# BASES DE DONNÉES
# COURS 1
# RÉVISIONS SQL + MODÉLISATION



Mickaël Coustaty

Jean-Loup Guillaume

**Laboratoire Informatique Image Interaction** (L3I)
Université de La Rochelle - Pôle Sciences et Technologie - Avenue Michel Crépeau - 17042 LA ROCHELLE CEDEX 1 France
Tél : +33 (0)5 46 45 82 62 – Fax : 05.46.45.82.42 – Site internet : http://l3i.univ-larochelle.fr/

# BASES DE DONNÉES EN L3

Deux EC au S5 et S6 (10CM et 10TP en tout)

Comprendre le fonctionnement interne des SGBD

- Indexation, optimisation, transactions
- Aller plus loin qu'un cours de SQL
- 10 séances = 10 sujets

Notation : contrôle continu

- Projet = 50% (en plusieurs notes)
- TP (tous ou certains) notés en partie = 40% (en plusieurs notes)
- Quelques QCM = 10%

# OBJECTIFS DU COURS 1

Avoir une bonne connaissance de la syntaxe SQL :

- Création et modification de tables
- Requêtes SQL simples
- Jointures internes et externes
- Agrégation

+ Notions sur :

- Clés primaires et étrangères
- Typage des données

Principes de bases de la modélisation et du modèle entités-associations

# PRÉREQUIS ET LIENS AVEC D'AUTRES COURS

Prérequis :

- Aucun (ce devrait être un cours de rappel pour la plupart)

Liens avec d'autres cours :

- Requêtes SQL un peu partout
- C2 normalisation = liens avec la modélisation et le modèle EA
- C3 triggers et autres = utilisation de SQL + autres langages

- C6 indexation = retour sur les types
- C7 optimisation = optimisation de requêtes SQL (et autres)

# BASES DE DONNÉES

Une base de données (BD) est un ensemble organisé de données en rapport avec un thème ou une activité

Un système de gestion de base de données (SGBD) est un logiciel qui permet d'interagir avec une BD via un certain nombre d'opérations :

- Définition de la structure des données
- Mise à jour des données
- Recherche de données
- Administration

Un SGBD doit être extrêmement optimisé pour un accès rapide aux données :

- Formats de fichiers spécifiques
- Utilisation d'index (cf cours ultérieur)
- Optimiseur de requêtes (cf cours ultérieur)
- Systèmes de cache…
- + réplication, distribution (cf cours ultérieur)

# BASES DE DONNÉES RELATIONNELLES

Dans une BD relationnelle :

- Les données sont stockées dans des tables (ou relations)
- Une table contient des lignes (ou enregistrements) qui correspondent à des « objets » et des colonnes qui décrivent les attributs de l'objet
- Les tables sont « connectées » entre elles par le biais de clés étrangères : un attribut d'un objet fait référence à une autre table

La cohérence de la BD est assurée en partie par les clés étrangères

- Ex : seul un produit existant (qui apparait dans une table produits) peut être commandé (apparaitre dans une table commandes)

Mais on peut ajouter d'autres contraintes sur les valeurs possibles (types et domaines), sur l'unicité de certains champs, etc.


# LE LANGAGE SQL

**Laboratoire Informatique Image Interaction** (L3I)
Université de La Rochelle - Pôle Sciences et Technologie - Avenue Michel Crépeau - 17042 LA ROCHELLE CEDEX 1 France
Tél : +33 (0)5 46 45 82 62 – Fax : 05.46.45.82.42 – Site internet : http://l3i.univ-larochelle.fr/

# SQL = STRUCTURED QUERY LANGUAGE

Le SQL est un langage utilisé pour accéder et gérer une BD. Il permet :

- De définir la structure des tables
- De modifier le contenu des tables
- De rechercher des enregistrements
- D'administrer le SGBD

Le SQL est normalisé

- Syntaxe quasiment similaire pour tous les SGBD
- Dans ce cours tout a été fait en PostgreSQL

# PLAN DU COURS

Langage de définition des données

- Création d'une base de données et des tables
- Typage (simple) de données

Langage de modification des données

- Insertion, suppression, modification

Langage d'interrogation des données

- Sélection
- Jointure
- Agrégation

# LDD - SYNTAXE GÉNÉRALE

Le LDD offre un ensemble de commandes pour manipuler les structures

Créer / supprimer une base de données :

- **CREATE DATABASE** <nom_de_la_base>;
- **DROP DATABASE** <nom_de_la_base>;

Créer / supprimer /vider une table:

- **CREATE TABLE** <nom_de_la_table> (…);
- **DROP TABLE** <nom_de_la_table>;
- **TRUNCATE** <nom_de_la_table>;

Modifier une table:

- **ALTER TABLE** <nom_de_la_table> <modification>;

# CRÉATION DE TABLES - SYNTAXE

CREATE TABLE <nomtable> (

<nom_attribut1> <type_att1> <valeur_par_defaut> <contraintes_att1>,

<nom_attribut1> <type_att2> <valeur_par_defaut> <contraintes_att2>,

…

<contraintes_de_table>

);

```
-- suppression de la table commandes si elle existe
DROP TABLE IF EXISTS commandes;

-- création de la table commandes avec 4 attributs
CREATE TABLE commandes (
  numero_commande           char(3)                     PRIMARY KEY,
  date_commande             date,
  montant                   numeric                     CHECK (montant > 0),
  remise                    numeric       DEFAULT 0.0   CHECK (remise >= 0),
  CHECK (remise < montant)
);
```

# TYPES DE DONNÉES CLASSIQUES

| Type | Description | Limite | |
|---|---|---|---|
| | | PostgresQL | MySQL |
| integer | Nombre entier | 4 octets | |
| numeric(precision, scale)<br>Synonyme de decimal | Nombre décimal avec calculs exacts<br>precision = nombre de chiffres significatifs<br>scale = nombre de chiffres après la virgule | 1000 | 65 |
| float / double | Nombre décimal avec calculs non exacts | 4 / 8 octets | |
| text | Chaîne de taille quelconque | 4 octets + compression | |
| char(n) = character(n) | Chaîne de taille n (ajout d'espaces si nécessaire) | 4 octets | 255 |
| varchar(n) = character varying(n) | Chaîne de taille au plus n | 4 octets | 65535 |
| blob | Chaîne binaires (pas d'encodage de caractères) | 4 octets | |
| date, time, timestamp | Date, heure ou les deux | | |
| enum | Une valeur parmi un ensemble défini | | 65535 valeurs |
| Nombreux autres types selon les SGBD : ensembles, formes géométriques, adresses IP ou MAC, tableaux, JSON, etc. | | | |

# EXPRESSION DE CONTRAINTES

Contrainte = condition que doivent respecter les enregistrements

- Insertion ou modification ne vérifiant pas la contrainte ⟶ erreur

Contraintes sur une colonne

- Existence : la valeur de l'attribut ne peut être nulle
- Unicité : deux enregistrements différents ne peuvent avoir la même valeur
- Clé primaire : existence + unicité = identifiant
- Clé étrangère : les données sont liées à celles d'une autre table
- Domaine : les valeurs sont limités à une liste, un intervalle, un format donné

Contraintes sur une table

- Idem mais portant sur plusieurs attributs

# CLÉS PRIMAIRES / ÉTRANGÈRES

Un clé primaire est un ensemble d'attributs qui identifient de manière unique chaque enregistrement. Formellement, une clé primaire est :

- Unique + non nulle
  - Contraintes vérifiées par le SGBD à chaque insertion/modification
  - Un index est créé pour les clés primaires
- Minimale : on peut pas enlever d'attribut sans perdre l'unicité
  - Contrainte de conception, pas vérifiable par un SGBD

Une clé étrangère permet de lier deux tables en indiquant qu'un (ou plusieurs) attribut d'une table fait référence à un (ou plusieurs) attribut d'une autre table :

- La cible doit être unique (c'est souvent la clé primaire de sa table)
- La source peut être NULL dans certains cas
  - Exemple : livres et auteurs, si on ne connait pas l'auteur d'un livre on va stocker NULL pour le numéro d'auteur dans la table livres

# CONTRAINTES D'EXISTENCE, D'UNICITÉ - SYNTAXE

Existence (NOT NULL)

Unicité (UNIQUE) : 3 manières de la déclarer

- Contrainte de colonne (sur un attribut)
- Contrainte de table (sur un ou plusieurs attributs)
- Modification de la table a posteriori

Clé primaire (PRIMARY KEY) : même syntaxe que unique

```
CREATE TABLE commandes (
  numero_membre char(10) NOT NULL
  numero_commande char(3) UNIQUE -- contrainte de colonne
);


CREATE TABLE commandes (
  numero_commande char(3),
  reference varchar(20),
  UNIQUE (numero_commande, reference) -- contrainte de table
);


ALTER TABLE commandes
  ADD UNIQUE (numero_commande);
```

# CONTRAINTES DE CLÉS ÉTRANGÈRES - SYNTAXE

Clé étrangère (FOREIGN KEY REFERENCES) : 3 types de déclaration

- Contrainte de colonne (sur un attribut)
- Contrainte de table (sur un ou plusieurs attributs)
- Modification de la table a posteriori

```
CREATE TABLE commandes (
  numero_produit  char(3) REFERENCES produits(ref_produit)
);

CREATE TABLE commandes (
  numero_produit char(3),
  FOREIGN KEY (numero_produit) REFERENCES produits(ref_produit)
);

ALTER TABLE commandes
  ADD FOREIGN KEY (numero_produit) REFERENCES produits(ref_produit);
```

# CONTRAINTES DE CLÉS ÉTRANGÈRES - ACTIONS

Automatisation si modification ou suppression de la référence

- CASCADE : répercussion dans la table qui référence
- RESTRICT : la référence ne peut pas être modifiée si elle est référencée
- SET DEFAULT, SET NULL : prend la valeur par défaut ou null
- Triggers : voir cours suivants

```
produits(numero_produit, designation, prix)
commandes(numero_commande, numero_produit, quantite)

-- si un produit est modifié/supprimé dans produits, le changement sera répercuté dans
commandes
numero_produit char(3)
        REFERENCES produits(ref_produit)
        ON UPDATE CASCADE ON DELETE CASCADE

-- si un produit est utilisé dans commande, on peut pas le supprimer dans produits
numero_produit char(3)
        REFERENCES produits(ref_produit)
        ON DELETE RESTRICT

-- si un produit est supprimé dans produits, la référence dans commandes est mise à NULL
numero_produit char(3)
        REFERENCES produits(ref_produit)
        ON DELETE SET NULL
```

