SQL

LE LANGAGE D'INTERROGATION DES DONNEES

LID

Les Jointures

Synthèse de données

Requêtes élaborées

LE LANGAGE DE MANIPULATION DES DONNEES

LMD

LE LANGAGE DE MANIPULATION DES DONNEES

LMD Elaboré

LE LANGAGE DE DEFINITION DE DONNEES

LDD

Les Vues

Les procédures stockées

LE LANGAGE D'INTERROGATION DES DONNEES

LID

Utilisation de l'instruction SELECT

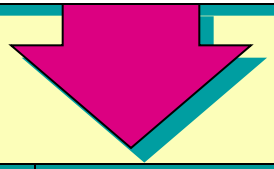
- **La liste de sélection spécifie les colonnes**
- **La clause WHERE spécifie les lignes**
- **La clause FROM spécifie la table**

Syntaxe partielle

```
SELECT [ALL | DISTINCT] <liste_sélection>
 FROM {<table_source>} [,…n]
 WHERE <condition_recherche>
```

Spécification de colonnes

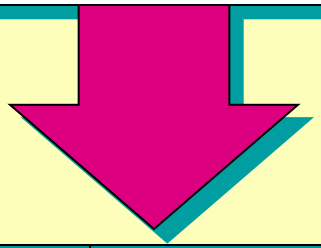
```
SELECT employeeid, lastname, firstname, title
FROM employees
```



employeeid	lastname	firstname	title
1	Davolio	Nancy	Sales Representative
2	Fuller	Andrew	Vice President, Sales
3	Leverling	Janet	Sales Representative
4	Peacock	Margaret	Sales Representative
5	Buchanan	Steven	Sales Manager
6	Suyama	Michael	Sales Representative
7	King	Robert	Sales Representative
8	Callahan	Laura	Inside Sales Coordinator
9	Dodsworth	Anne	Sales Representative

Utilisation de la clause WHERE pour spécifier des lignes

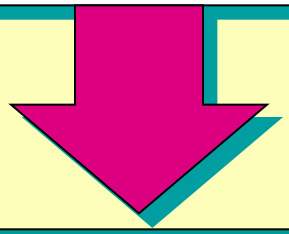
```
SELECT employeeid, lastname, firstname, title
FROM employees
WHERE employeeid = 5
```



employeeid	lastname	firstname	title
5	Buchanan	Steven	Sales Manager

Utilisation de comparaisons de chaînes

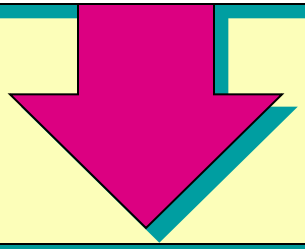
```
SELECT companyname
FROM customers
WHERE companyname LIKE '%Restaurant%'
```



companyname
GROSELLA-Restaurante
Lonesome Pine Restaurant
Tortuga Restaurante

Utilisation d'opérateurs logiques

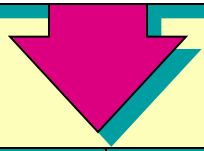
```
SELECT productid, productname, supplierid, unitprice
FROM products
WHERE (productname LIKE 'T%' OR productid = 46) AND
      (unitprice > 16.00)
```



productid	productname	supplierid	unitprice
14	Tofu	6	23.25
29	Thüringer Rostbratwurst	12	123.79
62	Tarte au sucre	29	49.3

Extraction d'une plage de valeurs

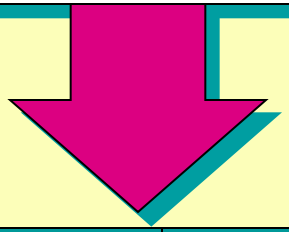
```
SELECT productname, unitprice
FROM products
WHERE unitprice BETWEEN 10 AND 20
```



productname	unitprice
Chai	18
Chang	19
Aniseed Syrup	10
Genen Shouyu	15.5
Pavlova	17.45
Sir Rodney’s Scones	10
.	.
.	.
.	.

