ISEN ALL IS DIGITAL! OUEST | yncréa

Le langage SQL

Avec PostgreSQL

benoit.lardeux@isen-ouest.yncrea.fr

Le langage SQL

- SEQUEL proposé en 1974 par D. Chamberlain
- 1976/1977 devient SQL: Structured Query Language
- 1986: Standardisation ANSI puis ISO
- Nouveaux standards en 1989, 1992, 1999, 2003, 2008, 2016

Le langage SQL

- Langage déclaratif non procédural
- De requêtes
- Commandes de contrôle dans des extensions
  - Procédural: PL/SQL, PL/pgSQL, mySQL/SPL, …
  - Intégré: Pro*C, ECPG, SQLJ, …
- Drivers pour les différents langages: ODBC, JDBC, mysqli,…

Le langage SQL

- Plusieurs parties
  - Langage de définition de données (LDD): définition de la structure (CREATE, ALTER,…)
  - Langage de manipulation de données (LMD): mise à jour des données (INSERT, UPDATE, DELETE)
  - Langage d'interrogation (LID): SELECT
  - Langage de contrôle (LCD): sécurité (GRANT, COMMIT, REVOKE, …)

Le langage SQL: LDD

Langage de définition des données

- Pour les tables, les vues, les index
- 3 commandes:
  - CREATE
  - ALTER
  - DROP

Le langage SQL: LDD

Création de table: CREATE TABLE

- Terminal de commande
- Outil de modélisation
- Requêteur (phpPgAdmin)

Le langage SQL: LDD

Types de données « caractères »

- CHAR: chaîne de longueur fixe (complétée par des blancs)
- VARCHAR: chaîne de longueur variable (longueur réelle stockée)
- Chaîne longue: TEXT
  - Dépend du SGBD
  - A utiliser avec discernement
  - Restriction sur les requêtes
  - Peut contenir plusieurs Go

Le langage SQL: LDD

Types de données « numériques »

- Dépend du SGBD
- Entiers: BIGINT (8 octets), SMALLINT (2 octets)
- Décimal: NUMERIC (p,s) (4 à 8 octets) dont la précision est à spécifier

FLOAT (4 octets), DOUBLE (8 octets)

Le langage SQL: LDD

Types de données « date et heure »

- Dépend du SGBD
- Date: DATE (AAAA-MM-JJ)
- Heure: TIME (HH:MM:SS)
- Date et heure: TIMESTAMP (AAAA-MM-JJ HH:MM:SS)

Le langage SQL: LDD

Types de données « BYTEA »

- Dépend du SGBD
- Stockage de données binaires
  - PostgreSQL: BYTEA (1Go)
  - Oracle: BLOB (1To)

Le langage SQL: LDD

Les contraintes d'intégrité

- Données obligatoires: NOT NULL
- Contrainte de domaine
- Contrainte d'unicité ou clé primaire
  - Sur un champs unique
  - Sur plusieurs champs

Le langage SQL: LDD

Les contraintes de domaine

- Possibilité de créer un nouveau domaine de valeurs
  - Ex: domaine sans valeur nulle ni espace dans le nom

```
CREATE DOMAIN service_name AS VARCHAR
     NOT NULL CHECK (value !~ '\s');

CREATE TABLE service
(
                    num_service INTEGER,
                    type_service service_name,
                    ville VARCHAR (20) NOT NULL,
    PRIMARY KEY(num_service) );
```

Le langage SQL: LDD

Les contraintes d'intégrité

- Contrainte d'unicité ou clé primaire
  - Sur un champs unique

```
CREATE TABLE service
(
                    num_service INTEGER,
                    type_service VARCHAR (20) NOT NULL,
                    ville VARCHAR (20) NOT NULL,
    PRIMARY KEY(num_service) );
```

Le langage SQL: LDD

Les contraintes d'intégrité

- Contrainte d'unicité ou clé primaire
  - Sur plusieurs champs

```
CREATE TABLE service
(
                    num_service INTEGER,
                    type_service VARCHAR (20) NOT NULL,
                    ville VARCHAR (20) NOT NULL,
    PRIMARY KEY(num_service, type_service) );
```

Le langage SQL: LDD

Les contraintes d'intégrité référentielle

- Contrainte sur les clés étrangères
  - Syntaxe simple
  - Syntaxe avec action référentielle

