Traitement de données semi-structurées et optimisation logique

Master DAC – Bases de Données Large Echelle
Mohamed-Amine Baazizi
baazizi@ia.lip6.fr
2019-2020

Plan

- Interrogation données semi-structurées
- Optimisation logique

Données semi-structurées

- Caractéristiques
  - très répandues : crawl api, data-sets publiques, …
  - flexibilité, pas de schéma préétabli (schéma a posteriori)
  - imbrication sur plusieurs niveaux, structure variable
  - difficiles à manipuler et à cerner
- JSON : modèle le plus répondu
  - syntaxe plus simple que XML
  - exprime à la fois une séquence de valeurs (arrays) et des tuples (record)
- Quelques modèles plus expressifs utilisent :
  - Map (tableaux associatifs)
  - Bags (arrays sans la notion d'ordre)

JSON : modèle de données, schémas

- Data model
  - Valeurs atomiques : null, bool, number, string
  - records : ensemble de paires (clé,valeur)
  - arrays : séquence de valeurs
- Spécification du schéma
  - pas de standard, propre à chaque système
  - quelques similitudes et un candidate en lice (JSON-Schema) pour définir un schéma a priori
  - expressivité variable : utilisation d'expression régulière, opérateur d'union, comptage, négation

```
{
  "email" : "abc@ef",
  "first" : "li",
  "coord" : [{
    "lat" : 45,
    "long" : 12
            }],
  "last" : null
}
```

un objet JSON

Manipulation de JSON dans Spark

- Chargement
  - Traduction vers modèle Spark SQL guidée par le schéma
  - Schéma fourni ou inféré automatiquement
- Interrogation
  - Algèbre Dataset
  - Notation pointée pour naviguer dans la hiérarchie
  - Fonctions prédéfinie pour manipuler les arrays

Inférence du schéma et chargement

```
{
  "person" : {
    "firstname" : "Melena",
    "lastname" : "RYZIK",
    "role" : "reported",
    "rank" : 1,
    "organization" : ""
  }
}
{
  "person" : {
    "firstname" : "other",
    "lastname" : "ABCD",
    "rank" : 1,
    "organization" : "OO"
  }
}
```

Collection de 2 objets

```
testNyt: org.apache.spark.sql.DataFrame
scala> testNyt.printSchema
root
 |-- person: struct (nullable = true)
 |    |-- firstname: string (nullable = true)
 |    |-- lastname: string (nullable = true)
 |    |-- organization: string (nullable = true)
 |    |-- rank: long (nullable = true)
 |    |-- role: string (nullable = true)
```

role devrait être le seul attribut optionnel

```
scala> testNyt.show
```

person
[Melena, RYZIK, , 1, reported]
[other, ABCD, OO, 1,]

de type SparkSQL Struct

```
scala> testNyt.select("person.*").show
```

firstname	lastname	organization	rank	role
Melena	RYZIK		1	reported
other	ABCD	OO	1	null

Inférence du schéma et chargement

```
{
  "first" : "al",
  "coord" : [],
  "last" : "jr"
}
{
  "first" : "al",
  "coord" : null,
  "last" : "jr"
}
{
  "email" : "abc@ef",
  "first" : "li",
  "coord" : {
    "lat" : 45,
    "long" : 12
  },
  "last" : null
}
```

Collection de 3 objets

```
scala> test.printSchema
root
 |-- coord: string (nullable = true)
 |-- email: string (nullable = true)
 |-- first: string (nullable = true)
 |-- last: string (nullable = true)
```

de type SparkSQL String

```
scala> test.show
```

coord	email	first	last
[ ]	null	al	jr
null	null	al	jr
{"long":12,"lat":45}	abc@ef	li	null

impossible de récupérer long et lat sans parsing préalable

Inférence du schéma et chargement

```
{
  "person" : [
    {
      "firstname" : "Melena",
      "lastname" : "RYZIK",
      "role" : "reported",
      "rank" : 1,
      "organization" : ""
    },
    {
      "firstname" : "derba",
      "lastname" : "OKYZ",
      "role" : "reported",
      "rank" : 1,
      "organization" : ""
    }
  ]
}
{
  "person" : [
    {
      "firstname" : "other",
      "lastname" : "ABCD",
      "rank" : 1,
      "organization" : "OO"
    }
  ]
}
```

