Apprendre le langage SQL par l'exemple
Partie 2 : le DML

Présentation

Le DML, c'est essentiellement quatre instructions :

- **INSERT**
  Ajout d'enregistrement à une table
- **UPDATE**
  Mise à jour d'enregistrement(s)
- **DELETE**
  Suppression d'enregistrement(s)
- **SELECT**
  Extraction de données issues d'une ou plusieurs tables ou vues

Transactions

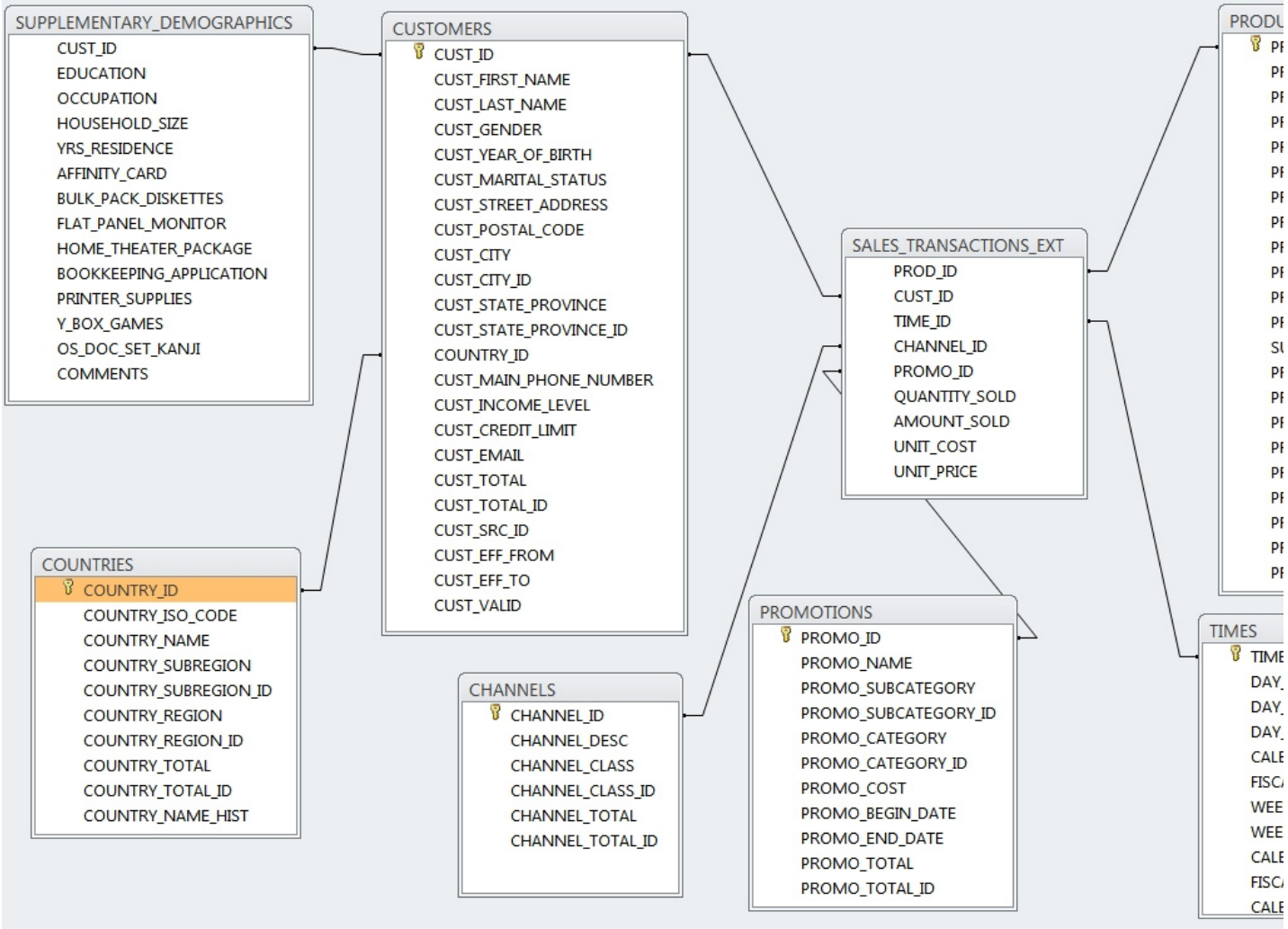
Les modifications apportées à une table par l'emploi des commandes INSERT, UPDATE ou DELETE ne sont visibles des autres utilisateurs qu'à partir de l'exécution de l'instruction **COMMIT** par l'utilisateur qui a procédé aux modifications.