# CONTRAINTES DE DOMAINE - SYNTAXE

Liste de valeurs possibles : CHECK et IN

Intervalle de valeurs possibles : CHECK et BETWEEN

Expression d'un format de données :

- CHECK + LIKE + _ (caractère quelconque) ou % (chaine quelconque)
- CHECK + ~ + expression régulière

```
CREATE TABLE produits (
  -- la tva vaut 5.5 ou 19.6
  tva float CHECK (tva IN (5.5, 19.6)),

  -- la quantité est comprise entre 1 et 100
  quantite integer CHECK (quantite BETWEEN 1 AND 100),

  -- utilisation d'une expression régulière simplifiée
  reference char(3) CHECK (reference LIKE 'R__'),

  -- utilisation d'une expression régulière
  intitule text CHECK (intitule ~ '^Description : %')
);
```

# CONTRAINTES - SYNTHÈSE

Clé primaire : spécifique à la table concernée

- Création plutôt en fin de table pour distinguer la/les clés des attributs

Clé étrangère : un attribut référencé doit toujours exister

- Création des tables dans le bon ordre (si c'est un graphe dirigé acyclique)
  - Plus simple : créer les clés (alter table) après avoir créé les tables
- Suppression des tables dans le bon ordre
  - Ou utiliser cascade (attention !) : DROP TABLE commande CASCADE
- Pas de modification d'attribut qui romprait les contraintes
  - Ou utiliser cascade : FOREIGN KEY (numero_produit) REFERENCES produits(numero_produit) ON DELETE CASCADE
- Désactivation de la vérification des clés possible dans certains SGBD

# MODIFICATION DE TABLE - SYNTAXE

ALTER TABLE <nom_de_la_table> <modification>;

Modifications de base :

- Ajout d'une colonne (ADD COLUMN)
- Modification d'une colonne (ALTER COLUMN)
- Suppression d'une colonne (DROP COLUMN)
- Ajout d'une contrainte (ADD CONSTRAINT)
- Suppression d'une contrainte (DROP CONSTRAINT)
- …

```
ALTER TABLE clients ADD COLUMN pseudoFB varchar;
ALTER TABLE clients ALTER COLUMN pseudoFB SET DATA TYPE text;
ALTER TABLE clients ALTER COLUMN pseudoFB SET NOT NULL;
ALTER TABLE clients ADD CONSTRAINT check_pseudoFB
                    CHECK (char_length(pseudoFB) < 50);
ALTER TABLE clients DROP CONSTRAINT check_pseudoFB;
ALTER TABLE clients DROP COLUMN pseudoFB;
```

# PLAN DU COURS

Langage de définition des données

- Création d'une base de données et des tables
- Typage (simple) de données

Langage de modification des données

- Insertion, suppression, modification

Langage d'interrogation des données

- Sélection
- Jointure
- Agrégation

# SYNTAXE GÉNÉRALE

Le langage de modification de données s'intéresse aux enregistrements et pas à la structure des tables

Les trois opérations sont :

- Insérer des données dans une table : INSERT
- Modifier des données dans une table: UPDATE
- Supprimer des données de la table : DELETE

# INSERTION D'ENREGISTREMENTS

Syntaxe complète :

- INSERT INTO <table> (<attribut_1>, <attribut_2>, ..., <attribut_n>
- VALUES (<valeur_1>, <valeur_2>, ..., <valeur_n>);

Syntaxe courte - insertion des attributs dans l'ordre :

- INSERT INTO <table>
- VALUES (<valeur_1>, <valeur_2>, ..., <valeur_n>);

```
CREATE TABLE produits (
  ref_produit char(4),
  designation varchar(20) NOT NULL,
  prix_unitaire numeric(9,2) DEFAULT 0.00,
  no_four char(3),
  PRIMARY KEY (ref_produit)
);

-- syntaxe longue
INSERT INTO produits (no_four, ref_produit, designation)
  VALUES (10, 'R01', 'trombone')

-- syntaxe courte
INSERT INTO produits
  VALUES ('R02', 'cahier', 0.50)
```

# MODIFICATION, SUPPRESSION D'ENREGISTREMENTS

Modification :

- UPDATE <table> SET <att_1>=<val_1>, <att_2>=<val_2>, …
- WHERE <conditions>;

Suppression :

- DELETE FROM <table> WHERE <conditions>;

```
CREATE TABLE produits (
  ref_produit char(4),
  designation varchar(20) NOT NULL,
  prix_unitaire numeric(9,2) DEFAULT 0.00,
  no_four char(3),
  PRIMARY KEY (ref_produit)
);

UPDATE produits
SET ref_produit='R001', designation = 'Trombone'
WHERE ref_produit='R01';

DELETE FROM produits
WHERE no_four=1 AND prix_unitaire<10;
```

# PLAN DU COURS

Langage de définition des données

- Création d'une base de données et des tables
- Typage (simple) de données

Langage de modification des données

- Insertion, suppression, modification

Langage d'interrogation des données

- Sélection
- Jointure
- Agrégation

# LANGAGE D'INTERROGATION DES DONNÉES

algèbre relationnelle = ensemble des opérations qui peuvent être effectuées sur des tables

5 fonctions permettent d'obtenir toutes les autres par composition

- Sélection
- Projection
- Produit cartésien
- Union
- Différence

# LANGAGE D'INTERROGATION DES DONNÉES

Ces fonctions se divisent en trois catégories, celles qui agissent

- Sur des attributs : projection, sélection
- Sur deux tables de même nature (même schéma) : union, différence
  - Même schéma = attributs de même type dans le même ordre
- Sur deux tables de nature (schéma) différentes : produit cartésien

Tout se fait en SQL avec **SELECT**

# 1. PROJECTION

La projection permet de ne conserver que certains attributs d'une table

- SELECT attribut_1, attribut_2, …, attribut_k
- FROM table

DISTINCT : supprimer les doublons

AS : pour renommer un attribut = alias

ORDER BY : trier selon un attribut (ASC : croissant, DESC : décroissant)

```
-- Ex 1 : la référence produit et la désignation de tous les produtis
SELECT ref_produit, designation
FROM produits;

-- Ex 2 : toutes les désignations (sans répétition)
SELECT DISTINCT(designation)
FROM produits;

-- Ex 3 : renommer un attribut
SELECT ref_produit AS ref, designation
FROM produits
ORDER BY designation ASC;
```

# 2. SÉLECTION

La sélection extrait les enregistrements (lignes) qui vérifient un critère

- Ce critère s'exprime comme les contraintes de domaine

```
-- Ex 1 : tous les produits de prix_unitaire >=155
SELECT *
FROM produits
WHERE prix_unitaire >=155;

-- Ex 2 : tous les produits de prix unitaire >=155 du fournisseur 1369
SELECT *
FROM produits
WHERE prix_unitaire >=155
  AND no_four = 1369;

-- Ex 3 : tous les produits dont le prix HT est entre 0 et 20
SELECT *
FROM produits
WHERE prixht BETWEEN 0 AND 20;
```

# COMPOSITION DE SÉLECTION ET PROJECTION

On peut bien sur mélanger sélection et projection

| ref_produit | designation | prix_unitaire | no_four |
|---|---|---|---|
| 139 | lampe camping gaz rhapsody | 30.00 | 1623 |
| 248 | chaussures de marche camargue | 75.00 | 1623 |
| 258 | chaussures de marche camargue | 87.00 | 1623 |
| 416 | sac a dos dolpo | 100.00 | 1369 |
| 426 | sac a dos nepal | 155.00 | 1369 |
| 765 | tente caravane | 300.00 | 1502 |

```
SELECT ref_produit, designation
FROM produits
WHERE prix_unitaire >=155
```

| ref_produit | designation |
|---|---|
| 426 | sac a dos népal |
| 765 | tente caravane |

# LES REQUÊTES IMBRIQUÉES

On peut utiliser le résultat d'une sous requête pour faire une autre requête

- Comparaison simple
- Comparaison à tous les enregistrements : ALL
- Comparaison à au moins un enregistrement : ANY

```
-- les produits dont le prix est >= à celui du produit 12
SELECT *
FROM produits
WHERE prix_unitaire >= (SELECT prix_unitaire FROM produits WHERE prod_id=12);

-- les produits dont le prix est >= à tous les prix
-- c'est-à-dire les produits les plus chers
SELECT *
FROM produits
WHERE prix_unitaire >= ALL (SELECT prix_unitaire FROM produits);

-- les produits dont le prix est < à au moins un autre produit
-- c'est-à-dire tous les produits sauf les plus chers
SELECT *
FROM produits
WHERE prix_unitaire < ANY (SELECT prix_unitaire FROM produits);
```

# 3. PRODUIT CARTÉSIEN

Le produit cartésien permet de calculer tous les croisements possibles entre deux tables

Exemple :

| ref_produit | designation | prix_unitaire | no_four |
|---|---|---|---|
| 139 | lampe camping gaz rhapsody | 30.00 | 1623 |
| 248 | chaussures de marche camargue | 75.00 | 1623 |
| 258 | chaussures de marche camargue | 87.00 | 1623 |
| 416 | sac a dos dolpo | 100.00 | 1369 |
| 426 | sac a dos nepal | 155.00 | 1369 |
| 765 | tente caravane | 300.00 | 1502 |

| no_four | raison_sociale | ville_four |
|---|---|---|
| 1369 | denecker sarl | Lyon |
| 1370 | chen'alpe diffussion | Lyon |
| 1502 | rodenas francois | Toulouse |
| 1623 | adidas | Dettwiller |

