Licences Mathématiques et Informatique
3ème année - Formation Initiale et par Apprentissage

Bases de données relationnelles

Polycopié de cours - SQL

Maude Manouvrier

Table des matières

Chapitre 6

SQL

De nombreux exemples et des rappels de la syntaxe du SQL sont donnés à l'adresse `http://sql.sh/`. *Un interpréteur de SQL en ligne est disponible à l'adresse* `https://www.db-fiddle.com/`

6.1 Définition et grammaire

6.1.1 Qu'est-ce que le SQL ?

SQL signifie *Structured Query Language*. Il s'agit d'un langage de requête qui exprime sous forme logique ce que l'on souhaite obtenir comme résultat de requête, alors que pour rappel l'algèbre relationnelle [21] exprime comment le SGBD calcule le résultat d'une requête sous la forme d'opérations ensemblistes sur des ensembles de nuplets.

Il existe plusieurs révisions du langage SQL. SQL-92, noté également SQL2, est un standard adopté en 1992 [7]. SQL3 (également appelé SQL :1999) est une extension de ce standard qui prend en compte en particulier les déclencheurs ou *triggers* en anglais (des mini-programmes qui s'exécutent (se déclenchent) suite à un événement tel qu'une insertion de nuplet par exemple). Le standard a encore évolué par la suite. La dernière révision date de 2020. Ce standard, en particulier après la révision de 1992, n'est pas toujours implémenté (en entier du moins) ou pas toujours implémenté de la même manière sur les SGBD, qu'ils soient gratuits ou non. Il font donc se référer à la documentation du SGBD (pour les déclencheurs notamment).

SQL est à la fois :

- un langage de manipulation de données (DML) qui permet d'interroger et de modifier les données d'une base de données existante (cf. les sites [1] de quizz en ligne),
- un langage de définitions de données (DDL) qui permet de définir la structure d'une base de données (de la créer),
- et un langage de contrôle d'accès aux données (i.e. les droits d'accès utilisateur).

6.1.2 Grammaire SQL

- SQL manipule principalement des variables de type nuplet, des relations (tables) et des attributs (et leurs valeurs) [11].

1. `http://webtic.free.fr/sql/exint/q1.htm` et `https://eric.univ-lyon2.fr/~jdarmont/tutoriel-sql/`

- SQL ne tient pas compte des minuscules et des majuscules, du moins au niveau des noms de variables nuplet et des noms des attributs (mais attention cela peut varier d'un SGBD à un autre pour la prise en compte des maj. et min. pour les valeurs d'attributs). Les accents ne sont pas acceptés.
- Une instruction SQL peut être répartie sur plusieurs lignes (pour plus de lisibilité).
- Un nom de variable nuplet est un mot commençant obligatoirement par une lettre de l'alphabet. Il ne doit pas figurer dans la liste des mots-clés du SQL.
- Un nom de relation (table) peut-être précédé d'un nom de schéma ou d'utilisateur.
- Un nom d'attribut (colonne) est précédé du nom de la table suivi d'un point. Le nom de la relation (table) peut être omis lorsqu'il n'y a pas d'ambiguïté sur la relation (table) à laquelle appartient l'attribut.

6.2 Langage de manipulation de données

6.2.1 Requête d'interrogation : instruction SELECT

Syntaxe :

```
SELECT [DISTINCT] *
 FROM table_1 [variable_1], table_2 [variable_2], ... ;

SELECT [DISTINCT] exp_1 [AS nom_1], exp_2 [AS nom_2], ...
 FROM table_1 [variable_1], table_2 [variable_2], ... ;
```

Où :

- `DISTINCT` signifie que les doublons seront supprimés.
- `*` signifie que toutes les colonnes de la relation (table) sont sélectionnées.
- `exp_i` est une expression. Il peut s'agir d'un nom d'attribut ou d'une expression (opération arithmétique) sur des noms d'attributs. Le `AS` permet de renommer l'expression (uniquement pour l'affichage).
- `table_i` est un nom de table et `variable_i` est nom facultatif donné à un nuplet de la table pour le temps de la sélection.

