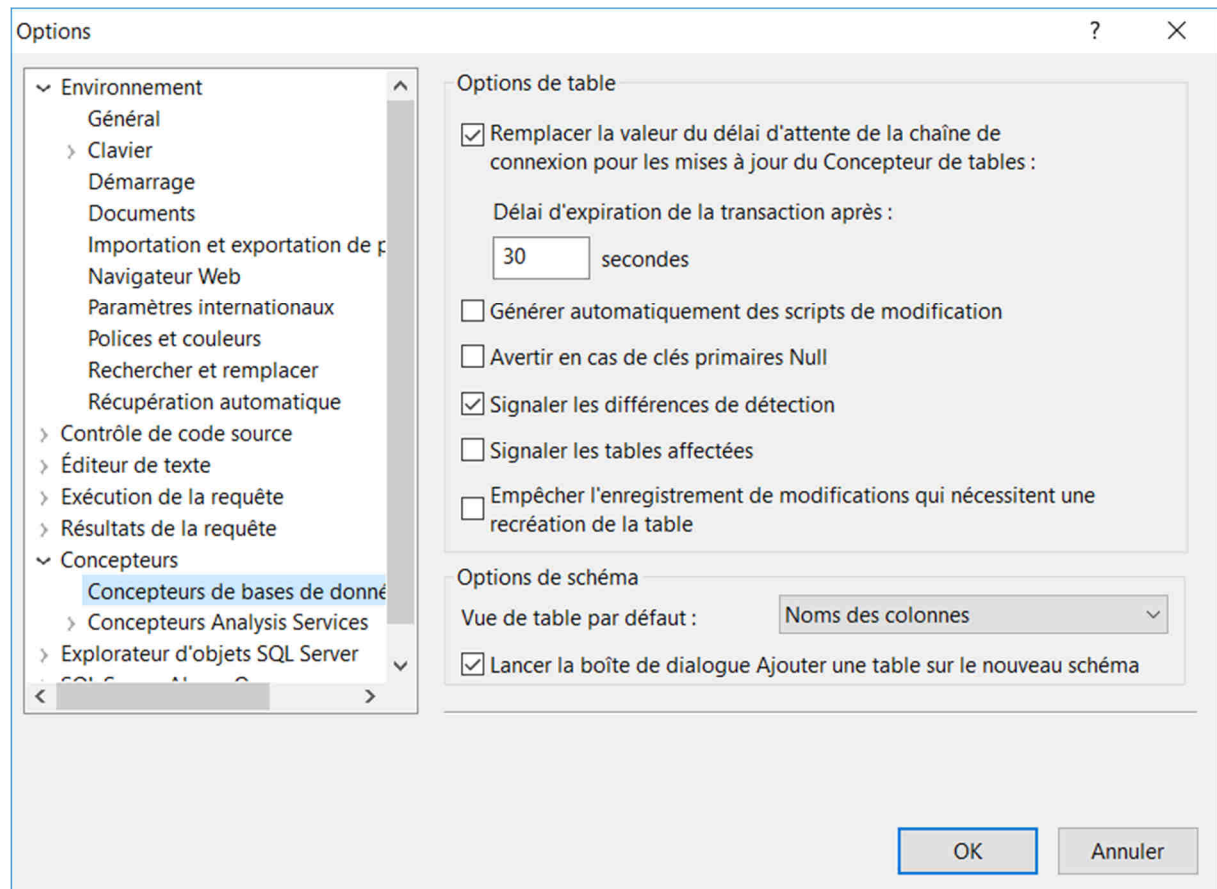


Déclencheur-Procédure stockée

1 -	Notions.....	2
2 -	Cahier des charges	3
3 -	Définition.....	3
3.1 -	Contraintes	3
3.1.1 -	Type de données	3
3.1.2 -	Champs (clé primaire, null interdit, index)	4
3.1.3 -	Validation de table	4
3.1.4 -	Intégrité référentielle avec relation entre tables	4
3.2 -	Trigger ou déclencheur.....	4
3.2.1 -	Notion générale	4
3.2.2 -	Création	4
3.3 -	Déclencheur standard	5
3.3.1 -	Gérer la casse d'un texte.....	5
3.3.2 -	Archiver automatiquement ;.....	5
3.3.3 -	Modifier une propriété (N1) ;.....	5
3.3.4 -	Modifier une propriété (N2) ;.....	6
3.3.5 -	Modifier une propriété (N3) ;.....	6
3.3.6 -	Générer une identification (login/mot de passe) ;.....	6
3.4 -	Déclencheur sur vue.....	6
3.4.1 -	Notions d'une vue	6
3.4.2 -	Création de la vue « VoitureCategorie »	7
3.4.3 -	Amélioration du déclencheur sur la vue « VoitureCategorie »	7
3.4.4 -	Modification d'informations dans la vue V_01_VoitureCategorie.....	7
3.4.5 -	Suppression d'informations dans la vue LocationClient.....	8
3.5 -	Gestion de l'héritage.....	8
3.5.1 -	Notions.....	8
3.5.2 -	Création des vues.....	8
3.5.3 -	Création des triggers	9
3.5.4 -	Tester dans chaque vue	10
3.6 -	Insertion avec compteur	10
3.7 -	Procédure stockée.....	10
3.7.1 -	Notions.....	10
3.7.2 -	Création	11
3.7.3 -	Modification.....	11
3.7.4 -	Procédure stockée sans paramètre.....	11
3.7.5 -	Procédure stockée avec paramètre.....	11
3.7.6 -	Appel d'une PS paramétrée avec PDO	13

