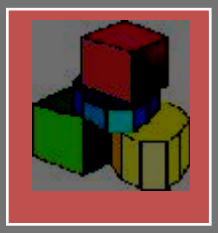


Elcaro

La Pythie du logiciel

Simulateur de SGBDR

1 Objectif

Le but de ce travail est de réaliser une modélisation et une rélalisation objet en utilisant le langage Java en version 1.7 maximum fournissant l'environnement **Elcaro** décrit dans la suite de ce document. Vous prendrez soin de concevoir un programme structuré mettant en évidence les concepts majeurs. Vous aurez à l'esprit lors de son élaboration que votre programme doit pouvoir : Evoluer dans le temps, admettre de nouvelles fonctionnalités venant naturellement compléter votre réalisation, accepter des optimisations permettant une plus grande performance de votre environnement, etc.

Ce travail permet aussi de revoir les principes de base de construction et d'évaluation qui sont sous-jacents à toute machine d'exécution et les techniques principales d'un SGBD.

2 Présentation générale de l'environnement Elcaro

On souhaite réaliser un système "de simulation" de SGBD relationnel. Cependant il ne serait pas raisonnable de vouloir construire l'intégralité des fonctionnalités d'un tel outil, nous nous restreindrons donc à un sous-ensemble en éliminant les aspects trop ambitieux dans le cadre de ce travail.

2.1 Spécifier les classes principales

Vous compléterez ce travail en définissant plus précisément la spécification de chaque classe élaborée : constructeur et méthodes de son interface.

Chaque classe sera pertinemment commentée. Vous préciserez, de façon circonstanciée, les choix qui ont présidé aux solutions que vous avez retenues. Vous utiliserez à cet effet tous les moyens de spécifications qui sont à votre disposition.

2.2 Structurer l'ensemble des concepts

Proposez une décomposition en package qui permette une bonne répartition des différentes classes à réaliser. Ceci facilitera la lisibilité du projet et rendra plus aisé son développement et sa maintenance. Une des utilisations possibles du mécanisme de package permet de mettre en évidence ceux qui peuvent être complétés. Vous pourrez aussi dans votre projet scinder l'espace des sources en plusieurs parties permettant, là encore, de rendre plus lisible l'ensemble du code développé.

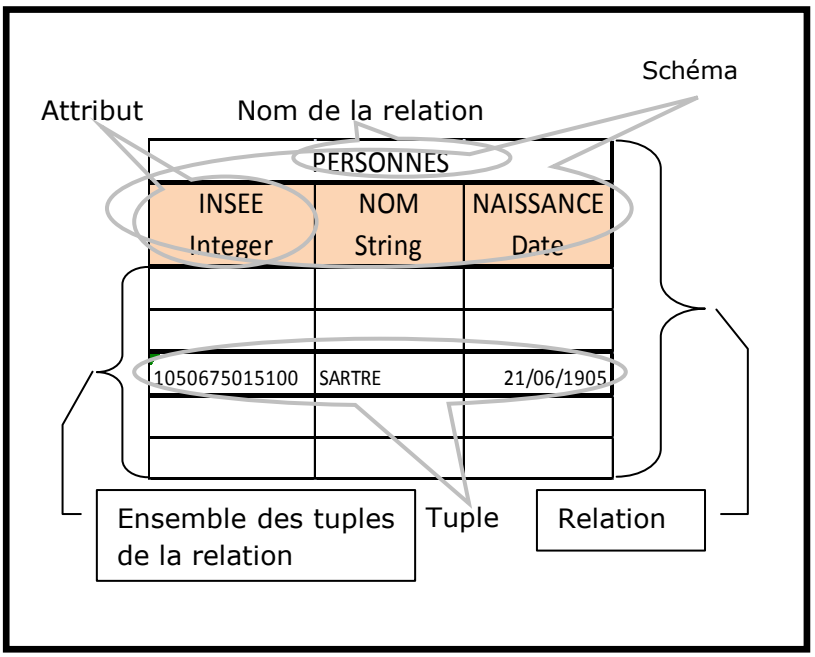
3 Réalisations à élaborer

Ayant structuré l'application, on doit commencer par réaliser les éléments conceptuels majeurs inhérents au domaine de l'application à concevoir (noyau). Ensuite seront développés les éléments secondaires qui viennent compléter utilement l'ensemble des outils logiciels à disposition, enfin un environnement interactif dédié à l'utilisateur final.

3.1 Le noyau (kernel) du système Elcaro

Réalisez les classes qui constituent le noyau de l'application **Elcaro**. Celles-ci sont donc indépendantes des aspects d'interaction dans le pur respect du modèle MVC. A ce point du développement on disposera d'un framework (cadre) minimal qui pourra être complété par la suite pour construire une application finalisée destinée à un utilisateur. Le principe d'un framework se met naturellement en œuvre dans le modèle objet. Il a pour but de fixer les éléments conceptuels minimaux qui permettront, sur sa base, de construire des variantes d'une application, voire des applications différentes mais recouvrant une thématique commune. L'héritage est un moyen qui se prête bien à l'élaboration d'une telle architecture, offrant la possibilité d'étendre, d'adapter les concepts initiaux vers ceux nécessaires dans l'application à réaliser. Le schéma architectural du framework peut préciser les possibilités d'extensions offertes et les parties qui resteront figées.

3.1.1 Présentation synthétique d'une BD



Dans le modèle de base de données relationnelle le concept principal est la notion de relation qui associe des attributs (au sens base de données) constituants des tuples. Chaque relation est caractérisée par un schéma qui indique quels sont les attributs et leurs types respectifs (pris souvent dans un ensemble fini de types autorisés, par exemple Integer, String, Date, ...). On admettra que chacun des types possibles supporte une relation d'ordre implicite. La relation constitue une table, comme le montre l'exemple ci-contre, contenant un nombre variable de tuples représentant la valeur de la relation à un instant donné. Chaque attribut d'un schéma indique le nom de celui-ci (nom de la colonne de la table) ainsi que son type que l'on caractérisera par le type Java correspondant (sa classe). Un tuple, qui constitue une ligne de la relation, contient une valeur de chacun des attributs du schéma. On considérera, pour simplification, que toute relation dispose d'une clé primaire (ce qui garantit l'unicité du tuple dans la relation) et l'on peut aussi considérer dans un premier temps que celle-ci ne soit constituée que d'un seul attribut. Comme tout type supporte une relation d'ordre, les tuples d'une relation pourront être ordonnés selon l'ordre de sa clé. Le degré d'une relation détermine le nombre de colonnes de la relation et sa cardinalité son nombre de tuples.

On ne prendra pas en compte tous les nombreux aspects qui caractérisent un SGBD, car ceci nécessiterait un investissement trop important, au moins dans un premier temps. Voici une liste non exhaustive des aspects qui ne feront partie des objectifs de ce Tp :

- gestion du modèle transactionnel,
- gestion de la concurrence,

- gestion d'évolution de schéma, tant au niveau d'une relation qu'au niveau d'une base de données
- Extension à la notion de cluster.
- traitement de BD aux dimensions démesurées.
- pour les aspects intégrés on ne développera pas nécessairement toutes les variantes envisageables.
- gestion des dépendances fonctionnelles
- gestion des index en mémoire secondaire
- voire d'autres auxquels l'auteur n'aurait pas pensé.

3.1.2 Type primitifs du SGBD

Comme dit précédemment, la nature des informations permettant de peupler une BD sera fixée préalablement. Cependant ceci ne veut pas dire que la liste ne puisse être élargie pour constituer un nouveau SGBD. Nous trouverons naturellement les types primitifs classiques dont nous disposons dans un langage de programmation comme Java : Byte, Short, Integer, Long, Float, Double, Character, Boolean ; Auxquels on ajoute les types String et Date considérés comme élémentaires et incontournables lors de l'usage d'une BD. Ceci constitue un panel de départ largement suffisant pour développer des exemples de BD réalistes. On pourra utiliser notre langage hôte (Java) comme support de ces types primitifs. On assurera l'hypothèse que tout type primitif du SGBD supporte une relation d'ordre. Il va de soi qu'ils disposent aussi d'une relation d'équivalence (égalité). La plupart des SGBDs travaillent en taille fixe, c-a-d que chaque type primitif occupe un espace mémoire prédéterminé. Pour cela les chaines de caractères sont définies avec une borne max et l'allocation est constante. Si on n'opte pas pour cette restriction, la mise en œuvre du stockage sera légèrement plus complexe.

3.1.3 Attribut de relation

Un attribut d'une relation constitue une "colonne" de la relation et a un type pris parmi les types du SGBD. Formellement, il n'y a aucun ordre entre les différents attributs d'une relation, cependant nous pouvons en déterminer un, correspondant à l'ordre d'énonciation de ceux-ci. L'ensemble des attributs constituant une relation sera caractérisé par un schéma qui énumérera tous les attributs avec pour chacun d'eux son nom et son type. Deux attributs d'un même schéma ne peuvent avoir le même nom.

