

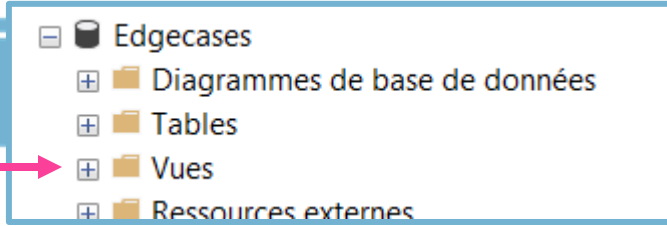
# R04

## Vues, fonctions et procédures

Bases de données et programmation Web



- ❖ Vues
- ❖ Fonctions
- ❖ Transact-SQL
- ❖ Procédures stockées



## ❖ Vue

### ◆ Table « virtuelle » basée sur le résultat d'une requête SELECT.

- Permet de créer une représentation simplifiée de certaines données qui se prête mieux à certaines fonctionnalités que les tables « brutes ».
- On peut faire des requêtes SELECT sur des vues exactement comme on le ferait sur des tables ordinaires.

### ◆ Créer une vue :

On peut personnaliser la requête  
**SELECT** autant que nécessaire.  
(Agrégation, jointures, etc.)

```
CREATE VIEW schema.vw_nomVue AS  
SELECT colonne1, colonne2, ...  
FROM nom_table  
WHERE ...
```

### ◆ Supprimer une vue :

```
DROP VIEW schema.vw_nomVue
```



## ❖ Utilisation la plus fréquente:

- ◆ Simplement pour montrer toutes les données qu'on veut voir sur un sujet, même si ces données sont dans plusieurs tables.

```
USE BDRecettes_R06
GO
-- Comme quand on veut montrer une recette, on veut aussi toujours voir sa catégorie et son thème, s'il existe, on se fait une vue
CREATE VIEW Recettes.vw_CategorieRecetteTheme
AS
    SELECT NomCategorie, NomRecette, NomTheme, NbPortions, TempsDePreparation, TempsDeCuisson, Calories, R.RecetteID, C.CategorieID, T.ThemeID
    FROM Recettes.Categorie C
    INNER JOIN Recettes.Recette R
    ON C.CategorieID = R.CategorieID
    LEFT JOIN Recettes.Theme T
    ON R.ThemeID = T.ThemeID
GO
```

} Requête



❖ On peut utiliser une vue comme une table.

```
SELECT NomCategorie, NomRecette, NomTheme, NbPortions, TempsDePreparation, TempsDeCuisson, Calories
FROM Recettes.vw_CategorieRecetteTheme
ORDER BY NomCategorie, NomRecette
```

% ▾

Résultats						
NomCategorie	NomRecette	NomTheme	NbPortions	TempsDePreparation	TempsDeCuisson	Calories
Accompagnements	Les meilleurs pomme de terre rôties au four	NULL	6	10 minutes	25 minutes	419
Bouchées	Bouchées croustillantes végété	NULL	16	15 minutes	8 minutes	44
Bouchées	Muffins farcis choco-Noisette	NULL	12	15 minutes	55 minutes	336
Collations	Boules d'énergie "araignées" aux carottes	Halloween	20	15 minutes	NULL	155
Desserts	Gâteau choco-noisettes sur la plaque	NULL	15	15 minutes	25 minutes	280
Desserts	Muffins brioches à la cannelle	NULL	10	15 minutes	20 minutes	213
Entrées et soupes	Soupe aux légumes, boeuf et orge à la mijoteuse	NULL	4	15 minutes	8 heures à la mijoteuse	318
Plats principaux	Boeuf au brocoli express	NULL	4	15 minutes	4 minutes	414
Plats principaux	Couppelles de porc effiloché, salade de chou et pomme	NULL	8	15 minutes	12 minutes	247
Plats principaux	Macaroni chinois au sans-viande hachée	NULL	4	15 minutes	16 minutes	555
Plats principaux	Mini quiches jambon et bacon	NULL	12	15 minutes	18 minutes	380
Plats principaux	Plaque de saucisses, poulet et légumes marinés	NULL	4	15 minutes	140 minutes	380
Plats principaux	Salade de pâtes à l'italienne	NULL	4	15 minutes	10 minutes	499

(13 lignes affectées)



- ❖ Autre utilisation: transformer une requête compliquée qu'on veut faire assez souvent en une vue simple d'utilisation.



