



**SafeJDBC v2.0**  
**Driver de chiffrement SQL**  
**Documentation Utilisateur**  
**31 janvier 2005**

# Table des Matières

<b>1</b>	<b>INTRODUCTION.....</b>	<b>4</b>
1.1	PRESENTATION DE SAFEJDBC .....	4
1.2	DESTINATAIRES DU DOCUMENT .....	4
1.3	ENVIRONNEMENT TECHNIQUE DE FONCTIONNEMENT .....	5
<b>2</b>	<b>INSTALLATION DE SAFEJDBC.....</b>	<b>6</b>
2.1	INSTALLATION DU JDK 1.4 DE SUN MICROSYSTEMS (TELECHARGEMENT) .....	6
2.2	INSTALLATION DES FICHIERS BINAIRES DE SAFEJDBC.....	6
2.2.1	Téléchargement de SafeJDBC.....	6
2.2.2	Installation standard.....	6
2.2.3	Installation personnalisée .....	7
2.2.4	FICHER DE LICENCE.....	7
2.2.5	Mise à jour CLASSPATH.....	8
2.3	DOCUMENTATION JAVADOC .....	8
2.4	DOCUMENTATION JAVADOC EN LIGNE.....	8
2.5	PROGRAMMES D'EXEMPLE EN JAVA.....	8
<b>3</b>	<b>PRINCIPES DE FONCTIONNEMENT DE SAFEJDBC.....</b>	<b>9</b>
3.1	TERMINOLOGIE.....	9
3.2	CHARGEMENT DU DRIVER SAFEJDBC.....	9
3.3	EXEMPLE DE CHIFFREMENT DES VALEURS PAR UN INSERT.....	11
3.4	EXEMPLE DE DECHIFFREMENT DES VALEURS PAR UN SELECT.....	12
3.5	CONTENU DE LA DATABASE SQL APRES L'INSERT .....	13
<b>4</b>	<b>ASPECTS SQL .....</b>	<b>14</b>
4.1	PRISE EN COMPTE DES CONTRAINTES DE PRODUCTION .....	14
4.2	SYNTAXE SQL AUTORISEE .....	14
4.3	IMPACT DU CHIFFREMENT SUR LA TAILLE DES DONNEES.....	16
4.4	DATABASE SQL SAFEJDBC_CATALOG.....	16
4.5	INTEGRITE & SAUVEGARDES DE SAFEJDBC_CATALOG .....	16

<b>5</b>	<b>MISE EN ŒUVRE &amp; API.....</b>	<b>17</b>
5.1	CREATION DE LA DATABASE SAFEJDBC_CATALOG .....	17
5.2	SAFEJDBCSETTER - API DE PARAMETRAGE DU DRIVER SAFEJDBC.....	17
5.2.1	getInstance() - Création d'une instance de SafeJdbcSetter .....	17
5.2.2	setWrappedDriverName() - Paramétrage du Driver natif à encapsuler .....	18
5.2.3	setSafeJdbcDbUrl() - Paramétrage de l'URL de SAFEJDBC_CATALOG .....	18
5.2.4	Définition du fichier de Logging .....	19
5.2.5	Chargement du Driver SafeJDBC .....	19
5.3	SAFEJDBCSETTER - API DE PARAMETRAGE DES ELEMENTS DE CRYPTOGRAPHIE .....	20
5.3.1	Généralités .....	20
5.3.2	Providers.....	20
5.3.3	Algorithmes.....	21
5.3.4	setProvider() - Paramétrage du Provider à utiliser .....	21
5.3.5	setAlgorithm() - Paramétrage de l'algorithme à utiliser .....	22
5.3.6	setKey() - Paramétrage de la clé secrète de chiffrement .....	22
5.4	APPEL DE SAFEJDBC EN MODE PUR JDBC DRIVER .....	24
5.4.1	Principe.....	24
5.4.2	Paramétrage de con_safejdbc.ini.....	24
5.5	VERIFICATION DU STATUT SECURISE D'UNE CONNECTION .....	26
5.6	RECUPERATION DE LA CONNECTION «NATIVE» .....	26
5.7	CIPHERCONNECTIONFACTORY – API DE CREATION D'UNE «CONNECTION» .....	27
5.8	CREATION DE CONNECTION SECURISEE A PARTIR DE DATASOURCES .....	28
5.8.1	Principe Général.....	28
5.8.2	Le JavaBean SafeDataSourceFactory .....	28
5.9	EXEMPLE D'UTILISATION DE DATASOURCES AVEC TOMCAT 5.0.....	29
5.9.1	Etape 1 : Définition du DataSource de l'application.....	30
5.9.2	Etape 2 : Définition du DataSource du Catalogue SafeJDBC .....	31
5.9.3	Etape 3 : Définition du SafeDataSource sécurisé .....	32
5.10	COLUMNSCIPHER - API DE SELECTION DES COLONNES A CHIFFRER/DECHIFFRER.....	34
5.10.1	Généralités .....	34
5.10.2	getInstance () - Création d'une instance de ColumnsCipher.....	34
5.10.3	setOutputStream() - Paramétrage du flux de sortie .....	34
5.10.4	setFileOutputStream() - Paramétrage du flux de sortie sur un fichier .....	35
5.10.5	addColumn() - Ajout d'une colonne .....	35
5.10.6	addColumnFromFile() - Ajout des colonnes à partir d'un fichier «ini».....	36
5.10.7	defineJoinColumn() – Définition de jointures chiffrées .....	37
5.10.8	defineJoinColumnsFromFile() - Définition des colonnes de Jointure à partir d'un fichier «ini» .....	38
5.10.9	execute() – Lancement du process de chiffrement ou de déchiffrement.....	39
5.11	IMPLEMENTATION DE COLUMNSCIPHER : BATCH COLUMNSCIPHERMAIN.....	41
5.12	COMMENT DEFINIR DES COLONNES CHIFFREES DANS DES VUES .....	43
<b>6</b>	<b>SUPPORT .....</b>	<b>46</b>

# 1 INTRODUCTION

## 1.1 Présentation de SafeJDBC

SafeJDBC est un driver JDBC qui s'utilise en complément de tout driver JDBC 2.x ou 3.x. SafeJDBC permet le chiffrement et le déchiffrement à la volée de colonnes SQL sans ajouter de programmation dans les sources Java.

SafeJDBC se charge :

- Du chiffrement des valeurs des colonnes SQL à partir des valeurs en clair en Java *avant* la mise à jour de la base par un ordre SQL 92 de type INSERT ou UPDATE.
- Du déchiffrement sous la JVM des valeurs des colonnes SQL *après* leur récupération dans la database (SELECT ou colonnes des conditions WHERE des ordres UPDATE ou DELETE).

SafeJDBC encapsule de façon transparente le driver habituel de l'applicatif Java, qui est simplement configuré par des instructions préalables Java.

## 1.2 Destinataires du document

Ce document s'adresse :

- 1) Aux développeurs Java qui souhaitent inclure SafeJDBC dans leurs programmes (classes Java, Scripts, JSP, etc.)
- 2) Aux DBA (database Administrator) SQL qui souhaitent se documenter sur les impacts de SafeJDBC sur l'exploitation de leur database.

### 1.3 Environnement technique de fonctionnement

SafeJDBC est écrit en 100 % Java et fonctionne de façon identique sous Windows NT/2000/XP, Linux et tous les Unix qui supportent Java et JDBC.

Voici les environnements supportés dans cette version :

Environnement technique	Versions supportées
Windows NT	<ul style="list-style-type: none"> <li>• Windows NT Server V 4.0 SP 4 ou supérieure.</li> <li>• Windows NT Workstation V 4.0 SP 4 ou supérieure.</li> </ul>
Windows 2000/XP	<ul style="list-style-type: none"> <li>• Windows 2000 Server SP 2 ou supérieure.</li> <li>• Windows 2000 Professionnal SP 2 ou supérieure.</li> <li>• Windows XP Professionnal SP 1 ou supérieure.</li> </ul>
Unix	<ul style="list-style-type: none"> <li>• Sun Solaris 7.x/8.x/9.x (SPARC et Intel).</li> <li>• Linux (Intel) RedHat 6.1/7.1/8.0/9.0.</li> <li>• Autres Unix qui supportent Java.</li> </ul>
JVM (Java Virtual Machine)	<ul style="list-style-type: none"> <li>• JDK 1.4.1 de Sun Microsystems ou supérieur.</li> </ul>

L'API est supportée pour toutes les versions égales ou supérieures de ces environnements Java.

## 2 INSTALLATION DE SAFEJDBC

### 2.1 Installation du JDK 1.4 de Sun Microsystems (téléchargement)

SafeJDBC nécessite l'installation du JDK 1.4 de Sun Microsystems.

Le JDK pour Windows/Unix/Linux est disponible en téléchargement sur le site de Sun à l'adresse : <http://java.sun.com/j2se>.

