SQL

Traitement des résultat

Fonctions sur les colonnes

- **Attributs calculés**
  - Exemple : SELECT nom, population*1000/surface FROM Pays

- **Opérateurs sur attributs numériques**
  - SUM: somme des valeurs des tuples sélectionnés
  - AVG: moyenne

- **Opérateurs sur tous types d'attributs**
  - MIN: minimum
  - MAX: maximum
  - COUNT: nombre de tuples sélectionnés

Opérateurs d'agrégation

Opérateurs d'agrégation

pays

Nom	Capitale	Population	Surface	Continent
Irlande	Dublin	5	70	Europe
Autriche	Vienne	10	83	Europe
UK	Londres	50	244	Europe
Suisse	Berne	7	41	Europe
USA	Washington	350	441	Amérique

```
SELECT MIN(population), MAX(population), AVG(population),
SUM(surface), COUNT(*)
FROM Pays WHERE continent = 'Europe'
```

Donne le résultat :

MIN(population)	MAX(population)	AVG(population)	SUM(surface)	COUNT(*)
5	50	18	438	4

DISTINCT

pays

Nom	Capitale	Population	Surface	Continent
Irlande	Dublin	5	70	Europe
Autriche	Vienne	10	83	Europe
UK	Londres	50	244	Europe
Suisse	Berne	7	41	Europe
USA	Washington	350	441	Amérique

Suppression des doubles

```
SELECT DISTINCT continent
FROM Pays
```

Donne le résultat :

Continent
Europe
Amérique

ORDER BY

Tri des tuples du résultat

```
SELECT continent, nom, population
FROM Pays
WHERE surface > 60
ORDER BY continent, nom ASC
```

2 possibilités : ASC / DESC

Continent	Nom	Population
Amérique	USA	350
Europe	Autriche	10
Europe	Irlande	5
Europe	UK	50

GROUP BY

Partition de l'ensemble des tuples en groupes homogènes:

```
SELECT continent, MIN(population), MAX(population),AVG(population),
SUM(surface), COUNT(*)
FROM Pays GROUP BY continent ;
```

Continent	MIN(population)	MAX(population)	AVG(population)	SUM(surface)	COUNT(*)
Europe	5	50	18	438	4
Amérique	350	350	350	441	1

A noter : cette commande doit toujours s'utiliser après la commande **WHERE** et avant la commande HAVING.

HAVING

Conditions sur les fonctions d'agrégation

- Il n'est pas possible d'utiliser la clause WHERE pour faire des conditions sur une fonction d'agrégation.
- Donc, si l'on veut afficher les pays dont on possède plus de 3 individus**, la requête suivante ne fonctionnera pas.**

```
SELECT continent, COUNT(*)
FROM Pays
WHERE COUNT(*) > 3
GROUP BY continent ;
```

- Il faut utiliser HAVING qui se place juste après le GROUP BY

```
SELECT continent, COUNT(*)
FROM Pays
GROUP BY continent
HAVING COUNT(*) > 3;
```

Renommage des attributs : **AS**

```
SELECT MIN(population) AS min_pop,
MAX(population) AS max_pop,
AVG(population) AS avg_pop,
SUM(surface) AS sum_surface,
COUNT(*) AS count
FROM Pays
WHERE continent = 'Europe' ;
```

min_pop	max_pop	avg_pop	sum_surface	count
5	50	18	438	4

Opérateurs ensemblistes en SQL

1. UNION

Syntaxe:

```
requête_SELECT
UNION [ ALL] requête_SELECT
[ UNION [ ALL] requête_SELECT  ….]
```

L'opérateur UNION supprime les données redondantes par défaut, sauf si l'option ALL est explicité.

