420-KBA-LG, programmation de bases de données

Saliha Yacoub

Programmation de bases de données:
Introduction à la sécurité des données

Plan de la séance:

- Introduction
- Menaces courantes
- Rôles serveur
- Rôles base de données
- Création d’une connexion
- Création d’un user
- Attribution de rôle BD

Sécurité des données

Introduction

- Aucune méthode universelle n'existe pour créer une application cliente SQL Server sécurisée.
- Chaque application est unique au niveau de sa configuration, de son environnement de déploiement et de ses utilisateurs.
- Une application relativement sécurisée lors de son déploiement initial peut devenir moins sécurisée avec le temps.
- Il est impossible d'anticiper avec précision sur les menaces qui peuvent survenir dans le futur.

Sécurité des données

Menaces courantes:

La sécurité peut être envisagée comme une chaîne dans laquelle un maillon manquant compromet la solidité de l'ensemble. La liste suivante comprend quelques menaces de sécurité courantes évoquées plus en détail dans les rubriques de cette section.

- **Injection SQL:**

L'injection SQL est le processus qui permet à un utilisateur malveillant d'entrer des instructions Transact-SQL au lieu d'une entrée valide. Si l'entrée est transmise directement au serveur sans validation et si l'application exécute accidentellement le code injecté, l'attaque risque d'endommager ou de détruire des données.

Solutions:

- Valider toutes les entrées
- Utiliser des procédures stockées ou des requêtes paramétrées

Sécurité des données

Menaces courantes:

Exemples :

SELECT * from utilisateurs where nom = @nom; (requête avec paramètre) →Bonne pratique

SELECT * from utilisateurs where nom =‘Patoche’ ; → Mauvaise pratique

Il suffit que quelqu’un de malintentionné remplace la requête par:

SELECT * from utilisateurs where nom =‘Patoche’ OR 1=1;

Dans l’exemple 2, Il suffit que quelqu’un de malintentionné va ajouter la requête UNION ALL

SELECT nom, Description FROM produits

WHERE Description like '%*Chaises'*

UNION ALL

SELECT username ,password FROM dba_users

Sécurité des données

Menaces courantes:

› **Élévation de privilège :**

Les attaques d'élévation de privilège se produisent lorsqu'un utilisateur s'empare des privilèges d'un compte approuvé, un administrateur ou un propriétaire.

› Solutions:
  - Évitez l'utilisation des comptes d'administrateur (comme Sa pour SQL Server, root pour MySQL et system pour Oracle) pour l'exécution du code.
  - Supprimer les comptes utilisateurs non utilisés
  - Supprimer les comptes utilisateurs par défaut
  - Donnez les privilèges selon les besoins.
  - Idéalement, interdire les accès distants aux comptes administrateurs par défaut (root, sa, sys)

Sécurité des données

Menaces courantes:

› **Détection des attaques et surveillance intelligente**

Une attaque de détection peut utiliser des messages d'erreur générés par une application pour rechercher des vulnérabilités dans la sécurité.

› Solutions:

- Ne pas afficher de messages d'erreur explicites affichant la requête ou une partie de la requête SQL. Personnalisez vos messages erreur.

Sécurité des données

Menaces courantes:

- **Mots de passe**

De nombreuses attaques réussissent lorsqu'un intrus a su deviner ou se procurer le mot de passe d'un utilisateur privilégié. Les mots de passe représentent la première ligne de défense contre les intrus, la définition de mots de passe forts est donc un élément essentiel de la sécurité de votre système.

- Solutions:
  - Renforcer les mots de passe.
  - Utilisez la stratégie des mots de passe pour les comptes sa, root et system.
  - Supprimer les comptes sans mot de passe.

Sécurité des données

Les privilèges, les rôles ou les droits peuvent être attribués:

- sur le serveur au complet
- sur une ou plusieurs bases de données
- sur une ou plusieurs tables, vues, procédures
- sur une colonne ou plusieurs colonnes.

La plus part des SGBD regroupent un ensemble de privilèges dans des ROLES prédéfinis . Certains rôles sont sur le serveur, d'autres sont sur la base de données.

Sécurité des données

Rôle du serveur:

SQL Server fournit des rôles au niveau du serveur pour vous aider à gérer les autorisations sur les serveurs

› SQL Server fournit neuf rôles serveur fixes. Les autorisations accordées aux rôles serveur fixes (à l'exception de **public**) ne peuvent pas être changées.