Collection de 2 objets

```
scala> testNyt.printSchema
root
 |-- person: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- firstname: string (nullable = true)
 |    |    |-- lastname: string (nullable = true)
 |    |    |-- organization: string (nullable = true)
 |    |    |-- rank: long (nullable = true)
 |    |    |-- role: string (nullable = true)

scala> testNyt.show(truncate=false)
```

de type SparkSQL Array<Struct>

person
[[Melena, RYZIK, , 1, reported], [derba, OKYZ, , 1, reported]]
[[other, ABCD, OO, 1,]]

Interrogation


```
{
  "person" : {
    "firstname" : "Melena",
    "lastname" : "RYZIK",
    "role" : "reported",
    "rank" : 1,
    "organization" : ""
  }
}
{
  "person" : {
    "firstname" : "other",
    "lastname" : "ABCD",
    "rank" : 1,
    "organization" : "OO"
  }
}
```


Collection de 2 objets

scala> testNyt.show

person
[Melena, RYZIK, , 1, reported]
[other, ABCD, OO, 1,]

scala> testNyt.select("person.*").show

firstname	lastname	organization	rank	role
Melena	RYZIK		1	reported
other	ABCD	OO	1	null

Utilisation de l'algèbre Dataset : select, where, join,…
https://spark.apache.org/docs/…org.apache.spark.sql.Dataset

Interrogation


```
{
  "person" : [
    {
      "firstname" : "Melena",
      "lastname" : "RYZIK",
      "role" : "reported",
      "rank" : 1,
      "organization" : ""
    },
    {
      "firstname" : "derba",
      "lastname" : "OKYZ",
      "role" : "reported",
      "rank" : 1,
      "organization" : ""
    }
  ]
}
{
  "person" : [
    {
      "firstname" : "other",
      "lastname" : "ABCD",
      "rank" : 1,
      "organization" : "OO"
    }
  ]
}
```


Collection de 2 objets

scala> testNyt.show(truncate=false)

person
[[Melena, RYZIK, , 1, reported], [derba, OKYZ, , 1, reported]]
[[other, ABCD, OO, 1,]]

scala> testNyt.select(explode_outer($"person")).show(truncate=false)

col
[Melena, RYZIK, , 1, reported]
[derba, OKYZ, , 1, reported]
[other, ABCD, OO, 1,]

explode dés-imbrique le contenu d'une colonne de type ArrayType<T> en retournant une colonne de type T

Interrogation


```
{
  "person" : [
    {
      "firstname" : "Melena",
      "lastname" : "RYZIK",
      "role" : "reported",
      "rank" : 1,
      "organization" : ""
    },
    {
      "firstname" : "derba",
      "lastname" : "OKYZ",
      "role" : "reported",
      "rank" : 1,
      "organization" : ""
    }
  ]
}
{
  "person" : [
    {
      "firstname" : "other",
      "lastname" : "ABCD",
      "rank" : 1,
      "organization" : "OO"
    }
  ]
}
```


Collection de 2 objets

scala> testNyt.show(truncate=false)

person
[[Melena, RYZIK, , 1, reported], [derba, OKYZ, , 1, reported]]
[[other, ABCD, OO, 1,]]

scala> testNyt.select($"person",explode($"person")).show(truncate=false)

person	col
[[Melena, RYZIK, , 1, reported], [derba, OKYZ, , 1, reported]]	[Melena, RYZIK, , 1, reported]
[[Melena, RYZIK, , 1, reported], [derba, OKYZ, , 1, reported]]	[derba, OKYZ, , 1, reported]
[[other, ABCD, OO, 1,]]	[other, ABCD, OO, 1,]

Interrogation


```
{
  "person" : [
    {
      "firstname" : "Melena",
      "lastname" : "RYZIK",
      "role" : "reported",
      "rank" : 1,
      "organization" : ""
    },
    {
      "firstname" : "derba",
      "lastname" : "OKYZ",
      "role" : "reported",
      "rank" : 1,
      "organization" : ""
    }
  ]
}
{
  "person" : [
    {
      "firstname" : "other",
      "lastname" : "ABCD",
      "rank" : 1,
      "organization" : "OO"
    }
  ]
}
```


Collection de 2 objets

?