Exemple: lister tous les clients appartenant aux tables CLIENT et CLIENT_CASA.

```
SELECT idclient , nom, tel
FROM  Client
UNION
SELECT idclient , nom, tel
FROM  Client_CASA;
```

Opérateurs ensemblistes en SQL

1. UNION ALL

Syntaxe:

```
SELECT * FROM table1
UNION ALL
SELECT * FROM table2
```

- cette commande permet d'inclure tous les enregistrements, même **les doublons**.
- Ainsi, si un même enregistrement est présents normalement dans les résultats des 2 requêtes concaténées, alors l'union des 2 avec **UNION ALL** retournera 2 fois ce même résultat.
- **Exemple:**

```
SELECT * FROM magasin1_client
UNION ALL
SELECT * FROM magasin2_client
```

Opérateurs ensemblistes en SQL

1. UNION ALL

magasin1_client

prenom	nom	ville	date_naissance	total_achat
Léon	Dupuis	Paris	1983-03-06	135
Marie	Bernard	Paris	1993-07-03	75
Sophie	Dupond	Marseille	1986-02-22	27
Marcel	Martin	Paris	1976-11-24	39

magasin2_client

prenom	nom	ville	date_naissance	total_achat
Marion	Leroy	Lyon	1982-10-27	285
Paul	Moreau	Lyon	1976-04-19	133
Marie	Bernard	Paris	1993-07-03	75
Marcel	Martin	Paris	1976-11-24	39

Résultat:

prenom	nom	ville	date_naissance	total_achat
Léon	Dupuis	Paris	1983-03-06	135
Marie	Bernard	Paris	1993-07-03	75
Sophie	Dupond	Marseille	1986-02-22	27
Marcel	Martin	Paris	1976-11-24	39
Marion	Leroy	Lyon	1982-10-27	285
Paul	Moreau	Lyon	1976-04-19	133
Marie	Bernard	Paris	1993-07-03	75
Marcel	Martin	Paris	1976-11-24	39

Opérateurs ensemblistes en SQL

2. INTERSECTION

Syntaxe:

```
requête_SELECT
INTERSECT
requête_SELECT
```

Exemple: afficher les livres dont le prix est superieur à 500 et qui sont toujours empruntés.

```
SELECT  noliv
FROM livre l
WHERE l.prix>500
INTERSECT
SELECT  noliv
FROM emprunt e
WHERE e.retour IS NULL
```

Opérateurs ensemblistes en SQL

3. DIFFERENCE

Syntaxe:

```
requête_SELECT
MINUS
requête_SELECT
```

Exemple: trouver les livres non encore empruntés

```
SELECT noliv
FROM livre
MINUS
SELECT noliv
FROM emprunt
WHERE retour IS NULL
```

La Requête TRUNCATE en SQL

- ❑ En SQL, la commande **TRUNCATE** permet de supprimer toutes les données d'une table sans supprimer la table en elle-même.

- ❑ Cette instruction diffère de la commande **DROP** qui à pour but de supprimer les données ainsi que la table qui les contient.

- ✓ **A noter :** l'instruction **TRUNCATE** est semblable à l'instruction **DELETE** sans utilisation de WHERE. Parmi les petite différences TRUNCATE est toutefois plus rapide et utilise moins de ressource.

- ✓ Ces gains en performance se justifie notamment parce que la requête n'indiquera pas le nombre d'enregistrement supprimés et qu'il n'y aura pas d'enregistrement des modifications dans le journal.

La Requête TRUNCATE en SQL

Syntaxe:

- Cette instruction s'utilise dans une requête SQL semblable à celle-ci :

```
TRUNCATE TABLE `table`
```

- Dans cet exemple, les données de la table "table" seront perdues une fois cette requête exécutée.

La Requête TRUNCATE en SQL

Exemple

Table “fourniture” :

id	nom	date_ajout
1	Ordinateur	2023-04-05
2	Chaise	2023-04-14
3	Bureau	2023-07-18
4	Lampe	2023-09-27

- Il est possible de supprimer toutes les données de cette table en utilisant la requête:

```
TRUNCATE TABLE `fourniture`
```

- Une fois la requête exécutée, la table ne contiendra plus aucun enregistrement. En d’autres mots, toutes les lignes du tableau présenté ci-dessus auront été supprimées.

La Requête TRUNCATE en SQL

Différence entre TRUNCATE et DELETE

- La commande **TRUNCATE** s'avère être similaire à la commande **DELETE**, lorsqu'elle est utilisée de la façon suivante :

```
DELETE FROM `fourniture`
```

- Il y a toutefois une différence notable : la commande **TRUNCATE** va **ré-initialiser la valeur de l'auto-incrément**, s'il y en a un.

