# Projet d'intelligence artificielle et de base de données :

Extraction de règles d'association : algorithme APriori


Java Environment for Mining Association Rules

**DARNET Camille**
**VOURIOT Thierry**
DESS Informatique


*sous la direction de Mr Jerzy KORCZAK : jjk@dpt-info.u-strasbg.fr*

# Sommaire :

# 1. Introduction

Le stockage de données en provenance de divers domaines (ex : informations sur des clients, numérisation de textes, catalogues en ligne, …) est devenu essentiel dans le monde informatique.

C'est pour cela qu'il existe dorénavant un nombre impressionnant de SGBD (Système de gestion de base de données) fournissant de plus en plus d'outils permettant le traitement des données.

Parmi les principaux acteurs et logiciels du marché, de grandes sociétés ainsi que de nombreux développeurs indépendants (Oracle, Microsoft, Mysql, …) ont bien compris le potentiel de ces applications et proposent chacun des solutions plus performantes chaque année.

La diversité de l'offre bien que bénéfique pour la qualité et le coût présente un inconvénient majeur pour les développeurs souhaitant accéder aux données stockées ; le code à écrire pour accéder à un SGBD est très souvent propriétaire. Pour pallier à ce problème SUN fournit une API Java permettant aux développeurs d'écrire un code unique qui permettra d'accéder à n'importe quelle base de données supportant JDBC.

Après une première application réalisée dans le cadre du module d'intelligence artificielle consistant à extraire des règles d'association à partir de fichiers contenant des données à traiter, notre tâche a été dans un premier temps d'améliorer ce logiciel pour qu'il puisse s'interfacer avec une base de données relationnelle, et ensuite d'effectuer une série de tests sur une base de données contenant les informations d'une banque tchèque.

# 2. Rappels sur les règles d'association

Avec l'apparition des bases de données, la question de l'extraction de connaissances sous la forme de règles d'association a été de plus en plus étudiée du fait de l'énorme potentiel de ses applications (marketing, astronomie, botanique, médecine …).

Etant donnée une base de données constituée de transactions composées de un ou plusieurs articles ou items, on cherche à en déduire un ensemble de règles d'association sous la forme $X \rightarrow Y$, où X et Y sont des ensembles d'items de la base de donnée.
Un exemple d'une telle règle dans une base de donnée pourrait être le suivant : 98% des clients qui s'équipent en pneus et en accessoires pour automobiles se préoccupent également des services offerts pour leurs autos.

La résolution du problème se découpe en deux parties :

- Recherche des items fréquents :
  Trouver tous les ensembles d'items fréquents telles que leur représentation soit significative au sein de la base. Certaines règles d'association ne sont pas intéressantes car elles ne concernent qu'une faible part des objets.
- Recherche des règles d'association :
  Une fois les items fréquents trouvés, les règles d'association peuvent être déduites. Les règles d'association doivent vérifiées une proportion importante d'objets pour exprimer une association cohérente.

L'extraction ne va concerner que les règles d'association (valides) dont le support est supérieur ou égal au seuil (car les règles utiles concernent une population "importante") et dont la confiance est supérieure ou égale au seuil de confiance minimum (car on ne veut que les règles statistiquement significatives). Les seuils sont définis par l'utilisateur selon la nature des données analysées.

# 3. Outils et environnement de programmation

Le choix du langage de programmation fût très rapidement effectué. En effet, le langage Java a été choisi pour de nombreuses raisons :

- nous avions déjà une bonne connaissance de ce langage
- portabilité des applications sur toutes les plate-formes (Windows, Unix, Linux, Macintosh, ….).
- contrairement aux idées reçues, Java est souvent plus rapide que d'autres langages de programmation.
- programmation orientée objet qui permet un meilleur découpage de l'application.
- API Swing permettant la création d'interfaces graphiques très rapidement.
- réutilisation de package déjà développé dans d'autres applications.
- API JDBC permettant la connexion à différents gestionnaires de base de données.

De nombreux outils de développement sont disponibles pour Java, et ils ont l'avantage d'être très souvent en open source et/ou gratuit. Nous avons principalement utilisé deux environnements suivant nos préférences :

- Borland JBuilder 9.0 : http://www.borland.fr/jbuilder/
- NetBeans 3.5 : http://www.netbeans.org/

# 4. JDBC (Java Database Connectivity)

## Concepts et architecture

JDBC est une API Java (ensemble de classes et d'interfaces défini par SUN et les acteurs du domaine des BD) permettant d'accéder aux bases de données à l'aide du langage Java via des requêtes SQL. Cette API permet d'attendre de manière quasi-transparente des bases Sybase, Oracle, Informix, MySql, MS Access ... avec le même programme Java JDBC.

En fait cette API fournit des spécifications de ce que doit implanter un constructeur de BD pour que celles ci soient interrogeables par JDBC. De ce fait dans la programmation JDBC on utilise essentiellement des références d'interface (Connection, Statement, ...).

Sun et les constructeurs de BD se sont charger de fournir (vendre ou donner) des classes qui implémentent les interfaces précités qui permettent de soumettre des requêtes SQL et de récupérer le résultat.

*fig 1 : Architecture de JDBC*

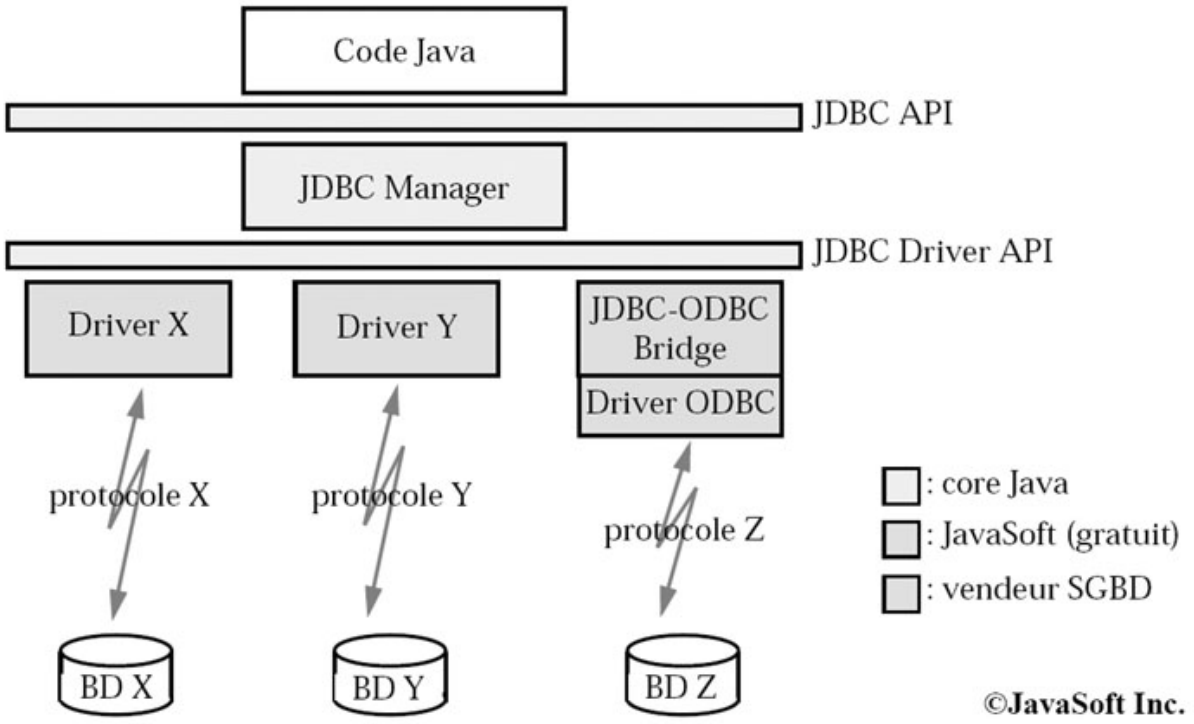
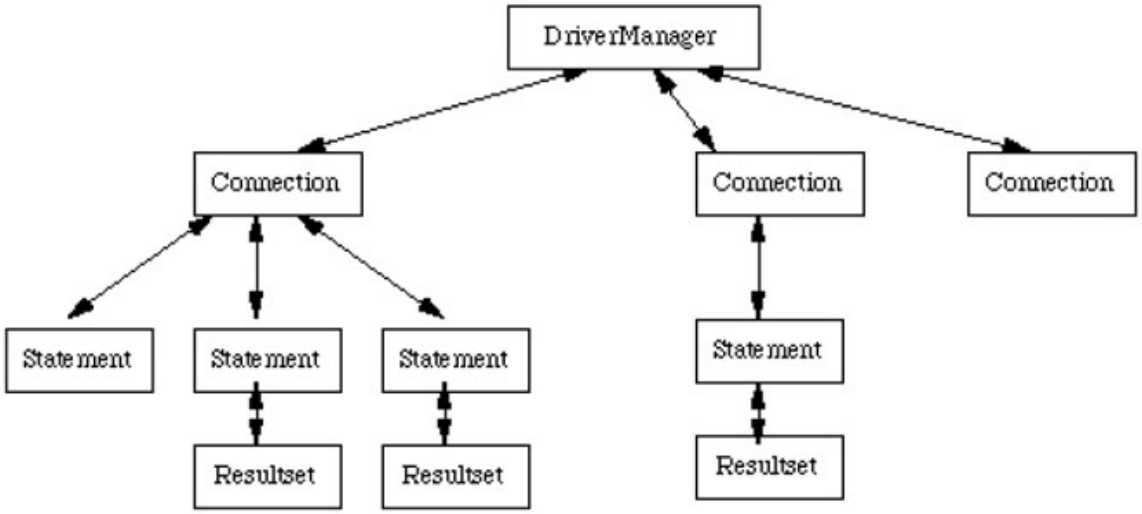