Le langage SQL: LDD

Les contraintes d'intégrité référentielle

- Contrainte sur les clés étrangères
  - Syntaxe simple

```
CREATE TABLE employe
(
                    numemp INTEGER,
                    nom VARCHAR (20) NOT NULL,
                    fonction VARCHAR (10) NOT NULL,
                    numemp_sup INTEGER,
                    date_embauche DATE NOT NULL,
                    salaire NUMERIC(7,2) NOT NULL,
                     comm NUMERIC(7,2),
                    num_service INTEGER NOT NULL,
    PRIMARY KEY(numemp),
    FOREIGN KEY (num_service) REFERENCES service (num_service),
    FOREIGN KEY (numemp_sup) REFERENCES employe(numemp) );
```

Le langage SQL: LDD

Les contraintes d'intégrité référentielle

- Contrainte sur les clés étrangères
  - Syntaxe avec action référentielle

```
CREATE TABLE employe
(
                    numemp INTEGER,
                    nom VARCHAR (20) NOT NULL,
                    fonction VARCHAR (10) NOT NULL,
                    numemp_sup INTEGER,
                    date_embauche DATE NOT NULL,
                    salaire NUMERIC(7,2) NOT NULL,
                     comm NUMERIC(7,2),
                    num_service INTEGER NOT NULL,
    PRIMARY KEY(numemp),
    FOREIGN KEY (num_service) REFERENCES service (num_service) ON
DELETE CASCADE,
    FOREIGN KEY (numemp_sup) REFERENCES employe(numemp) );
```

Le langage SQL: LDD

Les clauses optionnelles

- TABLESPACE (Oracle ou PostgreSQL)
  - Désigne l'emplacement dans le système de fichiers ou seront stockés les objets de la base de données (tables, index, …)

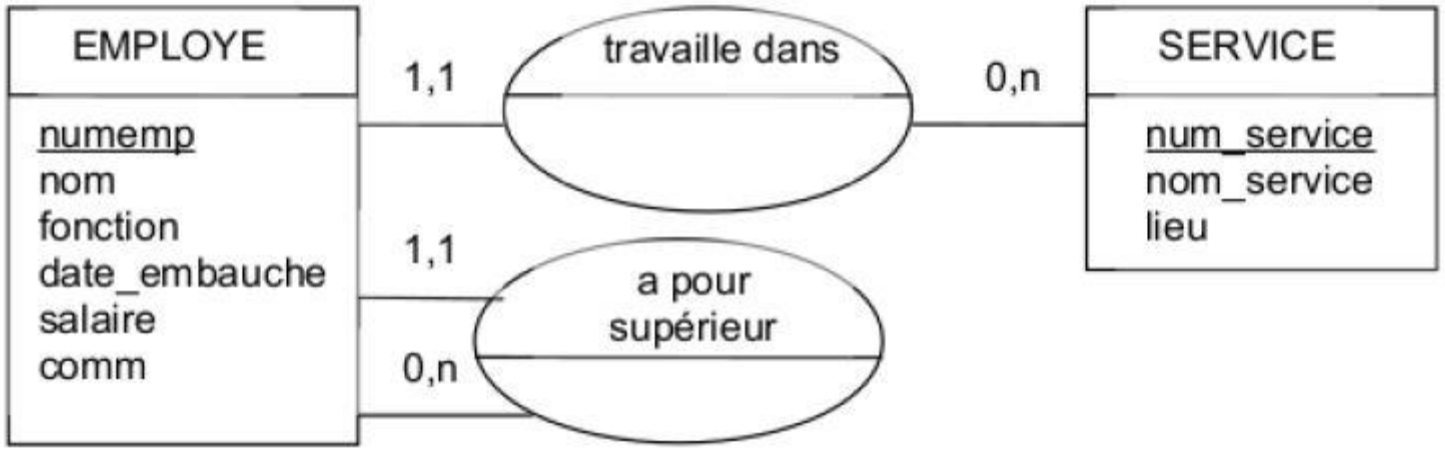
```
CREATE TABLESPACE rh LOCATION '/mnt/sda1/postgresql/data';

CREATE TABLE service
(
                        num_service INTEGER,
                        type_service VARCHAR (20),
                        ville VARCHAR (20) NOT NULL,
    PRIMARY KEY(num_service) ) TABLESPACE rh;
```