- Dès lors, il faut potentiellement noter la valeur de l'auto-incrément avant de faire un **TRUNCATE**, surtout s'il faut ré-indiquer l'ancienne valeur après avoir écrasé toutes les données.

SQL

Les Vues

Les Vues en SQL

- **Une vue** est une construction logique (table virtuelle) faites à partir de tables existantes (tables de base);

- elle ne contient aucune données en soit, **elle n'est qu'une représentation indirecte de données contenues dans d'autres tables;**

- les vues sont très utilisées pour simplifier et optimiser l'usage de structures intermédiaires souvent sollicitées;

- elles sont constamment mises à jour par le SGBD.

Les Vues en SQL

Création par:

CREATE VIEW *nom* (renommage facultatif des colonnes) *AS*
requête SELECT

Syntaxe générale:

CREATE [OR REPLACE]
[FORCE | NOFORCE] VIEW
nom-de-vue [(attr1, …, attrn)]
AS *requête*
[WITH CHECK OPTION
[CONSTRAINT *nom-contrainte]]*
[WITH READ ONLY]

Les Vues en SQL

- **L'instruction *OR REPLACE*** crée à nouveau la vue si elle existe déjà.
- **Les clauses *FORCE* et *NOFORCE*** indiquent respectivement que :
  - la vue est créée sans se soucier de l'existence des tables qu'elle référence ou des privilèges adéquats sur les tables,
  - la vue est créée seulement si les tables existent et si les permissions requises sont données.
- **La clause *WITH CHECK OPTION*** limite les insertions et les mises à jour exécutées par l'intermédiaire de la vue.
- **La clause *CONSTRAINT*** est un nom optionnel donné à la contrainte *WITH CHECK OPTION*.

Les Vues en SQL

- **Les vues peuvent être créées à partir d'autres vues.** Pour cela il suffit de référencer les vues dans la clause *FROM* de l'instruction *select*.

```
CREATE VIEW nom_vue
AS
SELECT * FROM nom_vue2;
```

Les Vues en SQL

Exemples:

```
CREATE VIEW rbati
AS  SELECT enum, ename
    FROM employee
    WHERE address="Rabat"
```

```
select * from rbati;

+------+-------+
| enum | ename |
+------+-------+
| E5   | Amina |
| E1   | Ali   |
+------+-------+
```

Avec renommage des attributs:

```
CREATE VIEW rbati  (numero, nom)
AS  SELECT enum, ename
    FROM employee
    WHERE address="Rabat"
```

```
select * from rbati;

+--------+-------+
| numero | nom   |
+--------+-------+
| E5     | Amina |
| E1     | Ali   |
+--------+-------+
```

Les Vues en SQL

Interrogation avec:

```
SELECT ..
FROM  nom_de_la_vue
WHERE ..
```

→comme pour les tables de base.

Exemple:

```
select nom
from rbati
where numero = 'E1‘;
```

Résultat:

```
+------+
| nom  |
+------+
| Ali  |
+------+
```

Les Vues en SQL

- Suppression d'une vue

  DROP VIEW ***nom-de-vue***

  →La suppression d'une vue n'entraîne pas la suppression des données

- Renommer une vue

  RENAME ***ancien-nom*** ***TO*** ***nouveau-nom***

Les Vues en SQL

Exemples 1:

- **CREATE VIEW** vue_personnel
  AS SELECT sexe, nom, prenom, adresse, cp, ville
  FROM tbl_personnel
  WHERE service = 'commercial'

- **CREATE VIEW** vue_pers_serv
  AS SELECT v.sexe, v.nom, v.prenom, t.poste, t.service
  FROM vue_personnel v, tbl_service t
  WHERE code_service = 'SC02354L' **GROUP BY** v.nom;

Les Vues en SQL

Utilisation d'une vue :

