'année, du mois, du jour, de l'heure, des minutes et des secondes.

DATETIME Date + Heure

TIME heure au format HH :MM ::SS +/-MS

- Les Contraintes

Les contraintes peuvent s'appliquer de 2 manières
- Des contraintes sur les colonnes.
- Des contraintes sur la table.

Les contraintes sur les colonnes sont

DEFAULT valeur
NOT NULL
UNIQUE colonne
CHECK (condition)

4 types de contraintes :

CONSTRAINT nomContrainte
UNIQUE (colonne1 [,colonne2]…)
PRIMARY KEY (colonne1 [,colonne2]…)
FOREIGN KEY (colonne1 [,colonne2]…)
REFERENCES [schéma.]nomtablepere (Colonne1 [,Colonne2]…)
[ON DELETE {CASCADE|SET NULL}]
CHECK (condition)

Attention : Il est fortement recommandé de nommer les contraintes.
Pk_ fk_ ck_

Lister les contraintes posées

SELECT * FROM USER_CONSTRAINTS ;

Obtenir les nom et descriptions des tables
Obtenir les noms des tables

SELECT TABLE_NAME **FROM** USER_TABLES ;

DESC nomtable ;

Permet d'obtenir la structure de la table dont le nom est indiqué.

Poser un commentaire sur une table ou une colonne d'une table en SQL
COMMENT ON TABLE [schéma.]NomTable
COLUMN [schéma.]NomTable.nomColonne
IS 'MON COMMENTAIRE' ;

Voir les commentaires posés :

SELECT * FROM USER_TAB_COMMENTS ;

Supprimer une table
DROP TABLE [schéma.]NomTable [CASCADE CONSTRAINTS] ;

CASCADE CONSTRAINTS permet d'indiquer de supprimer les contraintes associées à cette table.

Renommer une table
RENAME ancienNom TO nouveauNom ;

Les contraintes d'intégrité, index et prérogatives associées à l'ancienne table sont automatiquement transférées sur la nouvelle.

Attention : Les vues, synonymes, procédures catalogués sont inchangés, il faut donc les recréer.

Autre manière de renommer une table
ALTER TABLE noTable RENAME TO NouveauNom ;

Ajouter une colonne
ALTER TABLE NomTable ADD NomColonne TypeCol CONSTRAINTS nomContraintes typeContraintes ;

Renommer colonne
ALTER TABLE NomTable RENAME COLUMN Nomcolonne TO Nouveau NomColonne ;

Modifier type des colonnes
ALTER TABLE Nomtable MODIFY Nomcolonne TypeColonne Contraintes ;

Exemple

ALTER TABLE piste **MODIFY** nom CHAR(10) **NOT NULL** ;

Supprimer des colonnes
ALTER TABLE nomTable DROP COLUMN nomColonne ;

Attention : Pas toutes les versions, ne supprime pas les colonnes comportant une clé primaire, ne supprime pas les colonnes utilisées dans les index, et ne peut pas supprimer toutes les colonnes.

TP sur machine (Initiation)

Soit les tables suivant :

Segment (**indIP** , nomSegment , Etage)

Salle(**nSalle** , nomSalle , nbPoste , ***indIP***)

Poste(**nPoste** , NomPoste , adresse_ip , TypePoste , ***nSalle***)

Software(**nLog** , NomLog , DateA , Version , TypeLog , Prix)

Installer(**NumIns**, **nPoste** , nTag , DateIns , Duree)

Types(Typelp , Vtype, UType, NomType)

Règles : Les nom des segments, des salles et des postes sont non nul ;
// Le Domaine de valeurs de la colonne ad s'étend de 0 à 255 ;
La colonne prix est supérieur ou égal à 0 ;
La colonne DateIns est égale à la date du jour par défaut ;

1) Ecrire le schéma des relations entre les tables.

2) Ecrire les requêtes SQL de création des tables.

3) Posez des commentaires SQL sur chaque Tables.

4) Ecrire les requêtes SQL de visualisation des tables, vérifier la correspondance des tables avec le Schéma des relations.

5) Renommer la table software en Logiciel (Ecrire les deux requêtes possible)

6) Renommer les colonnes **DateA** de la table Logiciel en **DateAch** et **Durée** de la table Installer en **Délai**

7) Finalement la table Types sera écrite comme ceci Types(**Typelp** , TypeLog, NomType,**nlog**). Ecrire les requêtes permettant cette modification.

8) Supprimer la colonne « TypeLog » de la table Software. Ajouter la colonne Typelp qui permet de relier cette table avec la table Types. Ecrire la requête nécessaire.

9) Afficher et exporter dans un fichier « log_tp1.txt » le résultat final des modifications ainsi que l'ensemble des requêtes, en déduire le schéma relationnel final.

10) Ecrire les requêtes SQL de destructions des tables.

LMD Langage de Manipulation des Données

Création des séquences

Les séquences sont des objets qui permettent de créer des séquences de valeur avec incrément ou décrément automatique.

CREATE SEQUENCE nomSequence
MINVALUE valeurMin
MAXVALUE valeurMax
START WITH valeurDeDebut
INCREMENT BY valeurIncrément
CYCLE|NOCYCLE ;

Pour faire appel a une séquence ou donne son nom suivie de la commande NEXTVAL.
Il est également possible d'utiliser la commande CURVAL qui permet d'obtenir la valeur courante du compteur sans incrémenter celui ci (lecteur seule).