1 - Notions

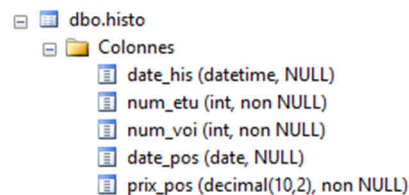
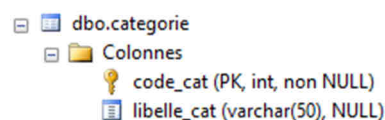
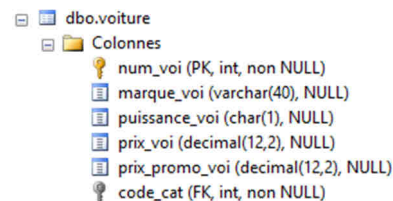
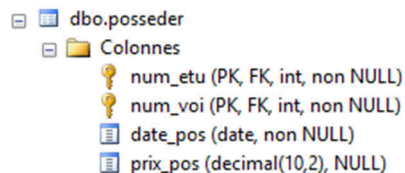
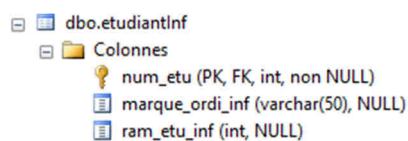
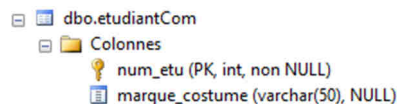
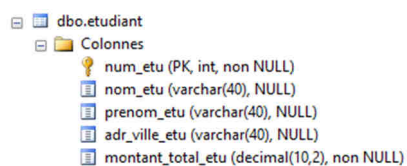
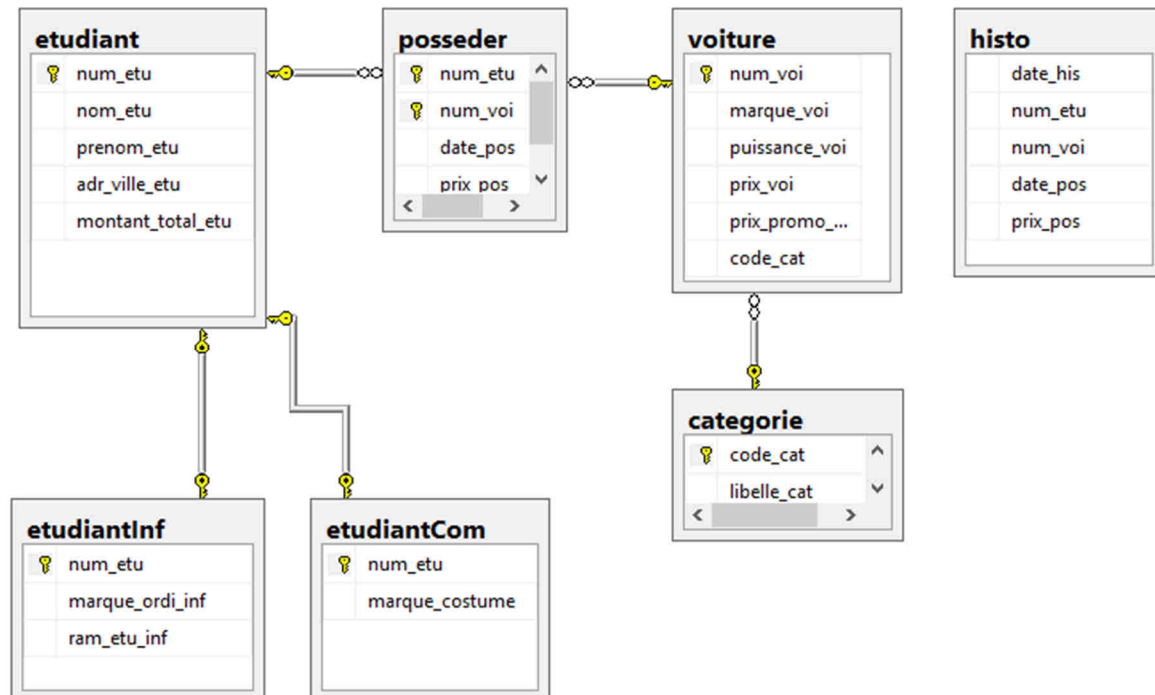
- Si vous ne pouvez plus modifier une table déjà créée, il faut changer une option dans le sgbd. Cliquer sur le menu « outil/option ». Sélectionner « concepteur/concepteur base de données ». Décocher « empêcher l'enregistrement de modifications qui nécessitent une recréation de tables ».



