BASE DE DONNÉES

JOURNÉE 3 - MATIN

1

Karine BRIFAULT

ORGANISATION

- Matinée
  - Cours de 1 h 30
    - Le langage SQL pour le développement
      - Procédures et fonctions
  - Exercices 1 h 30
    - Réaliser des procédures et fonctions
- Après Midi
  - Examen : 4h00
    - Sous la forme d'un exercice
      - Réalisation d'une base de données à partir d'un cahier des charges simplifié
      - Réalisation des requêtes SQL, trigger, procédure, …

SQL AVANCÉ
FONCTIONS

- Permet de capitaliser sur une fonctionnalité.
- Retourne **toujours** un résultat (doit contenir le mot clef *RETURN*).
- Peut prendre des paramètres d'entrées (*IN*) ou de sorties (*OUT*).
- Est exécutée par la base de données sur le serveur.
- Ecrit en
  - SQL + Langage de programmation associé à la base de données.
  - En langage de haut niveau (Java, .NET, Python, …) pour certaines grosses bases de données (Oracle)

SQL AVANCÉ
PROCEDURE

- Permet aussi de capitaliser sur une fonctionnalité.
- Ne retourne **pas** de résultat (ne doit pas contenir le mot clef *RETURN*).
  - Peut cependant retourner un ensemble de valeurs résultant d'un select
- Peut prendre des paramètres d'entrées (*IN*) ou de sorties (*OUT*).
- Est exécutée par la base de données sur le serveur.
- Ecrit en
  - SQL + Langage de programmation associé à la base de données.
  - En langage de haut niveau (Java, .NET, Python, …) pour certaines grosses bases de données (Oracle)

SQL AVANCÉ
PROCEDURES - FONCTIONS

- Pourquoi en faire ?
  - Eviter de faire réaliser un traitement par le code client ⇔ capitaliser sur la base
  - Garantir le même comportement d'une requête si plusieurs clients accèdent aux données
  - Optimiser les traitements gérés en natif par les bases de données (ensemble, tries)

SQL AVANCÉ
PROCEDURES - FONCTIONS

- Différence avec la trigger ?

SQL AVANCÉ
PROCEDURES - FONCTIONS

- La syntaxe de programmation d'une procédure, d'une fonction, d'une trigger est la même en MySQL.

- Seule l'instruction SQL de création change.

- **CREATE PROCEDURE** NomProcedure ([parameter[,...]]) [characteristic ...]

- **CREATE FUNCTION** NomFonction ([parameter[,...]]) [**RETURNS** type] [characteristic ...]

SQL AVANCÉ
PROCEDURES - FONCTIONS

- Exemple de procédure à un paramètre

```
CREATE PROCEDURE simpleproc (OUT param1 INT)
BEGIN
        SELECT COUNT(*) INTO param1 FROM client;
END
```

- Exemple de fonction à un paramètre

```
CREATE FUNCTION b(s VARCHAR(20))
                                RETURNS VARCHAR(50)
BEGIN
        RETURN CONCAT('Bonjour, ',s,'!');
END
```

SQL AVANCÉ
PROCEDURES - FONCTIONS

- Exemple de procédure sans paramètre qui retourne un ensemble de valeurs

```
CREATE PROCEDURE simpleproc () BEGIN
        SELECT * FROM client;
        SELECT * FROM compte;
END
```

- Cette procédure retourne tous les clients suivis de tous les comptes.

SQL AVANCÉ
PROCEDURES - FONCTIONS

- Un paramètre est de caractéristique :
  - **IN** : en lecture seule
  - **OUT** : en écriture seule = résultat d'une opération récupérable plus tard
  - **INOUT** : en lecture écriture

- Par défaut, si vous ne précisez rien, un paramètre est toujours de type IN (lecture seule)

SQL AVANCÉ
PROCEDURES - FONCTIONS

- Vous pouvez déclarer des variables dans vos procédures / fonctions.
- Elles serviront à contenir des informations de manière locales.
- Déclaration d'une variable :
  - **DECLARE** *nomVariable type* **DEFAULT** *valeurDefaut*;
- Affectation d'une variable :
  - **SET** *nomVariable* **=** *expression/valeur*;
- Affectation d'une variable dans un SELECT :
  - **SELECT** c1,c2 **INTO** nomVar1,nomVar2 *expression*;

SQL AVANCÉ
PROCEDURES - FONCTIONS

- Vous pouvez faire des boucles
  - **REPEAT** *instructions* **UNITL** *condition* **END REPEAT;**
  - **LOOP** *instructions* **END LOOP;**
  - **WHILE** *condition* **DO** *instructions* **END WHILE;**
- Pour sortir d'une boucle, vous pouvez utiliser le mot clef **LEAVE** (⇔ *break* en Java)
- Pour continuer une boucle, vous pouvez utiliser le mot clef **ITERATE** (⇔ *continue* en Java)

SQL AVANCÉ
PROCEDURES - FONCTIONS

- Vous pouvez faire des conditions
  - **IF** *condition* **THEN** *instructions* **END IF;**
  - **IF** *condition* **THEN** *instructions*
    ELSEIF *condition* **THEN** *instructions* **END IF;**
  - **IF** *condition* **THEN** *instructions*
    ELSE *instructions* **END IF;**

  - **CASE** *nomVariable*
    WHEN *valeur1* **THEN** *instructions*;
    WHEN *valeur2* **THEN** *instructions*;
    ELSE *instructions*;
    END CASE;

SQL AVANCÉ
PROCEDURES - FONCTIONS

- Les variables de type curseurs
  - Permettent de stocker un résultat complexe d'une requête SELECT.
  - Parcourir des tables.