Utilisation d'une liste de valeurs en tant que critère de recherche

```
SELECT companyname, country
FROM suppliers
WHERE country IN ('Japan', 'Italy')
```



companyname	country
Tokyo Traders	Japan
Mayumi’s	Japan
Formaggi Fortini s.r.l.	Italy
Pasta Buttini s.r.l.	Italy

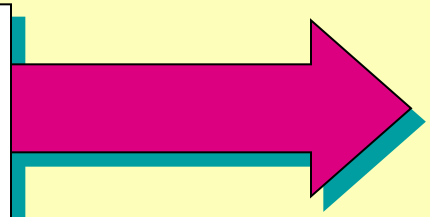
Tri des données

```
SELECT productid, productname, categoryid, unitprice
FROM products
ORDER BY categoryid, unitprice DESC
```

productid	productname	categoryid	unitprice
38	Cote de Blaye	1	263.5
43	Ipoh Coffee	1	46
2	Chang	1	19
.			
.			
.			
63	Vegie-spread	2	43.9
8	Northwoods Cranberry Sauce	2	40
61	Sirop d'érable	2	28.5
.			
.			
.			

Suppression des doublons : DISTINCT

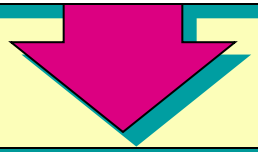
```
SELECT DISTINCT country
FROM suppliers
ORDER BY country
```



country
Australia
Brazil
Canada
Denmark
Finland
France
Germany
Italy
Japan
Netherlands
Norway
Singapore
Spain
Sweden
UK
USA

Modification des noms de colonne

```
SELECT firstname AS first, lastname AS last,
       employeeid AS 'employee ID:'
FROM employees
```



first	last	employee id:
Nancy	Davolio	1
Andrew	Fuller	2
Janet	Leverling	3
Margaret	Peacock	4
Steven	Buchanan	5
Michael	Suyama	6
Robert	King	7
Laura	Callahan	8
Anne	Dodsworth	9

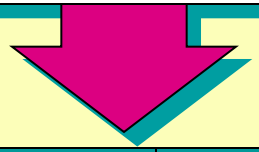
Calculs

```
SELECT productid, productname, categoryid,
       unitprice + 1 as newprice
FROM products
```

productid	productname	categoryid	newprice
38	Cote de Blaye	1	264.5
43	Ipoh Coffee	1	47
2	Chang	1	20
.			
.			
.			
63	Vegie-spread	2	44.9
8	Northwoods Cranberry Sauce	2	41
61	Sirop d'érable	2	29.5
.			
.			
.			

Utilisation de littéraux

```
SELECT firstname, lastname,
       'Identification number:', employeeid
FROM employees
```



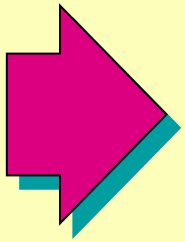
firstname	lastname	employeeid	
Nancy	Davolio	Identification number:	1
Andrew	Fuller	Identification number:	2
Janet	Leverling	Identification number:	3
Margaret	Peacock	Identification number:	4
Steven	Buchanan	Identification number:	5
Michael	Suyama	Identification number:	6
Robert	King	Identification number:	7
Laura	Callahan	Identification number:	8
Anne	Dodsworth	Identification number:	9

Transposition (les lignes sont des colonnes)

```
SELECT emp_no
         ,(CASE WHEN dept_no="d001" then 1 ELSE 0 END) d001
         ,(CASE WHEN dept_no="d002" then 1 ELSE 0 END) d002

FROM dept_emp
```

emp_no	dept_no
10019	d002
10020	d002
10021	d001
10022	d002



emp_no	d001	d002
10019	0	1
10020	0	1
10021	1	0
10022	0	1

En cas de besoin il est possible d'avoir :

```
Sum ((CASE WHEN dept_no="d001" then 1 ELSE 0 END)) d001
```