### 2.2 Installation des fichiers binaires de SafeJDBC

#### 2.2.1 Téléchargement de SafeJDBC

Téléchargez le fichier `safejdbc_v2.00.zip` à l'adresse : [http://www.safelogic.com/safejdbc/download/safejdbc\\_v2.00.zip](http://www.safelogic.com/safejdbc/download/safejdbc_v2.00.zip)

#### 2.2.2 Installation standard

Par défaut nous suggérons les répertoires d'installation suivants :

<b>Windows NT/2000/XP</b>	Dézipper le contenu du fichier <code>safejdbc_v2.00.zip</code> dans <code>c:\</code> . Cela va créer l'arborescence <code>c:\safelogic\safejdbc</code>
<b>Unix/Linux</b>	Créer un user <code>safelogic</code> et dézipper le contenu du fichier <code>safejdbc_v2.00.zip</code> dans <code>/home</code> . Cela va créer l'arborescence <code>/home/safelogic</code> .

On obtient l'arborescence Windows :

Nom répertoire Windows (par défaut)	Commentaires
<code>c:\safelogic\lib</code>	Bibliothèques Java communes de SafeLogic
<code>c:\safelogic\safejdbc</code>	Répertoire de base de SafeJDBC
<code>c:\safelogic\safejdbc\bin</code>	Scripts divers (classpath).
<code>c:\safelogic\safejdbc\conf</code>	Fichiers de configuration de SafeJDBC.
<code>c:\safelogic\safejdbc\doc</code>	Documentation d'utilisation
<code>c:\safelogic\safejdbc\examples</code>	Programmes d'exemples en Java
<code>c:\safelogic\safejdbc\javadoc</code>	Javadoc
<code>c:\safelogic\safejdbc\lib</code>	Bibliothèques Java spécifiques de SafeJDBC.

On obtient l'arborescence Unix/Linux :

Nom répertoire Unix/Linux (par défaut)	Commentaires
/home/safelogic/lib	Bibliothèques Java communes de SafeLogic
/home/safelogic/safejdbc	Répertoire de base de SafeJDBC
/home/safelogic/safejdbc/bin	Scripts divers (classpath).
/home/safelogic/safejdbc/conf	Fichiers de configuration de SafeJDBC.
/home/safelogic/safejdbc/doc	Documentation d'utilisation
/home/safelogic/safejdbc/examples	Programmes d'exemples en Java
/home/safelogic/safejdbc/javadoc	Javadoc
/home/safelogic/safejdbc/lib	Bibliothèques Java spécifiques de SafeJDBC.

### 2.2.3 Installation personnalisée

SafeJDBC peut être installé librement dans un répertoire racine différent de /home/safelogic ou c:\safelogic.

Il faut et il suffit simplement de respecter les contraintes de CLASSPATH décrites ci-dessous.

### 2.2.4 FICHER DE LICENCE

SafeJDBC utilise un fichier de licence `safejdbc_license.txt` - **fourni séparément du fichier WinZip** – et qui comporte plusieurs lignes avec dans l'ordre :

- Un code de licence suivant le type de produit.
- Un code de catégorie suivant le nombre connexions JDBC simultanées autorisées.
- La liste des nom de serveurs (HOSTNAME) d'installation, séparés par des «;».
- 99999999 ou une date de fin d'évaluation du produit.
- Un email d'identification de l'acheteur.
- Une chaîne hexadécimale qui correspond à la signature des éléments précédents avec une clé privée RSA.

La localisation de du fichier doit répondre aux deux contraintes :

- `safejdbc_license.txt` doit être situé dans le CLASSPATH et s'installe par défaut dans le sous répertoire `conf` ( `c:\safelogic\safjdbc\conf` ou `/home/safelogic/conf` ).
- `safejdbc_license.txt` doit être situé dans le même répertoire que le fichier [license@safelogic.com\\_1\\_RSA.pkf](mailto:license@safelogic.com_1_RSA.pkf). Ce fichier est localisé par défaut dans le sous-répertoire `conf`.

## 2.2.5 Mise à jour CLASSPATH

Le CLASSPATH java du Serveur doit contenir :

1. Le chemin d'accès aux deux librairies `cryptix.jar` et `safejdbc.jar`
2. Le chemin d'accès au sous-répertoire `conf` qui contient les fichiers de licence `safejdbc_license.txt` et `license@safelogic.com_1_RSA.pkf` de clé

### Exemple Windows NT/2000/XP

Avec l'installation standard, ajouter la chaîne suivante au CLASSPATH :

```
c:\safelogic\lib\cryptix.jar;c:\safelogic\safejdbc\lib\safejdbc.jar;c:\safelogic\safejdbc\conf
```

### Exemple Unix/Linux

Avec l'installation standard, ajouter la chaîne suivante au CLASSPATH :

```
/home/safelogic/lib/cryptix.jar:/home/safelogic/safejdbc/lib/safejdbc.jar:/home/safelogic/safejdbc/conf
```

## 2.3 Documentation Javadoc

La documentation Javadoc de toutes les API SafeJDBC est contenue dans le répertoire `/safelogic/safejdbc/javadoc` du fichier WinZip.

## 2.4 Documentation Javadoc en ligne

La documentation Javadoc des APIs est aussi accessible en ligne à l'adresse : <http://www.safelogic.com/safejdbc/v2.00/javadoc>.

## 2.5 Programmes d'exemple en Java

Le répertoire `/safelogic/safejdbc/examples` du fichier WinZip contient des programmes exemple d'utilisation de SafeJDBC repris dans ce document.

## 3 PRINCIPES DE FONCTIONNEMENT DE SAFEJDBC

### 3.1 Terminologie

Nous utiliserons par la suite les termes suivants pour clarifier le propos :

Terme	Signification
Applicatif Java	Il s'agit de l'application Java qui accède en JDBC à SQL et utilise SafeJDBC pour le chiffrement.
Driver JDBC natif	Le Driver de l'applicatif Java qui accède au moteur SQL en JDBC et est utilisé habituellement.

### 3.2 Chargement du Driver SafeJDBC

SafeJDBC « encapsule » tout Driver natif JDBC.

Soit l'instruction de chargement d'un Driver JDBC natif de MySQL pour un applicatif Java :

```
/**
 * Load the Driver & return a Connection
 * @param sPassword the password connection
 *                  to the sj_clients database
 */
public Connection loadJdbcDriver(String sPassword)
    throws Exception
{
    // Set JDBC Driver & application database address.
    String sDriver = "org.gjt.mm.mysql.Driver";
    String sDbUrl
        = "jdbc:mysql://localhost:3306/sj_clients?"
          + "user=safelogic&password="+ sPassword;

    // load the Driver and get a Connection to the database
    Class.forName(sDriver).newInstance();
    Connection connection = DriverManager.getConnection(sDbUrl);

    return connection;
}
```

Pour activer les chiffrements avec SafeJDBC, il suffit de modifier cette méthode comme suit :

- Définir le nom `com.safejdbc.api` pour `sDriver`.
- Définir le nom du Driver originel à « wrapper » ou encapsuler.
- Charger une instance de **SafeJdbcSetter**
- Définir une clé dérivée d'une passphrase.

On obtient la méthode **loadJdbcDriver** modifiée :

```
/**
 * Load the Driver & return a Connection
 * @param sUserId      the SafeJDBC UserId
 * @param sPassword    the password connection
 *                    to the sj_clients database
 * @param caPassprhase the passphrase to use as key
 *                    for the SafeJDBC Driver
 */
public Connection loadJdbcDriver(String sUserId,
                                char [] caPassword,
                                char [] caPassprhase)
    throws Exception
{
    // Set JDBC Driver & application database address.

    String sDriver = "com.safejdbc.api.Driver";
    String sDbUrl
        = "jdbc:mysql://localhost:3306/sj_clients?"
          + "user=safelogic&password=" + new String(caPassword);

    String sNativeDriver = "org.gjt.mm.mysql.Driver";

    String sJdbcUrl = "jdbc:mysql://localhost:3306/"
        + "safejdbc_catalog?" + "user=safelogic&password="
        + new String(caPassword);

    // create a SafeJDBC instance
    SafeJdbcSetter sjSetter = SafeJdbcSetter.getInstance();

    // set the Wrapped Driver
    sjSetter.setWrappedDriverName(sNativeDriver);

    // set SafeJdbc's own catalog
    sjSetter.setSafeJdbcDbUrl(sJdbcUrl);

    // set the encryption key for cipher & decipher operations
    sjSetter.setKey(sUserId, caPassprhase);

    // load the Driver and get a Connection to the database
    Class.forName(sDriver).newInstance();

    Connection connection = DriverManager.getConnection(sDbUrl);

    if (! (connection instanceof CipherConnection))
    {
        throw new SQLException("SafeJDBC Driver not loaded!");
    }

    return connection;
}
```

### 3.3 Exemple de chiffrement des valeurs par un INSERT

SafeJDBC reçoit les instructions SQL à envoyer au SGBD et chiffre les valeurs correspondants aux colonnes qui ont été définies comme « à chiffrer ». Nous verrons plus bas comment et où définir les colonnes à chiffrer.

