



Auteur : Patrick BON  
Éric LEMIÈRE

Référence :

Version : 1.0

Date : 30/06/2000

État : Approuvée

Approbateur : Guy AUTIER

Philippe CHAPSAL

**CELAR/CASSI**  
**Étude des logiciels libres**  
**Rapport d'étude 7-9 : Sécurisation des SGBDR**

## Historique des révisions

<b>N°</b>	<b>Date</b>	<b>Auteur</b>	<b>Contenu</b>
0.1	23/05/00	Patrick BON	Création
0.2	25/05/00	Patrick BON	Prise en compte des relectures internes.
0.3	27/06/00	Patrick BON	Rédaction du rapport.
1.0	30/06/00	Patrick BON	Prise en compte des relectures internes.

## Table des matières

1.	Introduction	5
1.1	Objet du document	5
1.2	Documents de référence	5
1.3	Structure du document	5
2.	Présentation préliminaire des objets de l'étude	6
2.1	SGBD étudiés	6
2.1.1	Caractéristiques générales de MySQL	6
2.1.2	Caractéristiques générales de PostgreSQL	7
2.1.3	Comparaison succincte des deux produits	8
2.2	Critères de sécurité	9
3.	Évaluation des produits	10
3.1	Fonctions natives de sécurité des produits	10
3.1.1	Identification / authentification des utilisateurs	10
3.1.2	Contrôle d'accès aux données	14
3.1.3	Contrôle d'accès aux services d'administration	17
3.1.4	Protection des échanges réseau	20
3.1.5	Protection des données stockées	22
3.1.6	Intégrité du logiciel	24
3.1.7	Services d'audit	25
3.2	Éléments de robustesse des produits	26
3.2.1	Tolérance aux anomalies	27
3.2.2	Tolérance à la congestion	27
3.2.3	Qualité du code source	28
3.2.4	Problèmes connus et corrections	28
3.3	Extensions fonctionnelles des produits	29
3.3.1	Extensions dynamiques	29
3.3.2	Modification du code source	30
3.4	Résultats de l'évaluation	30
3.4.1	Tableau d'évaluation des produits	30

3.4.2	Synthèse sur les lacunes et vulnérabilités	32
3.4.3	Synthèse sur les solutions pour améliorer la sécurité	32
3.4.4	Préconisations	33
4.	<b>Maquette</b>	<b>35</b>
4.1	Objectifs de la maquette	35
4.1.1	Orientation générale	35
4.1.2	Mécanisme d'authentification mis en œuvre	35
4.2	Organisation de la maquette	36
4.3	Scénario de démonstration	36
4.4	Résultats	37
5.	<b>Conclusions de l'étude</b>	<b>38</b>

# 1. Introduction

## 1.1 Objet du document

Ce document constitue le rapport d'étude concernant la sécurisation (au sens de la sécurité des systèmes d'information) des systèmes de gestion de bases de données relationnels (SGBDR) disponibles en tant que logiciels libres.

Cette étude repose sur l'analyse de deux produits : MySQL et PostgreSQL.

## 1.2 Documents de référence

[Ref_MySQL]	MySQL Reference Manual  (documentation électronique disponible sur <a href="http://www.mysql.com">www.mysql.com</a> )
[Ref_PostgreSQL]	PostgreSQL  The PostgreSQL Development Team  Edited by Thomas Lockhart  (documentation électronique disponible sur <a href="http://www.postgresql.org">www.postgresql.org</a> )
[Concepts_PostgreSQL]	PostgreSQL : Introduction and Concepts  Bruce Momjian  <i>Ouvrage en cours de rédaction, destiné à être publié par Addison-Wesley</i>  (documentation électronique disponible sur <a href="http://www.postgresql.org/docs/awbook.html">www.postgresql.org/docs/awbook.html</a> )

## 1.3 Structure du document

Le document présente successivement :

- Les objets de l'étude : produits analysés et critères d'analyse (chapitre 2).
- L'analyse des produits, sur la base d'une recherche documentaire (chapitre 3).
- La maquette réalisée dans le cadre de l'étude (chapitre 4).
- Les conclusions du rapport d'étude (chapitre 5).

## 2. Présentation préliminaire des objets de l'étude

### 2.1 SGBD étudiés

#### 2.1.1 Caractéristiques générales de MySQL

MySQL est un serveur de base de données SQL multi-utilisateurs et multi-threads. Les principaux objectifs de MySQL sont la rapidité, la robustesse et la facilité d'utilisation.

MySQL a été développé par la société suédoise TcX, initialement pour répondre au besoin d'un serveur SQL qui puisse gérer des grandes bases de données, sur du matériel bon marché, de manière plus rapide que ce que pouvaient offrir les SGBD commerciaux de l'époque. Des bases de données MySQL sont opérationnelles depuis 1996 et le produit continue à évoluer. Le produit est disponible gratuitement sous Unix et pour les clients Windows. Le serveur est également disponible sous Windows NT, sous forme de shareware (une contribution annuelle de 200 dollars, incluant le support de TcX, est demandée au delà de 30 jours d'utilisation).

MySQL est architecturé selon une architecture client/serveur, sur la base d'un process démon serveur `mysqld`, de programmes clients et de bibliothèques. Le stockage des données est réalisé sous forme de B-Tree, à l'aide d'un gestionnaire ISAM.

Quelques caractéristiques intéressantes de MySQL :

- la disponibilité du produit sous forme du code source,
- un bon niveau de conformité à SQL 92,
- des capacités élevées en terme de dimensionnement (nombre de bases, de tables, taille des données...),
- des API pour divers langages de développement : C, C++, Eiffel, Java, Perl, PHP, Python et TCL et un driver ODBC, pour les clients Windows,
- la présence des procédures permettant l'exécution de modules (SQL et C) sur le serveur, avant de retourner la réponse à une requête,
- une conception permettant une optimisation poussée en terme de performances,
- une bonne intégration avec le produit Apache et le langage PHP qui constitue une base particulièrement intéressante pour le développement de serveur Web, sur la base de logiciels libres.

La principale limitation de MySQL concerne l'absence de services transactionnels (avec « roll back » en cas d'incident lors de l'exécution d'une transaction). Ce qui fait que, de ce point de vue, il ne s'agit pas d'un véritable SGBD relationnel. MySQL ne dispose pas non plus de vues, ni de « triggers ».

La version de MySQL étudiée dans le présent document est la **version 2.23**.

## 2.1.2 Caractéristiques générales de PostgreSQL

PostgreSQL est un SGBD « relationnel-objet » développé à l'origine à Berkeley, sous le nom de Postgres, par l'équipe du professeur Stonebraker, créateur de INGRES dans les années 80. Le développement (financé par la DARPA, la NSF et quelques autres grosses agences américaines) a été abandonné à Berkeley en 1994, et est maintenant poursuivi par une équipe de volontaires, à la mode « Open Source ». Vis-à-vis des systèmes d'exploitation, PostgreSQL a été validé avec la plupart des UNIX et avec Windows NT (à l'aide des bibliothèques RedHat/Cygnus pour la partie serveur).

Quelques caractéristiques intéressantes de PostgreSQL :

- une licence très ouverte (en fait sans aucune limitation, sauf l'indication que l'Université de Berkeley décline toute responsabilité ...),
- une architecture client-serveur basée sur un back-end (le serveur proprement dit) et des front-ends (qui prennent en compte chaque client) qui fonctionnent sur le serveur, et des clients pouvant fonctionner sur des machines distantes,
- bonne compatibilité SQL 92 / SQL 3 (mais les requêtes imbriquées ne sont pas supportées), avec en particulier la quasi totalité des types et des opérateurs,
- un index spatial intégré (B-Tree), avec de bonnes performances, tous les opérateurs nécessaires et même plus (quelques opérateurs topologiques),
- des drivers ODBC et JDBC, ce qui offre (en prime) des outils graphiques d'administration sous Windows,
- des accès sous plusieurs langages (en C avec la librairie libpq et le pré-processeur egpc, en C++ avec la librairie libpq++, en TCL avec la librairie lipgtcl, en PERL et en Python, ...), des passerelles avec les serveurs Web, ...
- la présence des « triggers » et règles permettant l'exécution de modules (SQL et C) avant ou après certains événements (INSERT, DELETE, UPDATE),
- une architecture ouverte permettant l'ajout de nouveaux types, opérateurs et index,
- des interfaces avec Kerberos et SSL,
- et un mécanisme d'héritage (d'où le qualificatif de « relationnel-objet »).

PostgreSQL connaît encore certaines limitations :

- comme indiqué ci-dessus, quelques lacunes dans l'implémentation de SQL 92,
- chaque objet est limité à 8192 octets (au delà de cette limite, il faut utiliser la fonctionnalité spécifique des « larges objects »),
- la taille d'une requête SQL est limitée à 8192 octets <sup>1</sup>.

---

<sup>1</sup> Des modifications simples du source permettent de repousser la limite à 32 ko.

La version de PostgreSQL étudiée dans le présent document est la **version 7.0**.

### **2.1.3 Comparaison succincte des deux produits**

En résumé, les caractéristiques dimensionnantes pour différencier les deux produits étudiés sont les suivantes :

- PostgreSQL est plus évolué fonctionnellement : il s'agit d'un SGBD Objet-Relationnel, il inclut des types graphiques et des fonctions mathématiques, des triggers, des vues et dispose de véritables services transactionnels. Ses caractéristiques fonctionnelles sont proches des SGBDR commerciaux tels que ORACLE. Son orientation objet, lui donne une grande ouverture fonctionnelle (ajout de nouveaux types d'objets par exemple).
- MySQL est plus léger (plus simple à mettre en œuvre) et plus performant du point de vue des temps de réponse et du stockage de données volumineuses. Mais MySQL pêche par l'absence de mécanismes transactionnels (COMMIT-ROLLBACK). Il est bien adapté à des utilisations où les performances priment sur l'intégrité de la base de données.