- Le type « booléen » n'existe pas. Choisir le type « bit ».
- Enfin, préférer le type « decimal(10,2) » pour les réels.
- Pour visualiser certains changements de la bd, d'une table, ..., cliquer sur le choix « Actualiser »
- Pour réaliser un trigger, une procédure stockée la première fois, utiliser la clause « CREATE », sinon la clause « ALTER » pour les modifier.

2 - Cahier des charges

Cas Faculté



3 - Définition

Réaliser des règles de gestion d'entreprise en utilisant des contraintes d'intégrité.

3.1 - Contraintes

3.1.1 - Type de données

- Contrôler une donnée si elle est de type booléen, entier, chaîne de caractères, ...

3.1.2 - Champs (clé primaire, null interdit, index)

- Unicité de chaque étudiant (Numéro étudiant) possède un numéro unique ;
- Interdiction d'avoir une valeur nulle pour le nom ;

3.1.3 - Validation de table

- Prix hors saison < prix en saison ;
 - Prix promotionnel < prix courant du produit ;
- Contraintes CHECK entre prix promo et prix courant

3.1.4 - Intégrité référentielle avec relation entre tables

Une voiture possède une catégorie (collection, occasion, neuve).

- Création d'une voiture si la catégorie a été créée auparavant ;
- Interdiction de supprimer une catégorie si une voiture l'a possédé ;

3.2 - Trigger ou déclencheur


3.2.1 - Notion générale

Déclencher l'exécution d'une ou plusieurs instructions, lorsqu'un évènement associée à une table ou une vue se réalise.

- Evènements :
 - o INSERT : insertion ;
 - o UPDATE : modification ;
 - o DELETE : suppression ;
- Type de déclencheurs :
 - o AFTER ou FOR ; seulement sur les tables ;
 - o BEFORE ;
 - o INSTEAD OF :
 - Réaliser des contrôles d'intégrité sur des données obtenues à l'aide des instructions INSERT et UPDATE ;
 - Mises à jour de vues en fonction d'action ;
- Création de tables temporaires :
 - o Inserted : copie des lignes insérées ou modifiées ;
 - o Deleted : copie des lignes supprimées ;

3.2.2 - Création

- Sélectionner et développer la base de données (bouton +) ;
- Sélectionner et développer la table ou vue (bouton +) ;
- Sélectionner « déclencheur » et cliquer sur le bouton droit de la souris et choisir « nouveau déclencheur » ;

- Saisir le déclencheur, le tester  et l'exécuter  ;

```
CREATE [ OR ALTER ] TRIGGER [ schema_name . ]trigger_name
ON { table | view }
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS {
    BEGIN
        .....
    END
GO
}
```

- Remarque ;
 - o Exécution sans erreur d'un déclencheur implique son apparition dans l'arborescence ;
 - o Modifier un déclencheur ; remplacer la clause CREATE par ALTER ;
 - o Sauvegarder le déclencheur ;

- Tester le déclencheur ;
 - o Ouvrir la table ou la vue ;
 - o Réaliser l'instruction correspondante (INSERT, UPDATE, DELETE) ;
 - o Vérifier si le déclencheur a bien été réalisé ;

3.3 - Déclencheur standard

3.3.1 - Gérer la casse d'un texte

- Transformer en majuscule les noms des étudiants (insert, update).

```
CREATE TRIGGER T_01_NomMaj
ON etudiant
AFTER INSERT, UPDATE
AS
BEGIN
    UPDATE
        etudiant
    SET
        nom_etu = UPPER(inserted.nom_etu)
    FROM
        etudiant, inserted
    WHERE
        etudiant.num_etu=inserted.num_etu
END
```

- Vérifier en insérer un étudiant dans la table etudiant.

Exercice : T_02_NomMaj

Noms des étudiants en majuscule (insert, update).

Première lettre du prénom en majuscule et le reste du prénom en minuscule.