Les rôles du serveur sont attribués aux connexions

Sécurité des données, rôle niveau serveur

Rôles	Description
sysadmin	Les membres du rôle serveur fixe **sysadmin** peuvent effectuer n'importe quelle activité sur le serveur.
serveradmin	Les membres du rôle serveur fixe **serveradmin** peuvent modifier les options de configuration à l'échelle du serveur et arrêter le serveur.
securityadmin	Les membres du rôle serveur fixe **securityadmin** gèrent les connexions et leurs propriétés. Ils peuvent attribuer des autorisations GRANT, DENY et REVOKE au niveau du serveur. Ils peuvent également attribuer des autorisations GRANT, DENY et REVOKE au niveau de la base de données, s'ils ont accès à une base de données. En outre, ils peuvent réinitialiser les mots de passe pour les connexions SQL Server .
processadmin	Les membres du rôle serveur fixe **processadmin** peuvent mettre fin aux processus en cours d'exécution dans une instance de SQL Server.
setupadmin	Les membres du rôle serveur fixe **setupadmin** peuvent ajouter et supprimer des serveurs liés à l'aide d'instructions Transact-SQL. (L'appartenance au rôle**sysadmin** est nécessaire pour utiliser Management Studio.)
bulkadmin	Les membres du rôle serveur fixe **bulkadmin** peuvent exécuter l'instruction BULK INSERT.
diskadmin	Le rôle serveur fixe **diskadmin** permet de gérer les fichiers disque.
dbcreator	Les membres du rôle serveur fixe **dbcreator** peuvent créer, modifier, supprimer et restaurer n'importe quelle base de données.
public	Chaque connexion SQL Server appartient au rôle serveur **public**. Lorsqu'un principal de serveur ne s'est pas vu accorder ou refuser des autorisations spécifiques sur un objet sécurisable, l'utilisateur hérite des autorisations accordées à public sur cet objet. Vous ne devez affecter des autorisations publiques à un objet que lorsque vous souhaitez que ce dernier soit disponible pour tous les utilisateurs. Vous ne pouvez pas modifier l'appartenance au rôle public.

Sécurité des données

Rôle niveau base de données:

- Les rôles niveau base de données sont attribués à un utilisateur mappé sur la connexion

Sécurité des données , les Rôles BD

Rôles	Description
db_owner	Les membres du rôle de base de données fixe db_owner peuvent effectuer toutes les activités de configuration et de maintenance sur la base de données et peuvent également supprimer la base de données dans SQL Server. (Dans SQL Database et SQL Data Warehouse, certaines activités de maintenance requièrent des autorisations de niveau serveur et ne peuvent pas être effectuées par db_owners.)
db_securityadmin	Les membres du rôle de base de données fixe **db_securityadmin** peuvent modifier l'appartenance au rôle pour les rôles personnalisés uniquement et gérer les autorisations. Les membres de ce rôle peuvent potentiellement élever leurs privilèges et leurs actions doivent être supervisées.
db_accessadmin	Les membres du rôle de base de données fixe **db_accessadmin** peuvent ajouter ou supprimer l'accès à la base de données des connexions Windows, des groupes Windows et des connexions SQL Server.
db_backupoperator	Les membres du rôle de base de données fixe **db_backupoperator** peuvent sauvegarder la base de données.
db_ddladmin	Les membres du rôle de base de données fixe **db_ddladmin** peuvent exécuter n'importe quelle commande DDL (Data Definition Language) dans une base de données.
db_datawriter	Les membres du rôle de base de données fixe **db_datawriter** peuvent ajouter, supprimer et modifier des données dans toutes les tables utilisateur.
db_datareader	Les membres du rôle de base de données fixe **db_datareader**peuvent lire toutes les données de toutes les tables utilisateur.
db_denydatawriter	Les membres du rôle de base de données fixe **db_denydatawriter** ne peuvent ajouter, modifier ou supprimer aucune donnée des tables utilisateur d'une base de données.
db_denydatareader	Les membres du rôle de base de données fixe **db_denydatareader** ne peuvent lire aucune donnée des tables utilisateur d'une base de données.
Public	Permet de faire un USE sur la BD

Sécurité des données

Les privilèges sur les objets:

Les privilèges sur les objets sont attribués aux utilisateurs et non aux connexions.

En général un privilège ou une permission correspond à une commande SQL: SELECT, INSERT, DELETE, UPDATE, EXECUTE, CREATE…

Un objet, correspond à une table, une vue, une colonne, une procédure stockée ou une fonction.

Sécurité des données

Conclusion pour cette partie

- Éviter le code sql (utiliser des procédures stockées ou requêtes paramétrées) dans vos applications clientes pour prévenir les injection SQL
- Valider vos entrées des formulaires
- Utiliser une stratégie de mot de passes
- Restreindre les privilèges utilisateurs
- Éviter d'afficher les messages erreur du SGBD
- N'utilisez pas vos comptes admin pour vos opérations courantes.
- Créer des connexions avec des rôles suffisants. Pas plus
- Pour les user, donner les rôles BD suffisants . Pas plus

Sécurité des données

Attribution de ROLE

Lorsqu'une connexion est créée, elle a le rôle public sur le serveur et sur la BD qui lui est associée.