L'utilisation pour la première fois de NEXTVAL pour un compteur donne la valeur courante de celui ci, les utilisation suivantes donnent les valeurs suivantes (fonction incrémentation activé).

TestSequence.NEXTVAL

Insérer des données dans vos tables

Pour créer une ou plusieurs lignes dans une table, SQL offre l'instruction INSERT INTO.

INSERT INTO nomtable(colonne1,colonne2,….) **VALUES**(valeur1,valeur2,….) ;

Plusieurs précautions à prendre :

- Les valeurs données dans le VALUES doivent être dans le même ordre et le même type que ceux de la colonne.

- Toutes les colonnes qui ne sont pas précisées, reçoivent alors la valeur par défaut. (Souvent NULL ou celle défini par la contrainte DEFAULT valeur1)

- Si la moindre valeur ne vérifie pas les contraintes de la table, alors l'instruction INSERT INTO échoue.

Exemple : INSERT INTO clients(numero,nom,prenom,code_postal) VALUES ('120','Tondeur','Hervé','59300') ;

Exemple d'utilisation des séquence :

***CREATE SEQUENCE** AutoInc*
***MINVALUE** 1*
***MAXVALUE** 999999*
***START WITH** 1*
***INCREMENT BY** 1 ;*

***INSERT INTO** clients(numero,nom,prenom,code_postal) **VALUES** (AutoInc.NEXTVAL,'Tondeur','Hervé','59300') ;*

Utilisation de sous requête pour l'insertion de données dans un table

Il est possible d'utiliser une sous requête qui renvoie un ensemble de tuples, qui pourront être réintégré dans votre table (souvent utilisé pour ma migration de données).

Dans ce cas particulier on n'utilise pas de clause VALUE, elle est implicite.

Exemple : ***INSERT INTO*** *commandes(numero,jour,client)*
SELECT *numero,jour,client*
FROM *commandes_anciennes*
WHERE *(SYSDATE-jour)>=180 ;*

Modifier des données dans vos tables

SQL vous permet de modifier une ou plusieurs données de vos tables grâce à la commande :

UPDATE nomTable **SET** colonne='valeur' [**WHERE** condition] ;

Exemple : ***UPDATE*** *articles* ***SET*** *prix=prix/6.55957 ;*

Exemple : ***UPDATE*** *articles* ***SET*** *prix=prix/6.55957* ***WHERE*** *devise='EURO' ;*

Lorsque la clause conditionnelle est omise, toutes les lignes de la colonne sont mise à jour.
Il est possible de mettre plusieurs colonne à jour en même temps, il faut se méfier des corrélations entre colonnes.

Quelques fonctions de conversions utiles

TO_CHAR(colonneDate) convertir une date en chaîne de caractères.

TO_DATE(ChaîneCaractéres) convertir une chaîne en type Date.

EXTRACT({YEAR|MONTH|DAY|HOUR|MINUTE|SECOND} FROM {Expression-Date| Expression-Interval}) extrait une parte données d'une date ou d'un intervalle.

NUMTOYMINTERVAL(expressionnumerique,{'YEAR'|'MONTH'}) convertir un nombre dans un type INTERVAL YEAR TO MONTH

NUMTODSINTERVAL(expressionnumerique,{'DAY'|'HOUR'|'MINUTE'|'SECOND'}) convertir un nombre dans un type INTERVAL DAY TO SECOND.

Supprimer des données dans vos tables

Pour supprimer une ou plusieurs ligne dans un seule table, SQL prévoit l'instruction :

DELETE FROM nomTable [**WHERE** condition] ;

Exemple : DELETE FROM clients ;
Supprimer tous les clients

***DELETE FROM** commandes **WHERE** (SYSDATE – jour)>=180 ;*

NB : Si une des lignes est référencé par une clé étrangère alors la requête est refusée.

Sélection de données

Pour afficher les données sous forme d'un tableau dont les colonnes porte un intitulé, SQL propose l'instruction :

SELECT colonnes **FROM** NomTable **WHERE** Conditions ;

- **les colonnes et le caractères joker ***

Il est possible dans le nom des colonnes d'utiliser le caractère spécial * qui permet de nommer toutes les colonnes d'une ou plusieurs tables.

- **Les colonnes calculées**

Dans la clause SELECT, on peut utiliser les opérateurs Arithmétiques(+ - * /), l'opérateur de concaténation des chaînes de caractères (||) ou des fonctions SQL (comme MOD pour le modulo, FLOOR pour la troncature entière ou SUBSTR pour l'extraction de sous chaînes de caractères)

Dés que l'on à un doute sur les priorités des opérateurs, il faut utiliser les parenthèses afin de lever toute ambiguïté.

A partir du moment ou l'on utilise des colonnes calculées, il est conseillé de lui donner un nom (un alias) avec le mot clef AS pour en expliquer le contenue.
Un alias est donné entre double quotes « prixTTC »

*Exemple : **SELECT** SUBSTR(prenom,1,1) || '.' Nom **AS** « identité »,*
*FLOOR ((SYSDATE – naissance) / 365.25) **AS** « Age »,*
*MOD((SYSDATE – naissance), 365) **AS** « Nb de jour depuis l'anniversaire »*
***FROM** clients ;*

- **Les fonctions d'agrégation SQL**

Syntaxe	Description
COUNT	Dénombrement des lignes
SUM	Somme des valeurs numériques
MIN	Valeur Numérique Minimale
MAX	Valeur Numérique Maximale
AVG	Moyenne des valeurs numériques

Ces fonctions ignorent les valeurs NULL dans leur calcul et n'éliminent pas les doublons.