3.1.4 Clé primaire

Tous les types primitifs disposant d'une relation d'équivalence, ils sont donc potentiellement candidat à être le type d'une clé primaire. Dans le cas où l'on considère un seul attribut caractérisant la clé, la mise en œuvre est aisée. Dans le cas plus sophistiqué où l'on admet que plusieurs attributs participent à la construction de la clé, il faut produire une key (information représentative de la clé) qui permette de constituer une bijection avec l'ensemble des valeurs possibles de la table. Il faut aussi choisir une solution généraliste pouvant s'appliquer indistinctement aux différents usages. On admettra que l'ordre d'énonciation des éléments de la clé forme l'ordre lexicographique à suivre pour établir la relation d'ordre. Si un schéma ne possède pas à priori de clé primaire, il est bien évidemment possible de construire un schéma issu du premier auquel on adjoint en tête un attribut au nom prédéterminé et d'un type permettant une grande discrimination (par exemple entier) pour jouer le rôle de clé unique grâce à une génération automatique de valeurs distinctes, ou bien on considère que c'est l'ensemble des attributs qui constitue la clé.

3.1.5 Relation

Une relation peut être considérée comme une table ayant X colonnes, correspondant aux différents attributs du schéma de sa relation, et Y lignes correspondant aux différentes valeurs contenues à un instant donné dans cette relation. X est défini comme étant le degré de la relation qui est de-facto le degré de son schéma et Y est défini comme étant la cardinalité de la relation. Le degré est invariable alors que la cardinalité varie en fonction des ajouts ou retraits effectués sur cette relation.

Ce qui caractérise notablement une relation est le fait que l'on puisse énumérer l'ensemble de ses tuples, ne serait-ce à terme que pour les l'afficher auprès d'un utilisateur en ayant fait la demande.

3.1.5.1 Nature d'une Relation

Les relations peuvent être de différentes natures, en tout premier lieu on considère les relations pleines et entières (StateFull) qui sont en extension (tabulée) et construite par l'utilisateur via des ajouts (ou retraits). Elles constituent les données primitives dans une BD. A contrario on peut considérer des relations virtuelles (StateLess) qui sont les résultats d'opérations mettant en jeu d'autres relations (par exemple des relations StateFull) et pour lesquelles il est impossible d'effectuer des ajouts ou des retraits. Dans le premier cas l'ensemble des informations constituant la relation doit être mémorisé (d'ou le terme StateFull), dans le second cas, ceci n'est pas nécessaire, car l'ensemble des tuples de cette relation peut être énuméré par calcul (d'ou le terme StateLess).

Du fait que toute relation de nature StateFull possède une clé primaire, chaque élément (ligne), que l'on nomme tuple, de la relation est caractérisée par une clé unique (2 tuples d'une même relation ne possèdent pas une même clé). Il s'ensuit qu'un élément de la relation peut être retrouvé par sa clé. Dans le cas des relations StateLess, la notion de clé est assujettie aux relations sous-jacentes et donc ne nécessite pas une définition de clé primaire spécifique.

3.1.5.2 Lien entre Relation StateFull & StateLess

Une relation de nature StateFull nécessite un support de stockage pour conserver l'intégralité de l'information. Les supports de stockage peuvent être variés, classiquement un SGBD utilise un support rémanent, soit le SGF offert par l'environnement sous-jacent, soit un dispositif ad-hoc. On peut aussi envisager pour des usages plus légers de proposer un simple stockage en mémoire centrale, ce qui n'interdit pas la possibilité de sauvegarder l'information sur un support rémanent pour une exploitation en multi-sessions.

Quoiqu'il en soit, il doit toujours être envisageable de tabuler une relation StateLess, c-a-d de produire une relation StateFull correspondante. Dans cette opération, il faudra s'assurer que la relation créée respecte les contraintes d'une relation StateFull et entre-autres le fait qu'elle dispose d'un schéma ayant une clé.

De même on peut envisager de passer une relation StateFull d'un format de stockage vers un autre.

3.1.5.3 Internalisation vs externalisation des fonctionnalités d'une relation

L'ensemble des fonctionnalités (méthodes) attachées à la notion de relation doit être établi avec pertinence. La question se pose en termes d'internalisation ou d'externalisation de la fonctionnalité. Un exemple caractéristique de ceci est l'intégration de la notion de relation d'ordre avec les interfaces Comparable et Comparator qui pour la première est une forme internalisée et pour la seconde une forme externalisé de ce concept ; Attention la seconde n'est pas en toute circonstance la meilleure. Cette question se pose de façon très similaire dans le cadre des relations quand on souhaite leurs appliquer un ordonnancement selon un critère particulier ; Une première approche consiste à concevoir une fonctionnalité (méthode), cependant ceci pourrait être vu comme une nouvelle relation de type stateLess. De même l'impression est soumise à la même dualité. Si dans le cas de l'ordonnancement l'argumentaire est relativement facile en faveur de la seconde solution (externalisation), le cas de l'impression est peut être plus discutable.

3.1.6 Tuple

Le tuple correspond une ligne de la relation, il est constitué des valeurs de chaque type d'attribut placées dans l'ordre du schéma de la relation.

A noter : Les SGBD autorisent normalement la présence de valeurs nulles dans le tuple. Pour simplifier le traitement nous ne considérerons pas cette possibilité car cela ne nuit pas à l'élaboration d'exemples réalistes.

3.1.7 Index

Un index est une permutation de l'ensemble des tuples d'une relation dont l'intérêt réside dans l'ordre d'énumération des tuples qu'il autorise. Les index sont particulièrement utiles pour certains calculs et tout particulièrement celui de jointure de 2 relations. La notion d'index peut être considérée comme fugace : l'index est construit de façon temporaire pour assurer un calcul précis et ne perdure pas au delà de ce calcul, cependant on pourra le mémoriser soit durant une session d'usage d'une base de données, soit de façon encore plus rémanente. Dans ces 2 derniers cas il faut envisager la mise à jour de l'index en fonction des évolutions de la relation sous-jacente.

3.1.8 Base de données

Une base de données est constituée d'un ensemble de relations. Elle se spécifie essentiellement par l'ensemble des schémas de ses relations et est concrétisée par l'ensemble des tuples de ces relations.

3.1.9 SGBD

L'environnement qui permet de gérer différentes bases de données. Il constitue l'objectif de ce Tp.

3.1.10 Les opérateurs

L'algèbre relationnelle propose un certain nombre d'opérateurs. L'usage de ceux-ci se fait, classiquement, via un langage d'interrogation de base de données, cependant il ne faut pas confondre les 2 niveaux : les opérateurs existent indépendamment du langage et le langage peut offrir des facilités n'étant pas considérées du niveau de l'algèbre. SQL[1], qui est certainement le plus utilisé des langages d'interrogation, offre un large panel d'opérateurs qui permet de réaliser les calculs classiques utiles à l'exploitation d'une base de données. Là encore nous resterons raisonnables en réduisant au strict nécessaire les opérateurs que nous proposerons tout en assurant la possibilité de les compléter ultérieurement. Nous reviendrons plus tard sur le langage SQL quand il s'agira d'offrir une interface utilisateur "raisonnable". Les grandes catégories d'opérateurs sont :

- Les opérateurs relationnels (unaires ou binaires)
- Les opérateurs ne relevant pas spécifiquement de l'algèbre
  - *opérateurs et fonctions arithmétiques*
  - *opérateurs et fonctions sur les chaînes de caractères*
  - *opérateurs d'agrégation (de regroupement)*

L'ensemble des opérateurs unaires est essentiellement constitué des opérateurs relationnels, on y trouve les incontournables :
- Selection
- Projection

L'ensemble des opérateurs binaires est constitué des opérateurs ensemblistes comme :
- Union
- Intersection
- Différence
- Produit
- Jointure

Ceci est complété par les opérations de création, d'impression, etc.