Un rôle PUBLIC sur le serveur, veut dire que l'on peut se connecter et uniquement se connecter au serveur

Un rôle public sur la BD, veut dire que l'on peut faire un USE et uniquement un USE sur la BD.

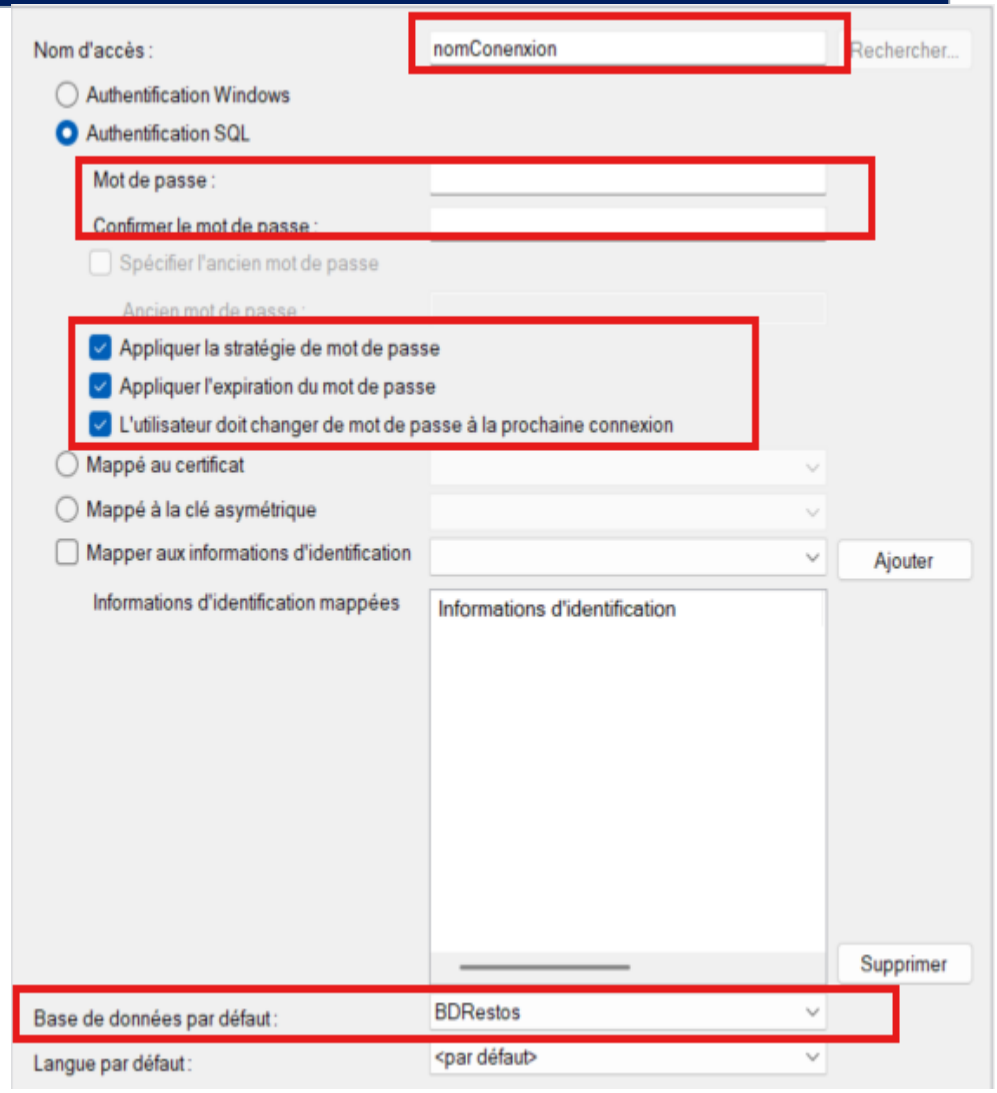
Pour attribuer des permissions ou des privilèges, il faudra mapper la connexion à un utilisateur (car les permissions sont données aux utilisateurs).

Pour garantir la sécurité des données il est conseillé de donner les rôles et les permissions au besoin.

Sécurité des données

Création d'une connexion: Pour connaître les paramètres d'une connexion, on peut se baser sur la figure ci-après. Ces paramètres sont:

- Le nom de la connexion
- Le mot de passe
  - Appliquer ou pas la stratégie du mot de passe.
  - Appliquer l'expiration du mot de passe
  - Doit changer le mot de passe à la prochaine connexion
- La BD par défaut



Sécurité des données

Pour créer une connexion , voici le code SQL

```
CREATE LOGIN nomConenxion WITH PASSWORD='420KBAlg!2026'
MUST_CHANGE, DEFAULT_DATABASE=EmpclgDB,
CHECK_EXPIRATION=ON, CHECK_POLICY=ON
```

Maintenant pour créer le user , il faut faire:

```
USE EmpclgDB
CREATE USER nomConenxion FOR LOGIN nomConenxion;
```