Exemple :

***SELECT** MAX(prix) **AS** « prix le plus cher »,*
*MIN(prix) **AS** « Prix le moins cher »,*
*AVG(prix) **AS** « prix moyen »*
***FROM** articles ;*

SELECT COUNT*(numero)* ***AS*** *« Nombre de clients »*
FROM *clients ;*

- **Les Alias**

Les alias permettent de renommer des colonnes à l'affichage ou des tables dans la requête. Les alias de colonnes sont utiles pour les calculs.

On utilise le mot clef AS

Exemple :

SELECT *compa AS C1, nom* ***AS*** *NomEtPrenom, brevet C3* ***FROM*** *pilote ;*
SELECT *pl.compa* ***AS*** *C1, pl.nom* ***AS*** *NomEtPrenom, pl.brevet* ***AS*** *C3 FROM pilote* ***AS*** *pl ;*

NB : Oracle traduit les noms des alias en majuscules.
L'utilisation du mot clef AS est facultative
Il faut préfixer les colonnes par l'alias de la table lorsqu'il existe.

- **quelques fonctions Oracles**

ASCII(c) Retourne la valeur du caractère ASCII équivalent.
CHR(n) Retourne le caractère ASCII équivalent
CONCAT (c1,c2) Equivalent à ||
INITCAP (c) Première lettre de chaque mot mis en majuscule
UPPER (c) Met en majuscule la chaîne
LENGTH(c) Longueur de la chaîne
REPLACE(c1,c2,c3) recherche c2 dans c1 et le remplace par c3

ABS(n) valeur absolue
CEIL(n) plus petit entier
FLOOR(n) plus grand entier
ROUND(n,d) arrondi a d décimale la valeur n
SIGN(n) signe de n
TRUNC(n,d) coupure de n à d décimales

SYSDATE ou CURRENT_DATE Date courante
LAST_DAY(d) retourne le dernier jour du mois

GREATEST(exp1,exp2,exp3) retourne la plus grande des expressions
LEAST(exp1,exp2,exp3) retourne la plus petite des expression
NULLIF(exp1,exp2) retour NULL si exp1=exp2 sinon Exp1
NVL(exp1,exp2) converti exp1 en exp2 si exp1=NULL

- **La clause WHERE**

La clause WHERE peut être très riche en conditions de sélection

- **Les opérateurs de comparaisons**

< = > <= >= <>

- **Comparateur de vacuité**

  La Syntaxe =NULL ou <>NULL renvoie toujours « faux », on ne peut tester la vacuité d'une colonne qu'avec la syntaxe IS [NOT] NULL

  *Exemple SELECT * FROM clients WHERE cp IS NOT NULL ;*

- **Les opérateurs Logiques**

  Les mots clés **AND** et **OR** sont mis à disposition

  *Exemple **SELECT * FROM** articles **WHERE** couleur='bleu' **AND** (prix>100 **OR** prix<10) ;*

- **Les bornes d'inclusions**

  L'opérateur BETWEEN permet de remplacer avantageusement la séquence
  >= …. AND …. <=

  *Exemple : **SELECT * FROM** articles **WHERE** prix **BETWEEN** 10 **AND** 100 ;*

- **Sélection dans une liste**

  L 'opérateur IN permet vérifier qu'une valeur est contenue dans une liste de valeurs.
  Nb : il peut être associé à l'opérateur NOT pour signifier n'est pas dans.

  *Exemple : **SELECT * FROM** clients **WHERE** nom **NOT IN** ('Dupont','Durand') ;*

- **Comparateur de Motifs**

  Il est possible de balayer une colonne de type chaîne de caractères à la recherche d'un motif, grâce à l'opérateur de filtre LIKE et du caractère % qui remplace toute série de caractère(s) (y compris vide).

  *Exemple : **SELECT * FROM** clients **WHERE** nom **LIKE** 'D%' ;*
  *__SELECT * FROM__ clients __WHERE__ nom __LIKE__ '%e' ;*

  Il existe également le caractère _ qui remplace 1 caractère, on peut l'utiliser plusieurs fois de suite.

- **Opérations sur le résultat**

  Le résultat d'une requête SELECT peut comporter plusieurs fois la même ligne. Pour éliminer les lignes doublons, il existe les mot-clés DISTINCT, UNIQUE, ALL

  DISTINCT et UNIQUE permettent d'éliminer les éventuels duplicatas.

  ALL affiche l'ensemble des données (c'est l valeur par défaut).

  *Exemple : **SELECT DISTINCT** nom **FROM** clients ;*

Il est également possible de trier les lignes du résultat, nous disposons de la clause ORDER BY.

Exemple : ***SELECT * FROM*** *clients* ***ORDER BY*** *nom* ***ASC NULLS FIRST*** *;*

On utilise les mots clés :

ASC pour classer dans l'ordre croissant
DESC pour classe dans l'ordre décroissant
NULLS FIRST les valeurs nulles en haut
NULLS LAST les valeurs nulles en bas

On peut utiliser dans la clause ORDER BY les alias de la clause SELECT

Enfin pour garder que les x premières lignes du résultat, Oracle à prévu de numéroter les lignes du résultat dans un colonne cachée nommée ROWNUM

Exemple : ***SELECT DISTINCT*** *nom* ***FROM*** *clients* ***WHERE*** *ROWNUM<=100* ***ORDER BY*** *nom* ***ASC NULLS LAST*** *;*

Les opérations ensemblistes

On peut articuler les résultats de plusieurs requêtes homogènes à l'aide des opérations ensemblistes UNION, INTERSECT et MINUS.