Les Jointures

Introduction aux Jointures

- **Sélection de colonnes à partir d'une ou de plusieurs tables**
  - Le mot clé JOIN spécifie que les tables sont jointes et comment les joindre
  - Le mot clé ON spécifie la condition de jointure
- **Interrogation de deux tables ou plus pour produire un ensemble de résultats**
  - Utiliser les clés primaires et étrangères comme conditions de jointure
  - Utiliser les colonnes communes aux tables spécifiées pour joindre des tables

Utiliser les jointures internes

```
SELECT buyer_name, sales.buyer_id, qty
FROM buyers  INNER JOIN sales
ON buyers.buyer_id = sales.buyer_id
```

Example 1

buyers

buyer_name	buyer_id
Adam Barr	1
Sean Chai	2
Eva Corets	3
Erin O'Melia	4

sales

buyer_id	prod_id	qty
1	2	15
1	3	5
4	1	37
3	5	11
4	2	1003

Result

buyer_name	buyer_id	qty
Adam Barr	1	15
Adam Barr	1	5
Erin O'Melia	4	37
Eva Corets	3	11
Erin O'Melia	4	1003

Utiliser les jointures externes

```
SELECT buyer_name, sales.buyer_id, qty
 FROM buyers  LEFT OUTER JOIN sales
  ON buyers.buyer_id = sales.buyer_id
```

Example 1

buyers

buyer_name	buyer_id
Adam Barr	1
Sean Chai	2
Eva Corets	3
Erin O'Melia	4

sales

buyer_id	prod_id	qty
1	2	15
1	3	5
4	1	37
3	5	11
4	2	1003

Result

buyer_name	buyer_id	qty
Adam Barr	1	15
Adam Barr	1	5
Erin O'Melia	4	37
Eva Corets	3	11
Erin O'Melia	4	1003
Sean Chai	NULL	NULL

Joindre plus de deux tables

```
SELECT buyer_name, prod_name, qty
 FROM buyers
 INNER JOIN sales
  ON buyers.buyer_id = sales.buyer_id
 INNER JOIN produce
  ON sales.prod_id = produce.prod_id
```

Example 1

buyers

buyer_i	buyer_name
1	Adam Barr
2	Sean Chai
3	Eva Corets
4	Erin O'Melia

sales

buyer_id	prod_id	qty
1	2	15
1	3	5
3	1	37
4	5	11
2	2	1003

produce

prod_id	prod_name
1	Apples
2	Pears
3	Oranges
4	Bananas
5	Peaches

Result

buyer_name	prod_name	qty
Erin O'Melia	Apples	37
Adam Barr	Pears	15
Erin O'Melia	Pears	1003
Adam Barr	Oranges	5
Eva Corets	Peaches	11

Joindre une table avec elle même

```
SELECT a.buyer_id AS buyer1, a.prod_id
      ,b.buyer_id AS buyer2
 FROM sales AS a
 JOIN sales AS b
  ON a.prod_id = b.prod_id
WHERE  a.buyer_id > b.buyer_id
```

Example 3

sales a

buyer_id	prod_id	qty
1	2	15
1	3	5
4	1	37
3	5	11
4	2	1003

sales b

buyer_id	prod_id	qty
1	2	15
1	3	5
4	1	37
3	5	11
4	2	1003

Result

buyer1	prod_id	buyer2
4	2	1

Utiliser un alias pour nommer les tables

- **Exemple 1 (sans nom d'alias)**

```
SELECT buyer_name, sales.buyer_id, qty
FROM buyers  INNER JOIN sales
 ON buyers.buyer_id = sales.buyer_id
```

- **Exemple 2 (avec un nom d'alias)**

```
SELECT buyer_name, s.buyer_id, qty
 FROM buyers AS b  INNER JOIN sales AS s
  ON b.buyer_id = s.buyer_id
```

