Interroger des bases de données avec le langage SQL

Propriété intellectuelle

Paris Brest Nantes Bordeaux
Toulouse Montpellier Lyon Paris

Plan

- 1. Introduction
  - Rappels sur le modèle relationnel
  - Les caractéristiques du langage SQL

- 2. Le Langage d'Interrogation de Données (LID)
  - La sélection de données
  - La gestion des valeurs null
  - Les restrictions ou conditions
  - Les tris
  - Les jointures

Plan

- 3- Utilisation des fonctions
- 4- Utilisation des opérateurs ensemblistes
  - Union
  - Intersect
  - Minus (ou Except)
- 5. Utilisation des sous interrogations
- 6. Le Langage de Manipulation de Données (LMD)
  - L'Insert
  - L'Update
  - Le Delete

Module 1

Introduction

Sommaire Module 1

Rappel sur le modèle relationnel

- **Les caractéristiques du langage SQL**

Rappel sur le modèle relationnel

Une base de données est un ensemble cohérent d'informations mémorisées sur support informatique.

Ces informations sont accessibles à l'aide d'une application appelée **système de gestion de base de données** (SGBD). Si ce SGBD est basé sur le modèle relationnel de CODD, on dit qu'il s'agit d'un système de gestion de base de données relationnel (SGBDR).

Pour dialoguer avec un SGBDR on utilise le langage SQL. Ce langage permet de soumettre des requêtes (des **questions**) **au SGBDR.**

Rappel sur le modèle relationnel

Le modèle relationnel est constitué d'un ensemble d'opérations formelles sur les relations.

Les données sont stockées dans des tables qu'on peut mettre en relation.

Une table est une relation, mais entre les différents champs qui la composent.

Rappel sur le modèle relationnel

La modélisation relationnelle permet de représenter les relations à l'aide de tables (à deux dimensions)

Une ligne d'une table représente donc une entité.

Un attribut est le nom des colonnes qui constitue la définition d'une table. Il comporte un typage de données.

On appelle tuple (ou n-uplet) une ligne de la table.

Rappel sur le modèle relationnel

La cardinalité d'une relation est le nombre de tuples qui la composent.

La clé principale (ou primaire) d'une relation est l'attribut, ou l'ensemble d'attributs, permettant de désigner de façon unique un tuple.

Une clé étrangère, par contre, est une clé faisant référence à une clé appartenant à une autre table.

Rappel sur le modèle relationnel

Cas pratique :
La base de données du stage

Rappel sur le modèle relationnel

Caractéristiques :

- Indépendance physique : le niveau physique peut être modifié indépendamment du niveau conceptuel. Cela signifie que tous les aspects matériels de la base de données n'apparaissent pas pour l'utilisateur, il s'agit simplement d'une structure transparente de représentation des informations

- Indépendance logique : le niveau conceptuel doit pouvoir être modifié sans remettre en cause le niveau physique, c'est-à-dire que l'administrateur de la base doit pouvoir la faire évoluer sans que cela gêne les utilisateurs

Rappel sur le modèle relationnel

Caractéristiques :

- Manipulabilité : des personnes ne connaissant pas la base de données doivent être capables de décrire leur requête sans faire référence à des éléments techniques de la base de données
- Rapidité des accès : le système doit pouvoir fournir les réponses aux requêtes le plus rapidement possible, cela implique des algorithmes de recherche rapides
- Administration centralisée : le SGBD doit permettre à l'administrateur de pouvoir manipuler les données, insérer des éléments, vérifier son intégrité de façon centralisée

Rappel sur le modèle relationnel

Caractéristiques :

- Limitation de la redondance : le SGBD doit pouvoir éviter dans la mesure du possible des informations redondantes, afin d'éviter d'une part un gaspillage d'espace mémoire mais aussi des erreurs

- Vérification de l'intégrité : les données doivent être cohérentes entre elles, de plus lorsque des éléments font référence à d'autres, ces derniers doivent être présents

Rappel sur le modèle relationnel

Caractéristiques :

- Partageabilité des données : le SGBD doit permettre l'accès simultané à la base de données par plusieurs utilisateurs

- Sécurité des données : le SGBD doit présenter des mécanismes permettant de gérer les droits d'accès aux données selon les utilisateurs

Rappel sur le modèle relationnel

Une requête est un ordre adressé à un SGBD. Cet ordre peut consister à extraire, à ajouter, à modifier, à administrer les données de la base. De façon générale, l'utilisateur, comme l'administrateur, dialogue avec le SGBD en lui soumettant des requêtes (des questions) et en récupérant en retour des résultats (les réponses).

Sommaire Module 1

Rappel sur le modèle relationnel

- **Les caractéristiques du langage SQL**

Les caractéristiques du langage SQL

Le langage SQL est devenu le standard en matière d'interface relationnelle, ceci probablement à cause des raisons suivantes :

- issu de SEQUEL (interface de System-R), SQL a été développé chez IBM à San José ... !
- basé sur des mots clefs anglais explicites, il est relativement simple et facile à apprendre pour des utilisateurs non-informaticiens. Il illustre bien la tendance des langages formels à s'orienter vers un certain "langage naturel".
- SQL est un langage normalisé

Les caractéristiques du langage SQL

Le standard Ansi a valeur nominative, en principe seulement aux Etats-Unis.

L'équivalent français est la norme Afnor. La norme internationale de SQL est la norme ISO (International Standards Organisation) numéro 9075 de 1987.

Les normes sont accompagnées de niveau qui indiquent le degré d'évolution de SQL. Ainsi l'ISO a définit les normes et les niveaux suivants :

- SQL89
- SQL92 (ou SQL2)
  - Entry
  - Transitional
  - Intermediate
  - Full
- SQL 1999 (ou SQL3)
- SQL 2003
- SQL 2006

Les caractéristiques du langage SQL

La norme définit deux langages SQL :

- un Langage de Manipulation de Données et de modules, (en anglais *SQLDML)*,pour déclarer les procédures d'exploitation et les appels à utiliser dans les programmes. On peut également rajouter une composante pour l'interrogation de la base : Langage d'Interrogation de Données.

- un Langage de Définition de Données (en anglais *SQL-DDL)*, à utiliser pour déclarer les structures logiques de données et leurs contraintes d'intégrité ;On peut également rajouter une composante pour la gestion des accès aux données : Langage de Contrôle de Données : (en anglais SQL-DCL)

Les caractéristiques du langage SQL

Les instructions incontournables de SQL sont les suivantes :

Langage SQL			
LMD		LDD	
LID		LCD	
SELECT	INSERT	GRANT	CREATE
	UPDATE	REVOKE	ALTER
	DELETE		TRUNCATE
			DROP
			RENAME

Module 2

Le Langage d'Interrogation de Données (LID)

Sommaire Module 2

- **La Sélection de données**
- Gestion des valeurs null
- Les restrictions ou conditions
- Les tris
- Les jointures

La commande SELECT

Le SELECT est la commande de base du SQL.

Elle permet d'extraire des données d'une base ou ,
d'en calculer de nouvelles à partir de données existantes.

La commande SELECT

SELECT [DISTINCT ou ALL] * ou liste de colonnes
FROM nom de table ou de la vue
[WHERE prédicats]
[GROUP BY ordre des groupes]
[HAVING condition]
[ORDER BY liste de colonnes*]*

La commande SELECT

SELECT
Spécification des colonnes du résultat
FROM
Spécification des tables sur lesquelles porte le select
WHERE Filtre portant sur les données (conditions à remplir pour faire afficher le résultat)
GROUP BY
Définition du sous ensemble
HAVING
conditions de regroupement des lignes
ORDER BY
Tri des données du résultat

La commande SELECT

Requête:

SELECT CLI_NOM, CLI_PRENOM FROM T_CLIENT

Résultat :

CLI_NOM	CLI_PRENOM
DUPONT	Alain
MARTIN	Marc
BOUVIER	Alain
DUBOIS	Paul
DREYFUS	Jean
FAURE	Alain
PAUL	Marcel
DUVAL	Arsène
PHILIPPE	André
CHABAUD	Daniel
BAILLY	Jean-François

...

La commande SELECT

Il existe plusieurs opérateur de sélection:

- Le caractère * récupère toutes les colonnes
- DISTINCT qui permet d'éliminer les doublons dans la réponse
- AS sert à donner un nom à de nouvelles colonnes créées par la requête
- L'opérateur || (double barre verticale) permet de concaténer des champs de type caractères (parfois c'est +)
- On, peut utiliser les opérateurs mathématiques de base pour combiner différentes colonnes (+,-, *, /)

Sommaire Module 2

- La Sélection de données
- **Gestion des valeurs null**
- Les restrictions ou conditions
- Les tris
- Les jointures

La gestion des valeurs null

La gestion du null sous SqlServer

ISNULL(check_expr, replacement_value)

ISNULL remplace NULL avec la valeur spécifiée replacement_value. La fonction retourne la valeur check_expr si celle-ci n’est pas NULL, sinon ,elle retourne replacement_value.

La gestion des valeurs null

Exemple

Ici si le prix est null on affiche 0.00

SELECT title, ISNULL(price, 0.00) AS price

La gestion des valeurs null

La gestion du null sous Oracle

NVL(val1, replace_with)

val1 est la valeur testé pour le null.
replace_with est la valeur qui est retournée si val1 est null.

Cela s’applique à:
Oracle 8i, Oracle 9i, Oracle 10g, Oracle 11g

La gestion des valeurs null

Exemple :

Ici si la ville du fournisseur n’est null on affiche ‘n/a’

```
select NVL(supplier_city, 'n/a')
from suppliers;
```

Atelier 2.1

Exercices :

1. Affichez tous les employés de la société
2. Affichez toutes les catégories de produits
3. Affichez les noms, prénoms et date de naissance, et leur commission (à 0 si pas de commission) de tous les employées de la société
4. Affichez la liste des fonctions des employés de la société
5. Affichez la liste des pays de nos clients
6. Affichez la liste des localités dans lesquelles il existe au moins un client.

Atelier 2.2

Exercices:

1. Affichez les produits commercialisés et la valeur de stock par produit (prix unitaire *quantité en stock)
2. Affichez le nom, le prénom, l'âge et l'ancienneté des employés, dans la société.
3. Écrivez la requête qui permet d'afficher:

Employé	a un	gain annuel	sur 12 mois
----------	----------	----------	----------
Fuller	gagne	120000	par an.

……

Sommaire Module 2

- La Sélection de données
- Gestion des valeurs null
- **Les restrictions ou conditions**
- Les tris
- Les jointures

La clause WHERE

Affichage des lignes vérifiant une condition

Syntaxe :

SELECT ....

WHERE *condition*

Ex: Select * from client where numclient=123;

La clause WHERE

Dans la clause Where ou de condition, on peut utiliser tout les opérateurs logiques.

Expression op Expression

Op peut être:

- =
- <, <=
- >, >=
- !=, <>,^=

La clause WHERE

Il existe d'autres opérateurs:

- Opérateur IN
  - WHERE TIT_CODE IN ('Mme.', 'Melle.')
- Opérateur BETWEEN
  - WHERE CODE_POSTAL BETWEEN 91000 AND 92000
- Opérateur LIKE
  - WHERE CLI_NOM LIKE 'B%'
- Comparaison logique
  - IS [NOT] {TRUE | FALSE | UNKNOWN} / IS [NOT] NULL
- Connecteurs logiques
  - {OR | AND}

Atelier 2.3

Exercices :

1. Affichez les nom de la société et le pays des clients qui habitent à Toulouse.
2. Affichez les nom, prénom et fonction des employés dirigés par l'employé numéro 2.
3. Affichez les nom, prénom et fonction des employés qui ne sont pas des représentants.
4. Affichez les nom, le prénom et fonction des employés qui ont un salaire inférieur à 3500
5. Affichez le nom de la société, la ville et le pays des clients qui n'ont pas de fax.
6. Affichez les nom, prénom et la fonction des employés qui n'ont pas de supérieur.

Sommaire Module 2

- La Sélection de données
- Gestion des valeurs null
- Les restrictions ou conditions
- **Les tris**
- Les jointures

La clause ORDER BY

Cette clause permet de définir le tri des colonnes

ASC spécifie l’ordre ascendant et DESC l’ordre descendant du tri. ASC ou DESC peut être omis, dans ce cas c'est l'ordre ascendant qui est utilisé par défaut.

Attention : le tri est un tri interne, il ne faut donc placer dans cette clause que les noms des colonnes dans la clause SELECT.

La clause ORDER BY

```
SELECT CLI_NOM, CLI_PRENOM
FROM T_CLIENT
ORDER BY CLI_NOM, CLI_PRENOM
```

ou

```
SELECT CLI_NOM, CLI_PRENOM
FROM T_CLIENT
ORDER BY 1, 2
```

Atelier 2.4

Exercices :

1. Trier les employés par nom de salarié en ordre décroissant
2. Trier les clients par pays
3. Trier les clients par pays et par ville

Sommaire Module 2

- La Sélection de données
- Gestion des valeurs null
- Les restrictions ou conditions
- Les tris
- **Les jointures**

Jointures

La jointure entre deux tables

Soient deux tables *table1* et *table2*. *table1* a les colonnes *col1* et *col2* et *table2* les colonnes *cola*, *colb*.

Supposons q

table1	col1	col2
	x	3
	y	4

table2	cola	colb
	a	7
	b	4

Soit la commande :

SELECT *col1, cola* FROM *table1, table2*

WHERE *table1.col2=table2.colb*

Cette requête extrait des données de deux tables : *table1* et *table2* avec une condition entre deux colonnes de tables différentes. On appelle ce type de requête, une jointure.

Jointures

Comment fonctionne-t-elle ?

Une nouvelle table est construite avec pour colonnes, l'ensemble des colonnes des deux tables et pour :

col1	col2	cola	colb
x	3	a	7
x	3	b	4
y	4	a	7
y	4	b	4

La condition *WHERE col2=colb* est appliquée à cette nouvelle table. On obtient donc la nouvelle table suivante :

col1	col2	cola	colb
y	4	b	4

Jointures

Il y a ensuite affichage des colonnes demandées (select):

col1	cola
y	b

Jointures

Syntaxe d'une requête multi-tables

syntaxe SELECT *colonne1, colonne2, ...*

FROM *table1, table2, ..., tablep*

WHERE *condition*

ORDER BY ...

La nouveauté ici vient du fait que les colonnes *colonne1, colonne2, ...* proviennent de plusieurs tables *table1, table2*, ... Si deux tables ont des colonnes de même nom, on lève l'ambiguïté par

la notation *tablei.colonnej*. La *condition* peut porter sur les colonnes des différentes tables.

Jointures

Fonctionnement:

1. La table produit cartésien de *table1, table2, ..., tablep* est réalisée. Si *ni* est le nombre de lignes de *tablei*, la table construite a donc *n1*n2*...*np* lignes comportant l'ensemble des colonnes des différentes tables.

2. La *condition* du WHERE est appliquée à cette table. Une nouvelle table est ainsi produite

3. Celle-ci est ordonnée selon le mode indiqué dans ORDER.

4. Les colonnes demandées derrière SELECT sont affichées.

Jointures

Sans condition (produit cartésien)

```
SELECT CLI_NOM, TEL_NUMERO
FROM T_CLIENT, T_TELEPHONE
```

Avec condition

```
SELECT CLI_NOM, TEL_NUMERO
FROM T_CLIENT, T_TELEPHONE
WHERE T_CLIENT.CLI_ID = T_TELEPHONE.CLI_ID
```

```
SELECT CLI_NOM, TEL_NUMERO
FROM T_CLIENT C, T_TELEPHONE T
WHERE C.CLI_ID = T.CLI_ID AND TYP_CODE = 'FAX'
```

Jointures

Jointure naturelle	SELECT ... FROM <table gauche> NATURAL JOIN <table droite> [USING <noms de colonnes>]

SELECT CLI_NOM, TEL_NUMERO

FROM T_CLIENT NATURAL JOIN T_TELEPHONE

La jointure se fait sur la colonne qui porte le même nom dans les deux tables

Jointures

Jointure interne	SELECT ... FROM <table gauche> [INNER]JOIN <table droite> ON <condition de jointure>

```
SELECT CLI_NOM, TEL_NUMERO
FROM T_CLIENT INNER JOIN T_TELEPHONE
ON T_CLIENT.CLI_ID = T_TELEPHONE.CLI_ID
```

Ici on spécifie le nom de la colonne sur lequel faire la jointure.

Jointures

Jointure externe	SELECT ... FROM <table gauche> LEFT | RIGHT | FULL OUTER JOIN <table droite> ON condition de jointure

SELECT CLI_NOM, TEL_NUMERO

FROM T_CLIENT C LEFT OUTER JOIN T_TELEPHONE T

ON C.CLI_ID = T.CLI_ID AND TYP_CODE IS NULL

Les jointures externes rapatrient les informations disponibles, même si des lignes de table ne sont pas renseignées entre les différentes tables jointes

Jointures

SELECT colonnes

FROM TGauche LEFT OUTER JOIN TDroite

ON condition de jointure

On recherche toutes les valeurs satisfaisant la condition de jointure précisée dans prédicat, puis on rajoute toutes les lignes de la table TGauche qui n'ont pas été prises en compte au titre de la satisfaction du critère.