# 3. PRODUIT CARTÉSIEN

Résultat :

| fournisseur.no_four | raison_sociale | ville_four | ref_produit | designation | prix_unitaire | produit.no_four |
|---|---|---|---|---|---|---|
| 1369 | denecker sarl | Lyon | 139 | lampe camping gaz rhapsody | 30.00 | 1623 |
| 1369 | denecker sarl | Lyon | 248 | chaussures de marche camargue | 75.00 | 1623 |
| 1369 | denecker sarl | Lyon | 258 | chaussures de marche camargue | 87.00 | 1623 |
| 1369 | denecker sarl | Lyon | 416 | sac a dos dolpo | 100.00 | 1369 |
| 1369 | denecker sarl | Lyon | 426 | sac a dos nepal | 155.00 | 1369 |
| 1369 | denecker sarl | Lyon | 765 | tente caravane | 300.00 | 1502 |
| 1370 | chen'alpe diffussion | Lyon | 139 | lampe camping gaz rhapsody | 30.00 | 1623 |
| 1370 | chen'alpe diffussion | Lyon | 248 | chaussures de marche camargue | 75.00 | 1623 |
| 1370 | chen'alpe diffussion | Lyon | 258 | chaussures de marche camargue | 87.00 | 1623 |
| 1370 | chen'alpe diffussion | Lyon | 416 | sac a dos dolpo | 100.00 | 1369 |
| 1370 | chen'alpe diffussion | Lyon | 426 | sac a dos nepal | 155.00 | 1369 |
| 1370 | chen'alpe diffussion | Lyon | 765 | tente caravane | 300.00 | 1502 |
| 1502 | rodenas francois | Toulouse | 139 | lampe camping gaz rhapsody | 30.00 | 1623 |
| 1502 | rodenas francois | Toulouse | 248 | chaussures de marche camargue | 75.00 | 1623 |
| 1502 | rodenas francois | Toulouse | 258 | chaussures de marche camargue | 87.00 | 1623 |
| 1502 | rodenas francois | Toulouse | 416 | sac a dos dolpo | 100.00 | 1369 |
| 1502 | rodenas francois | Toulouse | 426 | sac a dos nepal | 155.00 | 1369 |
| 1502 | rodenas francois | Toulouse | 765 | tente caravane | 300.00 | 1502 |
| 1623 | adidas | Dettwiller | 139 | lampe camping gaz rhapsody | 30.00 | 1623 |
| 1623 | adidas | Dettwiller | 248 | chaussures de marche camargue | 75.00 | 1623 |
| 1623 | adidas | Dettwiller | 258 | chaussures de marche camargue | 87.00 | 1623 |
| 1623 | adidas | Dettwiller | 416 | sac a dos dolpo | 100.00 | 1369 |
| 1623 | adidas | Dettwiller | 426 | sac a dos nepal | 155.00 | 1369 |
| 1623 | adidas | Dettwiller | 765 | tente caravane | 300.00 | 1502 |

# 3. PRODUIT CARTÉSIEN

Ex : produits x fournisseurs

- SELECT *
- FROM produits , fournisseurs;

| fournisseur.no_four | raison_sociale | ville_four | ref_produit | designation | prix_unitaire | produit.no_four |
|---|---|---|---|---|---|---|
| 1370 | chen'alpe diffussion | Lyon | 139 | lampe camping gaz rhapsody | 30.00 | 1623 |
| 1370 | chen'alpe diffussion | Lyon | 248 | chaussures de marche camargue | 75.00 | 1623 |
| 1370 | chen'alpe diffussion | Lyon | 258 | chaussures de marche camargue | 87.00 | 1623 |
| 1502 | rodenas francois | Toulouse | 426 | sac a dos nepal | 155.00 | 1369 |
| 1502 | rodenas francois | Toulouse | 765 | tente caravane | 300.00 | 1502 |
| 1623 | adidas | Dettwiller | 139 | lampe camping gaz rhapsody | 30.00 | 1623 |
| 1623 | adidas | Dettwiller | 248 | chaussures de marche camargue | 75.00 | 1623 |
| 1623 | adidas | Dettwiller | 258 | chaussures de marche camargue | 87.00 | 1623 |
| … | | | | | | |

La plupart des tuples du produit cartésien n'ont pas de « sens »

- Si no_four est identique alors le fournisseur vend le produit

# 3'. JOINTURE

Composition de produit cartésien et de sélection = jointure

- Produit cartésien : PRODUITS x FOURNISSEURS
- Sélection : fournisseurs.no_four = produits.no_four

```
SELECT *
FROM produits p, fournisseurs f -- utilisation d'alias
WHERE p.no_four = f.no_four;
```

| Fournisseur.no_four | raison_sociale | ville_four | ref_produit | designation | prix_unitaire | produit.no_four |
|---|---|---|---|---|---|---|
| 1369 | denecker sarl | Lyon | 416 | sac a dos dolpo | 100.00 | 1369 |
| 1369 | denecker sarl | Lyon | 426 | sac a dos nepal | 155.00 | 1369 |
| 1502 | rodenas francois | Toulouse | 765 | tente caravane | 300.00 | 1502 |
| 1623 | adidas | Dettwiller | 139 | lampe camping gaz rhapsody | 30.00 | 1623 |
| 1623 | adidas | Dettwiller | 248 | chaussures de marche camargue | 75.00 | 1623 |
| 1623 | adidas | Dettwiller | 258 | chaussures de marche camargue | 87.00 | 1623 |

# 3'. JOINTURE

Pour réaliser une jointure, il faut que les attributs soient du même type

- Pas nécessaire qu'il y ait une contrainte de type « clé étrangère » même si en pratique on fait presque toujours la jointure entre une clé étrangère et une clé primaire
- si la clause WHERE est incorrecte on risque d'obtenir un produit cartésien

La jointure est fondamentale dans les requêtes. Elle permet de

- Lier de manière cohérente les tables entre elles
- Récupérer dans le résultat (i.e. projeter) des attributs de tables différentes
- Récupérer dans le résultat (i.e. sélectionner) des enregistrements selon les attributs de tables différentes

En général si on fait une jointure entre n tables il y a n-1 sélections

# 3'. JOINTURE – AUTRES SYNTAXES

Syntaxe alternative : INNER JOIN + ON (inner optionnel)

```
SELECT *
FROM produits
INNER JOIN fournisseurs
  ON produits.no_four = fournisseurs.no_four;
```

Cette syntaxe permet de

- Mieux séparer la sélection de la jointure d'éventuelles autres sélections
- Ne pas oublier l'opérateur de jointure (le ON est obligatoire)
- Est obligatoire pour les jointures externes (cf dans 2 slides)

Conseil : n'utiliser que cette syntaxe…

# 3'. (SEMI-)JOINTURE – AUTRES SYNTAXES

Enregistrements pour lesquels le fournisseur est dans une liste spécifiée

```
SELECT *
FROM produits
WHERE no_four IN (
  SELECT no_four
  FROM fournisseurs)
```

Enregistrements pour lesquels il existe une correspondance

```
SELECT *
FROM produits p
WHERE EXISTS (
  SELECT no_four
  FROM fournisseurs
  WHERE no_four = p.no_four)
```

IN et EXISTS ne se limitent pas à faire des (semi-)jointures et peuvent être avec n'importe quelle sous-requête

# 3'. JOINTURES EXTERNES

INNER JOIN ne donne que les éléments qui correspondent. On peut aussi récupérer ceux qui ne correspondent pas avec

- LEFT JOIN : join + produits sans fournisseur
- RIGHT JOIN : join + fournisseurs sans produit
- FULL JOIN : join + produits sans fournisseur et fournisseurs sans produit

Dans chaque cas les champs absents sont mis à NULL

Attention : la syntaxe varie selon les SGBD

```
SELECT *
FROM produits
LEFT JOIN fournisseurs
  ON produits.no_four = fournisseurs.no_four;
```

# 4. UNION

Permet de récupérer l'union de deux requêtes distinctes

- Ex : tous les produits dont le prix est supérieur à 155€ OU qui sont fournis par Adidas

Contrainte : les deux relations sur lesquelles on appliquera l'UNION doivent être de même schéma :

- même nombre d'attributs,
- dans le même ordre,
- de mêmes types (domaine de valeurs)
- ... mais pas forcément de même nom (ce sont les noms des attributs de la table qui apparaît en premier dans la requête qui seront utilisés)

# 4. UNION

```
SELECT designation, prix_unitaire
FROM produits
WHERE prix_unitaire >= 155

UNION

SELECT  designation, prix_unitaire
FROM  produits, fournisseurs
WHERE produits.no_four = fournisseurs.no_four
     AND raison_sociale = 'Adidas';


-- version équivalente
SELECT designation, prix unitaire
FROM produits
JOIN fournisseurs ON produits.no_four = fournisseurs.no_four
WHERE prix_unitaire >= 155 OR raison_sociale = 'Adidas';
```

# 5. DIFFÉRENCE

Permet de récupérer les résultats d'une requête qui ne sont pas dans une second requête. Plusieurs méthodes :

- NOT IN
- EXCEPT (le nom peut varier d'un SGBD à un autre)
- INTERSECT (idem)

```
-- tous les produits qui ne sont pas dans la table ligne commande
SELECT ref_produit
FROM produits
WHERE ref_produit NOT IN (
  SELECT reference
  FROM lignes_cde) ;

-- tous les produits à l'exception de ceux dans la table ligne commande
SELECT ref_produit
FROM produits
EXCEPT
SELECT reference
FROM lignes_cde;
```

# PLAN DU COURS

Langage de définition des données

- Création d'une base de données et des tables
- Typage (simple) de données

Langage de modification des données

- Insertion, suppression, modification

Langage d'interrogation des données

- Sélection
- Jointure
- Agrégation = comment faire des calculs simples

# FONCTIONS D'AGRÉGATION

COUNT :

- Compte le nombre de tuples (count(*))
- Compte le nombre de valeurs non nulles (count(attribut)) pour un attribut

```
-- nombre de produits dans la base
SELECT COUNT(*)
FROM produits;

-- nombre de produits avec un fournisseurs
SELECT COUNT(num_fournisseur)
FROM produits;

-- nombre de fournisseurs différents pour des produits
-- le distinct s'applique avant le COUNT
SELECT COUNT(DISTINCT num_fournisseur)
FROM produits;
```

# FONCTIONS D'AGRÉGATION

MAX, MIN : Valeur maximale ou minimale

AVG, SUM : Moyenne ou somme de toutes les valeurs

```
-- prix maximum d'un produit
SELECT MAX (prix_unitaire)
FROM produits;

-- prix moyen des produits
SELECT AVG (prix_unitaire)
FROM produits;

-- somme du prix de tous les produits dans la base
SELECT SUM (prix_unitaire)
FROM produits;
```

# CONSTITUTION DE GROUPES

```
SELECT date_commande, SUM(quantite)
FROM commandes C
JOIN lignes_cde LC ON C.no_commande = LC.no_commande;
```

Que fait cette requête ?