Synthèse de données

Utilisation de fonctions d'agrégation

Fonction d'agrégation	Description
AVG	Moyenne des valeurs dans une expression numérique
COUNT	Nombre de valeurs dans une expression
COUNT (*)	Nombre de lignes sélectionnées
MAX	Valeur la plus grande de l'expression
MIN	Valeur la plus petite de l'expression
SUM	Total des valeurs d'une expression numérique
STDEV	Écart-type pour toutes les valeurs
STDEVP	Écart-type de la population
VAR	Variance pour toutes les valeurs
VARP	Variance de la population pour toutes les valeurs

Utilisation de fonctions d'agrégation avec des valeurs NULL

- La plupart des fonctions d'agrégation ignorent les valeurs NULL
- La fonction COUNT(*) compte les lignes contenant des valeurs NULL

```
SELECT COUNT (*)
FROM employees
```

Utilisation de la clause GROUP BY

```
SELECT productid, orderid, quantity
FROM orderhist
```

productid	orderid	quantity
1	1	5
1	1	10
2	1	10
2	2	25
3	1	15
3	2	30

```
SELECT productid, SUM(quantity)
AS total_quantity
FROM orderhist
GROUP BY productid
```

productid	total_quantity
1	15
2	35
3	45

Seules les lignes qui satisfont à la clause WHERE sont regroupées

productid	total_quantity
2	35

```
SELECT productid, SUM(quantity)
AS total_quantity
FROM orderhist
WHERE productid = 2
GROUP BY productid
```

Utilisation de la clause GROUP BY avec la clause HAVING

```
SELECT productid, orderid, quantity
FROM orderhist
```

productid	orderid	quantity
1	1	5
1	1	10
2	1	10
2	2	25
3	1	15
3	2	30

```
SELECT productid, SUM(quantity)
AS total_quantity
FROM orderhist
GROUP BY productid
HAVING SUM(quantity) >=30
```

productid	total_quantity
2	35
3	45

Requêtes élaborées

Combinaison de plusieurs jeux de résultats

- **Utilisation de l'opérateur UNION pour créer un jeu de résultats unique à partir de plusieurs requêtes**
- **Chaque requête doit avoir :**
  - des données de même type
  - le même nombre de colonnes
  - le même ordre de colonnes dans la liste de sélection

```
SELECT name = (firstname + ' ' + lastname),
              city, postalcode
FROM employees
UNION
SELECT companyname, city, postalcode
FROM customers
```

Présentation des sous-requêtes

- **Raisons motivant l'utilisation de sous-requêtes**
  - Elle permettent de décomposer une requête complexe en une série d'étapes logiques
  - Elles permettent de répondre à une requête qui repose sur les résultats d'une autre requête
- **Raisons motivant l'utilisation de jointures à la place de sous-requêtes**
  - SQL Server exécute les jointures plus vite que les sous-requêtes

Utilisation de sous-requêtes

- Vous devez mettre les sous-requêtes entre parenthèses
- Vous ne pouvez utiliser qu'une seule expression ou un seul nom de colonne dans la liste de sélection
- Vous pouvez utiliser des sous-requêtes à la place d'une expression
- Vous ne pouvez utiliser de sous-requêtes dans les colonnes qui contiennent des données de type texte et image
- Le nombre de niveaux de sous-requêtes est illimité

Renvoi d'une valeur unique

- **La sous-requête remplace une expression dans :**
  - la liste de sélection
  - la clause WHERE introduite à l'aide d'un opérateur de comparaison

```
SELECT orderid, customerid
FROM orders
WHERE orderdate = (SELECT max(orderdate)
                   FROM orders)
```

Renvoi d'une liste de valeurs

- **La sous-requête remplace une expression dans :**
  - la clause WHERE introduite avec le mot clé IN

```
SELECT companyname
FROM customers
WHERE customerid IN
                (SELECT customerid
                 FROM orders
                 WHERE orderdate >'1/1/95')
```

Sous-requêtes corrélées

- La requête interne repose sur les informations provenant de la requête externe
- Utilisez des alias pour distinguer les tables
- Envisagez d'utiliser des jointures