A l'occasion de ces modifications, Oracle pose des **verrous** sur ligne ou sur table. Ils empêchent les autres utilisateurs d'accéder aux données modifiées, dans un souci de de **cohérence** des données du système d'information.

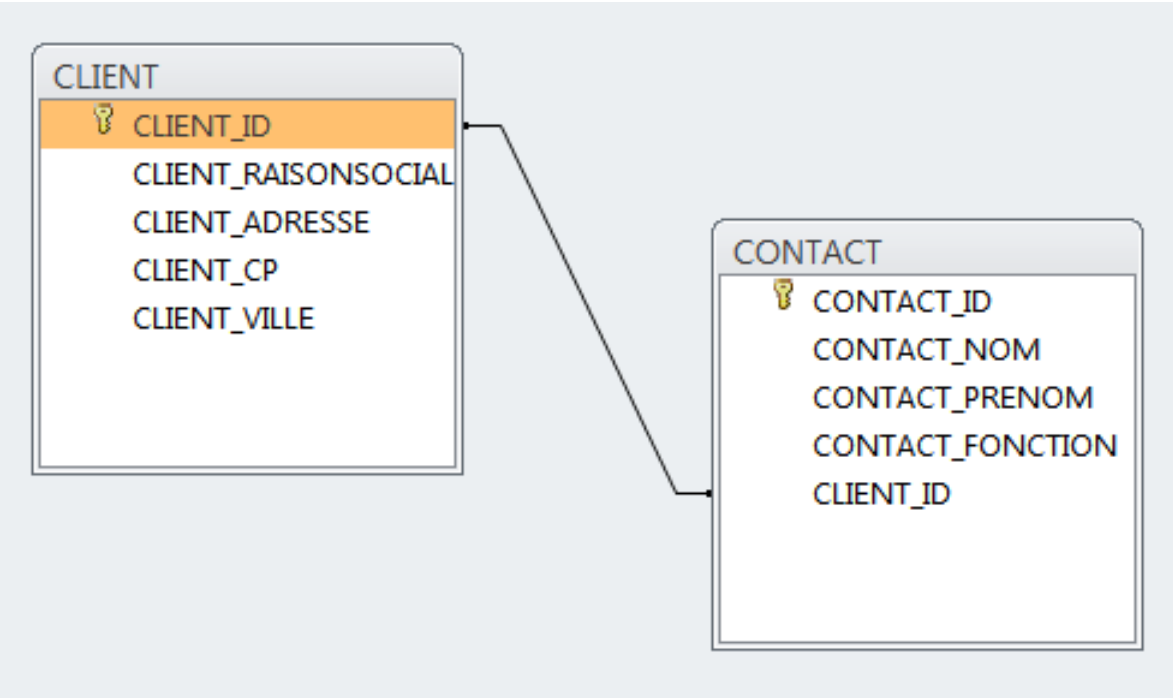
Les verrous posées au cours de la transaction sont détruits par l'exécution du COMMIT ou du **ROLLBACK**. ROLLBACK remet la base dans l'état qui prévalait avant les modifications.

Les tables du schéma SH

Dans la version 10g, la table SALES_TRANSACTIONS_EXT est remplacée par SALES !