*fig 2 : Interfaces de JDBC*




©JavaSoft Inc.

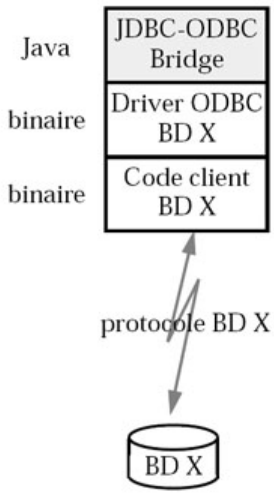
## Pilotes (drivers)

L'ensemble des classes qui implémentent les interfaces spécifiées par JDBC pour un gestionnaire de bases de données particulier est appelé un pilote JDBC. Les protocoles d'accès aux BD étant propriétaires il y a donc plusieurs drivers pour atteindre diverses BD.

Parmi les interfaces, l'une d'entre elles, l'interface Driver, décrit ce que doit faire tout objet d'une classe qui implémente l'essai de connexion à une base de données. Entre autre, un tel objet doit obligatoirement s'enregistrer auprès du DriverManager et retourner en cas de succès un objet d'un classe qui implémente l'interface Connection.

Le premier pilote (développé par SUN) est un pilote jdbc:odbc. La liste des pilotes disponibles se trouve à : http://industry.java.sun.com/products/jdbc/drivers

Les pilotes sont classés en quatre types :

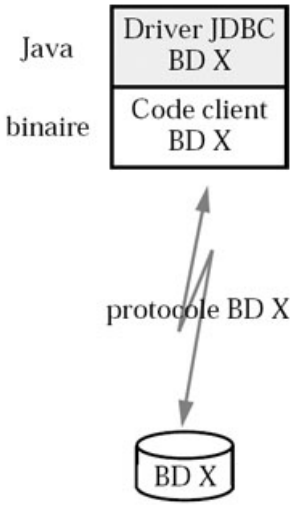



**Driver JDBC Type 1 :**

JDBC-ODBC Bridge :

Le driver accède à un SGBDR en passant par les drivers ODBC (standard Microsoft) via un pont JDBC-ODBC :

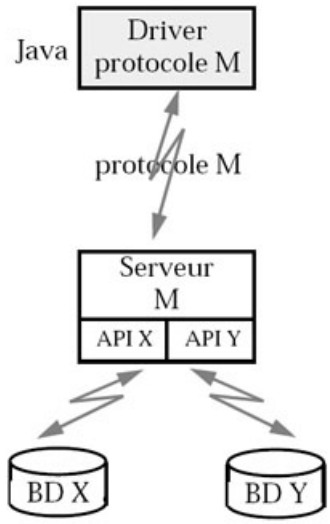
les appels JDBC sont traduits en appels ODBC, presque tous les SGBDR sont accessibles (monde Windows)




**Driver JDBC Type 2 :**

Native-API Partly Java Driver :

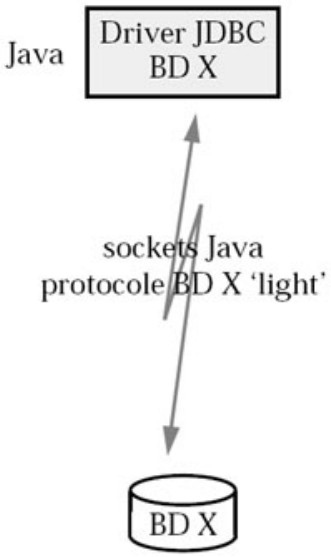
Fait appel à des fonctions natives (non Java) de l'API du SGBDR, gère des appels C/C++ directement avec la base.




**Driver JDBC Type 3 :**

Net Protocol All-Java Driver

Interagit avec une API réseau générique (Sockets) et communique avec une application intermédiaire (*middleware*) sur le serveur qui accède par un moyen quelconque (par exemple JDBC si écrit en Java) aux différents SGBDR portable car entièrement écrit en Java




**Driver JDBC Type 4 :**

Native Protocol All-Java Driver

Interagit avec la base de données via des *sockets.*
Driver généralement fourni par l'éditeur.

## Structure d'un programme JDBC

Un code JDBC est de la forme :

- recherche et chargement du driver approprié à la base de données.
- établissement de la connexion à la base de données.
- construction de la requête SQL.
- envoi de cette requête et récupération des réponses.
- parcours des réponses.

Ce qui donne par exemple :

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection conX = DriverManager.getConnection(...);
Statement stmt = conX.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a, b, c ... FROM
... WHERE ...");