Nécessite l'imbrication dans le select

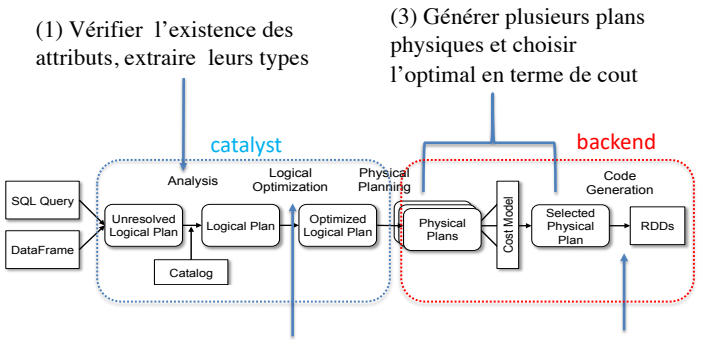

```
{
  "role" : "reported",
  "persons" : [
    {
      "firstname" : "Melena",
      "lastname" : "RYZIK",
      "rank" : 1,
      "organization" : ""
    },
    {
      "firstname" : "derba",
      "lastname" : "OKYZ",
      "rank" : 1,
      "organization" : ""
    }
  ]
}
```

Bilan

- Expressivité limitée du langage de schéma et du langage de requêtes
  - pas de distinction entre attributs optionnels et obligatoires
  - n'exprime pas l'union : String + [ ] + {long: String, lat: String}
  - quid de l'accès indexé aux éléments d'un Array et de l'imbrication dans le select?
- Autres pistes
  - Extensions SQL : *SQL++* de AsterixDB, *N1QL* de Couchbase, *SQL* de Apache Drill
  - Langages propriétaires : *Aggregation Pipelines* de Mongo, *JSONiq* de Zorba
  - Plusieurs connecteurs possibles avec Spark

Démo : prépartion TME

Optimisation de SQL sur Spark



(1) Génération du plan logique

- Plan logique = arbre d'opérateurs logique
- Analyse statique
  - résolution des noms d'attributs en utilisant le catalogue
  - Vérification du référencement des attributs
- Traduction SQL vers algèbre interne
  - Opérations arithmétiques : +, -, …
  - Fonctions d'agrégations : sum, avg, …
  - Algèbre interne (DSL) :
    - Project, Filter, Limit, Join, Union, Sort, Aggregate, UDFs, …

Illustration

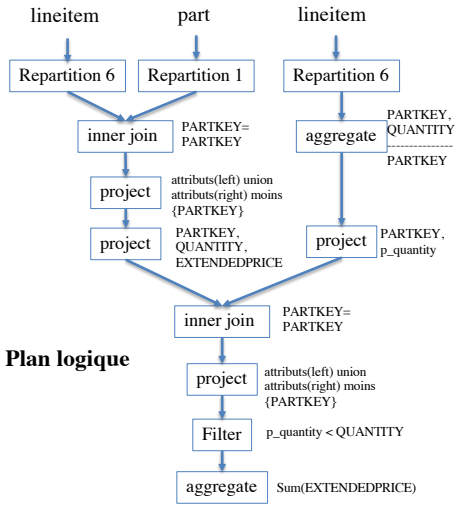
```
val lineitem = spark.read. ... .load(lineitem_t).coalesce(6)
val part = spark.read...load(part_t) coalesce(1)

val inner = lineitem.groupBy("PARTKEY")
            avg("QUANTITY").
            rename("avg(QUANTITY)","p_quantity")

val outer = lineitem.join(part, "PARTKEY")
            .select("PARTKEY",
                    "QUANTITY",
                        "EXTENDEDPRICE")


val q17 = inner.join(outer, "PARTKEY")
            .where("p_quantity<QUANTITY")
            .agg(sum($"EXTENDEDPRICE")/7)
```