Jointures

SELECT colonnes

FROM TGauche RIGHT OUTER JOIN TDroite

ON condition de jointure

On recherche toutes les valeurs satisfaisant la condition de jointure précisée dans prédicat, puis on rajoute toutes les lignes de la table TDroite qui n'ont pas été prises en compte au titre de la satisfaction du critère.

Jointures

SELECT colonnes

FROM TGauche FULL OUTER JOIN TDroite

ON condition de jointure

On recherche toutes les valeurs satisfaisant la condition de jointure précisée dans prédicat, puis on rajoute toutes les lignes de la table TGauche et TDroite qui n'ont pas été prises en compte au titre de la satisfaction du critère.

Jointures

Jointure croisée	SELECT ... FROM <table gauche> CROSS JOIN <table droite>

```
SELECT CHB_ID, TRF_DATE_DEBUT, 0 AS TRF_CHB_PRIX
FROM T_TARIF CROSS JOIN T_CHAMBRE
ORDER BY CHB_ID, TRF_DATE_DEBUT
```

On fait sciemment le produit cartésien .

Atelier 2.5

Exercices :

1. Affichez le nom, prénom, fonction et salaire des employés qui ont un salaire compris entre 2500 et 3500
2. Affichez le nom du produit, le nom du fournisseur, le nom de la catégorie et les quantités de produits qui ne sont pas d'une des catégories 1,3,5 et 7.
3. Affichez le nom du produit, le nom du fournisseur, le nom de la catégorie et les quantités des produits qui ont le numéro fournisseur entre 1 et 3 ou un code catégorie entre 1 et 3, et pour lesquelles les quantités sont données en boîtes ou en cartons.

Atelier 2.5

Exercices :

4. Écrivez la requête qui permet d'afficher le nom des employés qui ont effectué au moins une vente pour un client parisien.
5. Affichez le nom des produits et le nom des fournisseurs pour les produits des catégories 1,4 et 7.
6. Affichez la liste des employés ainsi que le nom de leur supérieur hiérarchique.

Sommaire Module 3

- **Utilisation de fonctions**

Fonctions

Transtypage :

Il permet de changer le type de données d'une colonne afin d’effectuer une comparaison de données de type hétérogène .

```
SELECT CHB_ID, CHB_NUMERO, CHB_POSTE_TEL
FROM T_CHAMBRE
WHERE
CAST(CHB_POSTE_TEL AS INTEGER) / 10 > CHB_NUMERO
```

Fonctions

Mise en majuscule / Minuscule :

Les opérateurs LOWER et UPPER permettent de mettre en majuscule ou en minuscule des chaînes de caractères dans les requêtes

SELECT upper(CLI_PRENOM), lower(CLI_NOM)

FROM T_CLIENT

CLI_NOM	CLI_PRENOM
ALAIN	dupont
MARC	martin
ALAIN	bouvier

Fonctions

Extraire une sous chaîne :

La fonction SUBSTRING (SUBSTR sous Oracle) permet d'extraire une sous chaîne d'une chaîne de caractère

SELECT CLI_NOM, CLI_PRENOM, SUBSTRING(CLI_PRENOM,1,1) + SUBSTRING (CLI_NOM,1,1) AS INITIALES FROM T_CLIENT

CLI_NOM	CLI_PRENOM	INITIALES
DUPONT	Alain	AD
MARTIN	Marc	MM
……		

Fonctions

Heure et date courante

```
SELECT distinct CHB_ID
FROM CHB_PLN_CLI WHERE
PLN_JOUR BETWEEN CURRENT_DATE
and CURRENT_DATE + 14
```

Oracle	SYSDATE CURRENT_DATE
Sybase	GETDATE()
SQL Server	GETDATE()
Access	NOW() Date()
MySQL	NOW() CURRENT_DATE
Paradox (QBE)	TODAY

Fonctions

Fonctions statistiques :

```
SELECT AVG(TRF_CHB_PRIX) as MOYENNE, MAX(TRF_CHB_PRIX) as MAXI,
MIN(TRF_CHB_PRIX) as MINI,
SUM(TRF_CHB_PRIX) as TOTAL, COUNT(TRF_CHB_PRIX) as NOMBRE
FROM TJ_TRF_CHB
WHERE TRF_DATE_DEBUT = '2001-01-01'
```

ABS	Valeur absolue
MOD	Modulo
SIGN	Signe
SQRT	Racine carrée
CEIL (SQL Server)	Plus grand entier
FLOOR	Plus petit entier
ROUND	Arrondi
TRUNC (SQL Server convert)	Tronqué
EXP	Exponentielle
LN	Logarithme népérien
LOG	Logarithme décimal
POWER	Puissance
COS	Cosinus
COSH	Cosinus hyperbolique
SIN	Sinus
SINH	Sinus Hyperbolique
TAN	tangente
TANH	Tangente hyperbolique
PI	Constante pi

Fonctions

Autres opérateurs de traitement des chaînes de caractères (non normalisés)

CONCAT (Mysql ; Oracle)	concaténation : utiliser de préférence || chez Oracle. + chez Sql Server
INITCAP (oracle)	initiales en lettres capitales
LPAD (Mysql ; Oracle)	complément ou troncature à n position à gauche
RPAD (Mysql ; Oracle)	complément ou troncature à n position à droite
LTRIM / RTRIM	suppression des espaces en tête/queue d'une chaîne
REPLACE	remplacement
SOUNDEX	code de consonance – Attention : phonétique souvent anglaise
INSTR(Mysql ; Oracle)	Position d'une chaîne dans une sous chaîne
PATINDEX (Sql Server)	Position d'une chaîne dans une sous chaîne

Fonctions

Autres opérateurs de traitement des chaînes de caractères (non normalisés)

LENGTH (Mysql ; Oracle)	longueur de la chaîne
LEN (Sql Server)	longueur de la chaîne
ASCII	code ASCII d'un caractère
CHR (Oracle)	caractère dont le code ASCII est donné
CHAR (Mysql et Sql Server)	caractère dont le code ASCII est donné
REVERSE	Inverse l'ordre des caractères d'une chaîne
FLIP	Pivote les parties droite et gauche d'une chaîne par rapport au n° du caractère servant de pivot.

Fonctions

Autres opérateurs sur les valeurs temporelles (non normalisés)

+ INTERVAL ‘1’ day (Oracle, Mysql)	ajoute des jours, des mois, des années à une date
DATEADD (day,1,champ) (Sql Server)	ajoute des jours, des mois, des années à une date
ADD_MONTHS (Oracle)	ajoute des mois à une date
NEXT_DAY (Oracle)	date du prochain jour d’un nom donné
LAST_DAY (Oracle, Mysql)	renvoie le n° du dernier jour d’un mois d’une date
MONTHS_BETWEEN (Oracle)	nombre de mois entre deux dates
DATEDIFF (Sql Server)	différence entre deux dates
DATEPART (Sql Server)	Renvoie la valeur du jour, mois ou année
EXTRACT (Oracle, Mysql)	Renvoie la valeur du jour, mois ou année

Fonctions

Autres opérateurs sur les valeurs temporelles (non normalisés)

TO_CHAR (Oracle)	date sous forme littérale – Attention : souvent en anglais
TO_DATE (Oracle)	Convertit une chaîne de caractère en date
Cast (… as datetime) (Sql Server)	Convertit une chaîne de caractère en date

La clause GROUP BY

La clause GROUP BY est nécessaire dès que l'on utilise des fonctions de calculs statistiques avec des données brutes . Cette clause groupe les lignes sélectionnées en se basant sur la valeur de colonnes spécifiées pour chaque ligne et renvoie une seule ligne par groupe

La clause GROUP BY

Sans group by :

SELECT COUNT(CHB_ID) AS NOMBRE, CHB_ETAGE FROM T_CHAMBRE

NOMBRE CHB_ETAGE

---------- ---------

1 RDC

1 RDC

1 RDC

1 RDC

1 1er

1 1er

1 1er

…….

La clause GROUP BY

Avec Group By

SELECT COUNT(CHB_ID) AS NOMBRE, CHB_ETAGE FROM T_CHAMBRE

GROUP BY CHB_ETAGE

NOMBRE CHB_ETAGE

---------- ---------

8 1er

8 2e

4 RDC

La clause HAVING

La clause HAVING remplace le WHERE sur les opérations résultant des regroupements

```
SELECT SUM(CHB_COUCHAGE) AS NOMBRE,
CHB_ETAGE
FROM T_CHAMBRE
GROUP BY CHB_ETAGE
HAVING SUM(CHB_COUCHAGE) >= 20
```

```
NOMBRE CHB_ETAGE
---------- ---------
23         1er
22         2e
```

Atelier 3.1

Exercices :

1. Affichez la somme des salaires et des commissions des employés en gérant les valeurs null.
2. Affichez la moyenne des salaires et des commissions des employés.
3. Affichez le salaire maximum et la plus petite commission des employés.
4. Affichez le nombre distinct de fonction.

Atelier 3.2

Exercices :

1. Écrivez la requête qui permet d'afficher la masse salariale des employés par fonction.
2. Affichez le total des commandes, pour les commandes qui comportent plus de 5 références de produit.
3. Afficher la valeur des produits en stock et la valeur des produits commandés par fournisseur, pour les fournisseurs qui ont un numéro compris entre 3 et 6.

Module 4

Opérateurs ensemblistes

Les opérateurs ensemblistes

Les opérations ensemblistes en SQL, sont celles définies dans l'algèbre relationnelle. Elles sont réalisées grâce aux opérateurs:

- UNION
- INTERSECT (ne fait pas partie de la norme SQL et n'est donc pas implémenté dans tous les SGBD)
- EXCEPT (ne fait pas partie de la norme SQL et n'est donc pas implémenté dans tous les SGBD)

Ces opérateurs s'utilisent entre deux clauses *SELECT*.

Les opérateurs ensemblistes

L'opérateur UNION :

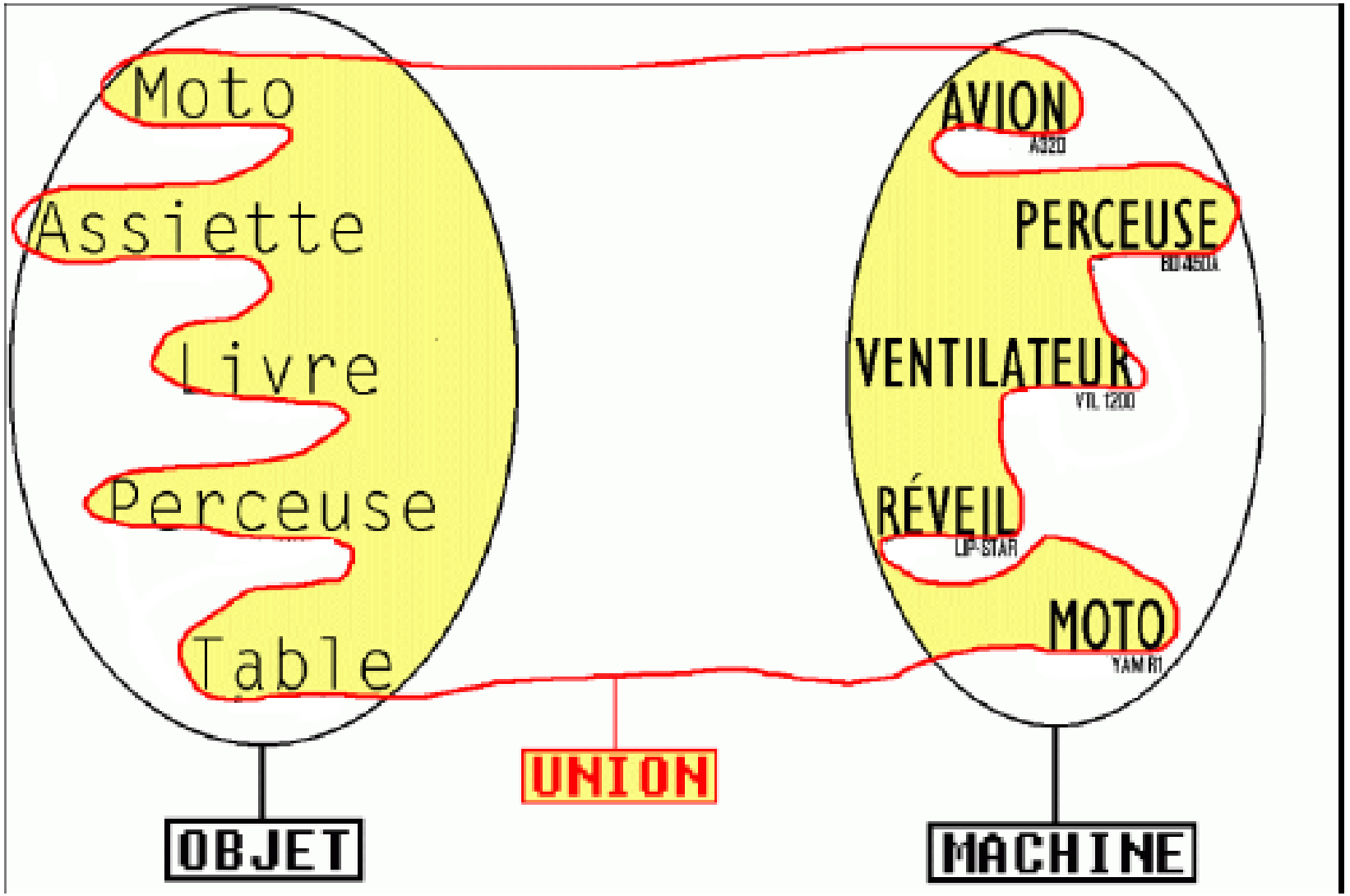
Cet opérateur permet d'effectuer une UNION des tuples sélectionnés par deux clauses *SELECT* (les deux tables sur lesquelles on travaille devant avoir le même schéma).

SELECT ---- FROM ---- WHERE ------ UNION

SELECT ---- FROM ---- WHERE ------

Par défaut les doublons sont automatiquement éliminés. Pour conserver les doublons, il est **possible d'utiliser une clause *UNION ALL***.

Opérateur union



Les opérateurs ensemblistes

L'opérateur INTERSECT :

Cet opérateur permet d'effectuer une INTERSECTION des enregistrements sélectionnés par deux clauses *SELECT* (les deux tables sur lesquelles on travaille devant avoir le même schéma).

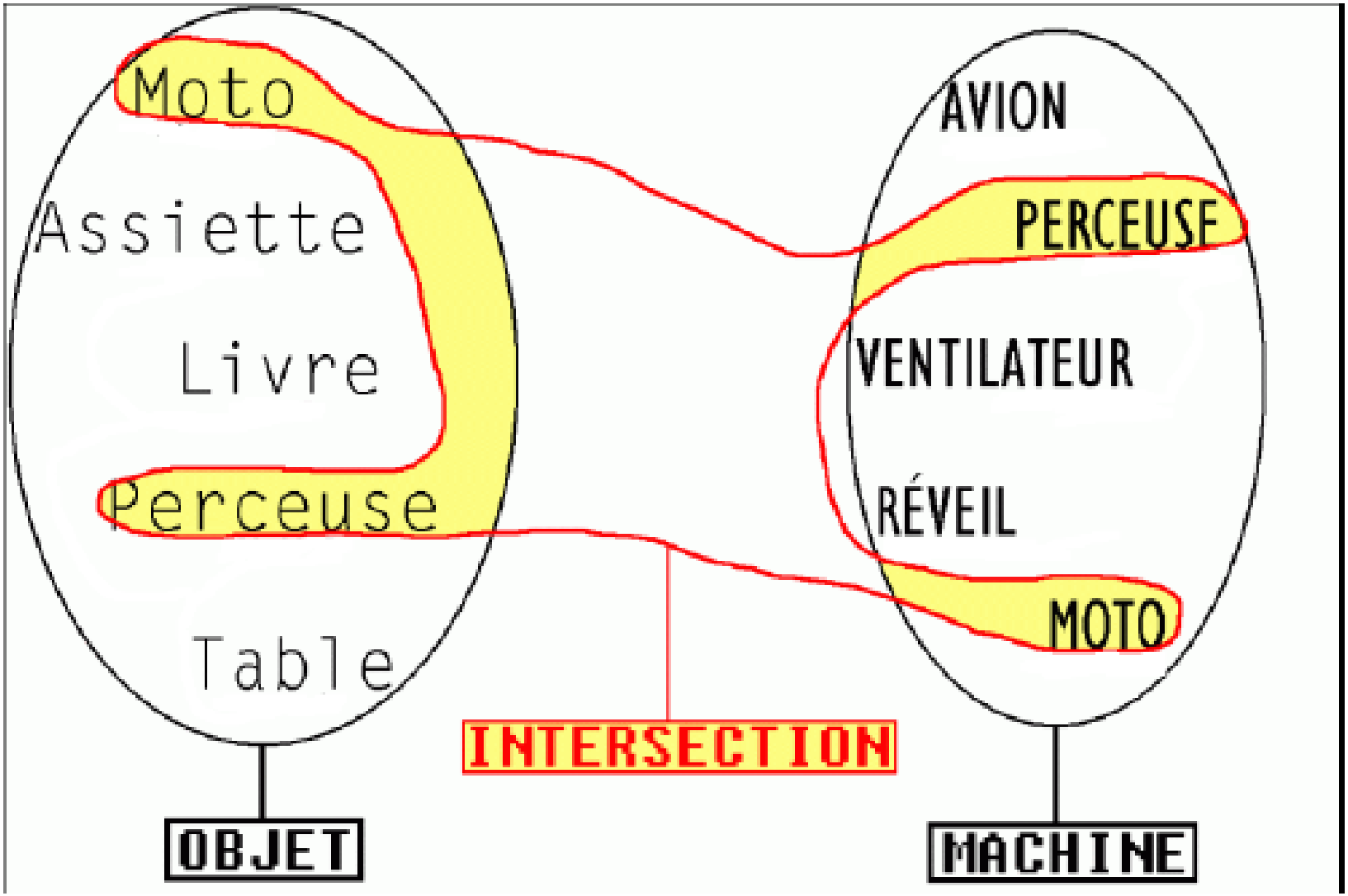
SELECT ---- FROM ---- WHERE ------ INTERSECT

SELECT ---- FROM ---- WHERE ------

L'opérateur *INTERSECT* n'étant pas implémenté dans tous les SGBD, il est possible de le remplacer par des commandes usuelles:

SELECT a,b FROM table1 WHERE EXISTS (SELECT c,d FROM table2 WHERE a=c AND b=d)

Opérateur Intersect



Les opérateurs ensemblistes

L'opérateur MINUS (ORACLE) ou EXCEPT (SQL SERVER) :

Cet opérateur permet d'effectuer une DIFFERENCE entre les enregistrements sélectionnés par deux clauses SELECT, c'est-à-dire sélectionner les enregistrements de la première table n'appartenant pas à la seconde.