En cas de mise en œuvre de la partie serveur sous Windows NT, il est à noter que MySQL est soumis à un coût de licence (contrairement au serveur sous UNIX, aux clients et à PostgreSQL).

Vis-à-vis des services de sécurité, ces deux produits présentent des caractéristiques suffisamment proches pour permettre une analyse groupée.

Note : TcX fournit, sur le site Web de MySQL ([www.mysql.com/crash-me-choose.html](http://www.mysql.com/crash-me-choose.html)) une analyse comparative très détaillée des caractéristiques fonctionnelles des principaux SGBD (libres et commerciaux). Malheureusement cette analyse ne prend pas en compte les services qui impactent directement la sécurité, telle que la gestion des droits d'accès.

## 2.2 Critères de sécurité

La sécurisation des SGBD est analysée selon les thèmes suivants :

1. **Fonctions de sécurité** disponibles de manière native, notamment en matière de confidentialité et d'intégrité des informations.
2. Éléments contribuant à la « **robustesse** » du service, c'est à dire à une disponibilité élevée.
3. Mécanismes permettant d'apporter des **extensions fonctionnelles** (ajout ou modification de fonctions de sécurité).

Pour chacun de ces thèmes, les critères suivants sont pris en compte :

- Fonctions de sécurité :
  - Identification et authentification des utilisateurs
  - Contrôle d'accès aux données
  - Contrôle d'accès aux services d'administration
  - Protection des échanges réseau
  - Protection des données stockées
  - Intégrité du logiciel
  - Services d'audit
- Éléments de robustesse :
  - Tolérance aux erreurs
  - Tolérance à la congestion
  - Qualité du code
  - Disponibilité de correctifs aux anomalies répertoriées
- Extensions fonctionnelles :
  - Ajout dynamique de fonctions ou procédures
  - Modification du code source

Chacun des critères identifiés ci-dessus fait l'objet d'une description dans un des paragraphes du chapitre 3. La description porte sur les deux produits analysés en précisant les différences lorsque nécessaire.

## 3. Évaluation des produits

L'évaluation présentée dans ce chapitre est basée sur un examen de la documentation des deux produits (cf. références du § 1.2) et, ponctuellement, du code source. L'analyse est axée essentiellement sur le thème « fonctions de sécurité », présentée au paragraphe 3.1.

### 3.1 Fonctions natives de sécurité des produits

Pour chaque fonction analysée sont présentées :

- La nature du besoin de cette fonction.
- Les caractéristiques standard des deux produits étudiés.
- Les failles ou vulnérabilités identifiées.
- Les solutions possibles pour y parer.
- Un résumé de l'analyse.

#### 3.1.1 Identification / authentification des utilisateurs

##### 3.1.1.1 Besoin

Il s'agit d'identifier les utilisateurs de la base de données, de manière sûre et non ambiguë, pour mettre en œuvre les contrôles d'accès, en fonction des droits de chaque utilisateur, et de permettre l'imputabilité des opérations effectuées. Dans ce paragraphe, le terme utilisateur couvre les différents clients du SGBD, c'est à dire à la fois l'utilisateur final et les administrateurs ou exploitants de la base de données.

##### 3.1.1.2 Fonctions natives

Pour les deux produits étudiés, l'identification des clients du SGBD est distincte de l'identification du système d'exploitation (typiquement du login UNIX), mais rien n'interdit d'utiliser le même nom d'utilisateur pour l'OS et le SGBD.

L'authentification et les contrôles de droit prennent en compte à la fois l'utilisateur lui-même, mais également la machine qui l'héberge.

###### 3.1.1.2.1 Identification et authentification MySQL

Pour le serveur MySQL, un client est identifié par un nom de machine (`host_name` ; par défaut `localhost`) et un nom d'utilisateur MySQL (`user_name` ; jusqu'à 16 caractères). Deux utilisateurs portant le même nom mais rattachés à des machines différentes sont deux utilisateurs distincts. Au niveau du client UNIX MySQL, la saisie du nom d'utilisateur est optionnelle ; par défaut c'est le nom du compte UNIX qui est utilisé. La création des

utilisateurs est réalisé à l'aide de la commande GRANT ou par mise à jour directe d'une base de données système (base `mysql`).

L'application serveur MySQL dispose de quelques options de lancement qui rendent plus difficile, à un utilisateur externe, d'usurper l'identité d'une machine (host) déclarée :

- `secure` : contrôle que l'adresse IP retournée par un appel système correspond bien à l'adresse de la machine,
- `skip-networking` : interdit les connexion via TCP/IP et n'autorise que les connexions par des sockets UNIX.

Le client MySQL est authentifié par un mot de passe. Le mot de passe peut être saisi par l'utilisateur ou être stocké dans un fichier de configuration du client (`.my.cnf`). Le mot de passe peut être vide, ce qui correspond à un utilisateur se connectant sans mot de passe. Le mot de passe est stocké par le serveur et transmis par le client sous forme chiffrée. Il s'agit d'un chiffrement non réversible, qui utilise un algorithme différent de celui utilisé par UNIX. Il n'existe pas de moyen technique permettant d'imposer une politique de mise en œuvre des mots de passe. En particulier, le mot de passe peut être vide, ce qui correspond à un utilisateur se connectant sans mot de passe.

#### 3.1.1.2.2 Identification et authentification PostgreSQL

Les utilisateurs PostgreSQL sont identifiés par un nom (`username`) et par un identifiant système numérique (`uid`) qui n'ont pas besoin d'être connus du système d'exploitation. La gestion des utilisateurs se fait à l'aide de commandes de création (`CREATE USER`), de modification (`ALTER USER`) et de suppression (`DROP USER`).

Le serveur PostgreSQL met en œuvre un programme spécialisé dans l'authentification des utilisateurs : `postmaster`.

L'authentification d'un utilisateur peut être autorisée :

- en local, c'est à dire pour les utilisateur loggés sur la même machine que l'application `postmaster`,
- par le réseau à l'aide d'une connexion TCP/IP.

Par défaut, seules les connexions locales sont autorisées ; il est nécessaire de lancer le l'application `postmaster` avec un paramètre particulier (`-i`) pour pouvoir autoriser des connexions réseau. Les connexions réseau peuvent être sécurisées par SSL. Pour cela, il est nécessaire de recompiler PostgreSQL avec SSL et de lancer `postmaster` avec une option particulière(`-l`).

Pour les connexions locales, l'authentification est basée sur le nom d'utilisateur (`user-id`) et, pour les connexions TCP/IP, est basée sur l'adresse IP.

PostgreSQL autorise diverses méthodes d'authentification. Le choix de la méthode d'authentification est précisé indépendamment pour chaque base, d'une part pour les connexions locales et, d'autre part, pour des plages d'adresses IP (adresse et masque).

Pour les connexions locales, il existe des méthodes d'authentifications « tout ou rien » et, de manière classique, des méthodes basées sur un mot de passe :

- `trust` : les connexions sont autorisées sans condition.
- `reject` : les connexions sont refusées sans condition.
- `password` : mot de passe transmis en clair.
- `crypt` : mot de passe transmis en chiffré (méthode analogue à celle de MySQL).

Pour les connexions TCP/IP, PostgreSQL permet d'utiliser les mêmes méthodes que l'authentification locale, mais également les méthodes suivantes :

- `ident` : mettant en œuvre un serveur d'authentification TCP/IP, tel que spécifiée par la RFC 1413, pour le client PostgreSQL,
- `krb4` ou `krb5` : Kerberos (V4 ou V5), à condition d'acquérir le code de Kerberos et de recompiler PostgreSQL ; il à noter que Kerberos est soumis à une licence d'exportation contrôlée par le gouvernement US.

### 3.1.1.3 Analyse des lacunes et failles de sécurité

La méthode d'authentification utilisée est particulièrement sensible car elle conditionne la confiance que l'on peut apporter dans les services de contrôles d'accès aux données et aux services d'administration du SGBD. Or les méthodes d'authentification fournies, en standard, par MySQL et PostgreSQL sont des méthodes d'authentification faibles qui ne présentent pas les caractéristiques nécessaires pour parer aux attaques d'un utilisateur hostile cherchant à usurper l'identité (et les droits) d'un utilisateur autorisé.

Nous ne détaillerons pas ici les risques liés à une mauvaises gestion des bases de données et qui peuvent conduire à tout autoriser à tous les utilisateurs, sans contrôle. La documentation de référence des produits présente un ensemble de précautions élémentaires à respecter impérativement.

Les méthodes d'authentification par mot de passe permettent facilement de rejouer une séquence de connexion dès que l'écoute du réseau est possible. Les méthodes de chiffrement de mot de passe (méthode d'authentification de MySQL et méthode `crypt` de PostgreSQL) n'offrent pas une meilleure sécurité que la transmission du mot de passe en clair. En effet, le mot de passe chiffré, transmis par le client et vérifié par le serveur, est le même d'une session à l'autre.

De plus, aucune protection n'est apportée contre la « divination » d'un mot de passe par essais systématiques. Ceci est d'autant plus gênant qu'aucun mécanisme n'interdit à un utilisateur d'utiliser un mot de passe faible, voire de se connecter sans mot de passe.

Enfin, les produits ne proposent pas de service d'authentification mutuelle permettant non seulement au serveur d'authentifier le client mais, également, au client d'authentifier le serveur. Il est ainsi possible à un attaquant de mettre en place un serveur pirate qui pourra, par exemple, servir de cheval de Troie pour usurper l'identité des clients, ou bien fournir de fausses informations en réponse aux requêtes des clients.