Tpch Q17 simplifiée



Plan logique

Spark Explain

```
scala> q17_simp.explain(true)
== Parsed Logical Plan ==
…..

== Analyzed Logical Plan ==
(sum(EXTENDEDPRICE) / 7): double
Aggregate [(sum(EXTENDEDPRICE#127) / cast(7 as double)) AS (sum(EXTENDEDPRICE) / 7)#148]
+- Filter (p_quantity#92 < cast(QUANTITY#126 as double))
   +- Project [PARTKEY#11, p_quantity#92, QUANTITY#126, EXTENDEDPRICE#127]
      +- Join Inner, (PARTKEY#11 = PARTKEY#123)
         :- Project [PARTKEY#11, avg(QUANTITY)#89 AS p_quantity#92]
         :  +- Aggregate [PARTKEY#11], [PARTKEY#11, avg(cast(QUANTITY#14 as bigint)) AS avg(QUANTITY)#89]
         :     +- Repartition 6, false
         :        +- Relation[ORDERKEY#10,PARTKEY#11,SUPPKEY#12,LINENUMBER#13,QUANTITY#14,EXTENDEDPRICE#15,DISCOUNT#
         +- Project [PARTKEY#123, QUANTITY#126, EXTENDEDPRICE#127]
            +- Project [PARTKEY#123, ORDERKEY#122, SUPPKEY#124, LINENUMBER#125, QUANTITY#126, EXTENDEDPRICE#127, DISC
               +- Join Inner, (PARTKEY#123 = PARTKEY#53)
                  :- Repartition 6, false
                  :  +- Relation[ORDERKEY#122,PARTKEY#123,SUPPKEY#124,LINENUMBER#125,QUANTITY#126,EXTENDEDPRICE#127,DIS
                  +- Repartition 1, false
                     +- Relation[PARTKEY#53,NAME#54,MFGR#55,BRAND#56,TYPE#57,SIZE#58,CONTAINER#59,RETAILPRICE#60,COMMENT
```


(2) Optimisation du plan logique

- Catalyst : réécriture de l'arbre d'opérateurs
  - Spark 2.4 : plus de 100 règles regroupées par lots (batch)
  - écrites en Scala, pas de documentation précise (analyse code)
  - Déclenchement : une seule fois, point fixe (nb itérations 100)
  - Quelques règles utiles
    - Elimination des sous-requêtes
    - *ColumnPruning* : Elimination des attributs inutiles
    - *CollapseProject* : Combinaison des projections
    - *PushDownPredicate* et *PushPredicateThroughJoin* : évaluation des filtres le plus en amont possible et/ou en concomitance des jointures
    - *InferFiltersFromConstraints* : rajouter des filtres en fonction de la sémantique des opérateurs
    - Elimination des distincts et des sorts
    - expansion des constantes, réécriture des filtres

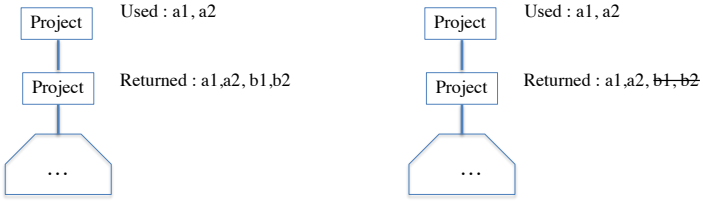
Signatures de opérateurs algébriques

- Structure récursive
  - Op(arg_1, …, arg_n, child), où child est un arbre désignant un plan logique
  - op.used désignent la liste des attributs utilisés
  - op.returned désignent la liste des attributs retournés
  - op.arg_i pour accéder à l'argument (comme pour un objet)
- Quelques exemples
  - Filter(cond, child) : cond est la condition à vérifier
  - Project(projList, child) : projList est la liste d'attributs à projeter
  - Aggregate(grpExp, aggExp, child)
    - grpExp : attributs de partitionnement
    - aggExp : fonctions d'agrégation
  - Join(left, right, joinType, condition)
    - left et right: plans
    - joinType : inner, outer, full, cross, …