while (rs.next()) {
     ...
     // traitement
}
```

# 5. Développement

## Connexion et ouverture d'une BDD

Pour la connexion, l'ouverture et la lecture d'une base de données une classe Java a été spécialement conçue en utilisant l'API JDBC. Cette classe propose un certain nombre de méthodes permettant de faciliter les traitements avec la base de données.

Quelques méthodes de la classe `Database` :

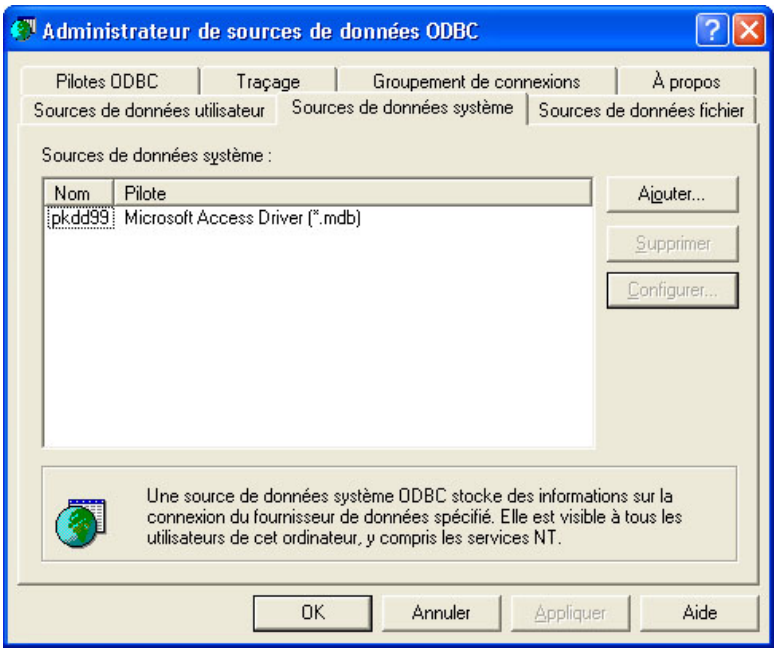
```
// Constructeur qui charge un pilote JDBC donné en paramètre
Database(String driver) ;
// Charge un pilote JDBC
void loadDriver(String driver);
// Permet de se connecter à une base de données spécifique
void connect(String url1,String user,String pass,int type);
// Execute une requète SQL de type SELECT
void executeSQLQuery(String sql);
// Execute une requète SQL de type INSERT, UPDATE, DELETE ou
CREATE TABLE
void executeSQLUpdate(String sql);
// Retourne le ResultSet contenant le résultat de la dernière
requête SQL
ResultSet getResultSet() ;
// Retourne la liste des tables d'une base de données
String[] getTablesNames() ;
// Retourne la liste des attributs (avec leurs types) d'une table
String[][] getAttributes(String tableName) ;
// Se déconnecte de la base de données courante
void disconnect() ;
```

*Chargement du driver :*

La première étape pour se connecter à une base de données consiste à charger un driver JDBC en fonction du type de SGBD (Access, Oracle, Mysql,…).
Dans Merja, trois drivers sont inclus par défaut :

- `sun.jdbc.odbc.JdbcOdbcDriver`, pont JDBC-ODBC fournit dans le JDK par SUN, il permet de se connecter à n'importe quelle base de données compatible ODBC (MS Access, SQL Server,…).
- `com.mysql.jdbc.Driver`, pilote pour la base de données MySQL.
- `org.postgresql.Driver`, pilote pour la base de données PostgreSQL.

Il est également possible d'ajouter d'autres drivers JDBC pour se connecter à des SGBD différents. Pour cela, il suffit simplement d'ajouter les nouveaux drivers soit dans le répertoire `lib/ext` de la machine virtuelle Java, soit d'ajouter le chemin d'accès dans la variable d'environnement `CLASSPATH` de Java. Ensuite, on spécifie uniquement la classe du driver dans Merja.

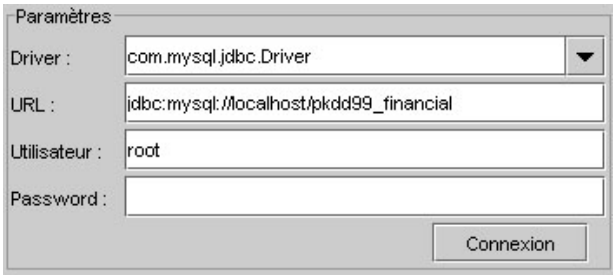



Pour déclarer une base de données comme étant compatible ODBC, il faut utiliser dans Windows le panneau « Sources de données ODBC » et ajouter la base que l'on désire rendre accessible.

*Ouverture de la base de données :*

Ensuite pour l'ouverture, il faut spécifier l'URL de la base de données ainsi que le login et le password de l'utilisateur autorisé à se connecter.

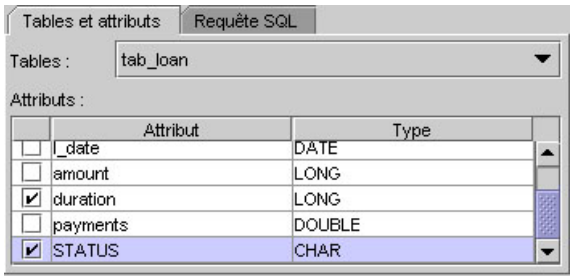
Pour cela une fenêtre à été réalisé permettant de faciliter la saisie des informations :



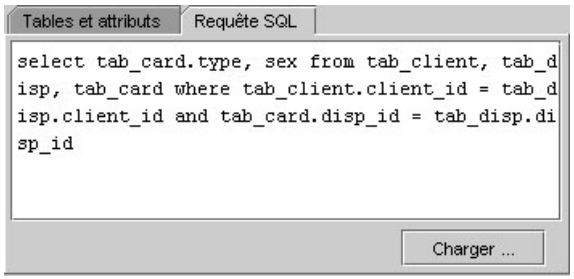

Le bouton connexion tente de se connecter à la base, en cas d'échec un message d'erreur est affiché à l'utilisateur.

## Lecture des données (requêtes SQL)

Pour la lecture des données, la première chose consiste pour l'utilisateur à choisir les tables ainsi que les attributs de la base de données. Pour cela, deux manières différentes ont été implantées :




Soit l'utilisateur choisi, une table dans la liste des tables de la BDD et ensuite sélectionne les attributs qui lui semblent intéressants. Dans ce cas une requête SQL est générée automatiquement pour interroger la base de données.

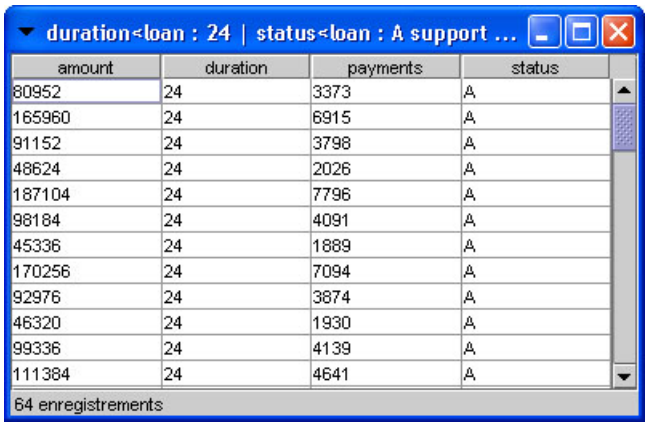



Soit l'utilisateur spécifie lui-même une requête SQL. Cette méthode permet notamment d'effectuer des jointures entre les tables et/ou de limiter le nombre de données en utilisant des clauses « where ».

Après qu'une requête SQL ait été généré, celle-ci permet d'interroger la base de données à l'aide de notre classe `Database` est fournit le résultat dans un objet `RecordSet`. Ensuite pour fournir les données à l'algorithme Apriori permettant de générer les règles d'association, on utilise une nouvelle classe `DBLoader` qui implémente les méthodes de l'interface `TransactionsReader`. Déjà les classes `CsvLoader` et `ArffLoader` permettaient la lecture des fichier CSV et ARFF et implantaient l'interface `TransactionsReader`. L'algorithme Apriori utilise uniquement les fonctions fournies par l'interface pour la lecture des données, donc il est très facile d'ajouter de nouveaux formats de fichiers ou d'améliorer la rapidité des lecteurs déjà présents.