SELECT a,b FROM table1 WHERE ------

MINUS

SELECT c,d FROM table2 WHERE ------

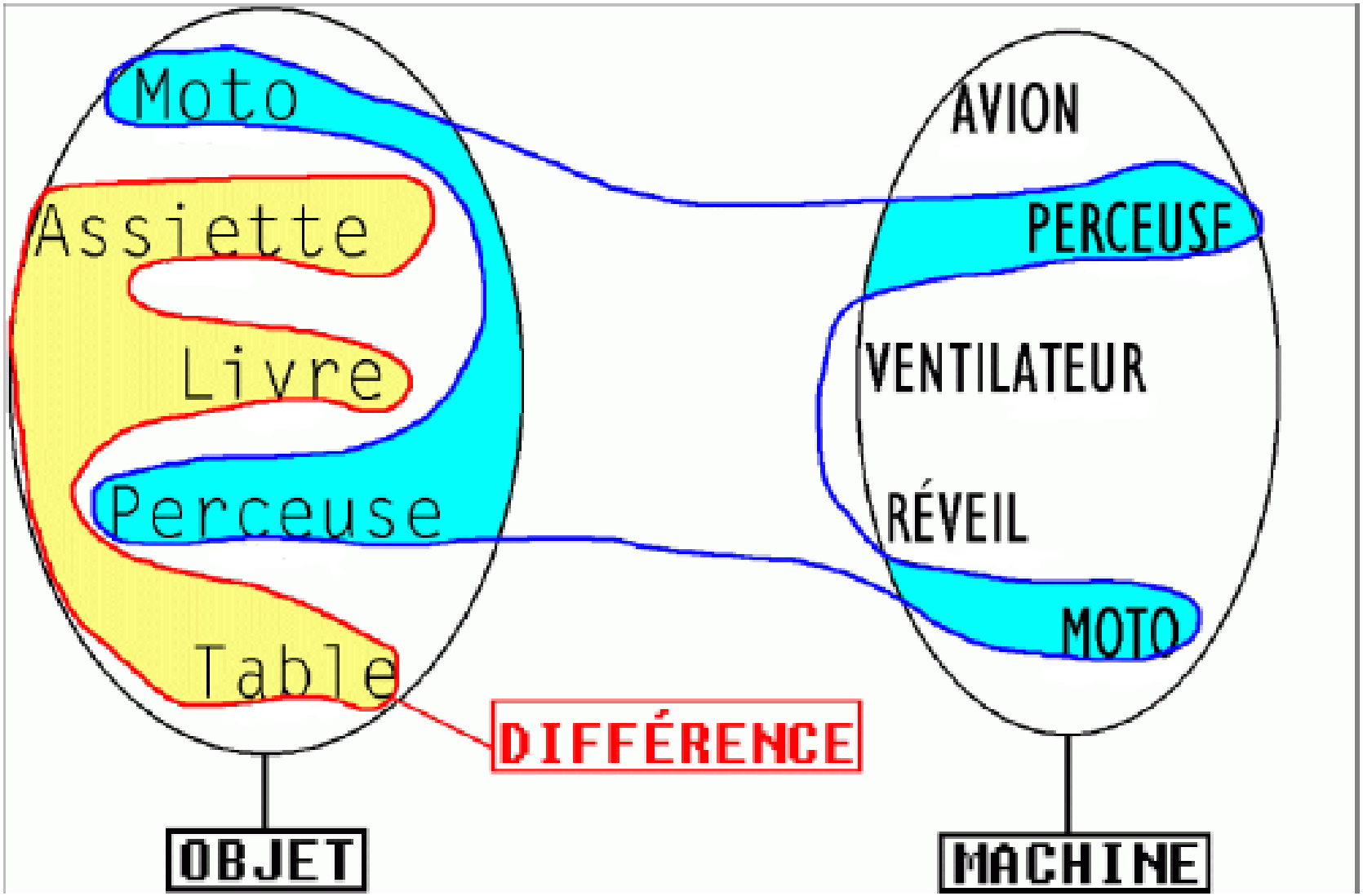
L'opérateur MINUS n'étant pas implémenté dans tous les SGBD, il est possible de le remplacer par des commandes usuelles:

SELECT a,b FROM table1

WHERE NOT EXISTS (SELECT c,d FROM table2

WHERE a=c AND b=d)

Opérateur Minus (ou Except)



Atelier 4

Exercices :

1. Affichez les sociétés, adresse et villes de résidence pour tous les tiers de l'entreprise.
2. Affichez toutes les commandes qui comportent en même temps des produits de catégorie 1 du fournisseur 1 et des produits de catégorie 2 du fournisseur 2.
3. Affichez la liste des produits que les clients parisiens ne commandent pas.

Module 5

Sous interrogations

Sommaire Module 5

- **Dans la clause WHERE**
- Dans la clause FROM
- Sous-interrogations synchronisées

Syntaxe

Une caractéristique puissante de SQL est la possibilité qu'un prédicat employé dans une clause WHERE (expression à droite d'un opérateur de comparaison) comporte un SELECT emboîté.

Sous-interrogation à une ligne et une colonne

Dans ce cas, le SELECT imbriqué équivaut à une valeur.

WHERE exp op (SELECT ...)

où op est un des opérateurs : =, !=, <>, <, >, <=, >=

Syntaxe

Exemple :

Liste des employés travaillant dans le même département que MERCIER :

```
SELECT NOME FROM EMP
WHERE DEPT = (SELECT DEPT FROM EMP
WHERE NOME = 'MERCIER')
```

Syntaxe

Sous-interrogation ramenant plusieurs lignes

Une sous-interrogation peut ramener plusieurs lignes à condition que l'opérateur de comparaison admette à sa droite un ensemble de valeurs.

Les opérateurs permettant de comparer une valeur à un ensemble de valeurs sont :

- l'opérateur IN
- les opérateurs obtenus en ajoutant ANY ou ALL à la suite des opérateurs de comparaison classique =, <>, <, >, <=, >=
  - ANY : la comparaison sera vraie si elle est vraie pour au moins un élément de l'ensemble (elle est donc fausse si l'ensemble est vide).
  - ALL : la comparaison sera vraie si elle est vraie pour tous les éléments de l'ensemble (elle est vraie si l'ensemble est vide).

Syntaxe

- WHERE exp op ANY (SELECT ...)
- WHERE exp op ALL (SELECT ...)
- WHERE exp IN (SELECT ...)
- WHERE exp NOT IN (SELECT ...)

où op est un des opérateurs =, !=, <>, <, >, <=, >=

Liste des employés gagnant plus que tous les employés du département 30 :

```
SELECT NOME, SAL FROM EMP
WHERE SAL > ALL (SELECT SAL FROM EMP
WHERE DEPT=30)
```

Sommaire Module 5

- Dans la clause WHERE
- **Dans la clause FROM**
- Sous-interrogations synchronisées

Dans la clause From

- Syntaxe :
  - Select A.x , A.y, B.z
  - from table A , (select x, z from table) B
  - Where A.x = B.x

Dans la clause From

Exemple : part du salaire de chaque employé par rapport à la masse salariale

```
Select nom, salaire, round(salaire/masse*100)
from employes, (select sum(salaire) masse from employes) b;
```

Sommaire Module 5

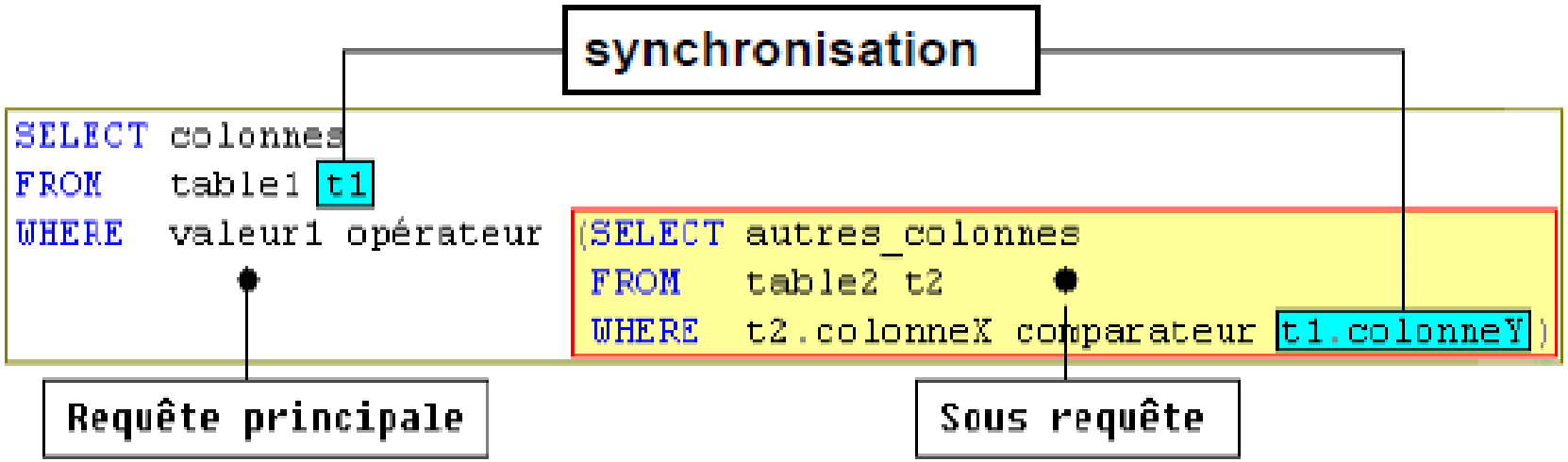
- Dans la clause WHERE
- Dans la clause FROM
- **Sous-interrogations synchronisées**

Sous requête synchronisées

- Une sous requête synchronisée est une sous requête qui s'exécute pour chaque ligne de la requête principale et non une fois pour toute.

- Pour arriver à ce résultat, il suffit de faire varier une condition en rappelant dans la sous requête la valeur de la ou des colonne de la requête principale qui doit servir de condition.

Sous requête synchronisées



Atelier 5

Exercices :

1. Affichez tous les produits pour lesquels la quantité en stock est inférieur à la moyenne des quantités en stock

2. Affichez toutes les commandes pour lesquelles les frais de ports dépassent la moyenne des frais de ports pour ce client.

3. Affichez les produits pour lesquels la quantité en stock est supérieure à la quantité en stock de chacun des produits de catégorie 3.

Atelier 5

Exercices :

4. Affichez les produits, fournisseurs et unités en stock pour les produits qui ont un stock inférieur à la moyenne des stocks des produits pour le même fournisseur.

5. Affichez les employés avec leur salaire et le % par rapport au total de la masse salariale par fonction.

Module 6

Le Langage de Manipulation de Données (LMD)

LMD

Le langage de manipulation de données (LMD) est le langage permettant de modifier les informations contenues dans la base.

Il existe trois commandes SQL permettant d'effectuer les trois types de modification des données :

- INSERT = ajout de lignes
- UPDATE = mise à jour de lignes
- DELETE = suppression de lignes

Ces trois commandes travaillent sur la base telle qu'elle était au début de l'exécution de la commande. Les modifications effectuées par les autres utilisateurs entre le début et la fin de l'exécution ne sont pas prises en compte (même pour les transactions validées).

Insertion

INSERT INTO table (col1,..., coln)

VALUES (val1,...,valn)

ou

INSERT INTO table (col1,..., coln)

SELECT ...

table est le nom de la table sur laquelle porte l'insertion.
col1,...,coln est la liste des noms des colonnes pour lesquelles on donne une valeur. Cette liste est optionnelle. Si elle est omise, le SGBD prendra par défaut l'ensemble des colonnes de la table dans l'ordre où elles ont été données lors de la création de la table. Si une liste de colonnes est spécifiée, les colonnes ne figurant pas dans la liste auront la valeur NULL.

Insertion

```
INSERT INTO dept
VALUES (10, 'FINANCES', 'PARIS')

INSERT INTO dept (lieu, nomd, dept)
VALUES ('GRENOBLE', 'RECHERCHE', 20)

INSERT INTO PARTICIPATION (MATR, CODEP)
SELECT MATR, 10 FROM EMP
WHERE NOME = 'MARTIN'
```

Modification

La commande UPDATE permet de modifier les valeurs d'un ou plusieurs champs, dans une ou plusieurs lignes existantes d'une table.

```
UPDATE table
SET col1 = exp1, col2 = exp2, ...
WHERE prédicat
ou
UPDATE table
SET (col1, col2,...) = (SELECT ...)
WHERE prédicat
```

table est le nom de la table mise à jour ; *col1, col2*, ... sont les noms des colonnes qui seront modifiées ; *exp1, exp2*,... sont des expressions.

Elles peuvent aussi être un ordre SELECT renvoyant les valeurs attribuées aux colonnes (deuxième variante de la syntaxe).

Les valeurs de *col1, col2*... sont mises à jour dans toutes les lignes satisfaisant le prédicat.

La clause WHERE est facultative. Si elle est absente, toutes les lignes sont mises à jour.

Modification

Faire passer MARTIN dans le département 10 :

```
UPDATE EMP
SET DEPT = 10
WHERE NOME = 'MARTIN'
```

Donner à CLEMENT un salaire 10 % au dessus de la moyenne des salaires des secrétaires :

```
UPDATE EMP
SET SAL = (SELECT AVG(SAL) * 1.10
           FROM EMP
           WHERE POSTE = 'SECRETAIRE')
WHERE NOME = 'CLEMENT'
```

Suppression

L'ordre DELETE permet de supprimer des lignes d'une table.

```
DELETE FROM table
WHERE prédicat
```

La clause WHERE indique quelles lignes doivent être supprimées.

ATTENTION : cette clause est facultative ; si elle n'est pas précisée, TOUTES LES LIGNES DE LA TABLE SONT SUPPRIMEES (heureusement qu'il existe ROLLBACK!).

Suppression

```
DELETE FROM dept
WHERE dept = 10

DELETE FROM dept  <-- j'efface toutes les lignes
```

Atelier 6

Exercices :

1. Insérez une nouvelle catégorie de produits nommée «fruits et légumes », en respectant les contraintes.
2. Créez un nouveau fournisseur « Grandma » (no_fournisseur = 30) avec les mêmes coordonnées que le fournisseur « Grandma Kellly’s Homestead ».
3. Attribuer les produits de « Grandma Kelly’s Homestead » au nouveau fournisseur créé.
4. Supprimez l’ancien fournisseur «Grandma Kelly’s Homestead» .