## ❖ Exemple #1

- ◆ Dans un *pawn shop* (Boutique de prêts sur gage), les **caissiers** « achètent » des **articles** divers à des clients. Ils essayent d'offrir un **prêt** (prix d'achat) inférieur à l'**estimation** de la vraie valeur de l'article.

Employes.Caissier		
CaissierID	Prenom	Nom
1	Danielle	Rainey
2	Olivia	Berry
3	Corey	Harrison
4	Austin	Russell

Articles.Article					
ArticleID	Nom	Estimation	ValeurPret	ClientID	CaissierID
1	Chaise	10	5	5	3
2	Autographe de Michael Jackson	2000	400	2	2
3	Volvo 200 1989	300	200	1	1
4	Lionel trains The Brute	250000	20	8	2

- Par exemple, on voit que le **caissier #2, Olivia Berry**, a offert un prêt de **20 \$** pour un ensemble de trains miniatures estimé à **250 000 \$**.



## ❖ Exemple #1 (suite)

- ◆ Le propriétaire du pawn shop supervise fréquemment le **nombre d'articles vendus** et les **profits moyens par caissier**. Étant donné que :
  - Le nombre d'articles et les profits moyens sont des données **dérivées**.
  - Cela implique de faire une même requête **très fréquemment**.
- ◆ On peut créer une vue à cette fin :

```
-- Vue 1 : Nombre de ventes et valeur moyenne par caissier

CREATE VIEW Employes.vw_VentesCaissier AS

SELECT C.CaisierID, C.Prenom, C.Nom,
FORMAT(AVG(A.Estimation - A.ValeurPret), 'C', 'en-us') AS ProfitMoyen,
COUNT(A.Estimation) AS NbArticles

FROM Employes.Caisier C
INNER JOIN Articles.Article A
ON C.CaisierID = A.CaisierID

GROUP BY C.CaisierID, C.Prenom, C.Nom
```

On peut même en profiter pour formater certaines données.

↓ Exemples de données dans la vue ↓

VW_VentesCaissier				
CaissierID	Prenom	Nom	ProfitMoyen	NbArticles
1	Danielle	Rainey	\$3,531.52	43
2	Olivia	Berry	\$212.17	72
3	Corey	Harrison	\$72.21	117
4	Austin	Russell	\$21,132.89	37



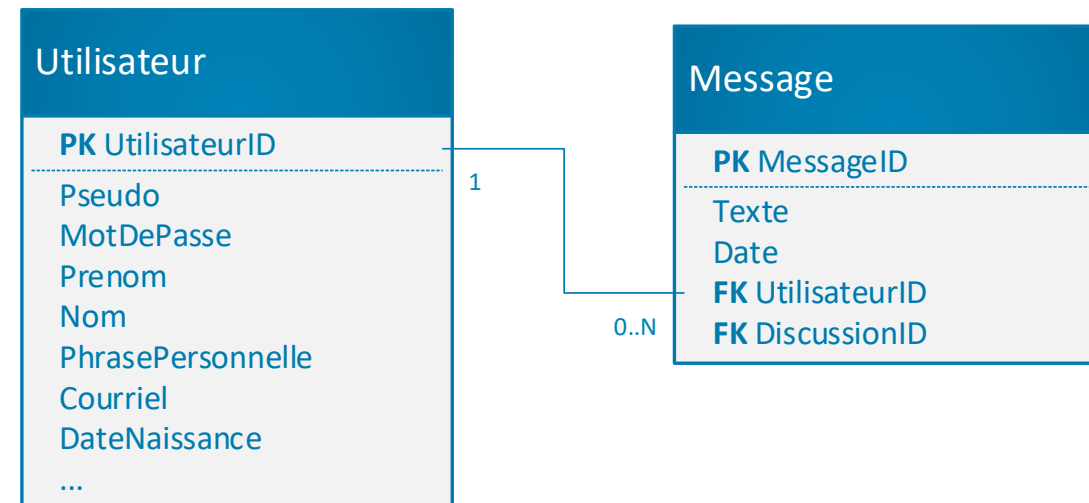


- ❖ Autre utilisation: protéger des données sensibles.
- ❖ On fait une vue qui montre les données et on donne aux utilisateurs le droit d'exécuter la vue. Mais ces mêmes utilisateurs n'ont pas de droits sur les tables utilisées par la vue.



