Sommaire

1 Introduction

Dans les cours précédents, nous avons vu SQL sous ses aspects les plus courants, il nous reste à voir d'une part les différents outils nous permettant d'optimiser ou de faciliter l'accés à une base de donnée, ainsi que les fonctions de manipulation ou de conversion de dates ou autres.

2 Modèle utilisé

Pour cette partie, nous reprendrons principalement le modèle Garage :

VEHICULE (Code, Type, Marque, Puissance) → les types de véhicule
GARAGE (Code, Nom, Adresse, Ville, Cpostal, Marque) → les garages
PLAQUE (Immat, CodeV#, Nom, Prenom, Adresse, Ville, Cpostal, Date) → véhicules immatriculé

CodeV# de PLAQUE est clé étrangère et référence Code de VEHICULE.

3 Les index

3.1 Qu'est ce qu'un index ?

Un index correspond à un moyen d'accéder plus rapidement à une information. Imaginons une table adresse ayant cette structure :

Dans la table GARAGE, Code est clé primaire, ce qui fait que la table est triée par code et donc toute recherche à partir du code sera très rapide (méthode de tri dichotomique, par arbre ou toute autre technique).

De façon automatique, un index est créé sur une clé primaire (simple ou composé), ce qui fait que toute recherche à partir d'une clé est très rapide. Ceci n'est pas vrai à partir des autres champs, puisque dans ce cas, il faudra parcourir l'ensemble de la table pour trouver la ou les bonnes informations. Une technique consiste à créer un index supplémentaire sur la table afin d'accélérer les recherches.
Dans notre exemple, un index pourra être créé sur le champs ville si des recherches fréquentes sont sur ce champs.

Attention, néanmoins créer des index à deux inconvénients :

- Les accés à la table sont ralentis en cas de mise à jour de données (Mise à jour, insertion ou effacement de données), puisque dans ce cas, non seulement la table est mise à jour, mais également les "tables d'index".
- La base grossit en fonction des index, puisque les index sont stockés dans la base, dans des tables supplémentaires

La création d'index doit donc être réfléchie, et se réalise généralement sur des champs de petite taille afin de minimiser à la fois la taille de la base et les accés. Il s'agit donc d'un compromis.

3.2 Comment créer un index ?

La création d'un index correspond à une modification sur une table. La commande est

```
CREATE [ UNIQUE ] INDEX
<nom de l'index>
ON <TABLE> ( nom du  champ)
```

UNIQUE permet d'ajouter une clause d'unicité (ce qui est le cas pour une clé)

Exemple: création d'un index sur le champs Ville:

```
CREATE INDEX  index_garage_ville
        ON GARAGE (Ville);
```

3.3 Comment effacer un index ?

De la même manière que pour effacer une contrainte, à l'aide de la commande DROP

```
DROP INDEX <nom de l'index>
```

Ex : `DROP INDEX  index_garage_ville`

3.4 Quelques règles pour le choix d'un index

- Ne pas créer d'index pour de petites tables (< 300 enregistrements)
- Ne pas créer d'index sur un champs qui ne possède que quelques valeurs différentes (ex : le nombre d'étoiles des restaurants
- Indexer les colonnes intervenant souvent dans des clauses Where ou Order By
- Indexer les colonnes de jointure

De manière générale, La création d'Index répond à des choix d'optimisation des accés sur la base. On y fait appel à partir du moment où des lenteurs excessives sont constatées.

4 Les vues

4.1 Qu'est ce qu'une vue ?

Le concept de vue revêt une importance particulière dés que l'on envisage des notions de facilité d'accés à la base et de sécurité des données.

Une vue est une table virtuelle construite à partir des tables d'une base de données existante. Elle n'a pas d'existence physique permanente, mais est construite dynamiquement, en cas d'utilisation, à partir des données figurant dans la base.

Par contre une vue s'utilise, à quelques conditions près, avec les mêmes instructions qu'une table (SELECT, DELETE, INSERT, UPDATE).

