

## Sommaire

---

### 1. Modélisation et script de création " avec AGL"

- a) Illustrations comparatives cours/AGL commentées d'une association fonctionnelle.
- b) Illustrations comparatives cours/AGL commentées d'une association maillée.
- c) Modèle physique de données réalisé avec l'AGL.
- d) Script SQL de création des tables généré automatiquement par l'AGL.
- e) Discussion sur les différences entre les scripts produits manuellement et automatiquement.

### 2. Peuplement des tables

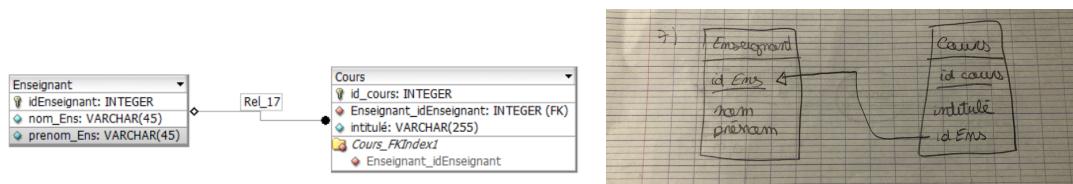
- a) Script manuel des créations des tables.
- b) Description commentée des différentes étapes de votre script de peuplement.

•

---

### 1. Modélisation et script de création "avec AGL"

#### a) Illustrations comparatives cours/AGL commentées d'une association fonctionnelle.



on retrouve bien la clé étrangère de la table "Enseignant" dans la table 'Cours', signalée par la notation 'FK', qui signifie 'foreign key'. Une association fonctionnelle ajoute effectivement une clé étrangère à la table ayant une cardinalité maximale de 1, uniquement en tant que clé étrangère. on constate rapidement que la modélisation sur un Atelier de Génie Logiciel (AGL) fournit beaucoup plus d'informations quant au type de données, information que l'on n'écrit pas forcément dans nos modélisations manuelles. Les symboles indiquent également les types de colonnes; par exemple, la clé primaire est représentée par une clé, tandis que sur papier, nous soulignons pour indiquer une clé primaire.

En ce qui concerne les relations, il est possible d'ajouter un nom à celles-ci sur l'AGL. À l'inverse, sur papier, nous ne spécifions généralement pas le nom de la relation lors de la représentation d'un schéma relationnel. Les numéros entre parenthèses signalent le nombre maximal de caractères à entrer. Pour l'intitulé de la table 'Cours', nous aurions effectivement pu simplement indiquer (45).

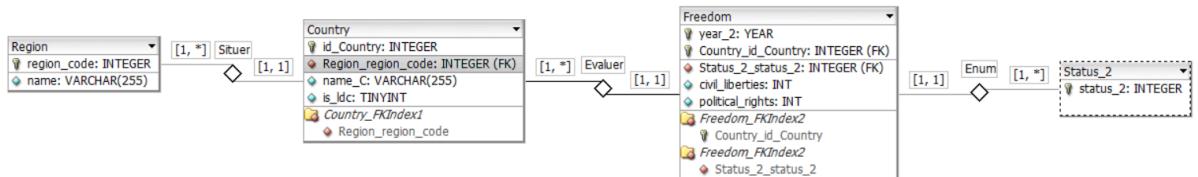
b) Illustrations comparatives cours/AGL commentées d'une association maillée.



En comparant les deux formats, AGL et papier, on observe clairement la création d'une nouvelle table qui était initialement une relation, caractéristique d'une association maillée. Dans cette dernière, on retrouve les deux clés primaires provenant respectivement des deux tables en relation, à savoir 'Personne' et 'Film'.

