

SQL

CM2 : SQL, intégrité et LDD

Mickaël Martin Nevot

V1.1.1



Cette œuvre de Mickaël Martin Nevot est mise à disposition sous licence Creative Commons Attribution - Utilisation non commerciale - Partage dans les mêmes conditions.

LDD

- I. Prés.
- II. BD et SGBD
- III. LDD
- IV. LMD
- V. LCT
- VI. Droits
- VII. LDSP
- VIII. SQL avancé

SQL

- *Structured query language* : langage de requête structurée
- **Un seul langage général** :
 - Langage de description de données (LDD)
 - Langage de manipulation de données (LMD)
 - Langage de description des schémas physiques (LDSP)
 - Contrôle et administration



SQL

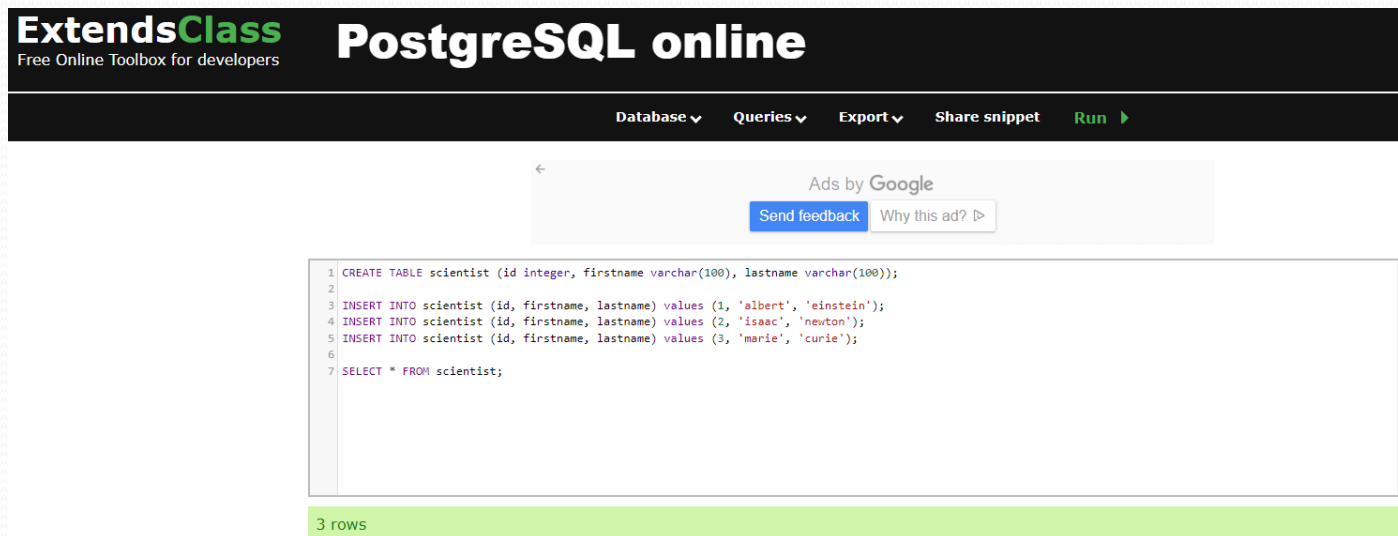
Pour tester

- Interpréteur en ligne :

<https://extendsclass.com/postgresql-online.html>

- Documentation :

<https://www.postgresql.org/docs/current>



ExtendsClass
Free Online Toolbox for developers

PostgreSQL online

Database ▾ Queries ▾ Export ▾ Share snippet Run ▶

Ads by Google
Send feedback Why this ad? ▶

```
1 CREATE TABLE scientist (id integer, firstname varchar(100), lastname varchar(100));
2
3 INSERT INTO scientist (id, firstname, lastname) values (1, 'albert', 'einstein');
4 INSERT INTO scientist (id, firstname, lastname) values (2, 'isaac', 'newton');
5 INSERT INTO scientist (id, firstname, lastname) values (3, 'marie', 'curie');
6
7 SELECT * FROM scientist;
```

