Le langage SQL : Le Langage de Définition des Données (LDD)

Sources du document :

- Livre bible Oracle 9i, J. Gabillaud, Editions Eni ;
- Support Oracle ;
- ISO Norme 2382:1999 parties 1 à 5, ISO Norme 9075 parties 1 à 14 corrigées en 2005.

Sources du document :

- Site sql.developpez.com ;
- Manuel SQL PostgreSQL (existe en plusieurs versions HTML, PDF…).

Olivier Mondet
http://unidentified-one.net

A. Introduction, notion de contrainte d'intégrité

Le langage de définition des données regroupe l'ensemble des possibilités permettant de créer une base de données, de décrire des données : tables, vues, colonnes, contraintes, index…

Les tables obéissent à une logique de création qui doit respecter des règles strictes. On associe des contraintes d'intégrité au contenu des tables.

Une contrainte d'intégrité est un automatisme du SGBD qui garantit que les valeurs d'une colonne ou d'un groupe de colonnes satisfont à une condition déclarée.

Les contraintes assurent la cohérence des données (concept d'intégrité). Elles sont décrites lors de la définition des tables de la base (ordre CREATE TABLE).

Types de contraintes :
- contrainte de valeurs (contrainte de domaine) ;
- unicité d'occurrences (unicité de lignes) ;
- clé primaire (unicité et indexation pour l'intégrité d'entité) ;
- clé étrangère (intégrité référentielle) ;
- caractère obligatoire ou facultatif des valeurs.

Les contraintes peuvent s'exprimer :
- soit au niveau colonne (contraintes locales) : valables pour une colonne ;
- soit au niveau table (contraintes globales) : valables pour un ensemble de colonnes d'une table.

Les contraintes se définissent :
- soit lors de la création des tables, dans l'ordre CREATE TABLE ;
- soit après la création des tables, par l'ordre ALTER TABLE permettant certaines modifications de la structure des tables.

Chaque contrainte définie est affectée d'un nom propre (nom de contrainte) stockée dans le dictionnaire des données de la base, implicitement par le SGBD (clé primaire) ou explicitement par la clause CONSTRAINT.

Les contraintes peuvent être activées (ENABLE) ou désactivées (DISABLE).

Les contraintes de colonnes :
- NULL ou NOT NULL (caractère obligatoire/facultatif) ;
- UNIQUE (unicité des valeurs) ;
- PRIMARY KEY (clé primaire élémentaire) ;
- REFERENCES (intégrité de référence) ;
- CHECK (contraintes de valeurs).

Les contraintes de tables :
- UNIQUE (composition des colonnes) ;
- PRIMARY KEY (clé primaire composée) ;
- FOREIGN KEY… REFERENCES… (composition de colonnes) ;
- CHECK (condition particulière sur les valeurs).

B. L'instruction CREATE TABLE

B.1. Normes

La norme SQL89/1 est une version simplifiée de CREATE TABLE. Cette norme rend fragile l'intégrité des données en imposant peu de contraintes.

La norme SQL92 incorpore directement la définition des contraintes d'intégrité, ce qui impose une formulation rigoureuse.

B.2. Syntaxe de l'instruction

```
CREATE TABLE [propriétaire.]nom_table
(col1 TYPE(taille) [DEFAULT expression] [NOT NULL],
col2 TYPE(taille) [DEFAULT expression] [NOT NULLl],
col3 TYPE(taille) [DEFAULT expression] [NOT NULL] ,
...
[PRIMARY KEY (col n, col M...)]
[FOREIGN KEY (col N, col M...) REFERENCES nom_table2 (col I)]
);
```

Le nombre de colonnes possibles dans une table est compris entre 1 et 254.
L'indicateur NOT NULL indique que l'attribut devra impérativement être valué.
[DEFAULT] spécifie une valeur qui sera affectée à cette colonne si, lors d'un INSERT, on ne lui en précise pas.
La clause PRIMARY KEY permet de déterminer la clef primaire de la table.
La clause FOREIGN KEY permet de définir une contrainte d'intégrité référentielle avec la clef primaire d'une autre table.

