Modèle relationnel des données

- Une base de données consiste en plusieurs tables (relations)
- Chaque table a un nom

- Dans une table chaque colonne a un nom
- Les noms des colonnes d'une table sont appelés attributs (ou champs)
- Chaque attribut a un domaine associé (i.e. l'ensemble des valeurs possibles pour cet attribut)

- Les données dans chaque table sont un ensemble de tuples (lignes)
  - un tuple fournit à chaque attribut une valeur de son domaine

Une table

Nom de la table

Attributs

Tuples

STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25

Exemple : les films sous forme de table

Films

titre	*annee*	*realisateur*
Alien	1979	Scott
Vertigo	1958	Hitchcock
Psychose	1960	Hitchcock
Kagemusha	1980	Kurosawa
Volte-face	1997	Woo
Pulp Fiction	1995	Tarantino
Titanic	1997	Cameron
Sacrifice	1986	Tarkovski

Schéma et instance d'une table

- La structure des données est rigide (toutes les lignes ont les mêmes attributs)
  - Cette structure est appelé schéma de la table :

STUDENT

Name	SSN	Phone	Adress	OfficePhone	Age	GPA

  - Aussi dénoté Student (Name, SSN, Phone, Adress, OfficePhone, Age, GPA)
  - L'ensemble des lignes est appelé instance de la table

- Le schéma est défini une fois pour toute
- L'instance varie dans le temps selon les données qu'on insère, modifie ou supprime

- Des contraintes d'intégrité sont en général associées à chaque table, qui empêchent les instances non valides (valeurs requises, uniques, etc…)

Schéma et instance d'une base de données

- Une bases de données est formée par plusieurs tables

STUDENT

Name	SSN	Phone	Adress	OfficePhone	Age	GPA
Bayer	347294	333	Albyn Pl.	367	18	3.24
Ashly	5784673	466	Queen St.	390	20	3.53
Davidson	4357387	589	Princes St.	678	25	3.25

COURSE

Course-Id	Title
12	CS
34	DB

EXAM

Student-SSN	Course-Id
347294	12
5784673	12
4357387	34
347294	34

- Schéma de la base de donnée : l'ensembles des schémas de toutes ses tables
- Instance de la base de données : l'ensemble des instances de toutes ses tables (i.e les données)

SQL

Langage déclaratif

Permet de manipuler à la fois le schéma et les instances d'une bases de données

Composante DDL (Data Definition Language)

- définir le schéma de la bases de données

Composante DML (Data Manipulation Language)

- manipuler les données (insérer/ supprimer/ mettre à jour)
- interroger (extraire de l'information de) des données de manière déclarative :

```
SELECT titre            -- l'attribut recherché
FROM Films               -- la table interrogée
WHERE realisateur = 'Varda';  -- le critère de sélection
```

Quelques éléments de SQL DDL (Data Definition Language) pour la création de schémas de BD

Création d'une base de données

```
CREATE DATABASE ma_base ; -- commande SQL
```

Au sein d'un serveur on peut créer plusieurs bases de données différentes

Chaque base à ses propres tables, mais une table appartient à une seule base

On peut associer des droits d'accès aux utilisateurs des bases, et des tables

Création d'une table (schéma)

Au sein de la base de données courante :

```
CREATE TABLE Films
(titre      VARCHAR(30),
 annee      INTEGER,
 realisateur  VARCHAR(30)
);
```

Les types des attributs varient d'un SGBD à l'autre

Cette déclaration permet de définir les noms et les domaines des attributs (leur « type ») et contient en général aussi la définition de plusieurs contraintes d'intégrité

À sa création, une table est "vide" (instance vide, on ne crée que le schéma)

Elle ne contient aucune donnée, i.e., aucun tuple

D'autres commandes SQL permettent d'insérer et/ou supprimer des lignes

Création d'une table : noms et domaines d'attributs

Chaque relation est définie par un nom de relation et une liste d'attributs :

```
CREATE TABLE nom_relation
(<element1 de la relation>,
 <element2 de la relation>,*        (* = optionnel)
 …
 <contraintes de la relation>*
);
```

Chaque attribut est défini par un nom d'attribut (tous distincts dans une relation) et un type de données :

```
<element de la relation> :=
<nom_attrib><type_donnes> [<contrainte_attrib>*]
```

Quelques types d'attributs SQL courants

- int : entier signé (sur 4 octets)
- bool : Booléen
- real : réel
- numeric(precision, echelle) : décimaux où echelle est le nombre de chiffres après la virgule et precision le nombre de chiffres totaux
- text : chaîne de caractères
- char(n) : chaîne d'*au plus* n caractères
- serial : entier à incrémentation automatique

C.f. https://www.postgresql.org/docs/9.6/static/datatype.html

Quelques types d'attributs SQL courants : dates et heures

- date : date au format Année-Mois-Jour (e.g., '2018-10-05')
- time : heure au format Heures-Minutes-Secondes (e.g. '09:01:42')
- timestamp : combine date et heure (e.g., '2018-10-05 09:01:42')
- year : stocke une année (e.g., '2018')

- Opérations sur ces types : l'égalité et l'ordre peuvent être utilisés.