Aucune idée, on ne sait pas ce qu'on veut :

1. Le nombre total d'articles commandés dans la base, avec la liste des dates de commandes ?
2. Le nombre d'articles commandés par jour ?
3. Le nombre d'articles par commande, avec le jour de la commande ?

# CONSTITUTION DE GROUPES (GROUP BY)

```
SELECT date_commande, SUM(quantite)
FROM commandes C
JOIN lignes_cde LC ON C.no_commande = LC.no_commande;
```

1. Le nombre total d'articles commandés dans la base (1 valeur), avec la liste des dates de commandes (plusieurs enregistrements)
   - Impossible avec une seule requête : le résultat d'une requête est une table
   - Donc, si on ne complète pas la requête, le SGBD renvoie une erreur !

Dès que l'on veut un attribut en plus de l'agrégat il faut regrouper.

# CONSTITUTION DE GROUPES (GROUP BY)

```
SELECT date_commande, SUM(quantite)
FROM commandes C
JOIN lignes_cde LC ON C.no_commande = LC.no_commande;
```

2. Le nombre d'articles commandés chaque jour ?

```
SELECT date_commande, SUM(quantite)
FROM commandes C
JOIN lignes_cde LC ON C.no_commande = LC.no_commande
GROUP BY date_commande;
```

GROUP BY :

- créé des groupes suivant un ou plusieurs attributs (ici date_commande)
- puis calcule une seule ligne de résultat pour chaque groupe

# CONSTITUTION DE GROUPES (GROUP BY)

```
SELECT date_commande, SUM(quantite)
FROM commandes C
JOIN lignes_cde LC ON C.no_commande=LC.no_commande;
```

3. Le nombre d'articles par commande, avec le jour de la commande (on veut donc regrouper par no_commande) ?

```
SELECT date_commande, SUM(quantite)
FROM commandes C
JOIN lignes_cde LC ON C.no_commande=LC.no_commande
GROUP BY no_commande, date_commande;
```

Le regroupement par date_commande est inutile car no_commande est clé primaire (donc date_commande dépend de no_commande)

- Mais tous les attributs du SELECT doivent apparaitre dans le GROUP BY (sur certains SGBD ça fonctionne sans)
- On voudrait sans doute ajouter no_commande dans le SELECT pour distinguer des commandes qui auraient lieu le même jour sinon on va avoir plusieurs plusieurs lignes avec la même date.

# GROUP BY - SYNTHÈSE

Forme générale :

```
SELECT [DISTINCT] att1, att2, ... att_n, aggregat
FROM tables
WHERE conditions
GROUP BY att1, att2, ... att_n, ...;
```

Fonctionnement :

- Tous les enregistrements sont extraits des tables
- Les attributs inutiles et les enregistrements qui ne vérifient pas les conditions sont rejetés
- Les enregistrements restants sont regroupés et un enregistrement est généré par groupe
- SI DISTINCT alors les enregistrements identiques sont supprimés

# SÉLECTION DE GROUPES - HAVING

Que veut-on faire avec :

```
SELECT prod_id, prod_nom, COUNT(no_four)
FROM produits p
JOIN fournisseurs f ON p.no_four = f.no_four
WHERE count(no_four) >1
GROUP BY prod_id, prod_nom;
```

Récupérer tous les produits ayant plus d'un fournisseur ?

- `ERROR: aggregates not allowed in WHERE clause`

# SÉLECTION DE GROUPES - HAVING

La sélection d'enregistrements s'exprime avec WHERE

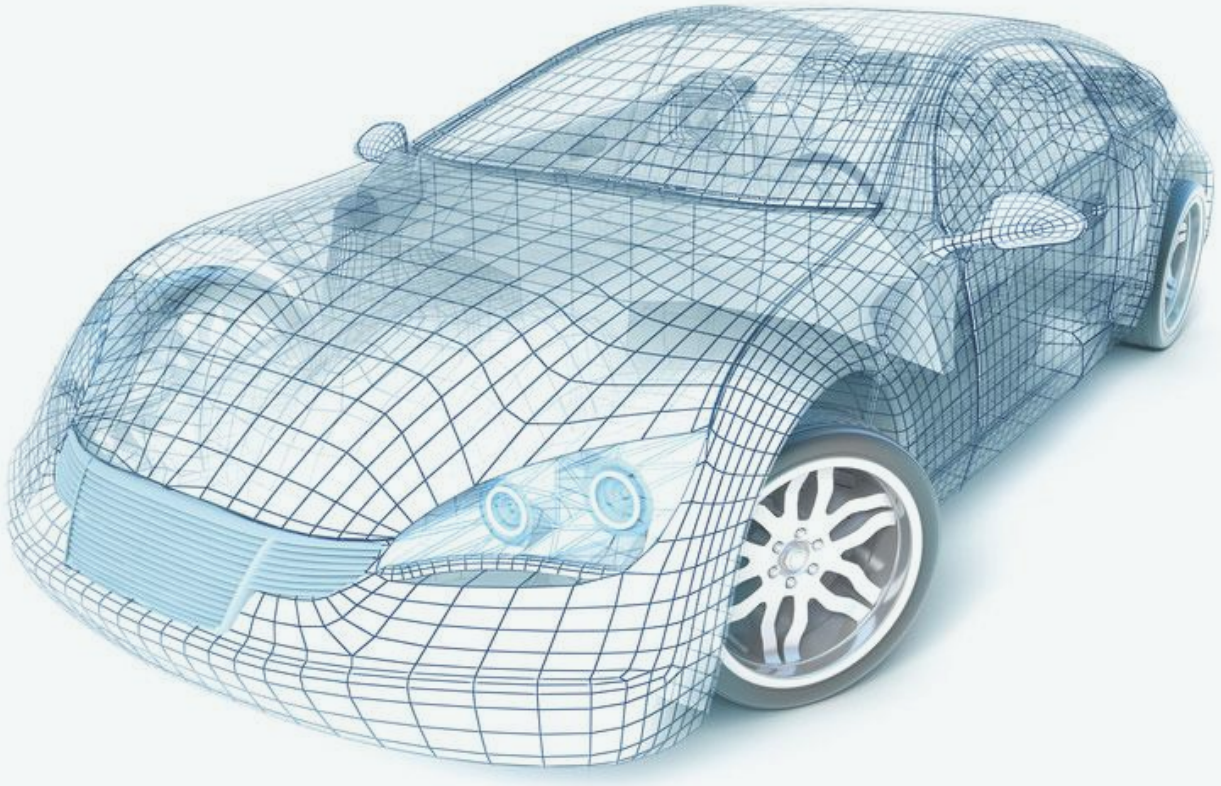
La sélection de groupes s'exprime avec HAVING

- N'a de sens que s'il y a un GROUP BY avant

```
SELECT prod_id, prod_nom, COUNT(no_four)
FROM produits p
JOIN fournisseurs f ON p.no_four = f.no_four
GROUP BY prod_id, prod_nom
HAVING count(no_four) >1;
```

# SELECT – SYNTHÈSE

```
SELECT [DISTINCT] column1[,column2,agregats]
FROM table1
[JOIN table2 ON table1.x = table2.x]*
[WHERE "conditions"]
[GROUP BY "column-list"
  [HAVING "conditions"]]
[ORDER BY "column-list" [ASC | DESC] ]
```



# MODÉLISATION

# MODÈLE ENTITÉ-ASSOCIATION

**Laboratoire Informatique Image Interaction** (L3I)
Université de La Rochelle - Pôle Sciences et Technologie - Avenue Michel Crépeau - 17042 LA ROCHELLE CEDEX 1 France
Tél : +33 (0)5 46 45 82 62 – Fax : 05.46.45.82.42 – Site internet : http://l3i.univ-larochelle.fr/

# POURQUOI MODÉLISER ?

La modélisation est une représentation simplifiée de la réalité :

- Trop forte complexité du monde réel
- Abstraction des aspects cruciaux du problème
- Omission des détails

Il faut permettre une conception progressive

- Abstractions et raffinements successifs
- Conception pour plusieurs utilisateurs
- Génération (automatique) des structures de données (et de traitements)

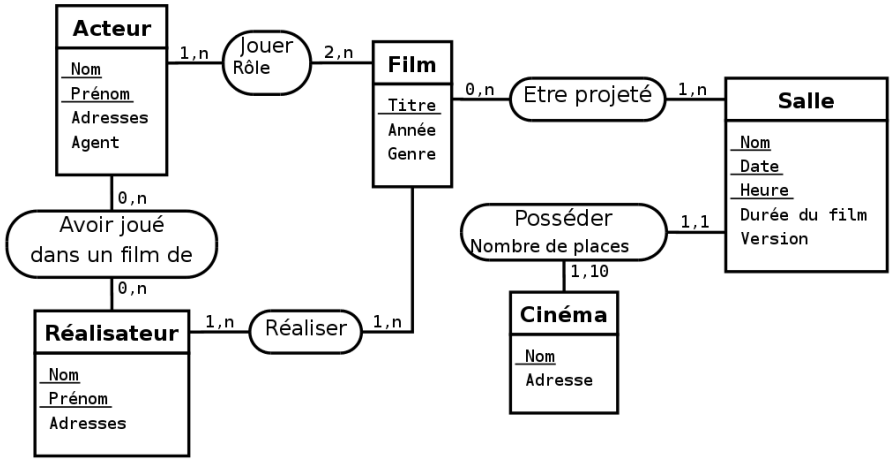
# POURQUOI MODÉLISER / BASES DE DONNÉES

Isoler les concepts fondamentaux

- Que vont représenter les données de la BD ?
- Quels sont les concepts de base du monde réel que l'on veut intégrer ?
- Comment décrire ces concepts ?

