BDLE projet 1

Barème et éléments de solution

Barème

Tâche 1 8

1 par relation et extraction avec expressions régulières ou équivalent. Explication des traitements effectués. Robustesse de la solution face aux irrégularités de format (exple date avec jour manquant).

Tâche 2 8

Etoile : Fait + dim : 3

Dim hiérachique 3

Requetes rollup,cube: 1

Requetes over 1

Tâche3 3

Autre fichier 1

Jointure avec données du projet 2

Tâche 4 : 1

Eléments de solution

Des éléments ci-dessous reprennent des extraits de solutions de certains étudiants.

Tâche 1

Ajout d'un attribut optionnel qui n'est pas toujours renseigné (death_date) : penser à la jointure EXTERNE : le résultat contient tous les tuples, y compris ceux qui n'ont pas de valeur pour l'attribut optionnel.

Exemple : ajouter la date de décès, le cas échéant, mais en gardant aussi les personnes vivantes.

```
select p1.id,p1.name, p1.value as birth_date,p2.value as death_date
from NameDetail p1 left outer join NameDetail p2 on (p1.id=p2.id  and p2.property="death date")
where p1.property="birth date"
```

Version du 17/11/2020 12:02

Extraction dans une chaine : expressions régulières

Extraire un nombre à 4 chiffres:

```
cast(regexp_extract(value,'([0-9][0-9][0-9][0-9])',1)as int) as year
```

Extraire un mot placé entre parenthèses, utilisation de caractères d'échapement

```
regexp_extract(value,'\\(  (.*?) \\)',1)
```

Extraire un mot placé entre guillemets,

```
regexp_replace(regexp_extract(value,'("(.*?)" )',1),"","")
```

Extraire la date d'une chaine pouvant contenir optionnellement le mois ou le jour

```
if(trim(cardinality(split(fin_tournage, ' '))) < 4, NULL, split(fin_tournage, ' ')[1]) as Jour_fin,
if(cardinality(split(fin_tournage, ' ')) < 2, NULL, if(cardinality(split(fin_tournage, ' ')) = 4,
split(fin_tournage, ' ')[2], split(fin_tournage, ' ')[1])) as Mois_fin,
right(fin_tournage, 5) as Annee_fin
```

Expression régulière avec plusieurs groupes

```
select id, title,
  regexp_extract(trim(valeur), "^(\\D+)([\\d,]+)$", 2) as budget,
  regexp_extract(trim(valeur), "^(\\D+)([\\d,]+)$", 1) as currency
from DetailFilm
where attribut = "budget"
```

Fonction UDF

Exemple de fonction python (pas encore UDF !) pour extraire des valeurs monétaires avec devise.

```
def get_currency_amount(s):
  """
  Conversion de la chaîne de caractères en deux attributs `symbole,`montant`.
  """
  for symbol in currencies.keys():
    if symbol in s:
      s = s.replace(symbol, '')
      for c in ' ,.;/:':
        s = s.replace(c, '')
      if not s.isnumeric():
        return (None, None)
      return (symbol, float(s))
  return (None, None)
```

Définir une UDF à partir de cette fonction : 2 possibilités

UDF pour SQL

spark.udf.**register**('get_currency_amount ', get_currency_amount)

Cette UDF est utilisée dans

```
%sql
Select get_currency_amount(budget)
From titleDetail
```

UDF pour Dataframe

```
schema = StructType([  StructField("currency", StringType(), True), StructField("amount", FloatType(), True)])
get_currency_amount_udf = udf(get_currency_amount, schema)
```

```
res1 = titleDetail.select(get_currency_amount_udf("budget"))

# retirer les films pour lesquels l'UDF n'a pas permis d'extraire le budget
res2 = res1.dropna(how='any')
```

Nommer une requête

Une expression **with** permet de nommer une liste de requêtes :

With T as (select …. From…), U as (select … from T…) select … from U.

Tâche 2

Mesure d'un fait vs. Attribut d'une dimension

Confusion entre attribut de dimension / attribut représentant une mesure d'un fait. Une mesure est déterminée par la clé (composée des clés des dimensions) du fait. Donc un attribut considéré comme une mesure n'est pas déterminé par la clé d'une seule dimension.