Évaluation d'une sous-requête corrélée

1. La requête externe passe une valeur de colonne à la requête interne

2. La requête interne utilise les valeurs transmises par la requête externe

```
SELECT orderid, customerid
FROM orders or1
WHERE 20 < (SELECT quantity
                  FROM [order details] od
                    WHERE or1.orderid = od.orderid
                  AND od.productid = 23)
```

3. La requête interne renvoie une valeur à la requête externe

4. Ce processus se répète pour la valeur de colonne suivante de la requête externe

Retour à l'étape 1

Utilisation des mots clés EXISTS et NOT EXISTS

- **Utilisation avec des sous-requêtes corrélées**
- **Détermination de l'existence de données dans une liste de valeurs**
- **Traitement**
  - La requête externe vérifie l'existence des lignes
  - La requête interne renvoie la valeur True ou False

```
SELECT lastname, employeeid
FROM employees e
WHERE EXISTS (SELECT * FROM orders
              WHERE e.employeeid = orders.employeeid
              AND orderdate = '9/5/97')
```

Critère	EXISTS	IN
Fonction	Vérifie si une sous-requête retourne des lignes	Compare une valeur à une liste
Performance	Meilleur pour les sous-requêtes corrélées	Moins efficace avec de grands ensembles
Impact des NULL	Non affecté	Peut poser des problèmes
Utilisation typique	Vérification d'existence	Comparaison directe

La division

- La division permet de répondre aux questions du type : "Donnez toutes les personnes qui pratiquent tous les métiers de la relation métier".
- Soit les deux relations suivantes :
  - Pratiques (#Nom, #Métier, Salaire)
  - Métiers (#Metier)

Soit les données

Pratiques		
Dupont	Ingénieur	35
Durand	Professeur	40
Dupont	Ingénieur	45
Martin	Ingénieur	50

Métiers
Ingénieur
Professeur

Donnez toutes les personnes qui pratiquent tous les métiers de la relation métier

Solution

Dupont	35

```
SELECT DISTINCT Nom
FROM Pratiques P1
WHERE NOT EXISTS (
    SELECT Metier
    FROM Metiers M
    WHERE NOT EXISTS (
        SELECT *
        FROM Pratiques P2
        WHERE P2.Nom = P1.Nom
        AND P2.Cours = M.Cours
    )
);
```

LE LANGAGE DE MANIPULATION DES DONNEES

LMD

Insertion de lignes

```
INSERT INTO customers
       (customerid, companyname, contactname, contacttitle,
       address, city, region, postalcode, country, phone,
       fax)

VALUES ('PECOF', 'Pecos Coffee Company', 'Michael Dunn',
       'Owner', '1900 Oak Street', 'Vancouver', 'BC',
       'V3F 2K1', 'Canada', '(604) 555-3392',
       '(604) 555-7293')
```

Insertion de données à l'aide de valeurs par défaut

- **Mot clé DEFAULT**
  - Insère des valeurs par défaut pour les colonnes spécifiées
  - Les colonnes doivent contenir une valeur par défaut ou autoriser les valeurs NULL

```
INSERT INTO shippers (companyname, phone)
VALUES ('Kenya Coffee Co.', DEFAULT)
```

- **Mot clé DEFAULT VALUES**
  - Insère des valeurs par défaut pour toutes les colonnes
  - Les colonnes doivent contenir une valeur par défaut ou autoriser les valeurs NULL

Insertion de données partielles

Ajout de données

```
INSERT INTO shippers (companyname)
VALUES ('Fitch & Mather')
```

Vérification des données ajoutées

```
SELECT *
FROM shippers
WHERE companyname = 'Fitch & Mather'
```

Autorisation des valeurs NULL

shipperid	companyname	phone
37	Fitch & Mather	NULL

Suppression de lignes

- **Instruction DELETE**
  - Permet de supprimer une ou plusieurs lignes d'une table
  - Inclut toujours une clause WHERE
  - Chaque ligne supprimée est consignée dans le journal des transactions