Les règles à respecter avec ces opérations ensemblistes sont les suivantes :

- Les colonnes affichées par les deux requêtes doivent être compatibles, en nombre , en ordre et en type de données.

- Les éventuels alias ne sont définis que dans la première clause SELECT.

- Une éventuelle clause ORDER BY n'es possible qu 'à la fin de la dernière requête, car elle agit sur le résultat final.

Dernières remarques :

- Par défaut, l'opération UNION élimine les doublons ; pour les conserver, il faut utiliser l'opérateur UNION ALL.

- Contrairement à UNION et INTERSECT, l'opérateur MINUS n'est pas commutatif, donc l'ordre dans lequel les requêtes sont écrites, de part et d'autre, a de l'importance.

Exemple :

-- affiche les identité de tous les clients et de tous les employés
SELECT *prenom || ' ' || nom* ***AS*** *« identité »* ***FROM*** *clients*
UNION
SELECT *prenom || ' ' || nom* ***AS*** *« identité »* ***FROM*** *employes* ***ORDER BY*** *« identité »* ***ASC*** *;*

-- affiche les identités de tous les clients qui ont un homonyme parmi les employés
SELECT *prenom || ' ' || nom* ***AS*** *« identité »* ***FROM*** *clients*
INTERSECT

***SELECT** prenom || ' ' || nom **AS** « identité » **FROM** employes **ORDER BY** « identité » **ASC** ;*

-- affiche les identités de tous les clients qui n'ont pas d'homonyme parmi les employés
***SELECT** prenom || ' ' || nom **AS** « identité » **FROM** clients*
MINUS
***SELECT** prenom || ' ' || nom **AS** « identité » **FROM** employes **ORDER BY** « identité » **ASC** ;*

TP sur machine (Suite)

Soit les tables suivant vu dans le TP précédent :

Segment (**indIP** , nomSegment , Etage)
-- Chaque étage est découpé en segment d'exploitation, qui représente un ensemble de salles contenant des machines.

Salle(**nSalle** , nomSalle , ***indIP***)
-- Cette table représente l'ensemble des salle contenant des machines.

Poste(**nPoste** , NomPoste , ***indIP*** , ad , TypePoste , ***nSalle***)
-- Représente l'ensemble des machines existantes ad est le nom du Host de la machine.

Logiciel(**nLog** , NomLog , DateAch , Version , **TypeLp** , Prix)
-- représente les différents logiciels achetés et à disposition

Installer(**NumIns**, **nPoste** , **nLog**, nTag , DateIns , Delai)
--Table qui représente les logiciels installés (ntag est un nombre qui représente la nieme installation du logiciel)

Types(**Typelp** , TypeLog, NomType)
-- Table qui permet de représenter un type complet de logiciel.

0) Afficher et exporter dans un fichier « log_tp1.txt » le résultat final des modifications ainsi que l'ensemble des requêtes.

1) Ecrire le schéma des relations entre les tables.

2) Ecrire les requêtes SQL de création des tables et posez des commentaires SQL sur chaque Tables. Ecrire les requêtes SQL de visualisation des tables, vérifier la correspondance des tables avec le Schéma des relations.

3) Insérer des données dans les différentes tables de la base de données ci dessus.

NB : Nsalle,indIP, nposte, nlog, TypeLp sont des numéro automatiques.

Segment

NomSegment	***Etage***
---	---

bâtiment ISTV 1 0
bâtiment ISTV 1 1
bâtiment ISTV 1 2
bâtiment ISTV 2 0
bâtiment ISTV 2 1
bâtiment ISTV 3 0
bâtiment ISTV 3 1

Je vous laisse le plaisir d'associé les salles et les machines aux différents segments données ci dessus.

4) Après restructuration, une nouvelle salle en ISTV2 etg 1 a été crée (221^{E}), ainsi qu'en ISTV3 Etg 0 (09T) .Faire apparaître dans la base ces nouvelles salles.

5)
Installer en Salle 07T, 200g microsoft Windows 2000
Installer en salle 01^{E}, 104^{e} Microsoft Windows NT4
Installer en Salle 07T et 200g Linux RedHat 9.2
Installer en Salle 07T le dernier SDK Java, ainsi que Jcreator version 3.2.
Installer en salle 104^{E} le Pack Microsoft Office 2000.
Installer en salle 200g Mathematica version 9.0a
Installer sur un poste unique de la salle 104^{E} Oracle 10g.

Mettre à jour votre base de données une fois ces actions effectuées.

6) Une erreur s'est glisser dans votre ordre de mission, ce n'est pas la version 2000 du pack office que vous deviez installer en salle 104^{E}, mais le version 97, après désinstallation et installation de la bonne version, mettez à jour votre base de données.

7) Finalement il n'est pas nécessaire d'installer Mathematica en salle 200g, donc après désinstallation, mettez à jour votre base de données.

8) Donner les nom des salles qui contiennent Windows NT4, donner le nom des salles qui contiennent Windows 2000.

9) Compter le nombre de salles puis de machines qui sont équipé d'un OS Microsoft. De même pour les salles et les machines qui possède un OS basé sur Linux.