Les tables du schéma CRM



INSERT

```
INSERT INTO sh.countries (country_id,
country_iso_code, country_name, country_subregion,
country_subregion_id, country_region,
country_region_id, country_total, country_total_id)
VALUES ('52999', 'MA', 'Maroc', 'Africa', '52792',
'Africa', '52800', 'World total', '52806')
```

Ou bien, lorsque toutes les colonnes sont renseignées dans l'ordre où elles sont présentées dans la table…

```
INSERT INTO sh.countries
VALUES ('52999', 'MA', 'Maroc', 'Africa', '52792',
'Africa', '52800', 'World total', '52806')
```

INSERT en présence d'un compteur

Lorsque la table possède un compteur (séquence + trigger), évitez de le renseigner !

```
INSERT INTO crm.client (client_raisonsociale,
client_adresse, client_cp, client_ville)
VALUES ('Winuxware', 'Les Bucailles', 27520, 'Saint
Léger du Gennetey')
```

DELETE

L'instruction DELETE s'emploie à titre principal avec la clause WHERE. Pour supprimer toutes les données d'une table de manière irréversible, vous disposez de l'instruction TRUNCATE. Pour valider l'effacement d'enregistrement, vous devez procéder, ensuite, à un COMMIT. **Plus dangereux, le TRUNCATE est beaucoup plus rapide !!!**

Mode transactionnel

```
DELETE FROM crm.contact;
COMMIT;
```

Mode non transactionnel

```
TRUNCATE TABLE crm.contact;
```

Utilisation de la clause WHERE

Suppression à partir de la valeur de la clé

```
DELETE FROM crm.client
WHERE client_id = 1;
COMMIT;
```

Suppression de plusieurs lignes

```
DELETE FROM crm.client
WHERE client_cp BETWEEN 93000 AND 93999;
COMMIT;
```

UPDATE

Mise à jour des valeurs des colonnes d'une table

```
UPDATE crm.client
SET
client_raisonsociale=UPPER(client_raisonsociale),
client_ville=UPPER(client_ville);
COMMIT;
```

Mise à jour de lignes

```
UPDATE crm.client
SET client_raisonsociale='Denis Szalkowski'
WHERE
client_raisonsociale = 'Dsfc' AND client_cp=27800;
COMMIT;
```

SELECT : la syntaxe de base

L'instruction SELECT permet d'afficher le contenu d'une table. Tapez votre instruction sur plusieurs lignes, en indentant le code SQL, afin de le rendre plus lisible. L'instruction ci-dessous affiche toutes les colonnes et toutes les lignes d'une table.

```
SELECT
  *
FROM
  sh.countries
```

Oracle n'est pas sensible à la casse au niveau des instructions et des clauses utilisées : SELECT et select, d'une part, FROM et from, d'autre part, seront interprétés de la même façon. Sous Oracle, nous avons pour habitude de taper instructions et clauses en majuscules. Les expressions introduites par l'utilisateur sont en général en minuscules.

Commentaires

Pour rendre plus compréhensible les instructions SQL, vous pouvez employer les commentaires dans vos fichiers.

```
REM Commentaire en ligne
REMARK commentaire en ligne
-- Commentaire en ligne
/*
Commentaire en bloc,
sur plusieurs lignes
*/
```

SELECT : affichage partiel

Vous pouvez, en les spécifiant, visualiser, du contenu de la table, le contenu de certaines colonnes.

```
SELECT
 country_region,
 country_name
FROM
 sh.countries
```

Alias de colonnes

Les applications utilisant des bases de données présentent très souvent des identifiants de colonnes en anglais. Vous pouvez associer un alias à ces colonnes.

```
SELECT
 country_region AS continent,
 country_name AS pays
FROM
 sh.countries
```

La clause AS est totalement facultative.