Exemple 1 :
```
CREATE TABLE ELEVE
(NOM CHAR(20) PRIMARY KEY,
PNOM CHAR(20) NOT NULL,
NOTE NUMBER(4,2) DEFAULT 0,
NUMCLASSE CHAR(4),
TELEPHONE CHAR(10) NOT NULL UNIQUE,
FOREIGN KEY (NUMCLASSE) REFERENCES CLASSE (CODECLASSE)
);
```

La clé primaire est déclarée dans ce cas par contrainte de colonne. Le cas suivant montre une déclaration de clé primaire par contrainte de table.

Attention à l'ordre de création des tables : une clé étrangère ne peut pas faire référence à la clé primaire d'une autre table qui n'existe pas.

Exemple 1 :

```
CREATE TABLE ELEVE
(NOM CHAR(20) NOT NULL,
PNOM CHAR(20) NOT NULL,
NOTE NUMBER(4,2),
NUMCLASSE CHAR(4),
TELEPHONE CHAR(10) NOT NULL UNIQUE,
PRIMARY KEY (NOM),
FOREIGN KEY (NUMCLASSE) REFERENCES CLASSE (CODECLASSE)
);
```

Exemple 2 :

```
CREATE TABLE APPAREIL
(CODETYPE VARCHAR2(3) CONSTRAINT PK_APPAREIL PRIMARY KEY,
NBPLACE INTEGER CONSTRAINT NBPLACE_APPAREIL
CHECK (NBPLACE >=0 AND NBPLACE <=500),
DESIGN VARCHAR2(25)
);
```

Exemple 3 :

```
CREATE TABLE ETUDE
        (NUMEROMAT INTEGER,
        CODECLASSE VARCHAR2(6),
        COURS INTEGER,
        TD INTEGER,
        CONSTRAINT PK_ETUDE PRIMARY KEY (NUMEROMAT, CODECLASSE),
        CONSTRAINT FK_ETUDE_ELEVE FOREIGN KEY (NUMEROMAT)
                REFERENCES MATIERE (NUMAT),
        CONSTRAINT FK_ETUDE_CLASSE FOREIGN KEY (CODECLASSE)
                REFERENCES CLASSE (CODECLASSE)
        );
```

Les contraintes de colonne :

```
[CONSTRAINT nom_contrainte]
[NOT] NULL / UNIQUE / PRIMARY KEY / REFERENCES [propriétaire.]nom_table
[ON DELETE CASCADE]
/ CHECK (condition)
```

Les contraintes de table :

```
[CONSTRAINT nom_contrainte]
UNIQUE / PRIMARY KEY (colonne1 [, colonne2]…) / FOREIGN KEY (colonne1 [, colonne2] …)
REFERENCES [propriétaire.]nom_table [(colonne1 [, colonne2] …)
[ON DELETE CASCADE]
/ CHECK (condition)
```

La clause [ON DELETE CASCADE] signifie que tout sera supprimé en cascade lors d'une suppression de table.

La table peut éventuellement être créée à partir d'une autre table :

```
CREATE TABLE ELEVE
AS SELECT nom, prenom
FROM etudiants
WHERE ...
```

La nouvelle table aura donc deux colonnes NOM et PRENOM et sera remplie avec les valeurs sélectionnées.

B.3. Commandes connexes

Pour lister les colonnes d'une table :
DESC[RIBE] [propriétaire.]nom_table ;

Pour lister les contraintes d'une table :
SELECT TABLE_NAME, CONSTRAINT_NAME, CONSTRAINT_TYPE, SEARCH_CONDITION
FROM USER_CONSTRAINTS
WHERE TABLE_NAME='nom_table';

Pour supprimer des tables :
DROP TABLE [propriétaire.]nom_table
[CASCADE CONSTRAINTS];

Ne pas oublier de valider : COMMIT ;

B.4. Créer / supprimer une Bases de données

Le préalable à toute création de tables est de créer une base de données. Cette commande est donnée un peu tardivement dans ce chapitre, puisque sans base de données aucune table ne peut être implémentée.

Pour créer une base de données :
CREATE DATABASE nom_bdd;

Pour supprimer une base de données (supprimera également le contenu) :
DROP DATABASE nom_bdd;

B.5. Les types

B.5.1. Type caractère fixe

CHAR(longueur) : permet de stocker une chaîne de caractères de longueur fixe maximum de 2000 caractères.

B.5.2. Type caractère variable

VARCHAR2(longueur) : permet de stocker une chaîne de caractères de longueur variable longueur maximale de 4000 caractères. Une chaîne plus courte que la longueur spécifiée n'occupera que l'emplacement correspondant à sa taille réelle.