- Un curseur étant un 'point de repère' sur une ligne d'une table, il faut penser impérativement à
  - L'ouvrir (mot clef **OPEN**) avant de s'en servir
    - Ce qui équivaut à demander l'exécution du SELECT
  - Le fermer (mot clef **CLOSE**) quand on en a plus besoin !
    - Sinon, la table risque de rester verrouillée ⇔ aucune autre requête ne pourra passer sur cette table

SQL AVANCÉ
PROCEDURES - FONCTIONS

```
CREATE PROCEDURE fabriquerListeMails (OUT resultat varchar(4000))
BEGIN
        DECLARE vEnd INTEGER DEFAULT 0;
        DECLARE vEmail varchar(100) DEFAULT "";

        DECLARE curseurEmail CURSOR FOR SELECT email FROM clients;

        DECLARE CONTINUE HANDLER FOR NOT FOUND SET vEnd = 1;

        OPEN curseurEmail;
        lpEmail: LOOP
                FETCH curseurEmail INTO vEmail;
                IF vEnd = 1 THEN
                        LEAVE lpEmail;
                END IF;
                SET resultat = CONCAT(vEmail,";",resultat);
        END LOOP lpEmail;

        CLOSE curseurEmail;
END
```

SQL AVANCÉ
PROCEDURES - FONCTIONS

- Gestion des erreurs se fait par des *HANDLER*
- Ils se déclarent comme des variables et permettent de gérer les traitements proprement.
  - Entre autre la fermeture des curseurs.
- **DECLARE** *handler_type* **HANDLER FOR** *conditions instructions*;

SQL AVANCÉ
PROCEDURES - FONCTIONS

- 2 types d'handler principaux
  - **CONTINUE** : le traitement continue et va jusqu'à la fin
  - **EXIT** : le traitement s'arrête

- 3 des possibilités les plus courantes :
  - **SQLWARNING** : Toutes les erreurs dont le SQLSTATE commencent par 01
  - **NOT FOUND** : Toutes les erreurs dont le SQLSTATE commencent par 02
  - **SQLEXCEPTION / EXCEPTION :** : Toutes les erreurs dont le SQLSTATE ne commencent pas par 01 ou 02

SQL AVANCÉ
PROCEDURES - FONCTIONS

- Pour exécuter une fonction ou une procédure il suffit d'utiliser la fonction SQL **CALL**

- Pour une procédure
  - **CALL** *maProcedure***(***p1, p2, p3***);**
- Pour une fonction
  - **SELECT** *maFontion***(***p1, p2, p3***);**

- Exemple pour une procédure:

```
SET @monResultat= "";
CALL fabriquerListeMails (@monResultat);
SELECT @monResultat;
```

SQL AVANCÉ
PROCEDURES - FONCTIONS

- Comment stocker un résultat de requête qui me ramène plusieurs valeurs ?
- => Dans une table temporaire.
- Les tables temporaires vous permette de stocker temporairement un résultat.
- Elles sont automatiquement effacées à la fin de l'exécution de la procédure / fonction.
- **CREATE TEMPORARY TABLE**
  IF NOT EXISTS *nomTable* **AS** (**SELECT** …)

SQL AVANCÉ
PROCEDURES - FONCTIONS

- Mais le **PL / SQL** c'est quoi ?
- Cela ressemble au SQL, mais c'est plus complet
  - Boucles, conditions, curseurs, …
- Cela permet de décrire des procédure/fonctions/traitements plus complexes qu'en SQL.
- Mais, ….., cela ne fonctionne que sur des bases **Oracle**.
- Cependant, (*Ouf*), cela ressemble à la syntaxe de MySQL

SQL AVANCÉ
PROCEDURES - FONCTIONS

- Mais le **PL / pgSQL** c'est quoi ?
- Cela ressemble au SQL, mais c'est plus complet
  - Boucles, conditions, curseurs, …
- Cela permet de décrire des procédure/fonctions/traitements plus complexes qu'en SQL.
- Mais, …., cela ne fonctionne que sur des bases **PostgreSQL**.
- Cependant, (*Ouf*), cela ressemble à la syntaxe de MySQL

SQL AVANCÉ
PROCEDURES - FONCTIONS

- Mais le **Transact - SQL** c'est quoi ?
- Cela ressemble au SQL, mais c'est plus complet
  - Boucles, conditions, curseurs, …
- Cela permet de décrire des procédure/fonctions/traitements plus complexes qu'en SQL.
- Mais, …., cela ne fonctionne que sur des bases **Sybase/SQL Server**.
- Cependant, (*Ouf*), cela ressemble à la syntaxe de MySQL

SQL AVANCÉ
PROCEDURES - FONCTIONS

- Bref, vous l'aurez compris
  - Dès que l'on fait des traitements complexes on passe sur du propriétaire

- Mais ce n'est pas une excuse pour ne pas se poser la question quand on doit réaliser un traitement complexe : "doit-on le faire sous la forme "
  - de **Trigger** gérée par la **base** ?
  - de **Procédure** gérée par la **base** ?
  - de **Fonction** gérée par la **base** ?
  - de **code** gérée par **l'application cliente** ?