DDL

Data Definition Language

Les séquences

Une séquence est un objet de base de données qui génère une séquence de valeurs numériques.

Les séquences sont parfaitement adaptées à la tâche de génération de valeurs de clé uniques. Elles sont très utiles lors de la déclaration de clé primaires.

Pour créer une séquence, vous devez utiliser la commande `CREATE SEQUENCE`

```
CREATE SEQUENCE ma_table_id_seq;
CREATE TABLE ma_table (
  id INT DEFAULT nextval('ma_table_id_seq'),
  CONSTRAINT pk_ma_table_id PRIMARY KEY (id));
ALTER SEQUENCE ma_table_id_seq OWNED BY ma_table.id;
```

Introduction aux requêtes DDL

Les requêtes DDL, ou langage de définition de données, sont un ensemble de commandes SQL utilisées pour définir, modifier et gérer la structure des bases de données.

Les requêtes DDL permettent

- de créer des tables
- de modifier leur structure (ajouter, supprimer ou modifier des colonnes)
- de définir des contraintes d'intégrité (comme les clés primaires et les clés étrangères)
- de créer des indexes

Ainsi que d'autres opérations liées à la gestion de la structure de la base de données.

Création et suppression d'une base de données

Syntaxe pour la création d'un base de données

```
CREATE DATABASE ma_bdd;
```

Syntaxe pour la suppression d'une base de données

```
DROP DATABASE IF EXISTS ma_bdd;
```

Le mot clé "IF EXISTS" est facultatif. Il permet de na pas générer d'erreurs si la base de données n'existe pas.

La visualisation des base de données d'un serveur PostgreSQL

```
SELECT datname FROM pg_database;
```

A l'aide de l'outil `PSQL`, la ligne de commande à exécuter est
`\l`

Création et suppression d'un schéma

Une fois connecté à une base de données, vous pouvez créer des Schémas.

Syntaxe pour la création d'un schéma

```
CREATE SCHEMA mon_schema;
```

Syntaxe pour supprimer tous les schémas

```
DROP SCHEMA myschema;
```

Ou si vous voulez également supprimer les objets contenu dans votre schéma

```
DROP SCHEMA myschema mon_schema;
```

Visualisation des schémas

A l'aide de l'outil PSQL

```
\dn
```

Ou dans la base de donnés concernée, tapez une des commandes suivantes

```
select * from information_schema.schemata;
```

Ou

```
select * from pg_catalog.pg_namespace;
```

Les Schémas `information_schema` ainsi que les schémas `pg_*` sont des schéma utilisés par PostgreSQL.

Visualisation des tables

Voici la requête SQL à utiliser pour visualiser toutes les tables d'un schéma donné

```
SELECT * FROM information_schema.tables
WHERE table_schema = 'public';
```

A partir de l'outil PSQL

Affichage de toutes les tables de tous les schémas

```
\dt *.*
```

Affichage de toutes les tables contenues dans un schéma donné

```
\dt public.*
```

Création des tables

Voici la syntaxe de création d'une table

```
CREATE TABLE sample_table (
    id INT,
    amount DOUBLE PRECISION,
    value DECIMAL(10, 2),
    is_active BOOLEAN,
    birthdate DATE,
    initial CHAR(1),
    name VARCHAR(20),
    description TEXT,
    binary_data BYTEA
);
```

Modification de tables

La commande à utiliser pour modifier une table est `ALTER TABLE`.

Modification d'un type de colonne

```
ALTER TABLE test.premiere ALTER COLUMN text TYPE varchar(10);
```

Si la modification d'un type de colonne nécessite une conversion (par exemple lors du passage d'un type varchar au type int), vous devez spécifier le type de conversion que vous devez effectuer à l'aide du mot clé `USING`

```
ALTER TABLE test.premiere ALTER COLUMN text TYPE INT USING text::integer;
```

Pour ajouter une contrainte non nullable (NOT NULL)

```
ALTER TABLE test.Client ALTER COLUMN id SET NOT NULL;
```

Pour ajouter une valeur par défaut à une colonne

```
ALTER TABLE test.Client ALTER COLUMN id SET DEFAULT 0;
```