### **3.1.1.4 Solutions d'amélioration**

Dans certains cas d'utilisation, il est possible d'interdire toute connexion via le réseau. Il est alors nécessaire de se connecter au système d'exploitation de la machine hébergeant le serveur de SGBD pour accéder à la base de données. Dans cette situation, la mise en place d'un mécanisme d'authentification fort au serveur (au niveau du système d'exploitation) ou de mesures organisationnelles pour contrôler l'accès physique à cette machine peuvent suffire.

En général, il est utile d'autoriser des accès via le réseau. Pour éviter les attaques liées à l'écoute (pour rejouer une séquence d'authentification), une solution possible consiste à protéger le réseau lui-même (cf. paragraphe 3.1.4). Ceci n'empêche pas toutefois les attaques par essais systématiques de dictionnaires de mot de passe. L'option `secure` de MySQL permet de rendre ces types d'attaque plus techniques, car elle nécessite d'usurper l'adresse IP de la machine du client autorisé.

Les protections périphériques basées sur la protection locale du serveur et la protection du réseau ne peuvent pas toujours être mises en œuvre avec un niveau de confiance et de granularité suffisants pour les besoins d'authentification des clients de la base de données. Par exemple, si la protection réseau est basée sur l'équipement hébergeant l'application cliente, elle ne suffit pas pour distinguer différentes personnes ou applications clientes se connectant depuis le même poste. Il est donc utile de mettre en œuvre des mécanismes d'authentification forte entre le client et le serveur du SGBD.

Dans le cas de PostgreSQL l'intégration d'un des mécanismes tiers prévus par le produit (Kerberos, SSL) permet de renforcer le mécanisme d'authentification. Dans tous les cas, la disponibilité des produits sous forme de sources permet, moyennant une intégration logicielle, le remplacement d'un mécanisme d'authentification natif par un autre mécanisme ou l'ajout d'une nouvelle méthode d'authentification mieux adaptée à un besoin donné. Cette potentialité permet de mettre en œuvre toute sorte de mécanisme d'authentification, y compris avec des interfaces à une ressource matérielle d'authentification (CAM ...), à des contrôles biométriques, à une infrastructure de gestion de clés publiques (IGC). Elle suppose d'adapter à la fois le client et le serveur.

### **3.1.1.5 Résumé**

L'authentification des utilisateurs est un service primordial vis-à-vis de la sécurité des SGBD et comme les produits étudiés ne disposent pas, en natif, de mécanisme d'authentification forte, des améliorations doivent être apportées.

Dans certains cas d'utilisation, cette authentification forte peut être réalisée de manière périphérique, à l'aide d'une sécurité au niveau du réseau (authentification IP) et au niveau local des équipements (OS sécurisé). Lorsque cette approche n'est pas suffisante, la disponibilité des produits sous forme de source permet d'envisager l'intégration d'un mécanisme tiers.

## 3.1.2 Contrôle d'accès aux données

### 3.1.2.1 Besoin

Il s'agit de n'autoriser l'accès aux données stockées qu'aux personnes autorisées. Ce contrôle doit permettre de distinguer différents modes d'accès, au moins lecture ou écriture, et une granularité variable, par exemple au niveau d'une base, d'une table ou d'une colonne.

De manière classique, on distingue :

- le contrôle d'accès discrétionnaire, pour lequel les droits (ou privilèges) sont alloués aux utilisateurs explicitement pour certaines opérations et données,
- le contrôle d'accès obligatoire (encore appelé « par mandat ») basé, d'une part, sur une étiquette de sécurité des données et, d'autre part, sur un niveau d'habilitation des utilisateurs.

Ces deux types de contrôles sont complémentaires :

- Le contrôle d'accès discrétionnaire permet établir un cloisonnement souple et informel entre différents besoins d'en connaître et différents droits.
- Le contrôle d'accès obligatoire est nécessaire pour gérer des informations classifiées (notamment au sens de la classification de défense) dans un système multi-niveaux, c'est à dire lorsque les utilisateurs n'ont pas nécessairement un niveau d'habilitation suffisant pour consulter les données les plus sensibles.

### 3.1.2.2 Fonctions natives

Les produits étudiés ne gèrent que le contrôle d'accès discrétionnaire. La gestion des droits est normalement réalisé à l'aide des requêtes SQL GRANT (ajout de privilèges) et REVOKE (suppression de privilège).

La commande GRANT, telle que spécifiée par SQL92, permet d'allouer à un utilisateur (ou à tous les utilisateurs) le droit d'effectuer une opération SQL donnée (ou toutes les opérations) sur une table, ou une colonne. Les opérations pour lesquelles des droits peuvent être alloués par GRANT regroupent :

- Des opérations de consultation de données : SELECT,
- Des opérations de mise à jour de données : INSERT, UPDATE, DELETE.

Initialement, seul le créateur de l'objet dispose du droit d'exécuter un GRANT pour allouer des droits sur cet objet à d'autres utilisateurs. La commande GRANT comporte une option (WITH GRANT OPTION) qui autorise l'utilisateur auquel un privilège est alloué à accorder, à son tour le même privilège (ou un sous-ensemble des droits) à d'autres utilisateurs.

Les produits étudiés implémentent GRANT avec quelques spécificités :

- MySQL autorise également l'allocation de privilège au niveau global (toutes les bases d'un serveur) et au niveau d'une base. GRANT est également utilisé pour créer les utilisateurs et pour allouer des droits d'administration (cf. § 3.1.3.2).
- PostgreSQL ne permet pas d'allouer directement des droits au niveau d'une colonne ; cette restriction peut être contournée en définissant des vues, la commande GRANT de PostgreSQL permettant d'allouer des droits sur les vues. WITH GRANT OPTION n'est pas supportée. L'affectation des droits peut être faite sur un groupe d'utilisateurs. GRANT est également utilisé pour allouer des droits sur les règles (RULES).

MySQL et PostgreSQL stockent les droits des utilisateurs dans des tables système. Ainsi, tout utilisateur autorisé à modifier le contenu de ces tables peut modifier les droits alloués aux utilisateurs sans utiliser GRANT ou REVOKE.

Une manière particulière d'accéder aux données consiste à exporter des données de la base vers un fichier ou d'importer le contenu d'un fichier dans une base.

MySQL permet ces importations ou exportations à l'aide d'utilitaires de conversion lancés sur le serveur, depuis le système d'exploitation (par exemple `importsqltext` et `exportsqltext` pour des conversions en format texte). Le seul contrôle d'accès, vis-à-vis des données de la base, mis en œuvre est celui de l'OS pour l'exécution de l'outil de conversion.

PostgreSQL permet d'importer et d'exporter des données à l'aide de la commande COPY. Lors du traitement de cette commande, les droits de l'utilisateur en lecture (SELECT) ou en mise à jour (UPDATE, INSERT) sont contrôlés par le SGBD.

### 3.1.2.3 Analyse des lacunes et failles de sécurité

Dans la pratique, toute la sécurité native des SGBD étudiés est basée sur les services de contrôle d'accès discrétionnaire. Pour ce type de service, le niveau fonctionnel est tout à fait satisfaisant. Vis-à-vis de la résistance de ce contrôle d'accès, il est à noter que :

- Le contrôle d'accès est basé sur les droits alloués aux utilisateurs ; en conséquence :
  - l'authentification des utilisateurs doit être suffisamment forte pour se protéger des usurpations d'identité (cf. § 3.1.1).
  - l'accès aux services d'administration de ces droits doit impérativement être réservé à des utilisateurs habilités et être contrôlé de manière sûre (cf. § 3.1.3) ; pour cela, en terme de mesures organisationnelles, WITH GRANT OPTION doit être utilisée avec parcimonie.
- Il est possible de contourner l'utilisation des fonctions administratives de gestion de droits (GRANT, REVOKE) par modification des tables de la base de données système `mysql`. L'accès en modification à cette base doit impérativement être réservé à des administrateurs habilités.

Il reste toutefois possible d'attaquer le mécanisme de contrôle d'accès soit en modifiant directement les droits stockés sur disque (sans utiliser le SGBD), soit en substituant au

logiciel serveur, un logiciel pirate qui permet de contourner les contrôles. Dans le cas de MySQL, il est également possible d'importer ou d'exporter des données, sans contrôle d'accès par le SGBD, en exécutant un utilitaire de conversion sur le serveur.

La principale lacune fonctionnelle concerne l'absence de contrôle d'accès obligatoire. Il est toutefois à noter que ce point est également vrai pour les SGBD commerciaux à de très rares exceptions près (version multi-niveaux d'ORACLE).

### **3.1.2.4 Solutions d'amélioration**

Vis-à-vis du besoin d'améliorer l'authentification des utilisateurs, se reporter au § 3.1.1.

Vis-à-vis de la menace d'attaque des droits d'accès par modification directe de données sur disque, se reporter à la protection des données stockées (§ 3.1.5).

Vis-à-vis de la menace de contournement du contrôle d'accès par modification du logiciel, se reporter au contrôle d'intégrité du logiciel (§ 3.1.6).

Le cas particulier d'exécution d'un utilitaire de conversion sur un serveur MySQL est à traiter en tant que contrôle d'accès à une fonction d'administration des données (cf. § 3.1.3). Dans la pratique, ceci relève du contrôle d'accès de l'OS du serveur.

La suite de ce paragraphe traite du besoin de contrôle d'accès obligatoire.