Sécurité des données

Pour créer la connexion et le USER mappé sur la connexion

```
USE EmpclgDB;

CREATE LOGIN nomConenxion WITH PASSWORD='420KBAlg!2026'
MUST_CHANGE, DEFAULT_DATABASE=EmpclgDB,
CHECK_EXPIRATION=ON, CHECK_POLICY=ON;

CREATE USER nomConenxion FOR LOGIN nomConenxion;
```

- La connexion et le user doivent avoir le même nom

Sécurité des données

- `DEFAULT_DATABASE` La BD par défaut. Si pas précisé ce sera MASTER
- `CHECK_POLICY=ON` Stratégie du mot de passe activée ou non. Par défaut cette stratégie est activée
- `CHECK_EXPIRATION=ON:` Le mot de passe va-t-il expiré ? Par défaut est à OFF
- `MUST_CHANGE` On peut aussi décider si l'on souhaite changer le mot de passe après la première connexion. Dans ce cas, l'option expiration du mot de passe doit-être activée.

Sécurité des données

Attribution de ROLE base de données

La connexion précédente `[nomConenxion]` a le rôle PUBLIC sur le serveur. → donc peut se connecter

Le user précédent `[nomConenxion]` a le rôle PUBLIC sur la base de données par défaut → donc peut faire USE.

Lorsqu'un ROLE **base de données** est attribué, nous dirons qu'un MEMBRE a été RAJOUTÉ au ROLE.

```
ALTER ROLE nomRoleBD ADD MEMBER nomUSER;
```

Sécurité des données

Exemples:

```
ALTER ROLE db_datareader ADD MEMBER nomConenxion;
ALTER ROLE db_datawriter ADD MEMBER nomConenxion;
ALTER ROLE db_ddladmin   ADD MEMBER nomConenxion;
```

Avec l'attribution des rôles précédents le user nomConenxion peut exécuter :

- Toutes les requêtes DDL sur toutes les tables de EmpclgDB
- Toutes les requêtes DML sur toutes les tables de EmpclgDB
- Toutes les requêtes SELECT sur toutes les tables de EmpclgDB

Cependant, sur le serveur la connexion n'a que le ROLE PUBLIC.

Sécurité des données

Retirer un ROLE

Pour retirer un ROLE base de données à un user on utilise la commande ALTER avec DROP

```
ALTER ROLE nomRoleBD DROP MEMBER nomUSER;
```

Exemple

```
ALTER ROLE db_ddladmin DROP MEMBER nomConenxion;
```

Sécurité des données

Attribution de ROLE SERVEUR

La connexion précédente `[nomConenxion]` a le rôle PUBLIC sur le serveur. → donc peut se connecter

Si nous souhaitons lui donner un rôle plus important sur le serveur ceci est possible avec la commande ALTER SERVER ROLE

```
ALTER SERVER ROLE nomRoleServeur ADD MEMBER nomDELaConenxion
```

```
ALTER SERVER ROLE dbcreator ADD MEMBER nomConenxion;
```

On peut également retirer un rôle serveur avec DROP

```
ALTER SERVER ROLE dbcreator DROP MEMBER nomConenxion;
```

Sécurité des données: les privilèges sur les objets:

Lorsque vous attribuez un ROLE sur la BD, les autorisations qui viennent avec le rôle s'appliquent à toutes la base de données et donc à toutes les tables, vues.

Pour donner des permissions sur des objets un seul à la fois , on utilise la commande **GRANT.**

Sécurité des données, GRANT

› La commande GRANT:

La commande GRANT permet d'attribuer des privilèges ou des permissions à un utilisateur sur un **objet**. Au lieu de donner des droits sur toutes les tables de la base de données par attribution de ROLE, on utilise la commande GRANT pour cibler les objets de la base de données qui seront affectés et restreindre les privilèges sur les objets ciblés pour les utilisateurs.

```
GRANT { ALL [ PRIVILEGES ] }
      | permission [ ( column [ ,...n ] ) ] [ ,...n ]
      [ ON [ class :: ] securable ] TO principal [ ,...n ]
      [ WITH GRANT OPTION ] [ AS principal ]
```

Sécurité des données, GRANT

- Si l'élément sécurisable est une fonction scalaire, ALL représente SELECT et REFERENCES.
- Si l'élément sécurisable est une fonction table, ALL représente DELETE, INSERT, REFERENCES, SELECT et UPDATE.
- Si l'élément sécurisable est une procédure stockée, ALL représente EXECUTE.
- Si l'élément sécurisable est une table, ALL représente DELETE, INSERT, REFERENCES, SELECT et UPDATE.
- Si l'élément sécurisable est une vue, ALL représente DELETE, INSERT, REFERENCES, SELECT et UPDATE.
- WITH GRANT OPTION, signifie que l'utilisateur qui a reçu le privilège peu donner le même privilège à un autre utilisateur sur le même objet.