  Si d_1 et d_2 sont des dates et d_1 < d_2, alors d_1 est avant d_2

Contraintes d'intégrité

Contraintes d'intégrité

- Contrainte d'intégrité : propriétés des données que l'on demande au système de garantir
- Exemples :
  - Un attribut doit toujours avoir une valeur
  - Un (ensemble d') attribut(s) identifie les lignes d’une table
  - Un attribut d'une table fait référence à l’identifiant d’une autre table
  - Un attribut ne peut prendre qu'une des valeurs prédéfinies d'un ensemble
  - etc.

Contrainte NOT NULL

Si la valeur d'un attribut n'est pas spécifiée pendant l'insertion, la valeur "vide" NULL lui sera affectée

Films

titre	*annee*	*realisateur*
Alien	1979	Scott
Vertigo	1958	Hitchcock
NULL	1960	Hitchcock
Kagemusha	NULL	Kurosawa
Volte-face	1997	Woo
Pulp Fiction	1995	Tarantino
Titanic	NULL	Cameron
Sacrifice	1986	Tarkovski

La contrainte NOT NULL sur un attribut d'une table impose que l'attribut ait une valeur non nulle

Contrainte NOT NULL

Spécifier des contraintes NOT NULL en SQL :

```
CREATE TABLE Films
(titre      VARCHAR(30) NOT NULL,
 annee      INTEGER NOT NULL,
 realisateur  VARCHAR(30)
);
```

Effet : génère une erreur à l'insertion d'une ligne qui ne spécifie pas la valeur du titre ou la valeur de l'année

En fait, on impose que le résultat des requêtes suivantes soit toujours vide :

```
SELECT titre FROM Films
WHERE titre IS NULL ;
```

```
SELECT titre FROM Films
WHERE annee IS NULL ;
```

Contraintes de clé

- Une clé d'une table:
  - Plus petit sous-ensemble d'attributs permettant d'identifier un tuple de manière unique
- Exemples :
  - nss est une clef de la table Personne(nss, nom, prénom, naissance, pays)
  - (ville, rue, numero) est une clef de la table Bâtiment (ville, rue, numero, #etages)
- Une table peut avoir plusieurs clefs
- Exemple :
  - pseudo est une clef de la table Utilisateur (pseudo, email, nom, prénom, mdp)
  - comme également email

Clés primaires

- Une table a toujours une clé dite
  - clé primaire

    (attributs soulignés ci-dessous)

    Film (titre, année, realisateur)

    Personne(nss, nom, prénom, naissance, pays)

    Utilisateur (pseudo, e-mail, nom, prénom, mdp)

    Bâtiment (ville, rue, numero, #etages)

  - les autres clés sont appelées clefs candidates (ou secondaires)
    - exemple : email pour la table Utilisateur

Spécifier les clés primaires en SQL

- Clé primaire comportant un seul attribut :

```
CREATE TABLE Films
(titre      VARCHAR(30) PRIMARY KEY,
 annee      INTEGER NOT NULL,
 realisateur  VARCHAR(30)
);
```

- On impose que le résultat de cette requête soit toujours vide :

```
SELECT F1.titre,F2.titre FROM Films F1, Films F2
WHERE F1.titre=F2.titre AND
(F1.realisateur<>F2.realisateur OR F1.annee<>F2.annee)
UNION
SELECT titre FROM Films WHERE titre IS NULL;
```

Spécifier les clés primaires en SQL

- Clé primaire comportant plusieurs attributs

```
CREATE TABLE Bâtiment (
    ville VARCHAR(50),
    rue VARCHAR(100),
    numero INTEGER,
    #etages INTEGER,
    PRIMARY KEY (ville, rue, numero)
);
```

- Remarque : PRIMARY KEY implique NOT NULL, pas besoin de le spécifier explicitement pour les attributs d'une clé
- Effet : la tentative d'insertion d'une clé primaire existante génère une erreur
- Chaque table devrait avoir une clé primaire

Clés candidates

- Les autres clés de la table, non choisies comme clés primaires, peuvent être spécifiées avec la contrainte UNIQUE

```
CREATE TABLE Personne (
      nss VARCHAR(20) PRIMARY KEY,
      nom VARCHAR(50) NOT NULL,
      prenom VARCHAR(50) NOT NULL,
      naissance INTEGER,
      pays VARCHAR(4),
      UNIQUE (nom, prenom, naissance)
);
```