Pour assurer l'impression d'une relation de façon claire et lisible, comme nous l'avons évoqué précédemment nous pouvons procéder soit par internalisation soit par externalisation. Quoiqu'il en soit cette impression devra être formatée pour être agréablement lisible, on aura une présentation enrichie avec une tabulation de qualité. Par exemple comme ci-dessous pour l'exemple du départ :

```
Personne(NUM:INT          NOM:STRING   PRENOM:STRING            DATE:DATE)with 10 tuples
NUM     NOM      PRENOM                DATE
 192  NOM192   PRENOM192   Thu Nov 17 14:25:58 CET 1977
 121  NOM121   PRENOM121   Thu Jul 04 22:00:29 CEST 1996
 408  NOM408   PRENOM408   Sun May 25 03:58:50 CET 1975
 294  NOM294   PRENOM294   Sun Aug 20 18:56:03 CEST 1989
 694  NOM694   PRENOM694   Mon Oct 06 02:19:03 CET 1975
 744  NOM744   PRENOM744   Sun Dec 17 04:55:36 CET 1978
 585  NOM585   PRENOM585   Sun May 08 12:00:23 CEST 1983
 783  NOM783   PRENOM783   Wed Oct 24 08:43:17 CET 1990
1717  NOM1717  PRENOM1717  Sat Jun 15 02:03:56 CEST 1996
1935  NOM1935  PRENOM1935  Tue Mar 04 21:52:50 CET 1975
```

Nous donnons ci-après la définition de quelques uns des opérateurs.

[1] Structured Query Language

3.1.10.1 Projection

- La projection est l'opérateur qui permet de produire une relation dont les tuples sont les mêmes que ceux de la relation initiale (sauf voir remarques) dont le schéma est un sous-schéma de la relation initiale.
- Notation : $R' = \prod_{a_i,\ldots,a_k} R$
- la relation R' correspond à la projection des attributs a_i ... a_k de la relation R.
- remarque :
  - La relation créée possède pour schéma le sous-ensemble des attributs sélectionnés de la relation initiale. Cette projection peut conduire à avoir des doublons, il faut garder l'aspect ensembliste.
  - Par notre définition le premier attribut de celui-ci constitue la clé primaire auquel cas tous les tuples de la relation initiale ne seront conservés quand des doublons apparaissent. On pourrait envisager d'ajouter une clé primaire automatique pour conserver l'ensemble des tuples initiaux.
  - Le premier attribut ne fait pas partie du schéma, il pourrait y avoir des doublons dans le résultat de la projection, ce qu'on interdit (le résultat final doit être ensembliste)

3.1.10.2 Sélection

- La sélection (parfois appelé restriction) est l'opérateur qui permet de sélectionner des tuples selon un prédicat donné.
- Notation : $R' = \sigma_{predicat} R$
- la relation R' correspond à la sélection des tuples de R respectant les conditions fixées par le prédicat.
- Remarque :
  - Les conditions sont exprimées à l'aide d'opérateurs portant sur les types primitifs. On peut citer les opérateurs d'(in)égalité, relationnels (relation d'ordre) et de composition logique. Dans un premier temps on se restreindra à la première catégorie.

3.1.10.3 Union

- L'union est un opérateur binaire permettant de produire une relation R' avec tous les tuples appartenant à R1 ou à R2.
- Notation : $R' = R1 \cup R2$
- la relation R' est le résultat de l'union des tuples des relations R1 et R2.
- Remarque :
  - R1 et R2 doivent être de même schéma sinon l'union est impossible. R' est de même schéma.
  - L'union élimine tous les doublons existants (aspect ensembliste de la relation).
  - Cet opérateur est commutatif.

3.1.10.4 Intersection

- L'intersection est un opérateur binaire permettant de produire une relation R' avec tous les tuples appartenant à R1 et à R2.
- Notation : $R' = R1 \cap R2$
- Remarque :
  - la relation R' est le résultat de l'intersection des tuples des relations R1 et R2.

- R1 et R2 doivent être de même schéma sinon l'intersection est impossible. R' est de même schéma.
- Cet opérateur est commutatif.

3.1.10.5 Différence

- L'intersection est un opérateur binaire permettant de produire une relation R' avec tous les tuples appartenant à R1 et n'appartenant pas à R2.
- Notation : $R' = R1 - R2$
- Remarque :
  - la relation R' est le résultat de la différence des tuples des relations R1 et R2.
  - R1 et R2 doivent être de même schéma sinon la différence est impossible. R' est de même schéma.
  - Cet opérateur n'est pas commutatif.

3.1.10.6 Produit (Cartésien)

- Le produit cartésien est un opérateur binaire permettant de produire une relation R' avec tous les tuples appartenant à R1 combinés à chacun des tuples de R2. Le schéma de R' est la concaténation des schémas de R1 et R2.
- Notation : $R' = R1 \times R2$
- Remarque :
  - la relation R' est le résultat du produit cartésien des tuples des relations R1 et R2.
  - le schéma de R' est la concaténation des schémas de R1 et R2 dans cet ordre.
  - Cet opérateur est commutatif modulo le schéma.

3.1.10.7 thêta-Jointure

- La théta-jointure est l'opération qui consiste à appliquer à la fois un produit cartésien et une sélection. Il existe de nombreuses formes de cet opérateur, nous pouvons en rester dans un premier temps à l'équijointure. La condition porte sur l'égalité d'attributs de R1 et R2.
- Notation : $R' = R1 \infty_{condition} R2$
- Cet opérateur, non minimal, peut être réalisé par combinaison d'opérateurs comme suit : $R' = \sigma_{condition}(R1 \times R2)$.
  Cette solution est relativement peu intéressante car la combinatoire du produit peut être très grande, si la cardinalité de R1 est c1 et celle de R2 est c2 alors le produit à pour cardinalité c1*c2. L'intégration d'index est une solution pour améliorer cette performance, car on peut alors tirer partie de l'ordonnancement des 2 relations sur l'attribut de jointure permettant ainsi une réduction logarithmique passant de l'ordre (c1*c2) à un ordre c1+c2.

3.2 Mise en œuvre du noyau.

Ayant réalisé le noyau de votre SGBD, vous pouvez réaliser des tests programmatiques permettant de vérifier le fonctionnement de votre moteur d'exécution.

3.2.1 Exemple à mettre en œuvre

On veut mettre en place une base de données simple, constituée de 3 relations, dont la définition est la suivante :

PRODUIT = DENOMINATION:STRING, ENERGIE:INT, PROTEINE:DOUBLE

PRODUCTEUR = ID_PRODUCTEUR:INT, NOM:STRING, PREMON:STRING

PRODUCTION = PRODUCTEUR:INT, PRODUIT:STRING, QUANTITE:INT, PRIX:DOUBLE

La dénomination est la clé primaire de la relation PRODUIT à laquelle on associe 2 informations (énergie et protéine). De même ID_PRODUCTEUR est la clé primaire de la relation PRODUCTEUR. La relation PRODUCTION possède une clé primaire double composée du PRODUCTEUR et du PRODUIT. On nommera "VENTE" la BD comportant ces relations

Pour ce faire, énoncer les différentes techniques programmatiques (des classes différentes nommées ou anonymes, des objets, ...) pouvant être mise en œuvre pour construire un programme qui concrétise cette base de données. On considérera que les relations contiennent les informations suivantes :

PRODUIT		
DENOMINATION	**ENERGIE**	**PROTEINE**
"Epeautre"	1340	11.5
"Orge"	1430	11.0
"Avoine"	1530	12.5
"Millet"	1510	10.5
"Mais"	1498	9.0
"Riz"	1492	7.5
"Seigle"	1323	8.8
"Ble"	1342	11.5

PRODUCTEUR		
ID_PRODUCTEUR	**NOM**	**PRENOM**
1	"BERNARD"	"ALAIN"
2	"PERRIER"	"CHARLES"
3	"LABBE"	"CAROLINE"

PRODUCTION			
PRODUCTEUR	**PRODUIT**	**QUANTITE**	**PRIX**
1	"Epeautre"	10	3.56
2	"Avoine"	150	2.37
3	"Mais"	25	9.12
3	"Epeautre"	120	2.99

Dans ce cadre on souhaite pouvoir :

- Afficher chacune de ces relations,
- Faire une projection conservant la totalité des tuples de PRODUCTION sur l'ensemble des attributs sauf le prix.

Projection Production ...		
PRODUCTEUR	**PRODUIT**	**QUANTITE**
1	"Epeautre"	10
2	"Avoine"	150
3	"Mais"	25
3	"Epeautre"	120

Dans un autre temps on souhaite :

- Afficher cette projection dans l'ordre des noms des produits.

Selection Production ...		
PRODUCTEUR	**PRODUIT**	**QUANTITE**

2	"Avoine"	150
1	"Epeautre"	10
3	"Epeautre"	120
3	"Mais"	25

- On veut pouvoir construire la sélection des productions d'épeautre.

Jointure Producteur Production ...			
PRODUCTEUR	**PRODUIT**	**QUANTITE**	**PRIX**
1	"Epeautre"	10	3.56
3	"Epeautre"	120	2.99

- Enfin on veut réaliser la jointure selon l'identification du producteur entre les relations PRODUCTEUR et PRODUCTION.

