Langage de Définition des Données (LDD) – Oracle et pgsql

1. Les types de données

CHAR (*lg*)	Type « caractère » de longueur fixe lg. Où lg appartient à l'interval [1 ; 255]
VARCHAR (*lg*)	Type « caractère » de longueur variable. Où lg appartient à l'interval [1 ; 2000], et lg est la taille maximale de la donnée considérée.
INTEGER	Type entier sur (au moins) 4 octets
FLOAT	Type réel
NUMBER (*p, s*)	Type numérique de précision p (nombre de chiffres significatifs) et d'échelle s (nombre de décimales)
DECIMAL (*p, s*)	Idem NUMBER
DATE	Type date/heure
BOOLEAN	Type booléen

2. Les séquences : un numéro auto … pas si automatique

2.1. Principe

Une séquence est un objet du SGBD qui permet de générer des numéros séquentiels (autrement appelés numéros indexés).

CREATE SEQUENCE *nomSequence*
INCREMENT BY *entier* *// valeur de l'incrément*
START WITH *entier* *// valeur du premier numéro*
MAXVALUE *entier* *// valeur du + grand numéro*
MINVALUE *entier* *// valeur du + petit numéro*
[NO]CYCLE *// retour à la valeur initiale lorsque valeur max atteinte*

Note : Une séquence peut être modifiée par **ALTER SEQUENCE** *nomSequence* ou supprimée par **DROP SEQUENCE** *nomSequence*

2.2. Utilisation sous Oracle

La valeur courante d'une séquence est accessible par : **SELECT** *nomSequence***.CURRVAL FROM DUAL**
La prochaine valeur dans la séquence est obtenue par : **SELECT** *nomSequence***.NEXTVAL FROM DUAL**

*nomSequence***.CURRVAL** et *nomSequence***.NEXTVAL** sont, par ailleurs, utilisables tels que dans un **INSERT** ou un **UPDATE**

2.3. Utilisation sous pgsql

La valeur courante d'une séquence est accessible par : **SELECT** last_value **FROM** *nomSequence*
La prochaine valeur dans la séquence est obtenue par : **SELECT nextval(**'*nomSequence*'***)***

nextval('*nomSequence*'***)*** est, par ailleurs, utilisable tel que dans un **INSERT** ou un **UPDATE**

3. Les contraintes d'intégrité

Une contrainte est un automatisme du SGBD qui garantit que les valeurs d'une colonne ou d'un ensemble de colonnes satisfont à une condition donnée. Les contraintes assurent la cohérence des données (concept d'intégrité). Elles sont décrites lors de la définition des tables de la base.

Il existe plusieurs types de contraintes. Elles portent sur :

- La liste des valeurs autorisées (**contrainte de domaine**)
- L'unicité de valeur (**contraintes d'unicité**)
- L'obligation de valoriser (**contrainte de non-nullité**)
- Le besoin de cohérence référentielle (**contrainte de clé étrangère**)
- Le besoin d'une donnée discriminante (**contrainte de clé primaire** qui réunit les contraintes d'unicité et de non-nullité)

Selon le besoin, les contraintes peuvent s'exprimer, soit au niveau colonne (valables pour une seule colonne), soit au niveau table (valables pour plusieurs colonnes d'une même table).

Résumé des clauses associées :

	Contrainte colonne	Contrainte table
Domaine	**CHECK (***condition***)**	**CHECK (***condition***)**
Unicité	**UNIQUE**	**UNIQUE (***liste-colonnes***)**
Non-nullité	**NOT NULL**	
Clé primaire	**PRIMARY KEY**	**PRIMARY KEY (***liste-colonnes***)**
Clé étrangère	**REFERENCES** *col*	**FOREIGN KEY (***liste-colonnes***) REFERENCES (***liste-col***)**

La contrainte **PRIMARY KEY** ne peut être utilisée qu'une fois par table. En conséquence, une clé primaire composée est nécessairement définie en une seule fois par une contrainte de table.