- (nom, prenom, naissance) : clé candidate
- Effet : la tentative d'insertion d'une clé candidate existante génère une erreur
- Remarque : UNIQUE n'implique pas NOT NULL

Clés candidates

- Les autres clés de la table, non choisies comme clés primaires, peuvent être spécifiées avec la contrainte UNIQUE

```
CREATE TABLE Personne (
        nss VARCHAR(20) PRIMARY KEY,
        nom VARCHAR(50) NOT NULL,
        prenom VARCHAR(50) NOT NULL,
        naissance INTEGER,
        pays VARCHAR(4),
        UNIQUE (nom, prenom, naissance)
);
```

Impose que le résultat de la requête suivante soit vide :

```
SELECT P1.nom, P2.nom, P1.prenom, P2.prenom, P1.naissance,
P2.naissance FROM Personne P1, Personne P2 WHERE P1.nom=P2.nom and
P1.prenom=P2.prenom and P1.naissance=P2.naissance and
(P1.pays<>P2.pays or P1.nss<>P2.nss) ;
```

Clés étrangères

Contrainte entre deux tables : sert à relier des attributs d'un table avec la clef primaire d'une autre table

`Exemple`

Personne(<u>nss</u>, nom, prénom, naissance, pays)

Pays (<u>code</u>, nom) Personne[pays] ⊆ Pays[code]

Impose que la colonne `pays` de la table Utilisateur contienne uniquement des valeurs qui apparaissent dans la colonne `code` de la table Pays

Clés étrangères

Contrainte entre deux tables : sert à relier des attributs d'un table avec la clef primaire d'une autre table

`Exemple`

Personne(<u>nss</u>, nom, prénom, naissance, pays)

Pays (<u>code</u>, nom) Personne[pays] ⊆ Pays[code]

Personne

nss	*nom*	*preno*	*naissan*	*pay*
SDHF45FR9	Dupont	Luc	1943	FR
ERYE75EFHS	White	Alfred	1899	UK
JFB9745HSD	White	Ted	1910	UK
SDHF8465SJ	Brown	John	1946	US
SF745HSFJ7	Smith	Ross	1987	US
SHDGG475	Brown	James	1954	UK

Pays

cod	*nom*
FR	France
UK	Grande Bretagne
US	Etas Unis

Cette instance de la base de données satisfait la contrainte

Clés étrangères

Contrainte entre deux tables : sert à relier des attributs d'un table avec la clef primaire d'une autre table

`Exemple`

Personne(<u>nss</u>, nom, prénom, naissance, pays)

Pays (<u>code</u>, nom) Personne[pays] ⊆ Pays[code]

Personne

nss	*nom*	*preno*	*naissan*	*pay*
SDHF45FR9	Dupont	Luc	1943	FR
ERYE75EFHS	White	Alfred	1899	UK
JFB9745HSD	White	Ted	1910	UK
SDHF8465SJ	Brown	John	1946	US
SF745HSFJ7	Smith	Ross	1987	IT
SHDGG475	Brown	James	1954	UK

Pays

cod	*nom*
FR	France
UK	Grande Bretagne
US	Etas Unis

Cette instance de la base de données VIOLE la contrainte

Clés étrangères

Spécifier une contrainte de clef étrangère en SQL :

```
CREATE TABLE Pays (
      code VARCHAR(4)
          PRIMARY KEY,
      nom VARCHAR(50)
);
```

```
CREATE TABLE Personne (
      nss VARCHAR(20) PRIMARY KEY,
      nom VARCHAR(50) NOT NULL,
      prenom VARCHAR(50) NOT NULL,
      naissance INTEGER,
      pays VARCHAR(4),
      UNIQUE (nom, prenom, naissance),
      FOREIGN KEY (pays) REFERENCES Pays(code)
);
```

Les attributs reliés doivent avoir exactement le même type

FOREIGN KEY : les attribut référencés (ceux qui suivent la clause REFERENCES) doivent être clef primaire de leur table

Clés étrangères

Spécifier une contrainte de clef étrangère en SQL :

```
CREATE TABLE Pays (
      code VARCHAR(4)
          PRIMARY KEY,
      nom VARCHAR(50)
);
```

```
CREATE TABLE Personne (
      nss VARCHAR(20) PRIMARY KEY,
      nom VARCHAR(50) NOT NULL,
      prenom VARCHAR(50) NOT NULL,
      naissance INTEGER,
      pays VARCHAR(4),
      UNIQUE (nom, prenom, naissance),
      FOREIGN KEY (pays) REFERENCES Pays
);
```

En l'absence de nom d'attribut après REFERENCES, c'est l'attribut clef primaire de la table référencée qui est utilisée comme attribut référencé.