Lot *ColumnPruning*
Règle d'élimination des attributs inutiles

cas op = Project(_, p_2 : Project)

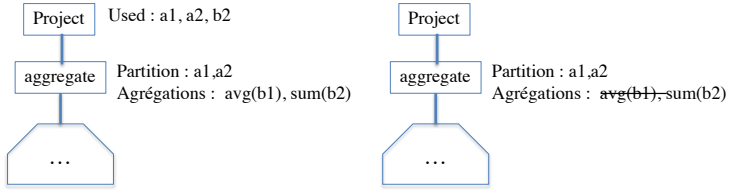
- si p2.returned ⊄ op.used /*certains attributs de p2 inutiles pour op*/
- alors p2.projList := p2.projList ∩ op.user /*les éliminer*/



Lot *ColumnPruning*
Règle d'éliminations des agrégations inutiles

cas op = Project(_, a: Aggregate)

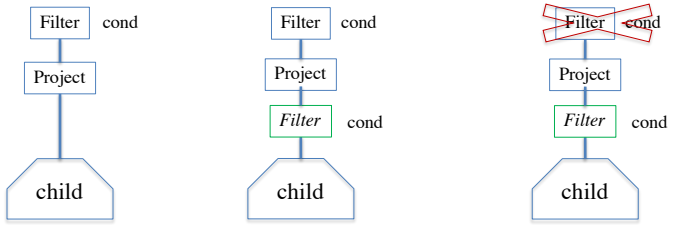
- si a.returned ⊄ op.used /*certains attributs de a inutiles pour op*/
- alors a.AggExp = a.AggExp.filter(op.used) /*n'effectuer que les agrégations sur les attributs utiles pour op*/



Règle *PushDownPredicate*

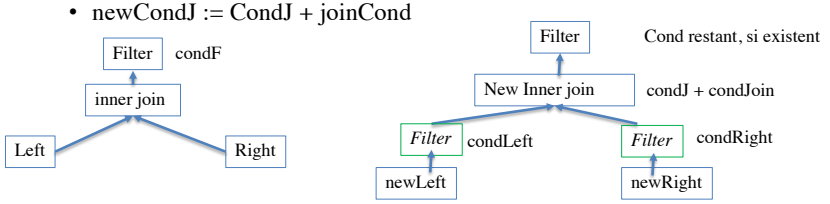
cas op=Filter(cond, Project(_, child))

- si cond est déterministe et peut être poussée
- alors Filter(cond, Project(_, Filter(cond, child)))
- Filter le plus externe sera éliminé après des vérifications



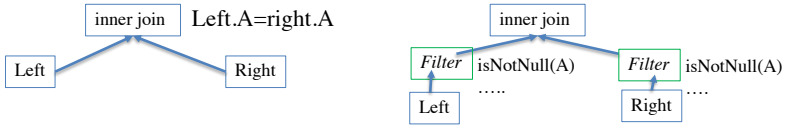
Règle *PushPredicateThroughJoin*

- Diviser la condition en 2 sous-conditions qui pourraient être évaluées dans une des branches de la jointure et une évaluée avec la jointure
- Filter(condF, Join(left, right, joinType, CondJ)
  - Diviser condF en leftCond, rightCond et joinCond
  - cas joinType = Inner
    - newLeft := left où child = Filter(condLeft, child)
    - newRight idem
    - newCondJ := CondJ + joinCond