Par exemple, nous définissons une table CLIENTS pour laquelle nous voulons chiffrer les champs noms, prénoms, téléphone et e-mail.

```

/**
 * Insert a row in the database
 * @param conn      The database Connection
 * @throws SQLException
 */
public void doInsert(Connection conn) throws SQLException
{
    String sStatement =
        "INSERT INTO CLIENTS VALUES "
        + " (1, 'Smith', 'John', 1, NULL, 'john@smith.com' ,"
        + " '01 45 72 25 15', '01 45 72 25 15', 7, "
        + " 'Bld de Dixmude', 75007, 'Paris', 'fr' )";

    Statement stmt = conn.createStatement();

    stmt.executeUpdate(sStatement);
    stmt.close();
}

```

SafeJDBC intercepte le code de **sStatement** et le « réécrit » comme suit avant de le transmettre au moteur SQL via le Driver JDBC originel :

```

INSERT INTO CLIENTS VALUES ( 1, 'VpOAdDo=', '+/LHjw==', 1, NULL,
'7qrhr5ZExgVrxkE8PR8=', 'xbQXUnruJQrVRNHkBG0=', '01 45 72 25 15', 7,
'Bld de Dixmude', 75007, 'Paris', 'fr' )

```

Le principe est identique pour une instruction UPDATE.

SafeJDBC fonctionne suivant le même principe avec un les paramètres associés du **PreparedStatement** : chaque valeur de paramètre est chiffrée par l'instruction `setXxxx()` avant transmission au moteur SQL.

### 3.4 Exemple de déchiffrement des valeurs par un SELECT

SafeJDBC récupère le contenu du Statement et chiffre les conditions qui portent sur des colonnes chiffrées.

```

/**
 * Select rows from the database
 * @param conn          The database Connection
 * @throws SQLException
 */
public void doSelect(Connection conn) throws Exception
{
    String sStatement =
        "SELECT CLI_REF, CLI_LASTNAME, CLI_FIRSTNAME, CLI_GENDER, "
        + " CLI_BIRTHDATE, CLI_EMAIL, CLI_PHONE, CLI_FAX, "
        + " CLI_ADDR_STREET_NUM, CLI_ADDR_STREET_NAME, "
        + " CLI_ADDR_ZIP, CLI_ADDR_TOWN "
        + " FROM CLIENTS WHERE (CLI_REF = ? OR CLI_REF = ?)"
        + "          AND (CLI_EMAIL = 'john@smith.com' "
        + "          OR CLI_EMAIL = 'robert@wesson.com')";

    PreparedStatement prepStmt
    = conn.prepareStatement(sStatement);

    prepStmt.setInt(1, 1);
    prepStmt.setInt(2, 10);

    ResultSet rs = prepStmt.executeQuery();

    while (rs.next())
    {
        String sCliRef          = rs.getString("CLI_REF");
        String sCliLastname     = rs.getString("CLI_LASTNAME");
        String sCliFirstname    = rs.getString("CLI_FIRSTNAME");
        // Etc..
    }

    rs.close();
    prepStmt.close();
}

```

Le **Statement** est traduit avant transmission au moteur SQL en :

```

SELECT CLI_REF, CLI_LASTNAME, CLI_FIRSTNAME, CLI_GENDER, CLI_BIRTHDATE,
LI_EMAIL, CLI_PHONE, CLI_FAX, CLI_ADDR_STREET_NUM, CLI_ADDR_STREET_NAME,
CLI_AR_ZIP, CLI_ADDR_TOWN FROM CLIENTS WHERE (CLI_REF = ? OR CLI_REF = ?)
AND (CLI_AIL = '7qrhr5ZExgVrxkE8PR8='
OR CLI_EMAIL = '9qrrpKRD6xufb3uqKVODGbg='):

```

Le ResultSet récupéré dans la JVM ne contient que des valeurs chiffrées pour les colonnes noms, prénoms, téléphone et e-mail.

Chaque `rs.getXxx("nom_colonne")` déchiffre la valeur correspondante du ResultSet (du Driver natif JDBC) avant renvoi par le ResultSet de SafeJDBC.

### 3.5 Contenu de la database SQL après l'INSERT

La base SQL contient un « mixte » d'informations chiffrées et déchiffrées.  
Voici un exemple de dump à partir d'une base MySQL 3.23 sous Windows 2000 :

```
***** 1. row *****
      cli_ref: 1
      cli_lastname: VpOAdDo=
      cli_firstname: +/LHjw==
      cli_gender: 1
      cli_birthdate: 20030124174250
      cli_email: 7qrhr5ZExgVrxkE8PR8=
      cli_phone: xbQXUnruJQrVRNHkBG0=
      cli_fax: 01 45 72 25 15
      cli_addr_street_num: 7
      cli_addr_street_name: Bld de Dixmude
      cli_addr_zip: 75007
      cli_addr_town: Paris
      cli_addr_country_code: fr
2 rows in set (0.00 sec)

mysql>
```

## 4 ASPECTS SQL

### 4.1 Prise en compte des contraintes de production

SafeJDBC a été conçu pour fonctionner dans un environnement SQL de production.

SafeJDBC ne provoque **aucun** overhead sur le moteur SQL lors du fonctionnement des applicatifs Java :

- SafeJDBC n'accède pas au catalogue SQL.
- SafeJDBC ne fabrique jamais de requête SQL supplémentaire.
- SafeJDBC ne transforme jamais les requêtes originelles (sauf pour « retirer » les fonctions UPPER() et LOWER()).
- SafeJDBC utilise une méta-database SQL isolée et qui est appelée uniquement au démarrage de l'applicatif client.

### 4.2 Syntaxe SQL autorisée

On distingue deux cas :

- SafeJDBC supporte toutes les syntaxes SQL pour les requêtes qui ne contiennent pas de colonne chiffrée. Dans ce cas, la requête est directement transmise au Driver JDBC natif.
- SafeJDBC supporte SQL 92 pour les requêtes qui contiennent des colonnes chiffrées, avec les restrictions décrites dans le tableau ci-dessous.

Domaine de limitation	Limitations pour les colonnes chiffrées SafeJDBC Version 1.02
Formats de colonne	Les colonnes chiffrées doivent être du type : INTEGER, CHAR, VARCHAR, LONGVARCHAR, BINARY, VARBINARY, LONGVARBINARY, BLOB, CLOB
UPDATE	Si un SET est fait sur une colonne chiffrée, la valeur doit être alphanumérique entre « quotes » ou numérique.
INSERT	L'INSERT ne doit pas préciser les noms de colonnes et doit comporter les valeurs pour toutes les colonnes.
Clause WHERE	<ul style="list-style-type: none"> <li>• Les opérateurs de comparaison autorisés sur les colonnes chiffrées sont : =, &lt;&gt;, !=, IN.</li> <li>• Les comparaisons d'une colonne chiffrée sont possibles uniquement sur une liste discrète de valeurs alphanumériques ou numériques.</li> <li>• Les opérateurs de comparaison suivants ne sont pas supportés pour les colonnes chiffrées : &gt;, &gt;=, &lt;, &lt;=, BETWEEN, LIKE</li> <li>• Les opérateur suivants ne sont pas autorisés sur des colonnes chiffrées : ALL, ANY, EXISTS, UNIQUE.</li> </ul>
Tri	Une clause ORDER BY sur une colonne chiffrée ne provoque pas de tri effectif.
Fonctions de groupe	GROUP BY [HAVING] Les fonctions de groupe ne sont pas autorisées sur les colonnes chiffrées, à l'exception de COUNT
Autres fonctions	<ul style="list-style-type: none"> <li>• Seules les fonctions UPPER, LOWER et TRIM sont supportées sur les colonnes chiffrées.</li> <li>• Les autres fonctions ne sont pas supportées sur des colonnes chiffrées : POSITION, CHAR_LENGTH, BIT_LENGTH, EXTRACT, SUBSTRING</li> </ul>
Jointure	Les jointures sur les colonnes chiffrées sont acceptées avec les restrictions : <ul style="list-style-type: none"> <li>• Seul les tests d'égalité et de différence sont acceptés.</li> <li>• Toutes les colonnes spécifiées dans une jointure doivent être chiffrées.</li> </ul>
SELECT	Si le format « wildcard » de type SELECT * FROM TABLE est utilisé : une et une seule wildcard est autorisée, qui correspond à l'unique expression, sans autre colonne ou expression.
Tous	L'utilisation de « SELECT imbriqués » n'est pas supporté pour les instructions qui contiennent des colonnes chiffrées.
UPDATE INSERT	Les méthodes de type PreparedStatement.setXxxStream() ne sont pas supportées sur des colonnes chiffrées.
ResultSet	Les méthodes ResultSet.updateXxx() ne sont pas supportées sur des colonnes chiffrées.