Sécurité des données, GRANT

Exemples:

- `GRANT SELECT , INSERT  ON articles TO kbaLog;`
- `GRANT SELECT, INSERT, UPDATE(adresse) ON  Clients TO user1,user2;`
- `GRANT SELECT ON Commandes TO kbaLog WITH GRANT OPTION;`

Sécurité des données: REVOKE

- La commande REVOKE:
- La commande REVOKE permet de retirer des droits. (Des privilèges) . En principe les privilèges ont été attribuer par la commande GRANT

```
REVOKE [ GRANT OPTION FOR ] <permission> [ ,...n ] ON
    [ OBJECT :: ][ schema_name ]. object_name [ ( column [ ,...n ] ) ]
        { FROM | TO } <database_principal> [ ,...n ]
    [ CASCADE ]
    [ AS <database_principal> ]
```

CASCADE, lorsque vous avez : WITH GRANT OPTION, enlève les privilèges en cascade

```
REVOKE INSERT ON articles FROM kbaLog

REVOKE GRANT OPTION FOR  SELECT on Commandes
FROM  kbaLog CASCADE;
```

Sécurité des données, DENY

- La commande DENY:
- La commande DENY permet de **retirer des droits hérités d'un ROLE**.

```
DENY   { ALL [ PRIVILEGES ] }
     | <permission>  [ ( column [ ,...n ] ) ] [ ,...n ]
    [ ON [ <class> :: ] securable ]
    TO principal [ ,...n ]
    [ CASCADE] [ AS principal ]
[;]
```

Exemples

DENY SELECT on Clients to kbaLog;

Sécurité des données

Conclusion pour cette partie:

- Ne pas donner un rôle serveur autre que PUBLIC à n'importe quelle connexion.
- Ne pas donner un rôle BD, autre que PUBLIC à n'importe quel user.
- Il vaut mieux ajouter un privilège au besoin, plutôt que d'en retirer après un problème. Il faut cibler l'octroi des privilèges aux utilisateurs.
- La commande GRANT permet de mieux cibler les permissions sur les objets.
- La commande REVOKE permet de retirer des privilèges attribués par GRANT.
- La commande DENY permet d'enlever un privilège hérité d'un ROLE
- Ces trois commandes font partie du SQL standard. Disponibles dans tous les SGBDs.

Sécurité des données

- Rappels
- Les rôles non prédéfinis;
- Démonstration
- Les vues pour la sécurité des données.
- Chiffrement des données
  - Définition
  - Le hachage
  - L’ encryption.

Sécurité des données, Démonstration

- Création de login
- Création de user mappés sur le login
- Attribution de ROLE DB
- Tester la commande DENY
- Tester la commande GRANT
- Tester la commande REVOKE

Sécurité des données, créer un rôle

Les ROLES non prédéfinis. (ceux créés par les utilisateurs)

Rappel: Un rôle va regrouper plusieurs privilèges . L'avantage de créer des rôles est de pouvoir attribuer le même ensemble de privilèges à plusieurs utilisateurs .

Imaginer la situation suivante:

- Vous avez 20 programmeurs qui travaillent sur le même projet utilisant une base de données VenteProduits
- Vous avez donné les permissions nécessaires aux 20 employés sur les tables de cette base de données.
- Deux semaines plus tard, on se rend compte que nous avons oublié d'attribuer le droit UPDATE du prix sur la table Articles , et le droit INSERT sur la table Commissions .

Sécurité des données, créer un rôle

- **Les ROLES non prédéfinis**. (ceux créés par les utilisateurs)

La solution serait de retrouver les 20 programmeurs et de leur ajouter ces privilèges . Pas évident.

Imaginez maintenant que vous avez recruté deux autres programmeurs pour travailler sur le même projet.

Comment allez-vous faire pour retrouver toutes les permissions attribuées aux 20 premiers pour les donner aux deux derniers ? Chercher dans la documentation à condition que vous ayez tout garder.

Une solution simple serait de créer un ROLE pour lequel on donne les permissions ensuite, on donne le role à qui en a besoin. Facile.

Sécurité des données, créer un role

- **Les ROLES non prédéfinis: Étapes:**
  - On crée le ROLE→ CREATE ROLE nomrole
  - On attribue les permissions au ROLE→ GRANT
  - On ajoute un user (MEMBRE) au ROLE en utilisant la commande suivante ALTER ROLE

```
CREATE ROLE  nomRole;
GRANT priveleges TO nomRole;
ALTER ROLE monRole ADD MEMBER nomUser
```

Sécurité des données

› Exemple:

```
USE StocksClg;

CREATE  ROLE roleprojet;

GRANT SELECT , INSERT ON Articles TO roleprojet;

GRANT SELECT , INSERT, UPDATE(adresse) on Clients to roleprojet;

ALTER ROLE roleprojet add member Simba;
```

Ce rôle, je peux l'attribuer à toute une équipe de projet.