## ❖ Exemple #2

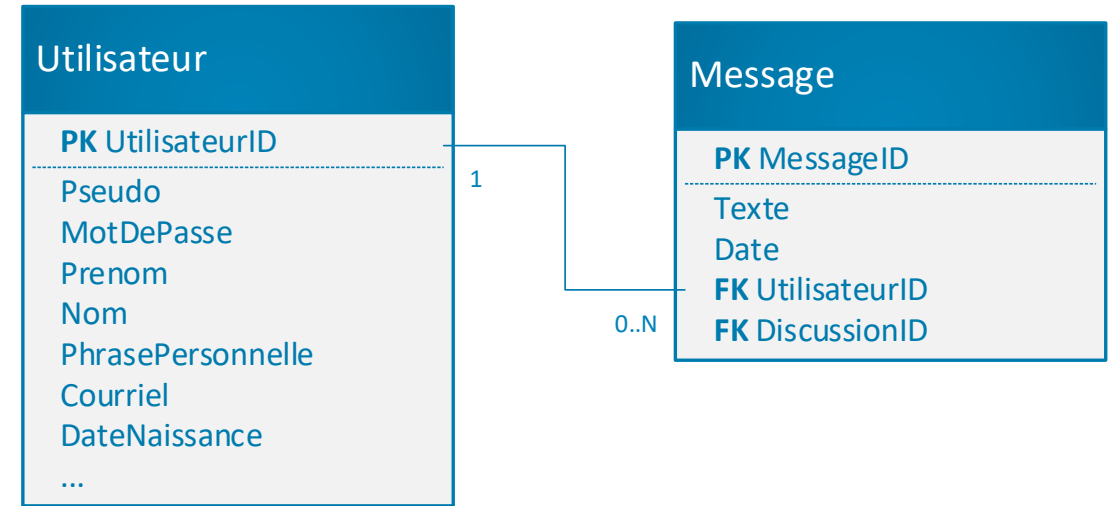
- ◆ Sur un forum Web de discussion, certaines informations clés de l'utilisateur sont affichées lorsqu'on consulte son profil :
  - Pseudo, adresse courriel, phrase personnelle et nombre de messages.
- ◆ Étant donné que :
  - La requête permettant d'obtenir les données à afficher dans un profil est **utilisée fréquemment**.
  - Il y a certaines informations **sensibles** dans la table **Utilisateur**.
  - Le **nombre de messages** est une donnée **dérivée** qu'on ne veut pas forcément avoir dans la table de l'**Utilisateur**.
- ◆ On peut créer une **vue** pour les profils utilisateurs.





## ❖ Exemple #2 (suite)

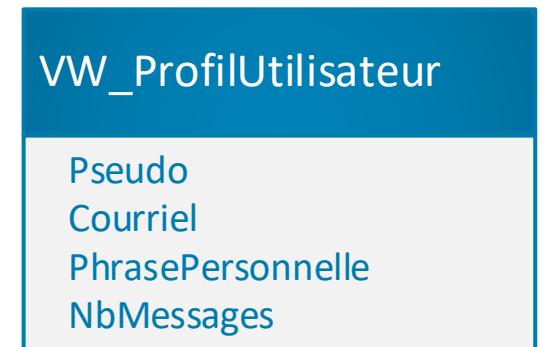
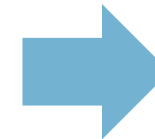
- ◆ Dans la BD, on peut donner des **permissions** sur les **vues**. (Et pas seulement sur des **tables**)
- ◆ Ici on aurait l'opportunité de donner des **permissions** sur la vue **VW\_ProfilUtilisateur** plutôt que sur la table **Utilisateur**.
  - Cela protège ainsi mieux les **données sensibles** des utilisateurs dans certains contextes.



```

CREATE VIEW Utilisateurs.vw_ProfilUtilisateur AS

WITH
M AS (
    SELECT COUNT(MessageID) AS NbMessages, UtilisateurID
    FROM Discussions.Message
    GROUP BY UtilisateurID
)
SELECT U.Pseudo, U.Courriel, U.PhrasePersonnelle, M.NbMessages
FROM Utilisateurs.Utilisateur U
INNER JOIN M
ON M.UtilisateurID = U.UtilisateurID
    
```