Partie 2 la manipulation des objets et fonctions avancées

- ✓ La création d'une table
- ✓ Les données et le contrôle des flux
- ✓ Les triggers
- ✓ Les clefs
- ✓ Les fonctions personnalisées
- ✓ Les procédures stockées
- ✓ Les fonctions analytiques

CREATION D'UNE TABLE

- Identifiants
- Les types de données
  - Systèmes
  - Définis par l'utilisateur
- Gérer les tables
  - Créer une table
  - Modifier une table
  - Supprimer une table
  - Nom complet d'une table

CREATION D'UNE TABLE

- Identification des éléments SQL Server par leur nom

Composés de 1 à 128 caractères

Différents des mots-clés Transact-SQL

- Identifiant régulier

Bonne pratique

Ex: Custtable

- Identifiant délimités

Permet de conserver des caractères spéciaux : accents ou espaces

Ex : [Customer Table]

CREATION D'UNE TABLE

Lors de la création d'une colonne, on doit préciser le format d'utilisation et le mode de stockage. On fait référence aux types de données

- **Système**

Disponible pour toutes les bases de données en standard

  – Caractères

  Char [(n)] : chaîne de caractère longueur fixe

  Varchar (n|max) : chaîne de caractère longueur variable

  Nchar [(n)] : chaîne de caractère longueur fixe, Unicode

  Nvarchar (n|max) : chaîne de caractère longueur variable, Unicode

CREATION D'UNE TABLE

– Decimal

Decimal [(p[,d])] : numérique exact de précision p (nbre total de chiffre) et d chiffres après la virgule

Numeric [(p[,d])] : identique à décimal

Bigint : Entier

Int : nombre entier

Smallint : nombre entier

Tinyint : nombre entier entre et 0 et 255

float [(n)] : numérique approché de n chiffres

Real : identique à float(24)

Money : numérique au format monétaire

Smallmoney : numérique au format monétaire

CREATION D'UNE TABLE

– Binaire

Binary

Varbinary

Type de données	Format	Plage	Précision
time	hh:mm:ss[. nnnnnnn]	00:00:00.0000000 à 23:59:59.9999999	100 nanosecondes
date	AAAA-MM-JJ	0001-01-01 à 9999-12-31	1 jour
smalldatetime	AAAA-MM-JJ hh:mm:ss	1900-01-01 à 2079-06-06	1 minute
datetime	YYYY-MM-DD hh:mm:ss[. nnn]	1753-01-01 à 9999-12-31	0,00333 seconde
datetime2	YYYY-MM-DD hh:mm:ss[. nnnnnnn]	0001-01-01 00:00:00.0000000 à 9999-12-31 23:59:59.9999999	100 nanosecondes
datetimeoffset	YYYY-MM-DD hh:mm:ss[. nnnnnnn] [+|-]hh:mm	0001-01-01 00:00:00,0000000 à 9999-12-31 23:59:59,9999999 (au format UTC)	100 nanosecondes

CREATION D'UNE TABLE

– Spéciaux

Bit : valeur entière 0,1 ou null

Timestamp : mise à jour automatique lors de la modification/insertion de la ligne

unique identifier : pour créer un identificateur unique basé sur la fonction NEWID()

Sql _variant: stocke plusieurs colonnes de types de données différents

Table : stocke un ensemble de valeurs dans un table temporaire (voir l'exemple)

Xml : stocke un document XMLdans une colonne

Cursor : fait référence un curseur

CREATION D'UNE TABLE

- **Définis par l'utilisateur**

Définition de ses propres types de données

  - par SSMS

  Programmabilité/Types
  - Commande CREATE TYPE

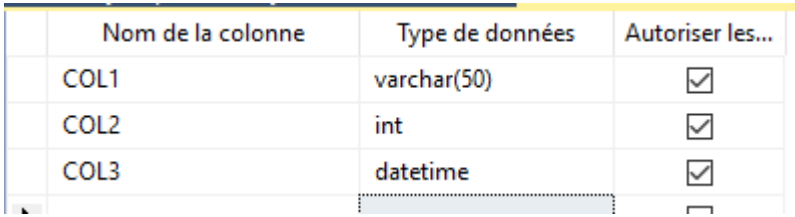
CREATION D'UNE TABLE

- Structure logique
- Enregistrement des données
- Structuration des données via des colonnes
- Etre administrateur, membre du rôle db_owner ou pouvoir exécuter l'instruction CREATE TABLE

- Créer une table
- Modifier une table
- Supprimer une table
- Nom complet d'une table
- Colonnes calculées

CREATION D'UNE TABLE

- Créer une table



Nom de la colonne	Type de données	Autoriser les...
COL1	varchar(50)	☑
COL2	int	☑
COL3	datetime	☑

- CREATE TABLE [dbo].[Table_1](
  [COL1] [varchar](50) NULL,
  [COL2] [int] NULL,
  [COL3] [datetime] NULL
) ON [PRIMARY]

CREATION D'UNE TABLE

- Modifier une table

ALTER TABLE

- Modification de type de données
- ADD : ajout d'un champ
- DROP : suppression d'un champ

```
ALTER TABLE Table_1
        ADD NOUVELLE_COL nchar(10) NULL
```

CREATION D'UNE TABLE

- Modifier une table

ALTER TABLE

- Modification de type de données
- ADD : ajout d'un champ
- DROP : suppression d'un champ

```
ALTER TABLE dbo.Table_1
        DROP COLUMN NOUVELLE_COL
```

CREATION D'UN TABLE

- Modifier une table

ALTER TABLE

- Modification de type de données
- ADD : ajout d'un champ
- DROP : suppression d'un champ

```
ALTER TABLE dbo.table1  ALTER COLUMN col3
DECIMAL (5, 2)
```

CREATION D'UNE TABLE

- Supprimer une table

DROP TABLE

  - Suppression de toutes les données
  - Suppression des déclencheurs et index associés

- Nom complet d'une table
  - Chemin relatif ou absolu des tables

  nomBase.nomSchema.nomObjet
  - Indexation des colonnes calculées

  PERSISTED

CREATION D'UNE TABLE

- Définition

La base de données contient des schémas qui contient des objets

  - Regroupement logique d'objet: le schéma possède un nom significatif et il permet de réaliser un regroupement logique des tables, vues, fonctions, procédures, ….
  - Entité de sécurité : l'autre aspect important des schémas est qu'ils constituent une entité de sécurité. Il est ainsi possible de délivrer des privilèges directement au niveau du schéma et donc concerne tous les objets présents dans le schéma.
  - CREATE SCHEMA

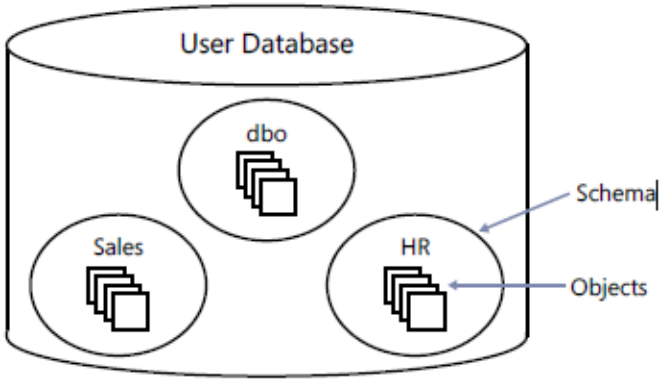


FIGURE 1-8 A database, schemas, and database objects.

CREATION D'UNE TABLE

- Forcer l'intégrité des données
- Concevoir les contraintes
- Manipuler les contraintes

CREATION D'UNE TABLE

- La propriété Identity
- Les contraintes de colonne
  - NOT NULL
  - PRIMARY KEY
  - UNIQUE
  - REFERENCES
  - DEFAULT (DEFAULT ('OUI'))

CREATION D'UNE TABLE

- NOT NULL
  - Valorisation de la colonne en NULL en création et modification
  - Propriété de colonne
- PRIMARY KEY
- UNIQUE
  - Même principe sur la contrainte PRIMARYKEY
  - Unicité des valeurs d'une colonne
  - Index de type unique sous-jacent
- REFERENCES
  - Intégrité référentielle entre clé primaire et clé étrangère
  - ON DELETE/UPDATE CASCADE : suppression/mis à jour des clés étrangères si suppression/modification de la clé primaire
  - SET NULL/DEFAULT : si suppression clé primaire, la clé étrangère devient NULL/la valeur par défaut définit au niveau de la colonne

CREATION D'UNE TABLE

- DEFAULT
  - Valeur par défaut si aucune information précisée en insertion
  - Utile si la contrainte NOT NULL est définie

Atelier 7

Exercices :

1. Créer la table stock qui contient pour chaque référence produit , date, une valeur de stock
2. Insérer dans la table stock, chaque référence produit, à la dernière date de commande, l'état des stocks→ insert

- CREATE TABLE [dbo].[Table_1](
  [COL1] [varchar](50) NULL,
  [COL2] [int] NULL,
  [COL3] [datetime] NULL
) ON [PRIMARY]

CREATION DES CLEFS

- CLEF PRIMAIRE

```
ALTER TABLE dbo.Table_1 ADD CONSTRAINT
PK_Table_1 PRIMARY KEY CLUSTERED
(
COL1
)
```

Attention une clef primaire ne peut pas être null.

Une clef primaire peut-être générée sur deux colonnes.

CREATION DES CLEFS

- CLEF ETRANGERE

```
ALTER TABLE [dbo].[Table_1]  ADD  CONSTRAINT
[FK_Table_1_CLIENTS] FOREIGN KEY([COL1])
REFERENCES [dbo].[CLIENTS] ([CODE_CLIENT])
```

Attention la colonne d'une clef étrangère dot avoir le même format que la colonne source.

Atelier 8

Exercices :

1. Définir la clef primaire de la table stock.

Contrôle de flux

- Fonctionnement du langage de contrôle de flux T-SQL
- IF…ELSE
- WHILE
- RETURN
- PRINT
- CASE
- BEGIN ... END
- OUTPUT

Fonctionnement du langage de contrôle de flux T-SQL

- SQL Server fournit des éléments de langue supplémentaires qui contrôlent le flux d'exécution des instructions T-SQL
  - Utilisé dans les lots, les procédures stockées, les fonctions à instructions multiples
- Les éléments de contrôle de flux permettent aux instructions de s'exécuter dans un ordre spécifié ou non
  - La valeur par défaut est l'exécution séquentielle des instructions
- Includes IF…ELSE, BEGIN…END, WHILE, RETURN, and others

BEGIN ... END

- BEGIN … END permet de delimiter une série d'instruction
  - Utilisé avec les tests IF et les boucles WHILE

```
BEGIN
        {instruction|bloc}
END
```

IF…ELSE

- IF… ELSE utilise un prédicat pour déterminer le flux du code
  - Le code du bloc IF est exécuté si le prédicat a la valeur TRUE
  - Le code du bloc ELSE est exécuté si le prédicat a la valeur FALSE ou UNKNOWN
- Très utile une fois combiné avec l'opérateur EXISTS

```
IF OBJECT_ID('dbo.t1') IS NULL
     PRINT 'Object does not exist';
ELSE
     DROP TABLE dbo.t1;
GO
```

CASE

- CASE utilize un prédicat pour determiner le flux du code
  - Les codes du bloc CASE WHEN sont executé lorsque les prédicats sont vérifiés
  - Le code du bloc ELSE est exécuté si le prédicat a la valeur FALSE ou UNKNOWN
  - Deux syntaxes

```
CASE code_cat
WHEN 10 THEN 'Imprimantes'
WHEN 20 THEN 'Scanners'
ELSE 'Autres'
END
```

WHILE

- WHILE permet au code de s'exécuter en boucle
- Les instructions du bloc WHILE se répètent si le prédicat a la valeur TRUE
- La boucle se termine lorsque le prédicat a la valeur FALSE ou UNKNOWN
- L'exécution ne peut être modifiée par BREAK ou CONTINUE

```
DECLARE @empid AS INT = 1, @lname AS NVARCHAR(20);
WHILE @empid <=5
    BEGIN
        SELECT @lname = lastname FROM HR.Employees
                WHERE empid = @empid;
        PRINT @lname;
        SET @empid += 1;
    END;
```

Gestion des curseurs

- Un curseur permet de traiter ligne par ligne le résultat d'une requête, contrairement au SQL (SELECT) qui traite un ensemble de lignes.

- Respect des étapes suivantes
  - DECLARE CURSOR
  - OPEN
  - FETCH
  - CLOSE
  - DEALLOCATE

Gestion des curseurs

- DECLARE CURSOR

  DECLARE cursor_name [INSENSITIVE] [SCROLL] CURSOR

  FOR select_statement

  [FOR { READ ONLY | UPDATE [OF column_name [,...n]] }]
- OPEN
  - Rend le curseur utilisable
- FETCH
  - Permet d'extraire une ligne du curseur et de valoriser des variables avec leur contenu.
  - Après le FETCH, @@FETCH_STATUS si le fetch s'est bien passé
- CLOSE
  - Fermeture du curseur
- DEALLOCATE
  - Suppression du curseur

Gestion des curseurs

- Les curseurs : permettent de parcourir une table afin d'utiliser les résultats au sein de la procédure.

```
DECLARE @Titre Varchar(10)

DECLARE curseur_variable CURSOR FOR SELECT colA FROM Table

OPEN curseur_variable
FETCH curseur_variable INTO @Titre

WHILE @@FETCH_STATUS = 0
BEGIN
        PRINT @nom
        FETCH curseur_variable INTO @Titre
END
CLOSE curseur_auteurs
DEALLOCATE curseur_auteurs
```

Gestion structurée des exceptions

- La gestion des exceptions structurées permet une réponse centralisée des erreurs d'exécution
  - TRY pour exécuter un bloc de commandes et CATCH pour détecter toutes les erreurs
  - L'exécution passe au bloc CATCH de commandes lorsqu'une erreur se produit
  - Inutile d'activer chaque instruction pour vérifier si une erreur s'est produite
  - Vous déterminez si la transaction doit être restaurée, les erreurs consignées, etc.
- Toutes les erreurs peuvent être interceptées par TRY/CATCH
  - Erreurs de syntaxe ou de compilation
  - Certaines erreurs de résolution de noms

Création de blocs TRY et CATCH

- Bloc TRY défini par les instructions BEGIN TRY… END
  - Placez tout le code qui peut générer une erreur entre elles
  - Aucun code ne peut être placé entre END TRY et BEGIN CATCH)
  - Les blocs TRY et CATCH peuvent être imbriqués
- Bloc CATCH défini par BEGIN CATCH… END CATCH
  - L'exécution passe au bloc CATCH lorsqu'une erreur pouvant être détectée se produit dans le bloc TRY

```
BEGIN TRY
        SELECT 1/0; --generate error
END TRY
BEGIN CATCH
        --Respond to error here
END CATCH;
```

LES FUNCTIONS PERSONALISEES

- Les fonctions personnalisées, permettent de créer des calculs auto

```
CREATE FUNCTION name ([@parameter1 type1,
@parameter2 type ...])
      RETURNS type
      BEGIN
            sql_statement
            RETURN @value
      END
```

- Exécution des procédures :

Select dbo.name(, …) from ….