```
public interface TransactionsReader {
    // Retourne le nom du fichier ou de la base de données
    public String getRelationName();
    // Retourne la prochaine transaction (tableau d'items)
    public Item[] nextTransaction();
    // Retourne l'ensemble des attributs avec leurs valeurs
    public Item[] getAttributes();
    // Retourne le nombre d'attributs
    public int getNbAttributes();
    // Place le pointeur sur la première transaction de la base
    public void goToFirstTransaction();
    // Retourne le nombre de transactions de la base de données
    public int getNbTransactions();
    // Retourne le nombre d'items total
    public int getNbItems();
    // Retourne le nombre d'items ayant des valeurs différentes
    public int getNbItemsDiff();
}
```

Lorsque la génération des itemsets fréquents et/ou des règles d'association est terminée, il est dorénavant possible d'afficher les données correspondantes à l'itemset ou à la règle dans une nouvelle fenêtre sous la forme suivante :




Pour retrouver les données précédentes, il suffit d'ajouter à la requête SQL plusieurs clauses « where » correspondant à l'itemset.

*Exemple :*
Requête SQL : `select `duration`,`status` from `loan``
Itemset fréquent : `duration: 12 | status: A support=93`

La requête SQL pour trouver les données correspondant à l'itemset est la suivante :
```
select `duration`,`status` from `loan` where `duration`=12 and
`status`=`A`
```

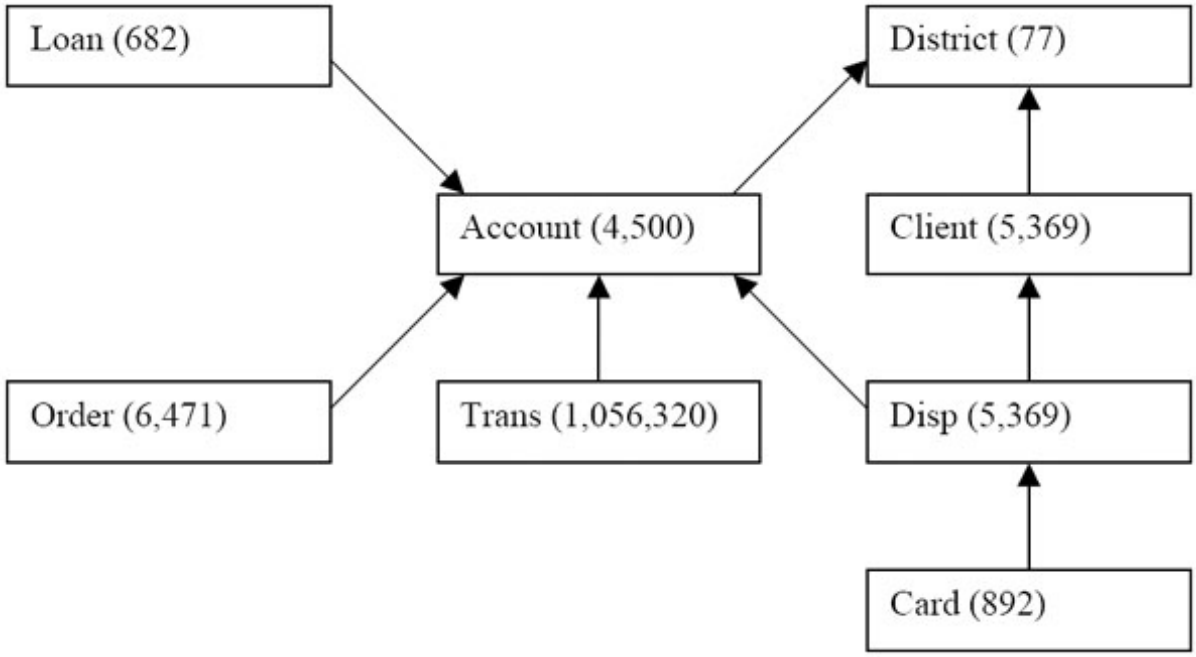
# 6. Base de données financières (PKDD99)

## Description

La base de données étudiée correspond à des données financières (banque de Prague) en provenance du PKDD99 (Practice of Knowledge Discovery in Databases de 1999) qui est un challenge de découverte de solutions.

La banque désire améliorer ses services. Pour cela, on désire par exemple trouver les bons clients pour leur offrir d'avantages de services et les mauvais clients pour minimiser les pertes.

*Schéma de la BDD*




Les données à propos des clients et de leurs comptes sont disponibles dans les tables suivantes :

- **account** : chaque enregistrement décrit les caractéristiques « statiques » d'un compte,
- **client** : chaque enregistrement décrit les caractéristiques d'un client,
- **disposition** : chaque enregistrement fait le lien entre un client et un compte,
- **order** : chaque enregistrement décrit les caractéristiques d'un ordre de paiement (retrait uniquement),
- **transaction** : chaque enregistrement décrit une opération sur un compte,
- **loan** : chaque enregistrement décrit un prêt accordé pour un compte,
- **card** : chaque enregistrement décrit un carte de crédit associée avec un compte,
- **district** : chaque enregistrement décrit les caractéristiques démographiques d'un district (région).

Chaque compte a des caractéristiques « statiques » (ex : date de création), et des caractéristiques « dynamiques » (ex : débits, crédits) données par les tables order et transaction. La table client correspond aux caractéristiques des personnes pouvant manipuler des comptes. Un client peut avoir plusieurs comptes et un compte peut être manipuler par plusieurs clients ; la relation entre les clients et les comptes est décrite dans la table disposition. Les tables loan et card décrivent des services que la banque offre aux clients ; plusieurs cartes peuvent exister pour un compte, mais un seul prêt peut être accordé pour un compte. La table district donne des informations a propos d'un région (ex : taux de chômage) ; des informations supplémentaires sur les clients peuvent être déduites de cette table.

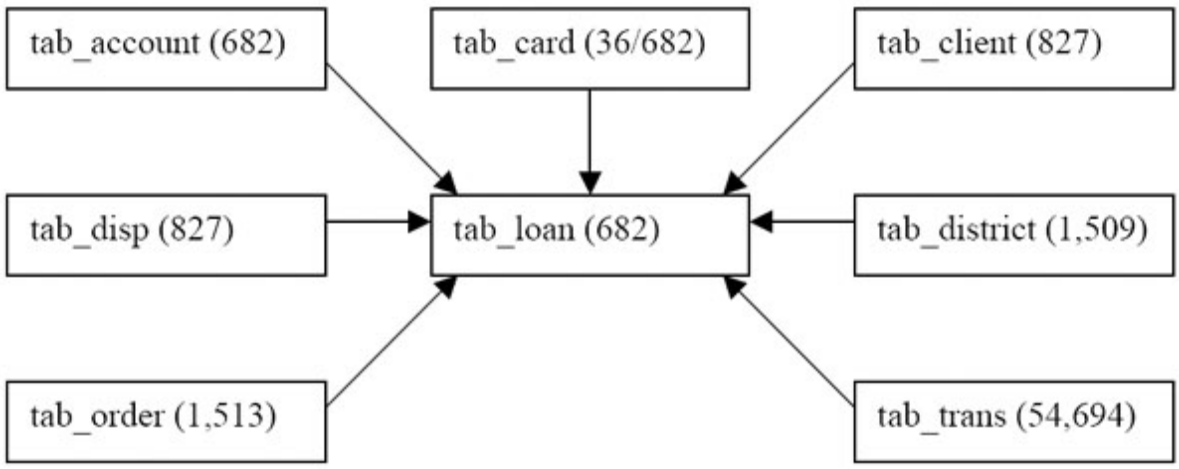
## Prétraitements

Pour permettre de découvrir plus facilement des règles, un prétraitement de la base de données a été nécessaire. Les changements suivant ont été effectués :

- réduction du nombre d'enregistrement pour ne garder que les données ayant un rapport avec la table loan. Par exemple toutes les transactions sur des comptes n'ayant pas de prêt ont été supprimées.
- Réduction du nombre enregistrement en fonction du temps. Par exemple, les données de transactions qui ont été effectuées après qu'un prêt ait été accordé pour le compte ont été supprimées. En fait, on veut fournir des informations aux banquiers pour accorder un prêt donc on ne doit considérer que les transactions effectuées après l'accord.
- Les dates qui étaient en chaîne de caractères ont été transformées au format Date.
- L'attribut birth_number qui contenait la date de naissance et le sexe d'un client a été partagé en deux attributs : birthday et sex.
- Dans toutes les tables ont a ajouté la clé étrangère loan_id.

Les prétraitements précédents ont été effectués par Mark-A Krogel dans le cadre du PKDD'99.

*Schéma de la nouvelle BDD*



Une grande majorité des données présentes dans les tables sont en majorité dans un format numérique et comme le logiciel Merja ne fait pas de discrétisation, cette phase a été effectuée directement dans la base en ajoutant les champs suivants dans les différentes tables :

Pour la table **client** :

- age (int) : contient l'age du client.
  SQL :
```
UPDATE tab_client SET age=1999-year(birthday)
```
- age_i (varchar) : discrétisation de l'age (tous les dix ans).
  SQL :
```
UPDATE tab_client SET age_i = '95-104' WHERE age < 105
UPDATE tab_client SET age_i = '85-94' WHERE age < 95
...
UPDATE tab_client SET age_i = '5-14' WHERE age < 15
```

Pour la table **district** :

- avg_sal (varchar) : discrétisation du salaire moyen d'une région
  SQL :
```
UPDATE tab_district SET avg_sal = '12000-13000' WHERE a11< 13000
UPDATE tab_district SET avg_sal = '11000-12000' WHERE a11< 12000
...
UPDATE tab_district SET avg_sal = '7000-8000' WHERE a11< 8000
```
- ur96 (varchar) : discrétisation du taux de chômage en 1996
  SQL :
```
UPDATE tab_district SET ur96 = '8-10%' WHERE a13< 10
UPDATE tab_district SET ur96 = '6-8%' WHERE a13< 8
...
UPDATE tab_district SET ur96 = '0-1%' WHERE a13< 1
```

Pour la table **loan** :

- status_b (varchar) : good ou bad, définit si un prêt s'est bien ou mal déroulé
  SQL :
```
UPDATE tab_loan SET status_b = 'good' WHERE status='a' or status='c';
UPDATE tab_loan SET status_b = 'bad' WHERE status='b' or status='d';
```
- amount_i (varchar) : discrétisation de la quantité d'un prêt
  SQL :
```
UPDATE tab_loan SET amount_i='500000-600000' WHERE amount <600000
UPDATE tab_loan SET amount_i='400000-500000' WHERE amount <500000
...
UPDATE tab_loan SET amount_i='0-100000' WHERE amount <100000
```

## Résultats

Voici quelques résultats obtenus avec la base de données prétraitée et le logiciel Merja. Pour pouvoir trouver ces résultats on exécute d'abord un requête SQL, pour effectuer les traitements de l'algorithme Apriori.
On n'a sélectionné ici que les lignes intéressantes et significatives permettant de trouver les personnes à qui les banques peuvent ou ne peuvent pas accorder un prêt.