### 4.3 Impact du chiffrement sur la taille des données

L'impact du chiffrement sur le stockage des données est décrit dans le tableau suivant :

Types de donnée SQL	Impact du chiffrement sur la taille des données stockées
INTEGER,	Aucun.
BINARY, VARBINARY, LONGVARBINARY, BLOB	Aucun.
CHAR, VARCHAR, LONGVARCHAR, CLOB	Expansion d'environ 25 % de la chaîne de caractères.

### 4.4 database SQL safejdbc\_catalog

SafeJDBC possède son propre « catalogue » qui recense :

- Les colonnes à chiffrer par table et database, avec les informations détaillées (rang, type, longueur, etc.)
- Des informations de chiffrement par colonne.

Ce catalogue a un rôle « statique » et n'est *pas* accédé pendant le fonctionnement des applicatifs :

- La database **safejdbc\_catalog** est accédée une et une seule fois lors du chargement de la JVM.
- Le Driver SafeJDBC charge alors en mémoire son contenu dans une structure dédiée et organisée pour des accès très rapides.

Note :

La base **safejdbc\_catalog** ne contient **aucune information confidentielle**.

### 4.5 Intégrité & sauvegardes de safejdbc\_catalog

L'**intégrité** de safejdbc\_catalog doit être assurée afin de garantir la cohérence des opérations de chiffrement et de déchiffrement. Nous vous recommandons d'effectuer des sauvegardes régulières de la database.

Les informations contenues dans la table **safejdbc\_store** et **safejdbc\_users** peuvent être « refabriquées » en cas de perte de données sur la table.

Par contre :

**Les informations de la table safejdbc\_iv ne peuvent être reconstruites, car il s'agit de valeurs aléatoires créées au moment de l'initialisation de la base.**

## 5 MISE EN ŒUVRE & API

### 5.1 Création de la database SAFEJDBC\_CATALOG

Le « catalogue » des tables SQL de SafeJDBC doit être créé au préalable à l'utilisation du Driver.

Les deux fichiers suivants contiennent les CREATE TABLE de la database SAFEJDBC\_CATALOG :

Environnement	Fichier de création
Windows NT/2000	c:\safelogic\safejdbc\sql\safejdbc_create_tables.sql
Unix/Linux	/home/safelogic/safejdbc/sql/safejdbc_create_tables.sql

Notes :

- Le nom de la database peut être librement changé/choisi.
- La syntaxe utilisée dans les, CREATE TABLE est un sous-ensemble restreint de SQL 92 et est supportée par tous les SGBD SQL modernes.

### 5.2 SafeJdbcSetter - API de paramétrage du Driver SafeJDBC

Le paramétrage du Driver SafeJDBC doit être fait dans l'application Java *avant* le chargement du Driver.

La classe **com.safejdbc.api.SafeJdbcSetter** est utilisée pour paramétrer la **Connexion** Java sécurisée :

#### 5.2.1 getInstance() - Création d'une instance de SafeJdbcSetter

Méthode	<code>SafeJdbcSetter.getInstance()</code>
Description	Retourne une instance de la classe SafeJdbcSetter.
Arguments	Aucun.
Retour	Une nouvelle instance de la classe SafeJdbcSetter.

Exemple :

```
SafeJdbcSetter sjSetter = SafeJdbcSetter.getInstance();
```

L'import suivant doit être déclaré :

```
import com.safejdbc.api.SafeJdbcSetter;
```

## 5.2.2 setWrappedDriverName() - Paramétrage du Driver natif à encapsuler

Méthode	SafeJdbcSetter.setWrappedDriverName()		
Description	Paramétrage du Driver à utiliser pour accéder aux bases SQL.		
Arguments	String	sDriver	Nom de la classe de chargement du Driver JDBC natif.
Retour	Aucun		

Exemple :

Paramétrage du Driver de MySQL :

```
// Set the Wrapped Driver
sjSetter.setWrappedDriverName("org.gjt.mm.mysql.Driver");
```

## 5.2.3 setSafeJdbcDbUrl() - Paramétrage de l'URL de SAFEJDBC\_CATALOG

Méthode	SafeJdbcSetter.setSafeJdbcDbUrl()		
Description	Paramétrage de l'URL de la database SAFEJDBC_CATALOG.		
Arguments	String	sUrl	URL complète de l'accès à la database SAFEJDBC_CATALOG.
Retour	Aucun.		

Exemple :

Paramétrage de l'URL de la base SafeJDBC avec MySQL :

```
// Set SafeJdbc's own catalog
sjSetter.setSafeJdbcDbUrl("jdbc:mysql://localhost:3306/"
    + "safejdbc_catalog?user=safelogic&password=my_password");
```

Méthode	SafeJdbcSetter.setSafeJdbcDbUrl()		
Description	Paramétrage de l'URL de la tatabase SAFEJDBC_CATALOG.		
Arguments	String	sUrl	URL complète de l'accès à la database SAFEJDBC_CATALOG.
	String	sUser	User name d'accès à la database.
	String	sPassword	Password d'accès à la database.
Retour	Aucun.		

Exemple :

```
// Set SafeJdbc's own catalog
sjSetter.setSafeJdbcDbUrl("jdbc:mysql://localhost:3306/safejdbc_catalog",
    "safelogic",
    "safejdbc_catalog_secret_password");
```

## 5.2.4 Définition du fichier de Logging

SafeJDBC permet de tracer/logger toutes les instructions SQL dans un fichier ASCII :

Méthode	SafeJdbcSetter.setSafeJdbcLogfile()		
Description	Paramétrage du fichier de Logging		
Arguments	String	sLogfile	Nom du fichier de Logging. Si le nom ne contient pas de directory, le fichier de logging sera créé dans System.getProperty("user.dir")
Retour	Aucun.		

Exemple :

Logging des instructions SQL dans le fichier safejdbc.log:

```
// Set SafeJdbc's logfile
sjSetter.setSafeJdbcLogfile("safejdbc.log");
```

## 5.2.5 Chargement du Driver SafeJDBC

Le chargement du Driver se fait par le couple d'instructions habituelles de chargement de tout Driver JDBC :

```
Class.forName("com.safejdbc.api.Driver").newInstance();
Connection connection = DriverManager.getConnection(sDbUrl);
```

Avec :

- **sDbUrl** : URL de la database applicative.

## 5.3 SafeJdbcSetter - API de paramétrage des éléments de cryptographie

### 5.3.1 Généralités

Les éléments de cryptographie à paramétrer sont :

- Le **Provider** (ou **Fournisseur**) de cryptographie à utiliser.
- **L'Algorithme** de chiffrement symétrique à utiliser, dans la une liste pré-établie.
- La valeur de la **clé secrète** de chiffrement symétrique à utiliser. C'est cette clé secrète qui sera utilisée pour toutes les opérations de chiffrement et de déchiffrement. La valeur de cette clé doit être tenue secrète et protégée du monde extérieur.

SafeJDBC applique cet algorithme avec la clé associée aux données à chiffrer et à déchiffrer. Le chiffrement est dit symétrique, car c'est la même clé qui est utilisée pour chiffrer les données (ou lignes SQL) et déchiffrer les données.

### 5.3.2 Providers

SafeJDBC permet de définir le **Provider** de cryptographie, c'est à dire la couche basse de cryptographie utilisée:

Provider	Remarque	Constante SafeJDBC (package com.safejdbc.api)
Sun JCE	Cf. <a href="http://java.sun.com/products/jce/">http://java.sun.com/products/jce/</a>	Provider.SUNJCE
Cryptix	Cf. <a href="http://www.cryptix.org/">http://www.cryptix.org/</a>	<b>Provider.CRYPTIX</b> (Provider par défaut)

#### Notes :

- L'utilisation du Provider Cryptix ne nécessite aucun paramétrage supplémentaire et est recommandée, car le code source est disponible et étudiable.
- L'utilisation du Provider Sun JCE nécessite le téléchargement des fichiers «JCE Unlimited Strength Jurisdiction Policy Files ». Cf. <http://java.sun.com/products/jce/> pour plus d'informations.

### 5.3.3 Algorithmes

Les algorithmes incluent le Mode et le Padding.

Voici la liste des algorithmes disponibles dans SafeJDBC et la constante SafeJDBC associée :

Algorithme	Mode	Padding	Constante SafeJDBC (package com.safejdbc.api)
Blowfish	CBC	Aucun	Algo.BLOWFISH_CBC
		PKCS#5	Algo.BLOWFISH_CBC_PKCS5
	CFB	Aucun	<b>Algo.BLOWFISH_CFB</b> (Algorithme par défaut)
CAST	ECB	Aucun	Algo.CAST_128_ECB
	CBC	Aucun	Algo.CAST_128_CBC
		PKCS#5	Algo.CAST_128_CBC_PKCS5
CFB	Aucun	Algo.CAST_128_CFB	
IDEA <sup>1</sup>	CBC	Aucun	Algo.IDEA_CBC
		PKCS#5	Algo.IDEA_CBC_PKCS5
	CFB	Aucun	Algo.IDEA_CFB

Notes :

- La clé de chiffrement est toujours de **128 bits**.
- L'algorithme par défaut est **Algo.BLOWFISH\_CFB**.