Faciliter la visualisation du système

- Diagrammes avec notations simples : meilleure compréhension visuelle



# REPRÉSENTATION DES DONNÉES

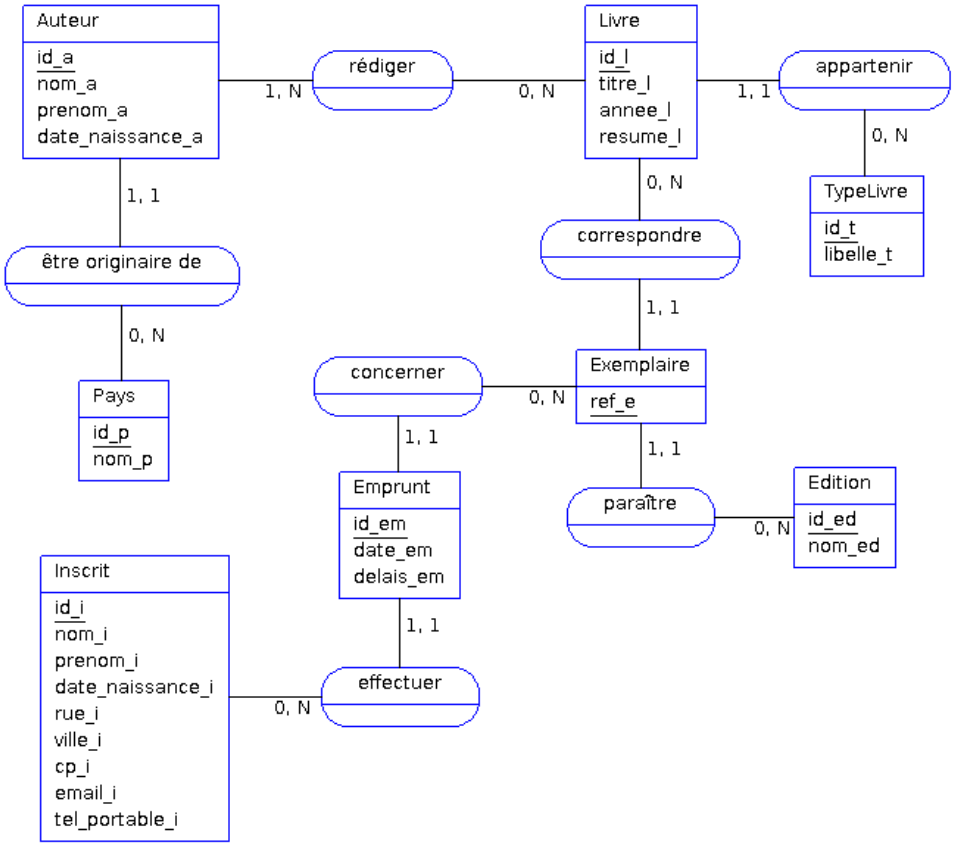
Trois niveaux de conception (schémas ou modèles)

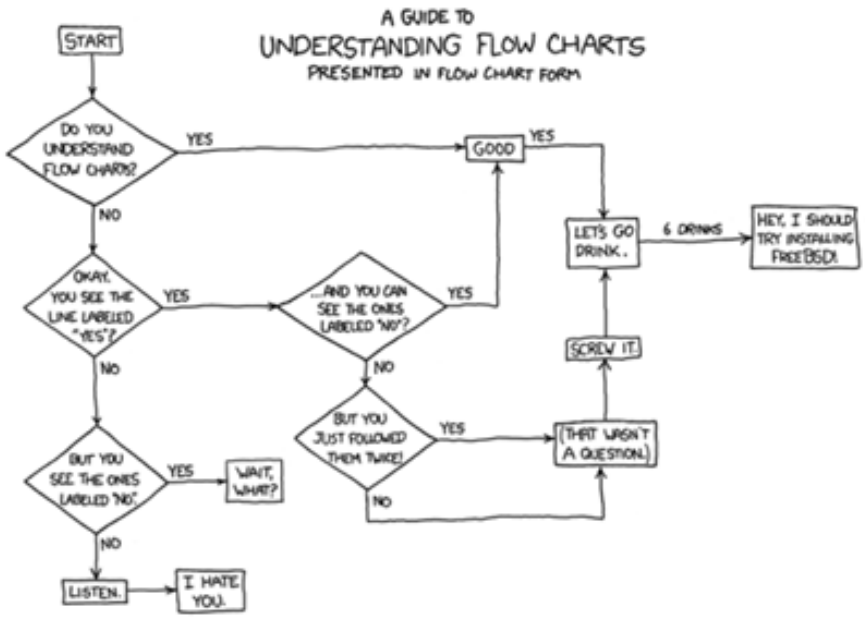
- Schéma externe
  - Personnalisé en fonction de la vision de l'utilisateur
- Schéma conceptuel
  - Regroupement des schémas externes
  - Utilisation du modèle Entité / Association
- Schéma interne
  - Méthodes de stockage des informations

Il faut distinguer

- La description des données
- La description des traitements

# DONNÉES VS. TRAITEMENTS



# DANS CE COURS

Modèle conceptuel des données (MCD)

- Schéma représentant la structure du SI vu sous l'angle des données
- Dépendances ou relations entre les différentes données
  - ex : le client, la commande, les produits, etc.

A distinguer du Modèle conceptuel des traitements (ou MCT)

- Schéma représentant les traitements, en réponse aux évènements à traiter
  - ex : prise en compte d'une commande

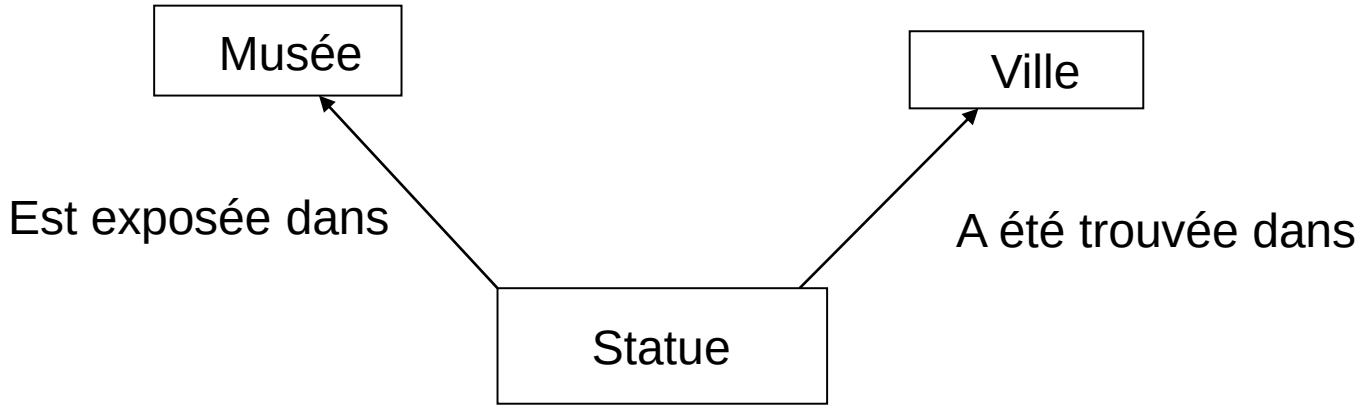
En général MCD et MCT sont censés être stables

- Sauf si changement des activités de l'entreprise
- Indépendant du logiciel utilisé pour traiter les données

# SCHÉMA CONCEPTUEL

Exemples de descriptions en français

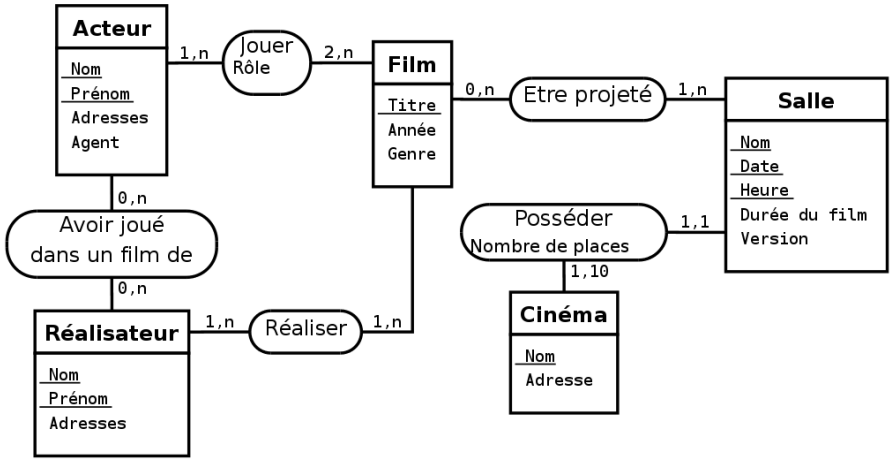
- Un musée a un nom, un numéro, et se trouve dans une ville
- Une statue a un numéro, une désignation, un type
- Une statue a été trouvée dans une ville
- Une statue n'est pas obligatoirement exposée dans un musée



# MODÈLE CONCEPTUEL DES DONNÉES

Modèle entité / association

- Formalisé en 1976 par Peter Chen
- Puis nombreuses modifications et extensions
- Modèle basé sur des concepts et des symboles graphiques associés
  - Donne une représentation de la structuration des données
  - Indépendant de l'implémentation effective du stockage



# EXEMPLE

Un service de ressources humaines dans une entreprise veut gérer le personnel. On veut pouvoir connaître le nom, la fonction, la date d'entrée, le salaire, la commission (part variable) de chaque employé et le numéro du département dans lequel travaille chaque employé.