3.3.2 - Archiver automatiquement ;

Exercice : T_03_HistoLocation

- Copier une ligne supprimée de la table « posséder » dans la table « histoPos ».

```
CREATE TRIGGER T_03_HistoPos
ON posseder
AFTER DELETE
AS
BEGIN
    INSERT INTO histo (date_his, num_etu, num_voi, date_pos, prix_pos)
    SELECT
        getdate(), num_etu, num_voi, date_pos, prix_pos
    FROM
        deleted
END
GO
```

3.3.3 - Modifier une propriété (N1) ;

Exercice : T_04_MajMontantEtudiant

- Mettre à jour la propriété « Montant_total_etu » de la table « etudiant » lors de l'enregistrement d'une possession.

```
CREATE TRIGGER T_04_MajMontantEtudiant
ON Posseder
FOR INSERT, UPDATE AS
BEGIN
    UPDATE
        Etudiant
    SET
        montant_total_etu= montant_total_etu + prix_pos
    FROM
        inserted
```

```

WHERE
    etudiant.num_etu = inserted.num_etu
END

```

Remarque :

Les valeurs des propriétés numériques des tables sont initialisées avec une valeur égale à 0. Réaliser une contrainte en affectant une valeur égale à 0 par défaut.

3.3.4 - Modifier une propriété (N2) ;

Exercice : T_05_MajMontantEtudiant2

Un déclencheur (INSERT) assure la modification de la propriété « montant_total_etu » de la table « etudiant » en fonction du prix récupéré depuis la table voiture (prix_voi).

```

CREATE TRIGGER [dbo].[T_05_MajMontantEtudiant2]
ON [dbo].[posseder]
FOR INSERT, UPDATE AS
BEGIN
    DECLARE @Prix INT
    SELECT
        @Prix = Prix_Voi
    FROM
        voiture, inserted
    WHERE
        Voiture.Num_Voi = inserted.Num_Voi

    UPDATE
        Etudiant
    SET
        Montant_Total_etu= Montant_Total_etu+Prix_pos
    FROM
        inserted
    WHERE
        etudiant.Num_etu=inserted.Num_etu
END

```

3.3.5 - Modifier une propriété (N3) ;

Exercice : T_06_MontantEtudiant2

Le prix cumulé à la propriété « montant_total_etu » dépend de la puissance de la voiture. Si celle-ci est inférieure à 6 cv, le prix correspond à la valeur du prix dans la table voiture, sinon la valeur du prix est promotionnel.

3.3.6 - Générer une identification (login/mot de passe) ;

Exercice : T_07_CreationLoginMdp

- Création d'un trigger pour générer un login et mot de passe si insertion d'un nouvel étudiant ;
 - o Ajouter deux propriétés « login » et « password » de type varchar(50) dans la table « etudiant ».
 - o « Login » est constitué de minuscule avec la première lettre du prénom et le nom ;
 - o « Password » est constitué des 4 premières lettres du login ;

3.4 - Déclencheur sur vue

3.4.1 - Notions d'une vue

- Table virtuelle contenant des données provenant d'une ou plusieurs tables à l'aide d'une clause SELECT ;
- Cacher la complexité d'une bd ;
- Syntaxe ;

```

CREATE VIEW NomVue (col1, col2, ...)
AS SELECT ...

```

- Insertion de lignes est possible seulement sur les vues contenant une seule table.
Utilisation d'un déclencheur pour insérer des lignes dans plusieurs tables d'une vue.

3.4.2 - Création de la vue « VoitureCategorie »

- Création 1

```
CREATE VIEW V_01_VoitureCategorie2 AS
SELECT
    Voiture.num_voi, Voiture.Marque_voi,
    categorie.code_cat, categorie.libelle_cat
FROM
    voiture INNER JOIN categorie ON voiture.Code_cat = categorie.Code_cat
```
- Création 2
Cliquer sur le bouton droit de la souris en se positionnant auparavant sur la vue de la table.
Choisir « nouvelle vue » et sélectionner les propriétés à visualiser.
- Insertion de données dans la vue VoitureCategorie

```
INSERT INTO V_01_VoitureCategorie
(num_voi, Marque_voi, code_cat, libelle_cat)
VALUES (50, 'renault', 6, 'collection2')
```
- Quel est le résultat obtenu ?

3.4.3 - Amélioration du déclencheur sur la vue « VoitureCategorie »

Exercice : T_08_InsVoitureCategorie

- Création du trigger ci-contre pour éviter l'erreur précédente.

```
CREATE TRIGGER [dbo].[T_08_InsVoitureCategorie] ON [dbo].[V_01_VoitureCategorie]
INSTEAD OF INSERT AS
BEGIN
    INSERT INTO categorie (code_cat, libelle_cat)
    SELECT code_cat, libelle_cat FROM inserted
    INSERT INTO voiture (Num_voi, marque_voi, code_cat)
    SELECT Num_voi, marque_voi, code_cat FROM inserted
END
```
- Insertion de données dans la vue V_01_VoitureCategorie

```
INSERT INTO V_01_VoitureCategorie
(num_voi, marque_voi, code_cat, libelle_cat)
VALUES (50, 'renault', 6, 'collection2')
```
- Quel est le résultat obtenu ?

