SAE.104 / Base de données

Enzo Le Meste
Groupe: Lerne

Sommaire:

1) Script manuel de la base de données:

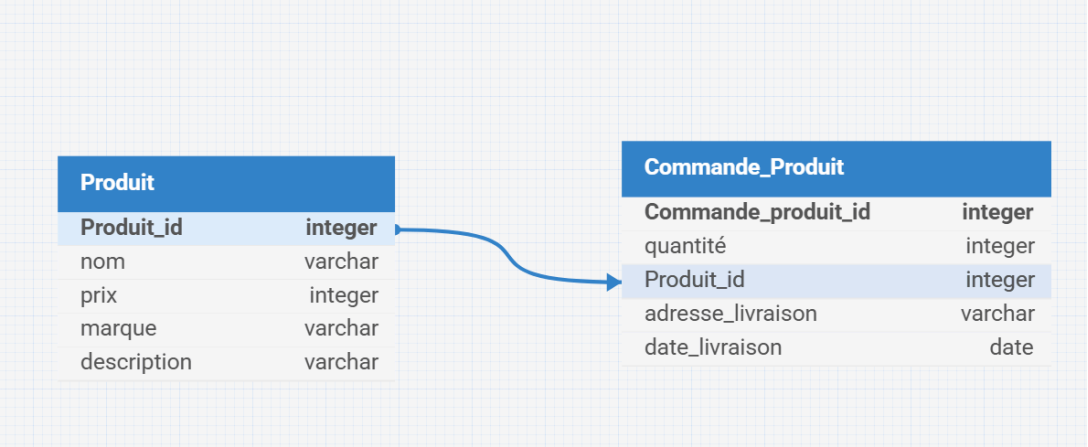
Création des tables :

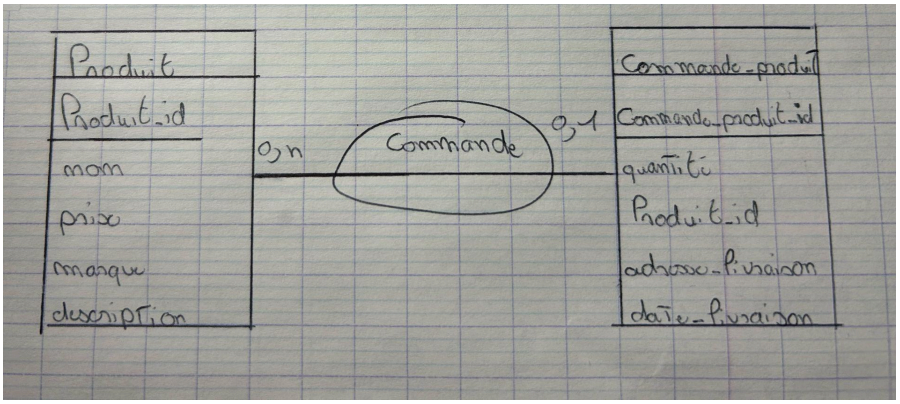
```
CREATE TABLE region (
     region_code INTEGER PRIMARY KEY,
     name VARCHAR NOT NULL
);

CREATE TABLE sub_region (
sub_region_code INTEGER PRIMARY KEY,
name VARCHAR,
region_code INTEGER REFERENCES region (region_code)
);

CREATE TABLE country (
country_code SERIAL PRIMARY KEY,
name VARCHAR,
iso2 VARCHAR,
iso3 INTEGER,
sub_region_code INTEGER REFERENCES sub_region
(sub_region_code)
);
```

```
CREATE TABLE disaster (
disaster_code INTEGER PRIMARY KEY,
disaster VARCHAR
);


CREATE TABLE climate_disaster (
country_code INTEGER REFERENCES country (country_code),
disaster_code INTEGER REFERENCES disaster
(disaster_code),
year INTEGER,
number INTEGER,
PRIMARY KEY (country_code, disaster_code, year)
);
```

2)Modélisation et script de la base de données:

Association Fonctionnelle :





Description des tables :

1. ***Produit***

   Représente les informations liées aux produits disponibles pour la commande.

   ***Colonnes** :*

   Produit_id : INTEGER(clé primaire) : Identifiant unique pour chaque produit.

   nom : VARCHAR Nom du produit (exemple : "Télévision", "Ordinateur").

   prix : INTEGER Prix du produit en entier (en euros, dollars, etc.).

   marque : VARCHAR Marque associée au produit (exemple : "Samsung", "Apple").

   description : VARCHAR Détails supplémentaires sur le produit.

2. ***Commande_Produit***

   Représente les détails des commandes liées à un produit particulier.