A défaut d'un produit présentant des mécanismes d'étiquetage de sécurité des données et de contrôle d'accès obligatoire, il est possible de répondre à des besoins de base de données multi-niveaux en intégrant un étiquetage de sécurité dans le modèle de données et en le traitant dans l'applicatif.

Ce type de besoin peut être traité par une architecture adaptée du schéma de données et des applications clientes adaptées. Voici, à titre d'exemple, deux orientations possibles pour ce type d'approche.

- Les données peuvent être distribuées dans différentes bases et/ou différentes tables de structure analogues et correspondant aux différents niveaux de classification utilisés. Les droits d'accès en lecture à ces bases ou tables sont alors attribués aux utilisateurs en fonction de leur niveau d'habilitation. Au niveau des applications clientes, il faut prévoir de développer des requêtes de création de données qui utilisent la table ou la base ad-hoc, en fonction du niveau de classification précisé dans la requête. Cette solution, qui multiplie le nombre de jointures entre tables, peut détériorer fortement les performances d'accès à la base.
- Cette deuxième orientation est applicable à PostgreSQL qui offre un mécanisme de vues (ou tables virtuelles)<sup>2</sup>. Les tables contenant des données classifiées peuvent disposer d'une colonne « étiquette de sécurité ». Lors de la création de données, ce champ doit

---

<sup>2</sup> MySQL ne fournit pas, dans sa version actuelle, de vues, mais cette fonctionnalité fait partie des extensions identifiées pour une future version du produit.

obligatoirement être renseigné. Différentes vues permettent de consulter les données restreintes à un niveau de classification maximal. Les utilisateurs ne disposant pas du niveau d'habilitation le plus élevé n'auront pas de droit d'accès direct à la table, mais seulement aux vues correspondant à leur habilitation.

De manière plus ambitieuse, il pourrait être envisagé de mettre à profit la disponibilité des produits sous forme de code source pour développer des extensions multi-niveaux dans le SGBD lui-même. En fait, la complexité d'une telle tâche rend cette approche peu réaliste.

### **3.1.2.5 Résumé**

La protection, en intégrité et en confidentialité, des données gérées par le SGBD repose sur les services de contrôles d'accès discrétionnaires offerts par les produits. Les mécanismes natifs offerts par les produits présentent une granularité satisfaisante. Pour avoir une confiance suffisante dans ce service, il est nécessaire de mettre en œuvre une authentification forte des utilisateurs et de protéger les données stockées sur le serveur ainsi que les échanges réseau, pour éviter le contournement du contrôle d'accès du SGBD.

Aucune fonction de contrôle d'accès obligatoire n'est prévue par les produits. Les besoins en la matière sont à traiter au niveau des applications utilisatrices.

## **3.1.3 Contrôle d'accès aux services d'administration**

### **3.1.3.1 Besoin**

Il s'agit de n'autoriser les opérations d'administration du SGBD qu'à des utilisateurs disposant de droits particuliers.

Fonctionnellement, différentes fonctions d'administration peuvent être distinguées. Nous retiendrons les trois catégories suivantes, sachant que cette répartition est nécessairement arbitraire et peut être adaptée, pour une mise en œuvre donnée, en fonction de choix organisationnels :

- L'administration des données qui gère la structure des bases de données (création et modification de bases, de tables, de vues, d'index) et les fonctions et triggers stockés dans le serveur.
- L'administration de la sécurité (ou administration des utilisateurs) qui gère les utilisateurs, leurs méthodes d'authentification, leurs droits et qui effectue les audits de sécurité sur la base de données.
- L'administration technique qui effectue les tâches à caractère technique sur le serveur du SGBD, telles que lancement ou arrêt du service, sauvegarde/restauration de bases, optimisation de la structure de la base (réorganisation des indexes par exemple), supervision, monitoring..

Comme cela a été indiqué (au § 3.1.2.3), le contrôle d'accès aux services de gestion des utilisateurs est particulièrement important pour assurer la sécurité du SGBD.

### 3.1.3.2 Fonctions natives

La présente analyse ne détaille pas l'utilisation des scripts ou utilitaires d'administration fournis avec les produits ni des produits d'administration tiers qui pourraient être mis en œuvre pour gérer et superviser les bases de données. En pratique, il s'agit d'applications clientes spécialisées qui s'appuient sur les principes basiques décrits ci-après.

MySQL et PostgreSQL comportent, tous deux, la notion de super-utilisateur (identifié par `root` pour MySQL et `postgres` pour PostgreSQL), créé lors de l'installation du serveur et qui dispose de tous les droits. Suite à l'installation du produit, le super-utilisateur est le seul utilisateur connu. C'est donc lui qui doit créer les premiers autres utilisateurs. Il est recommandé que le super-utilisateur du SGBD soit distinct du super-utilisateur du système d'exploitation (`root` UNIX).

#### Pour PostgreSQL :

- Les utilisateurs sont créés à l'aide de la commande `CREATE USER`. Cette commande permet de préciser si l'utilisateur a ou non le droit de créer des bases et s'il a ou non le droit de créer d'autres utilisateurs. Ces droits peuvent ensuite être modifiés par la commande `ALTER USER` et l'utilisateur peut être supprimé par `DROP USER`.
- Les bases sont créées à l'aide de la commande `CREATE DATABASE`. L'utilisateur qui exécute cette commande est le propriétaire de la base.
- Seul le propriétaire (le créateur) d'un objet peut accorder des droits d'accès à cet objet à d'autres utilisateurs (à l'aide de `GRANT`).
- Les commandes « techniques » applicable au serveur, sont contrôlées à l'aide des mécanismes du système d'exploitation.

#### Pour MySQL :

- Les commandes `GRANT` et `REVOKE` permettent, de gérer non seulement les droits d'accès aux données (cf § 3.1.2.2), mais également les droits sur les opérations administratives : gestion des données (`CREATE`, `ALTER`, `DROP`, `INDEX`) et gestion technique (`RELOAD`, `SHUTDOWN`, `PROCESS`<sup>3</sup>). L'utilisation de `GRANT` et `REVOKE` peut être contournée par modification de tables système (cf. § 3.1.2.2).
- Au départ, seul le super-utilisateur peut attribuer des droits à d'autres utilisateurs. La commande `GRANT` permet d'autoriser certains utilisateurs à attribuer, à leur tour, une partie de leurs privilèges à d'autres utilisateurs (via l'option « `WITH GRANT OPTION` », ou en donnant un privilège « `GRANT` »).
- Les utilisateurs sont gérés à l'aide de la table système `user`. Le contrôle d'accès aux opérations de gestion des utilisateurs (création, modification, suppression) correspond au contrôle d'accès aux données de cette table.

---

<sup>3</sup> Le droit `PROCESS` peut permettre à l'utilisateur qui en bénéficie de lire des informations confidentielles pour lesquelles il n'a pas le besoin d'en connaître : cf. § 3.1.5.3.

Il est à noter que le lancement de l'application serveur, qui est une application contrôlée par le système d'exploitation est une opération particulièrement sensible car :

- Pour MySQL, il est possible d'utiliser une option `--skip-grant-tables` et, dans ce cas, plus aucun contrôle d'accès n'est effectué.
- Pour PostgreSQL, il est possible de déterminer quel utilisateur sera super-utilisateur (en précisant son `uid`).

Par ailleurs, l'exécution malveillante de commandes d'administration technique peut porter atteinte à la disponibilité du service (arrêt du serveur, suppression de process actif).

### **3.1.3.3 Analyse des lacunes et failles de sécurité**

L'administration de la base de données repose sur l'utilisation de commandes gérées par le SGBD, mais également d'applications lancées à partir du système d'exploitation (en particulier, lancement de l'application serveur).

Pour les commandes incluses dans le SGBD, les deux produits étudiés présentent des mécanismes de contrôle d'accès discrétionnaires différents mais tous deux acceptables moyennant l'application de mesures organisationnelles :

- Toute la gestion des droits est initialisée et modifiable sans restriction par le super-utilisateur. Ce rôle doit être réservé à des personnes de confiance.
- Les droits peuvent être réattribués par les individus en disposant. Cette possibilité ne doit être autorisée que pour les cas d'emploi le justifiant, de manière à éviter une prolifération anarchique de droits non nécessaires.

Le contrôle d'accès aux droits d'administration est basé sur l'authentification des utilisateurs. Il n'y a aucune différence technique entre l'authentification d'un simple utilisateur ou d'un administrateur (ou du super-utilisateur). La résistance du contrôle d'accès aux services d'administration est donc directement conditionnée par la mise en œuvre d'un mécanisme d'authentification forte suffisamment robuste. Les menaces d'attaque du contrôle d'accès aux commandes d'administration gérées par le SGBD sont les mêmes que celles présentées au § 3.1.2.3 pour le contrôle d'accès discrétionnaire aux données (modification directe des droits sur disque ou logiciel pirate).

Les commandes de contrôle du SGBD passées au niveau du système d'exploitation peuvent être particulièrement sensibles, comme cela a été cité pour les commandes de lancement de l'application serveur (cf. fin du § 3.1.3.2).

### **3.1.3.4 Solutions d'amélioration**

Vis-à-vis des fonctions d'administration gérées par le SGBD, les améliorations sont identiques à celles liées au contrôle d'accès discrétionnaire aux données (cf. § 3.1.2.4).