## ❖ Bonnes raisons pour créer une vue

- ◆ Simplifier certaines requêtes **complexes** et **fréquentes** 
  - Si un ensemble de données qui implique plusieurs jointures et / ou plusieurs sous-requêtes est fréquemment utilisé, faire une vue qui permet de retrouver l'information nécessaire avec un simple WHERE peut être intéressant.
- ◆ Améliorer la **sécurité** 
  - Comme on peut donner des permissions sur des vues plutôt que sur des tables, si les vues excluent des données sensibles, cela limite certains risques.
- ◆ Simplifier l'usage de **données dérivées** 
  - Si certaines données dérivées (obtenues à l'aide d'agrégations ou de calculs impliquant d'autres données) sont très fréquemment utilisées, les vues représentent un bon compromis pour ne pas avoir à entretenir des données dérivées dans les vraies tables. Cela peut rendre l'intégrité des données dérivées plus simple à gérer.
  - Les vues peuvent d'ailleurs être un compromis à la **dénormalisation**.



## ❖ Précisions sur le fonctionnement

- ◆ Une vue n'est **pas une vraie table**
  - Il faut plutôt les voir comme des « **requêtes SELECT sauvegardées** ».
- ◆ **Mise à jour** des données dans la vue
  - Ajouter, modifier et supprimer des données dans les tables associées à une vue n'implique pas de « mettre à jour » la vue. (Ça ne génère pas d'opérations supplémentaires en background sur la vue)
  - La vue étant seulement une « requête sauvegardée », elle sera toujours **à jour** lorsqu'on l'appellera.
- ◆ Une vue doit être **entretenu**
  - Si la structure de la base de données change, il est possible qu'une vue doive être modifiée également pour respecter la nouvelle structure des tables d'où elle tire ses données.



## ❖ Précisions sur le fonctionnement

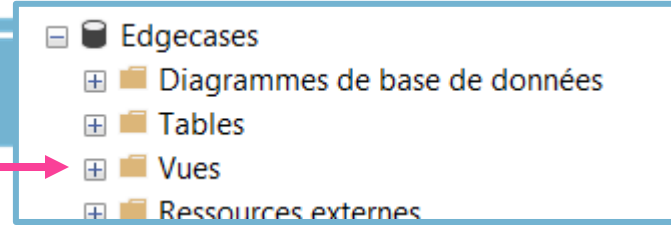
### ◆ Une vue peut avoir un **impact sur la performance**

- Les requêtes SELECT peuvent être **optimisées\***. Il y a plusieurs manières d'obtenir les mêmes données, mais le but est de trouver des manières **économiques**. Comme une vue est une **requête répétée fréquemment**, une vue mal optimisée (ou bien optimisée) peut avoir un impact notable sur la performance.

### ◆ Il est interdit d'utiliser **ORDER BY** pour une vue

- SAUF si on utilise **TOP** pour limiter le nombre de rangées. (Ex : **TOP 50** ou **TOP 10 PERCENT**)

\*Nous parlons d'optimisation de requêtes plus tard dans la session.