Exemples de requêtes d'interrogation :

- Les identifiants de étudiants :
```
SELECT Etudiant_ID
 FROM Etudiant ;
```
- Coût des cours en euros (chaque unité valant 150 euros) :
```
SELECT Titre, NbSeances*150
 FROM Cours;
```
- Même requête que précédemment avec un renommage :
```
SELECT Titre, NbSeances*150 AS Cout
 FROM Cours;
```

Des exemples sont également donnés sur le transparent 101.

6.2.2 Spécification d'un critère de sélection dans la clause WHERE

Syntaxe :

```
SELECT [DISTINCT] *
 FROM table_1 [variable_1], table_2 [variable_2], ...
```

```
WHERE prédicat1
 AND [ou OR] prédicat2
 AND [ou OR] ... ;
```

Prédicat simple

Un prédicat est de la forme :

```
exp1 = exp2
exp 1 != exp2
exp1 > exp2
exp1 < exp2
exp1 >= exp2
exp1 <= exp2
exp1 BETWEEN exp2 AND exp3
exp1 LIKE exp2
exp1 NOT LIKE exp2
exp1 IN (exp2, exp3, ...)
exp1 NOT IN (exp2, exp3, ...)
exp1 IS NULL
exp1 IS NOT NULL
```

Par exemples :

- Noms des étudiants habitant Paris :
```
SELECT Nom, Prenom
 FROM Etudiant
  WHERE Ville = 'Paris';
```
- Noms des étudiants habitant Paris et dont le nom de famille contient la lettre 'A' en 2eme position et la lettre 'M' en 3ème position :
```
SELECT Nom, Prenom
 FROM Etudiant
  WHERE Ville = 'Paris'
   AND Nom LIKE '_AM%';
```
Le caractère % sert de caractère générique. Il est équivalent à zéro ou plusieurs caractères [16]. Le caractère _ sert à remplacer un unique caractère (n'importe lequel).
- Noms des étudiants n'ayant pas de FAX :
```
SELECT Nom, Prenom
 FROM Etudiant
  WHERE Fax IS NULL;
```
Attention : pour tester si un attribut est (ou n'est pas) `NULL`, il faut utiliser `IS NULL` ou `IS NOT NULL` et non pas le symbole d'égalité ou de différence.
- Titres des cours dont le coût est compris entre 1000 et 2000 euros :
```
SELECT Titre
 FROM Cours
  WHERE (NbSeance *150 ) BETWEEN 1000 AND 2000;
```

- Noms des enseignants des départements 1, 2 ou 3 :
```
SELECT Nom, Prenom
 FROM Enseignant
  WHERE Department_ID IN (1,2,3);
```

Prédicat composé de sous-requête

Il est possible d'utiliser une sous requête dans la clause $WHERE$. Le prédicat de la clause $WHERE$ devient alors :

```
exp. op. ANY (SELECT ...)
exp. op. ALL (SELECT ...)
exp IN (SELECT)
exp NOT IN (SELECT ...)
```

op. est un des opérateurs $=, !=, <$, etc. *exp.* est une expression (i.e. un nom d ?attribut ou une opération arithmétique sur des noms d'attributs.
ANY retourne VRAI si la comparaison est vraie avec au moins un nuplet retourné par la requête entre parenthèses (FAUX sinon).
ALL retourne VRAI si la comparaison est vraie avec TOUS les nuplets retournés par la requête entre parenthèses (FAUX sinon).

Par exemple :

- Titres des cours dont les frais additionnels sont inférieurs à tous les frais additionnels :
```
SELECT Titre
 FROM Cours
  WHERE FraisAdditionnels <= ALL (SELECT FraisAdditionnels FROM Cours);
```
- Titres des cours dont les frais additionnels ne sont pas les plus bas (pour lesquels il existe au moins un cours avec des frais additionnels inférieurs) :
```
SELECT Titre
 FROM Cours
  WHERE FraisAdditionnels > ANY (SELECT FraisAdditionnels FROM Cours);
```
- Noms des enseignants des départements de MIDO ou DEP :
```
SELECT Nom, Prenom FROM Enseignant
  WHERE Department_ID IN (SELECT Department_ID FROM Departement
                          WHERE Nom_Departement = 'MIDO'
                           OR Nom_Departement = 'DEP');
```