Clés étrangères

Spécifier une contrainte de clef étrangère en SQL :

```
CREATE TABLE Pays (
      code VARCHAR(4)
          PRIMARY KEY,
      nom VARCHAR(50)
);
```

```
CREATE TABLE Personne (
      nss VARCHAR(20) PRIMARY KEY,
      nom VARCHAR(50) NOT NULL,
      prenom VARCHAR(50) NOT NULL,
      naissance INTEGER,
      pays VARCHAR(4),
      UNIQUE (nom, prenom, naissance),
      FOREIGN KEY (pays) REFERENCES Pays(code)
);
```

Impose que le résultat de la requête suivante est toujours nul :

```
SELECT pays
FROM Personne
WHERE pays NOT IN (SELECT code
                   FROM Pays) ;
```

Autres contraintes d'intégrité référentielle

Spécifier une contrainte d'inclusion en SQL :

```
CREATE TABLE Pays (
      code VARCHARR(4)
          PRIMARY KEY,
      nom VARCHAR(50) UNIQUE
);
```

```
CREATE TABLE Personne (
      nss VARCHAR(20) PRIMARY KEY,
      nom VARCHAR(50) NOT NULL,
      prenom VARCHAR(50) NOT NULL,
      naissance INTEGER,
      pays VARCHAR(50),
      UNIQUE (nom, prenom, naissance),
      pays REFERENCES Pays(nom)
);
```

Les attributs reliés doivent avoir exactement le même type

Les attribut référencés (ceux qui suivent la clause REFERENCES) doivent être clef candidate de leur table (UNIQUE).

Attention : ce n'est pas possible dans tous les systèmes.

Clés étrangères

Conséquences de la contrainte de clé étrangère
(sur les tables Personne/Pays par exemple)

- Lors d'une insertion dans la table Personne :
  - vérification : la valeur de *pays* doit être parmi les valeurs de code dans la table Pays
- Lors de la mise à jour / suppression d'un code dans la table Pays
  - vérification : aucune personne n'appartient à ce pays

- Si ces vérifications n'ont pas de succès
  - comportement par défaut : erreur
  - on peut demander explicitement un autre comportement (ON DELETE CASCADE….)

Contraintes génériques

D'autres contraintes sur les données ne peuvent pas être exprimées par les mécanismes vus jusqu'à maintenant

Exemple :

la note d'un examen est entre 0 et 20

le prix soldé d'un article est inférieur au prix entier

le salaire d'un manager est plus élevé que celui des ses subalternes

En SQL :

Contraintes sur une seule table : clause CHECK

Contraintes sur plusieurs tables : Assertions SQL

Clause CHECK

- Exemple : dans la table Article:

```
CREATE TABLE Article (
    id INTEGER PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    prix NUMERIC NOT NULL,
    prix_solde NUMERIC,
    CHECK (prix > prix_solde)
);
```

Impose que le résultat de la requête suivante est vide :

```
SELECT id FROM Article WHERE
    prix <= prix_solde ;
```

Clause CHECK

- Autre exemple :

```
CREATE TABLE Reservation (
   id INTEGER PRIMARY KEY,
   nom VARCHAR(50) NOT NULL,
   salle VARCHAR(50) ,
   debut TIMESTAMP,
   fin TIMESTAMP,
   CHECK (debut < fin)
);
```

Impose que le résultat de la requête suivante est vide :

```
SELECT id FROM Reservation WHERE
   fin <= debut ;
```

Clause CHECK

- Il est possible de nommer la contrainte :

```
CREATE TABLE Produits (
    prod_no INTEGER PRIMARY KEY,
    nom TEXT,
    prix NUMERIC CONSTRAINT prix_positif CHECK (prix > 0) ,
    debut TIMESTAMP,
    fin TIMESTAMP,
);
```

- Le mot clef CONSTRAINT permet de nommer la contrainte
- Clarifie les messages d'erreur
- Permet de faire référence à la contrainte ultérieurement pour la modifier

Détruire une table

Détruire la table (schéma et données):

```
DROP TABLE Films ;
```

Supprimer toutes les données (mais garder le schéma):

```
TRUNCATE TABLE Films ;
```

Modifier une table

Ajouter un attribut :

```
ALTER TABLE < nom > ADD COLUMN nouvelle_colonne type ;
```

Exemple :

```
ALTER TABLE Films ADD COLUMN durée int ;
```

Supprimer un attribut :

```
ALTER TABLE < nom > DROP COLUMN nom_colonne ;
```

Exemple :

```
ALTER TABLE Films DROP COLUMN acteur ;
```

Modifier une table

Renommer une table :

```
ALTER TABLE Films RENAME TO Movies ;
```

Renommer une colonne :

```
ALTER TABLE Movies RENAME COLUMN titre TO title ;
```

Modifier une table

Ajouter une contrainte NOT NULL :

```
ALTER TABLE Produits ALTER COLUMN prix
SET NOT NULL ;
```