## ❖ Fonctions

- ◆ Il existe de nombreux types de fonctions mais nous allons regarder uniquement les fonctions de type **scalaire**, soit celles qui retournent une seule valeur
  - Permet de faire des calculs comprenant des données autres que celles de la table où on est.
  - Utilisable dans les clauses SELECT , WHERE, HAVING ou pour des contraintes DEFAULT ou CHECK



## ❖ Créer une Fonction

Exemple 1: calculer les ventes d'un mois et d'une année

```
GO
CREATE FUNCTION dbo.ufn_montantMois
(@year int, @month int)
RETURNS decimal(10,2)
AS
BEGIN
    DECLARE @answer decimal(10,2);

    SELECT @answer = SUM((prixVendu * qty))
    FROM DetailsAchat DA
    INNER JOIN Achat A
    ON A.AchatID = DA.AchatID
    WHERE MONTH(orderDate) = @month AND YEAR(orderDate) = @year;

    RETURN @answer;
END
GO
```

Déclaration du type  
de valeur attendue

Code qui retourne une valeur





### ❖ Dans la clause SELECT

```
SELECT dbo.ufn_montantMois(2020,2) AS 'Montant vendu en février 2020'
```



## ❖ Créer une Fonction

Exemple 2: calculer la quantité vendue pour un livre

```
-- Fonction pour compter le NbLikes qu'un livre a eu,  
--en assumant à la base que si le livre a été vendu, c'est un like.  
GO  
CREATE FUNCTION dbo.ufn_CalculerQtyVenduePourUnLivre  
(@LivreID int)  
RETURNS int  
AS  
BEGIN  
    DECLARE @QtyVendue int;  
  
    SELECT @QtyVendue = ISNULL(SUM( qty),0)  
    FROM dbo.DetailsAchat  
    WHERE LivreID= @LivreID;  
  
    RETURN @QtyVendue;  
END  
GO
```



## ❖ Dans un UPDATE pour donner une valeur à un champ

```
UPDATE dbo.Livre
SET NbLikes = dbo.ufn_CalculerQtyVenduePourUnLivre(LivreID)

SELECT 1 as 'APRES UPDATE', LivreID, Titre, NbLikes FROM dbo.Livre
```

%

Résultats Messages

APRES UPDATE	LivreID	Titre	NbLikes
1	1	Le Rouge et le Noir	1
1	2	Nuits d'encre	6
1	3	Vingt mille lieues sous les mers	4
1	4	Waylander 1	3
1	5	La Mémoire dans la peau	4
1	6	Ivanohé	2
1	7	The Lincoln Lawyer	2
1	8	The Poet	0



## ❖ Transact-SQL (ou T-SQL)

- ◆ C'est une **extension** du langage **SQL** qui offre de nombreux outils supplémentaires.
  - Depuis le début de la session nous utilisons surtout du code SQL qui est très **standardisé** pour tous les SGBD relationnels.
- ◆ **Transact-SQL** fonctionne seulement avec **Microsoft SQL Server**.
  - Chaque **SGBD** utilise une extension différente du langage **SQL**, donc même si la majorité des instructions **SQL** se ressemblent (Car c'est un langage standardisé), certaines **syntaxes** et **outils** diffèrent d'un **SGBD** à l'autre.
    - Pour cette raison, nous n'aborderons pas en détails tout ce que **Transact-SQL** propose.
    - Gardez à l'esprit que la majorité des notions abordées en lien avec **Transact-SQL** ont généralement des **équivalences** avec une **syntaxe** légèrement différente dans les autres **SGBD**, donc les concepts que nous abordons seront facilement **réutilisables**.
  - Certains outils proposés par **Transact-SQL** nous permettront de mieux exploiter les **procédures stockées**, les **déclencheurs** et les **transactions**. C'est pour cela que nous devons minimalement l'aborder.



## ❖ Déclarer une variable

Déclaration d'une variable de type **int**.

```
DECLARE @maVariable int;  
  
SELECT @maVariable = COUNT(ProduitID) FROM Produits.Produit;  
  
SELECT @maVariable AS 'Nombre de produits';
```

On affecte à **@maVariable** la valeur de **COUNT(ProduitID)** en glissant le nom de la variable dans une requête SELECT.