6.2.3 Clause EXISTS

L'opérateur $EXISTS$ retourne VRAI, si au moins un nuplet est renvoyé par la requête située après le `EXISTS` ou FAUX si aucun nuplet n'est renvoyé par la requête située après le `EXISTS`.
Attention : la valeur NULL n'affecte pas le résultat de la clause EXISTS. Si la requête située après le `EXISTS` retourne un nuplet de valeur `(NULL)`, c'est un nuplet, la clause `EXISTS` retourne donc VRAI.

Par exemple, noms des enseignants qui n'enseignent à aucun cours :

```
SELECT Nom, Prenom
 FROM Enseignant e
  WHERE NOT EXISTS (SELECT * FROM Cours c WHERE e.Enseignant_ID = c.Enseignant_ID);
```

Il est à noter que dans cette requête deux variables nuplets sont utilisées : `e` qui représente un nuplet de la relation `Enseignant` et `c` qui représente un nuplet de la relation `Cours`.

6.2.4 Opérateurs d'agrégation

Les opérateurs d'agrégation sont : $COUNT$, SUM, AVG, MIN, MAX. Ces opérateurs se placent après le mot clé $SELECT$.

Par exemples :

- Nombre d'étudiant :
```
SELECT COUNT(*)
 FROM Etudiant ;
```
- Nombre maximum d'années passées à l'université par un étudiant :
```
SELECT MAX(NbAnnees)
 FROM Etudiant ;
```
- Somme et moyenne des frais additionnels des cours :
```
SELECT AVG(FraisAdditionnels), SUM(FraisAdditionnels)
 FROM Cours ;
```
- Cours dont frais additionnels sont supérieurs à la moyenne :
```
SELECT Nom_Departement
 FROM Cours
  WHERE FraisAdditionnels >= (SELECT AVG(FraisAdditionnels) FROM Cours);
```

6.2.5 Tri

La clause $ORDER\ BY$ permet de trier les données, sur une ou plusieurs colonnes (sinon les nuplets résultat sont affichés dans l'ordre dans lequel le SGBD les a trouvé - généralement ordonnés en fonction de la valeur de la clé primaire). Par exemples :

- Liste des enseignants triée par numéros de département, noms et prénoms :
```
SELECT Departement_ID, Nom, Prenom
 FROM Enseignant
  ORDER BY Departement_ID, Nom, Prenom;
```
- Liste des enseignants triée par numéros de département (ordre décroissant), noms et prénoms :
```
SELECT Departement_ID, Nom, Prenom
 FROM Enseignant
  ORDER BY Departement_ID DESC, Nom, Prenom;
```

Un exemple est donné sur le transparent 109.

6.2.6 Groupement des données et opérations d'agrégation sur des groupes

Le groupement de données se fait à l'aide de la clause $GROUP\ BY$ [$HAVING$]. La clause $HAVING$ permet d'exprimer une condition sur le groupe.

Par exemples :

- Nombre de cours par département :
```
SELECT Departement_ID, COUNT(*)
 FROM Cours
  GROUP BY Departement_ID;
```

- Nombre de départements ayant exactement quatre cours :
```
SELECT Departement_ID, COUNT(*)
 FROM  Cours
  GROUP BY Departement_ID
   HAVING Count(*) = 4;
```
- Moyenne des frais additionnels par département :
```
SELECT Departement_ID, AVG(FraisAdditionnels)
 FROM Cours
  GROUP BY Departement_ID;
```

Un exemple est donné sur le transparent 108 pour l'opérateur `COUNT`.