3 rows

id	firstname	lastname
1	albert	einstein
2	isaac	newton
3	marie	curie

En SQL, les retours à la ligne sont non déterministes

LDD

- Langage de description de données (LDD)
- Spécification du **schéma conceptuel** d'une BD :
 - Base de données
 - Relations
 - Contraintes
- Spécification des **vues** d'une BD











Permet de créer des « objets » SQL



Syntaxe

- **Dénomination** (identifieurs et mots clef) :
 - Insensibles à la casse (majuscules conseillées pour mots clefs)
 - Caractères spéciaux interdits ou à éviter
 - Ne pas utiliser les mots clefs comme identifieurs
- **Commentaires** :
 - Ligne : `--`
 - Bloc : `/* ... */`
- **Chaînes de caractères** :
 - Guillemets simples : `'Ceci est une chaîne'`
- **NULL** :
 - Valeur nulle, information inconnue ou incomplète ← Pas 0 ou ""



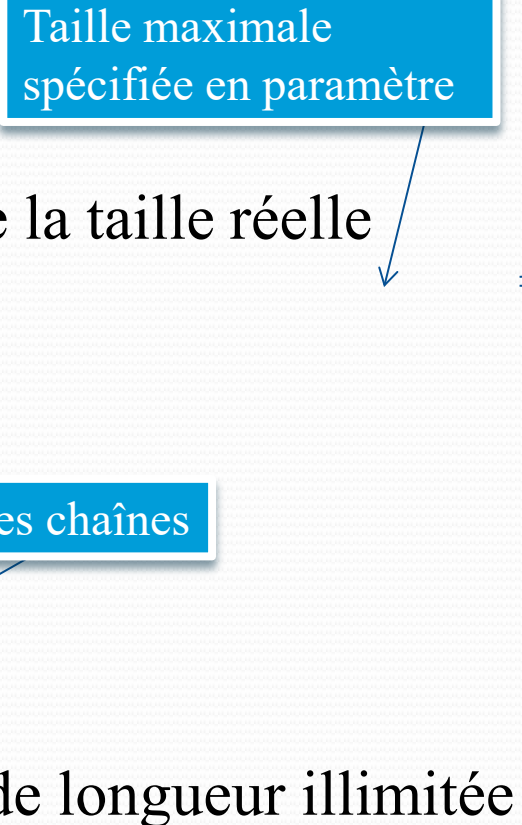
Les types numériques

Type	Taille mémoire	À la norme
SMALLINT	2 octets	
INTEGER	4 octets	
BIGINT	8 octets	
DECIMAL(M, D)	M, (D + 2 si M < D)	
NUMERIC(M, D)	M, (D + 2 si M < D)	
REAL	4 octets	
DOUBLE PRECISION	8 octets	
SMALLSERIAL	2 octets	
SERIAL	4 octets	
BIGSERIAL	8 octets	

M,D signifie : numérique de taille maximal M avec D décimales

Pas de UNSIGNED et de ZEROFILL en PostgreSQL

Les types caractères

- CHAR(...), CHARACTER(...) :  Taille maximale spécifiée en paramètre
 - Chaîne de **taille fixe**
 - Complétée/tronquée en fonction de la taille réelle
 - VARCHAR(...), VARYING(...) :
 - Chaîne de **longueur variable**
 - TEXT :  Longues chaînes
 - Chaîne de longueur illimitée
 - BYTEA :
 - Chaînes **binaires** (non textuelles) de longueur illimitée
- 

Taille maximale : en nombre de caractères

Quel type de caractères ?

Taille fixe

- Avantages :
 - Accès direct facile (rapide)
 - Pas de fragmentation
 - Réparation facile de table
- Inconvénient :
 - Pas d'économie de place

CHAR (...), CHARACTER (...)