Nombre de produits
4


On « imprime » la valeur de **@maVariable**



## ❖ Déclarer une variable de type table

```
DECLARE @maTableTemporaire Table(  
    Produit nvarchar(30),  
    Prix numeric(12,2)  
);  
  
INSERT INTO @maTableTemporaire (Produit, Prix)  
VALUES  
    ('Chaise jaune', 15.99),  
    ('Giraffe', 24000.00);  
  
INSERT INTO @maTableTemporaire (Produit, Prix)  
SELECT Nom, Prix FROM Produits.Produit;  
  
SELECT * FROM @maTableTemporaire;
```

- Comme les variables scalaires / atomiques, ce type de table est **temporaire**.
- Malgré tout, on peut y faire des **INSERT**, des **UPDATE**, des **DELETE**, des **SELECT**, etc.



Produit	Prix
Chaise jaune	15.99
Giraffe	24000.00
Chaise	15.99
Pneu	80.49
Dentifrice	2.99
Tapis	149.99
Rhinocéros	21500.00
Piano	3999.99
Parapluie	5.99
Cendrier	3.99
Volvo 1989	399.99
Tomate	0.99



## ❖ IF, ELSE IF et ELSE

```
DECLARE @NbProduitsPasChers int;

SELECT @NbProduitsPasChers = COUNT(ProduitID)
FROM Produits.Produit
WHERE Prix <= 10;

IF @NbProduitsPasChers > 10
    BEGIN
        SELECT 'Wow c''est pas cher ici !' AS Message;
    END
ELSE IF @NbProduitsPasChers < 5
    BEGIN
        SELECT 'Ilala ça coûte cher ici.' AS Message;
    END
ELSE
    BEGIN
        SELECT 'C''est pas pire ici.' AS Message;
    END
END
```

- Remarquez **BEGIN** et **END**, qui servent à délimiter un bloc d'instructions. (Un peu comme des accolades { ... })

- N'hésitez pas à utiliser **AND**, **NOT** et **OR** dans les conditions. On peut même utiliser **EXISTS**, **ANY**, **ALL**, etc.

Nom	Prix
Chaise	15,99
Pneu	80,49
Dentifrice	2,99
Tapis	149,99
Rhinocéros	21500,00
Piano	3999,99
Parapluie	5,99
Cendrier	3,99
Volvo 1989	399,99
Tomate	0,99



Message
Ilala ça coûte cher ici.



## ❖ Outils supplémentaires

### ◆ Transact-SQL propose également ...

- Boucles WHILE
- Blocs TRY ... CATCH
- Curseurs
- etc.

### ◆ Libre à vous d'en apprendre plus sur T-SQL, mais nous n'irons pas plus loin que ce qui a été abordé dans cette section pour le cours.





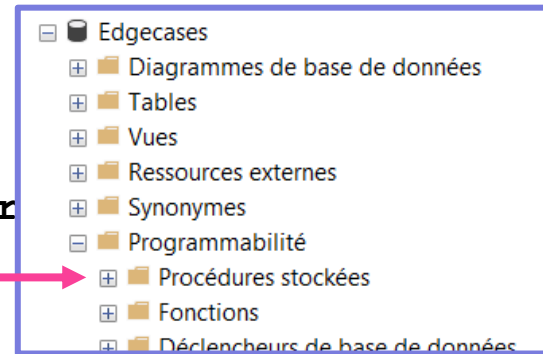
## ❖ Procédure stockée

- ◆ Ensemble d'une ou plusieurs instructions SQL encapsulées dans une « procédure » réutilisable.

- ◆ Créer une procédure stockée :

Peut contenir des SELECT, des INSERT, UPDATE, DELETE, etc.

```
GO
CREATE PROCEDURE schema.usp_nomProcédure
{
    Instruction(s) SQL
GO;
```