10) Donner les nom des sections et des salles qui contiennent à la fois un OS Linux et Jcreator.

11) Afficher par ordre croissant les numéro de poste et nom de poste qui sont du type PC. Puis donner le nombre de machines que cela représente.

12) Donner les noms des logiciels installé depuis moins de 2 jours.

LE LMD (Langage de Manipulation des Données) SUITE

Les Jointures

EquiJointure

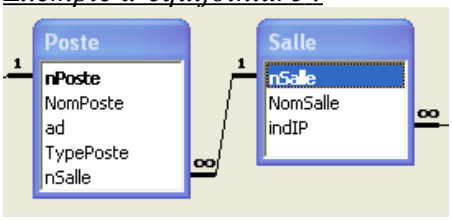
Une equijointure utilise l'opérateur = et compare en généralement des clés primaires avec des clés secondaires.

Oracle recommande d'utiliser des alias de tables pour améliorer les performances.

Les alias sont obligatoires pour des colonnes qui portent le même nom ou pour les auto jointures.

On va trouver deux formes d'écritures, relationnelle (=) et SQL2 (JOIN…ON condition), l'utilisation de INNER devant JOIN est optionnelle et est appliqué par défaut.

Exemple d'équijointure :



SELECT *NomSalle, NomPoste, TypePoste*
FROM *Poste,Salle*
WHERE *Poste.nSalle=Salle.nSalle* ***AND*** *Salle.NomSalle='07T' ;*

SELECT *NomSalle, NomPoste, TypePoste*
FROM *Salle* ***JOIN*** *Poste* ***ON*** *Poste.nSalle=Salle.nSalle*
WHERE *Salle.NomSalle='07T' ;*

Auto Jointure

Cas particulier de l'équijointure, l'auto jointure, relie une table à elle même.

Soit la table suivante :

Commandes(**numero**,client, jour)

Avec une autojointure il est possible de répondre à la question suivante : Afficher les numéros des clients qui ont commandé en même temps.

SELECT DISTINCT *a.client, b.client*
FROM *commandes a, commandes b*
WHERE *a.client < b.client*
AND *a.jour=b.jour ;*

SELECT DISTINCT *a.client, b.client*
FROM *commandes a* ***JOIN*** *commandes b* ***ON*** *a.client<b.client*
WHERE *a.jour=b.jour ;*

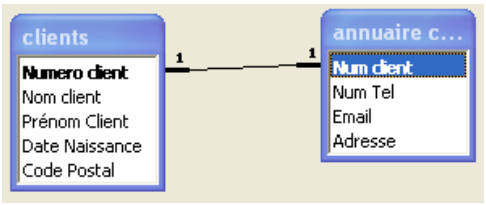
DISTINCT est obligatoire car sinon, deux clients qui ont commandé en même temps deux fois, apparaîtraient deux fois.

Jointure externe

Les jointures externes permettent d'extraire des enregistrements qui ne répondent pas aux critères de jointure. Lorsque deux tables sont en jointure externe, une tables est « dominante » par rapport à l'autre qui est dite « subordonnée ». Ce sont les enregistrements de la table dominante qui sont retournés (même s'il ne satisfont pas aux conditions de jointure).

Ecriture relationnelle
- La directive de jointure externe (+) se place du côté de la table subordonnée.
- Cette directive peut se placer à gauche ou à droite d'une clause de jointure, pas des deux côtés.
- Une clause de jointure externe ne peut ni utiliser l'opérateur IN ni être associé à une autre condition par l'opérateur OR.

Ecriture SQL2
- Le sens de la directive de jointure externe LEFT ou RIGHT de la clause OUTER JOIN désigne la table dominante.



--Affichage de toutes les informations relatives aux clients dont on connaît les informations complémentaires.

SELECT ***
FROM *clients a, annuaire_client b*
WHERE *a.numero_client=b.num_client ;*

Cette requête n'affiche pas tous les clients car il existe des clients pour lesquels on ne connaît aucune information complémentaire.

Il faut donc que la condition de jointure ne soit pas obligatoire, afin de ne pas supprimer à l'affichage les clients dont on n'a pas d'information complémentaire.

On parle alors de jointure externe. Il suffit pour cela d'ajouter le signe (+) dans la condition de jointure du coté de la table subordonnée :

Ecriture relationnelle

```
SELECT *
FROM clients a, annuaire_client b
WHERE a.numero_client=b.num_client (+) ;
```

Une requête équivalente

```
SELECT *
FROM clients a, annuaire_client b
WHERE b.num_client (+) = a.numero_client ;
```

Ecriture SQL2

```
SELECT *
FROM clients a LEFT OUTER JOIN annuaire_client b ON
(a.numero_client=b.num_client);
```

Une requête équivalente

```
SELECT *
FROM annuaire_client b RIGHT OUTER JOIN clients a ON
(a.numero_client=b.num_client);
```

Les Sous Requêtes ou jointures procédurales

Il est possible d'utiliser dans une requête principale, une sous requête SELECT à condition que cette dernière soit délimitée par des parenthèses.

Sous requête qui renvoient une seule valeur

Lorsque la ss-requête renvoie de manière sure une seule valeur, il est possible d'utiliser les opérateurs de comparaisons classique.

Exemple :

```
SELECT * FROM articles
WHERE prix >= (SELECT MAX(prix) FROM articles WHERE couleur='bleu') ;
```

Ces ss-requête peuvent également s'employer avec des opérateurs arithmétiques.