### 5.3.4 setProvider() - Paramétrage du Provider à utiliser

Méthode	SafeJdbcSetter.setAlgorithm()		
Description	Paramétrage du Provider de cryptographie à utiliser.		
Arguments	String	sProvider	Nom du Provider de cryptographie.
Retour	Aucun.		

Note :

- L'appel à cette méthode n'est pas obligatoire. Le Provider par défaut est Cryptix (**Provider.CRYPTIX**).

Exemple :

```
// Set SafeJdbc's Provider to use
sjSetter.setProvider(Provider.CRYPTIX);
```

<sup>1</sup> IDEA est breveté par ASCOMM et nécessite une licence spéciale auprès de SafeLogic pour être utilisable.

### 5.3.5 setAlgorithm() - Paramétrage de l'algorithme à utiliser

Méthode	SafeJdbcSetter.setAlgorithm()		
Description	Paramétrage de l'algorithme de chiffrement symétrique à utiliser.		
Arguments	String	sAlgorithm	Algorithme de chiffrement symétrique à utiliser, incluant le Mode et le Padding.
Retour	Aucun.		

Note :

- L'appel à cette méthode n'est pas obligatoire. La combinaison algorithme/Mode/padding par défaut sera Algo.BLOWFISH\_CFB

Exemple :

```
// Set SafeJdbc's algorithm to use
sjSetter.setAlgorithm(Algo.CAST_128_CBC_PKCS5);
```

### 5.3.6 setKey() - Paramétrage de la clé secrète de chiffrement

L'API **setKey()** permet de charger en mémoire dans la JVM la clé secrète à utiliser.

Il existe deux moyens de définir une clé de chiffrement dans SafeJDBC :

1. La clé de chiffrement est récupérée via une valeur d'objet Java **java.security.Key** à partir d'un programme externe à SafeJDBC. (Par exemple, le toolkit SafeAPI de SafeLogic permet de générer et stocker de façon sécurisée des clés secrètes. Cf. <http://www.safelogic.com/safeapi>.)
2. La clé de chiffrement est directement dérivée à partir d'une **passphrase** en utilisant une fonction de hachage MD5.

Méthode	SafeJdbcSetter.setKey()		
Description	Chargement dans SafeJDBC d'une clé symétrique au format Key de Java.		
Arguments	String	sUserId	Identifiant de l'utilisateur SafeJDBC.
	Key	key	Clé symétrique de 128 bits au format Key de Java.
Retour	Aucun.		

Notes :

- **sUserId** est un identifiant arbitraire à choisir. Il sert à « relier » le triplet (Utilisateur, algorithme, clé) pour sécuriser les utilisations suivantes. Le triplet est en effet stocké par SafeJDBC à la première utilisation. A partir de la deuxième utilisation, SafeJDBC vérifie que les valeurs (Utilisateur, algorithme, clé) sont conformes à ce qui est stocké. Ce principe permet d'éviter l'utilisation en déchiffrement d'algorithmes, clés ou passphrases qui ne correspondent pas à ceux du premier chiffrement effectué.
- La valeur de **key** doit être protégée et conservée secrète afin d'assurer la sécurité de SafeJDBC.

Exemple :

```
// Set SafeJdbc's secret Key from an external program/store
Key key = getKeyFromSomeStore(key_reference);
sjSetter.setKey("safejdbc", key);
```

Méthode	SafeJdbcSetter.setKey()		
Description	Chargement dans SafeJDBC d'une clé symétrique au format Key de Java dérivée d'une passphrase .		
Arguments	String	sUserId	Identifiant de l'utilisateur SafeJDBC.
	Char []	caPassphrase	Passphrase secrète à utiliser pour fabriquer la clé secrète au format Key de Java
Retour	Aucun.		

Exemple :

```
// Set SafeJdbc's secret Key from a passphrase
char[] caPassphrase= {'s', 'a', 'f', 'e', '*', 'l', 'o', 'g', 'i', 'c', ':', '!'};
sjSetter.setKey("safejdbc", caPassphrase);
```

Note :

- La clé de chiffrement est à chaque fois refabriquée « dynamiquement » à partir de la passphrase.
- La valeur de **caPassphrase** doit être protégée et conservée secrète afin d'assurer la sécurité de SafeJDBC.

## 5.4 Appel de SafeJDBC en mode pur JDBC Driver

### 5.4.1 Principe

SafeJDBC peut être appelé comme n'importe quel Driver, sans programmation des paramètres de SafeJdbcSetter en Java. Les paramètres sont contenus dans un fichier ASCII qui **doit être dans le CLASSPATH**.

Le nom par défaut du fichier de paramètres est `con_safejdbc.ini`. La variable d'environnement ( ou System Property) `SAFEJDBC_SETTER_INI` permet de définir un nom différent au moment de l'appel du lancement de la JVM :

```
Java -DSAFEJDBC_SETTER_INI=my_safejdbc_ini_name.ini MyProgram
```

L'appel du Driver SafeJDBC se fait alors normalement, le nom du Driver étant « `com.safejdbc.api.Driver` » :

```
// Load SafeJDBC as a pure Driver & get a Cipher Connection to the database
String sDriver = "com.safejdbc.api.Driver";
Class.forName(sDriver).newInstance();
Connection connection = DriverManager.getConnection(sDbUrl, prop);
```

### 5.4.2 Paramétrage de `con_safejdbc.ini`

A chaque méthode « set » de la classe SafeJdbcSetter correspond un ou plusieurs paramètres à définir dans `con_safejdbc.ini` sous la forme :

```
# Ligne de commentaire
NOM_PARAMETRE = valeur_parametre
```

Il n'est pas nécessaire de définir les paramètres qui ont des valeurs par défaut définies dans les méthodes « set » de SafeJdbcSetter. Voici la liste des paramètres de con\_safejdbc.ini :

Nom du paramètre	Rôle & Méthode correspondante de SafeJdbcSetter	Définition Obligatoire
DRIVER	Nom du Driver natif à « wrapper ». SetWrappedDriverName()	Oui
JDBC_URL	URL de la database SAFEJDBC_CATALOG. SetSafeJdbcDbUrl()	Oui
USER	User SQL d'accès à SAFEJDBC_CATALOG. SetSafeJdbcDbUrl()	Oui
PASSWORD	Password SQL d'accès à SAFEJDBC_CATALOG. SetSafeJdbcDbUrl()	Oui
SAFEJDBC_USERID	Userid SafeJDBC propriétaire de la clé ou de la passphrase. SetKey()	Oui
SAFEJDBC_KEY	Clé de chiffrement au format hexadécimal. SetKey()	Oui (un des deux)
SAFEJDBC_PASSPHRASE	Passphrase de chiffrement. SetKey()	
SAFEJDBC_LOGFILE	Fichier de Logging. SetSafeJdbcLogFile()	Non
SAFEJDBC_PROVIDER	Provider de Cryptographie. SetProvider()	Non
SAFEJDBC_ALGORITHM	Algorithme de Cryptographie. SetAlgorithm()	Non

Notes :

- Le fichier con\_safejdbc.ini est localisé automatiquement par SafeJDBC qui le recherche sans cet ordre :
  - Dans le répertoire correspondant à la variable d'environnement user.home.
  - Dans les répertoires définis dans le CLASSPATH.

Si plusieurs fichiers existent, SafeJDBC récupère le contenu du premier accédé.

## 5.5 Vérification du statut sécurisé d'une Connection

Une Connection sécurisée qui permet de chiffrer/déchiffrer de façon transparente est une instance de la classe **com.safejdbc.java.sql.CipherConnection**.

L'opérateur **instanceOf** permet de vérifier que la Connection a bien été chargée par SafeJDBC. Exemple :

```
import com.safejdbc.java.sql.CipherConnection;
// Etc...

if (connection instanceof CipherConnection)
{
    System.out.println("Good! This is a Cipher Connection.");
}
else
{
    System.out.println("Bad! This is *not* a Cipher Connection.");
}
```

## 5.6 Récupération de la Connection « native »

Il est en permanence possible de récupérer la Connection « native » et non-chiffrée pour accéder à la database applicative.

Il suffit d'appeler la méthode **getWrappedConnection()** sur une instance de **com.safejdbc.java.sql.CipherConnection** :

```
// Try to get back the native/wrapped Connection
// from the Cipher Connection
if (connection instanceof CipherConnection)
{
    Connection conWrapped = null;

    // getWrapped will get the native/wrapped Connection
    // to application database
    conWrapped = ((CipherConnection) connection).getWrappedConnection();

    ...
}
```

La récupération de la Connection « native » permet d'optimiser les performances d'accès des requêtes qui ne contiennent que des tables et colonnes non chiffrées. L'utilisation de **getWrappedConnection()** doit être le moyen préféré d'accéder les tables SQL qui ne contiennent aucune colonne chiffrée par SafeJDBC.