Variante : VARCHAR(longueur).

B.5.3. Type numérique :

NUMBER[(précision [, échelle])] : utilisé pour les nombres entiers, nombres décimaux et nombres en virgule flottante.
Précision : nombre entier de chiffres significatifs de 1 à 38 (38 par défaut) ;
Echelle : nombre de chiffres à droite du séparateur décimale de -84 à +127.

Variantes :

- INTEGER : sur 4 octets ;
- SMALLINT : sur 2 octets ;
- DEC[IMAL](longueur,décimales) ;
- FLOAT.

B.5.4. Type date

DATE : permet de stocker une date et/ou une heure. Le format est DD-MON-YY.

B.5.5. Type chaîne longue

LONG : permet de stocker une chaîne de caractères d'une longueur maximale de 2Go. Une seule colonne de ce type est autorisée par table.

B.5.6. Type binaire

RAW(n) : permet de stocker des données de type binaire de longueur fixe (2000 octets maximum). On doit insérer les données hexadécimales sous forme de chaîne de caractères.

B.5.7. Autres types

- BLOB : Binary Large Object, jusqu'à 4Go ;
- BFILE: type fichier jusqu'à 4Go;
- CLOB ;
- NCLOB ;
- LONG RAW ;
- ROWID ;
- NCHAR(n);
- NVARCHAR2(n).

C. Les transactions

Une transaction est une séquence de changements intervenants sur les tables d'une base de données. Elle est définie comme une unité logique de travail (LUW).

L'intérêt principal de gérer les transactions est la possibilité de l'annuler en cas d'erreur ou de problèmes techniques, pour retrouver un état antérieur de cohérence de la base de données :

- VALIDATION des modifications : COMMIT [WORK];
- ANNULATION des modifications : ROLLBACK [WORK] [TO <nom_point];
- POINT D'ENREGISTREMENT : SAVEPOINT nom_point;

Pour valider automatiquement : SET AUTOCOMMIT ON [OFF]

D. La commande ALTER TABLE

Il est possible de modifier la structure d'une table :
- ajout de colonnes (dénormalisation) ou/et de contraintes ;
- modification de certaines contraintes ;
- changement du caractère obligatoire/facultatif (NULL) ;
- rectification de la largeur d'une colonne (VARCHAR2).

```
ALTER TABLE [propriétaire.]nom_table
ADD [(col1 TYPE(taille) [DEFAULT expression] [NOT NULL])]
      [contrainte…];
```

Exemple :
```
ALTER TABLE ELEVE
ADD (NAISS date);
```

```
ALTER TABLE [propriétaire.]nom_table
[DROP PRIMARY KEY / UNIQUE (colonne1 [, colonne2]…
/ CONSTRAINT nom_contrainte
[CASCADE] ]
[ENABLE nom_contrainte / DISABLE nom_contrainte] ;
```

Attention : il est impossible de supprimer une clé primaire ou unique qui est utilisée dans une contrainte d'intégrité référentielle. Il faut supprimer à la fois la clé référencée et la clé étrangère.

On peut éventuellement modifier des colonnes existantes (si elles ne sont pas valuées, ou bien pour augmenter la taille d'une chaîne de caractères).

```
ALTER TABLE [propriétaire.]nom_table
MODIFY (colonne nouveau_type (nouvelle_taille) , ...);
```

```
ALTER TABLE [propriétaire.]nom_table
MODIFY (colonne NULL / NOT NULL , ...);
```

Exemple :
```
ALTER TABLE ELEVE
MODIFY (NOM char(35));
```

E. Autres notions et commandes usuelles

E.1. Les Synonymes (alias)

Il peut être pratique de donner un synonyme à un nom de table :
```
CREATE [PUBLIC] SYNONYM nom_synonyme
FOR [propriétaire.]nom_table_ou_vue;
```

```
DROP [PUBLIC] SYNONYM nom_synonyme ;
```

Exemple :
```
CREATE SYNONYM ELEVE
FOR THIERRY.ELEVE;
```

L'administrateur de la base peut également créer des synonymes reconnus par tous les utilisateurs en utilisant le mot clé **public** :
```
CREATE PUBLIC SYNONYM ELEVE
```

FOR THIERRY.ELEVE;

