SQL DDL : Data Definition Language

Fabien Coelho
MINES ParisTech

Composé avec LaTeX, révision 3579

SQL – DDL

commandes `CREATE ALTER DROP`

objets `TABLE VIEW SEQUENCE RULE COLLATION`

aussi `FUNCTION OPERATOR TRIGGER AGGREGATE CAST`

encore `INDEX DATABASE SCHEMA USER GROUP ROLE CONVERSION POLICY SERVER TABLESPACE TYPE...`

options `OR REPLACE`, `IF [NOT] EXISTS`, `CASCADE`/`RESTRICT`

Différents types de bases, installés par défaut

booléens `BOOL BOOLEAN`

entiers `INT(EGER) INT[248] SMALL,,BIGINT SMALL,,BIGSERIAL NUMERIC`

flottants `FLOAT4 REAL FLOAT8 FLOAT DOUBLE PRECISION DECIMAL`

textes `CHAR(8) VARCHAR(12) TEXT`

temps `DATE TIME TIMETZ TIMESTAMP INTERVAL`

fonctions spéciales `NOW() CURRENT_DATE CURRENT_TIME`

arithmétique de date et d'intervales...

binaires `BIT(12) VARBIT(24) BYTEA`

extensions géométrie, argent, réseau, bibliothèque

Relations `CREATE` / `ALTER` / `DROP TABLE`

— définition initiale d'une relation

nommage, attributs, types, contraintes...

```
CREATE TABLE
  Personnage( -- nom de la relation
    id INTEGER, -- attribut et type...
    nom TEXT,
  CONSTRAINT clef_personnage UNIQUE(id));
```

— modification ultérieure

```
ALTER TABLE Personnage ADD COLUMN age INTEGER;
```

— suppression définitive !

```
DROP TABLE Personnage;
```


Position des contraintes

colonne concernée après sa déclaration

```
CREATE TABLE info
  (id INTEGER PRIMARY KEY, nom TEXT NOT NULL);
```

anonyme table peut impliquer plusieurs colonnes

```
CREATE TABLE info
  (id INTEGER, nom TEXT,
   PRIMARY KEY(id),
   CHECK (nom IS NOT NULL));
```

nommée table manipulation explicite possible *(ajout, retrait, suspension)*

```
CREATE TABLE info
  (id INTEGER, nom TEXT,
   CONSTRAINT info_key PRIMARY KEY(id),
   CONSTRAINT info_nom CHECK (nom IS NOT NULL));
```


Types de contraintes

type de donnée de l'attribut !

texte avec taille maximale ou fixe, valeurs numériques...

```
CREATE TABLE SecuriteSociale(
  nom VARCHAR(32), prenom VARCHAR(32)
  secu CHAR(13), clef CHAR(2),
  remboursement NUMERIC(10,2));
```

clef(s) attribut ou groupe d'attribut `UNIQUE`

utilisation d'un index pour garantir l'unicité

```
CREATE TABLE Film
  (id INTEGER UNIQUE, titre TEXT, auteur TEXT,
 -- un auteur ne fait un film qu'une fois ?
   UNIQUE(titre, auteur));
```


la clef primaire `PRIMARY KEY`, implique `UNIQUE NOT NULL`

```
CREATE TABLE Voiture
  (plaque VARCHAR(10) PRIMARY KEY,
   marque TEXT, type TEXT);
CREATE TABLE Individu
  (prenom TEXT, nom TEXT, ne DATE, addr TEXT,
   PRIMARY KEY(prenom, nom));
```

clef étrangère `FOREIGN KEY (...) REFERENCES tab(...)`

```
CREATE TABLE CarteGrise
  (numero TEXT PRIMARY KEY,
   plaque VARCHAR(10) REFERENCES Voiture,
   prenom TEXT, nom TEXT, delivree DATE,
   FOREIGN KEY (prenom,nom) REFERENCES Individu);
```

conséquence si disparition de la clef référencée ! ?