## 5.7 CipherConnectionFactory – Api de création d'une «Connection»

Il peut être nécessaire de créer une Connection sécurisée sans accéder à la classe « Driver ». Ceci est le cas avec les gestionnaires de Pool de Connections.

**CipherConnectionFactory** permet de créer une Connection sécurisée (de type CipherConnection) , sans charger de Driver et en utilisant une Connection « native » à la database applicative et qui est encapsulée.

Comme chaque CipherConnection encapsule une Connection native ce procédé autorise l'utilisation de mécanismes avancés d'établissement de chargements de « Pool » de Connections sécurisées.

Une Connection Sécurisée CipherConnection est construite en utilisant deux Connection :

1. Une première Connection à la database de l'application.
2. Une seconde Connection à la database du Catalogue SafeJDBC.

L'appel à **CipherConnectionFactory** doit être précédé des paramétrages avec SafeJdbcSetter ou via un fichier ini pour définir :

- Un Userid SafeJDBC.
- Une clé (Key Java ou valeur de Passphrase).

L'exemple suivant utilise les paramètres de cryptographie définis dans le fichier con\_safejdbc.ini :

```
// This is the application database url
String sDbUrl = "jdbc:mysql://localhost:3306/sj_clients";

// These are the application database username & password
// Properties propsApp = new Properties();
Properties propsApp = new Properties();
propsApp.put("user", "safelogic");
propsApp.put("password", "safelogic*$");

// This is the SafeJdbc Catalog database url
String sJdbcDbUrl = "jdbc:mysql://localhost:3306/safejdbc_store";

// Set SafeJdbc's own catalog properties
Properties propsSjdbc = new Properties();
propsSjdbc.put("user", "safelogic");
propsSjdbc.put("password", "safelogic*$");

// Load the native Driver and get a normal Connection to the application
// database & a Connection to the SafeJdbc Catalog database
// Note: this could be done using the DataSource mechanism in a J2EE
// environment.
```

```

Class.forName("org.gjt.mm.mysql.Driver").newInstance();

Connection conApplication = DriverManager.getConnection(sDbUrl, propsApp);
Connection conSafeJdbcCat
    = DriverManager.getConnection(sJdbcDbUrl, propsSjdbc);

// Get the Cipher Connection for encryption of statements and result
Connection connection =
    CipherConnectionFactory.getInstance(conApplication, conSafeJdbcCat);

```

## 5.8 Création de Connection sécurisée à partir de DataSources

### 5.8.1 Principe Général

Les serveurs d'application J2EE/EJB modernes gèrent des fonctions avancées de « Connection Pooling ».

Ces fonctions avancées sont implémentées via les classes Interfaces du package **javax.sql** et sont accessibles par la création d'objets de type **DataSource**, et en utilisant les mécanismes de « lookup » dans la syntaxe JNDI.

Les Connection sont alors accessibles avec la méthode `DataSource.getConnection()`:

Voici une Connection établie avec la database applicative en utilisant un DataSource identifié par son nom JDNI «jdbc/application ».

```

// Get a Connection to application Database.
//
InitialContext initCtx = new InitialContext();

// Get a DataSource for connection to application database
DataSource ds
    = (DataSource) initCtx.lookup("java:/comp/env/jdbc/application");

// Get the Connection from the DataSource.
// Connection is extracted from the Connection Pool:
Connection connection = ds.getConnection() ;

// Etc...

```

### 5.8.2 Le JavaBean SafeDataSourceFactory

Pour établir une Cipher Connection à la database précédente, sans changement de code source, SafeJDBC fournit le « JavaBean » **factory** **com.safejdbc.javax.sql.SafeDataSourceFactory**.

**SafeDataSourceFactory** permet de “wrapper” de façon transparente les deux DataSources qui correspondant à la database applicative et à la database du Catalogue SafeJDBC.

**SafeDataSourceFactory** prend comme paramètres :

- `base_jndi_context`:  
Le nom du Contexte de Base JNDI utilisé pour le « lookup » des noms de `DataSource`. Ce paramètre est facultatif et a pour valeur par défaut `"java:/comp/env/"`.
- `ds_wrapped_lookup_name`:  
Le nom de « lookup » dans la syntaxe JNDI du `DataSource` de la database applicative.
- `ds_safejdbc_catalog_lookup_name`:  
Le nom de « lookup » dans la syntaxe JNDI du `DataSource` de la database du Catalogue SafeJDBC.

Il suffit alors de faire « pointer » dans le Serveur d'Application la factory **SafeDataSourceFactory** sur le `DataSource` « `env/jdbc/application` » pour établir une Connection sécurisée de type **CipherConnection**.

## 5.9 Exemple d'utilisation de DataSources avec Tomcat 5.0

Dans le cas de Tomcat 5.0 (et de tous les serveurs d'application qui respectent le standard J2EE), les `DataSources` implémentent les fonctions avancées de Connection Pooling.

Voici le mode général de configuration de `DataSource` avec Tomcat 5.0 :

- La définition d'un `DataSource` est définie dans `server.xml` via un tag `<Resource>`. Cette définition inclut le paramétrage du Pool de Connection.
- Le nom `DataSource` est récupéré dans le programme Java via le mécanisme de lookup de JNDI dans l'espace `"java:/comp/env/jdbc/"`.
- Le `DataSource` est créé par instanciation d'une class « factory » de nom **`org.apache.commons.dbcp.BasicDataSourceFactory`**.

La configuration SafeJDBC pour créer un `DataSource` sécurisé qui gère le « Connection Pooling » se fait en trois étapes.

### 5.9.1 Etape 1 : Définition du DataSource de l'application

Cette étape permet de définir le nom de la Resource qui correspond au DataSource d'accès à la database applicative. Voici un exemple à insérer dans `server.xml` :

```
<!--Step 1 : this is the normal connection pool to application database -->
<Resource name="jdbc/application" auth="Container"
          type="javax.sql.DataSource"/>

  <ResourceParams name="jdbc/application_wrapped">
    <parameter>
      <name>factory</name>
      <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
    </parameter>

    <parameter>
      <name>maxActive</name>
      <value>100</value>
    </parameter>

    <parameter>
      <name>maxIdle</name>
      <value>30</value>
    </parameter>

    <parameter>
      <name>maxWait</name>
      <value>10000</value>
    </parameter>

    <!-- MySQL sj_clients username and password for dB connections -->
    <parameter>
      <name>username</name>
      <value>safelogic</value>
    </parameter>

    <parameter>
      <name>password</name>
      <value>safelogic*${</value>
    </parameter>

    <!-- Class name for the official MySQL Connector/J driver -->
    <parameter>
      <name>driverClassName</name>
      <value>com.mysql.jdbc.Driver</value>
    </parameter>

    <!-- The sj_clients example database -->
    <parameter>
      <name>url</name>
<value>jdbc:mysql://localhost:3306/sj_clients?autoReconnect=true</value>
    </parameter>
  </ResourceParams>
```

## 5.9.2 Etape 2 : Définition du DataSource du Catalogue SafeJDBC

On définit ensuite dans `server.xml` le DataSource d'accès à la database Catalogue SafeJDBC (`safejdbc_catalog`) :

```
<!-- Step 2: Add the jdbc/safejdbc_catalog Resource
This is the connection pool to safejdbc catalog database -->

<Resource name="jdbc/safejdbc_catalog" auth="Container"
type="javax.sql.DataSource"/>

  <ResourceParams name="jdbc/safejdbc_catalog">
    <parameter>
      <name>factory</name>
      <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
    </parameter>

    <parameter>
      <name>maxActive</name>
      <value>100</value>
    </parameter>

    <parameter>
      <name>maxIdle</name>
      <value>30</value>
    </parameter>

    <parameter>
      <name>maxWait</name>
      <value>10000</value>
    </parameter>

    <!-- MySQL safejdbc_catalog dB username and password for dB -->
    <parameter>
      <name>username</name>
      <value>safelogic</value>
    </parameter>

    <parameter>
      <name>password</name>
      <value>safelogic*${</value>
    </parameter>

    <!-- Class name for the official MySQL Connector/J driver -->
    <parameter>
      <name>driverClassName</name>
      <value>com.mysql.jdbc.Driver</value>
    </parameter>

    <!-- The SafeJDBC Catalog database -->
    <parameter>
      <name>url</name>

<value>jdbc:mysql://localhost:3306/safejdbc_catalog?autoReconnect=true</val
ue>
    </parameter>
  </ResourceParams>
```

### 5.9.3 Etape 3 : Définition du SafeDataSource sécurisé

On définit pour terminer le **DataSource** sécurisé qui va « wrapper » les deux DataSources database applicative et database Catalogue SafeJDBC.