```
select `status_b`,`amount_i` from `tab_loan`
amount_i : 400000-500000 (21)  => status_b : bad :  (6)  0.285
amount_i : 0-50000 (126)  => status_b : good :  (121)  0.96
amount_i : 100000-200000 (192)  => status_b : good :  (174)  0.906
amount_i : 50000-100000 (179)  => status_b : good :  (161)  0.899
```

On remarque ici que 28,5% des prêts d'un montant important 400000-500000 se sont mal déroulés. Alors que les prêts d'un montant inférieur à 200000 se déroule bien en général.

```
select status_b, avg_sal from tab_loan, tab_district where
tab_district.loan_id = tab_loan.loan_id group by tab_loan.loan_id
avg_sal : 12000-13000 (79)  => status_b : good : (73)  0.924
```

Les personnes ayant un salaire très élevé ont beaucoup de chances de ne pas avoir de problèmes lors d'un prêt.

```
select status_b, amount_i, avg_sal from tab_loan, tab_district where
tab_district.loan_id = tab_loan.loan_id group by tab_loan.loan_id
amount_i:200000-300000, avg_sal:8000-9000 (51) => status_b:bad : (12) 0.235
amount_i:0-50000 , avg_sal:12000-13000 (17)  => status_b:good : (17)  1.0
amount_i:50000-100000, avg_sal:12000-13000 (22) => status_b:good : (22) 1.0
```

Ici on voit que 23,5% des personnes avec un salaire faible et ayant pris un prêt d'un montant élevé ont eu des problèmes de remboursement. Alors que toutes les personnes ayant un salaire de 12000 à 13000 (élevé) et ayant fait un prêt d'un montant se situant entre 0 et 100000 n'ont eu aucun problème de remboursement.

```
select status_b, ur96 from tab_loan, tab_district where tab_district.loan_id =
tab_loan.loan_id group by tab_loan.loan_id
ur96 : 0-1% (94)  => status_b : bad :  (7)  0.074
ur96 : 1-2% (87)  => status_b : bad :  (9)  0.103
ur96 : 2-4% (235)  => status_b : bad :  (28)  0.119
ur96 : 4-6% (191)  => status_b : bad :  (23)  0.12
ur96 : 6-8% (56)  => status_b : bad :  (8)  0.142
```

Ces résultats montrent que plus le taux de chômage augmente plus le taux d'échec lors d'un prêt augmente sensiblement.

```
select status_b, age_i from tab_loan, tab_client where tab_client.loan_id =
tab_loan.loan_id group by tab_loan.loan_id
age_i : 15-24 (84)  => status_b : good :  (72)  0.857
age_i : 25-34 (155)  => status_b : good :  (140)  0.903
age_i : 35-44 (163)  => status_b : good :  (146)  0.895
age_i : 45-54 (154)  => status_b : good :  (135)  0.876
age_i : 55-64 (126)  => status_b : good :  (113)  0.896
```

Ici on ne peut rien déduire sur l'age d'un client et l'échec ou la réussite d'un prêt.

```
select status_b, age_i, sex from tab_loan,tab_client where tab_client.loan_id =
tab_loan.loan_id and group by tab_loan.loan_id
age_i : 15-24 , sex : m (33)  => status_b : good :  (30)  0.909
age_i : 25-34 , sex : m (76)  => status_b : good :  (70)  0.921
age_i : 35-44 , sex : f (84)  => status_b : good :  (77)  0.916
age_i : 55-64 , sex : f (50)  => status_b : good :  (45)  0.9
```

On montre ici la classe d'age avec le sexe des personnes ayant de très bonnes chances de réussir un prêt.

```
select status_b, a3 from tab_loan, tab_district where tab_district.loan_id =
tab_loan.loan_id group by tab_loan.loan_id
a3 : Prague (79)  => status_b : good :  (73)  0.924
a3 : east Bohemia (87)  => status_b : good :  (79)  0.908
a3 : north Bohemia (62)  => status_b : good :  (61)  0.983
```

Les trois régions précédentes sont celles ayant les plus gros taux de réussite lors d'un prêt.

```
select status_b, k_symbol from tab_loan, tab_trans where tab_trans.loan_id
= tab_loan.loan_id group by trans_id
k_symbol : SIPO (3916)  => status_b : good :  (3664)  0.935
status_b : bad (5338)  => k_symbol : UROK :  (1115)  0.208
```

Les personnes effectuant beaucoup d'opérations bancaires pour le ménage (SIPO) ont beaucoup de chances de ne pas poser de problèmes. Par contre, on remarque que sur les comptes dont les prêts ont des problèmes de remboursement, il y a 20% des opérations qui sont des remboursements de crédit. Donc les personnes ayant des crédits sont susceptibles de poser problème lors d'un prêt.

```
select status_b, type from tab_loan, tab_disp where tab_disp.loan_id =
tab_loan.loan_id
type : DISPONENT (145)  => status_b : good :  (145)  1.0
```

Tous les comptes ayant une personne autre que le possesseur du compte qui peut effectuer des opérations n'ont aucun problème de remboursement de prêt.