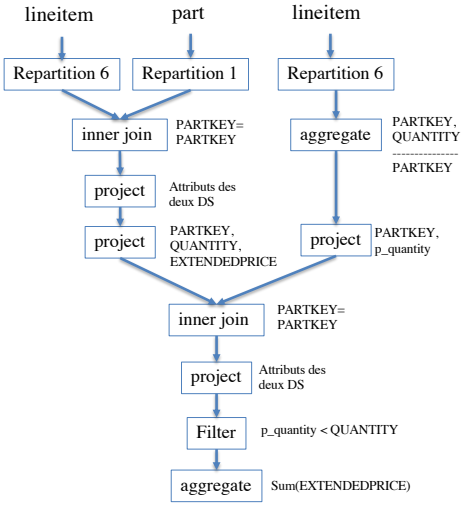


Règle *InferFiltersFromConstraints*

- Conditions qui s'ajoutent à des filtres existants ou à des jointures en function des contraintes sémantiques des opérateurs
  - Ex. l'attribut de jointure ne doit pas être *Null*
- Comme pour *PushPredicateThroughJoin*. diviser la condition en sous-conditions à propager
- Join(left, right, type, CondJ)
  - cas joinType = Inner
    - Extraire toutes les contraintes pour left et right et en extraire les filtres
    - Rajouter le filtre isNotNull sur les attributs de jointure

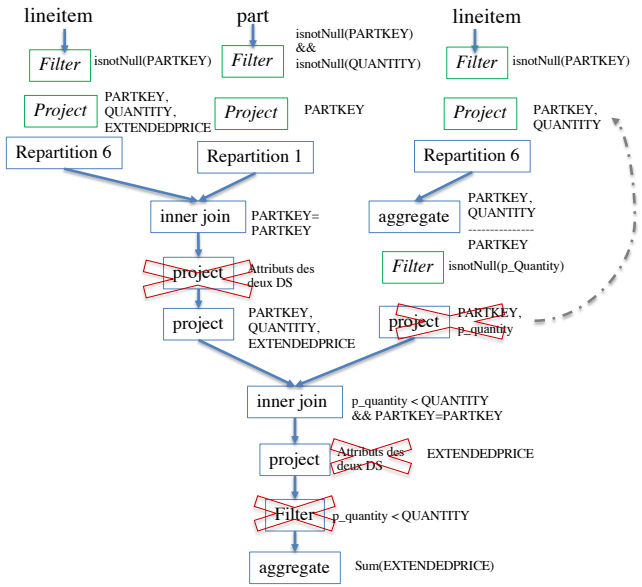


Plan logique Q17



Attention sens de lecture inversé

Plan logique Q17 optimisé



Attention sens de lecture inversé

Spark Explain

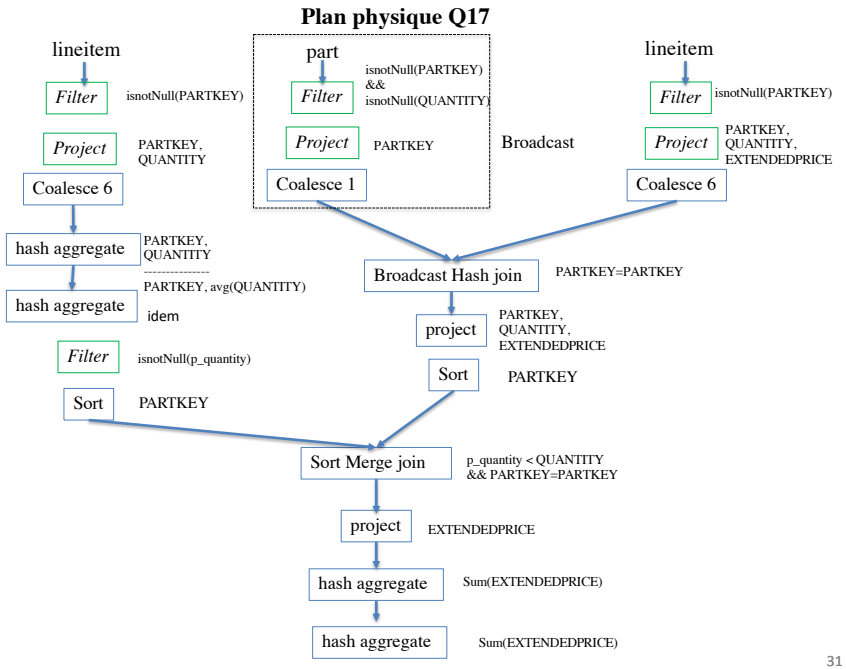
```
…
== Analyzed Logical Plan ==
…
== Optimized Logical Plan ==
Aggregate [(sum(EXTENDEDPRICE#127) / 7.0) AS (sum(EXTENDEDPRICE) / 7)#148]
+- Project [EXTENDEDPRICE#127]
   +- Join Inner, ((p_quantity#92 < cast(QUANTITY#126 as double)) && (PARTKEY#11 = PARTKEY#123))
      :- Filter isnotnull(p_quantity#92)
      :  +- Aggregate [PARTKEY#11], [PARTKEY#11, avg(cast(QUANTITY#14 as bigint)) AS p_quantity#92]
      :     +- Repartition 6, false
      :        +- Project [PARTKEY#11, QUANTITY#14]
      :           +- Filter isnotnull(PARTKEY#11)
      :              +- Relation[ORDERKEY#10,PARTKEY#11,SUPPKEY#12,LINENUMBER#13,QUANTITY#14,EXTENDEDPRICE#15,DISCOUN
      +- Project [PARTKEY#123, QUANTITY#126, EXTENDEDPRICE#127]
         +- Join Inner, (PARTKEY#123 = PARTKEY#53)
            :- Repartition 6, false
            :  +- Project [PARTKEY#123, QUANTITY#126, EXTENDEDPRICE#127]
            :     +- Filter (isnotnull(PARTKEY#123) && isnotnull(QUANTITY#126))
            :        +- Relation[ORDERKEY#122,PARTKEY#123,SUPPKEY#124,LINENUMBER#125,QUANTITY#126,EXTENDEDPRICE#127,DIS
            +- Repartition 1, false
               +- Project [PARTKEY#53]
                  +- Filter isnotnull(PARTKEY#53)
                     +- Relation[PARTKEY#53,NAME#54,MFGR#55,BRAND#56,TYPE#57,SIZE#58,CONTAINER#59,RETAILPRICE#60,COMMENT
```

