HyDRa Code Generation Algorithm

## Main

### Signature and location

generateElement(conceptualSchema : ConceptualSchema)

generate.mtl

### Input

Conceptual Schema

### Output

* Utility classes (Exceptions, Logger, Util transform and escape fonctions, DBConnection , SparkConnectionMgr)
* Conditions related classes : Simple/And/Or Conditon, POJOAttribute enum , Operator
* IPojo interface, LoggingPojo (implements Serializable, Cloneable, IPojo)
* [Entity Type](#_Entity_Type) classes
* [Entity Type TDO (Technical Data Object)](#_Entity_Type_TDO) classes
* Entity Type Service abstract classes
* Entity Type Service Implementation classes
* Relationship POJO classes
* Relationship TDO
* Relationship Services
* Relationship Service Implementation

### Description

Generates generic and utility classes then iterates on entity types and relationship types to generate POJO, TDO, Services and Implementation classes.

## Entity Type POJO class

### Signature and location

generateEntityTypePojoClass(entityType : EntityType, conceptualSchema: ConceptualSchema)

generateEntityTypeRelatedClasses.mtl

### Description

Generate pojo classes of conceptual entity types.

* With types private attrbutes for simple conceptual attributes.
* Enums with role name containing the current entity.
* List if cardinality of role is 0-N or 1-N
* List <POJO Opposite role Entity> attributes for simple (binary and no attributes) relationship types where entity is involved
* List<POJO REL> for complex relationships (with attributes or # roles > 2)

## Relationship Type POJO class

Generates pojo classes of conceptual relationship types.

* Private attributes for relationship type attributes (if any)
* Private POJO Entity type class attributes for each role of the rel.

## Entity Type TDO (Technical Data Object)

### Signature and location

generateEntityTypeTechnicalDataObjectsClass(ent : EntityType, conceptualSchema : ConceptualSchema)

generateEntityTypeRelatedClasses.mtl

### Description

Entity Type TDO object are object extending EntityType Pojo classes where we add attributes corresponding to the reference fields (source as well as target) mapped to a role of the specific entity type.

We also take target because the id used in refs may not be a mapped conceptual attribute.

Used for join condition when reading with role based methods.

* For all fields of reference block mapped to role of current entity in a binary relationship , we declare an attribute
* If the opposite role is the current entity and is not hosted in the same PhysicalStructure as the first role we also generate the attribute.

## Relationship Type TDO

Extend Rel POJO class and adds String attributes for each source and target fields of each refrence block mapped to any of roles of the rel.

For role

For ref

For ref.sourceField

For ref.targetField

Attribute declaration/setter/getter.

## Entity Type Service

generateEntityTypeServices

main.services.generateEntityTypeServices.mtl

### Description

Generates an abstract class EntityService containing select/insert/delete and update methods.

Methods are declaration and implementation are generated using the same file but providing a Boolean argument ‘override’ which indicates the implementation or not.

### Select

generateSimpleSelectMethods(entityType, conceptualSchema, override)

Generates select methods, one generic for the entity type and several others depending on the mapped physical structures (being not an Embedded structure).

* Dataset<Entity> getEntityList(Condition c ,…) ([Implementation](#_getEList(Condition_<EAttribute>_c)))
* For each mapped Structure of Entity

For each Database

Dataset getEntityInSTRUCTFromDB

* Also generates getEntityListByATTRIBUTE

generateRoleBasedSelectMethods(entityType, conceptualSchema, override)

main.services.select.entitytype.generateRoleBasedSelecMethods.mtl

## getEList(Condition <EAttribute> c)

### Signature and location

### generateSimpleSelectMethodsImpl

### main.impl.select.entitytype.generateSimpleSelectMethodsImpl.mtl

Implementation of this method is in the abstract class.

### Description

Top level function reconciling calls to sub select methods getEntityInSTRUCTFromDB and returning a joined Dataset <Entity> of all complete entity data object found in all mapped databases.

### Algorithm Join

If Entity have an identifier

Produces a Dataset<Row> , from a full outer join on id fields , with row being a sequence of all columns of subsequent attributes of retrieved Dataset<Entity> , column are concatenated with ‘\_i’ I being the number of the Dataset.