Produit Producteur Production							
IKEY	ID_PRODUCTEUR	NOM	PRENOM	PRODUCTEUR	PRODUIT	QUANTITE	PRIX
0	1	BERNARD	ALAIN	1	Epeautre	10	3.56
1	1	BERNARD	ALAIN	2	Avoine	150	2.37
2	1	BERNARD	ALAIN	3	Mais	25	9.12
3	1	BERNARD	ALAIN	3	Epeautre	120	2.99
4	2	PERRIER	CHARLES	1	Epeautre	10	3.56
5	2	PERRIER	CHARLES	2	Avoine	150	2.37
6	2	PERRIER	CHARLES	3	Mais	25	9.12
7	2	PERRIER	CHARLES	3	Epeautre	120	2.99
8	3	LABBE	CAROLINE	1	Epeautre	10	3.56
9	3	LABBE	CAROLINE	2	Avoine	150	2.37
10	3	LABBE	CAROLINE	3	Mais	25	9.12
11	3	LABBE	CAROLINE	3	Epeautre	120	2.99

3.2.2 Réalisation d'un programme de test

Dans un premier temps vous pouvez construire un programme de test qui construit une base de données, la peuple et effectue quelques requêtes classiques et ce de façon programmatique.

Une solution de mise en œuvre consiste en la construction d'une classe par relation et d'une pour la BD. Chaque relation de la BD est nécessairement une relation de nature "stateFull" dont la structure est généralisable et comporte la description du schéma de la relation sous la forme d'une classe et la description d'un tuple de cette relation sous la forme d'une classe aussi. La solution proposée ci-après n'est pas l'unique possible et ne se veut pas une injonction. Pour plus de clarté on identifie la notion d'initialisation d'une relation qui permet d'isoler la séquence d'instructions peuplant la relation et ceci soit par en extension soit par calcul. Dans le cas de l'exemple nous devrons initialiser les 3 relations PRODUIT, PRODUCTEUR et PRODUCTION.

```
interface Initializator<T extends State Full Relation>{
   /**
    * le code d'initialisation de la relation
    * @param bd la bd de la relation
    * @param relation la relation à initialiser
    */
   public void init(BD bd, T relation);
}
```

Le test ce décompose en 3 phases :

1. Ouverture ou création de la BD, dans le cas de la création on utilisera les initialisations décrites précédemment.
2. Construction des requêtes que l'utilisateur souhaite réaliser. Une requête est une expression en termes des opérateurs relationnels définis précédemment et mettant en jeu les relations de la BD. La notion de requête peut être repoussée à plus tard lorsqu'on

évoquera le langage SQL, auquel cas on inclura directement la construction de l'expression.

3. Exécution des requêtes, on optera pour une impression systématique du résultat du calcul de l'expression relationnelle.

```
/**
 * @author morat
 */
public class Test{
//-------------------------------------------------------------------------------------
//- initialisation pour le test
//-------------------------------------------------------------------------------------
   static Initializator<RELATION> init_RELATION = new Initializator<RELATION>(){
      @Override public void init(BD bd, RELATION relation){
         for(int i = 0, num; i<...; i++)
            relation.add(...);
      }
   };
   /**
    * @param args
    * @throws Exception
    */
   public static void main(String[] args) throws Exception {
      /* Ouverture ou création d'une BD */
      /* création des requêtes devant être exécutées */
      /* exécution des requêtes crées */
      /* fermeture de la BD */
   }
}
```

4 Mémorisation externe des tables

Dans un premier nous avons conservé les tables des relations en mémoire. Il est souhaitable de pouvoir les stocker sur un support rémanent. De nombreux formats sont envisageables : format textuel, format primitif, format de sérialisation, voire des combinaisons. Quoiqu'il en soit, si on utilise un fichier externe l'information principale (l'ensemble des tuples) ne devra pas être chargée en totalité en mémoire. On peut donc envisager d'utiliser des fichiers dits à accès direct permettant d'acquérir qu'un seul tuple par sa position dans l'ensemble des tuples sauvegardés. Il faut alors disposer d'une association qui lie la clé primaire et la position dans le fichier pour réaliser cet accès.

5 Fonctions de calcul

Les fonctions arithmétiques permettent de faire différents calculs portant sur les tuples d'une relation. Les plus classiques caractérisent des cumuls dont l'objet est de calculer des accumulations d'informations issues de l'ensemble des tuples de la relation à laquelle ils s'appliquent :

1. Nombre : celle-ci est relativement triviale et restitue simplement le nombre de tuples de la relation
2. Celles-ci s'applique sur un attribut dont la nature est numériques et support l'addition et la division
   a. Somme : la somme des valeurs de l'attribut sélectionné
   b. Maximum : le maximum des valeurs de l'attribut sélectionné

c. Minimum : le minimum des valeurs () de l'attribut sélectionné
d. Moyenne : la moyenne des valeurs de l'attribut sélectionné

Par exemple, appliqué à la relation produit, on souhaite calculer le nombre de produits répertoriés, la somme énergétique de l'ensemble de ces produits, celui qui a la valeur protéinique minimale et celui qui a celle maximale ainsi que la moyenne des valeurs énergétiques de l'ensemble des ces produits. Dans ce cas le résultat proposé sera :

```
Nombre de produits=8
Total énergétique=11465
Valeur protéinique min=7.5
Valeur protéinique max=12.5
Moyenne énergétique=1433
```

Il faut envisager que ceci peut être étendu par d'autres fonctions que nous n'avons pas citées ici.

6 Compilation d'une BD

Proposez un mécanisme qui permette de passer d'une représentation xml de votre schéma de BD vers la mise en œuvre que vous avez choisie précédemment.

```xml
<?xml version="1.0"  encoding="UTF-8"?>
<!DOCTYPE encyclopedie SYSTEM .../BD.dtd">
<BD name="Vente">
    <Relation name="Producteur" infile ...>
        <Attribut name="ID_PRODUCTEUR" type="INT"/>
        <Attribut name="NOM" type="STRING"/>
        <Attribut name="PRENOM" type="STRING"/>
    </Relation>
    <Relation name="Produit" ...>
        <Attribut name="DENOMINATION" type="STRING"/>
        <Attribut name="Energie" type="INT"/>
        <Attribut name="Proteine" type="DOUBLE"/>
    </Relation>
    <Relation name="Production" ...>
        <Attribut name="PRODUCTEUR" type="INT"/>
        <Attribut name="PRODUIT" type="STRING"/>
        <Attribut name="QUANTITE" type="INT"/>
        <Attribut name="PRIX" type="DOUBLE"/>
    </Relation>
</BD>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT BD (Relation+)>
    <!ATTLIST BD
        name CDATA #REQUIRED
    >
<!ELEMENT Relation (Attribut+)>
    <!ATTLIST Relation
        name CDATA #REQUIRED
    >
<!ELEMENT Attribut EMPTY>
    <!ATTLIST Attribut
        name NMTOKEN #REQUIRED
        type CDATA #REQUIRED
    >
```

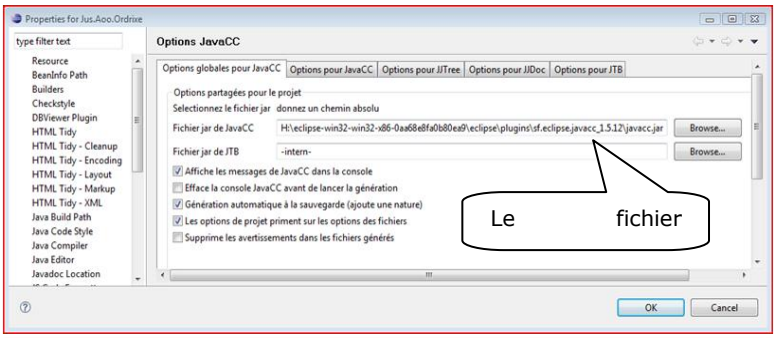
Ou sous la forme d'un schéma xml :