   ***Colonnes** :*

   Commande_produit_id (clé primaire) : INTEGER Identifiant unique pour chaque commande produit.

   quantité : INTEGER Quantité commandée pour ce produit.

   *Produit_id (clé étrangère) : INTEGER Référence au produit commandé (relation avec la table **Produit**).*

   adresse_livraison : VARCHAR Adresse où le produit sera livré.

   date_livraison : DATE Date prévue pour la livraison.

Relation entre les tables :

La table Commande_Produit contient une clé étrangère Produit_id qui fait référence à Produit_id dans la table ***Produit****.*

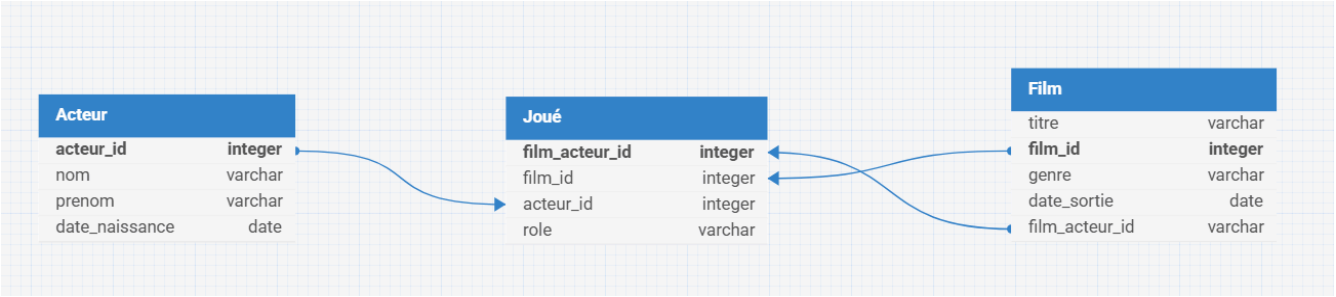
Cela indique qu'une commande spécifique est liée à un produit particulier.

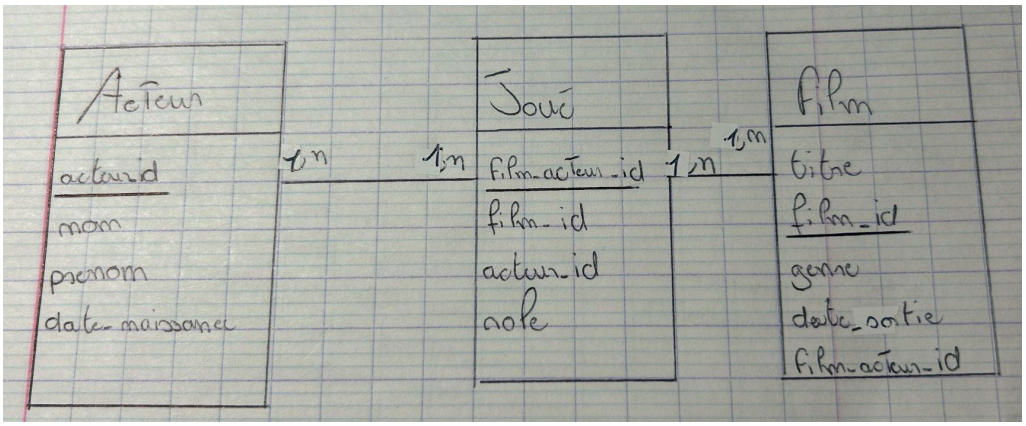
Cardinalité *:*

Un produit (dans la table ***Produit****) peut être associé à* ***plusieurs commandes*** *dans la table* ***Commande_Produit****.*

Chaque commande dans ***Commande_Produit*** *ne peut référencer qu'un seul produit.*

Association Maillé:





Description des tables :

1. ***Acteur***

   Représente les informations sur les acteurs dans la base de données.

   ***Colonnes** :*

   acteur_id (clé primaire) : INTEGER Identifiant unique pour chaque acteur.

   nom : VARCHAR Nom de l'acteur.

   prenom : VARCHAR Prénom de l'acteur.

   date_naissance : DATE Date de naissance de l'acteur.

2. ***Film***

   Représente les informations sur les films.

   ***Colonnes** :*

   film_id (clé primaire) : INTEGER Identifiant unique pour chaque film.

   titre : VARCHAR Titre du film.

   genre : VARCHAR Genre du film (exemple : "Action", "Drame").

   date_sortie : DATE Date de sortie du film.

   film_acteur_id : INTEGER Champ permettant de gérer une relation supplémentaire, potentiellement redondante.