Supprimer une contrainte NOT NULL :

```
ALTER TABLE Produits ALTER COLUMN prix
DROP NOT NULL ;
```

Modifier une table

Ajouter une contrainte de clef étrangère :

```
ALTER TABLE Reservation ADD CONSTRAINT nomfk FOREIGN KEY (nom)
REFERENCES Personne(nom) ;
```

Ajouter une clef primaire générée de façon automatique :

```
ALTER TABLE Films ADD PRIMARY KEY (id_film) ;
```

(attention : possible seulement s'il n'y en avait pas déjà)

Modifier une table

Ajouter une contrainte CHECK :

```
ALTER TABLE Produits ADD CONSTRAINT prix_positif
CHECK (prix > 0) ;
```

Supprimer une contrainte CHECK :

```
ALTER TABLE <nom > DROP CONSTRAINT prix_positif ;
```

Renommer une contrainte CHECK :

```
ALTER TABLE Produits RENAME CONSTRAINT prix_positif
TO prix_sup_zero ;
```

Manipulation de données
(SQL DML : Data Modification Language)

Insérer des données

```
INSERT INTO Films
VALUES (‘Wild Style’, 1983, 'Ahearn');
```

Note : on peut ne saisir que quelques attributs, et dans un ordre différent de celui défini pour la table.

```
INSERT INTO Films (realisateur, titre)
VALUES ('Korine', 'Gummo’);
```

Si la valeur d’un attribut n’est pas spécifiée pendant l’insertion, la valeur NULL lui sera affectée

Insérer des données

Resultat :

Films

titre	*annee*	*realisateur*
Alien	1979	Scott
Vertigo	1958	Hitchcock
Psychose	1960	Hitchcock
Kagemusha	1980	Kurosawa
Volte-face	1997	Woo
Wild Style	1983	Ahearn
Titanic	1997	Cameron
Sacrifice	1986	Tarkovski
Gummo	NULL	Korine

Attention : génère une erreur si les attributs non spécifiés ont une contrainte NOT NULL

Insérer des données

- On peut faire une insertion par le biais d'une requête
- La clause VALUES est remplacée par une requête
- les résultats de la requête sont insérés comme valeurs dans la table

Exemple : alimentation d'une nouvelle table à l'aide des données d'une autre

```
INSERT INTO FournisseursParisiens
   (SELECT *
   FROM Fournisseur
   WHERE Ville='Paris');
```

Insertion de données et valeurs par défaut

Une clause **DEFAULT** peut être spécifiée pour un attribut :

```
CREATE TABLE Notation (
   titre_film VARCHAR(50) NOT NULL,
   pseudo VARCHAR(50) NOT NULL,
   note INTEGER NOT NULL DEFAULT 0
);
```

Si la valeur de l'attribut n'est pas spécifiée lors d'une insertion, sa valeur sera celle définie par la clause DEFAULT.

`INSERT INTO Notation VALUES ('Alien', 'lili66')` ne génère pas d'erreur et insère la ligne :

Notation

titre_film	pseudo	note
…	…	…
Alien	lili66	0

Insertion : clés à incrémentation automatique

- La plupart des SGBDs offrent la possibilité de définir des attributs dont la valeur est un entier automatiquement incrémenté à chaque insertion
  (pas présent dans le standard SQL)
- Très utile pour définir des identifiants “internes”

(syntaxe PostgreSQL)

```
CREATE TABLE Artiste (
        id SERIAL PRIMARY KEY,
        nom VARCHAR(50) NOT NULL,
        prenom VARCHAR(50) NOT NULL,
        naissance INTEGER,
        UNIQUE (nom, prenom, naissance)
);
```

- Pas besoin de fournir l’`id` lors de l’insertion d’un Artiste : il sera automatiquement affecté au dernier id de la séquence + 1

Insertion : clés à incrémentation automatique

Exemple. Supposer la table `Artiste` initialement vide.

```
INSERT INTO Artiste (nom, prenom, naissance)
VALUES ('Scott', 'Ridley', 1943);

INSERT INTO Artiste (nom, prenom, naissance)
VALUES ('Hitchcock', 'Alfred', 1899);
```

- Remarque : on ne précise pas la clé "id".