Exemple :

```
SELECT
(
SELECT COUNT(*)
FROM clients
WHERE FLOOR((SYSDATE – b.naissance) /365.25) <18
)
/
```

```
(
SELECT COUNT(*)
FROM clients
) * 100 AS « proportion de clients mineurs »
FROM DUAL
```

Sous requête qui renvoient une colonne

Lorsque la sous requête renvoie une colonne de valeurs, la requête principale peut utiliser un opérateur de comparaison classique, mais accompagné soit de ALL soit de ANY et IN :

Exemple :

```
SELECT * FROM articles
WHERE prix >= ANY (SELECT prix
                   FROM articles
                   WHERE couleur='bleu') ;
```

ALL ⇔ a tous
ANY ⇔ au moins un

IN ⇔ = ANY

NOT IN ⇔ <>ALL

Sous requête qui renvoie plusieurs colonnes

Une sous requête peut renvoyer plusieurs colonnes de valeurs, auquel cas elle peut être utilisée dans la clause FROM de la requête principale.

Exemple :

```
SELECT *, FLOOR(SYSDATE – b.naissance) / 365.25) AS « Age »
FROM (SELECT *
        FROM clients
       WHERE ROWNUM <=10
        ORDER BY naissance DESC) a , annuaire_client b
WHERE a.client = b.numero (+) ;
```

Une sous requête multi colonnes est également utilisable avec un opérateur de comparaison entre couples et tous les autres tuples.

Corrélation

Une sous requête peut avoir besoin d'information provenant de la requête principale. On dit alors que la requête est corrélée et le passage d'information se fais par l'utilisation d'alias.

Attention avec la corrélation il risque d'y avoir beaucoup de calcul et de pertes de temps, préférer utiliser une technique différente si possible.

Autres Utilisations

L'emploie d'une sous requête peut être appelé également par une requête INSERT et également dans une requête DELETE

Les clauses de Groupements

Il est parfois nécessaire de calculer la même fonction d'agrégation sur plusieurs de lignes.

La clause GROUP BY

Exemple pour calculer le montant total de chaque commande, il est nécessaire de grouper les lignes de commandes.

Exemple :

```
SELECT a.commande, SUM(a.quantité * b.prix) AS « Montant total »
FROM ligne_commande a , articles b
WHERE a.article = b.numéro
GROUP BY a.commande ;
```

Pour reconnaître une requête SELECT qui nécessite une clause GROUP BY, il suffit de se demander si le résultat doit afficher plusieurs valeurs d'une même fonction d'agrégat.

Attention : il existe deux contraintes impératives sur les colonnes de la clause SELECT ;
Toutes les colonnes de la clause GROUP BY doivent figurer sur la clause GROUP BY
Toutes les colonnes de la clause SELECT sans fonction d'agrégat, doivent figurer dans la clause GROUP BY

Il est parfois indispensable de procéder par plusieurs vue préliminaires avant de pouvoir calculer.

Exemple :

```
--calcul des revenus
CREATE OR REPLACE VIEW revenus (formation, revenu)
AS
SELECT a.formation, SUM(a.présence * b.frais)
FROM participations a, formations b
WHERE a.formation=b.numero
GROUP BY a.formation ;

--calcul des marges
SELECT a.formation, a.revenu – b.coût AS « profit »
FROM revenus a, couts b
WHERE a.formation=b.formation ;

--calcul des coûts
CREATE OR REPLACE VIEW coûts (formation, coût)
AS
SELECT a.formation, SUM(a.heures * b.bonus * c.tarifs + b.prime + b.fixe)
FROM animation a, formations b, animateurs c
WHERE a.formation=b.numero AND a.animateur=c.numero
GROUP BY a.formation ;
```

Regroupement sur plusieurs colonnes

La clause GROUP BY permet d'effectuer des groupements sur plusieurs colonnes (puis des calculs d'agrégation sur ces sous-groupes).

Exemple :

```
SELECT a.client, b.article, MAX(a.quantité) AS « Quantité MAX »
FROM commandes a , ligne_commandes b
WHERE a.numero=b.commande
GROUP BY a.client, b.article ;
```

L'ordre des colonnes dans la clause GROUP BY n'a pas d'importance et l'ordre des mêmes colonnes dans la clause SELECT n'a pas besoin d'être le même.

Par contre il est conseillé d'utiliser la clause ORDER BY pour laquelle l'ordre à de l'importance.

Exemple :

```
--Pour afficher le résultat d'abord par N° client croissant puis
-- (à l'intérieur d'un même N° client) par N° article décroissant
SELECT a.client,b.article, MAX(a.quantite) AS « Quantite MAX »
FROM commandes a, lignes_commandes b
WHERE a.numero=b.commande
GROUP BY a.client, b.article
ORDRE BY a.client ASC, b.article DESC ;
```

La clause HAVING

Il est parfois utilise d'exclure certains groupes issus d'une requête GROUP BY à l'aide d'une ou plusieurs conditions de sélection.
Ces conditions ne peuvent être écrites dans la clause WHERE qui est réservée à l'exclusion des lignes avant groupement, mais dans une dernière clause, la clause HAVING.

Exemple :

```
SELECT client, COUNT(numero) AS « Nombre de commandes »
FROM commandes
GROUP BY client
HAVING COUNT(numero)>=4 ;
```

Les alias ne peuvent être utilisé dans une clause HAVING, il faut en conséquence de quoi répéter la formule.

