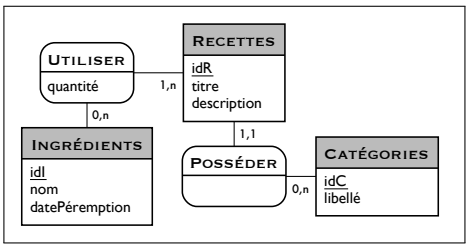
Tutoriel simplifié pour modélisation physique en SQL (PostgreSQL)

UCBL - Département Informatique de Lyon 1 – BDW - automne 2024

1. Une base de données contient un **ensemble de tables (ou relations)**. Ci-contre le schéma relationnel d'une base de données, et le diagramme entité/association correspondant. Ce schéma relationnel comporte 4 tables, que l'on va créer dans la base de données grâce au **langage SQL**.

CATÉGORIES (idC, *libellé*)
INGRÉDIENTS (idI, *nom*, *datePéremption*)
RECETTES (idR, *titre*, *description*, *#idC*)
UTILISER (*#idI*, *#idR*, *quantité*)



2. C'est l'instruction `CREATE TABLE` qui permet de **créer une nouvelle table**. Pour chaque table, on définit ses attributs avec leur type, et éventuellement des contraintes. Ici, la table `Catégories` contient un attribut `idC` de type entier, qui ne peut pas être nul et un attribut libellé de type chaîne de caractères (limitée à 42). La dernière ligne définit une contrainte de clé primaire sur `idC`, qui sert donc d'identifiant. De même pour la table `Recettes`, qui possède un `idR` de type `SERIAL`, c'est à dire un identifiant auto-incrémenté.
Dans le terminal, on saisit l'instruction SQL, puis on affiche le schéma de la table nouvellement créée avec `\d`.

```sql
CREATE TABLE Categories (
  idC INTEGER NOT NULL,
  libelle VARCHAR(42),
  CONSTRAINT pk_categories PRIMARY
  ↪  KEY (idC)
);
```

```sql
CREATE TABLE Recettes (
  idR SERIAL NOT NULL,
  titre VARCHAR(200),
  description TEXT,
  idC INTEGER NOT NULL,
  PRIMARY KEY (idR)
);
```

```sql
CREATE TABLE Ingredients (
  idI INTEGER NOT NULL,
  nom VARCHAR(200),
  datePeremption DATE
);
```

```
fduchateau=> CREATE TABLE Categories (
  idC INTEGER NOT NULL,
  libelle VARCHAR(42),
  CONSTRAINT pk_categories PRIMARY KEY (idC)
);
CREATE TABLE
fduchateau=> \d categories
                    Table "public.categories"
 Column  |         Type          | Collation | Nullable | Default
---------+-----------------------+-----------+----------+---------
 idc     | integer               |           | not null |
 libelle | character varying(42) |           |          |
Indexes:
    "pk_categories" PRIMARY KEY, btree (idc)
```

```
fduchateau=> \d recettes
                                      Table "public.recettes"
  Column     |          Type          | Collation | Nullable |                Default
-------------+------------------------+-----------+----------+---------------------------------------
 idr         | integer                |           | not null | nextval('recettes_idr_seq'::regclass)
 titre       | character varying(200) |           |          |
 description | text                   |           |          |
 idc         | integer                |           | not null |
Indexes:
    "recettes_pkey" PRIMARY KEY, btree (idr)
```

```
fduchateau=> \d ingredients
                    Table "public.ingredients"
     Column     |          Type          | Collation | Nullable | Default
----------------+------------------------+-----------+----------+---------
 idi            | integer                |           | not null |
 nom            | character varying(200) |           |          |
 dateperemption | date                   |           |          |
```