Ce DataSource permettra de créer un DataSource qui sera une instance de **com.safejdbc.javax.sql.SafeDataSource**, exactement suivant le principe d'une Connection sécurisée qui est une instance de **CipherConnection** :

```
<!--
Step 3: Add the new encrypted jdbc/application Resource.

This is the Cipher Connection pool.
Is will use the two previous connection pool defined by their
Resource/lookup Name:
    - 1) jdbc/application_wrapped
    - 2) jdbc/safejdbc_catalog
-->

<Resource name="jdbc/application" auth="Container"
type="javax.sql.DataSource" />

    <ResourceParams name="jdbc/application">

        <!-- We are using a special SafeJDBC factory -->
        <parameter>
            <name>factory</name>
            <value>com.safejdbc.javax.sql.SafeDataSourceFactory</value>
        </parameter>

        <parameter>
            <name>ds_wrapped_lookup_name</name>
            <value>jdbc/application_wrapped</value>
        </parameter>

        <parameter>
            <name>ds_safejdbc_catalog_lookup_name</name>
            <value>jdbc/safejdbc_catalog</value>
        </parameter>

    </ResourceParams>
```

On peut alors compléter le code source défini en [5.8.1 Principe Général](#) pour vérifier que la Connection est bien sécurisée (les parties ajoutées sont en **gras**):

```
// Get a Connection to application Database.

InitialContext initCtx = new InitialContext();

// Get a DataSource for connection to application database
DataSource ds
    = (DataSource) initCtx.lookup("java:/comp/env/jdbc/application");

// Get the Connection from the DataSource.
// Connection is extracted from the Connection Pool:
Connection connection = ds.getConnection() ;

if (connection instanceof CipherConnection)
{
    System.out.println("OK! Cipher Connection extracted from the pool!");
}

// Etc...
```

Les Connection sécurisées ainsi obtenues sont gérées en utilisant intégralement le mécanisme de « Connection Pooling » implémenté par Tomcat 5.0.

## 5.10 ColumnsCipher - API de sélection des colonnes à chiffrer/déchiffrer

### 5.10.1 Généralités

Les API de sélection des colonnes à chiffrer/déchiffrer permettent :

- De définir pour SafeJDBC une liste de colonnes à chiffrer (ou déchiffrer).
- De paramétrer le flux de sortie pour l'affichage de l'exécution de la classe.
- De lancer l'exécution des instructions SQL d'UPDATE qui correspond à la liste des colonnes choisies afin de chiffrer ou déchiffrer les valeurs de celles-ci..

La classe **com.safejdbc.api.ColumnsCipher** est utilisée pour paramétrer la connexion Java (Connection) sécurisée.

Le Driver SafeJDBC doit être chargé dans la classe appelante **avant** les appels à **ColumnsCipher**.

### 5.10.2 getInstance () - Création d'une instance de ColumnsCipher

Méthode	ColumnsCipher.getInstance()		
Description	Retourne une instance de la classe ColumnsCipher.		
Arguments	Connection	conCipher	Connection créée par le Driver SafeJDBC.
Retour	Une instance de la classe ColumnsCipher.		

Exemple :

```
ColumnsCipher columnsCipher = ColumnsCipher.getInstance(cipherConnection)
```

L'import suivant doit être déclaré :

```
import com.safejdbc.api.ColumnsCipher;
```

### 5.10.3 setOutputStream() - Paramétrage du flux de sortie

Ce paramétrage permet de définir la zone de sortie pour le déroulement de l'exécution :

Méthode	ColumnsCipher.setOutputStream()		
Description	Définit un OutputStream pour le flux de sortie, i.e. les informations de déroulement de l'exécution de la classe.		
Arguments	OutputStream	outputStream	OutputStream à utiliser pour afficher le déroulement.
Retour	Aucun.		

Exemple (Sortie sur la console) :

```
columnsCipher.getOutputStream(System.out);
```

#### 5.10.4 setFileOutputStream() - Paramétrage du flux de sortie sur un fichier

Méthode	ColumnsCipher.setFileOutputStream()		
Description	Définit un fichier pour le flux de sortie, i.e. les informations de déroulement de l'exécution de la classe.		
Arguments	String	sFile	Nom du fichier de sortie à utiliser pour afficher le déroulement.
Retour	Aucun.		

Exemple :

```
columnsCipher.setFileOutputStream("c:\\temp\\cipher.output.txt");
```

#### 5.10.5 addColumn() - Ajout d'une colonne

Les colonnes à chiffrer sont à ajouter une par une avec la méthode **addColumn()** qui supporte deux syntaxes pour préciser le nom de table :

##### Syntaxe 1

Méthode	ColumnsCipher.addColumn()		
Description	Ajout d'une colonne à chiffrer/déchiffrer sur la pile.		
Arguments	String	sColumn	Nom de la colonne avec la syntaxe « table.colonne ».
Retour	Aucun.		

Exemple :

```
columnsCipher.addColumn("clients.cli_lastname");
columnsCipher.addColumn("clients.cli_firstname");
```

##### Syntaxe 2

Méthode	ColumnsCipher.addColumn()		
Description	Ajout d'une colonne à chiffrer/déchiffrer sur la pile.		
Arguments	String	sTable	Nom de la table.
	String	sColumn	Nom de la colonne.
Retour	Aucun.		

Exemple :

```
columnsCipher.addColumn("clients", "cli_lastname");
columnsCipher.addColumn("clients", "cli_firstname");
```

### 5.10.6 addColumnsFromFile() - Ajout des colonnes à partir d'un fichier «ini»

Méthode	ColumnsCipher.addColumnsFromFile()		
Description	Ajout d'une liste de colonnes à partir d'un fichier « ini ».		
Arguments	String	sIniFile	Nom du fichier, sans le chemin d'accès, qui contient les colonnes à ajouter avec la syntaxe "table.colonne".
Retour	Aucun.		

Notes :

- Le fichier « ini » doit contenir une colonne par ligne, avec la syntaxe **"table.colonne"**.
- Le caractère # est utilisé comme commentaire en début de ligne.

Voici un exemple valide :

```
# cipher_columns.ini
# Register all the cipher columns of a database

# keep secret clients personal infos!
clients.cli_lastname
clients.cli_firstname
clients.cli_email
clients.cli_phone

# keep secret clients credit cards infos!

credit_cards.cb_type
credit_cards.cb_number
credit_cards.cb_expiration_date

# keep secret sales infos!

sales.cli_lastname
sales.cli_firstname
sales.sa_order
sales.sa_cli_remarks

#end
```

### 5.10.7 defineJoinColumn() – Définition de jointures chiffrées

**defineJoinColumn()** permet de préciser les jointures entre colonnes chiffrées.

Une jointure entre deux colonnes chiffrées se définit par paire en précisant :

- La colonne de **Jointure**.
- La colonne de **Référence**

Avec les règles pour éviter les erreurs (non existence, références circulaires, etc.)

1. Les deux colonnes doivent avoir été au préalable définies avec une fonction **addColumn()** ou **addColumnFromFile()**.
2. Une colonne de Jointure ne peut être aussi définie comme une colonne de Référence.
3. Une colonne de Référence ne peut être aussi définie comme une colonne de Jointure.

**defineJoinColumn()** supporte deux syntaxes pour préciser le nom des tables de Jointure et de Référence :

#### Syntaxe 1

Méthode	ColumnsCipher.defineJoinColumn()		
Description	Définition d'une colonne de Jointure chiffrée.		
Arguments	String	sJoinColumn	Nom de la colonne de Jointure avec la syntaxe « table.colonne ».
	String	sRefColumn	Nom de la colonne de Référence avec la syntaxe « table.colonne ».
Retour	Aucun.		

Exemple :

```
columnsCipher.defineJoinColumn("sales.cli_lastname",
                               "clients.cli_lastname");
columnsCipher.defineJoinColumn("products.cli_lastname",
                               "clients.cli_lastname");
```

## Syntaxe 2

Méthode	ColumnsCipher.defineJoinColumn()		
Description	Définition d'une colonne de Jointure chiffrée.		
Arguments	String	sTableJoin	Nom de la table de Jointure.
	String	sJoinColumn	Nom de la colonne de Jointure.
	String	sTableRef	Nom de la table de Référence.
	String	sRefColumn	Nom de la colonne de Référence.
Retour	Aucun.		

Exemple :

```
columnsCipher.defineJoinColumn("sales", "cli_lastname",
                               "clients", "cli_lastname");
columnsCipher.defineJoinColumn("products", "cli_lastname",
                               "clients", "cli_lastname");
```

### 5.10.8 defineJoinColumnsFromFile() - Définition des colonnes de Jointure à partir d'un fichier «ini»

Méthode	ColumnsCipher.defineJoinColumnsFromFile()		
Description	Définition d'une liste de colonnes de Jointures à partir d'un fichier « ini ».		
Arguments	String	sIniFile	Nom du fichier, sans le chemin d'accès, qui contient les colonnes de Jointure et de Référence sous le format :  Table1.colonne_jointure = Table2.colonne_ref  Le signe « = » servant de séparateur.
Retour	Aucun.		