3. ***Joué***

   Représente la table d'association entre les acteurs et les films.

   ***Colonnes** :*

film_acteur_id (clé primaire) : INTEGER Identifiant unique pour chaque association entre un acteur et un film.

film_id (clé étrangère) : INTEGER Référence au film concerné (relation avec la table Film).

acteur_id (clé étrangère) : INTEGER Référence à l'acteur concerné (relation avec la table Acteur).

role : VARCHAR Rôle joué par l'acteur dans le film.

Relation entre les tables :

1. ***Acteur ↔ Joué***

   Un acteur peut avoir joué dans plusieurs films (relation 1-N entre Acteur et Joué).

   Chaque enregistrement dans la table Joué correspond à un rôle joué par un acteur dans un film spécifique.

2. ***Film ↔ Joué***

   Un film peut avoir plusieurs acteurs jouant différents rôles (relation 1-N entre Film et Joué).

   Chaque enregistrement dans la table Joué capture les détails de l'association entre un film et un acteur.

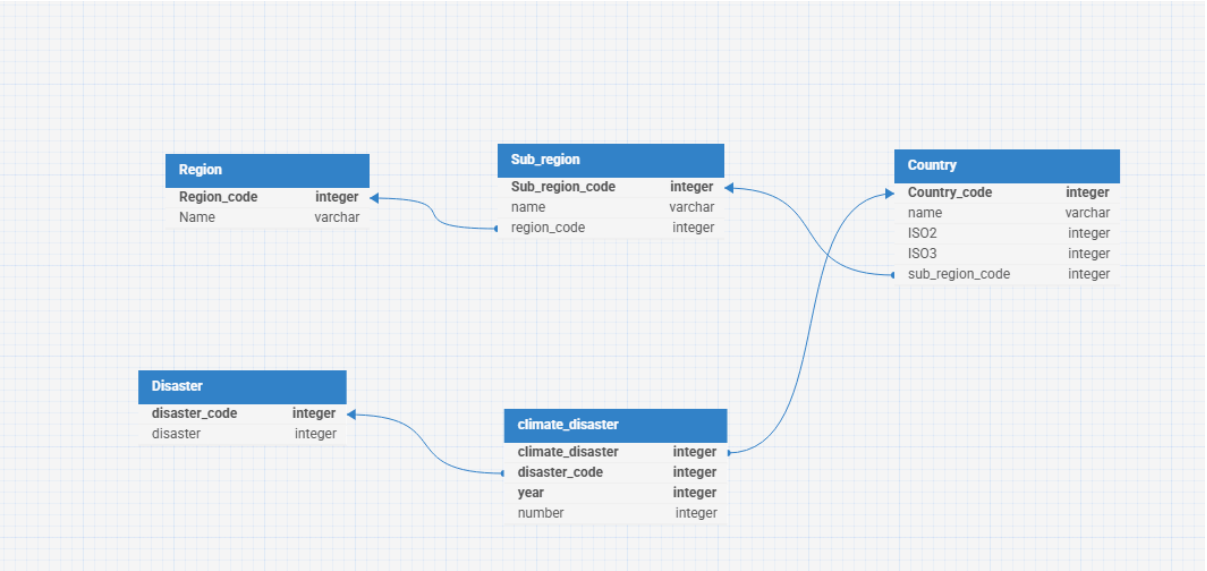
3. ***Joué** (Table d'association)*

   Cette table établit une relation 1-N entre les acteurs et les films :

   Un acteur peut apparaître dans plusieurs films.

   Un film peut avoir plusieurs acteurs.

Modèles physique de données réaliser par l'AGL:



Script SQL de création des tables généré auto par l'AGL:

```
CREATE TABLE IF NOT EXISTS "Region" (
        "Region_code" serial NOT NULL UNIQUE,
        "Name" varchar(255) NOT NULL,
        PRIMARY KEY ("Region_code")
);

CREATE TABLE IF NOT EXISTS "Sub_region" (
        "Sub_region_code" serial NOT NULL UNIQUE,
        "name" varchar(255) NOT NULL,
        "region_code" bigint NOT NULL,
        PRIMARY KEY ("Sub_region_code")
);

CREATE TABLE IF NOT EXISTS "Country" (
        "Country_code" serial NOT NULL UNIQUE,
        "name" varchar(255) NOT NULL,
        "ISO2" bigint NOT NULL,
        "ISO3" bigint NOT NULL,
```

```
        "sub_region_code" serial NOT NULL,
        PRIMARY KEY ("Country_code")
);

CREATE TABLE IF NOT EXISTS "Disaster" (
        "disaster_code" serial NOT NULL UNIQUE,
        "disaster" bigint NOT NULL,
        PRIMARY KEY ("disaster_code")
);

CREATE TABLE IF NOT EXISTS "climate_disaster" (
        "climate_disaster" serial NOT NULL UNIQUE,
        "disaster_code" bigint NOT NULL,
        "year" bigint NOT NULL,
        "number" bigint NOT NULL,
        PRIMARY KEY ("climate_disaster", "disaster_code", "year")
);


ALTER TABLE "Sub_region" ADD CONSTRAINT "Sub_region_fk2"
FOREIGN KEY ("region_code") REFERENCES "Region"("Region_code");
ALTER TABLE "Country" ADD CONSTRAINT "Country_fk4" FOREIGN
KEY ("sub_region_code") REFERENCES
"Sub_region"("Sub_region_code");

ALTER TABLE "climate_disaster" ADD CONSTRAINT
"climate_disaster_fk0" FOREIGN KEY ("climate_disaster")
REFERENCES "Country"("Country_code");

ALTER TABLE "climate_disaster" ADD CONSTRAINT
"climate_disaster_fk1" FOREIGN KEY ("disaster_code") REFERENCES
"Disaster"("disaster_code");
```

Discussion sur les différences entre les scripts produits manuellement et automatiquement :

Il y a de nombreuses différences entre les scripts rédigé automatiquement et manuellement,

Manuellement, les noms des tables et des colonnes sont souvent en minuscules ou conformes à un style propre défini par le développeur (exemple : region_code, country_code, iso2).Le style peut varier et ne suit pas toujours un standard strict. Tandis que automatiquement,les noms des tables et colonnes sont souvent entourés de guillemets doubles (exemple : "Region_code", "Sub_region_code") pour des raisons de compatibilité avec différents SGBD (systèmes de gestion de bases de données).Les noms sont souvent formatés en CamelCase ou PascalCase, même si cela peut dépendre du générateur.
Ensuite,manuellement,le développeur choisit des types de données simples et cohérents selon ses besoins (par exemple, INTEGER, VARCHAR).
Les types sont choisis pour chaque colonne selon les contraintes métiers.
Alors que automatiquement, le générateur peut attribuer des types par défaut qui sont parfois surdimensionnés (par exemple, bigint ou varchar(255) même si une taille moindre suffirait).
Les types sont souvent standards pour garantir la portabilité (exemple : bigint pour les identifiants).
De plus, manuellement,les contraintes (clés primaires, clés étrangères, uniques) sont spécifiées de manière explicite et uniquement si nécessaire.
Les noms des contraintes ne sont pas toujours nommés, ou suivent une convention spécifique choisie par le développeur.
Pendant que automatiquement, les générateurs ajoutent systématiquement des contraintes pour garantir l'intégrité des données.
Les noms des contraintes sont souvent générés automatiquement (exemple : Sub_region_fk2, Country_fk4), ce qui peut rendre le script moins lisible.
En excédent, manuellement les scripts créés à la main ne contiennent souvent pas de vérifications supplémentaires comme IF NOT EXISTS.
Les colonnes obligatoires ou par défaut ne sont spécifiées que si nécessaire.
Quoique,automatiquement les scripts incluent souvent des instructions comme CREATE TABLE IF NOT EXISTS pour éviter des erreurs en cas de création multiple.
Des contraintes comme NOT NULL sont ajoutées systématiquement même si elles ne sont pas toujours nécessaires.
En outre,manuellement les clés étrangères sont définies explicitement dans la déclaration des colonnes ou dans des ALTER TABLE distincts.
Les relations sont pensées de manière personnalisée selon le modèle de données.Tandis que,automatiquement les références aux clés étrangères sont souvent générées dans des blocs ALTER TABLE séparés, comme dans votre script.

Les relations sont souvent générées par défaut et peuvent inclure des contraintes inutiles.
En supplément,manuellement un script manuel est souvent plus concis et ne contient pas de redondances inutiles.
Exemple : Pas de création de champs inutilisés.
Alors que, automatiquement un générateur peut introduire des colonnes ou des structures redondantes (comme des champs ou des contraintes inutilisées), ce qui peut alourdir le script.
Enfin manuellement,les conventions sont définies par le développeur (par exemple, tout en minuscules, noms explicites, etc.).
Cela peut être moins uniforme si plusieurs développeurs interviennent.
Pour finir, automatiquement les générateurs suivent des conventions strictes pour s'assurer que tout est uniforme (exemple : noms en PascalCase, utilisation de types standards).