Une vue peut être :

- La table GARAGE limitée au département de la Meuse.
- La table GARAGE ne comprenant comme champs que le code, le nom et l'adresse e-mail
- Le propriétaire avec ses coordonnées associées aux caractéristiques de sa voiture. (Dans ce cas, la vue provient de 2 tables)
- …

En synthèse, une vue est une table dynamique permettant de montrer à l'utilisateur le contenu d'une table limitée à ce que le créateur de la vue désire. L'utilisateur en manipulant une vue aura l'impression d'intervenir sur une table à part entière.

4.2 La création d'une vue

Créer une vue consiste à identifier sous un nom une requête SQL qui définit la vue.

```
CREATE VIEW nom_vue (liste_colonne)
        AS  commande SELECT
        WITH (options)…
```

- nom_vue correspond au nom de la vue (table virtuelle) créée
- liste_colonne est facultatif. Dans le cas où les noms de champ ne sont pas donnés, ils héritent des noms du SELECT
- commande correspond à une commande SELECT sur une ou plusieurs table
- WITH permet d'ajouter des contraintes sur la vue

Ex : Vue sur les clients du département de la Meuse :

```
CREATE VIEW GARMEUSE
AS SELECT FROM GARAGE WHERE Cpostal LIKE '55%';
```

Dans le cas ci-dessus :

- Tout ajout de Garage peut se faire sur la table.
- Une mise à jour d'un garage de la Meuse peut se faire sur la table ou la vue
- La mise à jour d'un garage hors Meuse ne peut se faire que sur la table

Vue uniquement sur les garages (noms et marque)

```
CREATE VIEW GARMARQUE
AS SELECT Code, Nom, Marque FROM GARAGE;
```

Dans ce cas,

- Les mises à jour sur la vue ne peuvent se faire que sur les champs de la vue.
- L'insertion d'un client dans la vue sera limitée aux champs visibles.
- Si des champs non présents dans la vue ont une contrainte NOT NUL, l'insertion ne pourra se faire à partir de la vue.

Vue associant la voiture avec son propriétaire

```
CREATE VIEW VOITNAME
    AS SELECT Immat, Nom, Marque, Type, Adresse, Ville, Cpostal
        FROM PLAQUE, VEHICULE
        WHERE Code = CodeV;
```

Dans ce cas, les mises à jours sont difficiles, voire impossibles puisque l'on n'a qu'une partie des informations des 2 tables. Ce type de vue est essentiellement pour faciliter les recherches de données.

4.3 Créer une vue modifiable

Pour qu'une vue soit modifiable, indépendamment des contraintes d'intégrité sur les tables, il faut respecter un certain nombre de règles :

- Pas de directives distinct
- Pas de fonctions d'agrégat (AVG, COUNT …)
- PAS de GROUP BY, ORDER BY, HAVING
- La vue ne doit pas être déclarée en lecture seule (WITH READ ONLY)

4.4 L'option with check option

Nous avons vu précédemment qu'il est possible de modifier ou d'ajouter des enregistrements n'appartenant pas à la vue (ajout d'un garagiste n'appartenant pas à la Meuse dans la vue GARMEUSE).

La directive WITH CHECK OPTION empêche un ajout ou une modification non conforme à la définition de la vue.

EX :

```
CREATE VIEW GARMEUSE
AS SELECT FROM GARAGE WHERE Cpostal LIKE '55%'
WITH CHECK OPTION;
```

4.5 La suppression d'une vue

Important : la suppression d'une table ne supprime pas la déclaration de la vue correspondante. Il est donc dans ce cas nécessaire si on veut garder une cohérence dans la base de supprimer la vue correspondante .

La commande utilisée est :

```
DROP VIEW  nom_vue
```

4.6 Quelques remarques

La notion de vue est implémentée différemment suivant les bases de données.

