Fonctions de chiffrement dans MySQL

MySQL fournit plusieurs fonctions permettant de chiffrer ou de hacher des données, notamment `AES_ENCRYPT()`, `AES_DECRYPT()`, `DES_ENCRYPT()`, `DES_DECRYPT()`, `MD5()`, `SHA1()` et `SHA2()`. Chaque fonction possède des caractéristiques particulières et est adaptée à des cas d'usage spécifiques.

1. Chiffrement avec AES

AES_ENCRYPT()

La fonction `AES_ENCRYPT()` permet de chiffrer une chaîne à l'aide de l'algorithme AES (Advanced Encryption Standard).

Syntaxe :

```
SELECT AES_ENCRYPT('texte', 'clé');
```

Cela renvoie une version chiffrée de `'texte'` à partir de la clé `'clé'`.

AES_DECRYPT()

Permet de déchiffrer une donnée chiffrée avec AES.

Syntaxe :

```
SELECT AES_DECRYPT(texte_chiffré, 'clé');
```

2. Chiffrement avec DES

La fonction `DES_ENCRYPT()` fonctionne de manière similaire à `AES_ENCRYPT()`, mais utilise l'algorithme DES (Data Encryption Standard), aujourd'hui considéré comme moins sécurisé et donc moins recommandé.

3. Fonctions de hachage

Contrairement au chiffrement, le hachage est unidirectionnel : on ne peut pas retrouver la valeur d'origine.

MD5()

Renvoie un hachage de 32 caractères.

```
SELECT MD5('texte');
```

SHA1()

Renvoie un hachage de 40 caractères.

```
SELECT SHA1('texte');
```

SHA2()

Fonction de hachage plus sécurisée, permettant plusieurs tailles : 224, 256, 384 et 512 bits.

```
SELECT SHA2('texte', 256);
```

Il est recommandé d'utiliser SHA2 plutôt que MD5 ou SHA1, devenus obsolètes pour les usages sensibles tels que la sécurité.

4. À propos du hachage

Les fonctions de hachage étant irréversibles, elles sont idéales pour stocker des mots de passe. Même en cas de fuite de la base de données, il est très difficile de retrouver la valeur originale.

5. Sécurisation des connexions via SSL

MySQL permet d'activer le chiffrement SSL pour sécuriser les échanges entre le client et le serveur. Cela empêche l'interception ou la modification des données transmises.

6. Bonnes pratiques de sécurité

Le chiffrement n'est qu'un élément de la stratégie de sécurité d'une base de données. Il doit être complété par :

- des mises à jour régulières du SGBD,
- l'usage de pare-feu,
- le contrôle strict des accès,
- des sauvegardes sécurisées,
- des tests préalables avant mise en production.

Conclusion

Les fonctions de chiffrement et de hachage de MySQL constituent des outils puissants pour protéger les données. Il est important de bien les comprendre et de les utiliser dans un environnement contrôlé avant une intégration réelle.

💳 PARTIE 2 : TABLE DES CLIENTS (20 min)

Exercice 2.1 : Créer la table clients

Créez une table `clients` avec les colonnes suivantes :

- id (clé primaire auto-incrémentée)
- nom (texte, 100 caractères max)
- prenom (texte, 100 caractères max)
- email_crypte (pour stocker l'email crypté)
- telephone_crypte (pour stocker le téléphone crypté)
- date_creation (timestamp automatique)

⚠️ **ATTENTION :** Choisissez le bon type de données pour les colonnes cryptées !

Exercice 2.2 : Insérer des données cryptées

Insérez 3 clients avec leurs informations cryptées en utilisant AES_ENCRYPT :

- Client 1 : Alice Dupont, alice.dupont@email.com, 0601020304
- Client 2 : Bob Martin, bob.martin@email.com, 0612345678
- Client 3 : Charlie Durand, charlie.durand@email.com, 0698765432

Clé de cryptage à utiliser : `BanqueSecure2024!`

Exercice 2.3 : Lire les données déchiffrées

Écrivez une requête SELECT qui affiche :

- L'id
- Le nom complet (nom + prénom)
- L'email déchiffré
- Le téléphone déchiffré

🔐 PARTIE 3 : AUTHENTIFICATION SÉCURISÉE (25 min)

Exercice 3.1 : Table des comptes utilisateurs

Créez une table `comptes_utilisateurs` :