Pour palier aux vulnérabilités liées à l'exécution des commandes passées depuis le système d'exploitation, une sécurité suffisante doit être apportée au niveau des services de contrôle d'accès du système d'exploitation lui-même : utilisation d'une version sécurisée de l'OS ou mise en œuvre d'un produit de sécurité locale. Il est également possible de mettre à profit la

disponibilité des produits sous forme de code source de manière à inhiber la prise en compte des options de lancement présentant des vulnérabilités importantes. Dans le cas où ces options sont utiles pour certains cas d'utilisation (aide au diagnostic d'anomalies, par exemple) et où il serait nécessaire de conserver les paramètres qui diminuent le niveau de sécurité, il serait utile de signaler à l'administrateur que le serveur fonctionne dans un mode dégradé. Cette signalisation pourrait être implémentée, par exemple, sous forme d'un message d'accueil particulier lors de la connexion, ou sous forme d'alarmes lors de l'exécution de requêtes.

### **3.1.3.5 Résumé**

Les fonctions d'administration du SGBD se répartissent entre des fonctions accessibles depuis le système d'exploitation (notamment le lancement du serveur du SGBD) et celles accessibles à l'aide de commandes du SGBD.

La première catégorie est particulièrement sensible et suppose une sécurité suffisante au niveau du système d'exploitation.

Pour la deuxième catégorie, la problématique est similaire au contrôle d'accès discrétionnaire aux données et rien ne permet de distinguer, en terme d'identification/authentification, un administrateur d'un simple utilisateur.

## **3.1.4 Protection des échanges réseau**

### **3.1.4.1 Besoins**

Les besoins de protection des échanges réseau concernent l'intégrité et, si nécessaire, la confidentialité des requêtes et des données échangées entre différents équipements mettant en œuvre ou utilisant le service de base de données. Il s'agit des échanges entre les postes clients (utilisateur final ou administrateur) et le serveur hébergeant le SGBD. Il peut s'agir également des échanges entre serveurs dans le cas d'un SGBD distribué (réplication de données).

Vis-à-vis de ces besoins, il est à noter que les produits étudiés n'offrent aucun service de réplication de données (MySQL cite seulement ce type de service en tant qu'évolution à réaliser).

### **3.1.4.2 Fonctions natives**

Aucun véritable service de protection en intégrité ou en confidentialité n'est offert de manière native par les produits étudiés.

### **3.1.4.3 Analyse des lacunes et failles de sécurité**

En cas de risque d'écoute réseau, les données échangées entre clients et serveurs sont en clair, ce qui n'est pas acceptable pour les données présentant une quelconque confidentialité.

En cas de risque d'altération malveillante des données lors de leur acheminement sur le réseau, la modification ne sera pas détectée (sauf si la syntaxe des requêtes n'est plus respectée).

### **3.1.4.4 Solutions d'amélioration**

Les solutions envisageables supposent la mise en œuvre de services cryptographiques tels que signature électronique et/ou chiffrement.

MySQL et PostgreSQL conseillent tous deux d'utiliser une solution de protection au niveau IP, telle que SSH, et d'établir un tunnel sûr entre le client et le serveur. Ce type d'approche présente l'avantage d'être transparent aux applications et de pouvoir couvrir tout type d'échange réseau (utilisant le protocole IP) et non pas seulement le service de base de données. Les solutions de protection au niveau IP sont largement disponibles sous des formes variées :

- Il peut s'agir d'un service mis en œuvre au niveau des équipements d'extrémités ou au niveau d'équipements réseau tels que routeurs ou boîtiers de chiffrement.
- Des services normalisés existent, notamment dans le cadre des travaux IPSEC.
- Des chiffreurs gouvernementaux sont en cours de développement (Echinops).

Pour certaines applications, il est également possible de protéger les informations au niveau du client, sans impact sur le SGBD.

Par exemple, l'application cliente peut protéger une information structurée à l'aide d'un service de sécurisation de données tel que S/MIME. Ces données protégées peuvent être stockées en base de données accompagnées d'informations non sensibles, stockées en clair, de manière à permettre d'exécuter des requêtes de recherches dans la base. Par exemple, pour une application de gestion électronique de documents classifiés, le contenu des documents doit être stocké chiffré, mais certaines données d'indexation peuvent rester en clair. Dans cette approche, tout client peut vérifier l'origine et l'intégrité des données protégées et seuls les clients autorisés peuvent déchiffrer les données protégées en confidentialité.

Ce type de solution n'est pas toujours applicable puisqu'il suppose le développement de services de protection au niveau des applications utilisatrices de la base de données. Son principal intérêt est qu'il apporte une protection de bout en bout entre le créateur des données et les utilisateurs de ces données, sur l'ensemble de la chaîne « réseau et stockage ».

Il est également envisageable de modifier le comportement des requêtes SQL, à la fois côté client et côté serveur, de manière à mettre en œuvre un SQL sécurisé (signature et, si nécessaire, chiffrement de la requête par l'émetteur ; contrôle et déchiffrement par le destinataire). Il s'agit du seul type de solution où l'utilisation d'un logiciel libre apporte un avantage (du fait de la possibilité de modifier le code source). Ce type de solution est toutefois plus coûteux à mettre en place qu'une solution de protection au niveau IP, sans apporter d'avantage notable s'il est restreint à la protection des données échangées sur le réseau. En revanche, une telle orientation pourrait présenter un intérêt si la solution est conçue pour protéger également les données stockées par le serveur (cf. § 3.1.5).

### **3.1.4.5 Résumé**

Les produits étudiés ne sont pas conçus pour protéger, en confidentialité ou en intégrité, les échanges sur le réseau. Toutefois, une protection efficace peut généralement être apportée de manière indépendante du SGBD, en mettant en œuvre une sécurité cryptologique au niveau du protocole IP.

## **3.1.5 Protection des données stockées**

### **3.1.5.1 Besoins**

Les besoins de protection des échanges réseau concernent l'intégrité et, si nécessaire, la confidentialité des données stockées, ou en cours de traitement, dans le serveur hébergeant le SGBD. Le but est de garantir que :

- toute modification ne peut être effectuée que par un utilisateur autorisé,
- les données sensibles ne peuvent être lues que par les utilisateurs autorisés (c'est à dire disposant d'une habilitation suffisante et du besoin d'en connaître).

### **3.1.5.2 Fonctions natives**

MySQL et PostgreSQL ne proposent rien (ni en intégrité, ni en confidentialité) en dehors du contrôle d'accès des utilisateurs (droits en lecture et écriture).

De plus, MySQL dispose d'une fonction `mysqld` qui permet à tout utilisateur ayant un privilège particulier (droit `PROCESS`) de lire le contenu des commandes en cours d'exécution sur le serveur et donc peut permettre l'accès à des informations confidentielles.

### **3.1.5.3 Analyse des lacunes et failles de sécurité**

En supposant que le contrôle d'accès des utilisateurs à la base de données soit sûr (cf. § 3.1.2), il subsiste des risques liés aux accès directs aux données (c'est à dire accès au contenu du disque, en contournant le logiciel SGBD).

### 3.1.5.4 Solutions d'amélioration

Les services de contrôle d'accès du système d'exploitation peut limiter les risques d'accès malveillant aux données, en réservant l'accès aux partitions ou fichiers utilisés par le SGBD à un compte privilégié. Il peut s'agir des services natifs du système d'exploitation ou de la mise en œuvre de services de sécurité locale mettant en œuvre des mécanismes cryptographiques (par exemple utilisation d'un service de chiffrement à la volée des accès disques, tel que celui mis en œuvre, sous Windows NT, par le produit gouvernemental ISATIS développé dans le cadre de l'Opération MUSE).

Vis-à-vis de solution de chiffrement des données intégrées à l'O/S, il est à noter que seules les solutions indépendantes du système de gestion de fichier (par exemple driver disque chiffrant) sont applicables à coup sûr. En effet, les produits SGBD utilisent souvent des accès directs à un espace disque, indépendamment du gestionnaire de fichiers de l'O/S. Par exemple, MySQL, depuis la version 3.23, fournit son propre gestionnaire ISAM.

Ce type de solution ne permet pas de différencier les droits accordés aux différents utilisateurs du SGBD. Il doit être mis en œuvre de manière à ce que seule l'application serveur SGBD ait le droit d'accéder directement aux données sur disque et que tout autre utilisateur doive impérativement faire appel à cette application pour accéder aux données.

Vis-à-vis des risques d'intrusion dans le serveur, depuis le réseau, en contournant les services du SGBD, il est possible de placer, en coupure réseau, un équipement « filtre SQL » (pare-feu spécialisé), dont le but est de rejeter toute trame IP ne correspondant pas à une requête SQL. Un tel équipement peut de plus effectuer un contrôle sur l'adresse IP de manière à n'accepter que les requêtes provenant d'adresses connues. Pour certains besoins gouvernementaux de protection d'informations classifiées, ce type de solution peut être bâtie sur la base d'un équipement FOX (pour lequel un filtre SQL pour ORACLE doit être disponible). Pour des besoins moins contraignants, il peut s'agir d'un filtre développé sur un firewall classique.

La mise en œuvre de services spécialisés pour le SGBD constitue une autre orientation envisageable.

Certaines solutions identifiées dans le cadre de la protection des échanges réseau (cf. § 3.1.4.4) apportent également la protection des données stockées :

- Protection applicative, au niveau du client.
- SQL sécurisé, à la fois côté client et côté serveur, si la mise en œuvre côté serveur permet de protéger les données stockées (cf. ci-après).

Si la protection des échanges réseau est assurée par une solution périphérique, il est également possible de modifier le comportement des requêtes SQL uniquement du côté serveur, de manière à mettre en œuvre un SQL sécurisé destiné à protéger les données stockées. Ce type de solution suppose :

- signature et, si nécessaire, chiffrement des données lors des opérations de mise à jour ;

- contrôle et déchiffrement lors des opérations de recherche ou consultation.

Il est à noter que ce type de solution est nécessairement très pénalisant en terme de performance, du fait des opérations cryptologiques nécessaires pour tout accès du serveur aux données de la base.