Then we reconstruct the Row object to Entity objects. First Non null attribute is compared with other ‘attribute\_i’ , if inconstistency is found we detect and log.

Else

We simply perform a Dataset.union on all datasets.

**RefilterFlag**

**This flag is a mutable boolean that we pass to sub select method in order to indicates if the final constructed dataset have to be refiltered according to the given Condition. This is needed because when a physical structure contains entity related data, it may not necessary contains data on the attribute specified in the given condition. When this occurs, we do not filter the rows in the sub select method in order to retrieve the complete set of attributes data. We set the ‘refilterFlag’ to true in order to mention that rows returned may not satisfy the condition, and filtering must be done when objects are fully reconstructed.**

DropDuplicates

Final dataset may contain duplicate rows when reading key value with patterns such as PRODUCT:id:REVIEW:reviewed (why? Don’t remember)

## getEListInSTRUCTFromRELATIONALDB or getETDO…

### Signature

generateSimpleSelectFromGivenRelationalTable(pojoName : String, attributes : Collection(Attribute), struct: AbstractPhysicalStructure, db : Database, references : Sequence(Sequence(OclAny)))

main.impl.select.entitytype.generateSimpleSelectFromGivenRelationalTable.mtl

### Description

Will return a Dataset<EntityType> or <EntityTypeTDO> from a relational table using SparkMgr Connection.

First gets the where clause using the [build recursive function](#_getSQLWhereClauseInSTRUCTFromRELDB)

Then we retrieve a Dataset<Row> and proceed to convert each row to the EntityType object.

**If refrences are passed as arguments we proceed to build a TDO object**

## getSQLWhereClauseInSTRUCTFromRELDB

### Signature and location

There exist 2 overloading methods with EnittyType or RelationshipType as argument which call the method below

generateGetSQLWhereClauseMethod(pojoName : String, pojoAttributes : Collection(Attribute), struct: AbstractPhysicalStructure, db : Database)

main.impl.select.entitytype.generateSimpleSelectFromGivenRelationalTable.mtl

### Example

Pair<String,List<String>>getSQLWhereClauseInDirectorTableFromMydb(Condition<DirectorAttribute> condition, MutableBoolean refilterFlag)

### Description

Generate a recursive method (for And & Or Condition) that based on the argument Condition constructs a Pair containing a 'where' clause statement on the attribute of the condition and using the specified operator + a wildcard for the value (ex : “id = ?” ), the second argument of the Pair is the values to replace in the where clause.

Note that if a conceptual attribute is mapped to a LongField. The where have to take the pattern into account. The operator will be LIKE or NOT LIKE with different wildcards for the component of the longfield. One for the searched attribute, one for possible others variables

If no attribute of the concerned PhysicalStructure is present in the given condition the returned where clause is always true , 1=1, in order to get all rows and refilterFlag is set to true.

## getE[TDO]InSTRUCTFromDOCDB

### Signature and location

generateSimpleSelectFromGivenDocumentCollection(pojoName : String, entOrRel : OclAny, attributes : Collection(Attribute), struct: AbstractPhysicalStructure, db : Database, references : Sequence(Sequence(OclAny)))

main.impl.select/entitype.generateSimpleSelectFromGivenDocumentCollection.mtl

### Description

Using the [match query generated](#_getBSONMatchQueryInSTRUCTFromDOCDB) will call the SaprkConnectionMgr and get a Dataset<Row> , then proceed to convert it to Dataset<Entity> or **<EntityTDO> if references argument is not empty.**

The conversion of Row to Entity object may return several entity object for one row because of possible complex structures ( A document containing an array field with other conceptual attributes inside each array element). So we used ‘flatMapFunction’ instead of MapFunction (as in [SQL implem](#_getEListInSTRUCTFromRELATIONALDB__o))

When retrieving the resulting Pojo Dataset we need to ‘dropDuplicates’ on identifier (Why?)

### Algo Row To Pojo

We look for attributes that are mapped to fields of the first level first.

We create a pojo at this level and set their found attribute values.