(3) Génération du plan physique

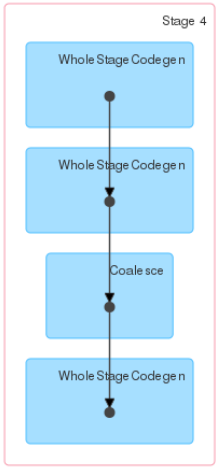
- Phase 1 : Transformer le plan logique en un plan physique
- Phase 2 : appliquer règles d'optimisation
- Stade préliminaire de développement
  - Pipelining d'opération (filter et project)
  - Choix entre jointure par hachage et diffusion en fonction de la taille des données
  - Utilisation du cout pour réordonner les jointures?

Spark Explain

```
…
== Optimized Logical Plan ==
…
== Physical Plan ==
*(9) HashAggregate(keys=[], functions=[sum(EXTENDEDPRICE#127)], output=[(sum(EXTENDEDPRICE) / 7)#148])
+- Exchange SinglePartition
   +- *(8) HashAggregate(keys=[], functions=[partial_sum(EXTENDEDPRICE#127)], output=[sum#169])
      +- *(8) Project [EXTENDEDPRICE#127]
         +- *(8) SortMergeJoin [PARTKEY#11], [PARTKEY#123], Inner, (p_quantity#92 < cast(QUANTITY#126 as double))
            :- *(3) Sort [PARTKEY#11 ASC NULLS FIRST], false, 0
            :  +- *(3) Filter isnotnull(p_quantity#92)
            :     +- *(3) HashAggregate(keys=[PARTKEY#11], functions=[avg(cast(QUANTITY#14 as bigint))], output=[PARTKEY#11, p_quantity#9
            :        +- Exchange hashpartitioning(PARTKEY#11, 200)
            :           +- *(2) HashAggregate(keys=[PARTKEY#11], functions=[partial_avg(cast(QUANTITY#14 as bigint))], output=[PARTKEY#11, su
            :              +- Coalesce 6
            :                 +- *(1) Project [PARTKEY#11, QUANTITY#14]
            :                    +- *(1) Filter isnotnull(PARTKEY#11)
            :                       +- *(1) FileScan csv [PARTKEY#11,QUANTITY#14] Batched: false, Format: CSV, Location: InMemoryFileIndex[hdfs://pp
            +- *(7) Sort [PARTKEY#123 ASC NULLS FIRST], false, 0
               +- Exchange hashpartitioning(PARTKEY#123, 200)
                  +- *(6) Project [PARTKEY#123, QUANTITY#126, EXTENDEDPRICE#127]
                     +- *(6) BroadcastHashJoin [PARTKEY#123], [PARTKEY#53], Inner, BuildRight
                        :- Coalesce 6
                        :  +- *(4) Project [PARTKEY#123, QUANTITY#126, EXTENDEDPRICE#127]
                        :     +- *(4) Filter (isnotnull(PARTKEY#123) && isnotnull(QUANTITY#126))
                        :        +- *(4) FileScan csv [PARTKEY#123,QUANTITY#126,EXTENDEDPRICE#127] Batched: false, Format: CSV, Location: In
                        +- BroadcastExchange HashedRelationBroadcastMode(List(cast(input[0, int, true] as bigint)))
                           +- Coalesce 1
                              +- *(5) Project [PARTKEY#53]
                                 +- *(5) Filter isnotnull(PARTKEY#53)
                                    +- *(5) FileScan csv [PARTKEY#53] Batched: false, Format: CSV, Location: InMemoryFileIndex[hdfs://ppti-dac-1:50100/
```