Insertion : clés à incrémentation automatique

```
SELECT * FROM Artiste;
+-------+-----------+---------+-----------+
| id    | nom       | prenom  | naissance |
+-------+-----------+---------+-----------+
|     1 | Scott     | Ridley  |      1943 |
|     2 | Hitchcock | Alfred  |      1899 |
+-------+-----------+---------+-----------+
```

- La clé a été automatiquement générée (séquence croissante)

Insertion : clés à incrémentation automatique

Une clé SERIAL peut également être précisée explicitement

```
INSERT INTO Artiste (id, nom, prenom, naissance)
VALUES (4, 'Woo', 'John', 1946);

SELECT * FROM Artiste;
```

```
+-------+-----------+---------+------------+
| id    | nom       | prenom  | naissance  |
+-------+-----------+---------+------------+
|     1 | Scott     | Ridley  |       1943 |
|     2 | Hitchcock | Alfred  |       1899 |
|     4 | Woo       | John    |       1946 |
+-------+-----------+---------+------------+
```

Insertion: reprise de l' incrémentation automatique

```
INSERT INTO Artiste (nom, prenom, naissance)
VALUES ('Kurosawa', 'Akira', 1910);

SELECT * FROM Artiste;
```

```
+-------+-----------+--------+-----------+
| ident | nom       | prenom | naissance |
+-------+-----------+--------+-----------+
|     1 | Scott     | Ridley |      1943 |
|     2 | Hitchcock | Alfred |      1899 |
|     4 | Woo       | John   |      1946 |
|     3 | Kurosawa  | Akira  |      1910 |
+-------+-----------+--------+-----------+
```

Insertion : reprise de l' incrémentation automatique

```
+-------+-----------+---------+------------+
| ident | nom       | prenom  | naissance  |
+-------+-----------+---------+------------+
|     1 | Scott     | Ridley  |       1943 |
|     2 | Hitchcock | Alfred  |       1899 |
|     4 | Woo       | John    |       1946 |
|     3 | Kurosawa  | Akira   |       1910 |
+-------+-----------+---------+------------+
```

```
INSERT INTO Artiste (nom, prenom, naissance)
VALUES ('Tarantino', 'Quentin', 1963);
```

```
ERROR:  duplicate key value violates unique constraint
DETAIL:  Key (id)=(4) already exists.
```

Insertion : reprise de l' incrémentation automatique

La séquence continue toujours du dernier id généré, même si sa ligne a été supprimée, ou si l'insertion a échoué

```
DELETE FROM Artiste WHERE id > 2;
INSERT INTO Artiste (nom, prenom, naissance)
  VALUES ('Kurosawa', 'Akira', 1910);
SELECT * FROM Artiste;
```

```
+-------+-----------+---------+-----------+
| ident | nom       | prenom | naissance |
+-------+-----------+---------+-----------+
|     1 | Scott     | Ridley |      1943 |
|     2 | Hitchcock | Alfred |      1899 |
|     5 | Kurosawa  | Akira  |      1910 |
+-------+-----------+---------+-----------+
```

Clés à incrémentation automatique : reinitialisation

Pour réinitialiser le compteur de la séquence :

```
ALTER SEQUENCE Artiste_id_seq  RESTART WITH 4;
```

NomTable_nomClef_seq

Clés à incrémentation automatique : réinitialisation

```
SELECT * FROM Artiste;
```

```
+-------+-----------+---------+------------+
| id    | nom       | prenom  | naissance  |
+-------+-----------+---------+------------+
|     1 | Scott     | Ridley  |       1943 |
|     2 | Hitchcock | Alfred  |       1899 |
+-------+-----------+---------+------------+
```

```
ALTER SEQUENCE Artiste_id_seq  RESTART WITH 4;
INSERT INTO Artiste (nom, prenom, naissance)
   VALUES ('Kurosawa', 'Akira', 1910);
SELECT * FROM Artiste;
```

```
+-------+-----------+---------+------------+
| id    | nom       | prenom  | naissance  |
+-------+-----------+---------+------------+
|     1 | Scott     | Ridley  |       1943 |
|     2 | Hitchcock | Alfred  |       1899 |
|     4 | Kurosawa  | Akira   |       1910 |
+-------+-----------+---------+------------+
```

Modifier des données

```
UPDATE Films
SET realisateur = 'Wu'
WHERE realisateur = 'Woo';
```

Met à jour le champs `réalisateur` de toutes les lignes sélectionnées par la clause `WHERE`.

Films

titre	*annee*	*realisateur*
Alien	1979	Scott
Kagemusha	1980	Kurosawa
Volte-face	1997	Wu
Pulp Fiction	1995	Tarantino
Titanic	1997	Cameron
Sacrifice	1986	Tarkovski
Match Point	NULL	Allen

Modifier des données

Forme generale :

```
UPDATE <table>
SET att1 = val1,…, attn = valn
WHERE <condition> ;
```

`employe (id_employe, nom, prenom, sexe, salaire, prime)`

augmenter le salaire de toutes les femmes de 10% et leur donner une prime de 150 euros

```
update employe
set salaire = salaire * 1.1,
      prime = prime + 150
where sexe = ‘F’;
```