Si toutes les grandes bases du marché implémentent la notion de vue, les accés, et plus particulièrement les mises à jour sont autorisées différemment.
Postgresql ne supporte pas de manière native la mise à jour des données à travers une vue. Il faut créer des fonctions pour assurer les mises à jour.

5 Les fonctions en SQL

Dans le cours précédent, nous avons vu un certain nombre de fonctions de base (Month, year…). Il existe en SQL une multitude de fonctions permettant de manipuler toutes sortes d'informations.
Les exemples ci-après sont ressortis de la doc de Postgresql.

5.1 Les fonctions mathématiques

Toutes les principales fonctions existent. Ci-joint un extrait :

Opérateur	Description	Exemple	Résultat
+	addition	2 + 3	5
-	soustraction	2 - 3	-1
*	multiplication	2 * 3	6
/	division (la division entière tronque les résultats)	4 / 2	2
%	modulo (reste)	5 % 4	1
^	exponentiel	2.0 ^ 3.0	8
|/	racine carrée	|/ 25.0	5
||/	racine cubique	||/ 27.0	3
!	factoriel	5 !	120
!!	factoriel (opérateur préfixe)	!! 5	120
@	valeur absolue	@ -5.0	5

D'autres fonctions permettent également de mettre en forme les résultats :

round(dp ou numeric)	(identique à l'argument)	arrondi à l'entier le plus proche	round(42.4)	42
round(v numeric, s int)	numeric	arrondi pour s décimales	round(42.4382, 2)	42.44

5.2 Fonctions sur les chaînes de caractères

Précédemment, nous avons vu les fonctions UPPER et LOWER. D'autres fonctions courantes existent comme :

- La concaténation de chaînes de caractères : string1 || string2
- La conversion d'une chaîne avec la premiére lettre en majuscule : initcap (text)
- Le remplacement d'une chaîne par une autre :
- L'extraction d'une partie d'une chaîne de caractère : substring
- La longueur d'une chaîne de caractère : length
- …

5.3 Les fonctions sur les dates

Comme nous avons vu, la date n'est pas nécessairement représenté dans le format que l'on désire. De plus, la date renvoyée peut être codée de différentes façons. Un certain nombre de fonctions permettent de manipuler ces dates :

Suivant la base, et le paramétrage, les dates sont exprimées en format anglo-saxon (mois, jour, année) ou français. De plus, certains paramètres permettent de changer les paramètres et de manipuler ces dates. Attention, ces différentes fonctions ne sont pas les même en fonction des bases, et il sera nécessaire de consulter l'aide afin de voir leur syntaxe précise.

5.3.1 date et heure courante

current_date, current_time, current time_stamp permettent de récupérer les informations courantes
now() est équivalent à current_timestamp (date et heure courante)

Ex : récupération de la date courante :

```
SELECT current_date ;
```

requête de mise à jour du champ date à la date du jour

```
UPDATE <TABLE>
SET date = current_date  WHERE … ;
```

5.3.2 Calcul sur les dates

Il est possible également de faire des calculs sur les dates:

```
UPDATE <TABLE>
SET date = current_date +5 WHERE … ;
```

Cette requête initialisera le champ date à date du jour + 5 jours

```
UPDATE COMPETITION
SET datcomp = datcomp + INTERVAL '1 month'
WHERE refcomp = 'comp9';
```

Dans ce cas tiré de la base SKI, le mois de datcomp sera augmenté de 1, indépendamment du nombre de jours du mois.

5.3.3 extraction sur les dates

Il est possible à partir d'une date de récupérer certaines informations de celles-ci :

DAY, MONTH, YEAR, WEEK (no de la semaine), DOY (nombre de jour depuis le début de l'année), Hour, Minute

Ces fonctions doivent être exécutées sur des champs date ou heure.

Certaines bases de données les implémentent de façon native

MONTH (date) → le mois de la date passée en paramètre.