- id (clé primaire)
- client_id (clé étrangère vers clients)
- username (unique, 50 caractères)
- password_hash (64 caractères pour SHA2-256)
- salt (32 caractères)
- derniere_connexion (timestamp nullable)
- tentatives_echec (entier, défaut 0)
- compte_verrouille (booléen, défaut FALSE)

Exercice 3.2 : Créer une procédure d'inscription

Créez une procédure stockée `inscrire_utilisateur` qui :

1. Génère automatiquement un salt avec UUID()
2. Hache le mot de passe avec SHA2-256 en utilisant le salt
3. Insère le nouvel utilisateur

Paramètres :

- IN p_client_id
- IN p_username
- IN p_password

Exercice 3.3 : Tester l'inscription

Inscrivez 3 utilisateurs :

- alice_d / motdepasse123
- bob_m / securePass456
- charlie_d / myPassword789

Exercice 3.4 : DÉFI - Procédure de connexion

Créez une procédure `se_connecter` qui :

1. Vérifie si le compte est verrouillé
2. Vérifie le username et password
3. Si correct : met à jour derniere_connexion et remet tentatives_echec à 0
4. Si incorrect : incrémente tentatives_echec
5. Si tentatives_echec >= 3 : verrouille le compte

Paramètres :

- IN p_username
- IN p_password
- OUT p_resultat (VARCHAR : 'SUCCESS', 'WRONG_PASSWORD', 'LOCKED', 'USER_NOT_FOUND')

💰 PARTIE 4 : DONNÉES BANCAIRES SENSIBLES (25 min)

Exercice 4.1 : Table des cartes bancaires

Créez une table `cartes_bancaires` :

- id (clé primaire)
- client_id (clé étrangère)
- numero_carte_crypte (16 chiffres cryptés)
- cvv_crypte (3 chiffres cryptés)
- date_expiration_cryptee
- type_carte (ENUM: 'VISA', 'MASTERCARD', 'AMEX')
- actif (booléen)

Exercice 4.2 : Procédure d'ajout de carte

Créez une procédure `ajouter_carte_bancaire` qui crypte automatiquement :

- Le numéro de carte
- Le CVV
- La date d'expiration

Clé de cryptage : Utilisez une variable de session `@cle_bancaire`

Exercice 4.3 : Ajouter des cartes

Ajoutez 2 cartes pour chaque client (utilisez des numéros fictifs !)

Exercice 4.4 : Vue sécurisée

Créez une vue `cartes_masquees` qui affiche :

- Le nom du client

- Les 4 derniers chiffres de la carte (masquez le reste : **** **** **** 1234)
- Le type de carte
- Si elle est active

ASTUCE : Utilisez les fonctions RIGHT() et CONCAT() après déchiffrement

---

🔍 PARTIE 5 : TRANSACTIONS ET AUDIT (20 min)

Exercice 5.1 : Table d'audit

Créez une table `audit_acces_donnees` :

- id
- utilisateur_mysql (VARCHAR 100)
- table_accedee (VARCHAR 100)
- action (ENUM: 'SELECT', 'INSERT', 'UPDATE', 'DELETE')
- horodatage (timestamp)
- ip_address (VARCHAR 45)

Exercice 5.2 : Trigger d'audit

Créez un trigger `after_carte_select` qui enregistre chaque lecture de carte bancaire dans la table d'audit.

BONUS : Recherchez comment capturer l'utilisateur MySQL actuel (fonction USER())

Exercice 5.3 : Procédure de consultation sécurisée

Créez une procédure `consulter_carte_client` qui :

1. Vérifie que le client_id existe
2. Déchiffre et affiche les informations de carte

3. Enregistre l'accès dans l'audit

🚀 PARTIE 6 : DÉFIS AVANCÉS (Optionnel)

Défi 1 : Rotation des clés de cryptage

Créez une procédure qui re-crypte toutes les données avec une nouvelle clé.

Défi 2 : Fonction de validation IBAN

Créez une fonction qui valide et crypte un IBAN français.

Défi 3 : Système de double authentification

Ajoutez une colonne `secret_2fa_crypte` et une procédure pour générer/vérifier les codes 2FA.

Défi 4 : Historique des mots de passe

Empêchez la réutilisation des 5 derniers mots de passe.