Utilisation des guillemets dans les alias de colonnes

Au niveau des alias de colonnes, Oracle n'est pas sensible à la casse. Pour obliger Oracle à afficher minuscules, majuscules ou à afficher des caractères spéciaux et des espaces dans les alias de colonnes, vous devez les encadrer de guillemets !

```
SELECT
 country_region "Continent",
 country_name "Pays"
FROM
 sh.countries
```

Tri des données

Le tri des lignes d'une table peut se faire de manière croissante ou décroissante, sur la valeur d'une ou de plusieurs colonnes. L'existence d'un index sur la colonne du tri accélère l'exécution de l'instruction SQL.

```
SELECT
 country_region,
 country_name
FROM
 sh.countries
ORDER BY
 country_region DESC,
 country_name ASC
```

Spécificité de l'emploi de la clause ORDER BY

La clause ORDER BY est la seule clause associée à L'ordre SQL qui peut utiliser les alias de colonnes en lieu et place des noms de colonnes.

```
SELECT
 country_region continent,
 country_name pays
FROM
 sh.countries
ORDER BY
 continent DESC,
 pays ASC
```

La clause DISTINCT

Pour éliminer l'affichage des doublons portant sur une, plusieurs ou la totalité des colonnes, vous pouvez utiliser la clause DISTINCT. Son utilisation trie les données dans l'ordre de présentation des colonnes définies dans l'instruction SELECT.

```
SELECT DISTINCT
 country_region
FROM
 sh.countries
```

Fonctions courantes

L'ordre SELECT permet l'utilisation de fonctions au niveau des colonnes.

- UPPER, LOWER
- SYSDATE : SELECT SYSDATE FROM DUAL affiche la date du jour
- DECODE
- TO_CHAR, TO_DATE, TO_NUMBER
- CONCAT

Ces fonctions sont utilisables également au niveau de la clause WHERE.
Les champs présentés dans l'ordre SELECT peuvent être des champs calculés.

La clause WHERE

La clause WHERE permet de filtrer les lignes selon des conditions portant sur les valeurs de colonnes.

```
SELECT
  *
FROM
  sh.customers
WHERE
  --date_of_birth between '01/01/1960' and '31/12/1970'
  --to_char(date_of_birth,'YYYY') between 1960 and 1970
  cust_year_of_birth BETWEEN 1960 AND 1970
AND
  UPPER(cust_gender)='F'
AND
  LOWER(cust_marital_status)='single'
AND
  (cust_credit_limit>12500 OR cust_credit_limit<=1500)
AND
   cust_city NOT IN ('Bordeaux','Paris', 'Berlin', 'Karlsruhe','Blatimore')
AND
   NOT cust_first_name LIKE 'D%'
```

Utilisation des expressions régulières

Les expressions régulières nous offrent d'immenses possibilités quant à l'extraction d'informations à partir de masques portant sur des chaînes de caractères.

```
SELECT
  *
FROM
  sh.customers
WHERE
  REGEXP_LIKE(cust_street_address,'^[0-9]7(.+)[aA]venue$')
```

Les regroupements statistiques

Le SQL comprend plusieurs fonctions de regroupement statistiques :

- COUNT,
- MIN, MAX,
- SUM,
- AVG,
- STDDEV,
- VARIANCE.

Compter le nombre d'enregistrements d'une table

Les fonctions de regroupement peuvent s'employer simplement.

```
SELECT
 COUNT(*)
FROM
 sh.customers
```

Vous devez compter sur une colonne renseignée, comme la clé, par exemple.

```
SELECT
 COUNT(cust_id)
FROM
 sh.customers
```

Utilisation du GROUP BY

Vous pouvez utiliser la clause GROUP BY à la manière d'un DISTINCT.

```
SELECT
  cust_city
FROM
  sh.customers
GROUP BY
  cust_city
```

Pour compter le nombre de clients par ville :

```
SELECT
  cust_city ville,
  COUNT(cust_id) total
FROM
  sh.customers
GROUP BY
  cust_city
```

Dans la clause GROUP BY, vous devez indiquer tous les champs figurant dans le SELECT, à l'exception de Ceux sur lesquels portent les statistiques.