Postgresql utilise la fonction EXTRACT :

EXTRACT (MONTH FROM DATE)

ex : SELECT Refcomp, EXTRACT (MONTH from datcomp) from COMPETITION;

5.3.4 Les fonctions de formatage

Les fonctions de formatage fournissent un ensemble d'outils puissants pour convertir différents types de données (date/heure, entier, nombre à virgule flottante, numérique) en des chaînes formatées et pour convertir des chaînes formatées en des types de données spécifiques. Ces fonctions suivent toute une convention d'appels commune : le premier argument est la valeur à formater et le second argument est un modèle définissant le format de sortie ou d'entrée.

Tableau de formatage sous Postgresql (ces fonctions existent aussi sous Oracle):

Fonction	Type en retour	Description	Exemple
to_char(timestamp, text)	text	convertit un champ de type timestamp en une chaîne	to_char(current_timestamp, 'HH12:MI:SS')
to_char(interval, text)	text	convertit un champ de type interval en une chaîne	to_char(interval '15h 2m 12s', 'HH24:MI:SS')
to_char(int, text)	text	convertit un champ de type integer en une chaîne	to_char(125, '999')
to_char(double precision, text)	text	convertit un champ de type real/double precision en une chaîne	to_char(125.8::real, '999D9')
to_char(numeric, text)	text	convertit un champ de type numeric en une chaîne	to_char(-125.8, '999D99S')
to_date(text, text)	date	convertit une chaîne en date	to_date('05 Dec 2000', 'DD Mon YYYY')
to_timestamp(text, text)	timestamp with time zone	convertit une chaîne string en un champ de type timestamp	to_timestamp('05 Dec 2000', 'DD Mon YYYY')
to_timestamp(double precision)	timestamp with time zone	convertit une valeur de type epoch UNIX en une valeur de type timestamp	to_timestamp(200120400)
to_number(text, text)	numeric	convertit une chaîne en champ de type numeric	to_number('12,454.8-', '99G999D9S')

Modèle les plus courants pour les formats de type date/heure :

Modèle	**Description**
HH	heure du jour (0-12)
HH24	Heure du jour (0-24)
MI	Minutes (0-59)
YYYY	Année sur quatre chiffres
YY	Année sur 2 chiffres
MONTH	nom complet du mois en majuscule
Mon	abréviation du nom du mois
DAY	Nom complet du jour
DY	abréviation du jour sur 3 caractères
DD	Jour du mois (1-31)
…	

Exemple d'utilisation :

SELECT refcomp, to_CHAR (datcomp,'Day,Month,YYYY') from competition;

Permet de lister les dates sous forme jour (littéral), mois (littéral) et année

```
UPDATE COMPETITION
SET datcomp = TO_DATE ('05 Dec 2000', 'DD Mon YYYY')
WHERE refcomp = 'comp9';
```

Cette requête permet de mettre à jour suivant le format donné en paramétre.

Il est possible également de formater la date pour toutes les requêtes :

```
SET datestyle TO dmy;
```

6 Les requêtes de requêtes

Important, le résultat d'une requête est une table.
Cette table résultante peut donc être une table appelée dans une requête

Exemple, reprenons la requête du TP Postgresql :

Donnez le nombre moyen de participants aux compétitions se déroulant à tignes.

Si on découpe la requête, il faut dans un premier temps considérer le nombre de participants concernant les compétitions de Tignes
Cette requête s'exprime sous la forme :

```
select count(nomski) as compteur
        from classement cl, competition co
        where co.refcomp=cl.refcomp
        and nomstat='tignes'
        group by co.refcomp
```

Il faut ensuite prendre la moyenne des nombres renvoyés par la requête, ce qui donne :

```
select avg(compteur)
        from (
                select count(nomski) as compteur
                from classement cl, competition co
                where co.refcomp=cl.refcomp
                and nomstat='tignes'
                group by co.refcomp
                ) as req2;
```

La requête précédente constitue la table REQ2 qui est utilisée comme table dans la requête principale.

Ce type de requête est utilisée dés lors où un résultat d'un résultat est nécessaire.

Important, la table créée n'a d'existence que pendant la requête.