Identifiant de dimension vs. niveau d'une dimension

Faire la distinction entre un attribut qui représente la dimension d'un Fait et un attribut qui représente un niveau d'une dimension.

Exple avec le fait Joue(person,title,role,date,pays,salaire) les dimensions sont: name,title, role, mais **pas** date, pays :

la date est un attribut de Film mais pas une dimension de Joue

idem pour le pays, sauf si le pays représente celui de la scène jouée.

Un Fait correct serait **Joue** (person,title,role, salaire) avec 3 dimensions et une mesure.

Autre exple, ne pas mettre l'attribut genre dans la table de Faits représentant les castings car un film est associé à plusieurs genres, mais définir un Fait (film, genre) sans mesure et avec 2 dimensions. Permet d'analyser le nombre de films par année de production, par genre, etc…

Autre exemple de Fait : Revenu (film, date, revenu)

Niveau plus général vs résumé de la valeur d'un attribut

Notion de niveau dans une dimension mal comprise : L'attribut *Trivia* n'est pas un niveau plus général que *biography* car on ne peut pas regrouper davantage de personnes avec cet attribut. Idem, *tagline* n'est pas un niveau au-dessus de l'attribut *plot* pour Film.

Niveau composé de plusieurs attributs

Certains niveaux sont définis en composant plusieurs attributs. Exple le niveau trimestriel est défini par le couple (année, trimestre) avec trimestre dans [1,4]. L'attribut trimestre seul ne détermine pas un niveau. En revanche l'attribut trimestreID dans [T1, …..Tn] (avec n = 4* le nombre d'année) détermine à lui seul le niveau trimestriel.

Schéma en flocon

une dimension de la table de faits peut se décliner en plusieurs dimensions.

Exemple Fait(pers, film, role, mesure) pour la dimension Film il peut y avoir une première dimension sur le budget du film Budget(film, budget, catégorie). On peut scinder le domaine du budget en plusieurs intervalles consécutifs et la catégorie représente un intervalle (petit, moyen, gros budget). Une deuxième dimension peut être sur l'année de production, avec le domaine des années scindé en décades.

Analyse: GROUP BY vs. OVER avec partition by

Une analyse qui agrège seulement des données selon un critère de regroupement n'a pas besoin de fenêtre.
Exple la table Revenu(titleID, kind, type, year, revenu)
et la requête Revenu moyen par année pour chaque genre et type d'œuvre. Pas besoin de OVER pour cette requête :

```
select kind, genre, year, avg(revenu)
from Revenu
GROUP BY kind, genre, year
```

En revanche un OVER serait nécessaire pour connaitre *l'évolution relative* du revenu annuel pour chaque genre et type d'œuvre…

Cube vs Rollup

Exple avec le fait Sortie(film, dateProj, pays, nbEntrées)

Group by rollup(film, dateProj, pays) : les attributs ne sont pas les niveaux d'une même dimension un group by **cube**(film, dateProj, pays) est plus pertinent pour des attributs représentant des dimensions différentes.

OVER

Dans une fenêtre exprimée avec OVER, préciser le critère de tri des tuples de la fenêtre avec

```
over(order by …)
```

Idem pour une fenêtre partitionnée, il est préférable de préciser le critère de tri

```
over(partition by …. order by …)
```

Tâche 3

Ne pas confondre l'union et la jointure. Pour relier une table extérieure avec une table du projet, il faut faire une jointure. Exple jointure sur le titre et l'année du film pour ajouter un attribut à un film, sur le nom complet d'une personne pour ajouter attribut à une personne. Se poser la question de la pertinence du résultat obtenu.

Attention aux jointures sur identifiant. Les identifiants sont souvent locaux à une base. L'identifiant dont le préfixe est tt ne correspond pas aux identifiants des films du projet.

Tenir compte des csv qui ont une première ligne d'entête : spark.read.option("header",True).csv(…)

Fichiers TSV avec un séparateur TAB :

```
schema_yago = "id STRING,src String,link STRING, dest STRING"
yagoFacts = spark.read.csv(path = dir + "yagoFacts.tsv", schema = schema_yago, sep="\t",
header=True).persist()
```

Données de Wikidata

Exemple : Films.csv avec info de popularité

Requête en invoquant manuellement le service https://query.wikidata.org/ et téléchargeant le résultat dans un CSV.

