Aide-Mémoire SQL PostgreSQL

Langage de définition des données

- Créer une base de données nommée `refcard` :

```
CREATE DATABASE refcard;
```

Se connecter à cette nouvelle base de données.

- Créer un schéma nommée `myevents` et l'utiliser :

```
CREATE SCHEMA myevents;
SET search_path = myevents, public;
ALTER DATABASE refcard
  SET search_path FROM CURRENT;
```

- Créer deux tables nommées `users` et `schedule` :

```
CREATE TABLE users (
  id bigint GENERATED BY DEFAULT AS IDENTITY,
  owner text,
  e_mail text,
  valid boolean NOT NULL DEFAULT true );

CREATE TABLE schedule (
  id bigint GENERATED BY DEFAULT AS IDENTITY,
  owner bigint NOT NULL,
  date_ev tstzrange NOT NULL,
  description text,
  metadata jsonb );
```

- Modifier les tables pour ajouter des clés primaires :

```
ALTER TABLE users ADD PRIMARY KEY ( id );
ALTER TABLE schedule ADD PRIMARY KEY ( id );
```

Chaque contrainte crée un index `btree`.

- Ajouter une clé étrangère entre les deux tables :

```
ALTER TABLE schedule ADD FOREIGN KEY ( owner )
  REFERENCES users ( id );
```

- Crées des index GIN :

```
CREATE INDEX schedule_metadata_idx
  ON schedule USING GIN (metadata);

CREATE INDEX schedule_desc_tsvector_idx
  ON schedule
  USING GIN (to_tsvector('english', description ));
```

- Ajouter une extension et l'utiliser pour créer un index :

```
CREATE EXTENSION pg_trgm ;
CREATE INDEX users_owner_idx
  ON users USING GIN (owner gin_trgm_ops);
```

- Créer une contrainte d'exclusion :

```
CREATE EXTENSION btree_gist ;
ALTER TABLE schedule
  ADD CONSTRAINT unique_event_by_owner
  EXCLUDE USING gist
  ( owner WITH =, date_ev WITH && );
```

- Création d'une vue :

```
CREATE VIEW schedule_working_hours AS
SELECT *
FROM schedule
WHERE lower(date_ev)::time
  BETWEEN time '9:00' and time '18:00';
```

Gestion des rôles

- Créer des rôles :

```
CREATE ROLE events_grp;
CREATE ROLE events_owner;
CREATE ROLE user_tests;
CREATE ROLE app_events LOGIN PASSWORD 'azerty'
  IN ROLE events_grp;
```

- Définir un propriétaire :

```
ALTER DATABASE refcard OWNER TO events_grp;
ALTER SCHEMA myevents OWNER TO events_grp;
ALTER TABLE users OWNER TO events_grp;
```

- Réassigner des rôles :

```
REASSIGN OWNED BY events_grp TO events_owner;
```

- Ajouter un rôle comme membre d'un autre rôle :

```
GRANT events_owner TO user_tests
  WITH INHERIT FALSE;
```

- Donner des autorisations à des rôles :

```
GRANT USAGE ON SCHEMA myevents TO events_grp;
GRANT SELECT, INSERT, UPDATE, DELETE
  ON TABLE users, schedule TO events_grp;
```

- Supprimer des rôles :

```
DROP ROLE IF EXISTS user_tests;
```

Langage de manipulation des données

- Ajouter des entrées dans la table `users` :

```
INSERT INTO users (owner, e_mail, valid)
VALUES ('Elaine Robinson'
       , 'e.rob@myhoster.dom', default );
```

- Ajouter des entrées dans la table `schedule` :

```
INSERT INTO schedule (owner, date_ev, description, metadata)
VALUES ( 2, '[2026-01-07 18:30+01:00, 2026-01-07 23:30+01)'
       , 'meeting with Mrs.', '{"location": "her home"
          , "attendees": ["Glenda", "Me"]}'::jsonb);
```

- Copier des données depuis un fichier :

```
COPY users FROM
  PROGRAM 'zcat /tmp/data/refcard.users.csv.gz'
  WITH FORMAT CSV;
```

- Copier des données vers un fichier :

```
COPY schedule TO '/tmp/data/refcard.schedule.csv'
  WITH FORMAT CSV;
```

- Mettre à jour la table `users` :

```
UPDATE users SET valid = true
WHERE e_mail = 'cs@smith.uny' ;
```

- Gestion des conflits à l'insertion :

```
INSERT INTO users as u (id, owner, e_mail, valid )
VALUES (5, 'Carl Smith', 'cs@smith.uny', true )
      ON CONFLICT (id)
      DO UPDATE SET valid = EXCLUDED.valid
      WHERE u.valid <> EXCLUDED.valid ;
```