Modification de tables

Pour ajouter une clé primaire à une table existante

```
ALTER TABLE test.Client ADD PRIMARY KEY (id);
```

Pour ajouter une clé étrangère

```
ALTER TABLE orders
ADD CONSTRAINT fk_orders_customers
FOREIGN KEY (customer_id) REFERENCES customers (id);
```

Ajout d'une colonne

```
ALTER TABLE sample_table ADD COLUMN nouv_col REAL;
```

Suppression d'une colonne

```
ALTER TABLE sample_table DROP COLUMN nouv_col;
```

Renommer une colonne

```
ALTER TABLE clients RENAME COLUMN email TO contact_email;
```

Suppression de tables

Voici la syntaxe pour la suppression d'une table

```
DROP TABLE sample_table;
```

Nous pouvons spécifier une option `CASCADE`

```
DROP TABLE sample_table CASCADE;
```

L'option `CASCADE` permet de supprimer la table ainsi que tous les objets qui dépendent de cette table.

Vider le contenu d’une table

Pour vider le contenu de toutes une table sans supprimer la structure de la table, vous pouvez utiliser l’instruction TRUNCATE.

La commande TRUNCATE supprime rapidement toutes les lignes d'un ensemble de tables. Car contrairement à la commande DELETE qui supprime les données ligne par ligne, la commande TRUNCATE supprime toutes les pages associées à une table donnée.

Exemple

```
TRUNCATE employee;
```

Si dans votre table, vous avez déclarer une colonne comme IDENTITY, vous pouvez redémarrer la séquence associée à votre champ d’identité en utilisant la syntaxe suivante

```
TRUNCATE employee RESTART IDENTITY;
```

Les principaux types de données

Voici les principaux types de données de PostgreSQL

INT: un entier (encodé sur 4 octets). 2 variantes possibles: INT2 et INT8 (sur 2 et 8 octets respectivement).

DOUBLE PRECISION: un nombre réel (encodé sur 8 octets). On peut également utiliser REAL (encodé sur 4 octets).

DECIMAL(p, s): un nombre décimal ayant p chiffres significatifs (precision) et s chiffres après la virgule (scale). Essentiellement utilisé pour la finance pour avoir un calcul exact sans approximations.

BOOLEAN: un boléen. La valeur contenue est true ou false.

Les principaux types de données

DATE: Contient une date (année, mois, jour).

TIMESTAMPTZ: Contient la date et l'heure avec les informations du fuseau horaire.

TIMETZ: l'heure de la journée avec les informations du fuseau horaire.

CHAR(n): Chaîne de caractères contenant un nombre fixe de caractères (n caractères).

VARCHAR(n): Chaîne de caractères contenant un nombre variable de caractères (max n caractères).

TEXT: Chaîne de caractères de n'importe quelle taille.

BYTEA: Donné binaire.

Les séquences

Le mot clé `PRIMARY KEY` signale l'existence d'une clé primaire. Une clé primaire identifie une ligne de la table de façon unique.

La commande `ALTER SEQUENCE` permet de modifier une séquence existante ici nous déclarons qu'elle est liées à la colonne `id` de la table `ma_table`.

Pour supprimer une séquence, vous devez utiliser la syntaxe suivante

```
DROP SEQUENCE ma_table_id_seq;
```

Les séquences

En PostgreSQL, il existe un moyen pour auto incrémenter le type INT automatiquement.

La syntaxe à ajouter est: GENERATED ALWAYS AS IDENTITY
La syntaxe précédente, indique que la colonne est auto-générée. Cette colonne également appelée séquence ou identité n'a pas besoin d'être spécifiée lors de l'insertion de nouvelles données. La colonne d'identité est une colonne NOT NULL à laquelle est attachée une séquence implicite.

L'exemple précédent peut ainsi s'écrire plus simplement comme suit

```
CREATE TABLE ma_table (
  id INT GENERATED ALWAYS AS IDENTITY,
  CONSTRAINT pk_ma_table_id PRIMARY KEY (id));
```

Notez que par défaut les tables sont créées dans le schéma `public`, mais vous pouvez spécifier votre schéma à l'aide de la syntaxe suivante

```
CREATE TABLE mon_schema.ma_table ( … );
```

Gestion des contraintes

Lors de la création des tables en base de données, vous pouvez spécifier les contraintes suivantes

1. Contrainte **PRIMARY KEY**. Spécifie les champs qui identifient la ligne.
2. Contrainte **FOREIGN KEY**. Spécifie les champs qui référencent une ligne d'une autre table.
3. Contrainte **NOT NULL**. La valeur ne peut pas être à NULL.
4. Contrainte **UNIQUE**. La valeur doit être unique dans la table.
5. Contrainte **CHECK**. La valeur doit respecter certaines conditions.

