

RAPPORT

BASES DE DONNÉES AVANCÉES

Joris IDJELLIDINE
Lucian PETIC

15.05.2021
M1 - INFO - BDA

1. SUJET DE NOTRE PROJET

Notre modélisation se penchera principalement sur les différents chiffres de la pandémie du Covid-19 à l'échelle mondiale sur le temps et par pays. On aura notamment des données sur les vaccinations, les tests, le nombre de morts, le nombre de vols ainsi que l'impact économique et écologique dans chaque pays.

2. SCHÉMA ENTITÉ/RELATION

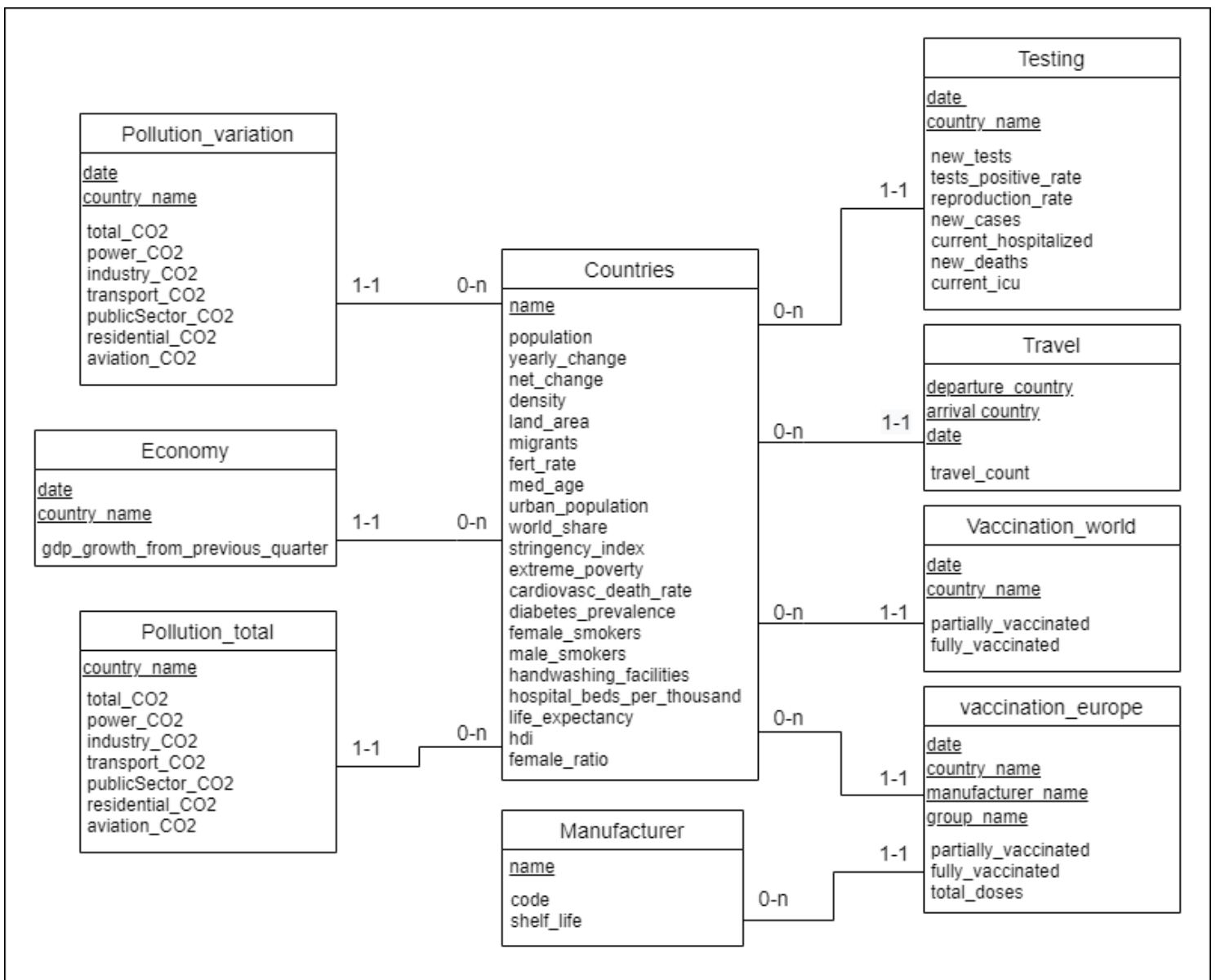


SCHÉMA RELATIONNEL

countries(name, population, yearly_change, net_change, density, land_area, migrants , fert_rate, med_age , urban_population, world_share, stringency_index, extreme_poverty, cardiovasc_death_rate, diabetes_prevalence, female_smokers, male_smokers, handwashing_facilities, hospital_beds_per_thousand, life_expectancy, hdi, female_ratio)