Mise à jour de lignes

- La clause WHERE spécifie les lignes à modifier
- Le mot clé SET spécifie les nouvelles données
- Les valeurs entrées doivent être identiques aux types de données définis pour les colonnes

```
UPDATE products
SET unitprice = (unitprice * 1.1)
```

Performances

- Utilisez des conditions de recherche positives
- Évitez d'utiliser la condition de recherche LIKE
- Utilisez des correspondances ou des plages exactes
- La clause ORDER BY peut ralentir l'extraction des données

LE LANGAGE DE MANIPULATION DES DONNEES

LMD Elaboré

Utilisation de l'instruction INSERT … SELECT

- Toutes les lignes qui répondent aux conditions de l'instruction SELECT sont insérées
- Vérifiez que la table dans laquelle sont insérées les lignes existe
- Vérifiez que les types de données sont compatibles
- Déterminez si des valeurs par défaut existent ou si les valeurs nulles sont autorisées

```
INSERT customers
SELECT substring (firstname, 1, 3) +
       substring (lastname, 1, 2),
                 lastname, firstname, title,
                 address, city, region, postalcode,
                 country, homephone, NULL
FROM employees
```

Suppression de lignes à l'aide des données contenues dans d'autres tables

- **Utilisation d'une clause FROM supplémentaire**
  - La première clause FROM indique la table à modifier
  - La deuxième clause FROM spécifie les critères restrictifs de l'instruction DELETE
- **Spécification de conditions dans la clause WHERE**
  - Les sous-requêtes déterminent les lignes à supprimer

Mise à jour de lignes à l'aide des données contenues dans d'autres tables

- **Utilisation de l'instruction UPDATE**
  - Ne met à jour les lignes qu'une fois
  - Met à jour les colonnes ou noms de variables qui suivent le mot clé SET
- **Utilisation de jointures**
  - Utilise la clause FROM
- **Utilisation de sous-requêtes**
  - Renvoie une valeur unique par ligne

LE LANGAGE DE DEFINITION DE DONNEES

LDD

Types de Données

- **Chaînes de caractères:**
  - CHAR (taille)
  - VARCHAR (taille)
- **Nombres:**
  - NUMBER (chiffres, décimales)
  - INT
  - FLOAT
  - DOUBLE
- **Date**
  - DATE
  - TIMESTAMP
  - INTERVAL

Création d'un Table

- La clé primaire peut être crée au moment de la création de la table ou après
- La clé étrangère peut aussi être créée au moment de la création de la table ou après, mais la table de référence doit toujours exister avant.

```
CREATE TABLE [IF NOT EXISTS] Nom_table (
    colonne1 description_colonne1 PRIMARY KEY,
    [colonne2 description_colonne2,
    colonne3 description_colonne3,
    ...,]
    [ [CONSTRAINT [symbole_contrainte]]  FOREIGN KEY
(colonne(s)_clé_étrangère) REFERENCES table_référence
(colonne(s)_référence)]
)
```

Création des tables salaires et employee

```
CREATE TABLE employees (
    emp_no integer NOT NULL,
    birth_date date,
    first_name character varying(32),
    last_name character varying(32),
    gender character varying(32),
    hire_date date
    CONSTRAINT pk_employees PRIMARY KEY (emp_no)
);
```

```
CREATE TABLE salaries (
    emp_no integer NOT NULL,
    salary bigint,
    start_date date NOT NULL,
    end_date date,
    CONSTRAINT pk_salaries PRIMARY KEY (emp_no, start_date),
    CONSTRAINT fk_salaries_employees FOREIGN KEY (emp_no)
       REFERENCES employees(emp_no)
);
```

Opérations sur les tables

- **Supprimer une table,**

```
DROP TABLE <nom_table>;
```

- **Renommer une table,**

```
RENAME <ancien_nom_table> TO <nouveau_nom_table>;
```

- **Modifier un champs**

```
ALTER TABLE <nom_table> ADD <champs> <type>;
```

```
ALTER TABLE <nom_table> DROP <champs>;
```