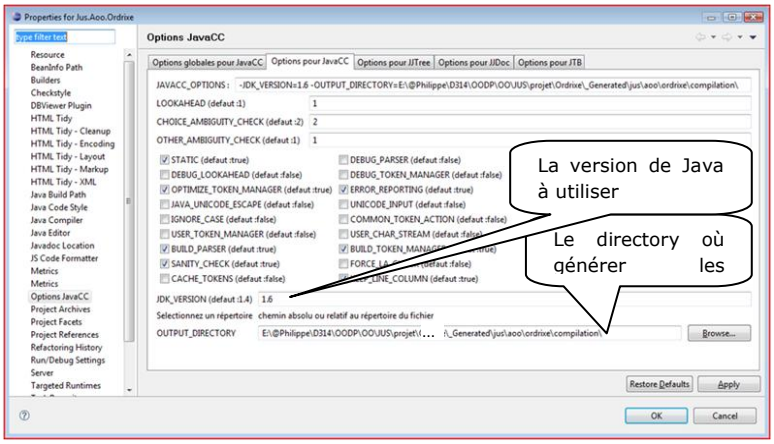
```
<?xml version="1.0"?>
<BD
    xmlns="http://www.w3schools.com"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="../../jus/aoo/sgbd/compilation/BD.xsd"
    name="Vente">
    <Relation name="Producteur" infile="False" autokey="False">
         <Attribut name="ID_PRODUCTEUR" type="INT"/>
         <Attribut name="NOM" type="STRING"/>
         <Attribut name="PRENOM" type="STRING"/>
    </Relation>
    <Relation name="Produit" infile="False" autokey="False">
         <Attribut name="DENOMINATION" type="STRING"/>
         <Attribut name="Energie" type="INT"/>
         <Attribut name="Proteine" type="DOUBLE"/>
    </Relation>
    <Relation name="Production" infile="False" autokey="True">
         <Attribut name="PRODUCTEUR" type="INT"/>
         <Attribut name="PRODUIT" type="STRING"/>
         <Attribut name="QUANTITE" type="INT"/>
         <Attribut name="PRIX" type="DOUBLE"/>
    </Relation>
</BD>
```

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="BD">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="Relation" maxOccurs="unbounded" minOccurs="0">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="Attribut" maxOccurs="unbounded" minOccurs="0" >
        <xs:complexType>
         <xs:simpleContent>
          <xs:extension base="xs:string">
           <xs:attribute type="xs:string" name="name" use="optional"/>
           <xs:attribute type="xs:string" name="type" use="optional"/>
          </xs:extension>
         </xs:simpleContent>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
      <xs:attribute type="xs:string" name="name" use="optional"/>
      <xs:attribute type="xs:string" name="infile" use="optional"/>
      <xs:attribute type="xs:string" name="autokey" use="optional"/>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
   <xs:attribute type="xs:string" name="name"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

7 L'acquisition d'une requête SQL

Ceci se fera via une lecture sur un support quelconque. Pour cela nous allons utiliser un générateur d'analyseur syntaxique qui s'intègre aisément à Java.





7.1.1 Configuration de JavaCC

Cette description se trouve dans un fichier qui peut se nommer « Reader.jj ». Si vous avez installé le plugin JavaCC (http://sourceforge.net/projects/eclipse-javacc) dans votre environnement Eclipse, la mise à jour et la génération des classes nécessaires se fera automatiquement du moment où vous aurez fixé les options dans les propriétés de votre projet.

Celles-ci permettent de déterminer le fichier jar à utiliser pour l'analyse de la grammaire, ainsi que le lieu où vous souhaitez engendrer les classes de l'analyseur.

On vous conseille d'utiliser un autre directory source dans votre projet pour isoler correctement les différentes parties de votre application.

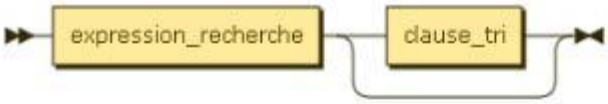
7.1.2 Prémisse de la grammaire

Le Reader permet de construire une requête. La classe Reader comportera principalement 1 méthode permettant la construction d'une requête qui pourra être exécutée ou voire stockée pour un usage ultérieur.

Le document sparql11-query.pdf est un diagramme de la syntaxe de SPARQL qui est un proche voisin de SQL et sql-mysql-minimal.pdf est un document réalisé par Michel Cartereau d'AgroParisTech décrivant le langage SQL dont la partie diagramme syntaxique est donnée ci-après.
Nous rappelons que notre objectif est notablement plus restreint que ce qui est proposé dans ces diagrammes et que seule la possibilité d'utiliser les opérateurs proposés dans ce document est nécessaire, il est donc utile de conserver qu'une version épurée du diagramme.