Tous les opérateurs qui sont autorisé dans une clause WHERE peuvent être utilisés dans la clause HAVING, y compris une sous requête.

Par contre les conditions de sélection avant groupement (et les conditions de jointure) doivent rester dans la clause WHERE.

Exemple :

```
SELECT a.article, b.client, SUM(a.quantite) AS « quantite totale »
FROM ligne_commandes a, commandes b
WHERE a.commande=b.numero
GROUP BY a.article, b.client
HAVING (a.article, SUM(a.quantite)) IN (SELECT article, MAX(quantite_totale)
                                        FROM QuantitesTotales
                                        GROUP BY client) ;
```

La clause WHERE peut porter sur les colonnes de la clause SELECT sans fonction d'agrégation ainsi que sur les colonnes non affichées.

La clause HAVING ne peut porter que sur les colonnes de la clause SELECT (agrégées ou non).

Créer des Vues

- On peut donner un nom à une requête, puis l'utiliser comme une table dans une autre requête.

- Une vue est une table virtuelle qui permet de regrouper des champs de plusieurs table en une seule.

- De simplifier les requêtes SQL.

- D'apporter une sécurité dans la manipulation des données qui se font via les vue et non plus directement à partir des tables.

- De permettre de montrer aux utilisateurs que les données qui les concernent et de cachés les schémas relationnels.

- De donner des intitulés aux colonnes plus parlant que ceux des tables, et parfois de traduire dans différentes langue ou langages métiers.

On utilise pour cela la clause SQL suivante :

CREATE OR REPLACE VIEW nomVue
(nomcolvue1,nomcolvue2,….)
AS
SELECT col1,col2, ….
FROM Nomtables
WHERE conditions

Les nouveaux intitulés de colonnes donnés lors de la définition de la vue pourront être réutilisés comme n'importe quel nom de colonne.

Les Vues permettent d'effectuer des requêtes INSERT, DELETE et UPDATE sur plusieurs tables à la fois, à conditions d'utiliser des déclencheurs.

Accélérer les recherche grâce aux Index

Un Index Oracle permet d'accélérer l'accès aux données d'une table.
Le but principal d'un index est d'éviter de parcourir une table séquentiellement du premier enregistrement jusqu'à celui visé.

Plusieurs type d'index sont proposés par Oracle :

- L'arbre équilibré (B-tree)
- Inverse (Reverse Key) qui concerne les tables clustérisées.
- Chaîne de bits (Bitmap) qui regroupe chaque valeur de la ou des colonnes indexées sous la forme d'une chaîne de bits.
- Basés sur les calculs entre colonnes (function-based indexes)

CREATE INDEX {UNIQUE|BITMAP} [schéma.]nomIndex
ON [schéma.]nomTable ({colonne1 | expressionCol1} [ASC | DESC] ….) ;

UNIQUE permet de créer un index qui ne supporte pas les doublons.
BITMAP fabrique un index « chaînes de bits »
ASC et DESC précisent l'ordre (croissant ou décroissant).

Exemple :

CREATE UNIQUE INDEX *Idx_compte* ***ON*** *CompteEpargne (titulaire* ***DESC****) ;*

Index B-tree ordre inverse

CREATE INDEX *idx_debitFF* ***ON*** *CompteEpargne (debit*6.55957) ;*

Index B-tree expression d'une colonne

CREATE BITMAP INDEX *Idx_Bitmap_txInt_CompteEpargne* ***ON*** *CompteEpargne (txInt) ;*

Index Bitmap

CREATE INDEX *Idx_fct_Solde_CompteEpargne* ***ON*** *CompteEpargne ((credit-Debit)*(1+(txInt/100))-agios, credti,debit,txInt,agios) ;*

Index basé sur une fonction

Mettre en place les déclencheurs

La syntaxe des déclencheurs sous Oracle peut être plus ou moins complexe.

```
CREATE [ OR REPLACE ] TRIGGER schéma.nom_déclencheur
AFTER | BEFORE { [ INSERT [ OR DELETE
          [ OR UPDATE OF nom_colonne,...,nom_colonneN ] ] ] }
```

```
ON schéma.nom_table
FOR EACH ROW nom_procédure (argument...argumentN);
```

La seconde syntaxe se révèle bien plus absconse, puisqu'elle intègre non seulement les déclencheurs LMD mais également d'autres types de **déclencheurs basés sur des commandes LDD ou *DATABASE***.

En outre, pour les déclencheurs LMD, **des clauses spécifiques aux vues** apparaissent, ainsi que **la clause *REFERENCING***.

```
CREATE [ OR REPLACE ] TRIGGER [ schéma. ] nom_déclencheur
{
  AFTER | BEFORE | INSTEAD OF
}
{
  {
  [ INSERT
  [ OR DELETE
  [ OR UPDATE [ OF nom_colonne [, nom_colonneN ] ] ] ] ]
  }
  ON
  {
   { [ schéma. ] nom_table }
       |
   { [ NESTED TABLE colonne_emboîtée OF ] [ schéma . ] nom_vue }
  }
  {
   [ REFERENCING ]
   {
        [ OLD [ AS ] ancienne_colonne
        | NEW [ AS ] nouvelle_colonne
        | PARENT [ AS ] colonne_parente ]
   }
  }
  [ FOR EACH ROW ] instruction_SQL;
}
  |
{
  {
  { Evénement_LDD [ OR EvénementN_LDD ] }
     |
  { Evénement_Base_Données [ OR EvénementN_Base_Données ]
  }
  ON
  {
  { [ schéma. ] SCHEMA } | DATABASE
  }
  WHEN ( Condition )
   { instruction_PL/SQL | instruction_procédure };
}
```

La commande *OR REPLACE* recrée le déclencheur s'il existe déjà.