Atelier 9

Exercices :

1. Créer une fonction POURC qui calcul le pourcentage entre la valeur 1 par rapport à la valeur 2.
2. Afficher pour les clients brésiliens le montant total de commandes ainsi que le pourcentage sur les ventes totales. (En utilisant la fonction)

LES TRIGGERS

- Les triggers ou déclencheurs servent à étendre les mécanismes de gestion de l'intégrité référentielle (liens d'héritage par exemple) et permettre le contrôle de saisie.
- Il s'agit de code déclenché (comme un script) lors de certains événements de la base de données. Un trigger est toujours rattaché à une table ou à une vue. Les événements qui déclenchent un trigger sont :
  - INSERT
  - DELETE
  - UPDATE

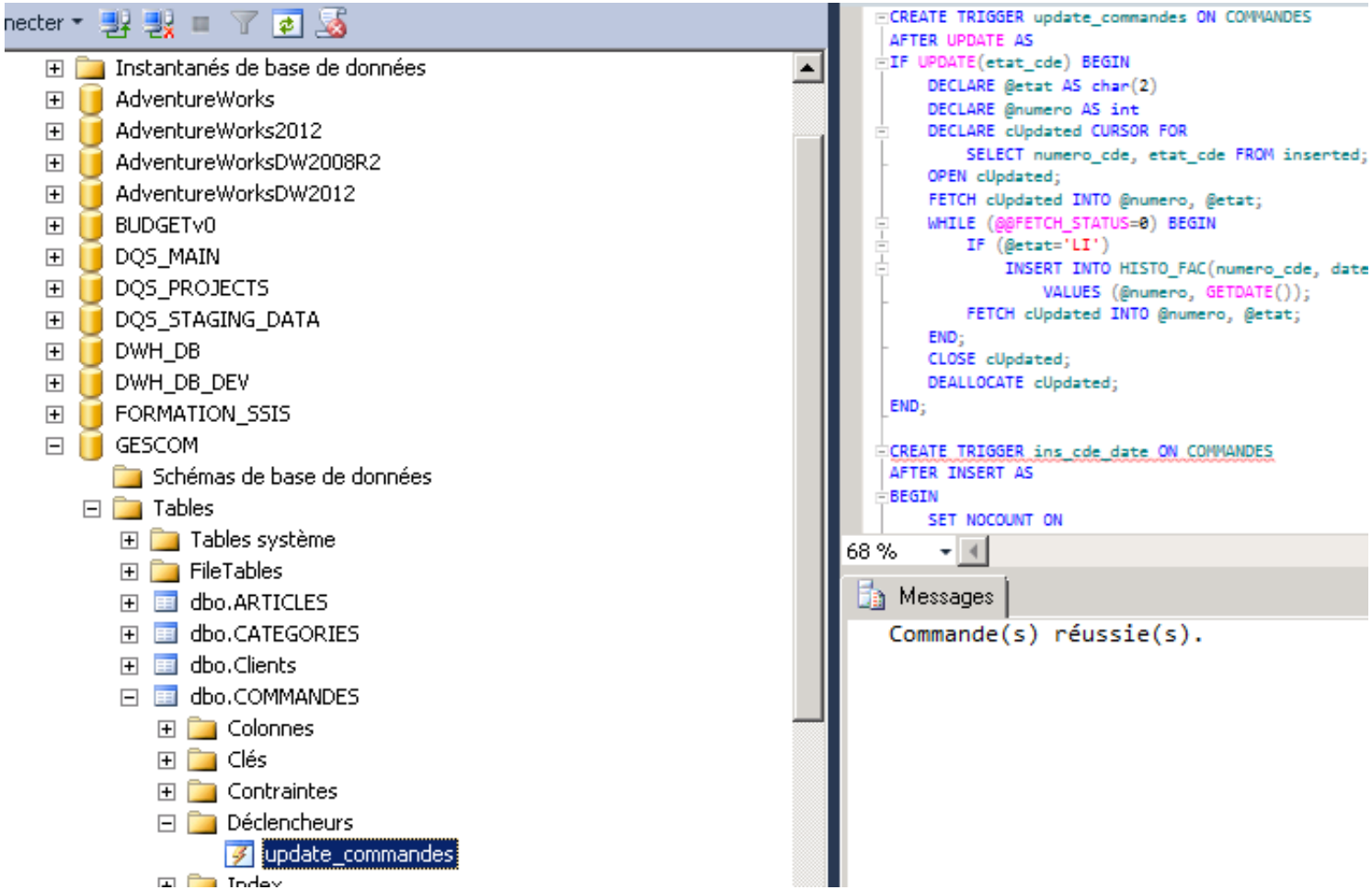
LES TRIGGERS

- SYNTAXE

```
CREATE TRIGGER <nom_trigger>
ON <table_ou_vue>
  {FOR | AFTER | INSTEAD   OF } [ INSERT ] [ , ] [ UPDATE ] [ , ] [DELETE]
[NOT FOR REPLICATION]
AS <instruction_SQL>
```

  - FOR : ordre SQL DML associé
  - AFTER : par défaut , après modification de données
  - INSTEAD OF : le corps du trigger est exécuté à la place de l'ordre SQL envoyé sur la table ou la vue.

LES TRIGGERS



LES TRIGGERS

- **Insert Operation:**

Lors de l'insertion d'un nouvel enregistrement, la table virtuelle "INSERTED" va contenir la nouvelle ligne insérée le temps de la durée de vie du trigger

- **Update Operation**:

Lors de la mise à jour d'un enregistrement, d'abord l'ancient enregistrement va être place dans la table "DELETED" et le nouveau dans la table.

- **Delete Operation**:

Lors de la suppression d'un enregistrement, ce dernier va être exploitable par le trigger dans la table virtuelle "DELETED"

DML Operation	Inserted Table Holds...	Deleted Table Holds...
INSERT	Rows to be inserted	Empty
UPDATE	New (proposed) version of rows modified by the update	Existing (pre-update) version of rows modified by the update
DELETE	Empty	Rows to be deleted

LES TRIGGERS

- **Structure**

Les tables temporaires INSERTED et DELETED possèdent la même structure que la table qui vient de subir l'insertion, la modification ou la suppression d'un enregistrement.

- **Requête**

Il est possible d'effectuer uniquement des requêtes SELECT sur ces tables dans le trigger :

  - SELECT * FROM INSERTED
  - SELECT * FROM DELETED

LES TRIGGERS

- **Structure**

Les tables temporaires INSERTED et DELETED possèdent la même structure que la table qui vient de subir l'insertion, la modification ou la suppression d'un enregistrement.

- **Requête**

Il est possible d'effectuer uniquement des requêtes SELECT sur ces tables dans le trigger :

  - SELECT * FROM INSERTED
  - SELECT * FROM DELETED

Atelier 10

Exercices :

1. Créer un trigger qui met à jour la table STOCK a chaque ajout de ligne dans DETAILS_COMMANDE
2. Créer un trigger qui met à jour la table STOCK à chaque suppression dans DETAILS_COMMANDE

La table de stock doit contenir les mouvements de stocks depuis le dernier état des stocks :

- Soit le delta entre le dernier stock et le stock à l'issu du mouvement.

Tester avec la commande : NO_COMMANDE : 11078, CODE_CLIENT : 'ALFKI, NO_EMPLOYE : 1, date de commande : getdate(), date d'envoi : getdate()+15, port : 10

Dans detail de commande : La reférence produit 1, prix unitaire 90, qte 1, remise 0

LES PROCEDURES STOCKEES

- Les procédures stockées sont des programmes SQL qui permettent d'effectuer plusieurs actions SQL cohérentes ou un traitement préparatoire à l'extraction d'un jeu de données

```
CREATE PROCEDURE NOM_PROC AS
BEGIN
SELECT ….
UPDATE ….
…
END
```

- Exécution des procédures :

```
EXECUTE PROCEDURE NOM_PROC
```

Étude des procédures stockées

- Les procédures stockées sont des collections d'instructions T-SQL enregistrées dans une base de données
- Les procédures peuvent retourner des résultats, manipuler des données et effectuer des actions administratives sur le serveur
- Avec d'autres objets, les procédures peuvent fournir une interface de programmation d'applications fiables à une base de données, isolant les applications des changements de structure de base de données
  - Utilisez les vues, fonctions et procédures pour retourner des données
  - Utilisez les procédures pour modifier et ajouter de nouvelles données
  - Modifiez la définition de la procédure à un emplacement, au lieu de mettre à jour le code de l'application

Exécution des procédures stockées

- Appelez une procédure stockée à l'aide de EXECUTE ou EXEC
- Appelez la procédure avec le nom en deux parties
- Transmettez des paramètres sous la forme de @name=value, à l'aide du type de données approprié

```
EXEC Production.ProductsbySuppliers
   @supplierid = 1;
```

```
EXEC Production.ProductsbySuppliers
   @supplierid = 1, @numrows = 2;
```

LES PROCEDURES STOCKEES

- Exemple

```
CREATE PROCEDURE GetNom AS
BEGIN
        SELECT NOM_COMPLET = NOM + ' ' + PRENOM
        FROM EMP
END GO

Execute GetNom
```

Passage des paramètres à des procédures stockées

- Les paramètres sont définis dans l'en-tête du code de procédure, notamment le nom, le type de données et la direction (par défaut, l'entrée)
- Les paramètres sont détectables à l'aide de SQL Server Management Studio et la vue sys.parameters
- Pour passer les paramètres d'une instruction EXEC, utiliser les noms définis dans la procédure

```
CREATE PROCEDURE
Production.ProductsbySuppliers
(@supplierid AS INT)
AS ...
```

```
EXEC Production.ProductsbySuppliers
   @supplierid = 1;
```

LES PROCEDURES STOCKEES

- Les paramètres : permettent d'exécuter une procédure en tenant compte d'un argument.
- Exemple : sélectionner les employés dont le Titre est 'M.'

```
CREATE PROCEDURE GetNom
@Titre Varchar(10)
AS
BEGIN
        SELECT NOM_COMPLET = NOM + ' ' + PRENOM
        FROM EMP
        WHERE TITRE=@Titre
END GO
```

. EXECUTE GetNom 'M. '

Utilisation des paramètres de sortie

- Les paramètres de sortie vous permettent de retourner les valeurs d'une procédure stockée
  - À comparer avec le retour d'un jeu de résultats
- Paramètre marqué pour la sortie dans l'en-tête de la procédure et dans la requête appelante

```
CREATE PROCEDURE <proc_name>
(@<input_param> AS <type>,
 @<output_param> AS <type> OUTPUT)
 AS ...
```

```
DECLARE @<output_param> AS <type>;
 EXEC <proc_name>
<input_parameter_list>,
@<output_param> OUTPUT;
 SELECT @output_param;
```

Création de procédures qui acceptent des paramètres

- Les paramètres d'entrée transmis à la procédure se comportent logiquement comme les variables locales dans le code de la procédure
- Attribuez un nom avec un préfixe @, un type de données dans l'en-tête de la procédure
- Consultez le paramètre dans le corps de la procédure

```
CREATE PROCEDURE Production.ProdsByCategory
(@numrows AS int, @catid AS int)
 AS
SELECT TOP(@numrows) productid,
      productname, unitprice
FROM  Production.Products
WHERE       categoryid = @catid;
```

RAPPEL : Contrôle de flux

- Fonctionnement du langage de contrôle de flux T-SQL
- IF…ELSE
- WHILE
- RETURN
- PRINT
- CASE
- BEGIN ... END
- OUTPUT

RAPPEL : Fonctionnement du langage de contrôle de flux T-SQL

- SQL Server fournit des éléments de langue supplémentaires qui contrôlent le flux d'exécution des instructions T-SQL
  - Utilisé dans les lots, les procédures stockées, les fonctions à instructions multiples
- Les éléments de contrôle de flux permettent aux instructions de s'exécuter dans un ordre spécifié ou non
  - La valeur par défaut est l'exécution séquentielle des instructions
- Includes IF…ELSE, BEGIN…END, WHILE, RETURN, and others

RAPPEL : BEGIN … END

- BEGIN … END permet de delimiter une série d'instruction
  - Utilisé avec les tests IF et les boucles WHILE

```
BEGIN
        {instruction|bloc}
END
```

RAPPEL : IF…ELSE

- IF… ELSE utilise un prédicat pour déterminer le flux du code
  - Le code du bloc IF est exécuté si le prédicat a la valeur TRUE
  - Le code du bloc ELSE est exécuté si le prédicat a la valeur FALSE ou UNKNOWN
- Très utile une fois combiné avec l'opérateur EXISTS

```
IF OBJECT_ID('dbo.t1') IS NULL
     PRINT 'Object does not exist';
ELSE
     DROP TABLE dbo.t1;
GO
```

RAPPEL : CASE

- CASE utilize un prédicat pour determiner le flux du code
  - Les codes du bloc CASE WHEN sont executé lorsque les prédicats sont vérifiés
  - Le code du bloc ELSE est exécuté si le prédicat a la valeur FALSE ou UNKNOWN
  - Deux syntaxes

```
CASE code_cat
WHEN 10 THEN 'Imprimantes'
WHEN 20 THEN 'Scanners'
ELSE 'Autres'
END
```

RAPPEL : WHILE

- WHILE permet au code de s'exécuter en boucle
- Les instructions du bloc WHILE se répètent si le prédicat a la valeur TRUE
- La boucle se termine lorsque le prédicat a la valeur FALSE ou UNKNOWN
- L'exécution ne peut être modifiée par BREAK ou CONTINUE

```
DECLARE @empid AS INT = 1, @lname AS NVARCHAR(20);
WHILE @empid <=5
    BEGIN
        SELECT @lname = lastname FROM HR.Employees
                WHERE empid = @empid;
        PRINT @lname;
        SET @empid += 1;
    END;
```

RAPPEL : Gestion des curseurs

- Un curseur permet de traiter ligne par ligne le résultat d'une requête, contrairement au SQL (SELECT) qui traite un ensemble de lignes.

- Respect des étapes suivantes
  - DECLARE CURSOR
  - OPEN
  - FETCH
  - CLOSE
  - DEALLOCATE

RAPPEL : Gestion des curseurs

- DECLARE CURSOR

  DECLARE cursor_name [INSENSITIVE] [SCROLL] CURSOR

  FOR select_statement

  [FOR { READ ONLY | UPDATE [OF column_name [,...n]] }]
- OPEN
  - Rend le curseur utilisable
- FETCH
  - Permet d'extraire une ligne du curseur et de valoriser des variables avec leur contenu.
  - Après le FETCH, @@FETCH_STATUS si le fetch s'est bien passé
- CLOSE
  - Fermeture du curseur
- DEALLOCATE
  - Suppression du curseur

RAPPEL : Gestion des curseurs

- Les curseurs : permettent de parcourir une table afin d'utiliser les résultats au sein de la procédure.

```
DECLARE @Titre Varchar(10)

DECLARE curseur_variable CURSOR FOR SELECT colA FROM Table

OPEN curseur_variable
FETCH curseur_variable INTO @Titre

WHILE @@FETCH_STATUS = 0
BEGIN
        PRINT @nom
        FETCH curseur_variable INTO @Titre
END
CLOSE curseur_auteurs
DEALLOCATE curseur_auteurs
```

Atelier 11

Exercices :

1. Créer une table PANIER qui contient les paniers en attente de validation : Colonnes (NUM_PANIER, NOM_CLIENT, NOM_PRODUIT, QTE)
2. Définir la clef primaire
3. Ajouter le panier suivant :

   Client: France restauration, Produit: Ikura, qte:2

   Client:France restauration, Produit: Konbu, qte:5
4. Créer une table PANIER_HISTORIQUE qui contient les colonnes de PANIER plus une colonne de date_de_cde qui correspond à la date de mise à jour de la table de « commandes » avec les lignes de panier.
5. Développer une procédure stockée UPDT_CDE qui va lire la table panier, pour mettre à jour les tables « commandes » et « détails de commandes » → prendre la date du jour comme date de commande et la date du jour plus 15 jours comme date d'envoi (Utilisation d'un curseur)
6. La procédure doit vider la table PANIER dans PANIER_HISTORIQUE en fin d'exécution

Atelier 12

Exercices :

1. Créer la table de SUIVI_CLIENT_JOUR dans le schéma devant contenir :

   CODE_CLIENT, SOCIETE, DATE_COMMANDE, CA JOUR, CA CUMULE

1. Créer une procédure MAJ_STAT_CLIENT_JOUR qui remplit la table SUIVI_CLIENT_JOUR en mode annule et remplace.

(Attention : la colonne CA_CUMULE doit contenir la somme CUMULE des CA par jour depuis le début, utilisation de curseur)

```
CREATE TABLE SUIVI_CLIENT_JOUR (                    CODE_CLIENT CHAR(5) NOT NULL,  SOCIETE VARCHAR(40) NOT NULL,  DATE_COMMANDE DATETIME
NOT NULL,          CA_COMMANDE NUMERIC(30,2) NOT NULL,            CA_CUMULE NUMERIC(30,2) NOT NULL)

ALTER PROCEDURE MAJ_STAT_CLIENT_JOUR AS
BEGIN
                    DECLARE @CODE_CLIENT CHAR(5)
                    DECLARE @SOCIETE VARCHAR(40)
                    DECLARE @DATE_COMMANDE DATETIME
                    DECLARE @CA_COMMANDE NUMERIC(30,2)
                    DECLARE @CA_CUMULE NUMERIC(30,2)
                    DECLARE @NO_COMMANDE NUMERIC(6,0)
                    DECLARE @CODE_CLIENT_TMP CHAR(5)
                    DECLARE curseur_commande CURSOR FOR SELECT NO_COMMANDE, CODE_CLIENT, DATE_COMMANDE
 FROM COMMANDES ORDER BY CODE_CLIENT, DATE_COMMANDE
                    OPEN curseur_commande
                    FETCH curseur_commande INTO @NO_COMMANDE, @CODE_CLIENT, @DATE_COMMANDE
                    -- Initialisation des variables pour les tests (le premier clients et forcément égal à lui même ;D)
                    SET @CODE_CLIENT_TMP = @CODE_CLIENT
                    SET @SOCIETE = (SELECT SOCIETE FROM CLIENTS WHERE CODE_CLIENT = @CODE_CLIENT)
                    SET @CA_CUMULE = 0
                    WHILE @@FETCH_STATUS = 0
                    BEGIN
                                        IF @CODE_CLIENT != @CODE_CLIENT_TMP
                                        BEGIN
                                        SET @CA_CUMULE = 0
                                        SET @CODE_CLIENT = @CODE_CLIENT_TMP
                                        SET @SOCIETE = (SELECT SOCIETE FROM CLIENTS WHERE CODE_CLIENT = @CODE_CLIENT)
                                        END
                    SET @CA_COMMANDE = (SELECT SUM(ISNULL(PRIX_UNITAIRE, 0) * ISNULL(QUANTITE, 0)) FROM DETAILS_COMMANDES WHERE
NO_COMMANDE = @NO_COMMANDE GROUP BY NO_COMMANDE)
                    SET @CA_CUMULE += @CA_COMMANDE
                    INSERT INTO SUIVI_CLIENT_JOUR VALUES(@CODE_CLIENT, @SOCIETE, @DATE_COMMANDE, @CA_COMMANDE, @CA_CUMULE)
                    FETCH curseur_commande INTO @NO_COMMANDE, @CODE_CLIENT_TMP, @DATE_COMMANDE
                    END
                    CLOSE curseur_commande
                    DEALLOCATE curseur_commande
END
```

```
Declare @CC as char(5)
Declare @CC_PREC as char(5)
Declare @SC as Varchar(40)
Declare @DC as datetime
Declare @CA as numeric(30,2)
Declare @CA_PREC as numeric(30,2)
Declare @CA_CUM as numeric(30,2)

SET @CA_PREC=0
SET @CA_CUM=0
SET @CC_PREC=''

DECLARE C_CDE CURSOR FOR
SELECT C.CODE_CLIENT, CL.SOCIETE, C.DATE_COMMANDE, DC.QUANTITE*DC.PRIX_UNITAIRE as CA
from COMMANDES C inner join CLIENTS CL on (C.CODE_CLIENT=CL.CODE_CLIENT)
inner join DETAILS_COMMANDES DC on (C.NO_COMMANDE=DC.NO_COMMANDE)
order  by C.CODE_CLIENT, C.DATE_COMMANDE

OPEN C_CDE
FETCH C_CDE INTO @CC,@SC,@DC,@CA

while @@FETCH_STATUS=0
BEGIN
IF @CC=@CC_PREC
@CA_CUM=@CA_CUM+@CA
ELSE
@CA_CUM=@CA

INSERT INTO SUIVI_CLIENT_JOUR VALUES (@CC, @SC, @DC, @CA, @CA_CUM)

@CC_PREC=@CC
FETCH C_CDE INTO @CC,@SC,@DC,@CA
END
CLOSE C_CDE
DEALLOCATE C_CDE
```

LES FONCTIONS ANALYTIQUES

- Les fonctions analytiques permettent :
  - De sortir des résultats agrégés au niveau des lignes détails

  Exemple : J'ai une table d'employé ou j'ai le salaire pour chaque mois, je veux afficher le salaire de chaque employé par mois avec le cumul depuis le début de l'année.