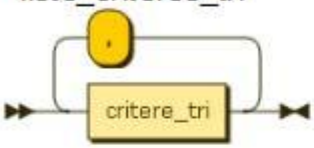
7.1.3 Diagramme syntaxique SQL

interrogation



clause_tri

liste_criteres_tri

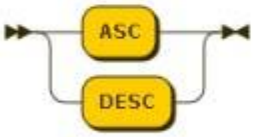


critere_tri

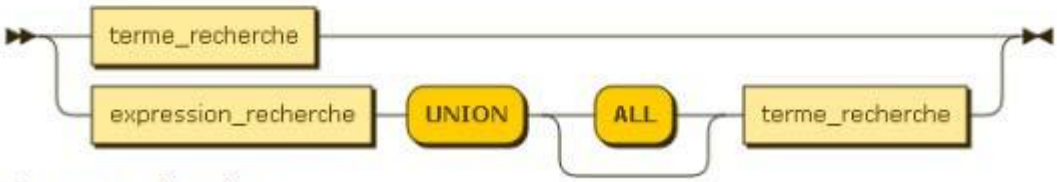
resultat_recherche

indication_interclassement

type_classement



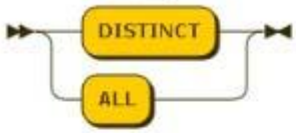
expression_recherche



terme_recherche

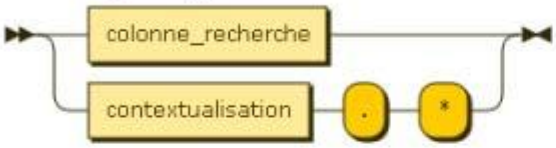
specification_recherche

distinction_valeurs

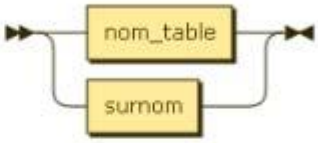


specification_resultats_recherche

element_liste_recherche

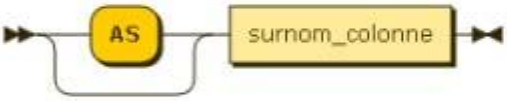


contextualisation



colonne_recherche

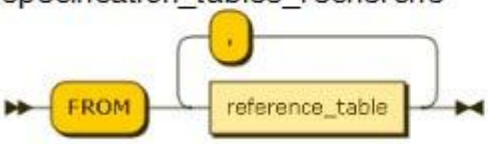
definition_surnom_colonne



surnom_colonne

specification_donnees_recherche

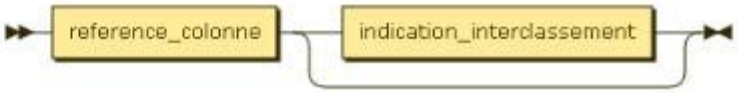
specification_tables_recherche



liste_references_colonne_groupe

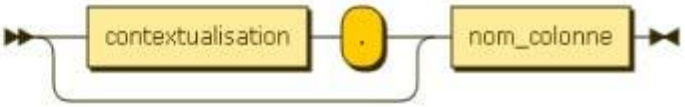
specification_condition_recherche

reference_colonne_groupe



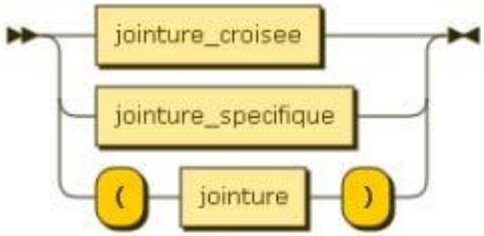
specification_regroupement

reference_colonne



specification_filtrage

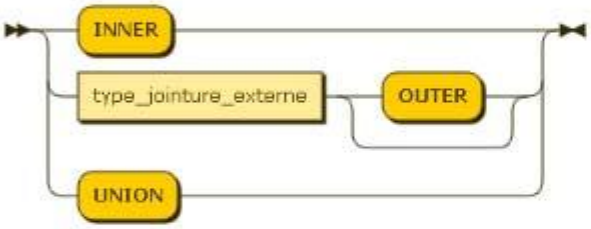
jointure



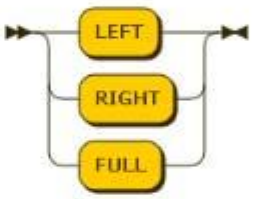
jointure_croisee

jointure_specifique

type_jointure_specifique

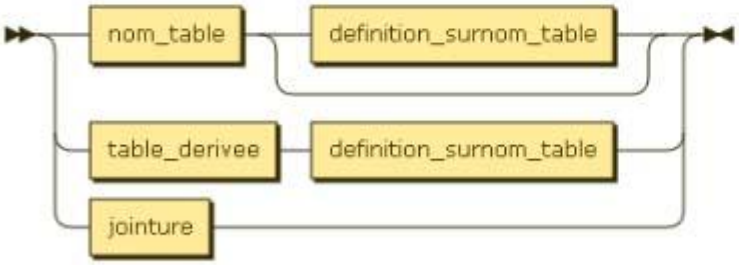


type_jointure_externe



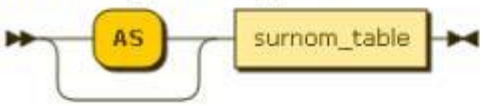
critere_jointure

reference_table



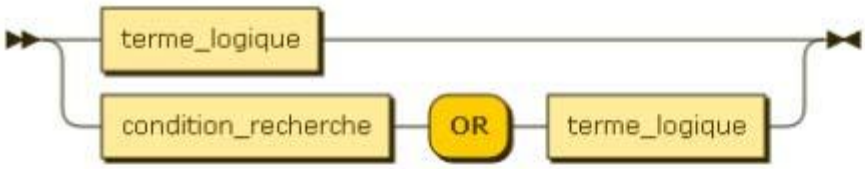
table_derivee

definition_surnom_table

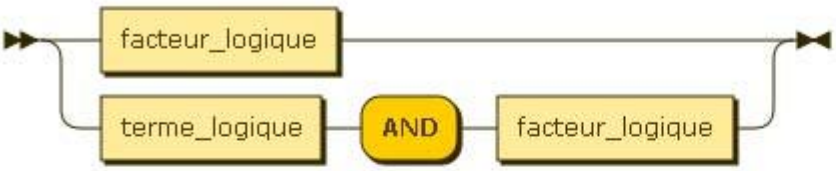


surnom_table

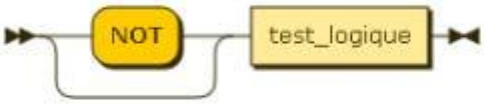