```
select status_b, type from tab_loan, tab_card where tab_card.loan_id =
tab_loan.loan_id
status_b : bad (76)  => type : null :  (76)  1.0
```

Toutes les personnes ayant eu des problèmes de remboursement de prêt n'avait pas de carte bancaire.

Pour pouvoir trouver encore d'autres informations il aurait fallu faire une discrétisation de toutes les données numériques et tester plusieurs façons de discrétiser différentes. De plus un expert du domaine bancaire peut également guider la recherche en sélectionnant les caractéristiques les plus intéressantes.

# 7. Manuel utilisateur

## Interface

L'interface de Merja utilise une représentation sous la forme de quatre onglets ; Fichier/BDD, Données, Itemsets fréquents, Règles d'association.
Une barre de boutons est présente à gauche de l'interface permettant d'effectuer plus rapidement certaines actions. Enfin un log en bas de l'interface tient au courant l'utilisateur sur les traitements en cours ainsi que sur les éventuelles erreurs.

*Bugs :*

Lorsque le nombre d'itemsets ou le nombre de règles est très important, de l'ordre de plus de 50 000 éléments. L'affichage de ces éléments dans les zones de textes, listes ou tables peut provoquer des exceptions (out of memory). Cela est dû à une limitation des composants Java qui ne peuvent afficher qu'un nombre limité de données.
Pour résoudre ce problème if faut soit arrêter le traitement à l'aide des boutons « stop », soit limité le nombre d'éléments en augmentant le support ou la confiance minimum.

*Barre de boutons :*

 Ouverture d'un fichier contenant une base de données au format ARFF ou CSV.
 Connexion et ouverture d'une base de données relationnelle.
 Lance la recherche des itemsets fréquents de la base courante.
 Lance la recherche des règles d'association de la base courante.
 Ferme la base de données courante et efface tous les champs.
 Stoppe le traitement de l'onglet courant.
 Affiche une boite de dialogue avec des informations sur Merja.

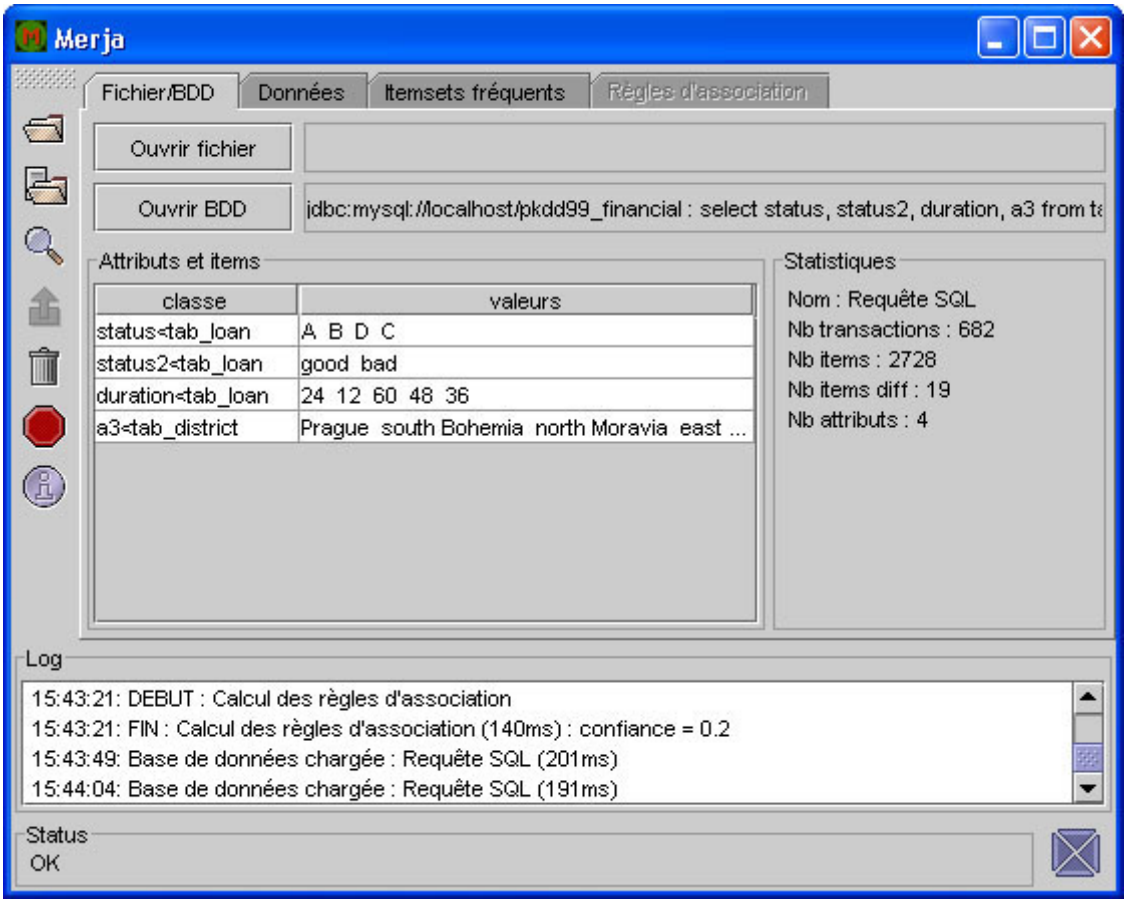
*Log :*

Le log permet d'afficher des informations sur le début, la fin et la durée d'un traitement. Les traitements peuvent être soit le calcul d'itemsets fréquents, de règles d'association soit l'ouverture et la lecture de données en provenance d'une base de données quelconque. Le log affiche également les différents messages d'erreurs pouvant survenir lors de l'utilisation de l'application. Un champ de texte « Status » informe si la dernière action a été un succès ou un échec.