Le langage SQL: LDD

Exemple de création de tables

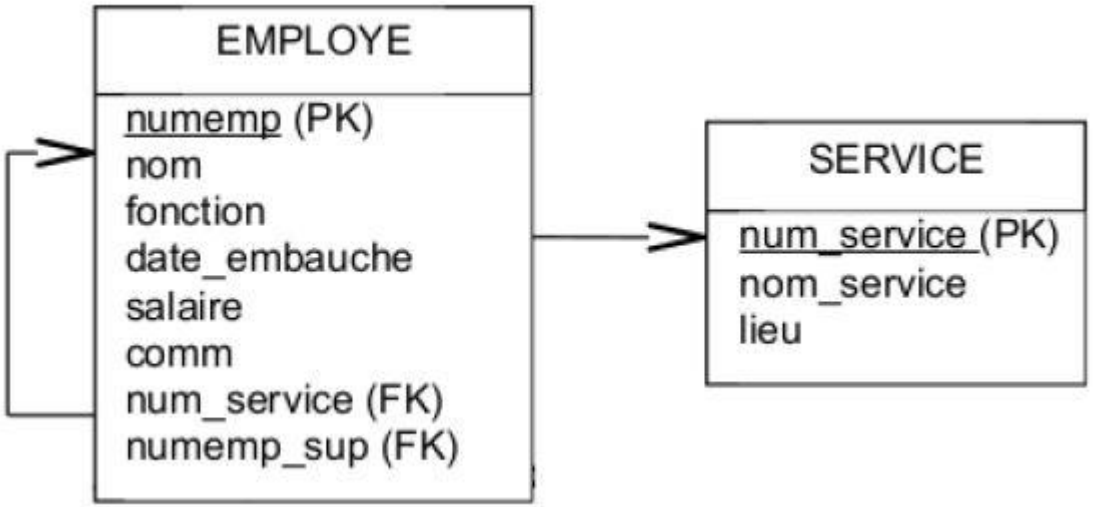
- MCD



Le langage SQL: LDD

Exemple de création de tables

- MPD



Le langage SQL: LDD

Exemple de création de tables

```
CREATE TABLE service
(
                    num_service INTEGER,
                    type_service VARCHAR (20) NOT NULL,
                    ville VARCHAR (20) NOT NULL,
    PRIMARY KEY(num_service) );
```

Le langage SQL: LDD

Exemple de création de tables

```
CREATE TABLE employe
(
                    numemp INTEGER,
                    nom VARCHAR (20) NOT NULL,
                    fonction VARCHAR (10) NOT NULL,
                    numemp_sup INTEGER,
                    date_embauche DATE NOT NULL,
                    salaire NUMERIC(7,2) NOT NULL,
                     comm NUMERIC(7,2),
                    num_service INTEGER NOT NULL,
    PRIMARY KEY(numemp),
    FOREIGN KEY (num_service) REFERENCES service (num_service),
    FOREIGN KEY (numemp_sup) REFERENCES employe(numemp) );
```

Le langage SQL: LDD

Création de table à partir d'un select

- Les champs sont créés à l'identique

```
CREATE TABLE employe_bis
     AS SELECT numemp, nom, salaire FROM employe;
```

Le langage SQL: LDD

Création avec séquence

- Clé artificielle

Oracle

```
create sequence seq_emp increment by 1 minvalue 1;
      CURRVAL
      NEXTVAL
```

PostgreSQL

```
CREATE TABLE service
(
                        num_service SERIAL,
                        type_service VARCHAR (20) NOT NULL,
                        ville VARCHAR (20) NOT NULL,
    PRIMARY KEY(num_service) );
```

Le langage SQL: LDD

Création des vues

- Table virtuelle (sur 1 ou plusieurs tables)
- Mise à jour pas implémentée sur tous les SGBDR

```
CREATE VIEW employe_comm AS SELECT numemp, nom, fonction,
numemp_sup, date_embauche FROM employe WHERE num_service = 30;
```

Le langage SQL: LDD

Création des indexes

- Permet de retrouver plus rapidement une information
- Stockage très important
- Indexes générés de base pour les clés primaires (unique) et clés étrangères (non unique)

```
CREATE [UNIQUE] INDEX title_idx ON films (title);
```

Le langage SQL: LDD

Destruction des tables avec DROP

- Permet la destruction des tables mais aussi des vues et des indexes

```
drop table matable;
```

- Pas de retour en arrière possible!!!