condition_recherche



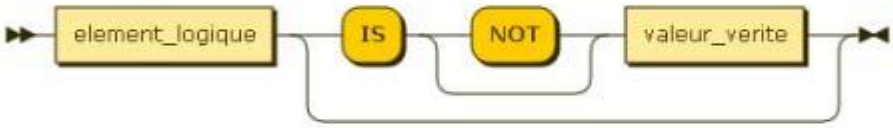
terme_logique



facteur_logique

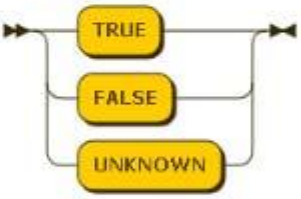


test_logique

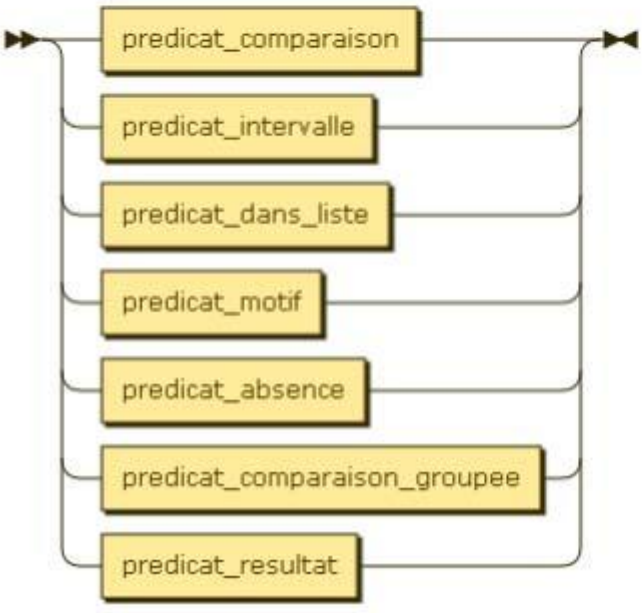


element_logique

valeur_verite

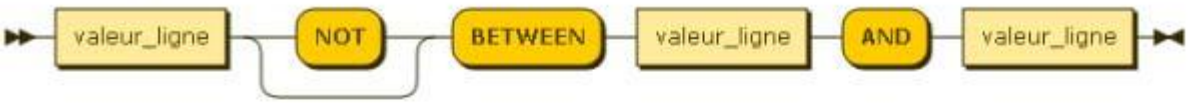


predicat

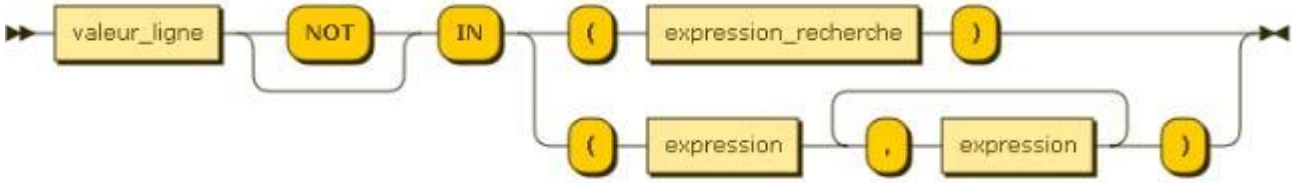


predicat_comparaison

predicat_intervalle

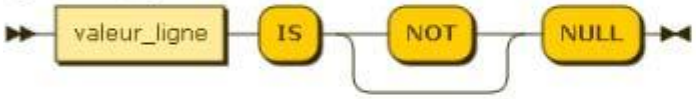


predicat_dans_liste

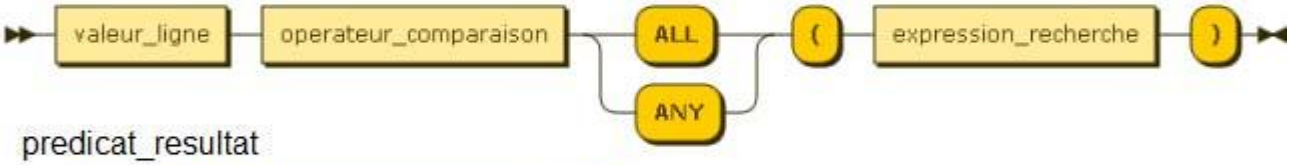


predicat_motif

predicat_absence

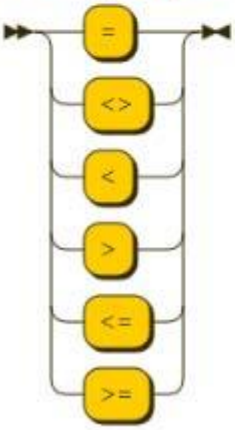


predicat_comparaison_groupee

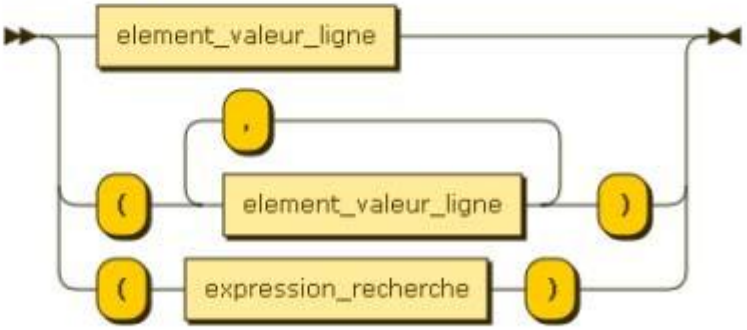


predicat_resultat

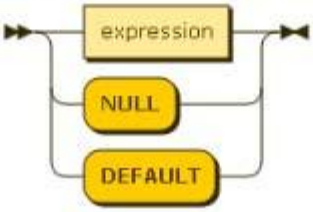
operateur_comparaison



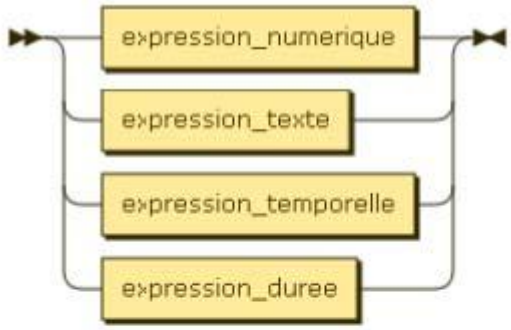
valeur_ligne



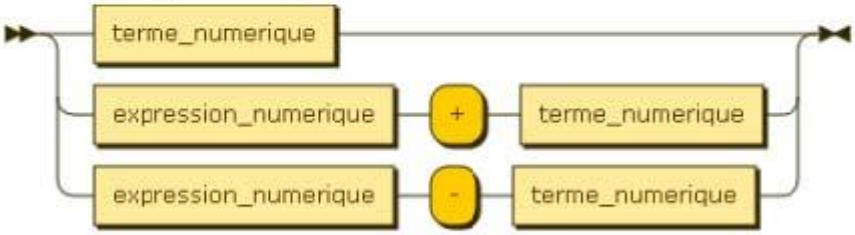
element_valeur_ligne



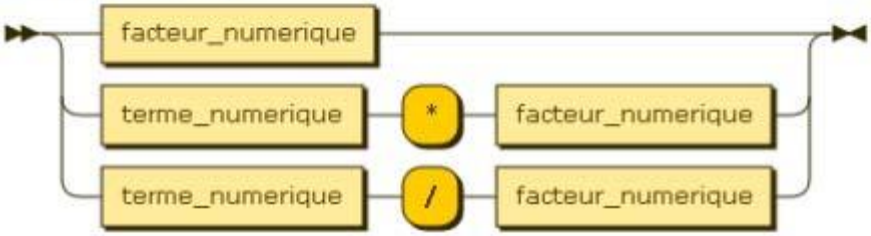
expression



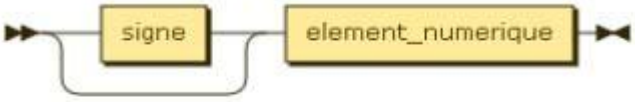
expression_numerique



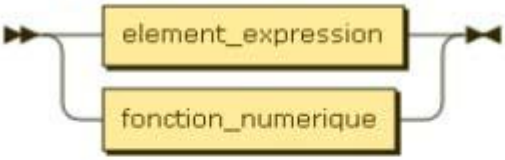
terme_numerique



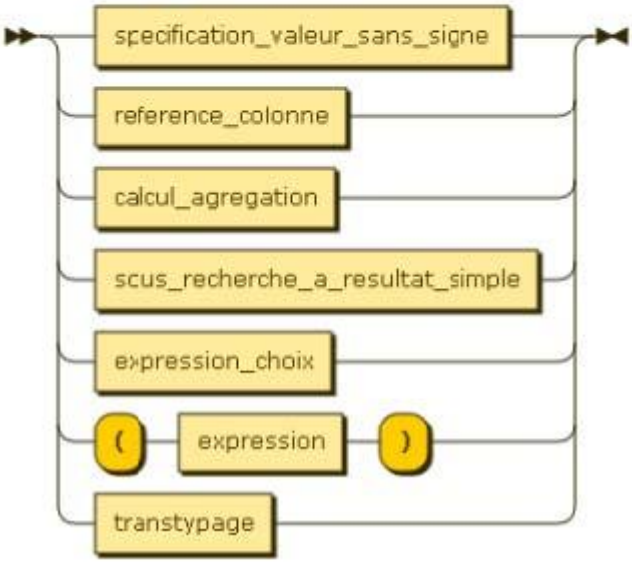
facteur_numerique



element_numerique



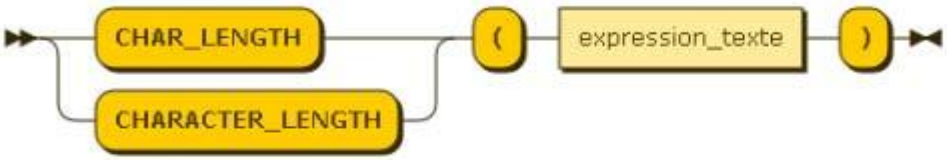
element_expression



fonction_numerique

expression_longueur

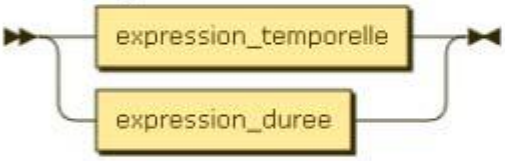
expression_longueur_texte



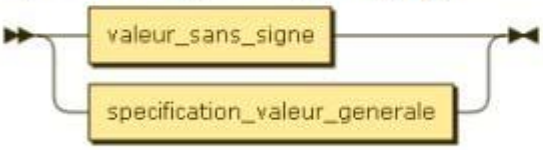
expression_position

expression_extraction

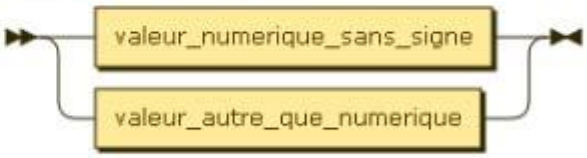
source_extraction



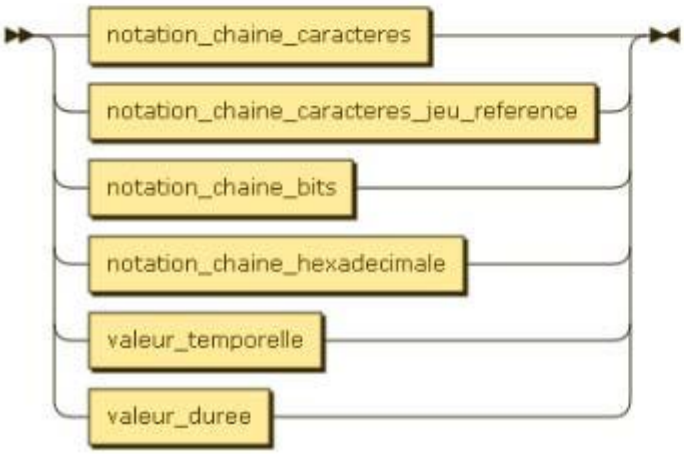
specification_valeur_sans_signe



valeur_sans_signe

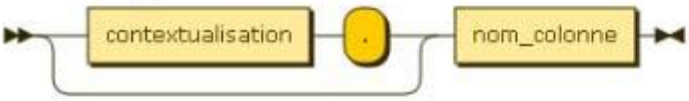


valeur_autre_que_numerique