Création d'un Table à partir d'un SELECT

```
CREATE TABLE [IF NOT EXISTS] Nom_table AS
 SELECT [ALL | DISTINCT] <liste_sélection>
 FROM {<table_source>} [,…n]
 WHERE <condition_recherche>
```

Ordres divers sur une base de données

- **Autorisation : GRANT**
  - Gère les autorisations sur une table ou une vue
  - Tout type de manipulations : insert, select, update ou all
- **Modification de structure : la clause ALTER**
- **Suppression de tables ou de vues : DROP**

Les Vues

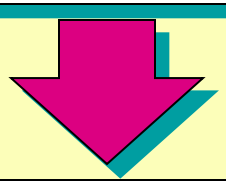
Création d’une vue

- **La syntaxe de création d'une vue est la suivante:**

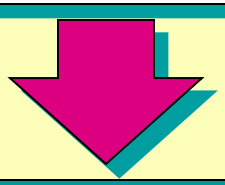
```
CREATE VIEW nom_de_la_vue AS
SELECT columns
FROM tables
WHERE conditions
GROUP BY conditions
HAVING conditions
ORDER BY conditions
```

Création et Utilisation

```
CREATE VIEW FirstView AS
SELECT productid, productname, supplierid, unitprice
FROM products
```



```
SELECT * FROM FirsView
WHERE (productname LIKE 'T%' OR productid = 46) AND
      (unitprice > 16.00)
```



productid	productname	supplierid	unitprice
14	Tofu	6	23.25
29	Thüringer Rostbratwurst	12	123.79
62	Tarte au sucre	29	49.3

Avantages liés aux vues

- **Mise en valeur des données pour les utilisateurs**
  - Mettre en valeur les données importantes ou appropriées seulement
  - Limiter l'accès aux données sensibles
- **Masquage de la complexité de la base de données**
  - Masquer la conception de bases de données complexes
  - Simplifier les requêtes complexes, notamment les requêtes distribuées sur des données hétérogènes
- **Simplification de la gestion des autorisations des utilisateurs**
- **Organisation des données pour les exporter vers d'autres applications**

Extraction de données à partir d'une vue

- **L'extraction de données à partir d'une vue est similaire à l'extraction à partir d'une table.**
- **Le moteur SQL, grâce à l'interface contenue dans le dictionnaire, convertit cette requête sur la vue en une requête équivalente sur la table de base qu'il peut parfaitement interpréter et exécuter pour obtenir le résultat.**

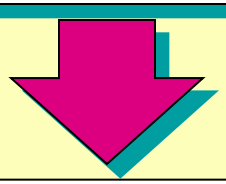
Mise à jour à partir d'une vue

- **Utiliser Insert, Update, Delete dans une vue**
- **Pour l'utilisateur, tout se passe comme s’il s'agissait d'une table de base. La vue peut lui permettre aussi d'agir directement sur le contenu des tables.**
- **La norme SQL-92 dénombre 6 règles à respecter par une vue pour que celle-ci soit modifiable. Les différents SGBD dérogent plus ou moins à ces règles. Selon la norme, si une vue n'est pas modifiable, c'est l'ensemble des opérations update, insert et delete qui ne sont pas permises. Ces règles sont :**

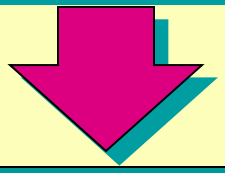
1. **L'expression de la table associé à la vue doit être un simple select. elle ne peut donc contenir les termes join, intersect, union ou except;**
2. **La clause from ne peut contenir qu'une seule table de base ou une vue elle-même modifiable;**
3. **L'expression select ne peut contenir la clause distinct;**
4. **La liste des colonnes du select ne peut contenir la clause distinct;**
5. **si le select contient une requête imbriquée, celle-ci ne peut faire référence à la même table que la requête externe;**
6. **La requête select ne peut contenir ni group by, ni having.**