Supprimer des données

```
SELECT * FROM Films ;
```

Films

titre	*annee*	*realisateur*
Alien	1979	Scott
Vertigo	1958	Hitchcock
Psychose	1960	Hitchcock
Kagemusha	1980	Kurosawa
Volte-face	1997	Woo
Pulp Fiction	1995	Tarantino
Titanic	1997	Cameron
Sacrifice	1986	Tarkovski
Match Point	NULL	Allen

```
DELETE FROM Films WHERE annee <= 1960 ;
```

Supprimer des données

```
DELETE FROM Films WHERE annee <= 1960;
SELECT * FROM Films
```

Films

titre	*annee*	*realisateur*
Alien	1979	Scott
Kagemusha	1980	Kurosawa
Volte-face	1997	Woo
Pulp Fiction	1995	Tarantino
Titanic	1997	Cameron
Sacrifice	1986	Tarkovski
Match Point	NULL	Allen

Supprime toutes les lignes pour lesquelles l'année est inférieure ou égale à 1960.

WHERE est suivi d'une condition à évaluer sur chaque ligne

condition WHERE : même syntaxe que dans les requêtes SELECT

Supprimer des données

```
DELETE FROM Films ;
```

Supprime toutes les lignes de la table Films

(équivalent à `TRUNCATE Films ;`)

TRUNCATE est toutefois plus rapide et utilise moins de ressources (pas d'indication du nombre de tuples supprimés et pas d'enregistrement des modifications dans le journal).

Autre différence : TRUNCATE réinitialise la valeur de l'auto-incrément, s'il y en a un (ce que ne fait pas DELETE).

Supprimer des données

Exemple :

```
DELETE FROM Seance
WHERE titre IN
(SELECT titre FROM Film
WHERE realisateur='Polanski');
```

Supprime de la table Séance tous les films de Polanski.

```
DELETE FROM Film
WHERE titre IN
(SELECT titre FROM Seance
WHERE cinema='Le Champo');
```

Y a-t-il un problème ?

Maintien de l'intégrité référentielle

```
DELETE FROM Film
WHERE titre IN
(SELECT titre FROM Seance
WHERE cinema='Le Champo');
```

- En fait, on peut effacer ici un tuple de Film dont la valeur de titre apparait comme valeur de titre d'un tuple de Séance (s'il y a des films qui passent au Champo).
  Or titre apparait comme clef étrangère dans Seance, où il fait référence à Film !

- Même problème avec :

```
UPDATE Film
SET titre = 'Style fou'
WHERE titre = 'Wild Style';
```

Maintien de l'intégrité référentielle

Plusieurs politiques pour gérer ces problèmes :

- Par défaut : le système rejette toute modification ne respectant pas les contraintes d'intégrité (mot clef RESTRICT)

- En cascade : les modifications sur l'attribut référencé sont effectuées aussi sur l'attribut qui référence (la clef étrangère)

- Set null : les modifications sur l'attribut référencé sont répercutées sur l'attribut qui référence en mettant sa valeur à NULL

- Set default : les modifications sur l'attribut référencé sont répercutées sur l'attribut qui référence en mettant sa valeur à défaut (quand il existe une valeur par défaut)

Maintien de l'intégrité référentielle

Il faut prévoir ce type de problèmes en amont :

```
CREATE TABLE Seance
(id_seance serial PRIMARY KEY
titre     VARCHAR(30), FOREIGN KEY REFERENCES Film(titre)
cinema  VARCHAR(30)
ON DELETE SET NULL
ON UPDATE CASCADE
);
```

Cette déclaration force :

- à répercuter l'effacement d'un tuple dans Film en mettant tous les champs titre à null lorsque ceux-ci avaient pour valeur celle du titre effacé
- à répercuter la mise à jour d'un tuple dans Film (pour ce qui concerne le champ titre) en opérant la même mise à jour dans les tuples de Seance concernés

Maintien de l'intégrité référentielle

Il faut prévoir ce type de problèmes en amont :

```
CREATE TABLE Seance
(id_seance serial PRIMARY KEY
titre     VARCHAR(30), FOREIGN KEY REFERENCES Film(titre)
cinema  VARCHAR(30)
ON DELETE CASCADE
ON UPDATE CASCADE
);
```

Cette déclaration force :

- à répercuter l'effacement d'un tuple dans Film en effaçant tous les tuples correspondant dans la table Séance
- à répercuter la mise à jour d'un tuple dans Film (pour ce qui concerne le champ titre) dans les tuples de Seance concernés

Maintien de l'intégrité référentielle

Un autre exemple :

```
CREATE TABLE Livraison
(id_seance serial PRIMARY KEY,
nomF      VARCHAR(30), NOT NULL REFERENCES Fournisseur ON UPDATE
CASCADE,
dateLiv  DATE ,
quantite Int(8),
nomP      VARCHAR(30) REFERENCES Piece ON DELETE SET NULL ,
);
```

i.e., les instructions peuvent être différentes pour différentes clefs étrangères