Notes :

- Le fichier « ini » doit contenir une colonne par ligne, avec la syntaxe " **table1.colonne\_jointure = table2.colonne\_ref** ".
- Le caractère # est utilisé comme commentaire en début de ligne.
- Les colonnes de Jointures/Référence peuvent être spécifiées dans le même fichier que les colonnes chiffrées.

Voici un exemple valide, ajouté à la fin du fichier ini précédent de définition des colonnes chiffrées :

```
# cipher_columns.ini
# Register all the cipher columns of a database
# and add Join and Reference columns

# keep secret clients personal infos!
clients.cli_lastname
clients.cli_firstname
clients.cli_email
clients.cli_phone

# keep secret clients credit cards infos!

credit_cards.cb_type
credit_cards.cb_number
credit_cards.cb_expiration_date

# keep secret sales infos!

sales.cli_lastname
sales.cli_firstname
sales.sa_order
sales.sa_cli_remarks

# define Join and Reference columns:

sales.cli_lastname = clients.cli_lastname
sales.cli_firstname = clients.cli_firstname

#end
```

### 5.10.9 execute() – Lancement du process de chiffrement ou de déchiffrement

**CipherColumns.execute()** lance le process de chiffrement ou de déchiffrement, avec deux étapes distinctes :

- 1) **Définition** : Les tables de la méta\_database **safejdbc\_catalog** sont remises à jour en fonction des colonnes spécifiées précédemment avec les fonctions **addColumn()** et **defineJoinColumn()**.
- 2) **Basculement des valeurs** : Sia la database applicative contient des valeurs, les valeurs à chiffrer (ou déchiffrer) des colonnes choisies sont mises à jour par génération et lancement automatique d'instructions SQL de type :  

```
UPDATE TABLE SET COLUMN = NEW_VALUE WHERE COLUMN = OLD_VALUE
```

La phase de **définition** doit être faite  **systématiquement**  avant l'utilisation de SafeJDBC, c'est pourquoi la fonction **execute()** doit être  lancée même si la database applicative est vide.

Méthode	ColumnsCipher.execute()		
Description	Lancement du process de chiffrement/déchiffrement des colonnes spécifiées avec addColumn()		
Arguments	int	nOpMode	Mode Opérateur. Chiffrement ou déchiffrement : <ul style="list-style-type: none"> <li>• ColumnsCipher.ENCRYPT_MODE</li> <li>• ColumnsCipher.DECRYPT_MODE</li> </ul>
Retour	Aucun.		

## Notes

- Il est recommandé de faire une sauvegarde de la database applicative et de **safejdbc\_catalog** avant le lancement de chaque **execute()**.
- Les opérations sont toujours réversibles, i.e. les instructions **execute(ColumnsCipher.ENCRYPT\_MODE)** et **execute(ColumnsCipher.DECRYPT\_MODE)** peuvent être lancées de façon alternée.
- Il n'est pas possible de spécifier deux fois de suite le même mode chiffrement ou déchiffrement sur une colonne.
- SafeJDBC inclut des sécurités qui permettent de bloquer une tentative erronée de « double chiffrement » ou de « double déchiffrement ». Un message d'erreur est alors produit sur la console.

Exemple :

```
columnsCipher.execute(ColumnsCipher.ENCRYPT_MODE);
```

## 5.11 Implémentation de ColumnsCipher : Batch ColumnsCipherMain

Le programme java `com.safejdbc.bat.ColumnsCipherMain` contient une implémentation avec paramètres de l'API `ColumnsCipherMain`.

Ce programme est auto-documenté et propose l'aide suivante lorsque il est appelé sans paramètres :

```
c:\Test>java com.safejdbc.bat.ColumnsCipherMain
Usage:
java com.safejdbc.bat.ColumnsCipherMain <parms>

parms: <mode> <con_application.ini> <con_safejdbc.ini> <cipher_columns.ini>
      <passphrase>
with:
- mode           : (e)ncrypt | (d)encrypt
- con_application.ini: ini file with JDBC connection parms
                  to application database.
- con_safejdbc.ini  : ini file with JDBC connection parms
                  to SafeJDBC database.
- cipher_columns.ini : ini file containing columns to encrypt/decrypt.
- passphrase       : the passphrase for crypto operations.
notes:
- Pass raw ini file names, without directory name.
- The ini files must be located in the CLASSPATH.

c:\Test>
```

Voici des exemples de contenu des fichiers ini de configuration SQL :

```
# con_application.ini
#
# ini file for com.safejdbc.bat.ColumnsCipherMain batch.
# contains connections parameters to application database.

# JDBC driver Class
DRIVER=org.gjt.mm.mysql.Driver

# DB URI root
DB_URI_ROOT=jdbc:mysql://localhost:3306/sj_clients?

# Login & password to use for database connection
USER=safelogic
PASSWORD=safelogic*$

#end
```

```
# con_safejdbc.ini
#
# ini file for com.safejdbc.bat.ColumnsCipherMain batch
# contains connections parameters to the SafeJDBC catalog

# JDBC driver Class
DRIVER=org.gjt.mm.mysql.Driver

# URL to SafeJDBC database
DB_URI_ROOT=jdbc:mysql://localhost:3306/safejdbc_catalog?

# SafeJDBC database user & passwords
USER=safelogic
PASSWORD=safelogic*$

#end
```

## 5.12 Comment définir des colonnes chiffrées dans des VUES

SafeJDBC v2.0 permet de définir des colonnes chiffrées dans des VUES en utilisant **addColumnFromFile()** et **defineJoinColumnFromFile()**.

Les étapes sont :

1. Créer en premier la VUE avec **CREATE VIEW** (*sans* clause **WHERE** sur des colonnes chiffrées)
2. Ajouter chaque colonne chiffrée dans le fichier .ini.
3. Ajout dans le fichier .ini chaque colonne chiffrée comme si c'était une colonne de jointure.
4. **ColumnsCipher.execute()**
5. Seulement si vous voulez rajouter des colonnes chiffrées dans la clause **WHERE** :
  - DROP de la VUE.
  - Utiliser un programme Java qui appelle SafeJDBC comme Driver pour re-crée la VUE.

### Exemple:

Nous voulons créer la VUE suivante qui contient une colonne chiffrée dans la clause **WHERE** :

```
create view myview as select cli_lastname, cli_firstname
      from clients where cli_lastname = 'SMITH';
```

### Etape 1

Arrêt de toutes les instances de SafeJDBC (Serveur J2EE, programmes Java, etc.) et lancement en SQL interactif de :

```
/* Note that there is no WHERE clause */
create view myview as select cli_lastname, cli_firstname
from clients;
```

## Etapes 2 et 3

Ajout des colonnes chiffrées de la VUE dans le fichier .ini :

```
# cipher_columns.ini
# Register all the cipher columns of a database
# and add Join and Reference columns
# add VIEW myview on CLIENTS table

# keep secret clients personal info!
clients.cli_lastname
clients.cli_firstname
clients.cli_email
clients.cli_phone

# keep secret myview!
myview.cli_lastname
myview.cli_firstname

# keep secret clients credit cards info!

credit_cards.cb_type
credit_cards.cb_number
credit_cards.cb_expiration_date

# keep secret sales info!

sales.cli_lastname
sales.cli_firstname
sales.sa_order
sales.sa_cli_remarks

# define Join and Reference columns:

sales.cli_lastname = clients.cli_lastname
sales.cli_firstname = clients.cli_firstname

#define the View myview columns Reference columns
myview.cli_lastname = clients.cli_lastname
myview.cli_firstname = clients.cli_firstname

#end
```

## Etape 4

Lancer `ColumnsCipher.execute()`

ou

```
java com.safejdbc.bat.ColumnsCipherMain e
```

## Step 5 (seulement si des colonnes chifrées sont utilisées dans la clause WHERE)

En SQL interactif :

```
DROP VIEW myview;
```

Re-cr er la VUE avec la clause WHERE en utilisant obligatoirement SafeJDBC comme Driver :

```
// Create a Secure Connection
...
if (connection instanceof CipherConnection)
{
    System.out.println("Good! This is a Cipher Connection.");
}
else
{
    System.out.println("Bad! This is *not* a Cipher Connection.");
    return;
}

String sStatement =
    "create view myview as select cli_lastname, cli_firstname "
    + "from clients where cli_lastname = 'Smith'";
Statement stat = connection.createStatement();
stat.execute(sStatement);

// End
```

## 6 SUPPORT

Pour toute question technique, veuillez vous adresser à :

SafeLogic  
27/29, rue Raffet  
75016 Paris

Tél : (33) (0)1 45 72 25 15

Fax : (33) (0)1 45 72 14 06

E mail : [contact@safelogic.com](mailto:contact@safelogic.com)

Web : <http://www.safelogic.com>

-----