assertion `CHECK(`*condition*`)`

expression booléenne simples (voir aussi `TRIGGER`)

attention, pas de sous requête (actuellement) !

```
CREATE TABLE Soldat
  (prenom TEXT,
   nom TEXT,
   ne DATE,
   -- nom bien défini
   CHECK (LENGTH(prenom)>1 AND LENGTH(nom)>1),
   CONSTRAINT soldat_majeur
     CHECK (CURRENT_DATE-ne >= INTERVAL '18 y')
);
```

facultatif `NOT NULL` ou `NULL` (par défaut peut être nulle)

la plupart des colonnes sont normallement `NOT NULL` !

```
CREATE TABLE Voiture
  (marque TEXT NOT NULL,
   conducteur TEXT NOT NULL,
   passager TEXT NULL);
```

valeur par défaut `DEFAULT expr`

utilisé lors d'une insertion si valeur non spécifiée

```
CREATE TABLE compte
  (insertion TIMESTAMP DEFAULT NOW(),
   combien MONEY DEFAULT MONEY '0',
   libel TEXT DEFAULT 'virement');
```


Vérification automatique des contraintes

```
CREATE TABLE Groupe
  (gid INTEGER PRIMARY KEY,
   nom TEXT NOT NULL,
   CONSTRAINT nom_long CHECK(LENGTH(nom)>2));

INSERT INTO Groupe VALUES(1, 'Beatles'); -- OK
INSERT INTO Groupe VALUES(1, 'Pink Floyd');
-- ERROR: duplicate key violates
--   unique constraint "groupe_pkey"
INSERT INTO Groupe VALUES(2, 'U2');
-- ERROR: new row for relation "groupe" violates
--   check constraint "nom_long"
INSERT INTO Groupe VALUES(3, NULL);
-- ERROR: null value in column "nom" violates
--   not-null constraint
```


Vérification automatique des contraintes (suite)

```
CREATE TABLE Artiste
  (aid INTEGER NOT NULL,
   nom TEXT NOT NULL,
   gid INTEGER NOT NULL REFERENCES Groupe);

INSERT INTO Artiste VALUES(1, 'John', 1); -- OK
INSERT INTO Artiste VALUES(2, 'Paul', 1); -- OK
INSERT INTO Artiste VALUES(3, 'Syd', 7);
-- ERROR: insert or update on table "Groupe"
--  violates foreign key constraint "$1"
--  key gid=7 is not present in table "Groupe".

DELETE FROM Artiste WHERE aid=1;
-- ERROR:  update or delete on "artiste"
--  violates foreign key constraint "$1" on "groupe"
--  key gid=1 is still referenced from table "groupe".
```


Vérification des contraintes décalée dans le temps

- décalable : `DEFERRABLE` vs `NOT DEFERRABLE`
- si décalable : `INITIALY IMMEDIATE` vs `INITIALY DEFERRED`

```
CREATE TABLE Auteur
  (aid INTEGER PRIMARY KEY, nom TEXT);

CREATE TABLE Poeme
 (aid INTEGER NOT NULL REFERENCES Auteur
    DEFERRABLE INITIALLY DEFERRED,
  titre TEXT);

BEGIN;
INSERT INTO Poeme VALUES(1, 'La prose du Transibérien...');
-- attend le commit pour vérifier l'auteur
INSERT INTO Auteur VALUES(1, 'Blaise Cendrar');
COMMIT;
```


Comportement lors de changements : `ON DELETE/ON UPDATE`

`NO ACTION` génère une erreur (défaut)

`RESTRICT` génère une erreur, mais non décalable

`CASCADE` modifie la référence ou efface le tuple

`SET NULL` retire la référence

`SET DEFAULT` valeur par défaut de la référence

Options diverses

`TEMPORARY` table détruite à la fin de la session

`UNLOGGED` rapide mais pas de transactions (pas de WAL)

`FILLFACTOR` taux de remplissage $100 - n\%$

- impact négatif : stockage, `SELECT INSERT`
- impact positif : `UPDATE`

`autovacuum_*` gestion du ramasse miettes

Séquence `CREATE` / `ALTER` / `DROP SEQUENCE`

- entier `INT8` géré par la base de données
  incrémentation automatique avec fonction `NEXTVAL`
- valeurs de départ (1), incrément (1), cyclique ou non...
- utilisé pour des numérotations automatiques

```
CREATE SEQUENCE compteur
  START WITH 123
  INCREMENT BY 1
  NO CYCLE;

SELECT NEXTVAL('compteur');
```