E.2. Les index

Afin d'améliorer les temps de réponses ou tout simplement traduire la notion de clef primaire d'une table, SQL propose un mécanisme d'index :
Un index est associé à une table, mais il est stocké à part.
Un index peut ne porter que sur une colonne (**index simple**) ou sur plusieurs (**index multiple**).
Chaque valeur d'index peut ne désigner qu'une et une seule ligne, on parlera alors d'**index unique** donc de **clef primaire**, dans le cas contraire on parlera d'**index dupliqué**.
L'intérêt d'indexer une table sur sa clé (avec un index unique) est la seule garantie de maintenir l'unicité de cette clé lors d'ajout de ligne, car cette opération générerait un message d'erreur.
Lors des ajouts et des suppressions de lignes, les index sont mis à jour.
On peut créer plusieurs index sur une même table, mais généralement on se contentera d'indexer les tables sur leur clé primaire (simple ou multiple).

Remarque : Un index est automatiquement créé lorsqu'une table est créée avec l'option PRIMARY KEY.

CREATE [UNIQUE] INDEX nom_index
ON nom_table (colonne1 [ASC/DESC] [, colonne2 [ASC/DESC],...);

DROP INDEX nom_index ;

Exemple :
CREATE UNIQUE INDEX I-ELEVE
ON ELEVE (NOM,PNOM);

E.3. Les vues

Une vue est une table virtuelle qui n'a aucune existence physique, seule sa description est stockée sous forme de requête faisant intervenir des tables de la base ou bien d'autres vues.
Les vues permettent de :

- Cacher certaines informations présentes dans les tables ;
- Eviter de redéfinir à chaque fois des requêtes complexes pour accéder à l'information ;
- Présenter de façon simplifiée une structure plus complexe.

CREATE [OR REPLACE] [FORCE / NOFORCE]
VIEW nom_vue [(colonne_vue1 [, colonne_vue2]…)]
AS SELECT …
[WITH CHECK OPTION [CONSTRAINT nom_contrainte)] ;

- OR REPLACE : permet de modifier la vue sans la supprimer;
- NOFORCE : la vue n'est créée que si la table existe ;
- FORCE : la vue est créée, même si la table n'existe pas.

Attention : la clause ORDER BY est interdite, on ne peut pas mettre à jour les tables au travers des vues (sauf si la vue ne comporte aucune jointure, aucun GROUP BY, aucune colonne calculée).

DROP VIEW nom_vue;

RENAME VIEW nom_vue TO nouveau_nom;

Exemple :
CREATE VIEW IG1
AS SELECT nom, prenom, (SYSDATE-datenaiss)/ 365 AGE
FROM eleve
WHERE classe = 'BTSIG1';

Remarque : AGE est un alias donné à la colonne issue du calcul.

Interrogation "à travers une vue"
SELECT *
FROM IG1 ;

Remarques : Une Vue s'utilise exactement comme une table.

E.4. Les séquences

La création de séquences permet de générer par exemple des valeurs séquentielles de clé primaire.

CREATE SEQUENCE [propriétaire.]nom_séquence
[INCREMENT BY 1_ou_autre_valeur]
[START WITH valeur]
[MAX VALUE valeur / NOMAXVALUE]
[MIN VALUE valeur / NOMINVALUE]
[CYCLE / NOCYCLE]
[CACHE valeur / NOCACHE];

CYCLE / NOCYCLE permet de recommencer à la valeur de départ ou de s'arrêter au maximum (ou au minimum en cas de décrémentation).

CACHE permet de générer à l'avance des valeur (par défaut 20 valeurs sont stockées en mémoire).

Exemple :
CREATE SEQUENCE compteurauto
START WITH 1000
INCREMENT BY 10
NOMAXVALUE
NOCYCLE;

On utilisera nom_séquence.NEXTVAL pour incrémenter la séquence et nom_séquence.CURRVAL pour connaître la valeur courante de la séquence.

Exemple :
INSERT INTO MACHINE (numeleve, nomeleve, prenomeleve)
VALUES (compteurauto.NEXTVAL, 'Pamalin', 'Antony');

Supprimer une séquence :
DROP SEQUENCE nom_séquence;

Modifier une séquence (mêmes paramètres que lors de la création) :
ALTER SEQUENCE …

Affichage des séquences :
SELECT *
FROM user_sequence
WHERE sequence_name = 'nom_séquence';