Taille variable

- Avantage :
 - Gain de place
- Inconvénients :
 - Insertion plus longue (car calcul de la taille exacte)
 - Fragmentation inévitable
 - Temps d'accès :
moindre performance

VARCHAR (...), VARYING (...), TEXT, BYTEA

Les types de date

- DATE :
 - Entre 4713 avant J.-C. et le 5874897 après J.-C.
- TIME (...):
 - Entre 00:00:00:00 et 24:00:00 ← Fuseaux horaires possibles
- TIMESTAMP (...):
 - Entre 4713 avant J.-C. et le 294276 après J.-C.
- INTERVAL (...): ← Longueur de l'affichage, 14 par défaut
 - Nombre de secondes écoulées depuis le 01/01/1970



TIME, TIMESTAMP, et INTERVAL acceptent une précision optionnelle du nombre de décimales pour les secondes

Le type booléen

Considéré comme TRUE

- TRUE
- 't'
- 'true'
- 'y'
- 'yes'
- '1'

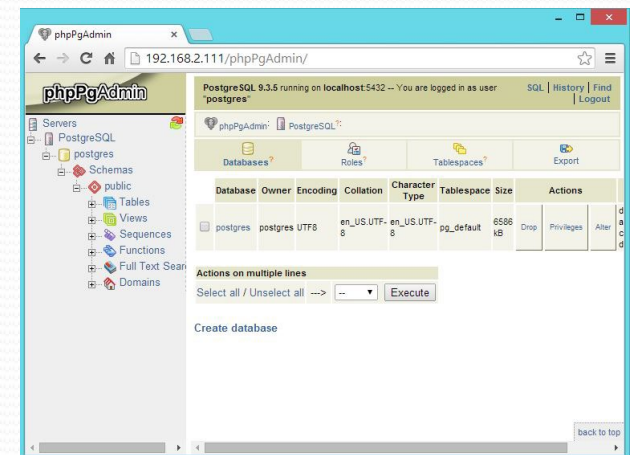
Considéré comme FALSE

- FALSE
- 'f'
- 'false'
- 'n'
- 'no'
- '0'

Il est recommandé d'utiliser TRUE et FALSE (à la norme)

Les commandes PostgreSQL

- Création de BD
- Destruction de BD
- Création des tables
- Destruction des tables
- Modifications de la structure des tables
- Création de clefs étrangères



Création/destruction de BD

- CREATE DATABASE :

Syntaxe :

```
CREATE DATABASE Db_name [...]
```

En SQL, les retours à la ligne sont non déterministes

- DROP DATABASE :

- Efface tous les fichiers des tables
- S'interrompt si la base n'existe pas :
 - Sauf si l'option IF EXISTS est spécifiée

Syntaxe :

```
DROP DATABASE [IF EXISTS] Db_name
```



Création/destruction de tables

- **CREATE** : création de table

Syntaxe :

```
CREATE [...] TABLE [IF NOT EXISTS] Tbl_name  
([<CREATION_CLAUSE> [, ...]]) [...]
```

<CREATION_CLAUSE> :

```
column_name type ... [column_constraint [...]]
```

```
CREATE TABLE Etudiant (  
    nom CHAR(32),  
    prenom VARCHAR(32)  
);
```

Une table est créée par défaut dans le schéma public (hors cadre du cours)

- **DROP** : destruction définitive d'une table

Syntaxe :

```
DROP TABLE [IF EXISTS] Tbl_name [, ...] [...]
```

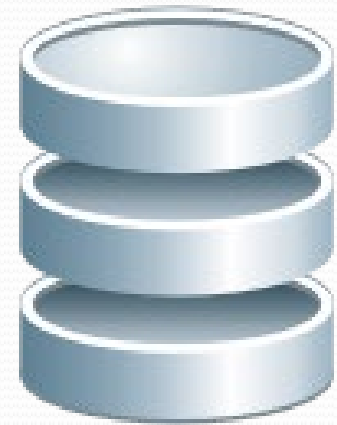
- **TRUNCATE** : effacement de toutes les données d'une table

Syntaxe :

```
TRUNCATE [TABLE] [...] Tbl_name [*] [, ...] [...]
```

Modification de tables

- ALTER TABLE : ALTER [ONLY] TABLE `Tbl` ACTION;
- ACTION :
 - ADD COLUMN `col` CREATION_CLAUSE
 - ALTER `col` SET DEFAULT value
 - DROP COLUMN `col`
 - RENAME TO `Tbl_name`
 - Etc.