Les clés primaires 'id\_Personne' et 'id\_Film' deviennent ainsi les clés primaires et étrangères de la table 'Jouer'. Sur papier, cette information est généralement représentée par le soulignement de la colonne en question, tandis que sur l'AGL, elle est symbolisée par l'icône de clé. Dans notre cas, nous observons bien le symbole de clé accompagné du 'FK', signifiant, comme mentionné précédemment, 'Foreign Key'. De plus, des flèches indiquent la provenance des clés étrangères.

c) Modèle physique de données réalisé avec l'AGL.



d) Script SQL de création des tables générée automatiquement par l'AGL.

```

CREATE TABLE Region (
    region_code INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(255) NULL,
    PRIMARY KEY (region_code)
);

CREATE TABLE Country (
    id_Country INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
    Region_region_code INTEGER UNSIGNED NOT NULL,
    name_C VARCHAR(255) NULL,
    is_ldc TINYINT UNSIGNED NULL,
    PRIMARY KEY (id_Country),
    INDEX Country_FKIndex1 (Region_region_code)
);

```

Vongsavanh Kevin  
BUT 1 EUROS

```
CREATE TABLE Status_2 (
    status_2 INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
    PRIMARY KEY(status_2)
) ;

CREATE TABLE Freedom (
    year_2 YEAR NOT NULL,
    Country_id_Country INTEGER UNSIGNED NOT NULL,
    Status_2_status_2 INTEGER UNSIGNED NOT NULL,
    civil_liberties INT NOT NULL,
    political_rights INT NOT NULL,
    PRIMARY KEY(year_2, Country_id_Country),
    INDEX Freedom_FKIndex2(Country_id_Country),
    INDEX Freedom_FKIndex2(Status_2_status_2)
) ;
```

- e) Discussion sur les différences entre les scripts produits manuellement et automatiquement.

L'utilisation de scripts générés automatiquement par un AGL offre plusieurs avantages significatifs par rapport à l'approche manuelle de codage. Tout d'abord, l'automatisation via un AGL accélère le processus de développement en réduisant considérablement le temps nécessaire à la création du code. Cela permet aux développeurs de se concentrer davantage sur la conception et la logique métier, en augmentant ainsi l'efficacité globale du cycle de développement.

De plus, les scripts générés automatiquement permettent de réduire les erreurs humaines, améliorant ainsi la qualité du code. Les AGL suivent des conventions prédéfinies et appliquent des bonnes pratiques, minimisant ainsi les risques d'incohérence ou de bugs résultant de la saisie manuelle. Cela conduit à des logiciels plus fiables et moins sujets aux erreurs.

D'un autre côté, le script manuel présente des avantages en termes de flexibilité et de contrôle. Les développeurs ont la possibilité d'ajuster le code en fonction de leur compréhension approfondie du système, ce qui peut être crucial dans des situations complexes ou pour répondre à des exigences spécifiques. Le codage manuel permet également une meilleure adaptation aux styles de programmation préférés, offrant ainsi une personnalisation plus approfondie.

En résumé, l'utilisation de scripts générés automatiquement via un AGL optimise l'efficacité et minimise les erreurs, tandis que le codage manuel offre une meilleure flexibilité et un contrôle plus large.

## **2. Peuplement des tables**

- a) Script manuel des créations des tables.

```
CREATE TABLE region (
    region_code INTEGER PRIMARY KEY,
    nameR VARCHAR(255) NOT NULL
) ;

CREATE TABLE country (
    id_country SERIAL PRIMARY KEY,
    nameC VARCHAR(255) NOT NULL,
    region_code integer references region,
    is_ldc BOOLEAN
) ;

CREATE TABLE STATUS (
    status VARCHAR(255) PRIMARY KEY
) ;
CREATE TABLE freedom (
    id_country int,
    year year ,
    status varchar(255),
    civil_liberties  INTEGER CHECK(civil_liberties > 0 AND
civil_liberties < 8),
    political_rights INTEGER CHECK(political_rights > 0 AND
political_rights < 8),
) ;

CREATE TABLE freedom (
    year INTEGER PRIMARY KEY,
    id_country INTEGER REFERENCES country,
    status VARCHAR(255) REFERENCES STATUS,
    civil_liberties  INTEGER CHECK(civil_liberties > 0 AND
civil_liberties < 8),
    political_rights INTEGER CHECK(political_rights > 0 AND
political_rights < 8)
) ;
```

b) Description commentée des différentes étapes de votre script de peuplement.