- Commande de destruction de la base de données

```
drop database mabase;
```

Le langage SQL: LDD

Modifications avec ALTER

- Permet de modifier les caractéristiques d'une table

- ADD: ajoute un champs

```
alter table matable add prenom Varchar(50) not null;
```

- MODIFY: modifie le type d'un champs

```
alter table matable modify col1 Numeric(7,2);
```

- RENAME: renomme la table

```
alter table matable rename to mytable;
```

- Ajoute ou détruit des contraintes d'intégrité

```
alter table matable set col1 not null;
```

Le langage SQL: LID

- Les jointures
  - Opération de récupération des données de plusieurs tables simultanément
  - Différents types de jointure
    - Jointure simple (equi-jointure)
    - Auto-jointure
    - Jointure externe

Le langage SQL: LID

- Les jointures implicites
  - Les tables se trouvent dans FROM
  - Les critères de jointures se trouvent dans le WHERE (critère de jointure)
  - Si plus de 2 tables, les critères de jointure se font de table à table
  - Le critère de jointure se fait sur des colonnes (sur chaque table) de même type et de même valeur
    - Typiquement clé étrangère => clé primaire

Le langage SQL: LID

- Les jointures implicites
  - SELECT * FROM table1,table2 WHERE table1.col1 = table2.col2
  - SELECT * FROM table1 t1, table2 t2 WHERE t1.col1 = t2.col2

Le langage SQL: LID

- Les jointures explicites
  - Utilisation du terme JOIN ou INNER JOIN
  - SELECT * FROM table1 JOIN table2 ON table1.col1 = table2.col2
  - SELECT * FROM table1 t1 JOIN table2 t2 ON t1.col1 = t2.col2

Le langage SQL: LID

- Les jointures explicites
  - Lorsque la colonne porte le même nom dans les deux tables
  - SELECT * FROM table1 JOIN table2 USING (col)
  - SELECT * FROM table1 NATURAL JOIN table2

Le langage SQL: LID

- Les jointures simples

```
tp1=> select e.nom, s.ville from employe e, service s where e.num_service = s.num_service;
   nom    | ville
----------+-------
 Leroy    | Paris
 Dupont   | Brest
 Duguet   | Brest
 Lefevre  | Brest
 Lenoir   | Lyon
 Lefort   | Lyon
 Garde    | Lyon
 Martin   | Lyon
 Leclerc  | Paris
 Lescaut  | Brest
 Tournier | Lyon
 Adam     | Brest
 Jan      | Lyon
 Meunier  | Paris
(14 rows)
```

Le langage SQL: LID

- Les jointures simples

```
tp1=> select e.nom, s.ville from employe e join service s on e.num_service = s.num_service;
   nom    | ville
----------+-------
 Leroy    | Paris
 Dupont   | Brest
 Duguet   | Brest
 Lefevre  | Brest
 Lenoir   | Lyon
 Lefort   | Lyon
 Garde    | Lyon
 Martin   | Lyon
 Leclerc  | Paris
 Lescaut  | Brest
 Tournier | Lyon
 Adam     | Brest
 Jan      | Lyon
 Meunier  | Paris
(14 rows)
```

Le langage SQL: LID

- Les auto-jointures
  - Jointure d'une table sur elle-même
  - Le critère de jointure doit exister

```
tp1=> select e.nom, es.nom as "superieur" from employe e join employe es on es.numemp = e.numemp_sup;
   nom    | superieur
----------+-----------
 Dupont   | Leroy
 Duguet   | Dupont
 Lefevre  | Duguet
 Lenoir   | Leroy
 Lefort   | Lenoir
 Garde    | Lenoir
 Martin   | Lenoir
 Leclerc  | Leroy
 Lescaut  | Dupont
 Tournier | Lenoir
 Adam     | Lescaut
 Jan      | Lenoir
 Meunier  | Leclerc
(13 rows)
```

Le langage SQL: LID

- Les jointures externes
  - Renvoie tous les résultats (à l'inverse des jointures internes)
  - Utilisation du terme OUTER JOIN
  - LEFT OUTER JOIN (LEFT JOIN avec PostgreSQL) renvoie tous les enregistrements de la table de gauche complétés par les infos de la table de droite si elles existent
  - RIGHT OUTER JOIN est peu utilisé