6.2.7 Jointure

Lorsque l'on précise plusieurs tables dans la clause $FROM$, on obtient le produit cartésien des tables [11] (voir l'exemple des transparents). La clause $WHERE$ permet de faire la jointure. Par exemple : Jointure entre les relations *Enseignant* et *Departement* :

```
SELECT Nom, Prenom, Nom_Departement
 FROM Enseignant e, Departement d
  WHERE e.Departement_ID = d.Departement_ID;
```

Attention, si on oublie la clause $WHERE$, le SGBD exécute un produit cartésien. Pour éviter cet oublie, il est possible d'écrire la même requête de la manière suivante :

```
SELECT Nom, Prenom, Nom_Departement
 FROM Enseignant e INNER JOIN Departement d
  ON e.Departement_ID = d.Departement_ID;
```

L'oubli du ON créera une erreur de syntaxe.

Cette requête étant une jointure naturelle ou **equijointure**, il est également possible de l'écrire de la manière suivante :

```
SELECT Nom, Prenom, Nom_Departement
 FROM Enseignant NATURAL JOIN Departement;
```

Des exemples sont donnés sur les transparents 118 à 120 et 122.

Il est possible de réaliser des **jointures externes** qui retourne également les nuplets ne satisfaisant pas la condition en utilisant des valeurs NULL. En SQL2, il s'agit de la commande OUTER-JOIN (LEFT, RIGHT ou FULL OUTER-JOIN) [7]. Par exemple, la requête :

```
SELECT *
 FROM Personnel p LEFT OUTER JOIN Employe e
  ON e.Nom_Employe = p.Nom_Employe;
```

renvoie tous les nuplets de *Personnel* joint, lorsque cela est possible, aux nuplets de *Employe* et avec des valeurs $NULL$ pour les attributs à *Employe*, lorsque la jointure n'est pas possible. La requête :

```
SELECT *
 FROM Personnel p RIGHT OUTER JOIN Employe e
  ON e.Nom_Employe = p.Nom_Employe;
```

renvoie tous les nuplets de *Employe* joint, lorsque cela est possible, aux nuplets de *Personnel* et avec des valeurs *NULL* pour les attributs à *Personnel*, lorsque la jointure n'est pas possible. La même requête avec le mot-clé *FULL* renvoie l'union des 2 précédentes requêtes.
Un exemple de résultat de ces requête est donné sur le transparent 123.

6.2.8 Opérateurs ensemblistes

Opérateur INTERSECT

Syntaxe :

```
instruction SELECT
 INTERSECT
instruction SELECT
[ORDER BY ...]
```

Attention : Les deux instructions *SELECT* ne peuvent pas contenir de clauses *ORDER BY*, et le nombre de colonnes des résultats des deux instructions doit être le même, les schémas de 2 requêtes séparées par *INTERSET* doivent être compatibles. Par exemple :

```
SELECT * FROM Enseignant WHERE Department_ID='INFO'
INTERSECT
SELECT * FROM Enseignant WHERE Department_ID='MATH'
```

Opérateur UNION

Syntaxe :

```
instruction SELECT
 UNION
instruction SELECT
[ORDER BY ...]
```

Par exemple, la requête :

```
SELECT Nom,Prenom FROM Enseignant
UNION
SELECT Nom,Prenom FROM Etudiant
```

Attention, la requête :

```
SELECT Nom,Prénom FROM Enseignant WHERE Département_ID = 'INFO'
UNION
SELECT Nom,Prénom FROM Enseignant WHERE Département_ID = 'MATH'
ORDER BY Nom,Prénom
```

n'est pas optimale. En effet, elle impose au SGBD de parcourir 2 fois la relation `Enseignant`, alors qu'il suifferait de la parcourir une seule fois avec un *OR*, comme dans la requête suivante :

```
SELECT Nom,Prénom FROM Enseignant
 WHERE Département_ID = 'INFO'
   OR Département_ID = ' MATH'
ORDER BY Nom,Prénom
```

Un exemple d'exécution de ce style de requêtes, avec une comparaison des temps d'exécution, est donné sur les transparents 138 et 139.

Opérateur EXCEPT

Cet opérateur permet d'ôter dans une sélection les lignes obtenues à partir d'une autre sélection. Par exemple :

```
SELECT Nom, Prenom FROM Etudiant
 EXCEPT
SELECT Nom, Prenom FROM Enseignant
```

6.2.9 Division exprimée en SQL

Il n'existe pas de mot clé permettant de définir la division en SQL. Pour signifier "tous", il faut utiliser la clause $NOT\ EXISTS$.