Création et Utilisation

```
CREATE VIEW SimpleOrders AS
SELECT OrderID, CustomerID, OrderDate
FROM Orders;
```



```
INSERT INTO SimpleOrders (OrderID, CustomerID, OrderDate)
VALUES (11078, 'ALFKI', '2023-09-18');
```



OrderID	CustomerID	OrderDate
11078	ALFKI	2023-09-18

Remarques

- a. Vue basée sur une seule table : L'insertion ne fonctionnera que si la vue est basée sur une seule table.
- b. Colonnes obligatoires : Assurez-vous d'inclure toutes les colonnes NOT NULL de la table sous-jacente qui n'ont pas de valeur par défaut.
- c. Contraintes de la vue : Si la vue a été créée avec WITH CHECK OPTION, l'insertion doit respecter les conditions de la vue (dans notre exemple, le département doit être 'Marketing').
- d. Colonnes calculées : Les vues contenant des colonnes calculées ou des agrégations ne permettent généralement pas l'insertion directe.
- e. Colonnes non incluses : Si la vue ne contient pas toutes les colonnes de la table sous-jacente, ces colonnes doivent soit autoriser NULL, soit avoir une valeur par défaut.

Modification d’une vue

- **La syntaxe de modification d'une vue est la suivante:**

```
ALTER VIEW nom_de_la_vue AS
SELECT columns
FROM tables
WHERE conditions;
```

Modification d’une vue

```
Alter VIEW SimpleOrders AS
SELECT OrderID, CustomerID, OrderDate,
            p.ProductName -- Colonne ajoutée
FROM Orders
JOIN -- Tables ajoutées
    [Order Details] od ON o.OrderID = od.OrderID
JOIN
    Products p ON od.ProductID = p.ProductID
WHERE
    o.OrderDate > '2020-01-01';        -- Contrainte ajoutée
```

Destruction d'une vue

- **La syntaxe de suppression d'une vue est la suivante:**

```
drop view nom_de_la_vue {restrict cascade}
```

- **restrict spécifie que si la vue intervient dans la définition d'une autre vue ou dans une contrainte d'intégrité, la commande sera rejeté.**
- **cascade spécifie que la vue sera supprimée ainsi que toutes les vues et contraintes où la vue intervient.**

Les procédures stockées

Définir une procédure stockée

- **Une procédure stockée est une collection précompilée d'instructions**
- **Une procédure stockée encapsule des tâches répétitives**
- **Les procédures stockées peuvent:**
  - Contenir les instructions qui effectuent des opérations
  - Accepter les paramètres d'entrée
  - Renvoyer la valeur du statut pour indiquer le succès ou l'échec
  - Renvoyer plusieurs paramètres de sortie

Avantages de l'utilisation de procédures stockées

- Partager la logique d'application
- Détails du schéma de base de données
- Fournir des mécanismes de sécurité
- Améliorer les performances
- Réduire le trafic réseau

Introduction aux déclencheurs (Triggers)

- **Un déclencheur est un type spécial de procédure stockée**
- **Un déclencheur est:**
  - Associé à une table
  - Invoqué automatiquement
  - Non appelé directement
  - Traité dans le cadre de la transaction qui l'a déclenché

Qu'est-ce qu'une fonction définie par l'utilisateur?

- **Fonctions scalaires**
  - Semblable à une fonction intégrée
  - Renvoie une seule valeur de données générée par une série d'instructions
- **Fonctions de table à plusieurs instructions**
  - Contenu comme une procédure stockée
  - Référencé comme une vue
- **Fonctions de table en ligne**
  - Similaire à une vue avec des paramètres
  - Renvoie une table à la suite de l'instruction SELECT unique

Création d'une fonction définie par l'utilisateur

```
USE northwind
GO
CREATE FUNCTION fn_NewRegion ( @myinput nvarchar(30) )
  RETURNS nvarchar(30)
BEGIN
  IF @myinput IS NULL
   SET @myinput = 'Not Applicable'

  RETURN @myinput
END
GO
```