Le langage SQL: LID

- Les jointures externes
  - Jointure gauche

```
tp1=> select e.nom, es.nom as "superieur" from employe e left join employe es on es.numemp = e.numemp_sup;
   nom    | superieur
----------+-----------
 Leroy    |
 Dupont   | Leroy
 Duguet   | Dupont
 Lefevre  | Duguet
 Lenoir   | Leroy
 Lefort   | Lenoir
 Garde    | Lenoir
 Martin   | Lenoir
 Leclerc  | Leroy
 Lescaut  | Dupont
 Tournier | Lenoir
 Adam     | Lescaut
 Jan      | Lenoir
 Meunier  | Leclerc
(14 rows)
```

Le langage SQL: LID

- Les jointures externes
  - Jointure droite

```
tp1=> select e.nom, s.ville from employe e right join service s on e.num_service = s.num_service;
   nom    | ville
----------+-------
 Leroy    | Paris
 Dupont   | Brest
 Duguet   | Brest
 Lefevre  | Brest
 Lenoir   | Lyon
 Lefort   | Lyon
 Garde    | Lyon
 Martin   | Lyon
 Leclerc  | Paris
 Lescaut  | Brest
 Tournier | Lyon
 Adam     | Brest
 Jan      | Lyon
 Meunier  | Paris
          | Lille
(15 rows)
```

Le langage SQL: LID

- Les jointures simples sur plusieurs tables

```
tp1=> select e.nom, s.ville, v.code_postal from employe e, service s, ville_adresse v where e.num_service = s.num_service and s.ville = v.ville;
   nom    | ville | code_postal
----------+-------+-------------
 Leroy    | Paris |       75000
 Dupont   | Brest |       29000
 Duguet   | Brest |       29000
 Lefevre  | Brest |       29000
 Lenoir   | Lyon  |       69000
 Lefort   | Lyon  |       69000
 Garde    | Lyon  |       69000
 Martin   | Lyon  |       69000
 Leclerc  | Paris |       75000
 Lescaut  | Brest |       29000
 Tournier | Lyon  |       69000
 Adam     | Brest |       29000
 Jan      | Lyon  |       69000
 Meunier  | Paris |       75000
(14 rows)
```

Le langage SQL: LID

- Les opérateurs ensemblistes
  - UNION
  - Renvoie l'ensemble des lignes de table1 et table 2

Le langage SQL: LID

- Les opérateurs ensemblistes
  - INTERSECT
  - Renvoie les lignes communes aux table1 et table 2

Le langage SQL: LID

- Les opérateurs ensemblistes
  - MINUS
  - Renvoie les lignes du premier SELECT moins les lignes communes aux table1 et table2

Le langage SQL: LID

- Les opérateurs ensemblistes
  - Attention aux contraintes
    - Même nombre de colonnes avec le même type
    - Doublons supprimés
    - ORDER BY sur numéro de colonnes
    - Combinaison possible sur plus de 2 SELECT

Le langage SQL: LID

- Les opérateurs ensemblistes

```
tp1=> select * from employe union select * from service;
ERROR:  each UNION query must have the same number of columns
LINE 1: select * from employe union select * from service;
```

Le langage SQL: LID

- Les requêtes imbriquées
  - Sous select sur une valeur
  - Sous select évalué en premier
  - Plusieurs sous-select imbriqués possible

```
tp1=> select nom, fonction from employe where fonction = (select fonction from employe where nom = 'Dupont');
   nom    | fonction
----------+-----------
 Dupont   | directeur
 Lenoir   | directeur
 Leclerc  | directeur
(3 rows)
```

Le langage SQL: LID

- Les requêtes imbriquées
  - Sous select sur une valeur
  - Sous select évalué en premier
  - Plusieurs sous-select imbriqués possible

```
tp1=> select nom, fonction from employe where fonction = (select fonction from employe where nom = 'Dupont') and salaire < (select salaire from employe where nom = 'Dupont');
   nom    | fonction
----------+-----------
 Lenoir   | directeur
 Leclerc  | directeur
(2 rows)
```