La clause WHERE

Elle permet d'appliquer les statistiques un filtre sur les enregistrements sur lesquels seront appliqués les indicateurs statistiques.

```
SELECT
 prod_category "Catégorie",
 prod_subcategory "Sous-Catégorie",
 ROUND(AVG(prod_list_price),2) "Prix"
FROM
 sh.products
WHERE
 prod_category LIKE '%ware%'
GROUP BY
 prod_subcategory,
 prod_category
```

La clause HAVING

Elle permet d'opérer un filtre sur l'indicateur de regroupement utilisé au niveau de l'ordre SELECT.

```
SELECT
  prod_category "Catégorie",
  prod_subcategory "Sous-Catégorie",
  ROUND(AVG(prod_list_price),2) "Prix"
FROM
  sh.products
HAVING
  AVG(prod_list_price)>20
GROUP BY
  prod_subcategory,
  prod_category
ORDER BY
  "Catégorie",
  "Sous-Catégorie",
  "Prix" DESC
```

Les requêtes imbriquées

L'utilisation des requêtes imbriquées suppose l'existence d'un champ commun aux deux requêtes. Dans l'exemple ci-dessous, il s'agit de la clé primaire de la table "parent", *sh.countries*, et la clé étrangère de la table "enfant", *sh.customers*.

```
SELECT DISTINCT
  cust_city
FROM
  sh.customers
WHERE
  country_id IN
  (
    SELECT
      country_id
    FROM
      sh.countries
    WHERE
      country_name = 'France'
  )
ORDER BY
  cust_city
```


Un cas d'école : liaison réflexive

Au niveau du schéma de l'utilisateur SCOTT, toujours, vous pouvez obtenir la liste des chefs à partir de la table des employés EMP.

```
SELECT
 ename employe,
 JOB metier
FROM
 scott.emp
WHERE
 empno IN
 (
  SELECT DISTINCT
   mgr
  FROM
   scott.emp
 )
```


Les opérateurs ensemblistes

Il existe, sous Oracle, trois opérateurs ensemblistes : UNION, INTERSECT et MINUS. Ils permettent de réaliser très simplement les requêtes, sans compliquer outrageusement le contenu de la clause WHERE. Ils peuvent coûteux !

Ils permettent aussi d'agréger des contenus issus de colonnes différentes, à la condition que le nombre et le type de colonnes soient identiques dans les différentes requêtes.

Utilisation de l'opérateur UNION

La requête ci-dessous permet d'agréger les valeurs des colonnes PROD_CATEGORY et PROD_SUBCATEGORY :

```
(
SELECT
  prod_category "Libellé",
  'Catégorie' "Type"
FROM
  sh.products
UNION
SELECT
  prod_subcategory "Libellé",
  'Sous-catégorie' "Type"
FROM
  sh.products
)
ORDER BY
  "Type",
  "Libellé"
```


Utilisation de l'opérateur MINUS

Evidemment, l'instruction pourrait être réalisée avec la clause WHERE, sans utiliser l'opérateur MINUS. Avouons, toutefois, que l'utilisation de l'opérateur MINUS est très élégante, même si elle peut être fort coûteuse !

Exemple : toutes les femmes de la table SH.CUSTOMERS non mariées

```
SELECT
  cust_first_name || ' ' || cust_last_name client
FROM
  sh.customers
WHERE
  cust_gender = 'F'
MINUS
SELECT
  cust_first_name || ' ' || cust_last_name client
FROM
  sh.customers
WHERE
  cust_marital_status = 'married'
```

Utilisation de l'opérateur INTERSECT

Exemple : toutes les femmes de la table SH.CUSTOMERS mariées

```
SELECT
  cust_first_name || ' ' || cust_last_name client
FROM
  sh.customers
WHERE
  cust_gender = 'F'
INTERSECT
SELECT
  cust_first_name || ' ' || cust_last_name client
FROM
  sh.customers
WHERE
  cust_marital_status = 'married'
```

Les jointures

Naguère réalisées au travers de la clause WHERE, les jointures, depuis la normalisation de 1999, se réalisent au niveau de la clause FROM utilisée au niveau de l'instruction SELECT !

Nous sommes amenés à utiliser les jointures lorsque nous voulons afficher des données issues de différentes tables. La réalisation d'une jointure entre deux tables exige la présence d'un champ du même type dans les deux tables ! En général, la jointure se fait sur la clé primaire d'un côté et sur la clé étrangère de l'autre, selon les caractéristiques définies au travers du modèle relationnel et du modèle physique des données.