We go deep in complex fields (this is rather complex as well, to specify? ) , arrays or embedded objects, and clone the previous level pojo for each array elements. We used the setter if found fields mapped to the concerned entity type.

If setter is used it means that this object is more complete than the one at the preivous level , so it becomes the master an flag it as to be added to the List of returned Pojos for this Row.

Before adding the Pojo to the list it is necessary to refilter as the match query returns complete documents that match the condition. If a document contains an array with multiple nested documents, only one of them may satisfy the condition but the whole document will be returned and Pojo objects will also be created for other non matching array elements. That’s why we evaluate the condition on the object before adding to the list.

Once it’s done for the first level, we recursevliy call the same method with an EmbeddedObject as argument, being then the second level and so on…

**When references are passed we also look for fields corresponding to the fields of the refs in order to build a TDO object**

## getBSONMatchQueryInSTRUCTFromDOCDB

### Signature and ocation

generateBSONMatchQueryMethod(pojoName : String, attributes : Collection(Attribute), struct: AbstractPhysicalStructure, db : Database)

main.impl.select/entitype.generateSimpleSelectFromGivenDocumentCollection.mtl

### Description

Similar to SQL where clause building function but for MongoDB.

We build a ‘$match : …” JSON query with $eq , $and , …

## getEListFromSTRUCTFromKEYVALUE

### Signature and llocation

generateSelectKeyValue(pojoName : String, attributes : Collection(Attribute), struct : KeyValuePair, db : Database, references : Sequence(Sequence(OclAny)))

main.impl.select.entitytype.generateSelectKeyValue.mtl

### Description

Retrieves pojo or pojoTDO from a Key Value database.

As we cannot filter directly on values in a Redis DB, refilter flag is set to true.

We build a keypattern in order to query a subset of keys. This key pattern by concatenating ‘\*’ for each BracketsField and other literals .

If the given Struct of the method is a KVComplexField it means we are in a complex structure of value.

Otherwise the Row of Dataset<Row> contains only 2 string values, key and value.

For all the conceptual attributes, we check if they are mapped in the key or in the value. And we retrieve the attribute value using regex.

Same for TDO attributes.

## getEListByATTRIBUTE(ATTRIBUTETYPE attrname)

### Description

Selection methods for entity based on a conceptual attribute.

Implementation is a redirect call to [getEList](#_getEList(Condition_<EAttribute>_c)) (Condition.simple(EAttribute.attrname, Operator.EQUALS, attrname);

## getROLEInRELTYPE methods

### Signature and location

generateRoleBasedSelectMethods(entity: EntityType, conceptualSchema: ConceptualSchema, override: Boolean)

main.services.select.entitytype.generateRoleBasedSelectMethods.mtl

Code generated in ENTITYService classes

### Description

Generates abstract methods for all accessing methods based on roles of the current Entity.

Override flag indicates if implementation is to be generated.

Generates:

* getROLEListInREL(Condition<ROLE1 ENTITYAttribute> cond , Condition<ROLE2 ENTITYAttribute> cond 2,…)

Abstract method . ([Implementation](#_getROLEEListInREL(Condition_,_…)))

* getROLEListInRELByROLE2Condition(Condition<ROLE2ENTITYAttribute> cond)

Makes the call to the abstract method.

* getROLEListInRELByROLECondition(Condition<ROLEENTITYAttribute> cond)
* getROLEInRELByROLE2(ROLE2ENTITY e)

If cardinality of ROLE2 is N returns Dataset, name becomes getROLEList.

Implementation builds a simple or complex condition, based on ROLE2ENTITY identifiers, and then calls the abstract method.

* If the REL of the entity ROLE have attributes (complex relationship)

## getROLEEListInREL(Condition role1, Condition role2, Condition attributeOfREL, …) Implementation

### Signature and location

generateRoleBasedSelectMethodsImpl(entity: EntityType, role: Role, conceptualSchema: ConceptualSchema)

main.impl.select.entitytype.generateRoleBasedSelectMethodsImpl.mtl

### Description

Current description and implementation is done for binary relationships only. We consider role1 of type A, role2 of type B, -> or <- are foreign keys.