Le langage SQL: LID

- Les requêtes imbriquées
  - Sous select synchronisé
  - Sous select évalué à chaque itération
  - Couteux
  - Souvent remplaçable par jointure

```
tp1=> select nom from employe e where num_service != (select num_service from employe where numemp = e.numemp_sup);
  nom
--------
 Dupont
 Lenoir
(2 rows)
```

Le langage SQL: LID

- Les requêtes imbriquées
  - Sous select avec test d'existence
  - Sous select évalué à chaque itération
  - Couteux
  - Souvent remplaçable par jointure

```
tp1=> select num_service, type_service from service s where exists (select * from employe e where e.num_service = s.num_service);
 num_service | type_service
-------------+--------------
          10 | siege
          20 | recherche
          30 | commercial
(3 rows)
```

Le langage SQL: LID

- Les requêtes imbriquées
  - Sous select avec test de non-existence (NOT EXISTS)
  - Sous select évalué à chaque itération
  - Couteux
  - Souvent remplaçable par jointure

```
tp1=> select num_service, type_service from service s where not exists (select * from employe e where e.num_service = s.num_service);
 num_service | type_service
-------------+--------------
          40 | production
(1 row)
```

Le langage SQL: LID

- Les requêtes imbriquées
  - Sous select avec plusieurs lignes
  - Sous select évalué une fois
  - Opérateurs IN et NOT IN
  - Beaucoup utilisé en développement

```
tp1=> select nom from employe e where num_service in (select num_service from service where ville like 'L%');
   nom
----------
 Lenoir
 Lefort
 Garde
 Martin
 Tournier
 Jan
(6 rows)
```

Le langage SQL: LMD

- La manipulation des données: LMD
  - Ajout => INSERT
  - Modification => UPDATE
  - Suppression => DELETE

Le langage SQL: LMD

- L'insertion des données => INSERT

  INSERT INTO table (col1, col2, …)

  VALUES (xxx, yyy, …)

  - Attention aux valeurs null => erreur lorsque le champs NOT NULL et pas de valeurs insérées

  - Respecter les types de champs

Le langage SQL: LMD

- L'insertion des données => INSERT

  INSERT INTO table (col1, col2, …)

  VALUES (xxx, yyy, …)

  - Insertion de plusieurs lignes simultanément

```
tp1=> insert into service (num_service, type_service, ville) values (60, 'logistique', 'Paris'), (70, 'gestion', 'Bordeaux');
INSERT 0 2
```

Le langage SQL: LMD

- L'insertion des données => INSERT

  INSERT INTO table (col1, col2, …)

  SELECT (xxx, yyy, …) FROM table2

  WHERE …

Le langage SQL: LMD

- L'insertion des données => INSERT
  - Insertion avec valeur auto-incrémentée

```
tp1=> create table appuser
(
iduser serial, nom varchar (20), prenom varchar(20));
CREATE TABLE
tp1=> insert into appuser (nom, prenom) values ('LeCarre', 'John');
INSERT 0 1
tp1=> select * from appuser;
 iduser |   nom   | prenom
--------+---------+--------
      1 | LeCarre | John
(1 row)

tp1=> insert into appuser (nom, prenom) values ('LeRouge', 'John');
INSERT 0 1
tp1=> select * from appuser;
 iduser |   nom   | prenom
--------+---------+--------
      1 | LeCarre | John
      2 | LeRouge | John
(2 rows)

tp1=> insert into appuser (iduser, nom, prenom) values (10, 'LeNoir', 'John');
INSERT 0 1
tp1=> select * from appuser;
 iduser |   nom   | prenom
--------+---------+--------
      1 | LeCarre | John
      2 | LeRouge | John
     10 | LeNoir  | John
(3 rows)
```

Le langage SQL: LMD

- La modification de données => UPDATE
  - Mise à jour d'une ou plusieurs lignes

```
UPDATE table
SET col1 = newval, col2 = newval2
WHERE …
```

Attention à la présence du WHERE!!!