Par exemple sur les transparents 143 à 146, on demande la liste des livres empruntés par **tous** les étudiants. Cette requête s'exprime

- en algèbre relationnelle par : $\Pi_{Titre,EtudiantID}(Livre \bowtie Emprunt) \div \Pi_{EtudiantID}(Etudiant)$
- en SQL :

```
SELECT t.Titre FROM Livre t WHERE NOT EXISTS
   ( SELECT * FROM Etudiant u WHERE NOT EXISTS
      ( SELECT * FROM Emprunt v
          WHERE u.EtudiantID=v.EtudiantID AND v.ISBN=t.ISBN
      )
   );
```

  Ce qui signifie en français que l'on souhaite obtenir dans le résultat l'attribut `Titre` des nuplets `t` de `Livre`, pour lesquels il n'est pas possible de trouver un étudiant (donc un nuplet `u` dans `Etudiant`) pour lequel il n'existe pas d'emprunt de ce livre par cet étudiant (i.e. un nuplet `v` dans `Emprunt` indiquant que `t.ISBN` a été emprunté par `u.EtudiantID`). Dit de manière positive, on souhaite obtenir dans le résultat l'attributs `Titre` des livres `t` tels que pour tous les étudiants u, on trouve un emprunt du livre t par l'étudiant u.

D'autres exemples sont donnés sur les transparents 147 et 148.

6.2.10 Insertion, Suppression et modification de nuplets

INSERT

L'insertion de nuplets dans la base se fait par la commande :

```
INSERT INTO table(col1, col2, ...,coln)
 VALUES (val1, val2, ..., valn), (val'1, val'2, ..., val'n) ;
```

La liste des colonnes est optionnelles. Si elle est omise, le SGBD prendra par défaut l'ensemble des colonnes de la relation dans l'ordre où elles ont été définies lors de la création de la relation (voir section 6.3). Si une liste de colonnes est spécifiée, les colonnes ne figurant pas dans la liste auront pour valeur $NULL$, une valeur par défaut ou une valeur auto-incrémentée en fonction de la définition de l'attribut.

Pour insérer des nuplets à partir d'une requête (lorsque l'on migre une base de données par exemple, i.e. que l'on crée une nouvelle base de données à partir d'une ancienne), on peut faire :

```
INSERT INTO table(col1, col2, ...,coln)
  SELECT
```

Un exemple d'insertion est donné sur le transparent 150.

UPDATE

La modification de nuplets se fait par la commande :

```
UPDATE table
 SET col1 = exp1, col2 = exp2 ...
  WHERE prédicat
```

La clause $WHERE$ indique quelles sont les lignes à modifier. Elle est facultative. Si elle n'est pas écrite, toutes les lignes de l'instance sont modifiées.

Un exemple de mise à jour est donné sur les transparents 150 et 151.

DELETE

La suppression de nuplets se fait par la commande :

```
DELETE FROM table
 WHERE prédicat
```

La clause $WHERE$ indique quelles sont les lignes à supprimer. Elle est facultative. Si elle n'est pas écrite, toutes les lignes de l'instance sont supprimées.

Des exemples de suppressions sont donnés sur le transparent 150.

Transactions

Une transaction est un ensemble de modifications qui forment un tout indivisible [11].

Pour valider une transaction, on utilise la commande $COMMIT$. Après la commande $COMMIT$, les modifications deviennent définitives et visibles à toutes les autres transactions.

En cas d'erreur, une transaction peut être annulée par la commande $ROLLBACK$. Toutes les modifications de la transactions seront annulées. Lorsqu'il y a une erreur, un $ROLLBACK$ est automatiquement effectué.

Une transaction commence après une instruction $COMMIT$, une instruction $ROLLBACK$ ou une connexion initiale [16], et se termine lorsque n'importe quel événement se produit :

- Une instruction $COMMIT$,
- Une instruction $ROLLBACK$,
- La fermeture de la session de base de données.