manufacturer(name, code, shelf_life)

vaccination_europe(date, country_name, manufacturer_name, group_name, partially_vaccinated, fully_vaccinated, total_doses)

vaccination_europe[country] \subseteq countries[name]

vaccination_europe[manufacturer_name] \subseteq manufacturer[name]

vaccination_world(date, country_name, partially_vaccinated, fully_vaccinated)

vaccination_world[country_name] \subseteq countries[name]

testing(date, country_name, new_tests, tests_positive_rate, reproduction_rate, new_cases, current_hospitalized, new_deaths, current_icu)

testing[country_name] \subseteq countries[name]

economy(date, country_name, gdp_growth_from_previous_quarter)

economy[country_name] \subseteq countries[name]

pollution_variation(date, country_name, total_CO2, power_CO2, industry_CO2, transport_CO2, publicSector_CO2, residential_CO2, aviation_CO2)

pollution_variation[country_name] \subseteq countries[name]

pollution_total(country_name, total_CO2, power_CO2, industry_CO2, transport_CO2, publicSector_CO2, residential_CO2, aviation_CO2)

pollution_total[country_name] \subseteq countries[name]

De plus, toutes les valeurs essentielles sont dotés de la contrainte NOT NULL, et les valeurs étant dans une limite donnée sont dotés de CHECK(condition), avec principalement la condition "valeur ≥ 0 ".

3. MODÉLISATION ET ARCHITECTURE

Étant donné que notre modélisation se porte principalement à l'échelle mondiale, la plupart des tables de notre base de données sont centrées autour de la table Countries.

Cette liaison permet de faire des comparaisons entre plusieurs pays et de déterminer comment ces pays s'en sortent face à la pandémie. Sachant qu'il n'existe que très peu de dépendances fonctionnelles à cause de la nature des données, toutes les dépendances fonctionnelles existantes sont des clés primaires.

Les index placés sont les suivants :

total_CO2_index : Pollution_variation(total_CO2) BTREE

total_CO2_index : Pollution_total(total_CO2) BTREE

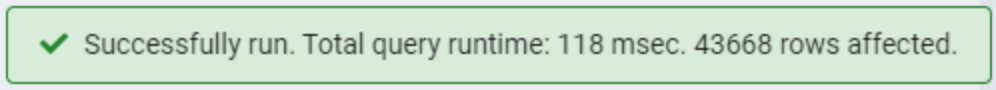
travel_count_index : Travel(travel_count) BTREE

population_index : Countries(population) BTREE

Grâce à ces index, on peut s'attendre à un gros gain de temps lors de l'exécution de requêtes impliquant uniquement ces attributs. Par exemple, pour la requête suivante :

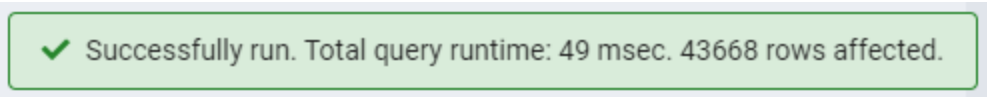
```
SELECT travel_count FROM travel WHERE travel_count > 50;
```

On passe d'un temps d'exécution sans index de :



✓ Successfully run. Total query runtime: 118 msec. 43668 rows affected.

A un temps d'exécution avec index BTREE :



✓ Successfully run. Total query runtime: 49 msec. 43668 rows affected.

Aucun index HASH n'a été utilisé car notre base de données ne comporte pas d'attributs où un critère d'égalité serait souvent utilisé sur un attribut non clé primaire.

4. INSTALLATION

La procédure d'installation est disponible dans le `README.md` à la racine du projet.