 Cette image permet de savoir si un traitement est en cours, en s'animant.

## Ouverture d'un fichier ou d'une BDD

Permet l'ouverture de fichiers aux formats ARFF, CSV ou de récupérer des données en provenance d'un SGBD et affiche les principales caractéristiques.

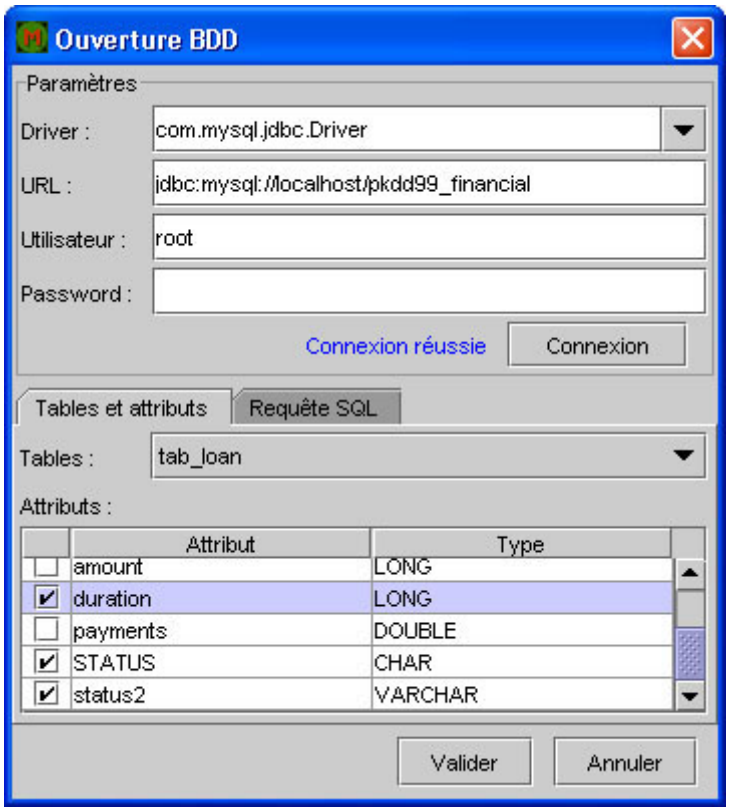



Le bouton « ouvrir un fichier » affiche une boite de dialogue permettant de choisir un fichier à traiter. Le champ à droite donne le chemin d'accès de ce fichier.
Le bouton « ouvrir BDD » affiche une boite de dialogue (voir page suivante) permettant de se connecter à une base de données et de choisir les tables avec les attributs à traiter. Le champ à droite donne l'URL de la base de données ainsi que la requête SQL associée.

Pour les fichiers ARFF et les bases de données la zone « Attribut et items » donne tous les attributs avec leurs valeurs possibles.
Pour les fichiers CSV la zone affiche tous les items différents que contient la base de transactions.

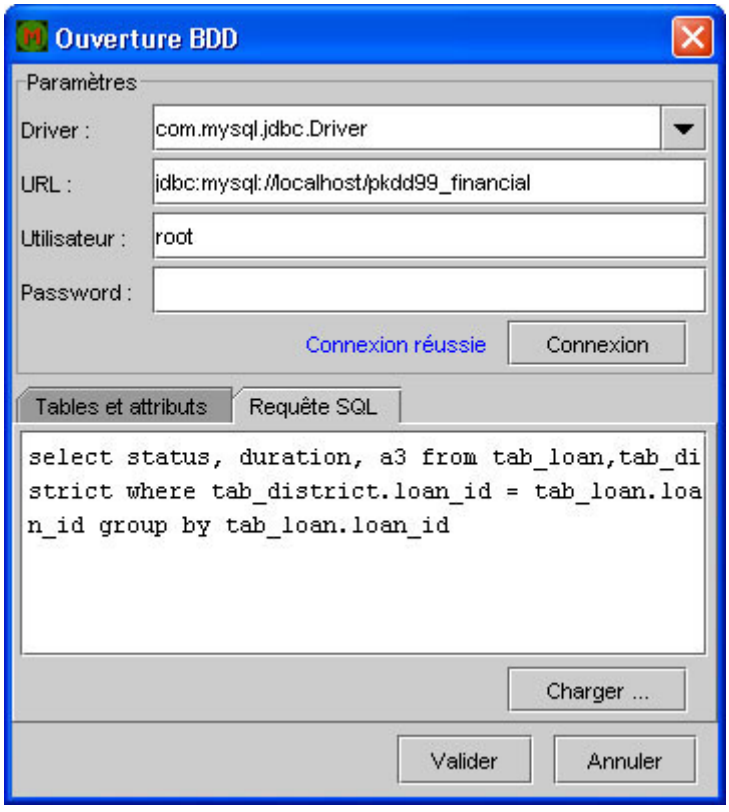
La partie « Statistiques affiche » diverses informations sur la base de données chargées.

La boite de dialogue ci-dessous permet la connexion et la sélection des données pour une base de données.




Quatre champs permettent de saisir les informations pour se connecter à la base de données. Ensuite le bouton connexion permet de se connecter à la base et de remplir la liste des tables.

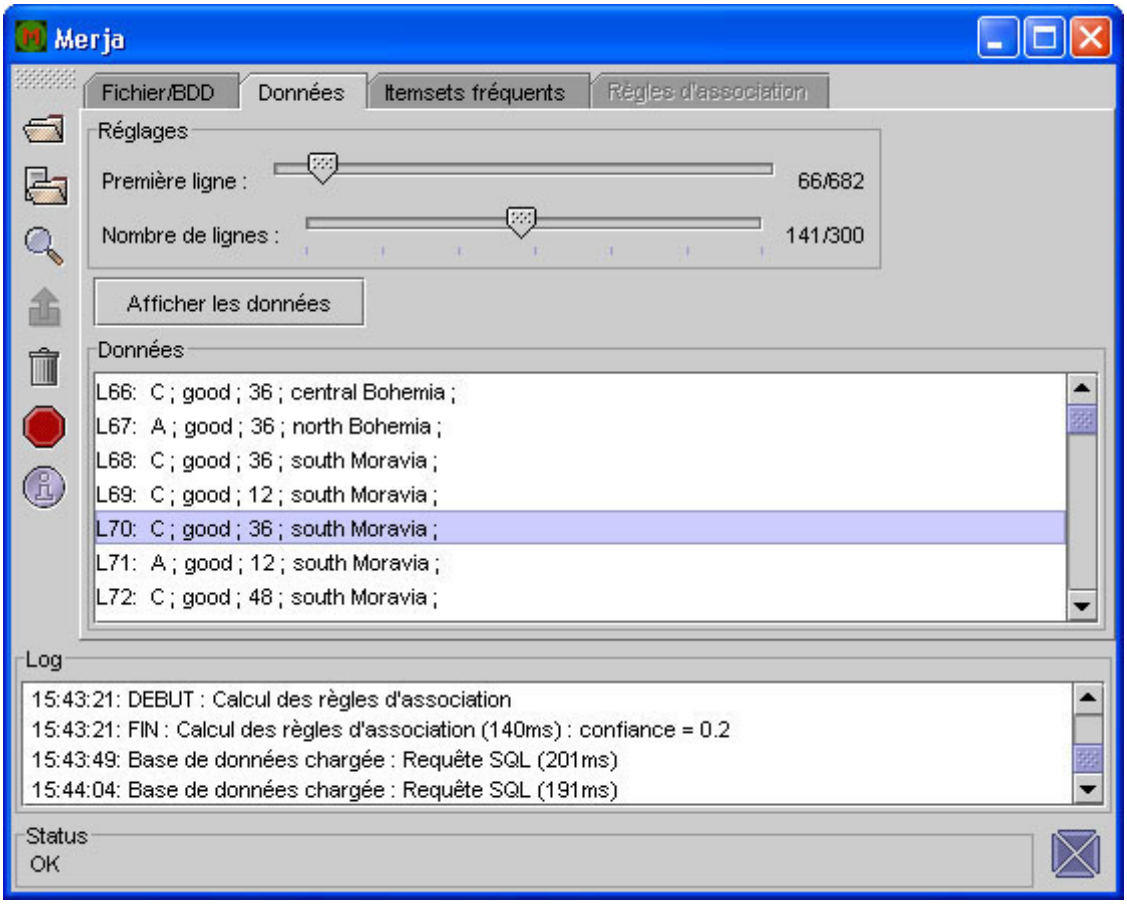
L'utilisateur doit alors choisir une table et sélectionner les champs qui seront utilisés pour les traitements.




Ce second onglet permet à la place de choisir une table et des champs, de spécifier une requête SQL. Cette méthode est particulièrement intéressante pour effectuer des jointures entre les différentes tables.

## Affichage des données

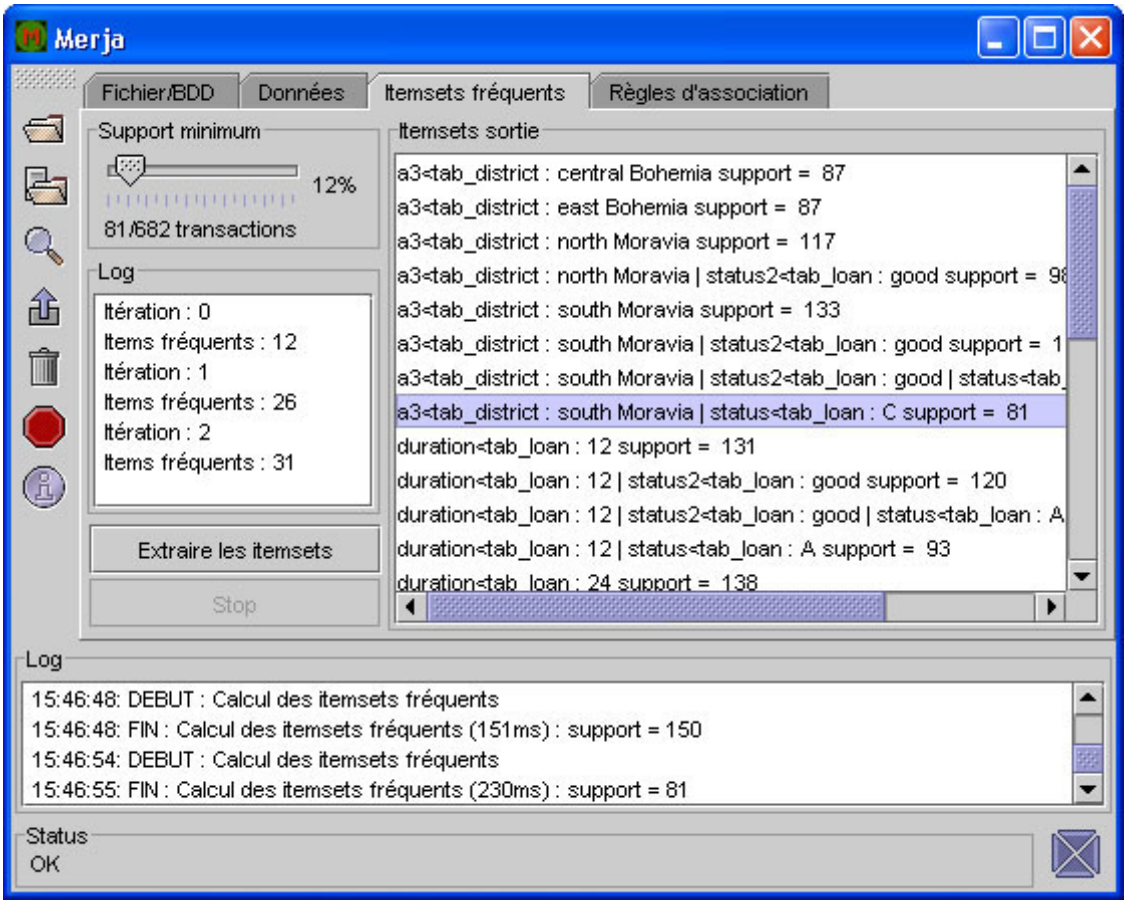
Permet d'afficher les données du fichier précédemment chargé.




Cette page permet d'afficher une partie des lignes de la base de données. Il suffit de choisir la première ligne ainsi que le nombre de lignes à afficher à partir de cette première ligne.

# Calcul des itemsets fréquents

Recherche et affichage des itemsets fréquents.




La spécification du support minimum s'effectue en choisissant un certain pourcentage, la correspondance entre ce pourcentage et le nombre de transactions est également affiché.