Il est également possible de placer des *points d'enregistrements*, par la commande :
`SAVEPOINT` *nom_du_point_d'enregistrement*.
Les points d'enregistrements permettent de défaire une transaction jusqu'à un une phase intermédiaire :
`ROLLBACK TO` *nom_du_point_d'enregistrement*.

Un exemple de transaction est donné sur le transparent 151.

6.3 Langage de définition de données

6.3.1 Création de tables

CREATE TABLE et PRIMARY KEY

La création de table se fait en SQL par la commande :

```
CREATE TABLE table (col1 type1 [NOT NULL], ...,
                    coln type n [NOT NULL] ...
                    CONSTRAINTS nom_contraine PRIMARY KEY(...));
```

Par exemple :

```
CREATE TABLE Departement
(Department_ID     Varchar2(20),
 Department_Name   Varchar2(25),
 CONSTRAINT PK_Department PRIMARY KEY (Department_ID));
```

PK_Department est le nom de la contrainte qui définit la clé primaire de la table *Departement*. Il est possible, lorsque la clé primaire n'est composée que d'un seul attribut, de le spécifier immédiatement après cet attribut. Par exemple :

```
CREATE TABLE Departement
(Department_ID     Varchar2(20) PRIMARY KEY,
 Department_Name   Varchar2(25)
);
```

Contraintes sur les valeurs non nulles

Il est possible de spécifier d'autres contraintes, comme les valeurs non nulles :

```
CREATE TABLE Departement
(Department_ID     Varchar2(20),
 Department_Name   Varchar2(25) NOT NULL
 CONSTRAINT NN_Department_Name NOT NULL,
 CONSTRAINT PK_Department PRIMARY KEY (Department_ID));
```

Par défaut, tout attribut, sauf ceux de la clé primaire, peuvent prendre la valeur nulle.

Clé étrangère

Il est également possible de définir les clés étrangères :

```
CREATE TABLE Enseignant
(
 Enseignant_ID     integer,
 Departement_ID    integer NOT NULL,
 Nom            varchar(25) NOT NULL,
 Prenom            varchar(25) NOT NULL,
 Telephone         varchar(10) DEFAULT NULL,
 Fax               varchar(10) DEFAULT NULL,
```

```
 Email             varchar(100) DEFAULT NULL,
 CONSTRAINT PK_Enseignant PRIMARY KEY (Enseignant_ID),
 CONSTRAINT "FK_Enseignant_Departement_ID" FOREIGN KEY (Departement_ID)
    REFERENCES Departement (Departement_ID)
);
```

Il est possible de définir des actions associées aux clés étrangères :

```
CONSTRAINT nom_contrainte FOREIGN KEY (attributs)
   REFERENCES table(attributs)
    [ON DELETE {RESTRICT | CASACADE | SET DEFAULT | SET NULL}]
    [ON UPDATE {NO RESTRICT | CASACADE | SET DEFAULT | SET NULL}]
```

L'option `ON DELETE` indique que l'action à effectuer sur les nuplets référençants (ceux pour lesquels est définie la clé étrangère) lors d'une suppression dans la table référencée. `RESTRICT` indique qu'une suppression d'un nuplet dans la table référencée (ex. `Departement`) ne sera autorisée que si aucun nuplet n'y font référence dans la table référençant (ex. `Enseignant`). `CASACDE` signifie que la suppression se fera en cascade dans la table référençant. `SET DEFAULT | SET NULL` modifie la valeur de la clé étrangère en la remplaçant par une valeur par défaut (si elle a été définie par $DEFAULT$) ou la valeur `NULL` (si l'attribut n'est pas `NOT NULL`). L'option `ON UPDATE` indique l'action à effectuer sur la clé étrangère en cas de mise à jour de la clé primaire [7].

Contraintes de domaine

La contrainte $CK\_Enseignant\_Grade$ du transparent 164 est une contrainte de domaine. Il est possible d'utiliser une égalité ou une inégalité, ou encore le résultat d'une requête. Par exemple, si on a crée une relation `PositionEnseignant` :

```
CREATE TABLE Enseignant
(
 Enseignant_ID     integer,
 Departement_ID    integer NOT NULL,
 Nom            varchar(25) NOT NULL,
 Prenom            varchar(25) NOT NULL,
 Grade             varchar(25)
 CONSTRAINT CK_Enseignant_Grade
   CHECK (Grade IN (SELECT Position FROM PositionEnseignant)),
 ...
);
```

D'autres exemples de contraintes de domaine sont données sur les transparents 167 et 168.