- Mettre à jour ou insérer des données :

```
MERGE INTO users AS u
  USING (VALUES (5, 'Carl Smith', 'cs@smith.uny', true ) )
        AS d (id, owner, e_mail, valid )
  ON u.id = d.id
  WHEN MATCHED AND u.valid <> d.valid
    THEN UPDATE SET valid = d.valid
  WHEN NOT MATCHED
    THEN INSERT (id, owner, e_mail, valid )
      VALUES ( d.id, d.owner, d.e_mail, d.valid );
```

- Supprimer des enregistements :

```
DELETE FROM schedule
  WHERE date_ev && '[2024-01-07 17:05+00:00
                  , 2024-01-07 17:10:00+00)';
```

- Vider des tables et ré-initialiser les séquences :

```
TRUNCATE TABLE users, schedule RESTART IDENTITY;
```

Langage de requêtage des données

- Joindre les données des tables :

```
SELECT u.id, u.owner
      , lower( date_ev ) AS start_ev , description
  FROM users AS u
      INNER JOIN schedule AS p ON u.id=p.owner
 WHERE u.id = 2
 ORDER BY start_ev DESC ;
```

- Utiliser une fonction fenêtrée :

```
SELECT u.id, u.owner
      , lower( date_ev ) AS start_ev
      , count( p.id ) OVER( PARTITION BY
         extract ('month' from lower( date_ev )))
  FROM users AS u
      INNER JOIN schedule AS p ON u.id=p.owner;
```

- Utiliser une expression « CTE » :

```
WITH cte_schedule AS (
SELECT u.id, u.owner , lower( p.date_ev ) AS start_ev
  , upper(lag( p.date_ev )
    OVER( PARTITION BY u.id ORDER BY p.date_ev )
      ) AS previous_end_ev
  FROM users AS u
      INNER JOIN schedule AS p ON u.id=p.owner )
SELECT id, owner, start_ev
      , start_ev - previous_end_ev AS itv_wh_prev_ev
  FROM cte_schedule;
```

- Utiliser une expression JSONPATH :

```
SELECT id
  , jsonb_path_query( metadata, '$.attendees.size()')
    as nb_attendees
  FROM myevents.schedule;
```

- Utiliser un regroupement et un agrégat :

```
SELECT count(*), metadata->'location' as location
  , date_trunc('day', lower(date_ev)) as day
  FROM schedule
 GROUP BY ROLLUP( metadata->'location'
              , date_trunc('day', lower(date_ev)))
 ORDER BY location, day;
```

Langage de contrôle des transactions

- Essayer quelques `INSERT` :

```
BEGIN;
INSERT INTO schedule ( owner, date_ev
  , description, metadata ) VALUES
  ( 1, '[2026-01-07 19:30+01:00, 2026-01-07 21:30+01)'
    , 'meeting with Elain', '{"location": ["at the zoo"]
      , "attendees": {"Her": 1, "Me": 1}}'::jsonb);
INSERT INTO schedule ( owner, date_ev
  , description, metadata ) VALUES
  ( 2, '[2026-01-07 19:30+01:00, 2026-01-07 21:30+01)'
    , 'meeting with Ben', '{"location": ["at the zoo"]
      , "attendees": {"Him": 1, "Me": 1}}'::jsonb);
COMMIT;
```

Étant donnée la contrainte d'exclusion, le second `INSERT` échoue, annulant la transaction.

Commandes de maintenance

- Nettoyer la table et collecter les statistiques :

```
VACUUM ANALYZE users, schedule ;
```

- Reconstruire la table dans un nouveau fichier :

```
VACUUM FULL users ;
```

- Retirer les identifiants de transaction pour les lignes les plus anciennes :

```
VACUUM FREEZE schedule ;
```

- Reconstruire un index :

```
REINDEX INDEX CONCURRENTLY schedule_desc_tsvector_idx ;
```

- Examiner le plan d'exécution d'une requête :

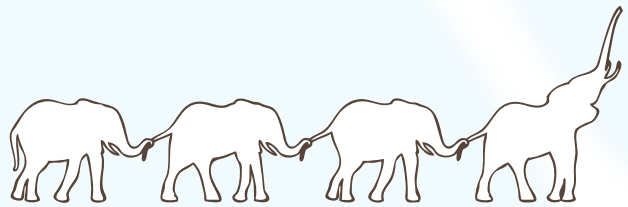
```
EXPLAIN (ANALYZE ON, COSTS OFF )
  SELECT id
    , jsonb_path_query( metadata, '$.attendees.size()')
      as nb_attendees FROM myevents.schedule;

ProjectSet (actual time=0.017..0.022 rows=6.00 loops=1)
  Buffers: shared hit=1
  -> Seq Scan on schedule (actual time=0.009..0.009
                                  rows=6.00 loops=1)
       Buffers: shared hit=1
Planning Time: 0.051 ms
Execution Time: 0.038 ms
```


téléchargez moi : https://www.loxodata.com/refcard

LOXODATA

Votre partenaire PostgreSQL

PostgreSQL
Aide-mémoire SQL



contact@loxodata.com +33 1 797 2 5775
www.loxodata.com