### **3.1.5.5 Résumé**

Les produits étudiés ne sont pas conçus pour protéger, en confidentialité ou en intégrité, les données stockées sur le serveur vis-à-vis d'accès qui contourneraient les mécanismes de contrôle du SGBD.

Il est possible d'apporter une protection locale au niveau du serveur en associant sécurisation du système d'exploitation et protection en coupure sur le réseau (firewall avec filtre SQL). Dans certains cas d'emploi, c'est l'application utilisatrice qui peut être sécurisée, par mise en œuvre de ressources cryptologiques au moins côté client.

## **3.1.6 Intégrité du logiciel**

### **3.1.6.1 Besoins**

Le logiciel du SGBD doit pouvoir faire l'objet de contrôles permettant de vérifier que l'application exécutée est bien celle qui a été validée par les autorités compétentes et non pas une version modifiée de manière non autorisée. Le besoin concerne à la fois le logiciel serveur et les logiciels clients, mais seul le cas du serveur est décrit dans le présent paragraphe.

### **3.1.6.2 Fonctions natives**

Les produits étudiés ne présentent aucune caractéristique permettant d'assurer que le logiciel exécuté est bien issu d'une chaîne de développement autorisée et n'a pas fait l'objet de modification malveillante.

En revanche, les produits introduisent des vulnérabilités particulières liées aux possibilités d'ajout dynamique de procédures ou de fonctions utilisateurs.

### **3.1.6.3 Analyse des lacunes et failles de sécurité**

Deux types de menaces doivent être traitées :

- La modification malveillante de l'application client ou serveur, liée par exemple à l'exécution d'un virus informatique ou à la substitution du binaire par une version pirate.
- L'ajout dynamique de fonctions qui peuvent s'exécuter à l'insu des utilisateurs lors du traitement de requêtes (cas des triggers ou des règles pour PostgreSQL et de procédures utilisateur pour MySQL). Pour la présentation de ces mécanismes d'extension dynamique du SGBD, se reporter au § 3.3.1.

### **3.1.6.4 Solutions d'amélioration**

Pour parer le premier type de menace (modification du logiciel), il est nécessaire de disposer d'un service de contrôle d'intégrité logiciel couplé au système d'exploitation. Au minimum, il peut s'agir de la mise en œuvre d'un logiciel anti-virus. Pour une sécurité renforcée, l'utilisation de produits évalués peut être requise. A titre d'exemple, le produit ISATIS développé pour les besoins du Ministère de la défense, permet d'apporter une solution de sécurité locale de niveau gouvernementale pour les équipements exploités sous Windows NT. Par ailleurs, dans une certaine mesure, une authentification mutuelle entre le client et le serveur rend plus difficile la modification malveillante du logiciel.

Pour se protéger du deuxième type de menaces les mécanismes de contrôles d'accès des produits permettent de limiter le droit de créer ou modifier des fonctions dynamiques à des utilisateurs de confiance. La problématique de cette approche est celle du contrôle d'accès aux fonctions d'administration (cf. § 3.1.2). Une approche complémentaire intéressante consiste à développer un mécanisme de contrôle d'intégrité, par le serveur, des fonctions ou procédures dynamiques.

Ceci suppose de mettre en œuvre :

- au niveau des chaînes de développement de ces procédures ou fonctions, une fonction de signature électronique de ces objets,
- au niveau du logiciel serveur, une fonction de contrôle de cette signature.

Un tel développement est faisable, du fait de la disponibilité du logiciel serveur sous forme de sources.

### **3.1.6.5 Résumé**

Le logiciel du SGBD doit être contrôlé en intégrité à l'aide de produits de sécurité locale (au minimum un anti-virus) sur les équipements clients et serveur.

La mise en œuvre d'extensions dynamiques, et notamment celles qui peuvent être exécutées à l'insu des utilisateurs (triggers et règles de PostgreSQL, procédures de MySQL) doivent faire l'objet au minimum d'une politique de contrôle d'accès adaptée. De manière plus sécurisante, des évolutions du produit sont souhaitables pour contrôler les extensions dynamiques sur la base d'une signature électronique.

## **3.1.7 Services d'audit**

### **3.1.7.1 Besoins**

Les services d'audit concernent la journalisation des événements relatifs aux accès au SGBD et la protection en intégrité de ces journaux.

Ce type de service est nécessaire, en complément des services de contrôle d'accès, pour permettre un contrôle a posteriori des opérations effectuées de manière à éviter que certains utilisateurs outrepassent leurs droits.

### **3.1.7.2 Fonctions natives**

MySQL permet d'enregistrer dans un fichier journal toutes les commandes SQL qui mettent à jour les données. Cette mise en œuvre est optionnelle et dépend d'un paramètre de lancement (option `--log-update=file_name`). Ce service est conçu pour mettre à jour la base de données à partir d'une sauvegarde, ou sur un serveur miroir, et n'est pas réellement un service d'audit. En particulier, l'origine de la requête n'est pas enregistré et il n'existe pas d'outil permettant à un administrateur d'exploiter le contenu de ce journal avec une ergonomie satisfaisante.

PostgreSQL exploite un fichier d'options (`pg_options`) pour imprimer une trace des opérations effectuées par le backend. Ce service est conçu comme un moyen de debug et ne permet pas de retrouver l'utilisateur à l'origine de l'opération tracée.

En dehors des services cités ci-dessus, aucun mécanisme documenté permet d'enregistrer une trace des opérations effectuées sur le SGBD.

### **3.1.7.3 Analyse des lacunes et failles de sécurité**

Aucun service d'audit de sécurité n'est prévu dans les produits étudiés.

### **3.1.7.4 Solutions d'amélioration**

Pour offrir un service d'audit, il est nécessaire de développer les fonctions de journalisation et les outils d'exploitation des journaux. Ce développement peut être réalisé par modification du code source. Dans le cas de PostgreSQL, l'utilisation de règles peut également permettre d'ajouter des fonctions de journalisation sous forme de modules dynamiques.

Le journal peut être une base de données, dans ce cas son exploitation peut être réalisée à l'aide de commandes SQL et le besoin d'intégrité peut être traité selon l'approche générale de protection des données présentée dans la présente étude.

### **3.1.7.5 Résumé**

Les produits étudiés ne disposent d'aucun service permettant un audit de sécurité. Une fonction de journalisation est à ajouter au serveur, en tant que développement complémentaire.

## **3.2 Éléments de robustesse des produits**

Les paragraphes qui suivent présentent quelques éléments relatifs à la confiance qui peut être apportée à la fiabilité et la robustesse des logiciels étudiés. Il s'agit, globalement, de critères contribuant à une disponibilité élevée du service de base de données.

### 3.2.1 Tolérance aux anomalies

Les produits étudiés effectuent, de manière classique, des contrôles de syntaxe et de droits sur les requêtes à traiter et retournent un message d'erreur si nécessaire. Le code des serveurs MySQL et PostgreSQL contient également des autotests qui permettent de valider la cohérence des bases.

Les mécanismes plus spécifiques permettant une certaine tolérance aux anomalies logicielles ou matérielles sont liés aux fonctions transactionnelles (ROLLBACK), dans le cas de PostgreSQL, et à des outils de contrôle de cohérence et de réparation logicielle des bases de données (ces outils permettent également d'optimiser la structure de la base de données). MySQL propose une application largement paramétrable `myisamchk` pour contrôler, optimiser et réparer les fichiers ISAM constituant les bases de données. PostgreSQL dispose de divers outils (`cluster`, `vacuum...`) destinés à réorganiser la structure des bases, mais ces outils sont conçus avant tout pour des besoins d'optimisation. Cette différence d'approche s'explique par les lacunes de MySQL en fonctions transactionnelles.

Par ailleurs, les deux produits étudiés disposent d'un utilitaire (`mysqldump` pour MySQL `pg_dumpall` pour PostgreSQL) qui permet d'effectuer une photographie du contenu des bases, à un instant donné de manière à permettre la sauvegarde des données sans arrêter le service.

### 3.2.2 Tolérance à la congestion

Dans la pratique, les deux produits ont été utilisés avec succès au sein d'applications opérationnelles mettant en œuvre un dimensionnement important en terme de nombre d'utilisateurs, de données et de requêtes. Cependant, aucun mécanisme particulier n'a été implémenté dans les produits étudiés pour identifier et prendre en compte des risques de congestion (dépassement d'un seuil de charge). Il y a donc des risques de pouvoir attenter à la disponibilité du service (déni de service) par des avalanches de requêtes. Mais ce type de problème est à considérer de manière globale car la congestion d'un serveur de SGBD (d'origine malveillante ou non) peut intervenir ailleurs qu'au niveau de l'application SGBD, par exemple au niveau de la pile IP.

Au niveau du SGBD lui-même, les risques de congestion du serveur sont liés à un nombre élevé de requêtes simultanées qui pourrait dépasser les capacités logicielles des files d'attente du produit, ou les capacités matérielles (espace mémoire, espace disque). Dans l'hypothèse où le nombre de clients potentiels est borné, ce risque peut provenir d'un ou plusieurs clients (éventuellement malveillants) qui exécuteraient un taux de requêtes très sensiblement supérieur à une utilisation nominale. Le comportement réel des produits dans ces cas de débordement de ressources du serveur n'a pas été testé, mais on peut s'attendre soit, dans le meilleur des cas, à des rejets des requêtes émis par les différents clients, soit, en cas de bug, à un arrêt du logiciel.

Pour éviter que certains clients puissent pénaliser les autres, en terme de disponibilité de l'application serveur, il serait nécessaire d'implémenter une gestion de quota (ressources serveur allouées, au maximum, à chaque client) qui n'existe pas dans les produits étudiés.