Exercice : T_09_MajLocCli

- Vérifier l'existence du client en modifiant le trigger précédent :
 - o La catégorie existe : récupération de son numéro et insérer les informations dans la table « voiture » ;
 - o Sinon, insérer les informations dans les tables « catégorie » et « voiture ».
 - o Insérer un client existant ;

```
INSERT INTO V_01_VoitureCategorie
(Num_voi, marque_voi, code_cat, libelle_cat)
VALUES (10, 'Renault', 10, collection2)
```
 - o Insérer un nouveau client ;

```
INSERT INTO V_01_VoitureCategorie
(Num_voi, marque_voi, code_cat, libelle_cat)
VALUES (11, 'Renault', 10, collection2)
```

3.4.4 - Modification d'informations dans la vue V_01_VoitureCategorie

Exercice : T_10_MajLocCli

- Définir un trigger de modification sur la vue « V_01_VoitureCategorie », exécuté lors de la modification d'un enregistrement dans la vue ;
 - o Récupérer les valeurs des données dans la table « inserted » dans des variables ;
 - o Modifier les lignes concernées ;

3.4.5 - Suppression d'informations dans la vue LocationClient

Exercice : T_11_SupLocCli

- Définir un trigger de suppression sur la vue « V_01_VoitureCategorie », exécuté lors de la suppression d'un enregistrement de la table « voiture » dans la vue ;
 - o Récupérer les valeurs des données dans la table « deleted » ;
 - o Supprimer les lignes concernées ;

3.5 - Gestion de l'héritage

3.5.1 - Notions

- Insertion des informations dans la table « mère » et une des tables « filles » ;
- Création d'une vue associant ces deux tables ;
- Déclenchement par un trigger avec le mot-clé **INSTEAD OF** avant l'exécution d'un traitement concernant une insertion, modification, suppression de données.
- Utilisation des tables « deleted » et « inserted » ;

Dans une table ou vue,

- La commande **INSERT** insère les données dans la table « inserted », et exécution du déclencheur ;
- La commande **DELETE** supprime les données qui sont insérées dans la table « deleted », et exécution du déclencheur ;
- La commande **UPDATE** modifie les données dans la table « inserted » (avant) et « deleted » (après), et exécution du déclencheur ;

Le déclencheur peut tester les valeurs avant et après modification, avant de procéder à la modification.

3.5.2 - Création des vues


```
CREATE VIEW V_02_EtudiantEtudiantInf AS
SELECT
    etudiant.num_etu, nom_etu, prenom_etu, adr_ville_etu, montant_total_etu,
    marque_ordi_inf, ram_etu_inf
FROM etudiant INNER JOIN etudiantInf ON etudiant.num_etu = etudiantInf.num_etu

CREATE VIEW V_03_EtudiantEtudiantCom AS
SELECT
    etudiant.num_etu, nom_etu, prenom_etu, adr_ville_etu, montant_total_etu,
    marque_costume_com
FROM etudiant INNER JOIN etudiantCom ON etudiant.num_etu = etudiantCom.num_etu
```

3.5.3 - Création des triggers

Exercice : T_12_InsLocClientCE

```
- Sur la vue V_02_EtudiantEtudiantInf ;
CREATE TRIGGER T_11_ins EtudiantEtudiantInf ON V_02_EtudiantEtudiantInf
INSTEAD OF INSERT
AS BEGIN
BEGIN TRY
    BEGIN TRANSACTION
        -- insertion dans les tables
        INSERT INTO etudiant
        (etudiant.num_etu, nom_etu, prenom_etu, adr_ville_etu, montant_total_etu)
        SELECT
            num_etu, nom_etu, prenom_etu,
            adr_ville_etu, montant_total_etu
        FROM inserted
        INSERT INTO etudiantInf (etudiant.num_etu, marque_ordi_inf, ram_etu_inf)
        SELECT num_etu, marque_ordi_inf, ram_etu_inf
        FROM inserted

    COMMIT TRANSACTION
        PRINT 'transaction correcte, insertion dans les tables Etudiant et EtudiantInf '
END TRY

BEGIN CATCH
    PRINT      'ErrNumber'+CONVERT(varchar(50), ERROR_NUMBER())+
              'ErrSeverity'+CONVERT(varchar(5), ERROR_SEVERITY())+
              'ErrState'+CONVERT(varchar(5), ERROR_STATE())+
              'ErrProcedure'+ISNULL(ERROR_PROCEDURE(),'-')+
              'ErrLine'+CONVERT(varchar(5), ERROR_LINE())
    PRINT      'ErrMsg'+ERROR_MESSAGE()
    PRINT      'Un étudiant est aussi un un étudiant en informatique'
    ROLLBACK TRANSACTION
END CATCH
END
```

Exercice : T_13_Ins EtudiantEtudiantCom

```
- Sur la vue V_03_EtudiantEtudiantCom ;
```

A faire.