Clefs (rappel)

- **Clef candidate, potentielle** (rappel) : ensemble des données permettant d'indexer chaque ligne de manière différenciée
- **Clef primaire** : **Obligatoire**
 - **Une seule par relation** (clef candidate retenue comme primaire)
 - **Simple** (un seul attribut) ou **composée** (plusieurs attributs)
 - **Unique et non nulle**
- **Clef étrangère** :
 - Clef primaire d'une autre relation de la BD

Employe			
nume	nome	daten	numd
1	Dupond	1/12/80	1
2	Jacques	21/04/68	1
3	Martin	03/25/52	2

Departement	
numd	nomd
1	ISMIN
2	ICM



Contraintes

- Types de contraintes :

- Contraintes d'intégrité :

- **Clef primaire**

- **Clef étrangère**

- Contraintes de valeurs :

- **Non nullité**

- **Unicité** (une valeur donnée n'apparaît qu'une fois)

- Définitions de contraintes :

- **Contraintes d'attributs** (spécifiques à un attribut donné)

- **Contraintes de tables** (portent sur plusieurs attributs)



Les contraintes apportent de la cohérence

Il est possible de nommer une contrainte

Contraintes d'intégrité

- Intégrité **d'entité** (ou de relation) :
 - Garanti un attribut (donc l'extension) sans doublon
- Intégrité **référentielle** :
 - Impose que toute valeur de la clef est une valeur de clef primaire de la relation associée
- Intégrité **sémantique** :
 - Pas toujours modélisable au niveau du schéma relationnel
 - *Triggers*
- Intégrité **applicative** :
 - Extérieure à la BD : liée à l'application

Des restrictions existent sur les mises à jour

Contraintes d'attributs

```

CREATE TABLE Etudiant (
  ide INTEGER PRIMARY KEY,
  nom CHAR(32) NOT NULL,
  prenom VARCHAR(20),
  email VARCHAR(25) UNIQUE,
  daten DATE,
  annee SMALLINT CHECK (annee < 4),
  tel CHAR(14)
  CONSTRAINT ck_tel
  CHECK (tel SIMILAR TO '[0-9][0-9]-[0-9][0-9]-[0-9][0-9]-[0-9][0-9]-[0-9][0-9]'),
  sexe CHAR(1) DEFAULT 'F'
);

```

Annotations de la syntaxe SQL :

- Clef primaire** : pointe vers `ide INTEGER PRIMARY KEY`
- Non nullité** : pointe vers `nom CHAR(32) NOT NULL`
- Unicité** : pointe vers `email VARCHAR(25) UNIQUE`
- Vérification, validation** : pointe vers `annee SMALLINT CHECK (annee < 4)`
- Une contrainte peut être nommée (pk, fk, uq, nn, ck)** : pointe vers `CONSTRAINT ck_tel`
- Valeur par défaut (si non renseignée)** : pointe vers `sexe CHAR(1) DEFAULT 'F'`

Ce n'est pas une contrainte, mais la syntaxe est la même : pointe vers `tel CHAR(14)`



Contraintes de tables

```
CREATE TABLE Convention (  
    ide INTEGER NOT NULL,  
    ids INTEGER NOT NULL,  
    datec DATE,  
    date_deb DATE,  
    duree INTEGER,  
    CONSTRAINT pk_con PRIMARY KEY (ide, ids),  
    CONSTRAINT fk_etu FOREIGN KEY (ide) REFERENCES Etudiant(ide)  
        -- Impactée (supprimée) comme la clef primaire.  
        ON DELETE CASCADE  
        -- Mise à nulle lorsque la clef primaire est impactée (mise à jour).  
        ON UPDATE SET NULL,  
    CONSTRAINT fk_soc FOREIGN KEY (ids) REFERENCES Societe(ids)  
        ON DELETE NO ACTION  
        -- Impact (suppression) interdit à la clef primaire.  
        ON UPDATE SET DEFAULT,  
        -- Prend la valeur par défaut lorsque la clef primaire est impactée (mise à jour).  
    CONSTRAINT uq_ide UNIQUE (ide, date_deb),  
    CONSTRAINT ch_dat CHECK (datec < date_deb)  
);
```