Sécurité des données

Maintenant, si nous avons oublié les privilèges UPDATE du prix sur la table Articles , et le privilège INSERT sur la table LigneCommande, alors on fait:

```
GRANT UPDATE(prix) ON Articles TO roleprojet;

GRANT INSERT ON lignecommande TO roleprojet;
```

On réhausse le ROLE roleprojet en ajoutant les privilèges nécessaires.

Evidemment, si on veut enlever des privilèges à toute l'équipe de projet, il suffit de faire un REVOKE sur le rôle.

```
REVOKE insert ON Clients FROM roleprojet
```

Sécurité des données

- Nous avons abordé les vues comme étant des objets de la base de données permettant la simplification de requêtes.
- Dans ce qui suit, nous allons voir comment les vues peuvent contribuer à la sécurité des données.

```
CREATE [ OR ALTER ] VIEW [ schema_name . ] view_name

AS select_statement   [ WITH CHECK OPTION ]   [ ; ]
```

- L'option WITH CHECK OPTION permet d'assurer que les modifications apportées à la table (dont la vue est issue) via la vue respectent la CLAUSE WHERE.
- Lorsqu'une ligne est modifiée via la vue, alors les données devraient rester cohérentes.

Sécurité des données

```
CREATE VIEW VSport as
SELECT  * FROM QuestionTrivia WHERE idcategorie =3
WITH CHECK OPTION;
GRANT SELECT, INSERT, UPDATE ON Vsport TO user1;
```

L'instruction suivante exécutée par le user1 va marcher car le code catégorie est 3:

```
INSERT INTO VSport VALUES('une question',1,'facile',3);
```

L'instruction suivante exécutée par le user1 NE va PAS marcher car le code catégorie est 2. Ne respecte pas la clause WHERE:

```
INSERT INTO VSport VALUES('une question',1,'facile',2);
```

De plus, le USER1, ne voit pas TOUT le contenu de la table `QuestionTrivia`

Le chiffrement des données: Le hachage

Définition: Le chiffrement est un procédé de la cryptographie qui consiste à rendre les données illisible ou impossible à lire sauf si vous avez une clé de déchiffrement.

- Techniques: **Le hachage** et le **chiffrement**.

› La technique de hachage des données à la particularité d'être **irréversible**; il n'est pas possible de retrouver les données originales après avoir été crypté avec une fonction de hachage.

› Il s'agit d'une fonction mathématique qui prend en entrée des données (chaine de caractères de longueur variable) et qui génère en sortie une chaine de caractères de longueur fixe appelée « hash »;

› La sortie (hash) est toujours la même pour des entrées identiques;

› Cette technique de chiffrement est couramment utilisée pour conserver des **mots de passe** ou vérifier l'intégrité d'un document;

› Algorithmes de hachage les plus utilisés: SHA2_256, SHA2_512

› appelé également chiffrement unidirectionnel.

› Dans **MS SQL Server** le chiffrement par hachage est fait avec la fonction **HASHBYTES**;

- La fonction prend deux paramètres: **HASHBYTES ('<algorithm>', { @input | 'input' })**
  - › L'algorithme à utiliser et les données à chiffrer;
  - › L'algorithme ne chiffre pas des chaines plus longues que 8000 caractères;

Le chiffrement des données: Le hachage

```
create table fournisseurs (
idfournisseur int identity,
nom varchar(40),
adresse varchar(50) not null,
mpasse varbinary(128) not null,
constraint pk_fournissuer primary key (idfournisseur));

--Les insertions…

insert into fournisseurs values
('le iga', 'laval etc', HASHBYTES('SHA2_512','local$33'));

insert into fournisseurs values
('le metro','rue de la liune ici',HASHBYTES('SHA2_512','passww$33'));

insert into fournisseurs
values ('Lafleur','pas loin',HASHBYTES('SHA2_512','Motdepasse123'));
```

Le chiffrement des données: Le hachage

```
select * from fournisseurs;
```

Résultats | Messages

	idfournisseur	nom	adresse	mpasse
1	1	le iga	laval etc	0x2E33B2AB38861BF326645AC58A85B5BDCA8AFDE897D7F5A...
2	2	le metro	rue de la liune ici	0x97780E3B6AED0C1764151E77B2D0ED539675F1CFF52FE64C...
3	3	Lafleur	pas loin	0xF8AE481DCC4A9D8AC4317BAC2FCBA7729B2CF0DABB5EF4...
4	4	un fournisseur	blaba	0x2E33B2AB38861BF326645AC58A85B5BDCA8AFDE897D7F5A...