  - De récupérer des valeurs de ligne précédente.

  Exemple : j'ai une table qui contient les appels de téléopérateur auprès des clients prospectés, je veux connaitre le temps moyen entre chaque appel.

LES FONCTIONS ANALYTIQUES

- Composition d'une fonction analytique :

Function(« champs à agréger ») over (partition by « champs de rupture » order by « champs de classement des données »)

Exemple : sum(salaire) over (partition by id_employe order by mois)

LES FONCTIONS ANALYTIQUES

- Les fonctions d'agrégat utilisables :

Sum, max, min, …

- Les fonctions analytiques utilisables :

First_Value : retourne la première valeur

Last_Value : retourne la dernière valeur

LAG : retourne la valeur d'une ligne précédente

LAG(champs, decalage, valeur par defaut) ou décalage est le nombre de ligne à remonter pour prendre la référence.

LEAD : retourne la valeur d'une ligne suivante

LEAD(champs, decalage, valeur par defaut) ou décalage est le nombre de ligne à descendre pour prendre la référence.

Atelier 13

Exercices :

1. Déterminer le délai moyen par client entre deux commandes en nombre de jour.
2. Créer la procédure MAJ_STAT_CLIENT_JOUR2 qui remplit la table SUIVI_CLIENT_JOUR2 en mode annule et remplace.

A l'image de la procédure MAJ_STAT_CLIENT_JOUR mais en utilisant une fonction analytique.

```
select SOCIETE, AVG(ECART) as nb_jours_moyen from (
select C.CODE_CLIENT, CL.SOCIETE, C.DATE_COMMANDE
,
LAG(C.DATE_COMMANDE, 1, null) over (partition by C.CODE_CLIENT order by
C.DATE_COMMANDE, C.NO_COMMANDE) as DATE_COMMANDE_PREC,
DATEDIFF(DD, LAG(C.DATE_COMMANDE, 1, null) over (partition by
C.CODE_CLIENT order by C.DATE_COMMANDE,
C.NO_COMMANDE),C.DATE_COMMANDE) as ecart
from CLIENTS CL inner join COMMANDES C
ON (CL.CODE_CLIENT=C.CODE_CLIENT)) A
group by SOCIETE
```

```
select CODE_CLIENT, SOCIETE, DATE_COMMANDE, SUM(CA), CA_CUMULE from (
select C.CODE_CLIENT, CL.SOCIETE, C.DATE_COMMANDE,
(DC.PRIX_UNITAIRE*DC.QUANTITE) as CA
,
sum(DC.PRIX_UNITAIRE*DC.QUANTITE) over (partition by C.CODE_CLIENT order by
C.DATE_COMMANDE, C.NO_COMMANDE) as CA_CUMULE
from CLIENTS CL inner join COMMANDES C
ON (CL.CODE_CLIENT=C.CODE_CLIENT)
inner join DETAILS_COMMANDES DC ON (C.NO_COMMANDE=DC.NO_COMMANDE)) A
GROUP BY CODE_CLIENT, SOCIETE, DATE_COMMANDE,  CA_CUMULE
```

L'EXISTANCE DES OBJETS

- Il est souvent utile de pouvoir lister les tables, index, vues, … existant dans une base.

Table des bases : dbo.sysdatabases

Table des objets table, vue, index, clef, … : [sys].[all_objects]

SYSTEM_TABLE

VIEW

DEFAULT_CONSTRAINT

SQL_STORED_PROCEDURE

AGGREGATE_FUNCTION

FOREIGN_KEY_CONSTRAINT

USER_TABLE

PRIMARY_KEY_CONSTRAINT

SQL_TRIGGER

- Autre possibilité : IF OBJECT_ID('MAJ_STAT_CLIENT2') IS NOT NULL → signifie que la table existe

Atelier 14

Exercices :

1. Modifier la procédure MAJ_STAT_CLIENT_JOUR2 qui remplit la table SUIVI_CLIENT_JOUR2 pour recréer la table si elle n’existe pas.

LES VUES

- Il est possible de stocker le code SQL exécuté dans un objet vue afin d'éviter de le recréer à chaque fois.

- La création d'une VUE :

- CREATE VIEW "nom de la vue" AS
  Code de la requête SQL

Exemple :

CREATE VIEW TEST AS SELECT * FROM COMMANDES

Atelier 15

Exercices :

1. Créez une vue qui affiche le nom de la société, l'adresse, le téléphone et la ville des clients qui habitent à Toulouse, à Strasbourg, à Nantes ou à Marseille

SELECT INTO

- Il est possible de créer une table dynamiquement sur le résultat d'un select.

- La création d'une VUE :

  select …. Into "nom_de_la_nouvelle_table" from nom_de_la_table_source

Exemple :

SELECT * INTO COMMANDE2 FROM COMMANDES

Atelier 15b

Exercices :

1. A partir de la fonction analytique qui donne le CA_CUMULE par client et date de commande

Et en utilisant le Select … into créer une procédure qui créer la table MAJ_STAT_CLIENT2 .

Vérifier l'existance de la table avec la fonction :

OBJECT_ID('…')

Et si necessaire supprimer la table.

Manipulation des variables sur SELECT

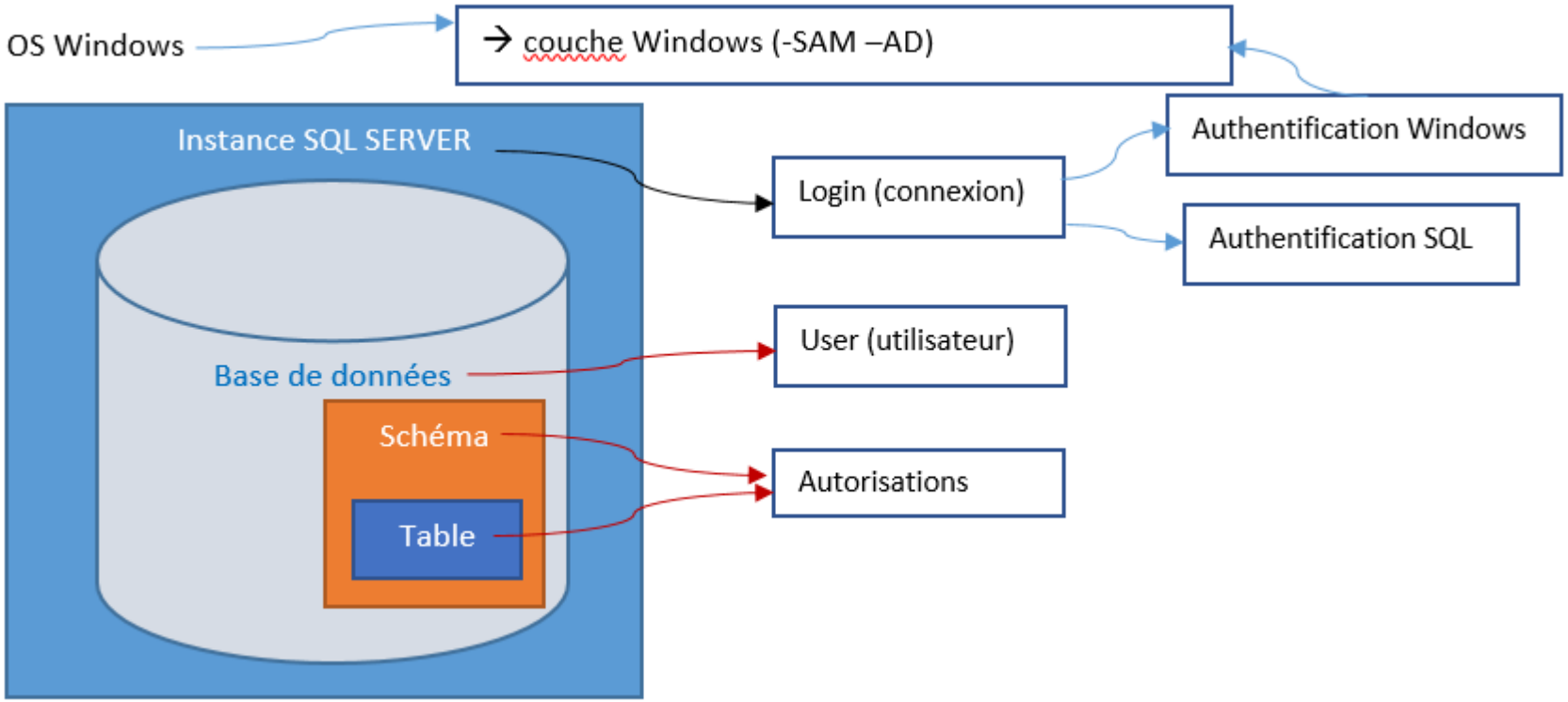
```
declare @nom_complet as varchar(100)
set @nom_complet =(SELECT TOP 1 NOM + ' ' + PRENOM FROM EMPLOYES WHERE TITRE=@Titre)
```

Atelier 17

- Créer une procédure qui en entrée reçoit le prénom et le nom d'un employé et un salaire
  - Si l'employé existe et que son salaire est inférieur à celui entré dans la procédure, lui affecter ce nouveau salaire
  - Si le salaire est inférieur ne rien faire
  - Si l'employé n'existe pas le créer avec le salaire.

ADMINISTRATION

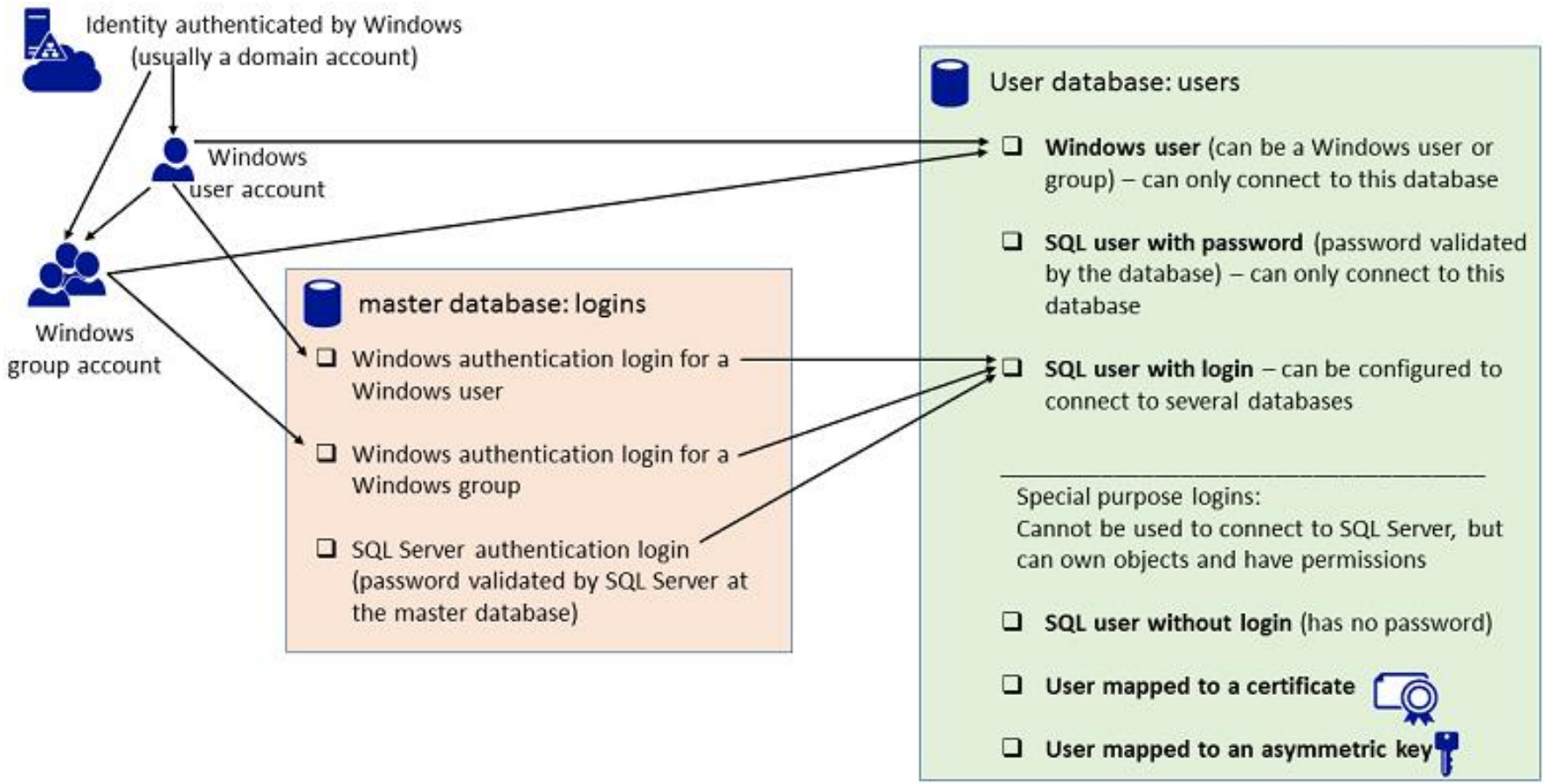
Sécurité


OS Windows
→ couche Windows (-SAM –AD)
Authentification Windows
Instance SQL SERVER
Login (connexion)
Authentification SQL
User (utilisateur)
Base de données
Schéma
Autorisations
Table

Sécurité

- Créer la connexion à l’instance

- Gérer les droits pour chaque base


Identity authenticated by Windows (usually a domain account)
Windows user account
Windows group account
master database: logins
Windows authentication login for a Windows user
Windows authentication login for a Windows group
SQL Server authentication login (password validated by SQL Server at the master database)
User database: users
Windows user (can be a Windows user or group) – can only connect to this database
SQL user with password (password validated by the database) – can only connect to this database
SQL user with login – can be configured to connect to several databases
Special purpose logins:
Cannot be used to connect to SQL Server, but can own objects and have permissions
SQL user without login (has no password)
User mapped to a certificate
User mapped to an asymmetric key

CREATE USER INSTANCE

1. Creates the login AbolrousHazem with password '340$Uuxwp7Mcxo7Khy‘ :

```
CREATE LOGIN AbolrousHazem WITH PASSWORD =
'340$Uuxwp7Mcxo7Khy'
```

2. Creates a database user for the login created above :

```
CREATE USER AbolrousHazem FOR LOGIN
AbolrousHazem
```

Sécurité Instance

- Rôles prédéfinis dans SQL SERVER au niveau d'une instance:
  - bulkadmin : transfert massifs de données (généralement les imports de données), possibilité d'importer des données de l'extérieur vers l'instance
  - dbcreator : create, alter et drop sur les bases.
  - diskadmin : possibilité de changer la capacité des fichiers
  - processadmin : concernant les jobs, administration des process en cours, arrêt des jobs qui s'exécutent (tâche planifiée par exemple)
  - public : pratiquement aucun droit particulier mais c'est le rôle de base pour les utilisateurs
  - securityadmin : creation d'objet logins, roles, create alter drop
  - serveradmin : possibilité d'arrêter le serveur (ou service – net stop de l'instance)
  - setupadmin : création de liaison de serveurs
  - sysadmin : rôle le plus élevé de l'instance → administrateur du système – c'est un rôle, les objets qui y sont contenus sont des logins.
- On parle de connexion (Login)

CREATE USER INSTANCE

1. Creation d’un utilisateur avec le privilège dbcreator et ne respectant la politique de mot de passe (expiration),(contrainte de mot de passe)

```
CREATE LOGIN [TEST] WITH PASSWORD=N'TEST01',
DEFAULT_DATABASE=[master],
CHECK_EXPIRATION=OFF, CHECK_POLICY=OFF;

ALTER SERVER ROLE [dbcreator] ADD MEMBER [TEST];
```

Sécurité BDD

- Rôles prédéfinis dans SQL SERVER au niveau d'une base :
  - db_accessadmin : gestion des autorisations
  - db_securityadmin : création d'objet logins, roles, create alter drop
  - db_datareader : droit select sur toutes les tables de la base
  - db_datawriter : droit insert, update, delete sur toutes les tables de la base
  - db_ddladmin : droit create, alter, drop table, schema
  - db_denydatareader et db_denydatawriter : droit de déni qui prime sur les droits db_datareader et db_datawriter
  - public : pratiquement aucun droit particulier mais c'est le rôle de base pour les utilisateurs
  - db_owner : tous les rôles et droits sont contenus dans celui-ci, généralement administrateur de la base (uniquement la base concernée)
- On parle d'utilisateur (User)

CREATE USER BDD

1. Déclarer l’utilisateur TEST comme utilisateur de la base de données LA400 en tant que reader

CREATE USER [TEST] FOR LOGIN [TEST] WITH DEFAULT_SCHEMA=[dbo]

ALTER ROLE [db_datareader] ADD MEMBER [TEST]

Schéma BDD

- Segmentation des tables dans la base de données
- Exemple : dbo.PRODUITS fait partie du schema dbo.