Clef primaire (composée)

Clef étrangère

Unicité

Vérification, validation

Exemple de création de table

```
CREATE TABLE Etudiant (  
  ide INTEGER NOT NULL PRIMARY KEY,  
  nom CHAR(25) NOT NULL,  
  prenom CHAR(20),  
  email VARCHAR(25) UNIQUE,  
  sexe CHAR(1)  
    CONSTRAINT ck_sex  
    CHECK (sexe IN ('M', 'F') OR sexe IS NULL),  
  daten DATE,  
  adresse CHAR(60),  
  annee SMALLINT DEFAULT 3 CHECK (annee < 4),  
  tel CHAR(14)  
    CONSTRAINT ck_tel  
    CHECK (tel SIMILAR TO '[0-9][0-9]-[0-9][0-9]-[0-9][0-9]-[0-9][0-9]-[0-9][0-9]'),  
  CONSTRAINT uq_nom UNIQUE (nom, prenom)  
);
```



Ecriture de données

LMD

- INSERT :

Syntaxe :

```
INSERT INTO Tbl_name [(column [, ...])] [...] VALUES (value [, ...])
```

```
INSERT INTO Etudiant (ide, nom, prenom) VALUES (1, 'Dupont', 'Leon');
```

```
INSERT INTO Etudiant VALUES (2, 'Martin', 'Michel');
```

```
INSERT INTO EtudiantClone SELECT * FROM Etudiant; -- La table doit être créée préalablement
```

- UPDATE :

Syntaxe :

```
UPDATE [...] Tbl_name [...] SET column = expr [, ...] [...] [WHERE condition ...] [...]
```

```
UPDATE Societe SET raisons = 'SA' WHERE raisons = 'SARL';
```

- DELETE :

Syntaxe :

```
DELETE FROM Tbl_name [*] [...] [WHERE condition ...] [...]
```

```
DELETE FROM Convention WHERE datec < '2020-01-01';
```

Attention : sans critère de filtrage, tous les enregistrements sont concernés !

Auto incrémentation

- Type : SMALLSERIAL, SERIAL ou BIGSERIAL
- Exemple :

```
CREATE TABLE Etudiant (  
    ide SERIAL PRIMARY KEY,  
    nom VARCHAR(64) NOT NULL  
);
```

Crée une séquence Etudiant_ide_seq
(hors cadre du cours)

```
INSERT INTO Etudiant VALUES (DEFAULT, 'Antonio Paz');  
INSERT INTO Etudiant VALUES (DEFAULT, 'Lilliana Angelovska');  
INSERT INTO Etudiant(nom) VALUES ('André Dopund');  
INSERT INTO Etudiant VALUES (NEXTVAL('Etudiant_ide_seq'), 'René Dupn');  
SELECT CURRVAL('Etudiant_ide_seq'); -- 4.  
SELECT SETVAL('Etudiant_ide_seq', 1); -- Change la valeur courante de la séquence.
```

ide	nom
1	Antonio Paz
2	Lilliana Angelovska
3	André Dopund
4	René Dupn

Varie grandement d'un SGBD à un autre

Liens

- Document classique :
 - Laurent Carmignac. *Programmation et administration des bases de données.*

Crédits

Auteur

Mickaël Martin Nevot

mmartin.nevot@gmail.com

- Laurent Carmignac



Carte de visite électronique

Relecteurs

Cours en ligne sur : www.mickael-martin-nevot.com