3)Peuplement des tables:

Peuplement des tables à partir de la table temporaire temp_disasters:

1. Création de la table temporaire

Nous allons commencer par créer une table temporaire nommée temp_disasters , qui servira à importer et manipuler les données avant de les répartir dans les différentes tables de la base de données.

```
CREATE TABLE temp_disasters (
country_name VARCHAR,
disaster_type VARCHAR,
year INTEGER,
region_name VARCHAR,
sub_region_name VARCHAR,
disaster_code INTEGER
);
```

Description de la table temp_disasters:

• country_name : Nom du pays
• disaster_type : Type de catastrophe (exemple : “Feux de fôret”, “sécheresse”).
• year : Année où la catastrophe a eu lieu.
• region_name : Nom de la région.
• sub_region_name : Nom de la sous-région.
• disaster_code : Code donnée par type de catastrophe.

2. Importation des données:

Les données ont été importées dans une table temporaire à partir d'un fichier CSV nommé Climate_related_disasters_frequency.csv.zip, que j'avais décompressé sous Linux à l'aide de la commande suivante :
unzip Climate_related_disasters_frequency.csv.zip
J’ai ensuite pris le fichier suivant Climate_related_disasters_frequency.csv que j’ai renommée disasters.csv.
Voici la commande que j’ai utilisée :
\copy temp_disasters FROM /home/lemeste/Downloads/disasters.csv WITH CSV HEADER DELIMITER ',' ;

• disasters.csv : Fichier contenant les données brutes.
• WITH CSV HEADER : Indique que la première ligne du fichier contient les noms des colonnes.
• DELIMITER ',' : Spécifie que les colonnes dans le fichier sont séparées par des virgules.

3. Peuplement des tables principales:

a. Table region

Nous avons inséré les informations relatives aux régions en éliminant les doublons présents dans la colonne region_name.

```
INSERT INTO region (region_code, name)
SELECT DISTINCT region_code, region
FROM temp_disasters
WHERE region_code IS NOT NULL;
```

- region_code : Code unique associé à chaque région.
- name : Nom de la région.

b. Table sub_region

Nous avons inséré les informations des sous-régions à partir de la colonne sub_region_name, en associant chaque sous-région à sa région correspondante via son code et en éliminant les doublons de la colonne sub_region_code:

```
INSERT INTO sub_region (sub_region_code, name, region_code)
SELECT DISTINCT sub_region_code, sub_region, region_code
FROM temp_disasters
WHERE sub_region_code IS NOT NULL;
```

- sub_region_code : Code unique pour chaque sous-région.
- name : Nom de la sous-région.
- region_code : Code faisant référence à la région correspondante.

c. Table disaster

Les types de catastrophes ont été ajoutés à la table disaster :

```
INSERT INTO disaster (disaster_code, name)
SELECT DISTINCT disaster
FROM temp_disasters;
WHERE disaster IS NOT NULL;
```

- disaster_code : Code unique pour chaque type de catastrophe.

• name : Nom du type de catastrophe.

d. Table country

Les informations sur les pays ont été insérées dans la table country, en les associant à leur sous-région respective.

```
INSERT INTO country (name, iso2, iso3 sub_region_code)
SELECT DISTINCT disaster_code, country_name, disaster_code
FROM temp_disasters
WHERE sub_region_code IS NOT NULL AND iso2 IS NOT NULL;
```

• country_code : Code unique pour chaque pays.
• name : Nom du pays.
• sub_region_code : Référence à la sous-région correspondante.

e. Table climate_disaster

Pour finir, les informations détaillées sur les catastrophes, incluant les dates, ont été insérées dans la table climate_disaster, en établissant un lien entre les pays et les types de catastrophes.

```
INSERT INTO climate_disaster (country_code, disaster_code, year,
number)
SELECT c.country_code, d.disaster_code, t.year, t.number
FROM temp_disasters t
JOIN country c ON t.country = c.name
JOIN disaster d ON t.disaster = d.disaster;
```

• country_code : Code faisant référence au pays affecté.
• disaster_code : Code faisant référence au type de catastrophe.
• year : Année où la catastrophe a eu lieu.

Explications

1. FROM temp_disasters t : Cela déclare la table temp_disasters et lui attribue l'alias t, que vous utilisez dans la requête.

2. JOIN country c : Lie la table country à temp_disasters en comparant le nom du pays (t.country = c.name).

3. JOIN disaster d : Lie la table disaster à temp_disasters en comparant le type de désastre (t.disaster = d.disaster).

4. Suppression de la table temporaire

Après l'insertion des données dans les tables principales, la table temporaire temp_disaster a été supprimée :

```
DROP TABLE temp_disasters;
```