- Définir un schéma par défaut à utilisateur, signifie que lors de la création d'un objet par l'utilisateur, le schéma associé sera le schéma par défaut.

Atelier 14

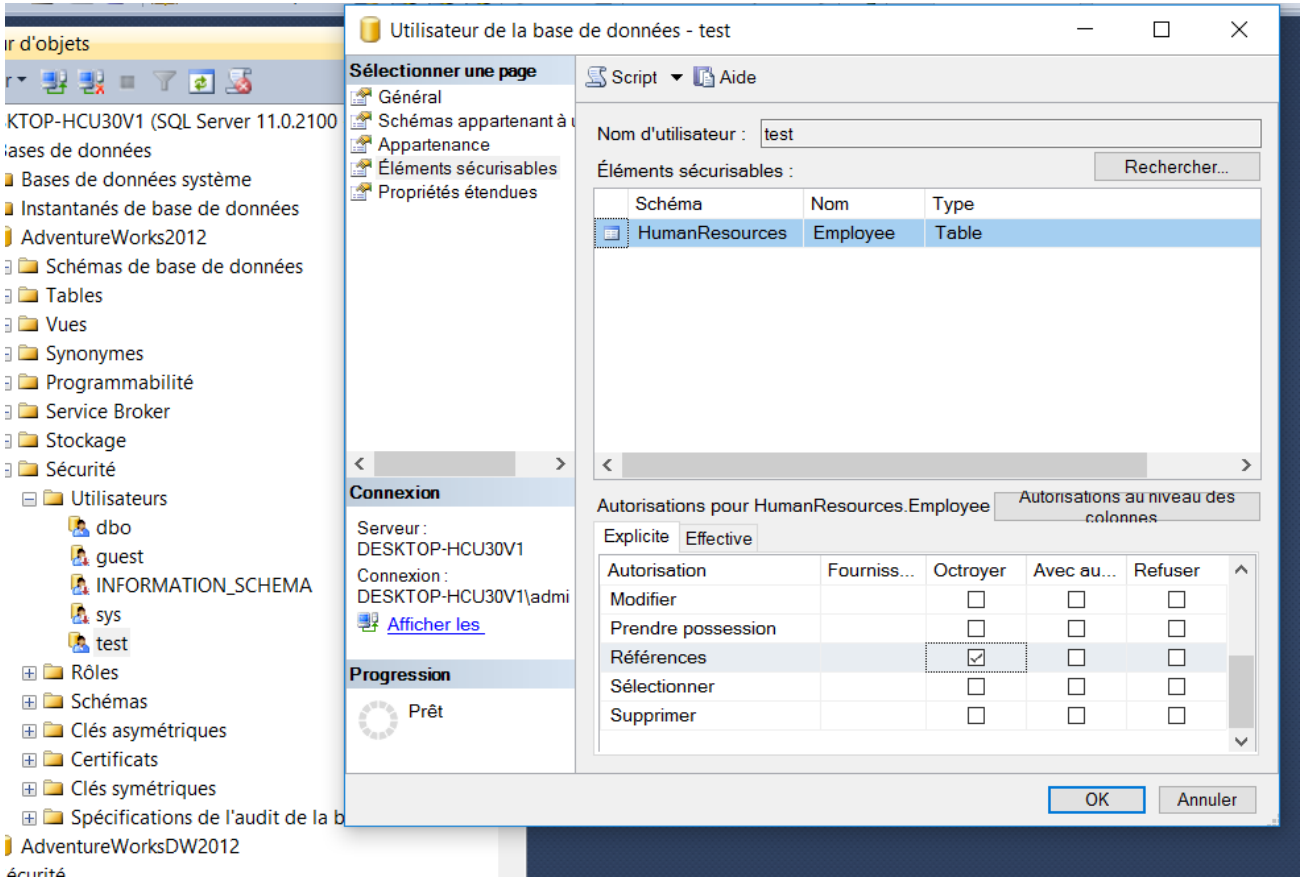
Exercices :

1. Créer l'utilisateur « LectLA300 » qui ne peut que lire les données de la base
2. Créer l'utilisateur « DevLA300 » qui peut lire, mais aussi mettre à jour les données
3. Créer l'utilisateur « AdmLA300 » qui peut lire, mettre à jour la base et créer une table
4. Créer une nouvelle table avec « AdmLA300 » :

   Table TEST avec une colonne ID, une colonne ACTION (varchar(30))
5. Insérer la valeur : 1, 'test insert' avec l'utilisateur « DevLA300 »
6. Faites le même test avec LectLA300

Sécurité

- Possibilité de personnaliser au plus près les droits :



- Sauvegarde et restauration

✓Stratégies de sauvegarde

✓Comprendre la journalisation des transactions SQL Server

✓Planification d'une stratégie de sauvegarde SQL Server

✓Sauvegarde des bases de données et journaux de transactions

✓Gestion des sauvegardes de bases de données

✓Comprendre le processus de restauration

✓Restauration des bases de données

✓Restauration des bases de données systèmes et des fichiers individuels

- Sauvegarde et restauration

Stratégies de sauvegarde

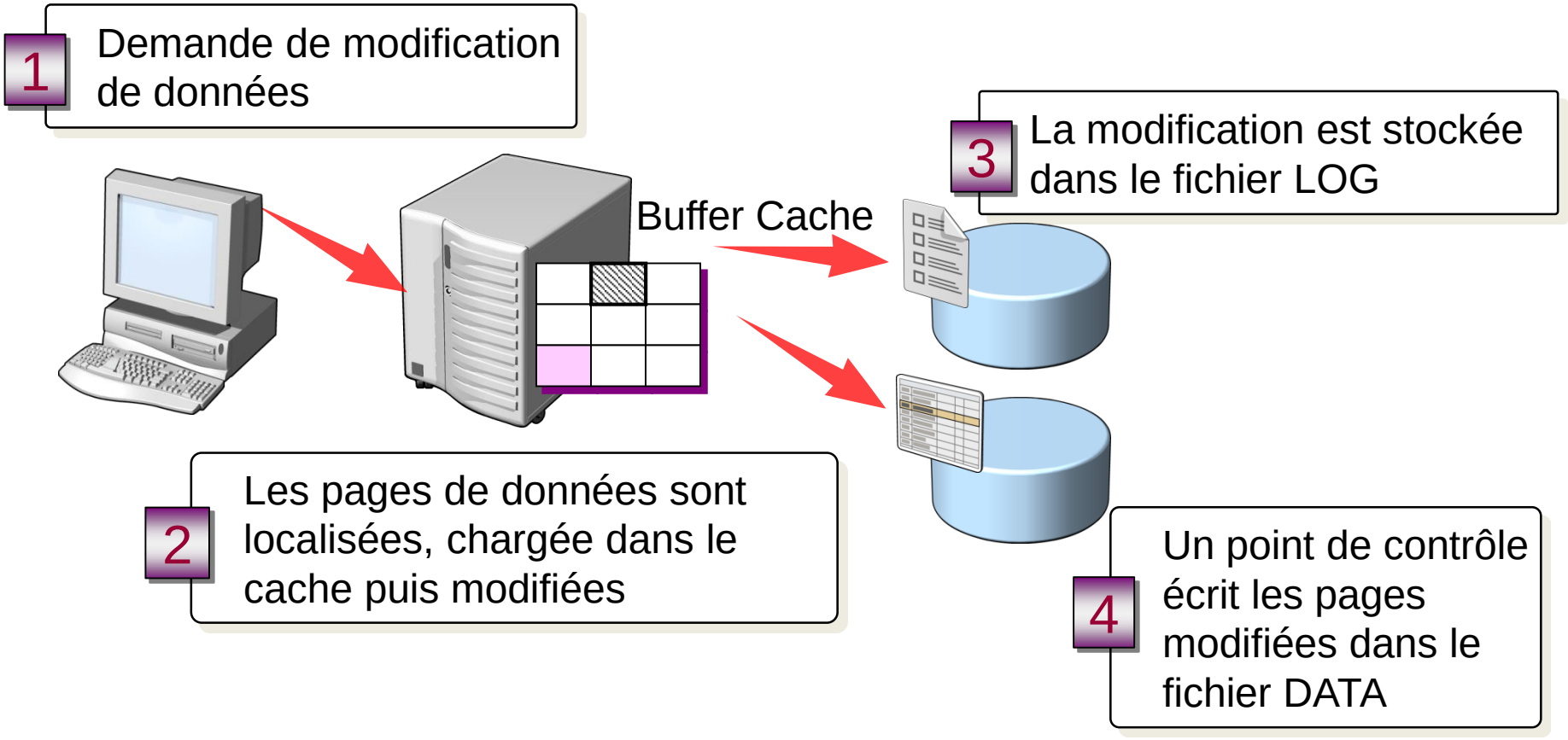
✓**Différents types de sauvegardes peuvent être combinés**
- Sauvegardes basées sur les données (Complet, Différentiel, Copie, Groupe de fichiers, Fichiers)
- Sauvegardes basées sur les journaux de transactions

✓**Des niveaux de sécurité doivent être déterminés**
- Combien de temps doit prendre la récupération ?
- Combien de données est-il acceptable de perdre?

✓**La stratégie de sauvegarde doit correspondre aux besoins**
- Types et fréquences de sauvegardes
- Les supports de sauvegarde à utiliser
- Période de rétention des sauvegardes
- Politique de test des sauvegardes

- Sauvegarde et restauration

Comprendre la journalisation des transactions



- Sauvegarde et restauration

Modes de récupération

Mode	Description
Simple	• N'autorise et ne nécessite pas les sauvegardes du journal • Vide automatiquement le fichier LOG et ne garde que le minimum d'informations
Complet	• Nécessite des sauvegardes du journal • Evite les pertes de données dues à un fichier endommagé • Permet la récupération à un point dans le temps
Journalisé en bloc	• Nécessite des sauvegardes du journal • Peut améliorer les performances des opérations de copie en bloc • Réduire l'espace dédié au journal

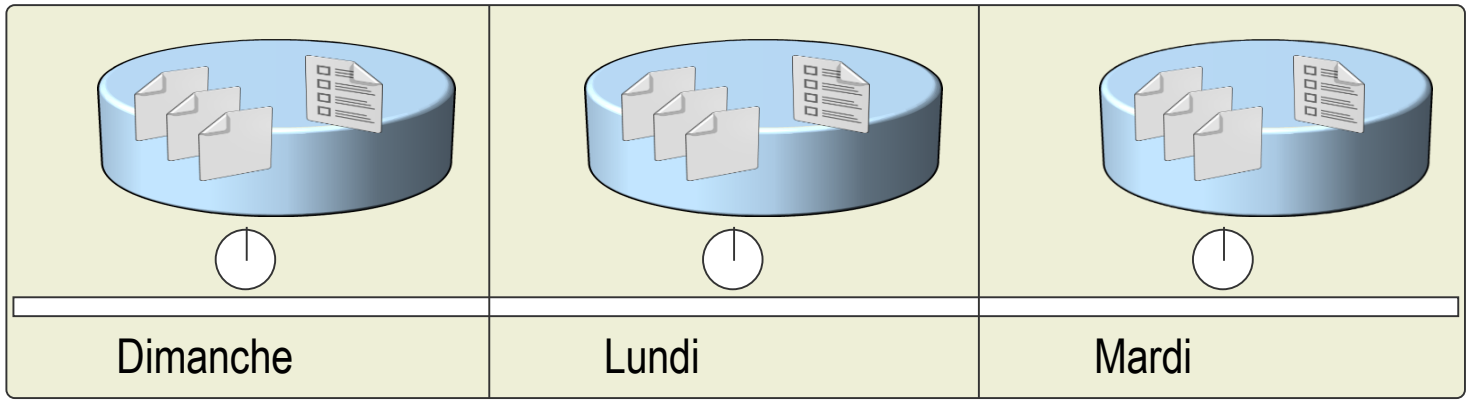
- Sauvegarde et restauration

Planification d'une stratégie de sauvegarde (Option de la base de données)

Types	Description
Complète	Tous les fichiers de données et la partie active du journal de transactions
Différentielle	La partie de la base de données qui avait été changée depuis la dernière sauvegarde complète
Partielle	Groupe de fichiers primaire, tous les groupes de fichiers (read/write) et des groupes de fichiers (read-only) spécifiques.
Journal de transactions	Toutes les modifications de la base de données enregistrées dans le fichier journal de transactions
Fichier journal après défaillance	Portion active du journal
Fichier/Groupe de fichiers	Fichiers ou groupes de fichiers spécifiques
Copie seule	Copie des fichiers de bases de données

- Sauvegarde et restauration

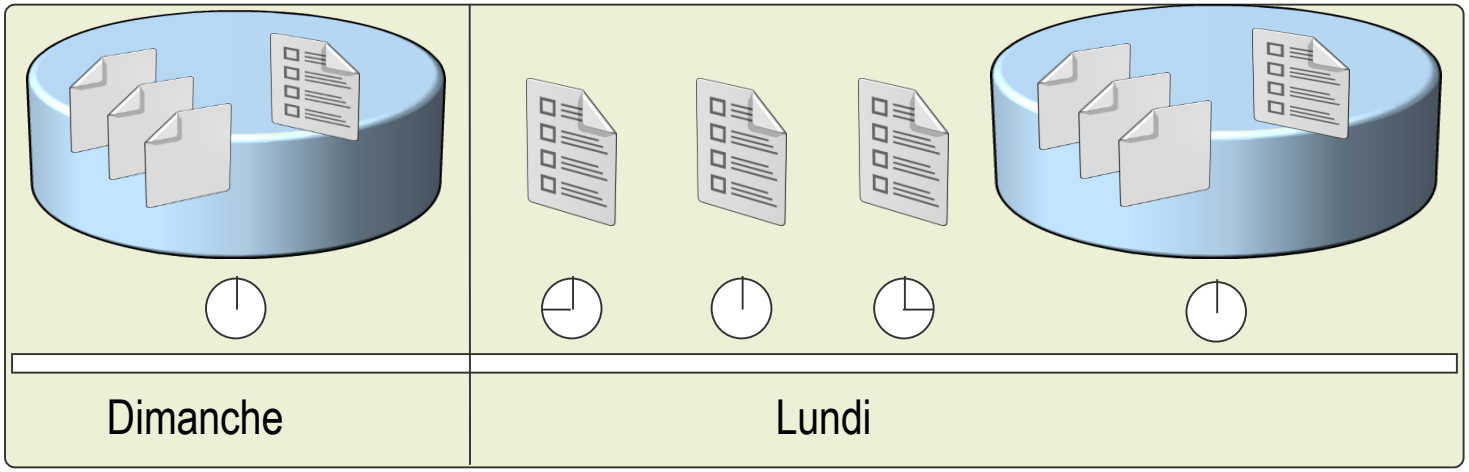
Stratégie de sauvegarde complète



- Sauvegarde toutes les données et une partie du journal de transactions
- Peut être utilisée pour récupérer la totalité de la base de données
- Permet la récupération à la date de sauvegarde uniquement