3. L'instruction `ALTER TABLE` permet de **modifier le schéma d'une table** (e.g., renommer un attribut ou changer de type, supprimer une contrainte).
Dans le code ci-contre, on modifie la table `Recettes` pour ajouter une contrainte de clé étrangère sur `idC`, qui référence désormais l'attribut `idC` de la table `Catégories`. L'instruction suivante ajoute une clé primaire dans la table `Ingrédients`.

```sql
ALTER TABLE Recettes
ADD FOREIGN KEY (idC) REFERENCES
↪  Categories (idC);
```

```sql
ALTER TABLE Ingredients ADD PRIMARY
↪  KEY (idI);
```

```
fduchateau=> ALTER TABLE Recettes ADD FOREIGN KEY (idC) REFERENCES Categories (idC);
ALTER TABLE
fduchateau=> \d recettes
                      Table "public.recettes"
   Column    |          Type          | Collation | Nullable | Default
-------------+------------------------+-----------+----------+---------
 idr         | integer                |           | not null |
 titre       | character varying(200) |           |          |
 description | text                   |           |          |
 idc         | integer                |           | not null |
Indexes:
    "recettes_pkey" PRIMARY KEY, btree (idr)
Foreign-key constraints:
    "recettes_idc_fkey" FOREIGN KEY (idc) REFERENCES categories(idc)
```

```
fduchateau=> \d ingredients
                    Table "public.ingredients"
     Column     |          Type          | Collation | Nullable | Default
----------------+------------------------+-----------+----------+---------
 idi            | integer                |           | not null |
 nom            | character varying(200) |           |          |
 dateperemption | date                   |           |          |
Indexes:
    "ingredients_pkey" PRIMARY KEY, btree (idi)
```

4. Avec l'instruction `INSERT`, on **crée des instances (ou tuples)** dans une table, en indiquant une valeur pour chaque attribut de la table. Pour la dernière insertion, on donne la valeur 0 pour l'identifiant (que PostgreSQL remplace par une valeur unique générée par le `SERIAL`). Notez le double guillemet (`''`) pour mettre un guillemet simple.
Dans un terminal, on vérifie le résultat avec une requête `SELECT`.

```
INSERT INTO Categories VALUES(1,
↪  'plat');
INSERT INTO Categories VALUES(2,
↪  'dessert');
INSERT INTO Recettes
↪  VALUES(DEFAULT, 'Cookies
↪  vegan', 'cookies au sirop
↪  d''érable', 2);
```

```
fduchateau=> select * from categories;
 idc | libelle
-----+---------
   1 | plat
   2 | dessert
(2 rows)
```

```
fduchateau=> select * from recettes;
 idr |     titre     |         description          | idc
-----+---------------+------------------------------+-----
   1 | Cookies vegan | cookies au sirop d'érable |   2
(1 row)
```

5. L'instruction `UPDATE` permet de **mettre à jour des instances**. La clause `SET` précise le ou les attributs pour lesquels la valeur sera modifiée. La clause `WHERE` permet de spécifier les instances concernées par la modification (sans condition exprimée par un`WHERE`, toutes les instances de la table sont modifiées).
Ici, on modifie la valeur de l'attribut `libellé` en *plat principal* uniquement pour l'instance identifiée par 1.

```
UPDATE Categories
SET libelle='plat principal'
where idc=1;
```

```
fduchateau=> select * from categories;
 idc |     libelle
-----+----------------
   2 | dessert
   1 | plat principal
(2 rows)
```

6. Pour **supprimer des instances**, on utilise l'instruction `DELETE FROM` en indiquant le nom de la table. La clause conditionnelle `WHERE` permet de sélectionner les instances à supprimer. **Attention** : sans clause `WHERE`, toutes les données de la table sont supprimées !
Dans l'exemple, on supprime la catégorie identifiée par 1 (*plat principal*).

```
DELETE FROM Categories
WHERE idC=1;
```

```
fduchateau=> DELETE FROM Categories
WHERE idC=1;
DELETE 1
fduchateau=> select * from categories;
 idc | libelle
-----+---------
   2 | dessert
(1 row)
```

7. Enfin, on **supprime une table (schéma et instances)** avec l'instruction `DROP TABLE`. L'option `IF EXISTS` permet d'éviter une erreur si la table n'existe pas.
Dans l'exemple, la table `Catégories` est supprimée, et la requête `SELECT` produit donc une erreur.

```
DROP TABLE IF EXISTS Categories;
```

```
fduchateau=> drop table categories;
DROP TABLE
fduchateau=> select * from categories;
ERROR:  relation "categories" does not exist
LINE 1: select * from categories;
                      ^
```

Conclusion

Ces instructions SQL (potentiellement variables selon le SGBD choisi) permettent de définir le schéma d'une base de données (LDD) et de manipuler les données (LMD).
Ce tutoriel est volontairement limité pour en faciliter la prise en main. Des détails supplémentaires sont donnés dans les diapositives de cours (https://perso.liris.cnrs.fr/fabien.duchateau/BDW) ou sur d'autres ressources comme sql.sh ou le livre ~~Not~~ Only SQL.