This methods implementation is dependent on the type of RELATIONSHIP, the role is involved with.

Conditions can be expressed on all components of the REL, roles and attributes of rel.

Using the mapped reference of role1 and role2 we determine if we are in situation :

* A <- AB -> B (jointable, if yes [code to join](#_generateCodeForJoinPhysicalStructur))
* A -> B where
  1. A and B are in the same db, so we can retrieve a REL POJO containing both A and B.
  2. A and B are not in the same DB or it involves at least one non relational db. So we retrieve ATDO and BTDO and join them.
* B -> A where
  1. A and B in same dbs, RELPOJO.
  2. A and B not in same dbs, ATDO, BTDO.
* A(B) . B is embedded in A. So REL POJO.
* B(A) . A is embedded in B. So REL POJO

Each retrieval block of A, B , REL have a refilterFlag condition. If true the refilter is done by joining with the previously inititialised ‘all’ Dataset, containing all the B objects satisfying the B condition. As it is possible that the B objects retrieved contains more than needed because the condition is expressed on a B attribute contained in another structure containing B objects not involved in REL. (Draw this)

Last step is to look if there are other structures containing A’s not involved in the REL in order to retrieve possible other attributes. We fullouter A1,A2 and then left full outer this with previously built A dataset.

### Algo

For each Reference of the current Role.

If opposite role in the same PhysicalStructure (not Embedded) (means that we are in a N-N relationship)

Else (2 refs not in same physical struct)

Generate [code to join](#_generateCodeForJoinPhysicalStructur) when a join table is involved.

### Notes

Etant donné A –r- B

* Pourquoi d’office aller rechercher les objets B sur B condition?
  1. Dans la condition sur B n’a pas pu etre vérifié dans le select, il faut rejoin avec , car le premier dataset retourné contient trop de datas.
* Pourquoi des fois ramener Dataset<R> et des fois RTDO ?

## generateCodeForJoinPhysicalStructure(entity: EntityType, role: Role, ref: Reference, role2 : Role, ref2: Reference)

### Description

Generates code to correctly join Datasets in case of N-N relationship involving a join table.

### Algo

Get the databases of targetField of both references.

Get the database of jointable (which is the physical structure containing the two refs)

Check if both refs dbs and join table are the same dbs and is a relational. => 3 SQL tables.

If only one db is the same has join table and relational . It means we join 2 SQL tables and a non relational one.

Different cases (here we still are in the case where we want to get A objects, getAByREL) :

* (A AB B) , all 3 in same db. RELPOJO is retrieved containing A & B, then map function on r.getA to cast to A objects.
* (A-AB) – (B) . 2 dbs. REL. RELPOJOTDO & BTDO are retrieved. Those are joined. Then we select A columns and cast.
* (A) – (AB – B). 2 dbs. RELPOJOTDO & ATDO, join and cast
* (A) (AB) (B) or (A B) (AB). Join table is in a different db. ATDO, RELTDO, BTDO. Double join, selection, cast.

# Other METHODS

## generateOuterJoinsMethods(entityType)

Signature and location

### Description

Generates 3 static methods returning Dataset<POJO> taking as arguments a List<Dataset<POJO>>.

They return left outer join, left full outer join or full outer joins.

# notes

## Insert (draft specification)

### InsertENT

* Check if the entity does not have mandatory linked elements (role 1-X in a RELTYPE) that are not provided (as inner class attributes or as arguments). If yes cancel.
* Different cases :
  1. Entity is not involved in a relationship type (standalone)
     + Generate insert statements for each dbs, reading mapping rules
       - Rel : INSERT(a,b,..) VALUES (…)
       - Doc : Can’t be in embedded structure if no role are mapped? So simple insert?
       - Key value : check mapping in key of in value, build series of set statements or hset…
  2. Entity is involved in relationships
     + A [1-1] – [0-N] B

### InsertR

Aslo consider add of R Pojo.

### Functions

* Check if the entity does not have mandatory linked elements (role 1-X in a RELTYPE) that are not provided (as inner class attributes or as arguments).
* Build TDO Object, by reading all refs involving E