Numérotation automatique `SERIAL`

- extension spécifique PostgreSQL
- très utile pour clef primaire entière
- équivalent à `INT4` + `CREATE SEQUENCE` + `DEFAULT`
- utilisation typique : `SERIAL PRIMARY KEY`

```
CREATE TABLE info1
  (id SERIAL PRIMARY KEY, contenu TEXT);

CREATE SEQUENCE info_sq;
CREATE TABLE info2
 (id INT4 PRIMARY KEY DEFAULT NEXTVAL('info_sq'),
  contenu TEXT);
```

Numérotation automatique (suite)

- standard SQL `GENERATED ... AS IDENTITY`
- options de génération `BY DEFAULT` vs `ALWAYS`
- forçage volontaire reste possible...

```
CREATE TABLE Foo
  (id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
   ...);
CREATE TABLE Bla
  (id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
   ...);
```


Quelques conseils

- identificateurs en minuscules `id nom adresse`
  éventuellement tables capitalisées ? `Artiste Groupe`
- prévoir une clef primaire entière `pid cid gid`
  de préférence `SERIAL` + `PRIMARY KEY`
  simple, courte, non-sémantique donc pas de modification ?
- nommer de la même façon les clefs primaires et étrangères ?
  permet des jointures naturelles `NATURAL JOIN`
- attention aux attributs homonymes pour les jointures naturelles
  en fait, éviter les jointures naturelles !
- savoir mettre (transactionnel)
  ou ne pas mettre (décisionnel, copie)
  ou mettre plus tard (chargement)

Vues `CREATE` / `ALTER` / `DROP VIEW`

- **relation virtuelle** créée à partir d'un `SELECT`
- compatibilité ascendante si évolution : anciennes tables en vues
- unité de gestion des droits d'accès
- simplification de requêtes, d'attributs calculés...
- vues matérialisées, modifiables...

```
-- les hauts ne sont pas accessibles !
CREATE VIEW bas_salaires AS
  SELECT nom, prenom, salaire
  FROM salaires AS s, personnes AS p
  WHERE s.pid=p.pid
    AND salaire < MONEY '30000';

GRANT SELECT ON bas_salaires TO PUBLIC;
```


Vues matérialisées

`CREATE/REFRESH/ALTER/DROP MATERIALIZED VIEW`

- **relation calculée persistante**
  recalculée à la demande avec `REFRESH`
- équivalent à `CREATE TABLE AS` + recalcul
- option `WITH NO DATA` : initialement vide et non accessible

```
-- création et remplissage initial
CREATE MATERIALIZED VIEW stuff AS
SELECT ...;

-- mise à jour du contenu, les anciennes valeurs
-- restent accessibles pendant le recalcul
REFRESH MATERIALIZED VIEW stuff CONCURRENTLY;
```


Rules `CREATE` / `ALTER` / `DROP RULE`

- réécritures systématique de `SELECT INSERT DELETE UPDATE`
  remplacement `INSTEAD` ou complément `ALSO`
- permet insertion/effacement/mise à jour sur une vue
- compatibilité d'une application lors des évolutions...
- spécifique PostgreSQL

```
CREATE RULE les_eleves_insert AS
ON INSERT TO les_eleves DO INSTEAD (
 INSERT INTO personne VALUES(new.id,new.nom,new.prenom);
 INSERT INTO eleve VALUES(new.id,new.classe);
);
```


Collation – tri textuel selon langue `COLLATION`

- déclaration de la langue : tri, indexation
- support niveau système...

```
CREATE COLLATION FR (LOCALE = 'fr_FR.UTF-8');
CREATE COLLATION EN (LOCALE = 'en_US.UTF-8');
CREATE COLLATION DE (LOCALE = 'de_DE.UTF-8');
CREATE COLLATION SP (LOCALE = 'es_ES.UTF-8');
CREATE TABLE Film(
    fid SERIAL PRIMARY KEY,
    titre TEXT COLLATE FR NOT NULL,
    title TEXT COLLATE EN NOT NULL
    titel TEXT COLLATE DE NOT NULL,
    título TEXT COLLATE SP NOT NULL);
```


List of Slides