### 3.2.3 Qualité du code source

Une analyse globale des fichiers source montre des caractéristiques tout à fait satisfaisantes du point de vue de leur qualité.

Serveur MySQL (version 3.22) :

- Langages C et C++
- Nombre de fichiers header (et shell) : 404
- Nombre de fichiers source : 579
- Nombre de lignes du source (avec commentaires) : 137 164 (C et C++) + 30 886 (header)
- Plus gros fichier (C++) : environ 2 000 lignes
- Commentaires : abondants (environ 17 000 lignes), bien structurés (toutes les fonctions sont commentées en en-tête).
- Autres caractéristiques : une partie client, une partie threads-serveur, du code pour autotest (sommaire).
- Avis général : fonctions courtes et lisibles.

Serveur PostgreSQL (version 6.5) :

- Langages C et C++
- Nombre de fichiers header (et shell) : 411
- Nombre de fichiers source : 494
- Nombre de lignes du source (avec commentaires) : 288 543 (C et C++) + 44 056 (header)
- Plus gros fichiers (C++) : environ 17 000 lignes, puis 12 000, puis 4 000.
- Commentaires : abondants.
- Autres caractéristiques : portabilité développée (templates pour les divers OS supportés) ; autotests très abondants (régression, performance), interface jdbc et odbc.
- Avis général : procédures souvent longues mais bien commentées.

### 3.2.4 Problèmes connus et corrections

Un examen rapide des historiques des versions successives des deux produits étudiés et des mailing lists qui leur sont consacrés sur Internet montre que :

- Les deux produits étudiés sont attachés d'anomalies, comme tout logiciel d'une certaine complexité et qui continue d'évoluer, mais il s'agit de produits matures dont les versions éditées en tant que version stabilisée (par opposition aux versions alpha ou beta) ne souffrent que de peu de problèmes gênants.
- Les anomalies identifiées sont rapidement documentées et font généralement l'objet de corrections sous forme de patches et/ou de mises à jour du produit.

Des efforts appréciables ont été apportés pour associer aux produits eux-mêmes des jeux de tests permettant, pour PostgreSQL, de vérifier la non régression des nouvelles versions et, pour MySQL, de tester les performances et la conformité à SQL 92. Sur ce point, se reporter au § 3.3.2.

En résumé, les produits étudiés présentent une bonne stabilité et une organisation efficace pour faire face à la correction des problèmes résiduels.

## **3.3 Extensions fonctionnelles des produits**

### **3.3.1 Extensions dynamiques**

Les deux produits permettent d'ajouter, dans le serveur, des fonctions et procédures utilisateur. Ces mécanismes peuvent permettre d'ajouter des services de sécurité sous réserve que leur mise en œuvre n'introduise pas de vulnérabilité vis-à-vis de l'intégrité du logiciel (voir § 3.1.6).

MySQL permet de créer de nouvelles fonctions développées en C, sans avoir à régénérer le binaire de l'application binaire. La prise en compte dynamique de ces extensions se fait à l'aide de la commande CREATE FUNCTION.

MySQL permet également de développer des procédures en C++. Une procédure, dans la terminologie MySQL, modifie les données contenues dans la réponse à une requête, avant envoi au client.

PostgreSQL permet de créer de nouvelles fonctions développées en SQL, en langage procédural (PL/PGSQL, PL/TCL, PL/PERL) ou en langage compilé (typiquement C ou C++). La prise en compte dynamique (c'est à dire sans régénérer le binaire de l'application serveur) de ces extensions se fait à l'aide de la commande CREATE FUNCTION. Un mécanisme similaire permet l'ajout de nouveaux opérateurs, types et agrégats. Les mécanismes de règles (rules) et de triggers permettent de lancer dynamiquement des fonctions lors de l'accès à une table. Le trigger appelle des fonctions annexes chaque fois qu'une ligne est modifiée ; cela permet de contrôler ou de modifier certaines valeurs avant d'exécuter la requête. La création d'un trigger se fait à l'aide de la commande INSERT TRIGGER. Les règles sont un mécanisme spécifique de PostgreSQL qui permettent de modifier le comportement de la requête. La création d'une règle se fait à l'aide de la commande CREATE RULE.

De manière générale, les capacités d'extension dynamiques posent des problèmes d'intégrité du logiciel (cf. § 3.1.6.3). En particulier, dans le cas de modification du comportement d'une requête SQL standard (procédures de MySQL, triggers et règles de PostgreSQL). Dans ce cas, les mécanismes mis en place (par un utilisateur autorisé à créer ou modifier ces

extensions fonctionnelles) sont exécutées pour les requêtes émises par les autres utilisateurs et ce, éventuellement, à leur insu.

### 3.3.2 Modification du code source

Pour ce qui concerne l'amélioration des fonctions ou mécanismes de sécurité, la principale différence des SGBD libres, par rapport aux produits commerciaux, est la disponibilité des produits sous forme de code source, avec l'autorisation de procéder à toute modification souhaitée de ce source.

Les deux produits étudiés ont été développés en C et C++ (uniquement en C pour le serveur MySQL). Les essais de recompilation des distributions sous forme de source ont montré qu'il n'y avait aucune difficulté particulière pour régénérer le binaire. Le source est d'une bonne qualité (cf. § 3.2.3) gage d'une bonne maintenabilité. Il ne contient malheureusement pas d'API de sécurité.

De manière classique, la principale difficulté liée à la modification du source consiste à s'assurer qu'elle n'introduit pas de régressions fonctionnelles dans le produit. Pour faciliter ce contrôle, PostgreSQL est livré avec un jeu de tests de non régression du produit (*Regression Test Package*). MySQL ne fournit pas les mêmes moyens, mais fournit tout de même deux jeux de tests qui peuvent servir de base à des essais de non régression : un benchmark destiné à mesurer les performance et un outil *crash-me* de comparaison des caractéristiques fonctionnelles SQL par rapport à la norme SQL 92. Il est à noter que ces deux outils sont indépendants du SGBD et peuvent donc être utilisés pour valider d'autres produits que MySQL.

## 3.4 Résultats de l'évaluation

### 3.4.1 Tableau d'évaluation des produits

Le tableau d'évaluation présente une note pour chaque critère ou, lorsque pertinent, deux notes :

- une pour les fonctionnalités offertes,
- une autre pour les mécanismes implémentés.

Chaque note est représentée sous forme graphique, comme suit :

☛ : Fonction inexistante ou mécanisme inutilisable.

☹ : Caractéristiques insuffisantes : par exemple fonction présentant des lacunes gênantes ou mécanisme de sécurité facile à contourner.

☺ : Caractéristiques exploitables (même si des améliorations peuvent être préconisées).

Thème	Critère	Note fonction/ mécanisme	Commentaire
<b>Fonctions de sécurité</b>	Identification et authentification des utilisateurs	☺/☹	Pas d'authentification forte, ni d'authentification mutuelle.
	Contrôle d'accès aux données	☹/☺	Pas de contrôle d'accès obligatoire. La confiance dans le contrôle d'accès suppose une approche globale de la sécurité (prise en compte des autres critères).
	Contrôle d'accès aux services d'administration	☹/☺	Pas de distinction explicite entre administrateur et simple utilisateur. La confiance dans le contrôle d'accès suppose une approche globale de la sécurité (prise en compte des autres critères).
	Protection des échanges réseau	💣	Aucun service natif.
	Protection des données stockées	💣	Aucun service natif.
	Intégrité du logiciel	💣	Aucun service natif. Vulnérabilités particulières liées aux extensions dynamiques (triggers...).
	Services d'audit	💣	Aucun service natif.
<b>Éléments de robustesse</b>	Tolérance aux erreurs	☺	Outils de contrôle et réparation des bases de données. Mécanismes transactionnels de PostgreSQL.
	Tolérance à la congestion	☹	Pas de mécanisme particulier identifié pour éviter le déni de service. Mais bonne disponibilité constatée sur les applications opérationnelles.
	Qualité du code	☺	Code (C et C++) lisible et bien commenté.
	Disponibilité de correctifs aux anomalies répertoriées	☺	Bonne réactivité pour les corrections d'anomalies.
<b>Extensions fonctionnelles</b>	Ajout dynamique de fonctions ou procédures	☺/☹	Mécanismes à protéger en intégrité (cf. fonction « intégrité du logiciel »).
	Modification du code source	☺	Source disponible et licence autorisant les modifications.

### 3.4.2 Synthèse sur les lacunes et vulnérabilités

Les caractéristiques, en matière de SSI, des deux produits étudiés sont globalement semblables. On notera toutefois que l'ouverture de PostgreSQL à l'utilisation de SSL et Kerberos peut permettre d'atteindre un meilleur niveau de confiance dans les fonctions d'authentification. Cette ouverture correspond cependant à une facilité d'intégration d'un code tiers et non pas à des services disponibles de matière native dans les distributions binaires.

Toute la sécurité native des deux produits étudiés repose sur les services de contrôle d'accès discrétionnaire, or :

- la confiance que l'on peut apporter à ces services est directement liée à la résistance des mécanismes d'authentification des utilisateurs,
- ce contrôle peut être contourné par des attaques au niveau des échanges sur le réseau ou du stockage des données sur disque.

Par ailleurs, il est nécessaire de pouvoir disposer d'une confiance suffisante dans les administrateurs du système ce qui suppose d'une part de contrôler l'accès aux fonctions d'administration mais également de pouvoir auditer les accès au SGBD. Enfin, le logiciel du SGBD ne doit pas pouvoir être altéré de façon malveillante ; son intégrité doit être assurée.

Ceci montre la nécessité d'une approche complète prenant en compte les différents critères analysés.