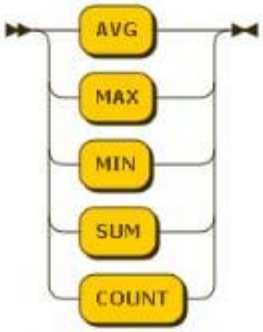


specification_valeur_generale

reference_colonne

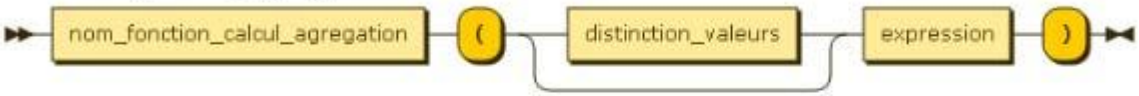


nom_fonction_calcul_agregation

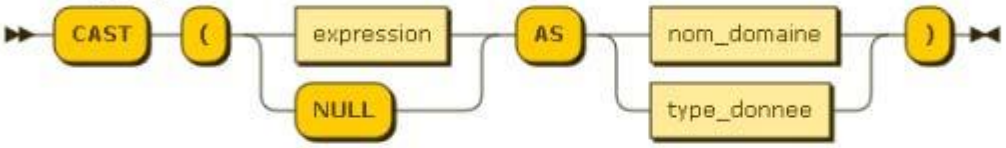


calcul_agregation

fonction_calcul_agregation



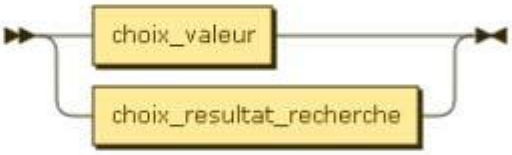
transtypage



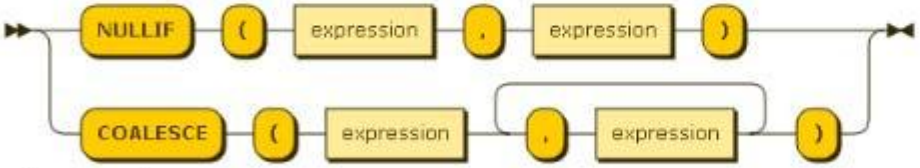
sous_recherche_a_resultat_simple

expression_choix

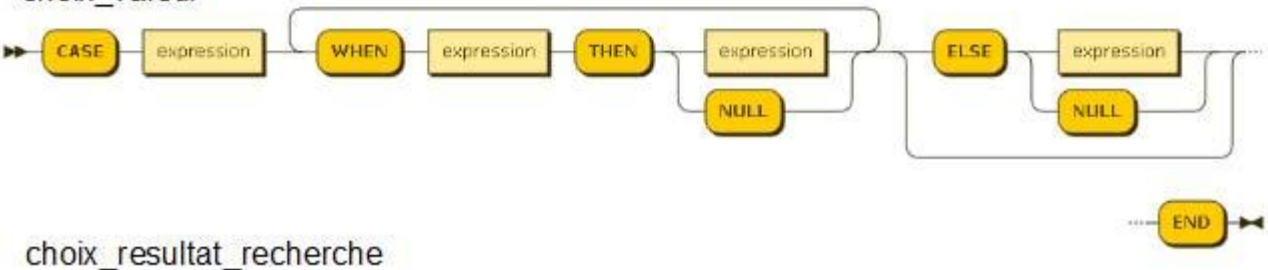
specification_choix



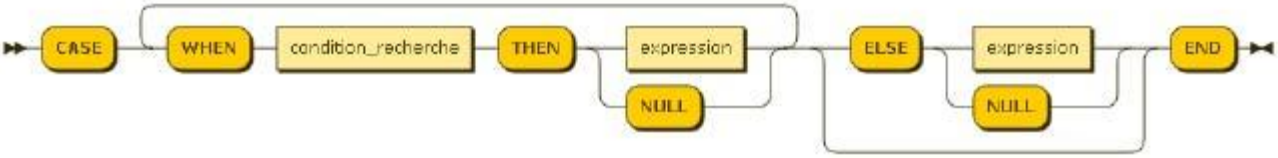
choix_abrege



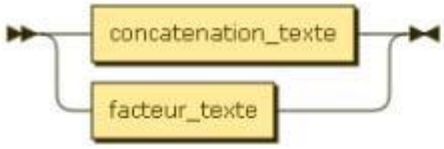
choix_valeur



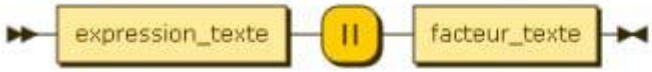
choix_resultat_recherche



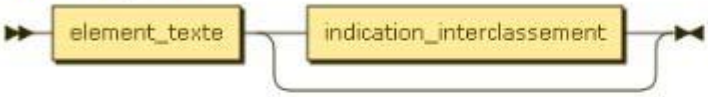
expression_texte



concatenation_texte

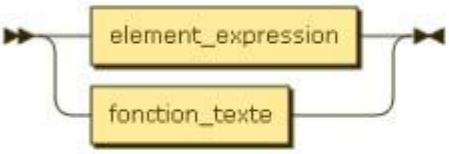


facteur_texte

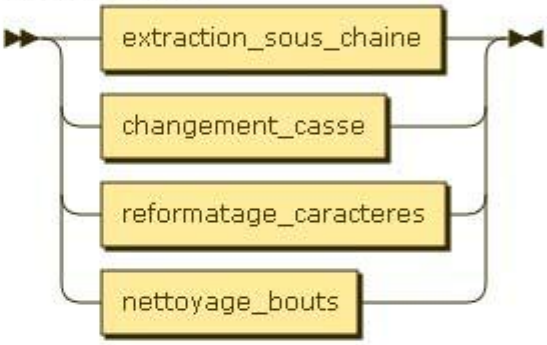


indication_interclassement

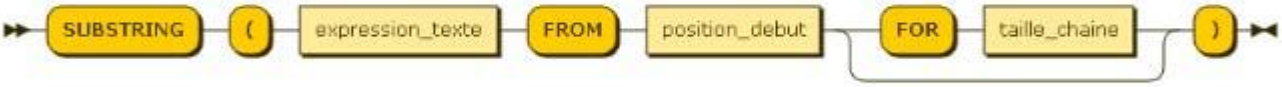
element_texte



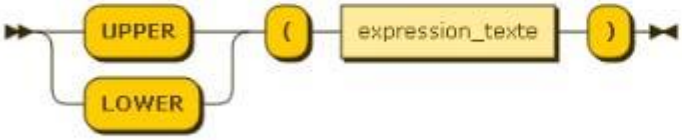
fonction_texte



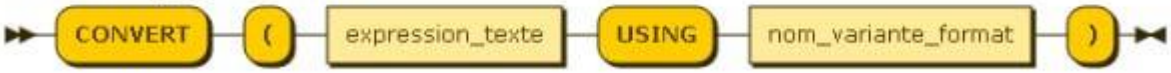
extraction_sous_chaine



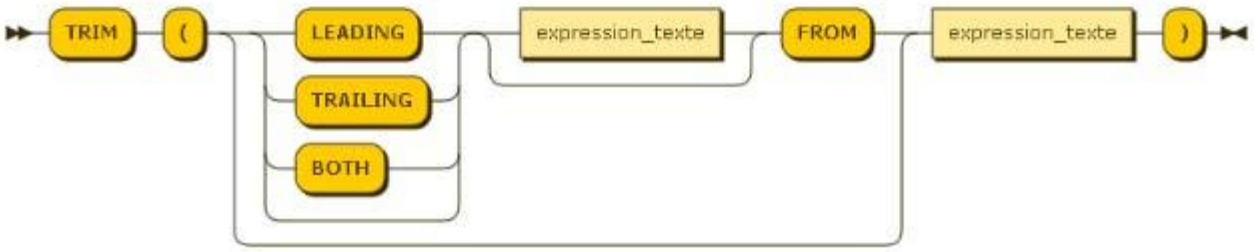
changement_casse



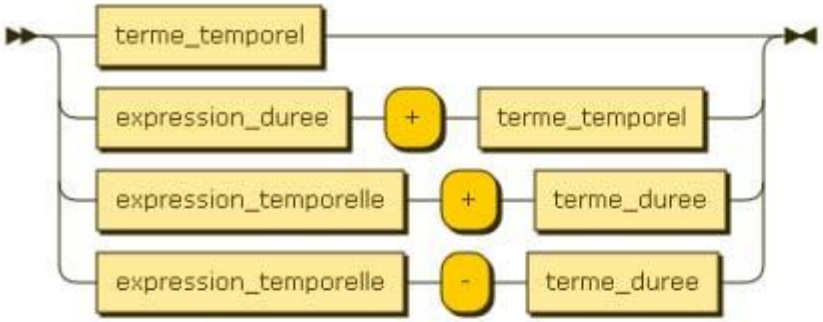
reformatage_caracteres



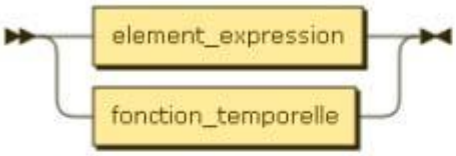
nettoyage_bouts



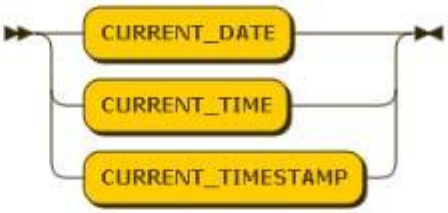
expression_temporelle



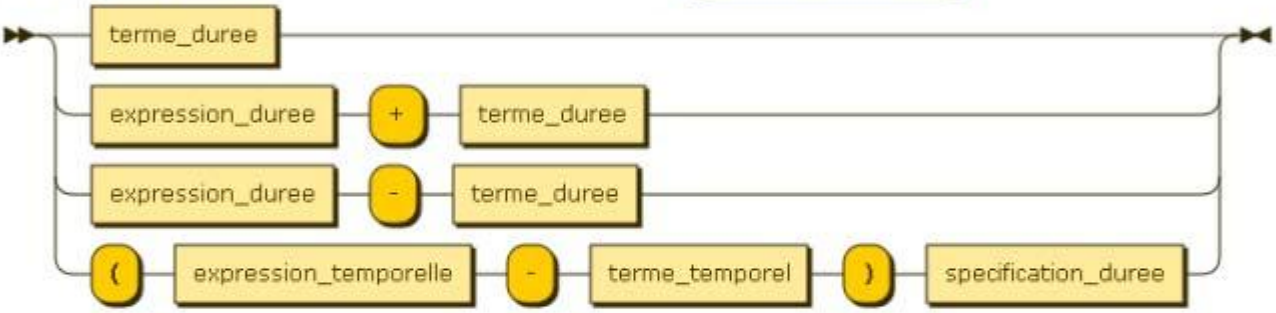
terme_temporel



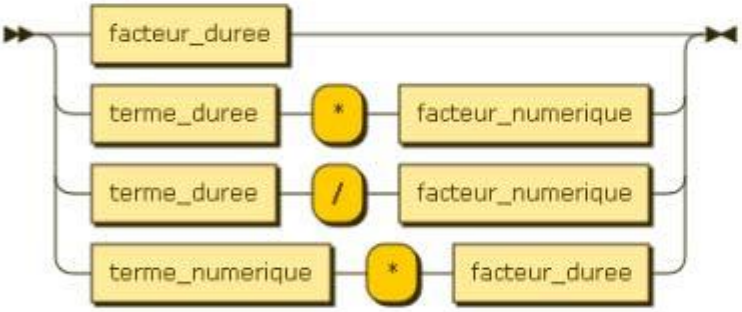
fonction_temporelle



expression_duree

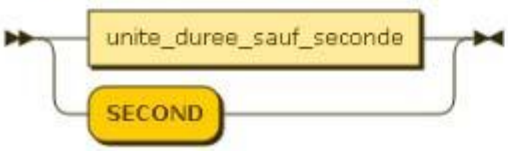


terme_duree



facteur_duree

specification_duree



unite_duree_sauf_seconde