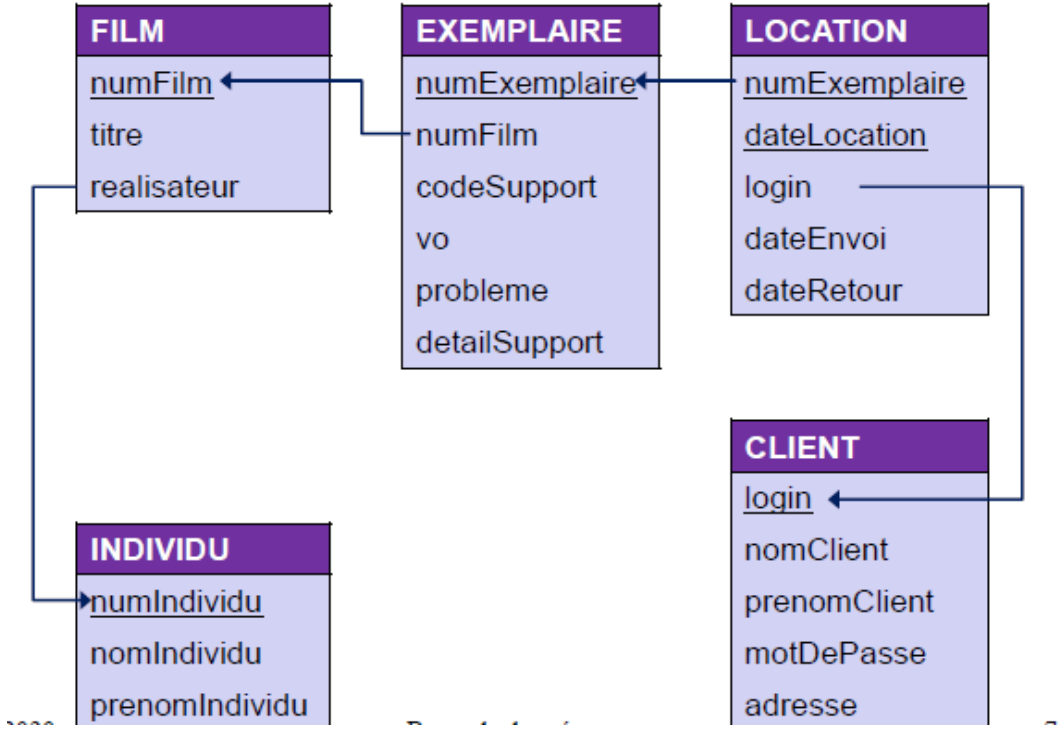
- **UPDATE** vue_pers_serv
  SET sexe = 'masculin' **WHERE** nom = 'Frédérique' **AND** prenom = 'Jean'

- **SELECT** v1.nom, v1.prenom, v2.service
  FROM vue_personnel AS v1, vue_services AS v2
  WHERE v1.id_service = v2.id_service

- **CREATE View** tbl_employes
  AS SELECT v1.id_service, v2.service, v1.nom, v1.prenom, v1.adresse
  FROM vue_personnel AS v1, **vue_services** AS v2
  WHERE v1.id_service = v2.id_service

Les Vues en SQL

Exemple 2:



Les Vues en SQL

Création de la vue:

- **CREATE OR REPLACE VIEW**
  exemplairePlus (num, titre, real, support)
  AS
  SELECT numExemplaire, titre,nomIndividu, codesupport
  FROM Exemplaire E, **Film** F, **Individu**
  WHERE E.numFilm = F.numFilm
  AND realisateur = numIndividu
  AND probleme IS NULL;

Les Vues en SQL

Sélection:

- **SELECT** num, titre, dateLocation, login
  FROM exemplairePlus, Location
  WHERE num = numExemplaire
  AND real = ‘Scorces'
  AND dateRetour IS NULL;

Les Vues en SQL

Insertion:

- **INSERT INTO exemplairePlus** (num, support)
  VALUES (150346, 'DVD');

Suppression:

- **DROP VIEW exemplairePlus**;

Les Vues en SQL

Contraintes d'intégrité (CHECK OPTION) :

- **CREATE VIEW anciensExemplaires**
  AS SELECT * FROM Exemplaire
  WHERE numExemplaire < 2000
  WITH CHECK OPTION;

- **UPDATE anciensExemplaires**
  SET numExemplaire = **3812**
  WHERE numExemplaire = 1318**;**

- **Sans 'WITH CHECK OPTION', c'est possible.**
- **Avec 'WITH CHECK OPTION', c'est impossible.**