Les lacunes gênantes concernent :

- la faiblesse des mécanismes natifs d'authentification,
- l'absence de protection des échanges réseau,
- l'absence de protection des données stockées (hors contrôle d'accès du SGBD),
- l'absence de service d'audit,
- l'absence de contrôle d'intégrité des logiciels, notamment vis-à-vis des possibilités de modification dynamique du comportement des requêtes.

Enfin, aucun service de contrôle d'accès obligatoire n'est prévu (pas de multi-niveaux).

### 3.4.3 Synthèse sur les solutions pour améliorer la sécurité

Un nombre important de vulnérabilités peut être évité à l'aide de protections externes indépendantes du SGBD lui-même :

- Protection cryptologique, en intégrité et confidentialité, des échanges réseaux entre client et serveur, au niveau IP.
- OS sécurisé et/ou application de sécurité locale offrant des services de contrôle d'accès des utilisateurs et de contrôle d'intégrité des logiciels.

- Filtre SQL (firewall) pour éviter le contournement du contrôle d'accès du SGBD à partir d'une connexion via le réseau.
- Dans certains cas d'emploi, protection au niveau des applications utilisatrices du SGBD.

Cependant des améliorations du logiciel SGBD sont souhaitables. Elles sont possibles, grâce à la disponibilité des sources :

- Intégration d'un service d'authentification forte, dans le serveur et les clients.
- Contrôle d'intégrité, par le serveur, des extensions dynamiques.
- Ajout d'un service d'audit (journalisation) sur le serveur.
- Avertissement de l'administrateur si le serveur a été lancé avec des paramètres qui diminuent la sécurité.

Les différentes lacunes et vulnérabilités techniques peuvent être hiérarchisées (de la plus grave à la moins gênante) en prenant en compte que :

- Une partie des menaces peut être couverte par des mesures organisationnelles, notamment celles liées aux besoins de sécurité locale (contrôle d'accès local aux équipements notamment).
- Certaines lacunes fonctionnelles peuvent être tolérables ou traitées au niveau des applications utilisatrices (absence de contrôle d'accès obligatoire par exemple).
- Certaines vulnérabilités sont plus facilement exploitables que d'autres par un attaquant. En particulier, l'absence de protection des échanges réseau n'est pas acceptable.

Ces considérations conduisent aux préconisations qui suivent, sachant qu'il s'agit de recommandations générales qui doivent être ajustées en fonctions des cas d'emploi.

### **3.4.4 Préconisations**

En première priorité, les améliorations recommandées pour assurer la sécurité d'un SGBD libre sont les suivantes :

- Mise en œuvre d'une protection cryptologique des échanges réseau entre les clients et le serveur, soit au niveau du service TCP/IP des équipements concernés, soit par des équipements en coupure (chiffreurs).
- Intégration logicielle, dans le SGBD (client et serveur), d'un service d'authentification forte.

Il est également nécessaire d'assurer un contrôle d'accès efficace au système d'exploitation de l'équipement hébergeant le serveur. Dans les cas où les mesures organisationnelles (contrôle d'accès physique) et les protections natives de l'OS se révéleraient insuffisantes, le recours à un OS sécurisé et/ou à des produits de sécurité locale (indépendants du SGBD) seront nécessaires pour assurer que seuls les administrateurs peuvent accéder à l'OS et pour contrôler l'intégrité du logiciel. Dans tous les cas, la mise en œuvre d'un anti-virus est préconisée. Lorsque la sécurité locale du serveur ne suffit pas pour éviter le risque d'un

accès malveillant depuis le réseau, en contournant les contrôles du SGBD, la protection doit être renforcée par la mise en œuvre d'un filtre SQL (firewall spécialisé).

Enfin des extensions fonctionnelles du logiciel serveur sont souhaitables pour contrôler l'intégrité des extensions dynamiques et pour disposer d'un service d'audit de sécurité.

## 4. Maquette

### 4.1 Objectifs de la maquette

#### 4.1.1 Orientation générale

La maquette réalisée au titre de la présente étude est destinée à illustrer l'intérêt des logiciels libres vis-à-vis de la mise en œuvre de services de SSI.

Les possibilités de sécurisation périphérique, c'est à dire externes au SGBD lui-même (par exemple sécurisation IP ou sécurisation de l'OS) ne sont pas caractéristiques des potentialités des logiciels libres et ne sont pas retenues pour la maquette.

L'objectif de la maquette est d'illustrer l'intérêt de la disponibilité des produits sous forme source pour remplacer un mécanisme de sécurité natif d'un des deux produits étudiés par un autre mécanisme. Cette illustration consiste à remplacer le mécanisme d'authentification de MySQL qui s'avère insuffisant en l'état (cf. § 3.1.1) par un mécanisme offrant une protection contre le rejeu et une d'authentification mutuelle.

#### 4.1.2 Mécanisme d'authentification mis en œuvre

Le système de mots passe actuellement utilisé par MySQL pose les deux problèmes suivants :

- Il est sujet au rejeu.
- Il ne permet pas d'authentification du serveur par le client.

Pour résoudre ces inconvénients, voici le protocole d'authentification mis en œuvre pour la maquette. Il nécessite une intervention (modification du code source) sur le client et une sur le serveur.

En introduisant deux aléas, à chaque requête de connexion :

Serveur		Client
	<-----	Sollicitation de connexion pour login client, Aléa 1
hash(password + Alea 1), Aléa 2	----->	Le serveur est OK
Le client est OK	<-----	hash(password + Alea 2)

#### Notes :

- On ne peut pas extraire le mot de passe des communications (apparaît dans des hashes).
- Le serveur sait générer le hash, car on suppose qu'il a les mots de passe des clients en clair (besoin de renforcer la sécurité locale).
- Le client compare le hash reçu avec celui qu'il calcule lui-même : il a le mot de passe et peut certifier que le serveur connaît bien son mot de passe. Or seul le serveur et lui-même connaissent ce mot de passe.
- Le serveur compare le hash reçu avec celui qu'il calcule lui-même, il sait si le client connaît le mot de passe.

#### Ce que le protocole résout :

- Le rejeu n'est plus permis : en effet, copier le hash n'a pas de sens : il fait réponse à l'aléa qui vient d'être émis par la partie opposée, et n'a que très peu de chances d'être semblable au hash intercepté (pas de collision mise en évidence avec MD5 à ce jour).
- Le serveur est authentifié de façon symétrique, en tant qu'unique co-détenteur du mot de passe client.
- Une attaque par « man in the middle » n'est pas possible car il faudrait, pour que l'attaquant puisse générer le hash, qu'il connaisse le mot de passe.

#### Ce que ce protocole ne résout pas :

- Le mot de passe est cessible, il n'est pas attaché à un support personnel. Il doit être protégé par ses détenteurs.
- Un mot de passe est sujet à attaques de force brute. Par contre, le mécanisme MySQL de protection par tables de droits, associant à la fois un mot de passe et une adresse IP (ou un nom d'hôte) à un login, réduit considérablement les chances de deviner la combinaison correcte.

## **4.2 Organisation de la maquette**

La maquette est constituée de deux PC raccordés par un réseau local : un serveur MySQL et un client MySQL (tous deux exploités sous Linux).

Sur chacun des deux PC, deux versions du produit (client ou serveur) MySQL sont installées : la version native (3.22.32) et la version modifiée (avec mécanisme d'authentification décrit ci-avant).

## **4.3 Scénario de démonstration**

Le scénario consiste à connecter/déconnecter plusieurs fois le client au serveur et d'observer le contenu des trames IP :

- En utilisant la version native, les attributs d'authentification sont constants (mot de passe chiffré) ; ce qui montre qu'il est facile, pour un attaquant qui écoute le réseau, de rejouer la séquence de connexion.
- En revanche, en utilisant la version modifiée, les attributs d'authentification (aléa et hash du mot de passe) changent à chaque fois.

## 4.4 Résultats

La réalisation de cette maquette a permis de confirmer l'intérêt de la disponibilité des sources pour améliorer la sécurité du SGBD.

L'implémentation du mécanisme d'authentification mis en œuvre est très perfectible (connexions simultanées, time-out ...), mais sa mise en œuvre n'a nécessité qu'une charge réduite de développement et intégration de logiciel (quelques jours pour une personne n'ayant pas de connaissance particulière du produit). En extrapolant, l'intégration dans un SGBD libre d'un mécanisme d'authentification forte quelconque, disponible sous forme de sources ou d'API, est possible, avec une charge d'intégration logicielle réduite.

L'exemple du mécanisme d'authentification peut être élargi aux différentes améliorations identifiées par l'étude (au § 3.1) pour lesquelles une modification du source est nécessaire.

## **5. Conclusions de l'étude**

L'étude a permis de mettre en évidence les améliorations nécessaires pour assurer la sécurité d'un SGBD libre. Une partie des préconisations (cf. § 3.4.4) peut être transposée à tout SGBD, car les solutions reposent sur des moyens externes au SGBD lui-même (par exemple protection des échanges réseau). L'intérêt des SGBD libres repose sur la disponibilité des sources et la possibilité de les modifier. Ceci permet d'envisager des solutions qui sont généralement inapplicables à un logiciel commercial.

Pour illustrer cette capacité, une maquette a permis de démontrer la faisabilité d'intégration d'un algorithme d'authentification au sein d'un SGBD libre, en remplacement du mécanisme natif.

En conclusion, les SGBD libres (et plus généralement les logiciels libres) présentent un avantage indiscutable en terme de sécurité, par rapport à leurs concurrents commerciaux, dès lors que l'intégration logicielle de fonctions SSI parfaitement maîtrisées est recherchée.