Le service du personnel souhaite aussi connaître le nom du département dans lequel l'employé travaille. Comme l'entreprise est répartie dans plusieurs villes, les départements sont caractérisés par leur nom et par leur ville. Chaque employé travaille dans un seul département.

# EXEMPLE

Un service de ressources humaines dans une entreprise veut gérer le personnel. On veut pouvoir connaître le nom, la fonction, la date d'entrée, le salaire, la commission (part variable) de chaque employé et le numéro du département dans lequel travaille chaque employé.

Le service du personnel souhaite aussi connaître le nom du département dans lequel l'employé travaille. Comme l'entreprise est répartie dans plusieurs villes, les départements sont caractérisés par leur nom et par leur ville. Chaque employé travaille dans un seul département.

Modélisation :

- Un employé = nom, fonction, date, salaire, commission
- Un département = numéro, nom, ville
- Plusieurs employés par département, un département par employé

# EXEMPLE

Définition de deux (type-)entités

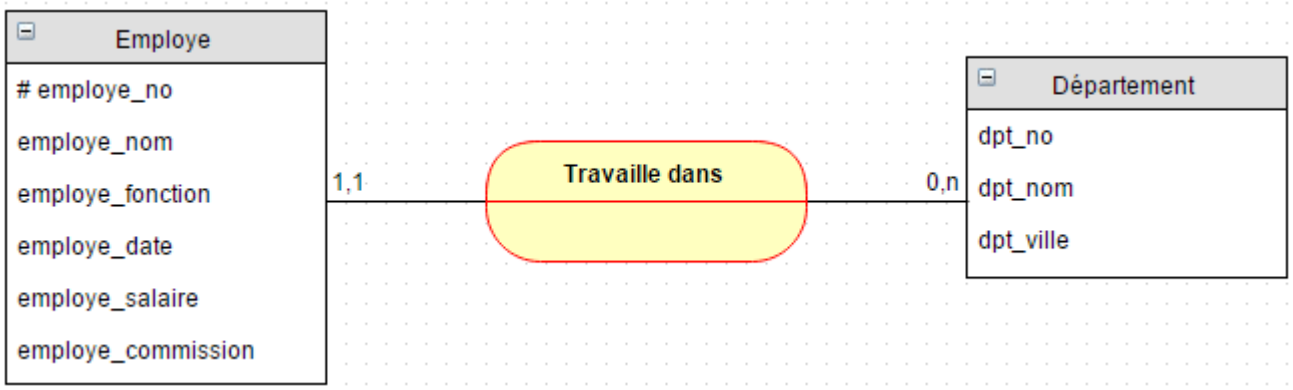
- Employé
- Département

reliées par une (type-)association

- Un employé travaille dans un département

avec des cardinalités

- Un employé est dans un seul département
- Un département peut accueillir plusieurs employés



# QUELQUES DÉFINITIONS - ENTITÉ

Entité :

- Objet, concret ou abstrait, mais qui peut-être identifié de manière unique
  - ex : ma télé, ma voiture, moi, etc.
- Une entité correspondra en général à un enregistrement dans une table

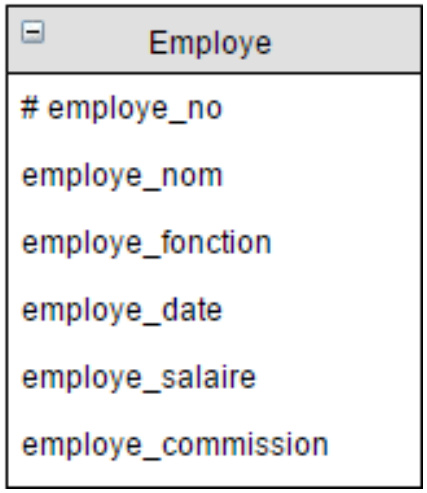
Type-entité (ou classe-entité, ou entité) :

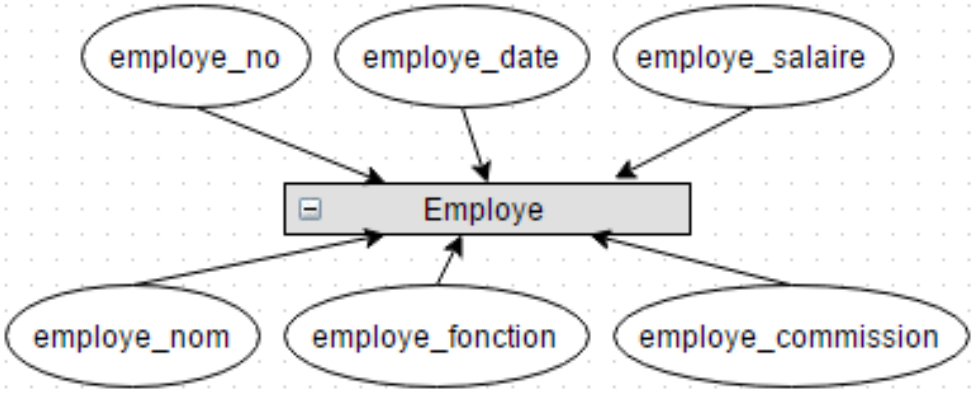
- Ensemble d'entités qui possèdent une sémantique commune
  - ex : la notion de télé, de voiture, de personne, etc.
- Un type-entité est décrit par un ensemble de d'attributs ou propriétés
- Un type-entité correspondra en général à une table

# QUELQUES DÉFINITIONS - ATTRIBUT

## Attribut :

- Caractéristique associée à un type-entité
  - ex : le numéro de sécurité sociale d'une personne, son âge, etc.
- Chaque attribut possède un domaine de valeurs possibles
  - ex : nombre, date, chaîne de caractères, etc.

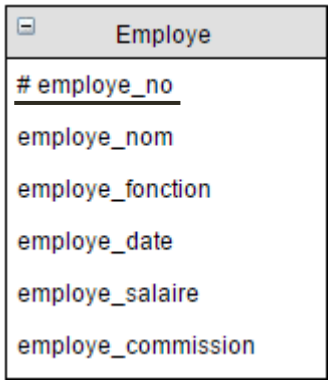


# QUELQUES DÉFINITIONS - IDENTIFIANT

## Identifiant ou clé :

- Un identifiant d'un type-entité est un ensemble d'attributs qui doivent avoir une valeur unique
- Cela correspond à la notion de clé (primaire ou candidate)
  - Si c'est la clé primaire on souligne le(s) attribut(s)



# PASSAGE EN SQL

Chaque type-entité devient une table

Les attributs des type-entités deviennent des attributs des tables

Les types des attributs donnent des types en SQL

- Et éventuellement des contraintes de domaine

Les identifiants deviennent des clés :

- Un identifiant est choisi pour devenir la clé primaire
- Les autres clés sont candidates et sont donc des contraintes d'unicité

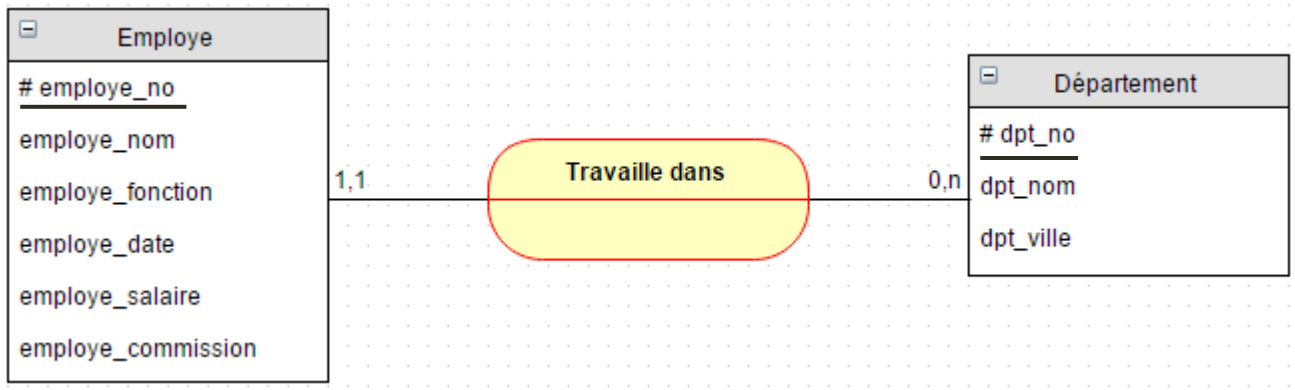
# QUELQUES DÉFINITIONS - ASSOCIATION

Association :

- Lien entre plusieurs entités (qui ont un rôle dans l'association)
  - ex : la personne x dans le département y

Type-association (ou type-relation, ou association, ou relation) :

- Ensemble d'associations de mêmes caractéristiques
  - ex : implication d'une personne dans un département
- L'identifiant du type-association est l'union des identifiants des type-entités

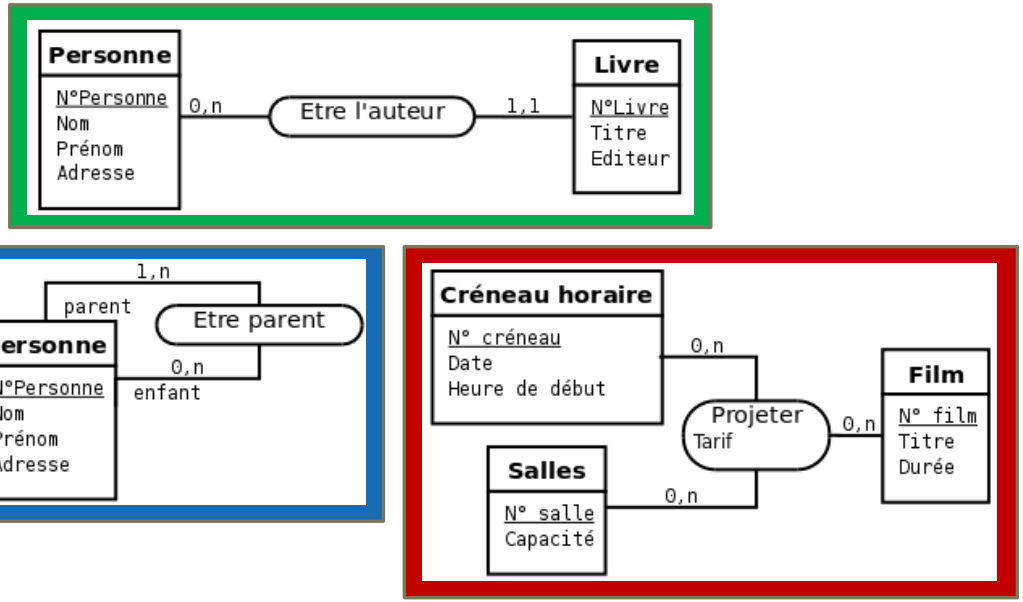


# ASSOCIATION

Un type-association peut :