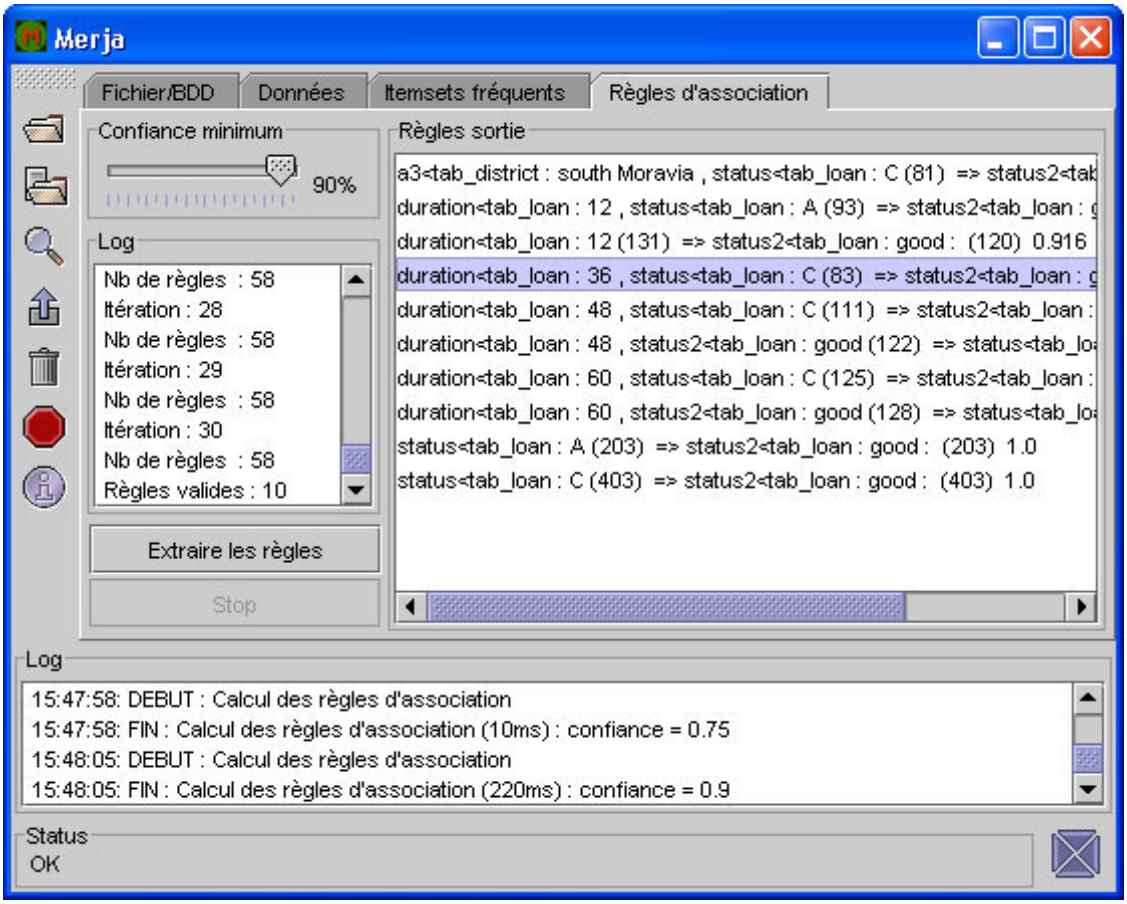
Lors de l'extraction des itemsets fréquents la zone « log » affiche les itérations de l'algorithme ainsi que le nombre d'itemsets pour chaque itération.

L'affichage des itemsets fréquents se fait de la façon suivante :
`| item | item | … | item => support`

Un double-click sur un itemset permet d'ouvrir une nouvelle fenêtre affichant tous les enregistrements (transactions) correspondants à l'itemset.

# Calcul des règles d'association

Recherche et affichage des règles d'association.




La spécification de la confiance minimum s'effectue en choisissant un pourcentage à l'aide d'une réglette.

Lors de la recherche des règles la zone « log » affiche les itérations ainsi que le nombre de règles trouvées par l'algorithme. A la fin du traitement le nombre de règles valides (conformément à la confiance minimum) est affiché.

L'affichage des règles d'association se fait de la façon suivante :
`item , … , item (support) => item , … (support) confiance`

Un double-click sur une règle d'association permet d'ouvrir une nouvelle fenêtre affichant tous les enregistrements (transactions) correspondants à la règle d'association.

# 8. Conclusion :

Bien qu'ayant connu quelques problèmes pour effectuer une sélection des tables et champs corrects, la réalisation du logiciel a été amenée à terme. Le logiciel qui fournissait déjà de très bons résultats sur des données contenues dans des fichiers, peut maintenant se connecter à une base de données. Ce nouvel aspect permet dorénavant de traiter des données plus intéressantes et bien plus proches de la réalité comme le montre la base de données financière étudiée.

Le langage Java c'est montré particulièrement intéressant dans la réalisation notamment grâce à l'API JDBC qui permet aux logiciels de se connecter à pratiquement n'importe quelle base de données.

Enfin il reste de nombreuses améliorations possibles :

- discrétisation des données numériques.
- ajout de nouveaux algorithmes comme AprioriHybrid et AprioriTid.
- Taxonomie (hiérarchisation *is-a*) des transactions.
- … etc …

Ces fonctions seront développés et ajoutés lors du projet 150h. Elles permettront au logiciel d'être très fonctionnel et lui permettront d'étudier un grand nombre de base de données très facilement.

*Darnet Camille : camsss@fr.st*
*Vouriot Thierry : yeri@fr.st*
*http://www.yeri.fr.st*