Maintien de l'intégrité référentielle

Les mêmes précautions sont nécessaires avec DROP, la séquence :

```
DROP Table Films ;
DROP Table Séance ;
```

générera un message d'erreur : si Séance est non vide, les contraintes de clef étrangères de Séance seront violées une fois la table Films supprimée

Une solution :

```
DROP Table Séance ;
DROP Table Films ;
```

Maintien de l'intégrité référentielle

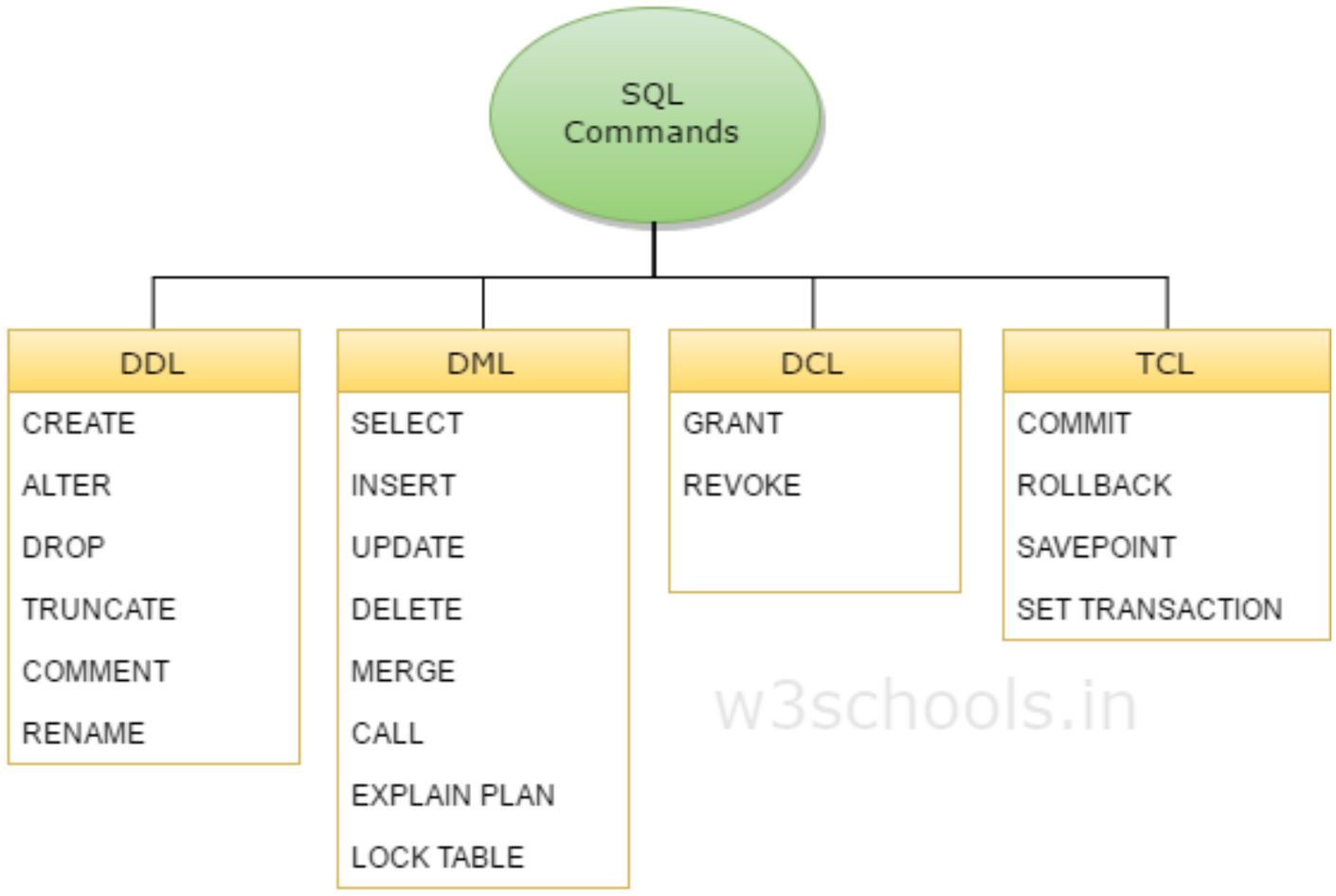
En revanche, il n'est pas toujours évident de hiérarchiser les dépendances au delà de deux tables et / ou dépendances (c.f. chaînes complexes de dépendances de clefs étrangères à travers plusieurs relations), préférer donc :

DROP Table Séance ON DELETE CASCADE ;

DROP Table Films ON DELETE CASCADE ;

DROP Table T ON DELETE CASCADE : supprime aussi toutes les tables qui dépendent de T

Les autres commandes SQL



DCL = data control language

TCL = transaction control language

(on ne verra ni l'un ni l'autre dans ce cours)