- Associer un type-entité à lui-même (association réflexive)
- Associer deux type-entités entre eux (association binaire)
- Associer 3 ou plus type-entités (association ternaire ou n-aire)

On parle alors de la dimension du type-association

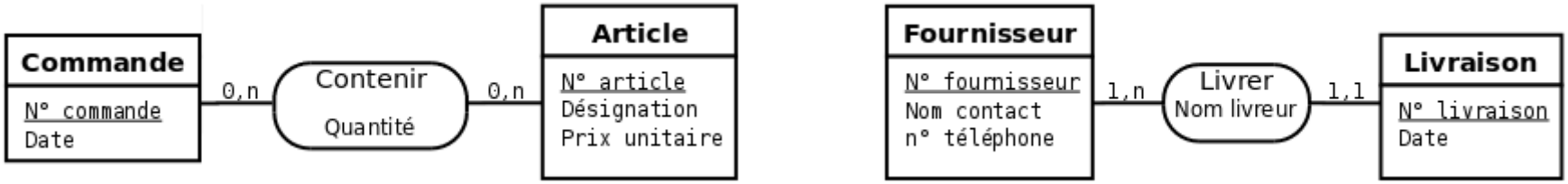


# ASSOCIATION ET ATTRIBUTS

Un type-association peut avoir des attributs supplémentaires :

- Informations liées à l'ensemble des types-entités
- N'ont de sens qu'en lien avec tous les types-entités qui sont associés

Que penser des attributs dans les deux types-associations ci-dessous ?




- Contenir : ok, la quantité n'a de sens que pour un article et une commande
- Livrer : le nom du livreur ne dépend que de la livraison donc il devrait être dans la livraison

De manière générale, pas d'attribut si type-association avec cardinalité 1,1

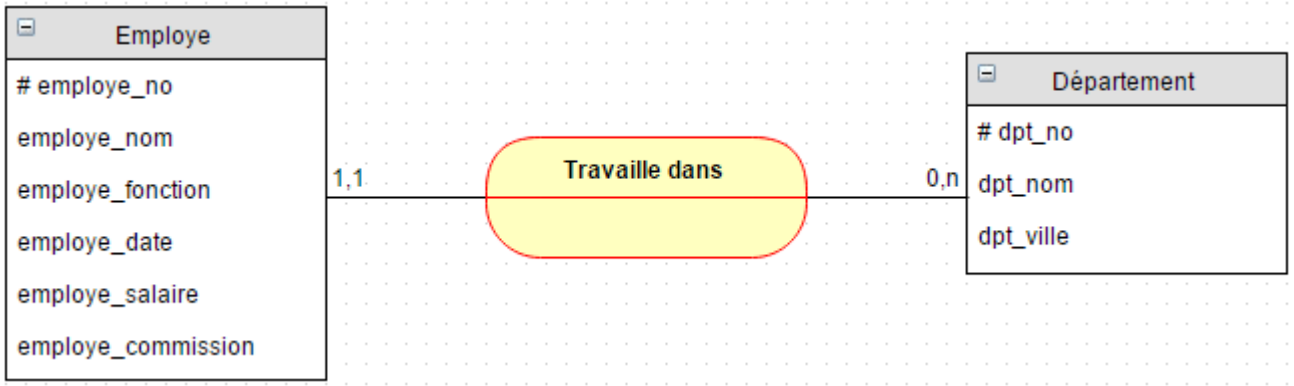
# PASSAGE EN SQL

Les type-associations deviennent souvent des tables

La clé de la table d'association est l'union des clés des type-entités qui la forment

- ex : travaille (employe_no#, dpt_no#)

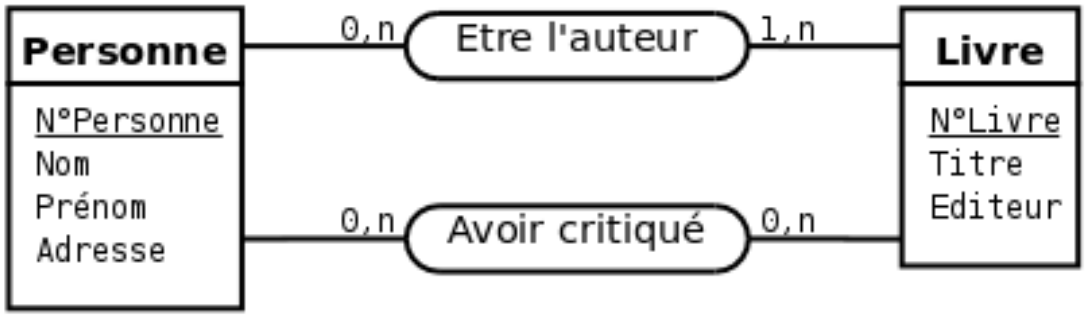
Les attributs supplémentaires (s'il y en a) seront dans la table d'association



# ASSOCIATION PLURIELLE

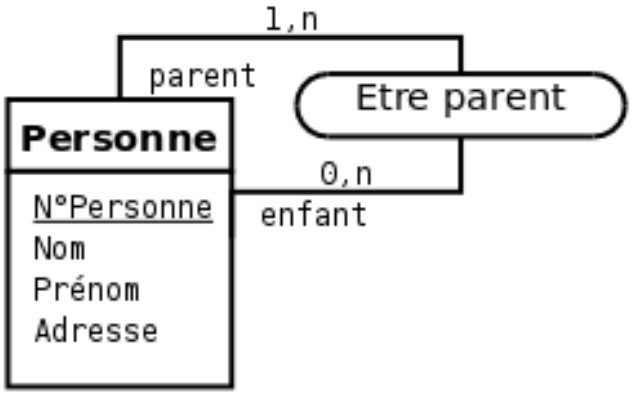
Deux mêmes type-entités peuvent être associés de différentes manières

- Pas de limite particulière au nombre d'associations



# ASSOCIATION RÉFLEXIVE

Un type-entité peut être en association avec lui-même

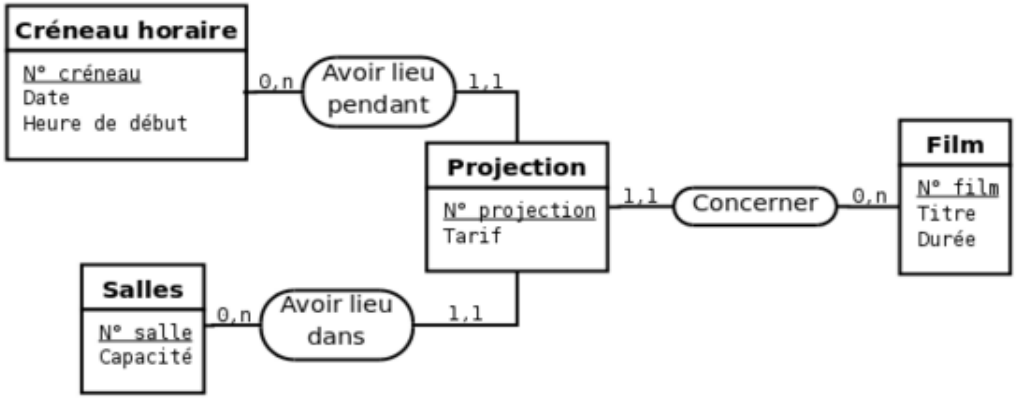


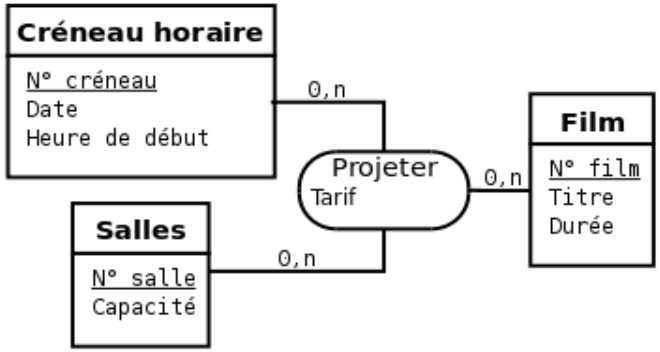

Autres exemples :

- Un produit peut être composé d'autres produits
- Un employé a pour chef un autre employé

# ASSOCIATION N-AIRE

Laquelle des deux modélisations ci-dessous est la plus pertinente ?