Le langage SQL: LMD

- La modification de données => UPDATE

```
dbtp1=> select * from employe where numemp=566;
 numemp |  nom   | fonction  | numemp_sup | date_embauche | salaire | comm | num_service
--------+--------+-----------+------------+---------------+---------+------+-------------
    566 | Dupont | directeur |        839 | 2001-04-02    | 2975.00 |      |          20
(1 row)

dbtp1=> update employe set comm=salaire*0.1 where numemp=566;
UPDATE 1
dbtp1=> select * from employe where numemp=566;
 numemp |  nom   | fonction  | numemp_sup | date_embauche | salaire |  comm  | num_service
--------+--------+-----------+------------+---------------+---------+--------+-------------
    566 | Dupont | directeur |        839 | 2001-04-02    | 2975.00 | 297.50 |          20
(1 row)
```

Le langage SQL: LMD

- La suppression de données => DELETE
  - Suppression d'une ou plusieurs lignes

DELETE FROM table

WHERE …

Attention à la présence du WHERE!!!

Le langage SQL: LMD

- La suppression de données => DELETE

```
tp1=> select * from employe where numemp=900;
 numemp | nom |  fonction   | numemp_sup | date_embauche | salaire | comm | num_service
--------+-----+-------------+------------+---------------+---------+------+-------------
    900 | Jan | secretaire |        698 | 2000-12-03    |  950.00 |      |          40
(1 row)

tp1=> delete from service where num_service=40;
ERROR:  update or delete on table "service" violates foreign key constraint "employe_num_service_fkey" on table "employe"
DETAIL:  Key (num_service)=(40) is still referenced from table "employe".
tp1=> delete from employe where numemp=900;
DELETE 1
tp1=> delete from service where num_service=40;
DELETE 1
```

Le langage SQL: LMD

- Gestion des contraintes d'intégrité

  - Clé primaire: par définition la clé primaire est unique => pas de possibilité d'insérer un enregistrement avec la même clé primaire

Le langage SQL: LMD

- Gestion des contraintes d'intégrité
  - Clé étrangère en insertion mise à jour

  Il y a contrôle au moment de l'insertion pour vérifier que la valeur de la clé étrangère existe

```
tp1=> insert into employe (numemp, nom, fonction, numemp_sup, date_embauche, salaire, comm, num_service) values (901, 'Jean', 'secretaire', 698, '2000-04-22', 990.00, NULL, 99);
ERROR:  insert or update on table "employe" violates foreign key constraint "employe_num_service_fkey"
DETAIL:  Key (num_service)=(99) is not present in table "service".
tp1=> insert into employe (numemp, nom, fonction, numemp_sup, date_embauche, salaire, comm, num_service) values (901, 'Jean', 'secretaire', 698, '2000-04-22', 990.00, NULL, 40);
INSERT 0 1
```

Le langage SQL: LMD

- Gestion des contraintes d'intégrité
  - Clé étrangère en suppression

  Il y a contrôle au moment de la suppression pour vérifier que la valeur de la clé étrangère n'est pas employée

```
tp1=> delete from service where num_service=40;
ERROR:  update or delete on table "service" violates foreign key constraint "employe_num_service_fkey" on table "employe"
DETAIL:  Key (num_service)=(40) is still referenced from table "employe".
tp1=> delete from employe where numemp=901;
DELETE 1
tp1=> delete from service where num_service=40;
DELETE 1
```

Le langage SQL: LMD

- Levée des contraintes d'intégrité
  - Opération risquée

ORACLE

```
ALTER TABLE matable
DISABLE CONSTRAINT macontrainte ;

ALTER TABLE matable
ENABLE CONSTRAINT macontrainte ;
```

MySql

```
Disable
SET FOREIGN_KEY_CHECKS=0;

Enable
SET FOREIGN_KEY_CHECKS=1;
```

Le langage SQL: LMD

- Levée des contraintes d'intégrité
  - Opération risquée

PostgreSQL:

```
ALTER TABLE table1 DISABLE TRIGGER ALL;

ALTER TABLE table1 ENABLE TRIGGER ALL;
```

Le langage SQL: LMD

- Levée des contraintes d'intégrité
  - Lors de la création de tables: ne pas mettre les contraintes dans la déclaration initiale de chaque table, mais en fin de fichier de chargement:

```
ALTER TABLE covoitureur ADD CONSTRAINT
    FK_covoitureur_code_insee FOREIGN KEY (code_insee)
    REFERENCES ville(code_insee);
ALTER TABLE covoitureur ADD CONSTRAINT
    FK_covoitureur_immatriculation FOREIGN KEY
    (immatriculation) REFERENCES
    vehicule(immatriculation);
```

Question?