Pour commencer cette partie, j'ai d'abord divisé le fichier csv en plusieurs parties pour pouvoir utiliser la commande insert. J'ai donc obtenu les fichiers suivants: freedom\_country.csv; freedom\_status.csv; freedom\_region.csv et freedom\_freedom.csv. J'ai donc dans ces fichiers la plupart des colonnes nécessaires au peuplement des ces tables. J'ai aussi utilisé une table temporaire afin de pouvoir utiliser la commande copy pour chaque table. L'insertion des données doit aussi être faite dans un ordre précis, on commence par peupler les tables sans clés étrangères, j'ai alors commencé par peupler la table "status".

```
CREATE TEMPORARY TABLE temp_utilisateurs (
    status VARCHAR(255),
) ;

COPY temp_utilisateurs FROM
'/Users/vongk/oneDrive/Bureau/sae_sql/freedom_status.csv'
DELIMITER E'\t' CSV HEADER;

INSERT INTO status
SELECT DISTINCT status
FROM temp_utilisateurs;
```

on ajoute d'ailleurs DELIMITER E'\t' indique que le fichier doit être interprété comme un fichier CSV avec des tabulations comme délimiteur et CSV HEADER indique que la première ligne du fichier CSV contient les en-têtes de colonne et qu'il ne faut pas copier cette ligne. De plus le SELECT DISTINCT permet d'éviter les répétitions pour ne pas violer la règle d'une clé primaire (valeur unique).

```
CREATE TEMPORARY TABLE temp_utilisateurs (
    region_code INTEGER,
    nameR VARCHAR(255) NOT NULL
) ;

COPY temp_utilisateurs FROM
'/Users/vongk/oneDrive/Bureau/sae_sql/freedom_region.csv'
DELIMITER E'\t' CSV HEADER;

INSERT INTO region
SELECT DISTINCT region_code, nameR
FROM temp_utilisateurs;
```

Pour la table 'country' sont id\_country n'est pas présent dans le fichier csv, pour cela nous allons donner le type SERIAL, ce type va permettre à cette colonne de créer automatiquement une valeur pour chaque ligne insérée à partir du fichier. J'ai seulement inséré les valeurs connues.

Vongsavanh Kevin  
BUT 1 EUROS

```
CREATE TEMPORARY TABLE temp_utilisateurs1 (
    nameC VARCHAR(255) NOT NULL,
    is_ldc BOOLEAN
);

COPY temp_utilisateurs FROM
'/Users/vongk/oneDrive/Bureau/sae_sql/freedom_country.csv'
DELIMITER E'\t' CSV HEADER;

INSERT INTO country (namec, region_code, is_ldc)
SELECT DISTINCT nameC, region_code, is_ldc
FROM temp_utilisateurs1;
```

Pour la table 'Freedom' on utilisera une approche différente afin de pouvoir insérer la clé étrangère 'id\_country'. On utilisera la requête SELECT qui utilise une jointure 'JOIN' entre temp\_utilisateurs1 et country pour obtenir l'identifiant du pays 'id\_country' à partir de la table 'country'.

```
CREATE TEMPORARY TABLE temp_utilisateurs1 (
    namec varchar(255),
    year int,
    status varchar(255),
    civil_liberties int,
    political_rights int
);

COPY temp_utilisateurs1 FROM
'/Users/vongk/oneDrive/Bureau/sae_sql/freedom_freedom.csv'
DELIMITER E'\t' CSV HEADER;

insert into freedom (id_country, year, status, civil_liberties,
political_rights)
select country.id_country, year, status, civil_liberties,
political_rights
from temp_utilisateurs1 join country on country.namec =
temp_utilisateurs1.namec;
```