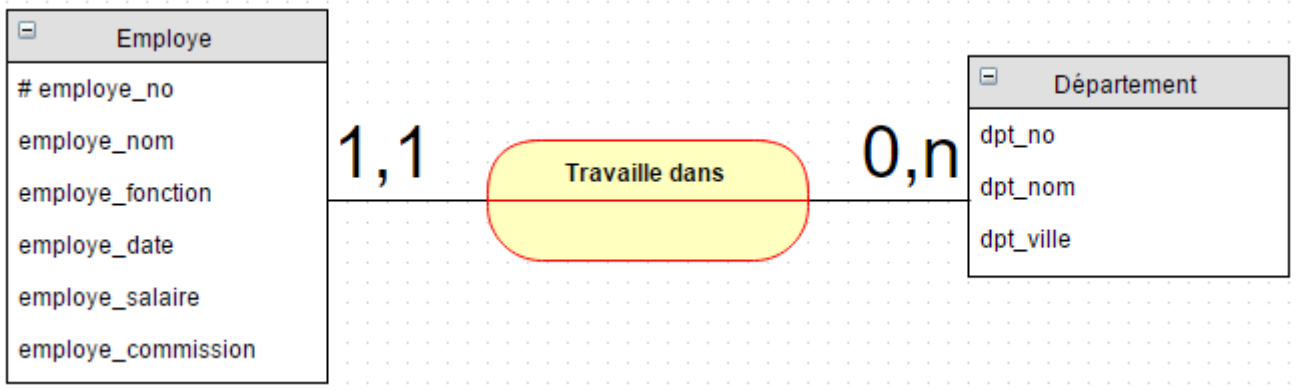




- La première est plus simple car il n'y a que des types-associations binaires
- La seconde a un seul type-association et n'a pas d'identifiant artificiel
  - n°projection = n°creneau + n°salle + n°film

# QUELQUES DÉFINITIONS - CARDINALITÉ

Cardinalité :

- Indique le nombre de fois qu'une entité peut participer à une association
- Pour chaque type-entité on a un minimum et maximum de participation
  - Un employé appartient à au moins 1 et au plus 1 département : 1,1
  - Un département contient au moins 0 employé et n'a pas de limite maximale sur le nombre d'employé : 0,n



# CARDINALITÉ

Cardinalité minimales possibles :

- 0 : l'entité peut exister même si elle n'est pas impliquée dans l'association
  - ex : on peut avoir des clients qui n'ont jamais fait de commande
- 1 : toute entité doit être impliquée dans (au moins) 1 association
  - ex : une facture est forcément associée à un client
- x (fixé) : toute entité doit être impliquée dans (au moins) x associations
  - ex : chaque étudiant doit suivre au moins 5 cours par semestre

Cardinalité maximales possibles :

- 0 : n'a pas de sens car l'association serait inutile
- 1 : toute entité est dans au plus une association
  - ex : une voiture a un seul propriétaire
- y (fixé) : toute entité appartient à au plus y associations
  - ex : chaque bachelier peut choisir au plus 2 options facultatives
- n : pas de limite supérieure
  - ex : on peut posséder autant de voitures que l'on veut

# CARDINALITÉ

Cardinalités utilisées en pratique :

- 0,1 : chaque entité est dans au plus une association
  - ex : on ne peut se marier qu'une fois au plus
- 0,n : chaque entité peut être associé autant que l'on veut
  - ex : on peut posséder autant de voitures que l'on veut, y compris 0
- 1,1 : chaque entité est dans une unique association obligatoire
  - ex : chaque salarié est affecté à un unique département
- 1,n : au moins une association par entité
  - ex : chaque facture contient au moins un produit

Autres associations

- 2,7 : au moins 2, au plus 7 est possible en théorie
- Il est fort probable que les valeurs changent un jour, il est donc préférable de mettre 0,n ou 1,n selon

# CARDINALITÉ

Comment choisir les cardinalités ?

- Cardinalité min : est-ce que l'entité existe même sans association ?
- Cardinalité max : est-ce que l'entité peut-être associée plusieurs fois ?

Exemple : Produit associé à une commande

- Doit-il être stocké si n'est jamais commandé ?
- Peut-il être commandé plusieurs fois ?
- Dépend du contexte :
  - Site de vente classique : oui et oui → 0,n
  - Fabrication de produits uniques sur commande : non et non → 1,1

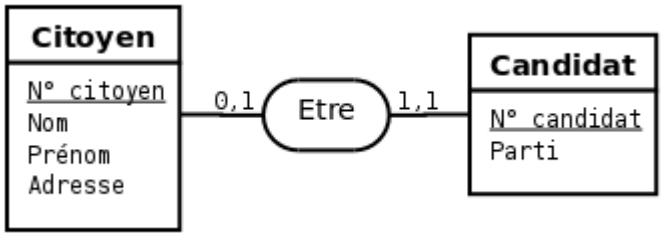
# CARDINALITÉ

Implémentation en SQL si les deux côtés sont 0,1 ou 1,1

- Si les deux côtés de l'association sont 1,1 alors les deux tables devraient être fusionnées

- Si on a 0,1 et 1,1 alors il vaut mieux (faut) mettre une clé étrangère du côté 1,1 car l'association est toujours existante
  - ex : table « candidat » contient une clé étrangère vers « citoyen »

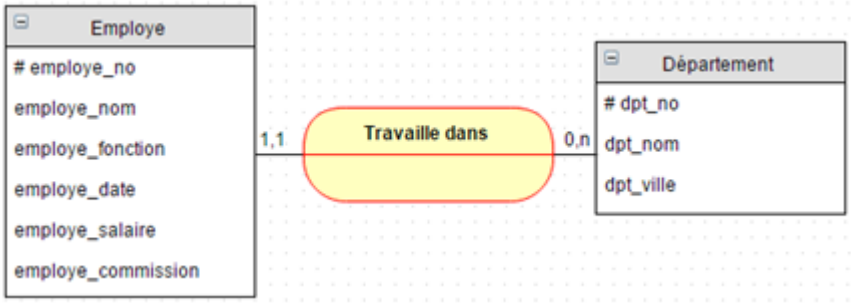
- Si on a 0,1 des deux côtés, on ajoute la clé où on veut

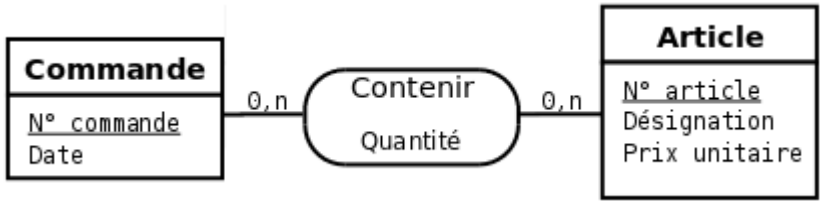


# CARDINALITÉ

Implémentation en SQL si un côté est 0,N ou 1,N

- Si l'autre cardinalité est 1,1 : clé étrangère du côté 1,1
  - ex : table « employe » contient une clé étrangère dpt_no
- Si l'autre cardinalité est 0,1 : clé étrangère du côté 0,1 avec NULL
- Si l'autre cardinalité est 0,N ou 1,N : table supplémentaire
  - La clé primaire de cette table est l'union des clés primaires des tables en association, les attributs supplémentaires sont ajoutés
  - ex : il faut une table « contenir »

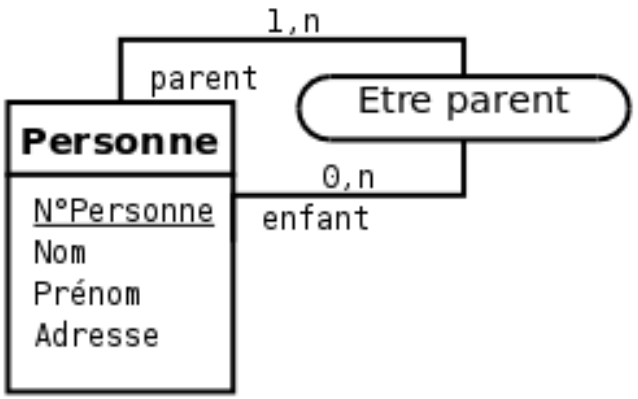


# CARDINALITÉ

Implémentation en SQL : association réflexive

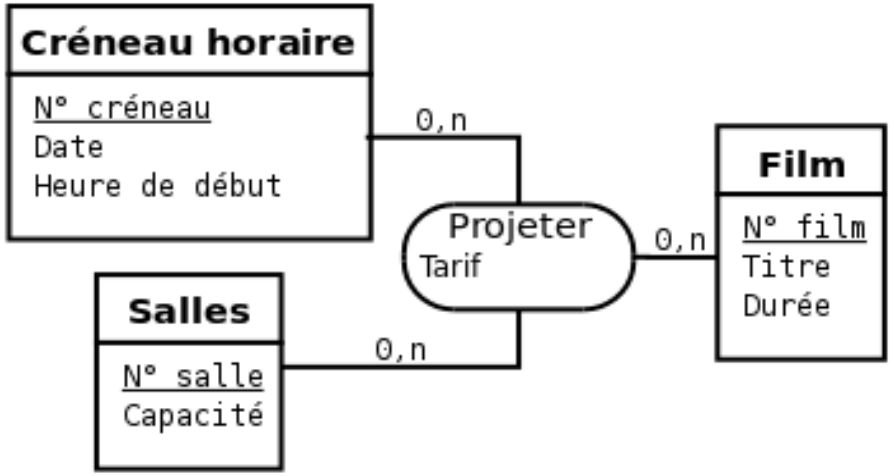
- Faire comme s'il y avait deux type-entités, appliquer les règles, puis « fusionner »



# CARDINALITÉ

Implémentation en SQL : type-association n-aire

- On créé toujours une table supplémentaire
- La clé primaire de la nouvelle table est l'union des clés
- Les attributs supplémentaires sont ajoutés dans la table
  - ex : trois tables + table projeter (n°creneau, n°salle, n°film, tarif)



# SUPPRESSION DE TABLES

Si on a une table qui ne contient qu'une clé et qui est en association 1,1 ou 1,N

```
Matiere (nom_matiere)
EstEnseignée (nom_matiere, nom_enseignant)
Enseignant (nom_enseignant, statut)
```

- La table Matière peut être supprimée car elle n'apporte rien de plus que la table EstEnseignée
- Attention, si l'association est 0,N alors il faut garder la table Matiere car alors certaines matières peuvent exister sans être enseignées

# RÈGLES DE BONNE CONCEPTION

Pas d'attribut calculé

- ex : prix total d'un facture calculable à partir des prix individuels

Séparation en plusieurs tables (cf cours suivant)

- Soit par l'algorithme reposant sur les DF
- Soit avec les formes normales
  - Vérifier que les tables sont bien en 3FN (voire en FNBC)

Fusionner les type-entités similaires

- ex : norealisateur, noproducteur, noacteur, nospect