- ◆ Créer une procédure stockée avec un ou plusieurs paramètres :

```
CREATE PROCEDURE schema.usp_nomProcédure
@param1 type, @param2 type, ...
AS

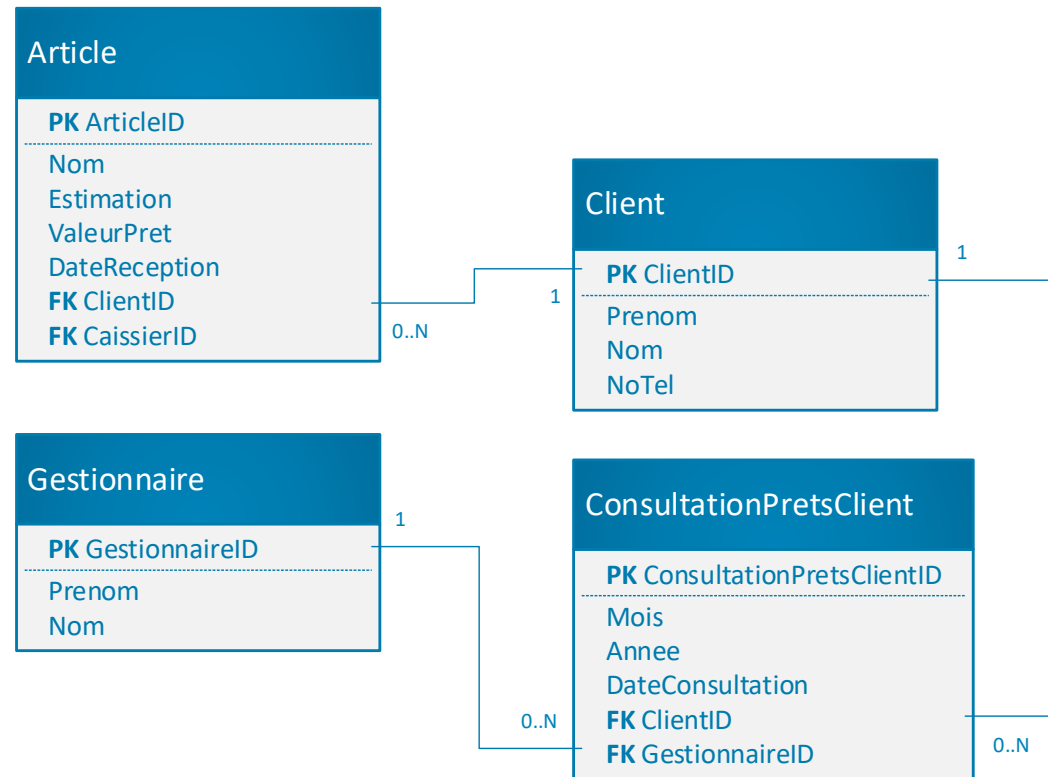
    Instruction(s) SQL

GO;
```



## ❖ Exemple #1

- ◆ Dans un *pawn shop*, les gestionnaires aimeraient pouvoir facilement consulter la liste des prêts octroyés à un **client** pour un **mois** et une **année** spécifique. Cela dit, à chaque fois qu'un gestionnaire consulte cette information, on veut noter la date de la consultation à des fins de sécurité.





## ❖ Exemple #1

```
-- Procédure stockée #1 : Consulter prêts sur gage d'un client
GO
CREATE PROCEDURE Archives.usp_ConsultationPretsClient
    @ClientID int,
    @GestionnaireID int,
    @Mois int,
    @Annee int
AS

INSERT INTO Archives.ConsultationPretsClient
(Mois, Annee, DateConsultation, ClientID, GestionnaireID)
VALUES
(@Mois, @Annee, GETDATE(), @ClientID, @GestionnaireID);

SELECT A.Nom, A.Estimation, A.ValeurPret, A.DateReception
FROM Article A
INNER JOIN Client C
ON A.ClientID = C.ClientID
WHERE MONTH(DateReception) = @Mois AND YEAR(DateReception) = @Annee;
GO
```

Données qui seront ajoutées  
dans la table  
ConsultationPretsClient

Données qui seront  
affichées par la requête  
SELECT



❖ Exemple #1

```
-- Exécution de la procédure #1

EXEC SP_ConsultationPretsClient
    @ClientID = 64,
    @GestionnaireID = 2,
    @Mois = 8,
    @Annee = 2021
```

- Le gestionnaire a obtenu la liste des prêts pour le client #64 en août 2021.
- L'id du gestionnaire, le moment de consultation et l'id du client ont été pris en note dans la base de données.