La contrainte de clé étrangère peut être complétée par la clause **ON DELETE CASCADE**, qui maintient l'intégrité référentielle en supprimant les clés référençantes lors de la suppression des clés référencées. Exemple : la suppression d'un client entraînera la suppression de ses commandes. Sans cette option, la suppression du client aurait été refusée. C'est une option dangereuse car sans contrôle.

La contrainte de clé étrangère ne peut référencer que des colonnes déjà existantes. L'ordonnancement des créations de tables est donc à envisager après mûre réflexion.

Chaque contrainte définie est affectée d'un nom (nom de contrainte) stocké dans le dictionnaire des données de la base. Ce nom est construit par le SGBD lui-même, sauf à être forcé par la clause **CONSTRAINT** qui permet de choisir un nom personnalisé.

4. Tables

4.1. Création

Création d'une table vide

```
CREATE TABLE nomTable (
    nomColonne1 type [contrainteColonne1]
    [, nomColonne2 type [contrainteColonne2] ]
    [, …]
    [, contrainteTable1]
    [, …]
) ;
```

Création d'une table remplie par une requête

```
CREATE TABLE nomTable ( nomColonnes ) AS
    SELECT … FROM …
;
```

4.2. Modification

La modification d'une table existante est obtenue par **ALTER TABLE** *nomTable* complété comme suit par :

Clause	Objet
ADD …	Pour l'ajout, la modification, la suppression ou le renommage d'une colonne ou d'une contrainte de table
MODIFY …	
DROP …	
RENAME …	

4.3. Suppression

La suppression d'une table est obtenue sans condition par **DROP TABLE** *nomTable* . La suppression d'une table peut provoquer la violation de contraintes d'intégrité. C'est pourquoi la clause **CASCADE CONSTRAINTS** peut être optionnellement ajoutée à cette suppression. Elle provoquera la suppression des contraintes d'intégrité liées à la table supprimée.

5. Vues

Une vue est une représentation logique issue de la combinaison des contenus d'une ou plusieurs tables. Elle obtient ses données à partir des tables sur lesquelles elle est basée. Elle hérite des caractéristiques des objets auxquels elle fait référence. Les vues sont utilisées pour :

- Assurer la sécurité des données (empêcher la visibilité de certaines données sans changer la structure)
- Masquer la complexité des données (masquer les jointures)
- Simplifier la formulation des requêtes
- Offrir des perspectives multiples sur les mêmes données

Une fois crée, une vue est perçue comme une table. Ce qui a pour conséquence directe d'autoriser l'emploi d'un nom de vue partout où un nom de table est accepté. En particulier :

- Une vue peut entrer dans la composition d'une autre vue
- Une vue peut être l'objet d'une mise à jour (**UPDATE, DELETE** ou **INSERT**). Mais cela ne fonctionne que dans un nombre de cas assez restreint.

Toutefois, il ne faut pas oublier qu'une vue n'est constituée que de sa définition (la requête sur laquelle elle est basée). Elle ne stocke aucune donnée. Les données qu'elle ramène ne sont obtenues qu'à l'exécution d'une requête.

5.1. Création

CREATE [**OR REPLACE**] **VIEW** *nomVue* [(*listeNomsColonnes*)] **AS** *requête*

5.2. Suppression

DROP VIEW *nomVue*

6. Index

Dans un SGBD, un index joue le même rôle que celui présent dans une encyclopédie : accéder le plus rapidement possible à un ensemble d'informations par le biais d'un mot-clé. Un index peut être créé pour une colonne ou un ensemble de colonnes. Pour chaque valeur des colonnes indexées, l'index contient le numéro de ligne de l'enregistrement correspondant. En règle générale, des index sont créés automatiquement sur les clés primaires et étrangères d'une table. Mais, pour des questions de performances, il peut être intéressant d'en créer de supplémentaires sur d'autres colonnes. Particulièrement, lorsque l'on sait qu'il y aura beaucoup de restrictions SQL réalisées sur une colonne donnée, celle-ci aura tout lieu d'être indexée, même si elle n'est pas clé.

6.1. Création

CREATE INDEX *nomIndex* **ON** *nomTable* (*listeNomsColonnes)*

6.2. Suppression

DROP INDEX *nomIndex*