La clause *BEFORE* indique que le déclencheur doit être lancé avant l'exécution de l'événement.

La clause *AFTER* indique que le déclencheur doit être lancé après l'exécution de l'événement.

Les instructions *INSERT* et *DELETE* indique au déclencheur de s'exécuter lors respectivement d'une insertion ou d'une suppression dans la table.

La clause *UPDATE OF* indique que le déclencheur doit être lancé lors de chaque mise à jour d'une des colonnes spécifiées. Si elle est omise, n'importe quelle colonne de la table modifiée provoque le déclenchement du *Trigger*.

La clause *ON* désigne le nom de la table associé à son schéma pour lequel le déclencheur a été spécifiquement créé.

La clause *FOR EACH ROW* désigne le déclencheur pour être un déclencheur de ligne. Oracle lance un déclencheur de ligne une fois pour chaque ligne qui est affectée par l'instruction de déclenchement. Si la clause est omise, le déclencheur est un déclencheur d'instruction. Oracle lance un déclencheur d'instruction une fois seulement lorsque l'instruction déclenchante est émise si la contrainte du déclencheur optionnelle est rencontrée.

La clause *REFERENCING* permet de spécifier des noms de corrélation. Il est possible d'utiliser les noms de corrélation dans des blocs d'instructions PL/SQL et dans la condition WHEN d'un déclencheur de ligne pour se référer spécifiquement à des valeurs anciennes et nouvelles de la ligne courante. Les noms de corrélation par défaut sont *OLD* et *NEW*. Si le déclencheur de ligne est associé à une table nommée *OLD* ou *NEW*, l'utilisation de cette clause permet de spécifier des noms de corrélations différents afin d'éviter une confusion entre les noms de table et les noms de corrélation.
Si le déclencheur est défini sur une table imbriquée, *OLD* et *NEW* se réfèrent à la ligne de la table imbriquée, et *PARENT* se réfère à la ligne courante de la table parente.
Si le déclencheur est défini sur une table ou une vue, *OLD* et *NEW* se réfère aux instances d'objet.
La clause *REFERENCING* n'est pas valide avec les déclencheurs *INSTEAD OF* sur les événements LDD de création.

Exemple

```
-- Premier exemple
CREATE OR REPLACE TRIGGER declencheur_suppression
  AFTER DELETE ON tbl_1
  FOR EACH ROW
  WHEN (1 = 1)
  DECLARE action_utilisateur VARCHAR2(50);
  BEGIN
  SELECT user INTO action_utilisateur FROM DUAL;
  INSERT INTO delete_log
  VALUES ('tbl_1',action_utilisateur,TO_CHAR(SYSDATE, 'DD/MON/YYYY'),
                          TO_CHAR(SYSDATE, 'HH24:MI:SS'));
  END;

-- Second exemple
CREATE TABLE tbl_1 (col_a INTEGER, col_b CHAR(20));
CREATE TABLE tbl_2 (col_c CHAR(20), col_d INTEGER);

CREATE TRIGGER declencheur_insertion
  AFTER INSERT ON tbl_1
  FOR EACH ROW
```

```
WHEN (NEW.col_a <= 10)
BEGIN
    INSERT INTO tbl_2 VALUES(:NEW.col_b, :NEW.col_a);
END;
```

Le premier exemple crée un déclencheur qui insère un champ log à l'intérieur d'une table, pour chaque ligne supprimée dans la table spécifiée.

Le second exemple crée un déclencheur qui insère un enregistrement à l'intérieur de la seconde table lorsqu' une opération d'insertion s'est accomplie dans la première table. Le déclencheur vérifie si le nouvel enregistrement possède un premier composant inférieur ou égal à 10 et si c'est le cas, inverse les enregistrements à l'intérieur de la seconde table.

Les variables spéciales *NEW* et *OLD* sont disponibles pour se référer respectivement à des nouveaux ou d'anciens enregistrements. Les deux points (:) précédent *NEW* et *OLD* dans *VALUES* sont dans ce cas obligatoires, par contre dans la clause conditionnelle *WHEN*, ils doivent être omis.

TP sur machine (Suite 2)

13) Pour accélérer les recherches sur les tables ***poste et salle***, créer des index sur ces tables portant sur les champs ***(nomSalle et typePoste)***

14) Afficher les salles ainsi que le segment correspondant dans lesquelles est installé une machine de type PC.

15) Créer une vue permettant de faire la même chose que la question 14. Faite une requête simple à partir de cette vue pour vérifier son fonctionnement.

16) Donner L'étage, nom de segment, nom des salles, numéro poste, nom du poste, type de poste, adresse du poste ou est installé le pack office 2000 et Windows 2000.

17) Même question en utilisant la vue de la question 15.

18) Compter le nombre de machines ayant des types d'OS différents.

19) Créer la table suivante :

LogInst (NomSegement,Etage,NomSalle,NomPoste,NomLogiciel,Version,DateDinst)

20) Créer un déclencheur qui permet lors de la suppression d'un logiciel dans la table Installer fourni dans la table LogInst les données correspondant à la suppression de l'installation du logiciel.