Le processus lors de l'installation lors du lancement de *make install* est le suivant :

Exécution du fichier `processing.py` : En utilisant la librairie `pandas`, ce fichier traite les fichiers sources au format `csv` et produit une version `-pro` où les valeurs inutiles, erronées ou invalides sont corrigées. Ce traitement effectue aussi la liaison de multiples fichiers sources afin de former le fichier souhaité. Par exemple, `flights_merged-pro`, dont les fichiers sources initiaux font un total de 6.4Go, utilisait comme fichiers sources `flightlist`, `airports.csv` et le fichier traité `countries_joined-pro.csv` pour former son produit final. Dans un souci d'espace de stockage, `flightlist.csv` a été remplacé par sa contrepartie déjà traitée et le code source utilisé pour son traitement a été laissé en commentaire.

`create_all.sql` contient le code nécessaire à la création de la base de données, ainsi que tous les `CHECK` et `NOT NULL` applicables et la création des indexs.

`create_triggers.sql` contient le code nécessaire à la création des triggers sur les tables.

`insert_data.sql` contient le code nécessaire pour copier les valeurs des fichiers `.csv` traités dans la base de données.

`queries.sql` contient le code nécessaire à la création des fonctions de la base de données.

5. FONCTIONS

Liste des fonctions :

CumulativeTesting :

Renvoie le cumul de la colonne *toSearch* dans la table *testing* entre les dates *date_lower* et *date_upper* pour le pays *country*

cumulativeTesting(toSearch TEXT, date_lower DATE = NULL, date_upper DATE = NULL, country TEXT = NULL)
-> (date date, vals DECIMAL)

Exemple : SELECT * FROM cumulativeTesting('new_cases','2021-04-10','2021-06-10','France');

CumulativeVaccinationEurope :

Renvoie le cumul de la colonne *toSearch* dans la table *vaccination_europe* entre les dates *date_lower* et *date_upper* pour le pays *country* et pour une entreprise *manufacturer_name* et un groupe *group_name*

cumulativeVaccinationEurope(toSearch TEXT, date_lower DATE = NULL, date_upper DATE = NULL, country TEXT = NULL, manufacturer_name TEXT = NULL, group_name TEXT = NULL)
-> (date date, vals DECIMAL)

Exemple : SELECT * FROM
cumulativeVaccinationEurope('total_doses','2021-04-10','2021-06-10','France','COM','ALL');

CumulativeVaccinationWorld :

Renvoie le cumul de la colonne *toSearch* dans la table *vaccination_world* entre les dates *date_lower* et *date_upper* pour le pays *country*

cumulativeVaccinationWorld(toSearch TEXT, date_lower DATE = NULL, date_upper DATE = NULL, country TEXT = NULL)
-> (date date, vals DECIMAL)

Exemple :
SELECT * FROM cumulativeVaccinationWorld('partially_vaccinated','2021-04-10','2021-06-10','United Kingdom');

CumulativeTravel :

Renvoie le cumul de la colonne *travel_count* dans la table *travel* entre les dates *date_lower* et *date_upper* pour un pays de départ *departure_country* et un pays d'arrivée *arrival_country*

cumulativeTravel(date_lower DATE = NULL, date_upper DATE = NULL, departure_country TEXT = NULL, arrival_country TEXT = NULL)
-> (date date, vals DECIMAL)

Exemple : SELECT * FROM cumulativeTravel('2021-04-10','2021-06-10','France','Italy');

CumulativePollution :

Renvoie le cumul de la colonne *toSearch* dans la table *pollution_variation* entre les dates *date_lower* et *date_upper* pour le pays *country*

cumulativePollution(toSearch TEXT, date_lower date = NULL, date_upper DATE = NULL, country TEXT = NULL)
-> (date date, vals DECIMAL)

Exemple : SELECT * FROM cumulativePollution('total_CO2','2020-01-10','2021-10-10','France');

InsertTesting :

Ajoute dans la table *testing* les valeur non nulles passées en argument, échoue si le pays passé en argument est invalide

InsertTesting(date DATE, country_name TEXT, new_tests DECIMAL, tests_positive_rate DECIMAL, reproduction_rate DECIMAL, new_cases DECIMAL, current_hospitalized DECIMAL, new_deaths DECIMAL, current_icu DECIMAL)
-> void

Exemple : SELECT * FROM insertTesting('2021-05-15','France',1,2,3);
SELECT * FROM testing WHERE date = '2021-05-15';

InsertVaccinationEurope :

Ajoute dans la table *vaccination_europe* les valeur non nulles passées en argument, échoue si le pays ou le l'entreprise passé en argument est invalide