Si je veux chercher un fournisseur selon son mot de passe:

```
select * from fournisseurs where mpasse =
        (
        select (HASHBYTES('SHA2_512','local$33'))
        ) and idfournisseur =1;
```

Exemple1: changer son mot de passe

```
create procedure changerMpasse(@id int,
                               @actuelMpasse varchar(30),
                               @nouveMpasse varchar(30)) AS
Begin
Declare @nouveauEncrypte varbinary(128), @currentMDP varbinary(128);
---on cherche le mot de passe actuel
    select @currentMDP =mpasse from fournisseurs where idfournisseur =@id;
    if(HASHBYTES('SHA2_512',@actuelMpasse) = @currentMDP)
        begin
        select @nouveauEncrypte= HASHBYTES('SHA2_512',@nouveMpasse );
        update fournisseurs set mpasse =@nouveauEncrypte where idfournisseur =@id;
        end;
   else print('les mots de passe ne correspondent pas');
end;
```

Le chiffrement des données: Le hachage

Pour executer:

```
execute changerMpasse
@id =1,
@actuelMpasse ='local$33',
@nouveMpasse ='Voila123';
```

Exemple2: Vérifier si un fournisseur existe

```
create function existFournisseur(@id int,
                                 @mpasse varchar(30)) returns int as
begin
declare @mpasseHach varbinary(128);
        select @mpasseHach = mpasse from fournisseurs
        where idfournisseur =@id;
                if(HASHBYTES('SHA2_512',@mpasse) = @mpasseHach)
                begin
                return 1;
                end;
 return 0;
end;
```

Pour tester:

```
select dbo.existFournisseur(2,'passww$33');
```

Le chiffrement des données: Le chiffrement (l'encryption)

Chiffrement symétrique: chiffrement rapide qui utilise une seule clé; la même clé est utilisée pour crypter et décrypter les données;

ENCRYPTBYPASSPHRASE, DECRYPTBYPASSPHRASE: encryption des données en utilisant l'algorithme Triple DES (Data Encryption Standard)

Ces fonctions utilisent une clé privée pour chiffrer les données. La clé est fournie sous la forme d'une chaine de caractères (un mot de passe par exemple).

EncryptByPassPhrase ({ 'passphrase' | @passphrase } , { 'cleartext' | @cleartext })

passphrase, phrase secrète à partir de laquelle générer une clé symétrique

@passphrase, Variable de type nvarchar, char, varchar, binary, varbinary ou nchar contenant

une phrase de passe à partir de laquelle générer une clé symétrique.

Cleartext : le texte à chiffrer

Le chiffrement des données: Le chiffrement (l'encryption)

DECRYPTBYPASSPHRASE fait l'opération inverse de **EncryptByPassPhrase**

DECRYPTBYPASSPHRASE ({ 'passphrase' | @passphrase } , { 'textecripte' | @textecripte })

Passphrase: phrase secrète utilisée pour générer la clé

Textecripte: Texte encripté.

Le chiffrement des données: Le chiffrement (l'encryption)

```
on ajoute la colonne pour la carte de crédit non cryptée

alter table fournisseurs add carteCredit varchar(20);

-- on met à jour normalement

update fournisseurs set carteCredit ='5555-5021-5789-4569'
 where idfournisseur=1;

update fournisseurs set carteCredit ='4567-8888-5789-9999'
where idfournisseur=2;

update fournisseurs set carteCredit ='8888-8888-5789-2222'
 where idfournisseur=3;

--On ajoute une colonne carte de crédit encrypteé (juste pour
vérifier)

alter table fournisseurs add carteCryptee varbinary(128);

update fournisseurs set carteCryptee
=ENCRYPTBYPASSPHRASE('local$33','5555-5021-5789-4569')  where
idfournisseur=1;
```

Le chiffrement des données: Le chiffrement (l'encryption)

Pour vérifier :

```
SELECT nom, adresse,carteCryptee AS 'carte encryptee',
CONVERT(varchar,DECRYPTBYPASSPHRASE('local$33',carteCryptee)) AS
'carte decryptée' FROM fournisseurs where idfournisseur =1;
```

nom	adresse	carte encryptee	carte decryptée
le iga	laval etc	0x020000002AA140B35692E931AAF74E9F4C925BD722590C1...	5555-5021-5789-4569

Le chiffrement des données: Le chiffrement (l'encryption)

Exemple 2:on fait les choses proprement avec une procédure stockée

```
alter table fournisseurs drop column carteCredit;

alter table fournisseurs  drop column carteCryptee ;

alter table fournisseurs add carteCryptee varbinary(128);
```

Contenue de la table fournisseurs:

idfournisseur	nom	adresse	mpasse	carteCryptee
1	le iga	laval etc	0x2E33B2AB38861BF326645AC58A85B5BDCA8AFDE897D7F5A...	NULL
2	le metro	rue de la liune ici	0x97780E3B6AED0C1764151E77B2D0ED539675F1CFF52FE64C...	NULL
3	Lafleur	pas loin	0xF8AE481DCC4A9D8AC4317BAC2FCBA7729B2CF0DABB5EF4...	NULL
4	un fournisseur	blaba	0x2E33B2AB38861BF326645AC58A85B5BDCA8AFDE897D7F5A...	NULL

La procédure stockée suivante va permettre de mettre à jour le numéro de carte de crédit uniquement si le mot de passe du fournisseur est valide.