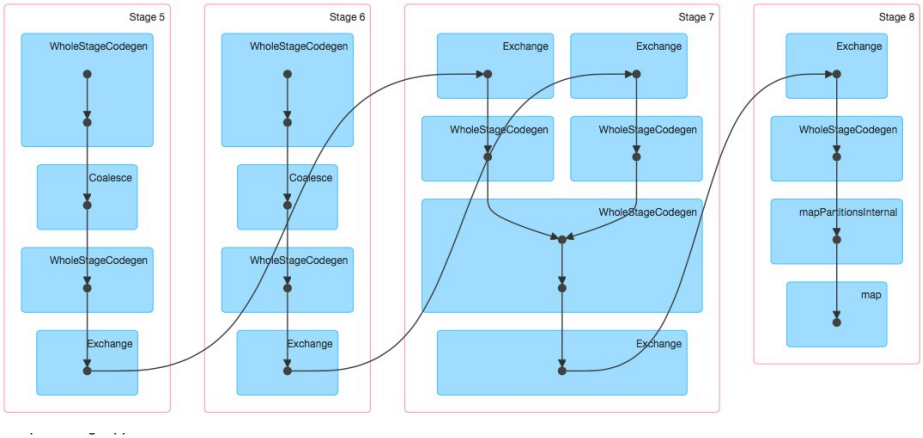



Visualisation plan



Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
4	run at ThreadPoolExecutor.java:1142 +details	2018/11/07 16:45:13	1 s	1/1	23.2 MB			

Visualisation plan



Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
8	show at <console>:26 +details	2018/10/22 19:36:26	84 ms	1/1			11.5 KB	
7	show at <console>:26 +details	2018/10/22 19:36:24	2 s	200/200			98.0 MB	11.5 KB
6	show at <console>:26 +details	2018/10/22 19:36:11	13 s	6/6	721.1 MB			86.2 MB
5	show at <console>:26 +details	2018/10/22 19:36:10	12 s	6/6	721.1 MB			11.8 MB

Autres pistes d'optimisation

- L'organisation physique des données compte!
  - Format de stockage colonnes : ORC, parquet
  - Partitionnement sur disque
  - Caching de donées accédés en répétition

Règles déclenchées pour tpch-q17

1. PushDownPredicate
2. ColumnPruning
3. CollapseProject
4. ConstantFolding
5. RemoveRedundantProject
6. PushPredicateThroughJoin
7. InferFiltersFromConstraints
8. PushPredicateThroughJoin
9. PushDownPredicate
10. PushPredicateThroughJoin
11. PushDownPredicate
12. ColumnPruning
13. CombineFilters
14. RemoveRedundantProject
15. PushDownPredicate