Les équijointures

Dans cet exemple, nous affichons la liste des pays de SH.COUNTRIES qui ont des villes correspondantes dans SH.CUSTOMERS. On parle alors d'équijointure. Le mot INNER est facultatif.

```
SELECT DISTINCT
 co.country_name pays,
 cu.cust_city ville
FROM
 sh.countries co
 INNER JOIN sh.customers cu
 ON co.country_id = cu.country_id
```

Variante

Quand le nom du champ sur lequel porte la jointure est commun aux deux tables, vous pouvez utiliser la clause USING. L'ennui de cette syntaxe est qu'elle ne permet pas l'utilisation de la clause WHERE.

```
SELECT DISTINCT
  co.country_name pays,
  cu.cust_city ville
FROM
  sh.countries co
  JOIN sh.customers cu
  USING(country_id)
```


Quoi de plus naturel ?

Oracle propose même des jointures naturelles : vous n'avez même plus à préciser le champ commun à l'une et à l'autre. Ce type de requête est beaucoup plus lent que les deux précédentes, dès que le nombre de tables est supérieur à 2 !

```
SELECT DISTINCT
  co.country_name pays,
  cu.cust_city ville
FROM
  sh.countries co
  NATURAL JOIN
  sh.customers cu
```


Jointure externe

Dans l'exemple précédent, il peut y avoir des enregistrements de SH.COUNTRIES qui n'aient pas de correspondance dans SH.CUSTOMERS. La jointure externe permet d'obtenir tous les enregistrements de l'une des deux tables sans qu'ils aient d'enregistrements corrélés dans l'autre. Le mot OUTER est parfaitement facultatif.

```
SELECT DISTINCT
  co.country_name pays,
  cu.cust_city ville
FROM
  sh.countries co
  LEFT OUTER JOIN sh.customers cu
  USING(country_id)
```

Tester les enregistrements sans correspondance

Pour obtenir les enregistrements de la table SH.COUNTRIES sans correspondance dans SH.CUSTOMERS, il suffit d'utiliser la clause WHERE dans l'instruction.

```
SELECT DISTINCT
  co.country_name pays,
  cu.cust_city ville
FROM
  sh.countries co
  LEFT JOIN sh.customers cu
  ON co.country_id=cu.country_id
WHERE
  cu.country_id IS NULL
```

LEFT ou RIGHT ?

Vous êtes gaucher ou droitier ? Adaptez alors votre syntaxe à votre sens de lecture ! ;+) L'instruction suivante est identique, en tout point, à celle de la page 36.

```
SELECT DISTINCT
  co.country_name pays,
  cu.cust_city ville
FROM
  sh.customers cu
  RIGHT JOIN  sh.countries co
  ON cu.country_id=co.country_id
```

FULL ?

Après avoir créé un nouveau produit sans lui associer de ventes, la requête suivante vous permettra de voir qu'il existe des pays sans produit et des produits sans pays.

```
SELECT DISTINCT
  co.country_name pays,
  pr.prod_category categorie
FROM
  sh.countries co
  FULL JOIN sh.customers cu ON cu.country_id = co.country_id
  FULL JOIN sh.sales_transactions_ext sa ON cu.cust_id=sa.cust_id
  FULL JOIN sh.products pr ON sa.prod_id=pr.prod_id
WHERE
  pr.prod_category IS NULL
OR
  co.country_name IS NULL
```

Un ordre SQL au complet !

Et celui-là, il est encore très simple !

```
SELECT DISTINCT
  co.country_name pays,
  pr.prod_category categorie,
  SUM(sa.amount_sold) ca
FROM
  sh.countries co
  JOIN sh.customers cu ON co.country_id=cu.country_id
  JOIN sh.sales_transactions_ext sa ON cu.cust_id_id=sa.cust_id
  JOIN sh.products pr ON sa.prod_id_id=sa.prod_id
WHERE
  pr.prod_category LIKE '%ware%'
GROUP BY
  co.country_name, pr.prod_category
HAVING
  SUM(sa.amount_sold) > 100000
ORDER BY
  pays, ca DESC
```