- Sauvegarde et restauration

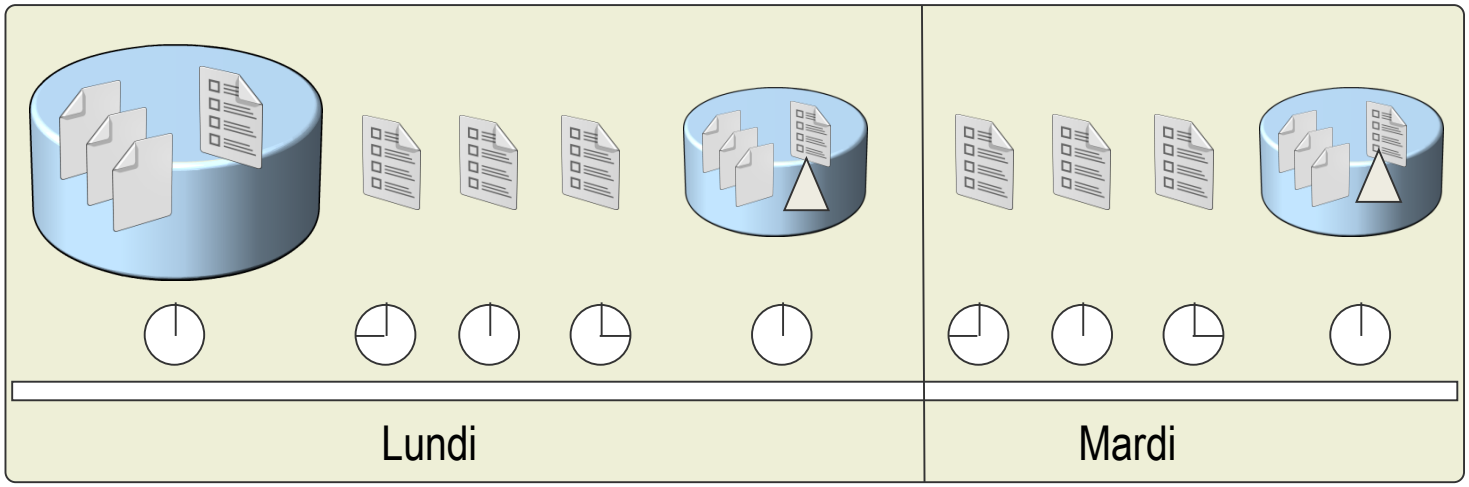
Stratégie de sauvegarde du journal de transactions



- ✓ Implique au moins une sauvegarde complète et une sauvegarde du journal
- ✓ Permet la récupération à un point donné dans le temps
- ✓ La base de données peut être récupérée en totalité

- Sauvegarde et restauration

Stratégie de sauvegarde différentielle



- ✓ Implique la création des sauvegardes complètes et différentielles
- ✓ Comprend des sauvegardes différentielles avec seulement les données modifiées
- ✓ Est utile seulement si une partie d'une base de données est plus fréquemment modifiée que le reste de la base de données

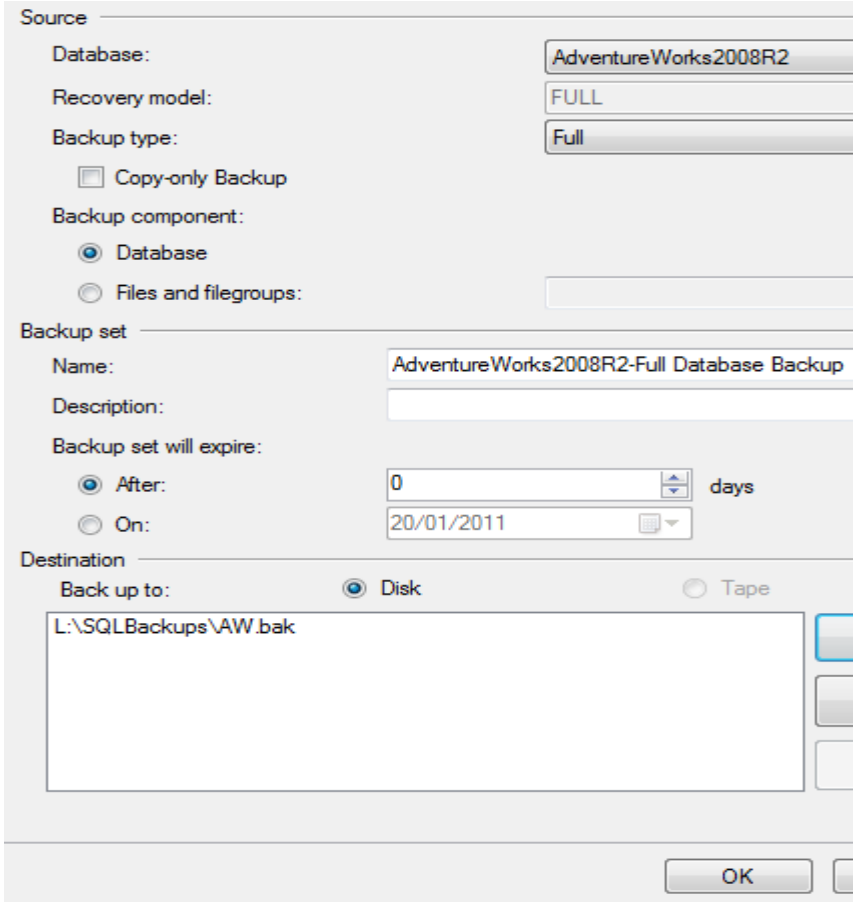
- Sauvegarde et restauration

Sauvegarde des bases de données

✓**Sauvegarde complète de la base de données**

✓Possibilité de compression du fichier de sauvegarde

```
BACKUP DATABASE
AdventureWorks2008R2
TO DISK =
  'L:\SQLBackups\AW.bak'
WITH INIT;
```



- Sauvegarde et restauration

Sauvegarde des bases de données

✓**Sauvegarde différentielle de la base de données**

✓Sauvegarder seulement les modifications depuis la dernière sauvegarde complète

✓Indépendante des autres sauvegardes différentielles

=> Impossible de créer une sauvegarde différentielle si aucune sauvegarde complète n'a jamais été effectuée


```
BACKUP DATABASE AdventureWorks2008R2
TO DISK = 'L:\SQLBackups\AW_Diff.bak'
WITH DIFFERENTIAL, INIT;
```

- Sauvegarde et restauration

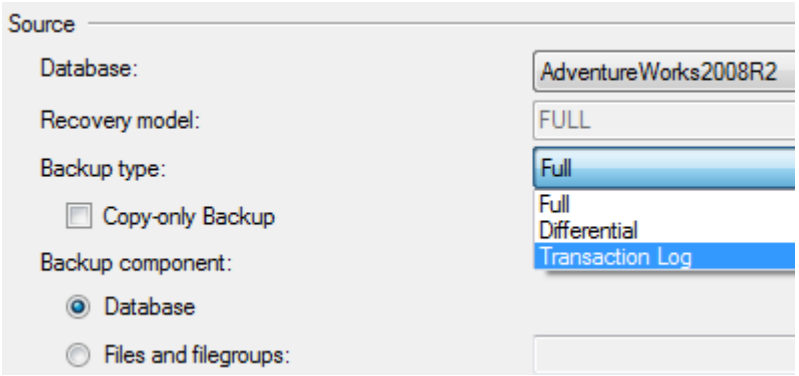
Sauvegarde du journal de transactions

✓**Sauvegarde du journal de transactions**

✓Sauvegarder le journal à partir de la dernière sauvegarde du journal avec succès jusqu'à la fin du journal courant.

✓Vider les transaction inactives

=> La base de données doit être en mode de récupération complet ou journalisé en bloc




```
BACKUP LOG AdventureWorks2008R2
TO DISK = 'L:\SQLBackups\AW_Log.bak'
WITH NOINIT;
```

- Sauvegarde et restauration

Gestion des sauvegardes de bases de données

✓**Option de sauvegarde CHECKSUM**

- Disponible pour tout type de sauvegarde
- Peut être utilisée pour vérifier la sauvegarde

✓**Vérification de la sauvegarde**

- RESTORE VERIFYONLY peut être utilisée pour vérifier la sauvegarde
- Utile lorsqu'elle est combinée avec l'option CHECKSUM

✓**Copie-seulement de sauvegarde**

- Ne change pas l'ordre de restauration
- N'affecte pas les sauvegardes différentielles dans le cas d'une copie-seulement d'une sauvegarde complète
- La copie-seulement de sauvegarde du journal de transaction ne purge pas ce dernier.

Atelier 14

Exercices :

1. Sauvegarder la base LA300 de manière complète ?

Comprendre le processus de restauration

Types de restauration :

- ✓ Restauration complète en mode de récupération simple
- ✓ Restauration complète en mode de récupération complète
- ✓ Restauration de bases de données système
- ✓ Restauration des fichiers endommagés seulement
- ✓ Restauration avancée avec les options Online, Piecemeal et Page restore

Préparer la restauration de sauvegardes

- Sauvegarde et restauration
  - ✓ Identifier les sauvegardes à restaurer
    - Dernière sauvegarde complète
    - Dernière sauvegarde différentielle
    - Toutes les sauvegardes du journal de transactions (si mode de récupération complet ou journalisé en bloc)
  - ✓ Les étapes de restauration:
    - Copier les données à partir des sauvegardes
    - Lancer les transactions à partir du journal jusqu'à atteindre un point de récupération cible
  - ✓ Les étapes de restauration s'appellent **séquences de restauration**

Exemple de séquences de restauration

- Sauvegarde et restauration

✓Sauvegardes planifiées
- Sauvegarde hebdomadaire complète samedi à 22h
- Sauvegarde différentielle Lundi, Mardi, jeudi, vendredi à 22h
- Sauvegarde du journal de transactions toutes les heures de 9h à 18h

✓Un échec au niveau de la base se produit jeudi à 10h30

✓Comment récupérer la base de données ?

Exemple de séquences de restauration (solution)

- Sauvegarde et restauration
  - ✓Créez une sauvegarde du **fichier** journal après défaillance
  - ✓Restaurez la sauvegarde complète de samedi soir (22h)
  - ✓Restaurez la sauvegarde différentielle de mardi soir (22h)
  - ✓Restaurez toutes les sauvegardes du journal de transactions de mercredi
  - ✓Restaurez les sauvegardes du journal de transactions de jeudi (09h et 10h)
  - ✓Restaurez la sauvegarde du **fichier** journal après défaillance si elle a été effectuée avec succès

- Sauvegarde et restauration

L'option de restauration WITH RECOVERY

✓Une base de données doit être récupérée avant d'être mise en ligne

✓L'option de restauration WITH RECOVERY (par défaut) permet la récupération de la base de données après sa restauration et la met en ligne

✓L'option de restauration WITH NORECOVERY passe la base de données au statut « en récupération »

=> Permet d'effectuer des opérations de restauration supplémentaires

✓Le processus de de restauration implique :

⇒ WITH NORECOVERY pour la restauration de toutes les sauvegardes sauf la dernière

⇒ WITH RECOVERY pour la restauration de la dernière sauvegarde

- Sauvegarde et restauration

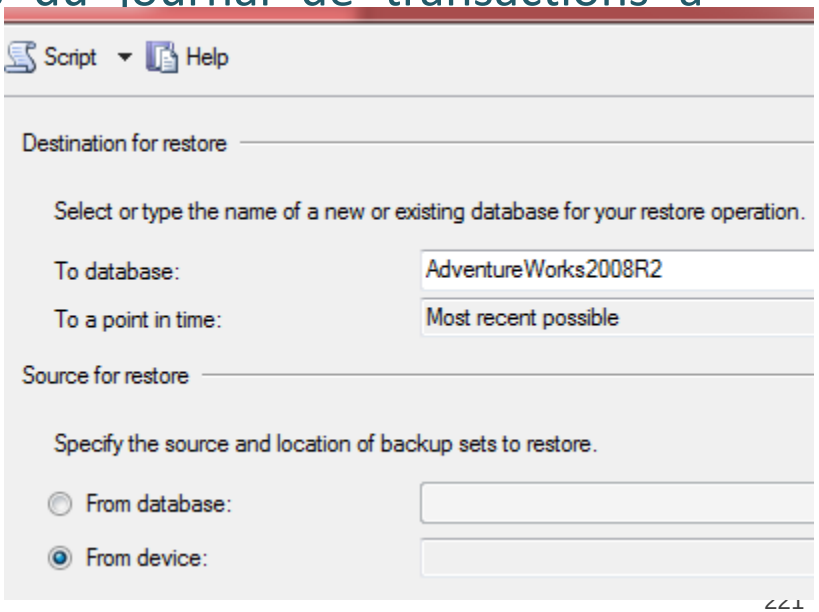
Restaurer une base de données

✓Les sauvegardes complètes et différentielles sont restaurées avec la commande RESTORE DATABASE

✓Seulement la dernière sauvegarde différentielle doit être restaurée

✓Restaurer chaque sauvegarde dans l'ordre avec l'option WITH NORECOVERY s'il reste des sauvegardes du journal de transactions à restaurer

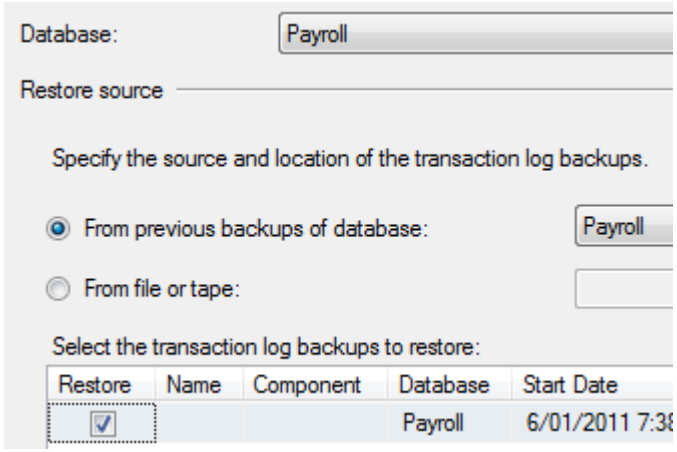
```
RESTORE DATABASE
AdventureWorks2008R2
FROM DISK =
'D:\SQLBackups\AW.bak'
WITH RECOVERY;
GO
```

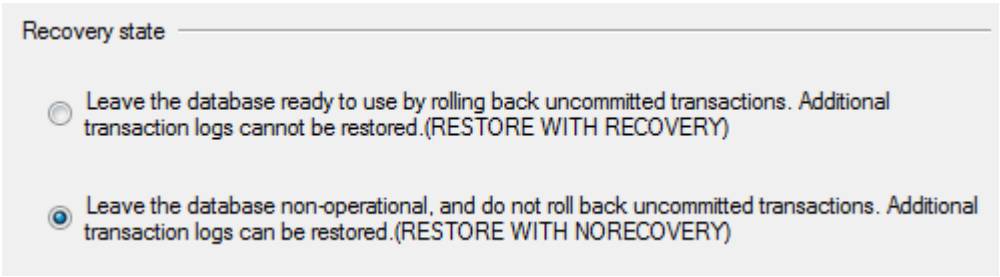


- Sauvegarde et restauration

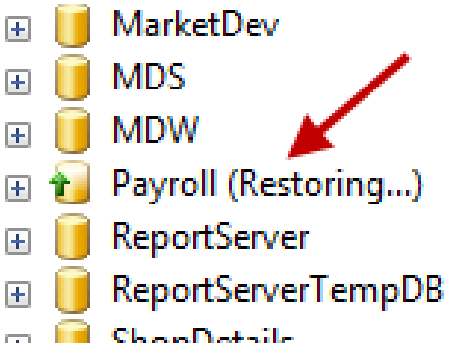
Restaurer un journal de transactions

✓Les sauvegardes du journal de transactions sont restaurées avec la commande RESTORE LOG






```
RESTORE LOG Payroll
FROM DISK =
'D:\SQLBackups\PayrollLogs.bak'
WITH NORECOVERY;
GO
```



Restauration des bases de données système

- Sauvegarde et restauration

Base de données	Description
master	Sauvegarde : Oui Mode de récupération : Simple À restaurer en mode utilisateur unique
model	Sauvegarde : Oui Mode de récupération : configurable par l'utilisateur À restaurer en utilisant la trace T3608
msdb	Sauvegarde : Oui Mode de récupération : Simple (default) À restaurer en mode base de données utilisateur
tempdb /resource	Sauvegardes impossibles tempdb est crée au démarrage du service SQL Restaurer la base resource en utilisant le SETUP

Atelier 14

Exercices :

1. Supprimer la base LA300 ?
2. Restaurer la base LA300 de manière complète ?

Autres opérations utiles

Détacher / attacher une base de données

Les Index

La commande UPDATE permet de modifier les valeurs d'un ou plusieurs champs, dans une ou plusieurs lignes existantes d'une table.

```
UPDATE table
SET col1 = exp1, col2 = exp2, ...
WHERE prédicat
ou
UPDATE table
SET (col1, col2,...) = (SELECT ...)
WHERE prédicat
```

table est le nom de la table mise à jour ; *col1, col2*, ... sont les noms des colonnes qui seront modifiées ; *exp1, exp2*,... sont des expressions.

Elles peuvent aussi être un ordre SELECT renvoyant les valeurs attribuées aux colonnes (deuxième variante de la syntaxe).

Les valeurs de *col1, col2*... sont mises à jour dans toutes les lignes satisfaisant le prédicat.

La clause WHERE est facultative. Si elle est absente, toutes les lignes sont mises à jour.