Données ajoutées dans la table ConsultationPretsClient

ConsultationPretsClient					
ConsultationPretsClientsID	Mois	Annee	DateConsultation	ClientID	GestionnaireID
4	8	2021	2021-09-14 15:17:53.422	63	2

Données affichées par la requête SELECT

Nom	Estimation	ValeurPret	DateReception
Chaise IKEA Lisabo	15	5	20210813
Piano Yamaha YDP165	2300	1500	20210817
Brouette de la 1ère guerre mondiale	750	300	20210824
Munitions carabine Lee Enfield III 1ère guerre mondiale	500	230	20210824



## ❖ Exemple #2

- ◆ Pour obtenir une liste d'achats faits entre deux dates

```
-- Faites une procédure pour obtenir la liste des achats faits entre deux dates
GO
] CREATE PROCEDURE dbo.usp_AchatIntervalle
] (@dateStart datetime2, @dateEnd datetime2)
] AS
] BEGIN
]     SELECT AchatID, cast(orderDate as date)'orderDate', nomClient + ', ' + prenomClient 'Nom acheteur' , telephone
]     FROM [dbo].[Achat] A
]     INNER JOIN [dbo].[Client] C
]     ON C.ClientID = A.ClientID
]     WHERE orderDate BETWEEN @dateStart AND @dateEnd
]
] END
] GO
```

---



## ❖ Exemple #2

- ◆ Notez l'utilisation du format international pour les dates

```
EXECUTE dbo.usp_AchatIntervalle @dateStart = '20200101' , @dateEnd = '20210101'  
GO
```

%

Résultats		Messages	
AchatID	orderDate	Nom acheteur	telephone
5	2020-01-10	Vaughn, Haley	1-244-871-9462
6	2020-08-10	Vaughn, Haley	1-244-871-9462



## ❖ Outils supplémentaires

### ◆ Supprimer une procédure stockée

```
DROP PROCEDURE schema.nom_procedure
```

### ◆ Modifier une procédure stockée existante

La seule différence est le mot-clé **ALTER** plutôt que **CREATE**. Les **paramètres** et les **instructions** spécifiés remplacent les anciens.

```
ALTER PROCEDURE schema.nom_procedure  
@param1 type, @param2 type, ...  
AS
```

```
Instruction(s) SQL
```

```
GO;
```



## ❖ Bonnes raisons pour créer une procédure stockée

### ◆ Effectuer des **requêtes complexes**

- Une procédure stockée peut encapsuler des requêtes complexes dans lesquelles il serait facile de faire des erreurs autrement. Pouvoir appeler la procédure en lui fournissant les bons paramètres peut simplifier l'interaction avec la base de données.

### ◆ Assurer l'**intégrité** des données

- Une procédure stockée peut gérer la manipulation de certaines données à la place d'une requête faite à la main pour éviter de violer leur intégrité.

### ◆ Améliorer la **sécurité**

- On peut octroyer des permissions sur l'usage des procédures plutôt que sur les tables. Cela permet d'encapsuler les requêtes associées à une table dans une procédure qui encadre ou limite l'accès aux données sensibles.

### ◆ Améliorer la **performance**

- Après avoir été utilisée par le serveur, le plan d'exécution d'une procédure est mis en cache. Elle est donc précompilée et la réutiliser est plus rapide qu'une requête conventionnelle qui serait identique. (Surtout avantageux à grande échelle)





### ❖ Différences avec une vue

- ◆ Une **procédure stockée** peut simplifier l'accès à des données qu'on obtient via une requête complexe. (Comme une **vue**) Cela dit, une procédure stockée se distingue sur plusieurs aspects :
  - Une procédure stockée peut recevoir des **paramètres**.
  - Une procédure stockée peut exécuter **plusieurs instructions** SQL et n'est pas limitée à l'usage de **SELECT**.
  - Une procédure stockée peut être **utilisée (appelée)** dans un déclencheur.