InsertVaccinationEurope(date DATE, country_name TEXT, manufacturer_name TEXT, group_name TEXT, partially_vaccinated INT = 0, fully_vaccinated INT = 0, total_doses INT = 0)
-> void

Exempe : SELECT * FROM insertVaccinationEurope('2021-05-15','France','AZ','ALL',1,2);
SELECT * FROM vaccination_europe WHERE date = '2021-05-15';

6. TRIGGERS

Liste des triggers :

Transfer_travel :

Après l'ajout d'une ligne à travel, transfère *travel_count* tests avec une variance de $\pm 20\%$ de *departure_country* à *arrival_country* si *departure_country* possède assez de tests pour faire le transfert et que date $\geq 2020-04-01$.

Countries_female_ratio_default :

Avant l'ajout d'une ligne à countries, mets la valeur de l'attribut *female_ratio* à 50 si jamais cette valeur était nulle de base.

Block_update :

Avant une mise à jour dans *vaccination_europe* ou *vaccination_world*, bloque cette mise à jour si la date de la ligne modifiée date d'il y a plus de 15 jours.

7. TESTS

Pour appeler un test, il faut utiliser la commande suivante :

```
make check ARGS='path_to_test.sql'
```

Par exemple, pour appeler le fichier SQL testant les fonctions de cumuls, la commande sera la suivante :

```
make check ARGS='test/functions/TestCumulative.sql'
```

Tous les tests affichent des informations complémentaires sur leurs déroulement afin de démontrer leurs bon fonctionnements

Liste des tests :

```
make check ARGS='test/functions/TestCumulative.sql'
# Teste toutes les fonctions de cumul
```

```
make check ARGS='test/functions/TestInsertVaccinationEurope.sql'
# Teste les fonctions insertTesting et insertVaccinationEurope
```

```
make check ARGS='test/queries/CompareTravelPollution.sql'
# Prends en entrée un pays et envoie une requête sélectionnant le
nombre de vols qui quittent le pays et la variance de la pollution
```

```
make check ARGS='test/triggers/block_update.sql'
# Effectue des mises à jours dans vaccination_europe afin de tester
le bon fonctionnement du trigger block_update
```

```
make check ARGS='test/triggers/transfer_travel.sql'
# Ajoute une ligne à travel et affiche les tests pour le pays et la
date utilisé afin de tester le bon fonctionnement du trigger
transfer_travel
```

8. GRAPHES

Etant donné qu'il n'est pas nécessairement facile de faire le lien entre les tables de notre base de donnée et leurs lien avec la pandémie du COVID-19 en utilisant simplement l'affichage offert par le terminal `psql` ou `PgAdmin4`, des scripts python montrant les corrélations entre les ajouts de notre base de données sont mis à disposition dans le dossier *graph*. Pour les utiliser, il suffit de les lancer comme un fichier python traditionnel, ou bien de les appeler en utilisant le Makefile avec la commande suivante :

```
make gr ARGS='path_to_graph.py'
```

Un fichier au format png sera ensuite généré avec le graphe correspondant au pays donné en argument. Par exemple, `PollutionTravelCorrelation.py` avec le pays France passé en argument donne le graphe suivant :



Des graphes pré-fait sont disponibles dans le dossier docs du projet.

9. SOURCES ET LIENS

delivery.csv : [COVID-19 Vaccine rollout overview](#)

Pollution_per_day.csv: [CO2 emissions during the COVID-19 forced confinement](#)

Pollution_total.csv: [CO2 emissions during the COVID-19 forced confinement](#)

flightlist.csv: [OpenSky COVID-19 Flight Dataset Featured](#)

airports.csv: [OpenSky COVID-19 Flight Dataset Featured](#)

GDP Growth.csv: [Quarterly Growth Rates of real GDP](#)

owid-covid-data.csv: [Our World in Data](#)

countries.csv: [Population by Country - 2020](#)

share-population-females.csv : [Our World in Data – Gender Ratio](#)

<https://www.ecdc.europa.eu/en/covid-19/vaccine-roll-out-overview>

<https://www.icos-cp.eu/gcp-covid19>

[OpenSky COVID-19 Flight Dataset Featured](#)

<https://stats.oecd.org/index.aspx?queryid=350>

[Our World in Data](#)

<https://www.kaggle.com/tanuprabhu/population-by-country-2020>

<https://ourworldindata.org/gender-ratio>