Exemple 1: ajouter info crédit

```
CREATE procedure ajouterCarteCredit (@id int,
                                     @motdepasse varchar(30),
                                     @carteNonCrypte varchar(20)) as
begin
declare @motdepasseHash varbinary(128);
select @motdepasseHash = mpasse from fournisseurs where idfournisseur =@id;
if(HASHBYTES('SHA2_512',@motdepasse ) = @motdepasseHash)
        begin
        update fournisseurs set carteCryptee  =
        ENCRYPTBYPASSPHRASE(@motdepasse, @carteNonCrypte)
        where idfournisseur =@id;
        end;
end;
```

Exemple 1: ajouter info crédit

```
execute ajouterCarteCredit
@id =1,
@motdepasse ='Voila123',
@carteNonCrypte ='5555-5021-5789-4569'

execute ajouterCarteCredit
@id =2,
@motdepasse ='passww$33',
@carteNonCrypte ='5555-1111-2222-4569';
--Ne marcher pas.
execute ajouterCarteCredit
@id =3,
@motdepasse ='123466',
@carteNonCrypte ='5555-3333-2222-4569'
```

idfournisseur	nom	adresse	mpasse	carteCryptee
1	le iga	laval etc	0x6E3976A1729F1E6EC0685ED08DE6FC973E2DAE6CAD61C16...	0x0200000038BFC8FFDBADFB09AEA8EEC7F8956FDEE24A18...
2	le metro	rue de la liune ici	0x97780E3B6AED0C1764151E77B2D0ED539675F1CFF52FE64...	0x02000000F3228DC4CB33F117CB91DF92A0E6653C5EDBA45...
3	Lafleur	pas loin	0xF8AE481DCC4A9D8AC4317BAC2FCBA7729B2CF0DABB5EF48...	NULL

Exemple 2: obtenir info crédit

```
create function obtenirInfoCreditFournisseur
(@idfournisseur varchar(50),
 @mdp varchar(30)) returns table as
return
select nom,
convert(varchar,DECRYPTBYPASSPHRASE(@mdp,carteCryptee)) as nocredit
      from fournisseurs where idfournisseur = @idfournisseur;


select * from obtenirInfoCreditFournisseur(1,'Voila123');
```

Le chiffrement des données: Le chiffrement (l'encryption)

Cette fonction utilise une clé privée pour chiffrer les données. Cette clé doit être préalablement créée avec la commande 'CREATE SYMMETRIC KEY'. C'est au moment de créer la clé symétrique que l'on peut spécifier l'algorithme de chiffrement.

Pour le chiffrement par clé symétrique, les algorithmes les plus utilisés sont:AES_128, AES_192 et AES_256;

- Supporté par MS SQL Server;
- Considéré le plus sécuritaire aujourd'hui;
- Choisi par le gouvernement américain pour remplacer l'algorithme de chiffrement DES;

- Les étapes:
  - On crée la clé symétrique protégée par un mot de passe
  - On ouvre la clé pour encrypter
  - On utilise la clé pour encrypter
  - On ferme la clé.

Le chiffrement des données: Le chiffrement : Exemple (encrypter le NAS)

```
ALTER TABLE fournisseurs ADD nassCrypteByKey varbinary(128);
CREATE SYMMETRIC KEY SymmetricKeyNass
WITH ALGORITHM = AES_128
ENCRYPTION BY password ='Voila123';
--Ouvrir la clé
OPEN SYMMETRIC KEY SymmetricKeyNass
Decryption BY password ='Voila123';

UPDATE fournisseurs SET nassCrypteByKey = EncryptByKey
(Key_GUID('SymmetricKeyNass'),'999-120-506') where idfournisseur=1;
--fermer la clé de chiffrement
CLOSE SYMMETRIC KEY SymmetricKeyNass;
select * from fournisseurs; --pour vérifier
```

Le chiffrement des données: Le chiffrement : Exemple (encrypter le NAS)

Pour décrypter:

- On ouvre la clé
- On décrypte.

```
OPEN SYMMETRIC KEY SymmetricKeyNass
Decryption BY password ='Voila123';

-- on affiche le nas décrypté
SELECT nom, nassCrypteByKey AS 'Nas encrypté',
CONVERT(varchar, DecryptByKey(nassCrypteByKey)) AS 'nas decrypté'
FROM fournisseurs where idfournisseur=1;
```





CONCLUSION

QUESTIONS ??