3.5.4 - Tester dans chaque vue

- Insertion d'un étudiant etudiantInf ;
INSERT INTO V_02_etudiantetudiantInf
(etudiant.num_etu, nom_etu, prenom_etu, adr_ville_etu, montant_total_etu,
marque_ordi_inf, ram_etu_inf)
VALUES (13, 'durand', 'Paul', 'Montpellier',0, 'pc2', '8')
- Insertion d'un étudiant etudiantCom ;
INSERT INTO V_02_etudiantetudiantInf
(etudiant.num_etu, nom_etu, prenom_etu, adr_ville_etu, montant_total_etu,
marque_ordi_inf, ram_etu_inf)
VALUES (13, 'durand', 'Paul', 'Montpellier',0, 'marque3')

3.6 - Insertion avec compteur

Exercice : T_16_InsVoitureCategorie

- Création d'un trigger nommé T_16_InsVoitureCategorie sur la vue V_01_VoitureCategorie2

```
CREATE TRIGGER T_16_InsVoitureCategorie ON V_01_VoitureCategorie2
INSTEAD OF INSERT, UPDATE AS
BEGIN
    INSERT INTO categorie (code_cat, libelle_cat)
    SELECT @@identity, libelle_cat FROM inserted
    INSERT INTO voiture (Num_voi, marque_voi, code_cat)
    SELECT Num_voi, marque_voi, code_cat FROM inserted
END
```

NB : la variable @@identity contient le dernier id compteur inséré

- Insertion de données dans la vue V_01_VoitureCategorie
INSERT INTO V_01_VoitureCategorie
(marque_voi, code_cat, libelle_cat)
VALUES ('renault', 6, 'collection2')

3.7 - Procédure stockée

3.7.1 - Notions

- Langage procédurale utilisant des instructions transact-sql correspondant à des traitements sur bd ;
- Exécution côté serveur ;
- Possibilité d'avoir des :
 - o Paramètres d'entrée, de sortie ;
 - o Variables locales ;
 - o Retourner des données, curseurs, ... ;
- Type
 - o ps systeme (préfixe sp_) : gestion du sgbd et afficher des informations sur les données et utilisateurs ;
 - o ps étendues (préfixe xp_) : bibliothèque de ddl ;
 - o ps utilisateur ;
- Avantage :
 - o Sécurité contre les attaques par injection ;
 - o Réduction du trafic réseau entre les applicatifs et le serveur ;

- Maintenabilité du code par une programmation réutilisable (passage par paramètres, ...) ;
- Compilation de la ps entraîne un stockage dans la mémoire « cache » pour diminuer le temps de travail du processeur ;

3.7.2 - Création

- Développer la bd ;
 - Menu programmabilité / Procédures stockées / Nouvelle PS
- ```
CREATE PROCEDURE Nom_PS(parametre)
AS
 BEGIN
 instruction
 END
GO
```

### 3.7.3 - Modification

Remplacer CREATE par ALTER.

### 3.7.4 - Procédure stockée sans paramètre

- Afficher les étudiants qui possèdent une voiture.
- ```
CREATE PROCEDURE PS_01_EtuPosVoi
AS
BEGIN
    SELECT etudiant.num_etu, nom_etu, prenom_etu, adr_ville_etu, montant_total_etu
    FROM etudiant, posseder
    WHERE etudiant.num_etu = posseder.num_etu
END
GO
```
- Création d'une nouvelle requête ;
- ```
EXEC PS_01_EtuPosVoi
```

### 3.7.5 - Procédure stockée avec paramètre

#### 3.7.5.1 - Notions

- 1024 paramètres possible ;
- Faire précéder l'identificateur du caractère @

#### 3.7.5.2 - Liste des voitures (numéro, marque, puissance\_voi, prix\_voi) possédées par un étudiant donné (nom\_etu).