Clés primaires et clés étrangères

Une clé primaire (primary key), permet d'identifier une ligne de la table de façon unique.
Elle est introduite par le mot clé `PRIMARY KEY`.
Vous ne pouvez avoir qu'une seule clé primaire par table.
Les clés primaires assurent que la ou les colonnes sont uniques et non nullables.

La clé primaire peut-être déclarée lors de la création de la table

```
CREATE TABLE classe
  (classe_id INT GENERATED ALWAYS AS IDENTITY,
  niveau VARCHAR(12) NOT NULL,
  libelle VARCHAR(20) NOT NULL,
  CONSTRAINT pk_classe_id PRIMARY KEY (classe_id));
```

Ou elle peut être déclarée après la création de la table

```
ALTER TABLE classe ADD CONSTRAINT pk_classe_id PRIMARY KEY (classe_id);
```

Clés primaires et clés étrangères

Une clé étrangère (foreign key) est une contrainte qui garantit l'intégrité référentielle entre 2 tables.

Une clé étrangère identifie une ou plusieurs colonnes d'une table comme référençant une clé primaire d'une autre table.

```
CREATE TABLE eleve (
  eleve_id INT GENERATED ALWAYS AS IDENTITY,
  nom varchar(20) NOT NULL,
  prenom varchar(20) NOT NULL,
  classe_id int NOT NULL,
  CONSTRAINT pk_eleve_id PRIMARY KEY (eleve_id),
  CONSTRAINT fk_eleve_classe_id FOREIGN KEY (classe_id) REFERENCES classe(classe_id));
```

La clé étrangère peut être également déclarée après la création de la table

```
ALTER TABLE classe ADD CONSTRAINT fk_eleve_classe_id FOREIGN KEY (classe_id) REFERENCES classe(classe_id);
```

Contrainte NOT NULL

Par défaut les champs créés dans une table peuvent valoir NULL. Pour spécifier qu'un champ n'est pas nullable, vous devez utiliser le mot clé `NOT NULL`.

Vous pouvez spécifier qu'un champ est nullable à l'aide du mot clé `NULL`.

Exemple

```
CREATE TABLE Employees (
  employee_id INT GENERATED ALWAYS AS IDENTITY,
  first_name VARCHAR(50) NOT NULL,
  last_name VARCHAR(50) NOT NULL,
  hire_date DATE NULL,
  CONSTRAINT pk_employee_id PRIMARY KEY (employee_id));
```

Contrainte d'unicité

Vous pouvez spécifier qu'un champ ou plusieurs champs sont uniques pour une table avec le mot clé `UNIQUE`.

Exemple

```
CREATE TABLE students (
  student_id INT GENERATED ALWAYS AS IDENTITY,
  student_name VARCHAR(100) NOT NULL
  email VARCHAR(100),
  CONSTRAINT uq_students_email UNIQUE(email));
```

Vous pouvez également spécifier qu'un ensemble de plusieurs champs soient uniques pour une table donnée.

```
ALTER TABLE students
ADD CONSTRAINT uq_students_student_name_email UNIQUE(student_name, email);
```

Contraintes sur les valeurs

Vous pouvez pour chaque champ d'une table, spécifier une contrainte sur les valeurs possibles

```
CREATE TABLE products (
    product_id INT GENERATED ALWAYS AS IDENTITY,
    product_name VARCHAR(100) NOT NULL,
    price DECIMAL(10, 2) CHECK (price >= 0),
    stock_quantity INT CHECK (stock_quantity >= 0));
```

Vous pouvez spécifier des contraintes impliquant plusieurs champs

```
ALTER TABLE eleves ADD CONSTRAINT ck_eleves_note CHECK ((note >= 0) and (note <= 20));
```