La propriété wdt:P31 signifie « instance of » permet de préciser le type ou la classe des éléments recherchés. Ici, on cherche les films dont le type est représenté par la classe (wd:Q11424). On cherche aussi à récupérer des informations sur la dénomination de la récompense ainsi que l'événement au cours duquel la récompense a été obtenue. L'instruction SERVICE permet d'afficher les labels associés à chaque variable de la requête. Le label représenté *?awardLabel* est celui de l'entité représentée par *?award,* idem pour les 2 autres labels. Tous les labels sont en anglais.

```
SELECT ?awardWorkLabel ?awardLabel ?awardEditionLabel
WHERE {
    ?awardWork wdt:P31 wd:Q11424 .
    ?awardWork p:P166 ?awardStat .
    ?awardStat ps:P166 ?award .
    ?awardStat pq:P805 ?awardEdition .
 SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE], en". }
}
```

Données de Wikipedia

Invocation **par API** d'un SPARQL endpoint et résultat dans un dataframe pandas, qu'il faut ensuite convertir en un dataframe spark. Cette solution est très intéressante et montre deux limitations : (i) elle ne fonctionnera pas avec des données volumineuses dont la traille dépasse la taille maximale d'un dataframe pandas. (ii)elle peut s'avérer trop lente car le résultat de la requête est lu séquentiellement.

```
!pip install rdflib
!pip install SPARQLWrapper

from SPARQLWrapper import SPARQLWrapper, JSON
```

```
import json
import pandas as pd
def execute_query(query):
  """lance une requete sparql et retoune sous format JSON"""
  sparql = SPARQLWrapper("http://dbpedia.org/sparql")
  sparql.setReturnFormat(JSON)
  sparql.setQuery(query)
  return sparql.query().convert()

def query_to_df(query,elt,property):
  result_dict = execute_query(query)
  dictionary = [{elt : item[elt]["value"], property : item[property]["value"]} for item in
result_dict["results"]["bindings"]]
  return pd.DataFrame(dictionary)
# par exemple recuperer de dbpedia les genres des films
query = """
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX film: <http://dbpedia.org/ontology/Film>
PREFIX genre: <http://dbpedia.org/ontology/genre>

SELECT ?movielab, ?genrelab
WHERE {
?movie a film: .
?movie genre: ?genre .
?movie rdfs:label ?movielab .
?genre rdfs:label ?genrelab
FILTER (langMatches( lang(?movielab), "EN" ) and langMatches( lang(?genrelab), "EN" ))
}"""
query_to_df(query, "movielab","genrelab")
```

Ajout de offset et limit à une requête SPARQL pour paginer le résultat.

Autre solution pour l'accès à WIKIPEDIA

%sh
pip install wikipedia
puis

```
import wikipedia

def get_wikipedia_summary(actor_name):
  try:
    return str(wikipedia.page(wikipedia.search(actor_name+" actor")[0]).summary)
  except :
    return ""

 spark.udf.register('get_wikipedia_summary', get_wikipedia_summary, pyspark.sql.types.StringType())
```

puis

```
%sql
SELECT id, get_wikipedia_summary(name) as wikipedia_summary FROM NameDetail limit 5
```

Version du 17/11/2020 12:02

Données de yago

```
select REPLACE(regexp_extract(y.sujet,'^<(.*)>$',1),'_',' ') as name,
regexp_extract(y.prop,'^<(.*)>$',1) as property,
REPLACE(regexp_extract(y.objet,'^<(.*)>$',1),'_',' ') as value
from yago y;
```

DIVERS

Installer des lib python auxiliaires

```
# l'installation de plotly pour la visualisation
%pip install plotly
import plotly.express as px
import plotly.graph_objects as go
```

wget depuis python

```
%pip install wget
```

Puis

dbutils.fs.cp('file:/databricks/driver/*nom_de_fichier_local* ' , 'dbfs:/FileStore')

wget peut être invoqué en bash

%sh wget *url_d_un_fichier* -O *nom_de_fichier_local*

Installer une librairie en utilisant l'utilitaire dbutils

```
dbutils.library.installPyPI("SPARQLWrapper")
dbutils.library.restartPython()
```

puis

```
from SPARQLWrapper import SPARQLWrapper, JSON
```