- Création ;
- ```
CREATE PROCEDURE PS_02_VoiEtu @NomEtudiant char(50)
AS
BEGIN
    SELECT
        Posseder.num_voi, marque_voi, puissance_voi, prix_voi
    FROM Voiture, Posséder, Etudiant
    WHERE
        Voiture.Num_voi = Posseder.Num_voi
        AND etudiant.Num_etu = Posseder.Num_etu
        AND etudiant.Nom_etu = @NomEtudiant
END
GO
```
- Exécution ;

```
EXEC PS_02_VoiEtu 'dupont'
```

3.7.5.3 - Prix d'une voiture (prix_promo_voi, prix_voi) possédée en fonction de 2 dates données en paramètres

- Création ;

```
CREATE PROCEDURE PS_04_PrixVoiture2Dates
    @DateDeb date,
    @DateFin date,
    @PrixVoiture money OUTPUT,
    @PrixPromoVoiture money OUTPUT
AS
BEGIN
    SELECT @PrixVoiture = Prix_voi, @PrixPromoVoiture = Prix_promo_voi
    FROM voiture, posseder
    WHERE voiture.num_voi = posseder.num_voi
    AND date_pos >= @DateDeb
    AND date_pos <= @DateFin
END
GO
```

- Exécution ;

Attention, évitez d'avoir des valeurs nulles dans les propriétés à afficher.

```
DECLARE @Prix1 float
```

```
DECLARE @Prix2 float
```

```
EXEC PS_04_PrixVoiture2Dates
    '2018/01/01',
    '2018/01/31',
    @PrixVoiture=@Prix1 OUTPUT,
    @PrixPromoVoiture=@Prix2 OUTPUT
```

```
PRINT @Prix1
```

```
PRINT @Prix2
```

3.7.5.4 - Gestion des erreurs : try...catch

- Notions ;

Intercepter une erreur ;

- Création ;

```
CREATE PROCEDURE PS_05_EnregVoiCat
    @NumVoi int,
    @MarqueVoi varchar(30),
    @PrixVoi money,
    @PrixPromoVoi money,
    @CodeCat int,
    @LibelleCat varchar(30)
AS
```

```
BEGIN TRY
```

```
-- insertion dans les tables
```

```

INSERT INTO Voiture (Num_voi, Marque_Voi, Prix_voi, PrixPromo_voi, Code_cat)
VALUES (@NumVoi, @MarqueVoi, @PrixVoi, @PrixPromoVoi, @CodeCat)
INSERT INTO Categorie (Code_Cat, Libelle_cat)
VALUES (@CodeCat, @LibelleCat)
END TRY

```

```

BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS NumeroErreur,
        ERROR_SEVERITY() AS SeverityErreur,
        ERROR_STATE() AS NumeroErreur,
        ERROR_PROCEDURE() AS ProcedureErreur,
        ERROR_LINE() AS LigneErreur,
        ERROR_MESSAGE() AS MessageErreur
END CATCH

```

- Exécution1 ;
- EXEC PS_05_EnregVoiCat 100, "Renault", 100, 110, 1, "categ2"
- Résultat1 ;

NumeroErreur	SeverityErreur	NumeroErreur	ProcedureErreur	LigneErreur
MessageErreur				
2627	14	1	PS_05_EnregVoiCat	11

Violation de la contrainte PRIMARY KEY « PK_voiture ». Impossible d'insérer une clé en double dans l'objet « dbo.voiture ». Valeur de clé dupliquée : (100).

- Exécution2 ;
- EXEC PS_05_EnregVoiCat null, "Renault", 100, 110, 1, "categ2"
- Résultat2 ;

NumeroErreur	SeverityErreur	NumeroErreur	ProcedureErreur	LigneErreur
MessageErreur				
515	16	2	PS_05_EnregVoiCat	11

Impossible d'insérer la valeur NULL dans la colonne 'num_voi', table 'bd_faculte.dbo.voiture'. Cette colonne n'accepte pas les valeurs NULL. Échec de INSERT.

3.7.6 - Appel d'une PS paramétrée avec PDO

3.7.6.1 - Nombre de clients d'une ville

- Création ;
- CREATE PROCEDURE PS_06_NbEtudiantVille
 @VilleEtu varchar(50) OUTPUT,
 @NbEtu int OUTPUT

```

AS
BEGIN
    SELECT @NbEtu = COUNT(*)
    FROM etudiant
    WHERE adr_ville_etu = @VilleEtu