6.3.2 Modification de la structure d'une table existante

Le schéma d'une table peut être modifié par la commande $ALTER\ TABLE$.

Ajout d'une colonne

```
   ALTER TABLE table
    ADD (col1 type1, col2 type 2,...);
```

La condition $NOT\ NULL$ ne peut être utilisée que si la table est vide, sinon les nouvelles colonnes seront nulles et la clause $NOT\ NULL$ ne pourra pas être appliquée.

Modification de colonne

```
ALTER TABLE table
 MODIFY (col1 type1, col2 type 2,...);
```

*col*1, *col*2 sont les noms des colonnes que l'on souhaite modifier. Elles doivent exister dans la table.

Suppression de contraintes

```
ALTER TABLE table
 DROP PRIMARY KEY;

ALTER TABLE table
 DROP CONSTRAINT nom_contrainte;
```

Des exemples sont donnés dans les transparents 191 et 192.

6.3.3 Suppression d'une table

Une table est supprimée par la commande $DROP\ TABLE$.

6.3.4 Vues

Il est possible d'enregistrer un ordre $SELECT$ dans une **vue**. Seule la définition en SQL et le plan d'exécution de la vue est stockée dans la base. Les données de la vue seront calculées à chaque utilisation de la vue par un utilisateur.

Une vue permet de restreindre la vision que certains utilisateurs ont de la base (restriction des droits d'accès à certaines colonnes et à certaines lignes de la table). Une vue permet également de simplifier la consultation de la base en enregistrant des $SELECT$ complexes [11].

Création de vue

Une vue est créée à partir de la commande :

```
CREATE VIEW NomVue (col1, col2, ..)
 AS SELECT ...
```

Le noms des colonnes de la vue est facultative, par défaut il s'agit de ceux du résultat de la requête.

Suppression de vue

```
DROP VIEW NomVue
```

Interrogation des vues

Les vues s'utilisent comme n'importe quelle autre table de la base, au niveau de l'interrogation.

6.3.5 Vue matérialisée

Les vues matérialisées sont des vues dont les données sont matérialisées, c'est-à-dire stockées. Elle est utilisée à des fins d'optimisation. Elle se crée de la manière suivante : Une vue est créée à partir de la commande :

```
CREATE MATERIALIZED VIEW NomVue (col1, col2, ..)
 AS SELECT ...
```

Les données d'une vue matérialisée ne sont pas forcément fraîches. Pour les rafraîchir il faut faire l'instruction :

```
REFRESH MATERIALIZED VIEW NomVue;
```

6.3.6 Index

Il est possible d'accélérer le temps de réponse d'une requête en définissant des index. Un index est mis à jour automatiquement lors de l'ajout de données dans la table indexée. Les index sont utilisés par l'optimiseur de requête.

Un index est créé par la commande :
`CREATE [UNIQUE] INDEX` *nom_index* `ON table (col1, col2, ...`
L'option $UNIQUE$ signifie que chaque valeur d'index doit être unique dans la table.

Lorsque l'utilisateur définit une contrainte de type $Primary\ Key$, le système crée par défaut un index $UNIQUE$ sur la clé primaire (empêchant ainsi toute insertion dupliquée de clé).

Un index est supprimé par la commande : `DROP INDEX` *nom_index*.

6.3.7 Déclencheurs

Un déclencheurs est un programme qui s'exécute avant ou après un événement (insertion, mise à jour ou suppression) sur une relation. Il permet d'afficher un message à l'utilisateur ou d'insérer, mettre à jour ou supprimer des nuplets dans l'instance ou dans une autre instance de la bases de données. On peut manipuler le nupet en cours d'insertion, si l ?événement est une insertion, le nuplet avant et après mise à jour, si l ?événement est une modification, et le nuplet en cours de suppression, si l ?événement est une suppression. Des exemples sont donnés dans les transparents 180 à 187, ainsi que dans le polycopié de TP et la correction de certains sujets d'examens sur moodle.