```

```
END
```

```
GO
```

- Exécution ;
- DECLARE @NbEtudiant int
- EXEC PS_06_NbEtudiantVille

```

        @VilleEtu = 'Nimes',
        @NbEtu = @NbEtudiant OUTPUT
SELECT @NbEtudiant
- Création d'un fichier nommé PS_06_nbEtudiantVille.php ;
<?php
// $dbh = new PDO("sqlsrv:Server=NomServeur;database=nomDB", "login", "password");
$ville=utf8_encode('Montpellier');
$NbEtu=100;
$stmt=$dbh->prepare("EXEC dbo.PS_06_NbEtudiantClient ?, ? WITH RECOMPILE");
// $dbh->exec('SET CHARACTER SET UTF8');
$stmt=bindParam(1, $Ville, PDO::PARAM_STR | PDO::PARAM_INPUT_OUTPUT,50);
$stmt=bindParam(2, $NbEtu, PDO::PARAM_INT | PDO::PARAM_INPUT_OUTPUT,3);
$stmt->execute();
print_r($stmt->errorInfo());
echo '<br/>';
print 'Nombre etudiant='.$NbEtu;

?>

```

3.7.6.2 - Gestion d'une liste déroulante par PS

```

- Création ;
CREATE PROCEDURE PS_07_VilleEtudiant
AS
BEGIN
    SELECT DISTINCT Ville_etu
    FROM Etudiant ;
END
GO
- Exécution ;
EXEC PS_07_VilleEtudiant
- Création d'un fichier nommé PS_07_VilleEtudiant.php ;
<?php
function execPsSansParam($ps) {
    try {
        //connexion
        $dbh=new PDO("sqlsrv:Server=NomServeur;
                        Database=NomDB","login","password") ;
        $stmt=$dbh->prepare('EXEC $ps') ;
        $stmt->execute() ;
        return $stmt ;
    }
    catch (PDOException $e) {
        echo 'erreur : '.$e->getMessage() ;
        exit ;
    }
}
?>
<-- Afficher et alimenter la combo -->
<p>Nb d'étudiants d'une ville </p>
<form action = "<?php echo $_SERVER['SCRIPT_NAME']?>" method="GET">

```

```

<select name="ChoixVille">
    <?php
        $ps="PS_07_VilleEtudiant" ;
        $lesVilles=execPSsansParam($ps) ;
        while ($ville= $lesvilles->fetch()){
            ?>
                <option> <?php echo $ville[0];?></option>
        <?php } ?>
    </select>
    <input type="submit" value="Rechercher">
</form>

```

3.7.6.3 - Afficher des données en fonction d'une liste déroulante

- Création ;

```

CREATE PROCEDURE PS_08_ListeEtudiantVille @ville VARCHAR(30) OUTPUT
AS
BEGIN
    SELECT GenreCli, NomCli, PrenomCli
    FROM Client
    WHERE villeCli = @ville;
END
GO

```

- Exécution ;

```
EXEC PS_08_ListeEtudiantVille 'Nimes';
```

- Création d'un fichier nommé PS_08_ListerEtudiantParVille.php ;

```

<?php
function listerEtudiantParVille($ps, $paramEntree) {
    $ville=$paramEntree[0];
    echo $ville;
    try {
        $dbh = new PDO("sqlsrv.Server=NomServeur;
        database=nomDB", "login", "password") ;
        $stmt=$dbh->prepare("EXEC $ps ?") ;
        //$dbh->exec('SET CHARACTER SET UTF8') ;
        $stmt=bindParam(1, $Ville,
            PDO::PARAM_STR | PDO::PARAM_INPUT_OUTPUT,25) ;
        $stmt->execute() ;
        return $stmt;
    }
    catch (PDOException $e){
        echo 'Erreur:'. $e->getMessage();
        exit;
    }
}
?>
<?php
if (isset($_GET['choixVille'])){
    //liste Etudiants d'une ville donnée

```

```

$ps="EtudiantParVille";
$paramsEntree[ ]=$_GET['choixVille'];
$EtudiantVille=listerEtudiantVille($ps,$paramsEntree);
if (empty($EtudiantVille)){
    echo "<p>Aucun Etudiant dans cette ville</p>";
}else{
    ?>
    <table border="1" align="left">
        <tr>
            <th colspan="3" align="center">Liste Etudiants</th>
        </tr>
        <tr>
            <th>numéro</th>
            <th>nom</th>
            <th>prenom</th>
        </tr>
        <?php
        while ($Etudiant=$EtudiantVille->fetch()){
            ?>
            <tr>
                <td align="center"><?php echo $Etudiants[0]; ?></td>
                <td align="center"><?php echo $Etudiants[1]; ?></td>
                <td align="center"><?php echo $Etudiants[2]; ?></td>
            </tr>
        <?php
        }
        ?>
    </table>

}
?>
<?php
//récupération de la valeur du dernier id inséré
$dbh = new PDO("sqlsrv:Server=NomServeur;
    database=nomDB", "login", "password") ;
$res = $dbh->exec("INSERT INTO Etudiant (Nom_etu) VALUES ('Dupont')");
$LASTROW=$dbh->lastInserted('Etudiant');
echo $LASTROW;
$stmt=$dbh->prepare("EXEC $ps ?") ;
?>

```