Bibliographie

[1] D. Austin, *Using Oracle8^{TM}*, Simple Solutions - Essential Skills, QUE, 1998, ISBN : 0-7897-1653-4

[2] R. Chapuis, *Oracle 8*, Editions Dunes et Laser, 1998, ISBN : 2-913010-07-5

[3] P. Chen, *The Entity-Relationship Model–Toward a Unified View of Data*, ACM Transactions on Database Systems, Vol. 1, No. 1, March 1976, Pages 9-36, http ://www.csc.lsu.edu/∼chen/pdf/erd.pdf

[4] T. Connolly, C. Begg et A. Strachan, *Database Systems - A pratical Approach to Design, Implementation and Management*, Addison-Wesley, 1998, ISBN : 0-201-34287-1, disponible à la BU 055.7 CON

[5] C.J. Date, *Introduction aux bases de données*, 6ème édition, Thomson Publishing, 1998, ISBN : 2-84180-964-1, disponible à la BU 005.7 DAT

[6] R. Elamsri et S.B. Navathe, *Fundamentals of Database Systems*, 3ème édition, Addison Wesley- disponible à la BU 005.7 ELM

[7] P. Delmal, *SQL2 - Application à Oracle, Access et RDB*, 2ème édition, Bibliothèque des Universités - Informatique, De Boeck Université, 1988, ISBN : 2-8041-2995-0,disponible à la BU 005.74 SQL

[8] S. Feuerstein, B. Pribyl et C. Dawes, *Oracle PL/SQL - précis et concis*, O'Reilly, 2000, ISBN : 2-84177-108-3

[9] G. Gardarin, *Bases de Données - objet & relationnel*, Eyrolles, 1999, ISBN : 2-212-09060-9, disponible à la BU 005.74 GAR

[10] R. Grin, *Introduction aux bases de données, modèle relationnel*, Université Sophia-Antipolis, jan. 2000

[11] R. Grin, *Langage SQL*, Université Sophia-Antipolis, jan. 2000

[12] G. Gardarin et O. Gardarin *Le Client-Serveur*, Eyrolles, 1999, disponible à la BU

[13] H. Garcia-Molina, J.D. Ulmann et J. Widow, *Database SYstem Implementation*, Prentice Hall, 2000, ISBN :0-13-040264-8, disponible à la BU 005.7 GAR

[14] H. Garcia-Molina, J.D. Ulmann et J. Widow, *Database Systems - The Complete Book*, Prentice Hall, 2002, ISBN :0-13-031995-3

[15] S. Krakowiak, *Gestion Physique des données*, Ecole Thématique "Jeunes Chercheurs" en Base de Données, Volume 2, Port Camargue, mars 1997

[16] D. Lockman, *Oracle8^{TM} Dévéloppement de bases de données*, Le programmeur - Formation en 21 jours, Editions Simon et Schuster Macmillan (S&SM), 1997, ISBN : 2-7440-0350-6, disponible à la BU 005.74 ORA

[17] P.J. Pratt, *Initiation à SQL - Cours et exercices corrigés*, Eyrolles, 2001, ISBN : 2-212-09285-7

[18] R. Ramakrishnan et J. Gehrke, *Database Management Systems*, Second Edition ; McGraw-Hill, 2000, ISBN : 0-07-232206-3, disponible à la BU 055.7 RAM

[19] A. Silberschatz, H.F. Korth et S. Sudarshan, *Database System Concepts*, Third Edition, McGraw-Hill, 1996, ISBN : 0-07-114810-8, disponible à la BU 005.7 DAT

[20] C. Soutou, *De UML à SQL - Conception de bases de données*, Eyrolles, 2002, ISBN : 2-212-11098-7

[21] J.D. Ullman et J. Widom, *A first Course in Database Systems*, Prentice Hall, 1997, ISBN : 0-13-887647-9, disponible à la BU 005.7 ULL