IFT3030
Base de données

Chapitre 6
SQL

Plan du cours

- Introduction
- Architecture
- Modèles de données
- Modèle relationnel
- Algèbre relationnelle
- **SQL**
- Conception
- Fonctions avancées
- Concepts avancés
- Modèle des objets
- BD à objets

Introduction

- Permet de retrouver et de manipuler les données sans préciser les détails
- Dérivé de SEQUEL 2 (76) lui-même dérivé de SQUARE (75)
- Proposé par IBM (82 puis 87)
- Première version normalisée SQL1 (ISO89)
- Deuxième version normalisée SQL2 (ISO92)
- SQL3 en cours de normalisation
- Sert de couche basse aux L4G (par exemple Access)

3

Définition des données

- Domaine
  - Définition de domaines peu utilisée
  - Définition à partir des types de bases

ANSI SQL	Oracle
CHARACTER(n), CHAR(n)	CHAR(n)
CHARACTER VARYING(n), CHAR VARYING(n)	VARCHAR(n)
NUMERIC(p,s), DECIMAL(p,s), DEC(p,s)[1]	NUMBER(p,s)
INTEGER, INT, SMALLINT	NUMBER(38)
FLOAT(b)[2], DOUBLE PRECISION[3], REAL[4]	NUMBER

4

Définition des données

- Domaine
  - Exemple

CREATE DOMAIN COULEUR CHAR(6) DEFAULT 'vert'
CONSTRAINT VALIDE_COULEUR
CHECK (VALUE IN 'rouge', 'blanc', 'vert', 'bleu', 'noir')

  - Modification de la définition (ALTER DOMAIN)
  - Destruction d'un domaine

DROP DOMAIN domaine [RESTRICT | CASCADE]

5

Définition des données

- Table
  - Création d 'une table (forme simple)
    - Exemple

CREATE TABLE VIN (
NV NUMBER,
CRU CHAR(10),
MIL NUMBER)

  - Contraintes d 'intégrité
    - NOT NULL
    - UNIQUE ou PRIMARY KEY
    - FOREIGN KEY
    - REFERENCES
    - CHECK

6

Définition des données

- Table
  - Avec contraintes d 'intégrité

Exemple (SQL 86)

```
CREATE TABLE COMMANDE (
      NC NUMBER UNIQUE NOT NULL,
      NV NUMBER NOT NULL
      QUANTITE NUMBER(6))
```

Exemple (SQL 89)

```
CREATE TABLE COMMANDE (
      NC NUMBER PRIMARY KEY,
      NV NUMBER NOT NULL REFERENCES VIN,
      QUANTITE NUMBER(6) CHECK(QUANTITE > 0))
```

SQL 92

```
NC NUMBER,
PRIMARY KEY (NC),
NV NUMBER NOT NULL,
 FOREIGN KEY (NV) REFERENCES VIN,
```


Définition des données

- Table
  - Modification du schéma d 'une table

    ALTER TABLE <table>

    ADD COLUMN<attribut> <type>, <attribut> <type>, …

    • Exemple

    ALTER TABLE COMMANDE

    ADD COLUMN DATE_COM DATE
  - Suppression d 'une table

    DROP TABLE <table>

    • Exemple

    DROP TABLE COMMANDE

Accès aux données

- Clause SELECT
  - Forme

SELECT [ALL | DISTINCT] {<liste_attributs> | *}

- Clause FROM
  - Forme

FROM <liste_tables>

- Exemple

Employe (mat#, nom, adresse, dept, sup)
Projet (num#, designation, budget)
Departement (dept#, dir, appellation)
Role (nume#, nump#, role_emp)

Accès aux données

- Exemple
  - Donner les noms et adresses des employés

SELECT nom, adresse
FROM Employe

  - Donner la liste des projets

SELECT *
FROM Projet

  - Donner les noms et les départements des employés

SELECT nom, appellation
FROM Employe, Departement

Accès aux données

- Clause WHERE
  - Forme

    WHERE <condition>
- Exemple
  - Donner les noms et adresses des employés qui travaillent au DIRO

    SELECT nom, adresse

    FROM Employe, Departement

    WHERE appellation = 'DIRO'
  - Donner les noms des employés responsable de projets

    SELECT nom

    FROM Employe E, Role R

    WHERE E.mat = R.nume

    AND role_emp = 'responsable'

11

Accès aux données

- Fonctions de calcul
  - COUNT, SUM, AVG, MAX, MIN
- Exemple
  - Donner le nombre d 'employés qui travaillent au DIRO

    SELECT COUNT (*)

    FROM Employe, Departement

    WHERE appellation = 'DIRO'
  - Donner le budget total des projets sous la responsabilité de l'employé numéro 35677

    SELECT SUM (budget)

    FROM Role R, Projet P

    WHERE P.num = R.nump

    AND role_emp = 'responsable'

    AND nume = 35677

12

Accès aux données

- Forme de conditions
  - IN (NOT IN), Donner la liste des directeurs de départements

```
SELECT *
FROM Employe
WHERE mat IN
      (SELECT dir
      FROM Departement)
```

  - ANY ou ALL, Donner la liste des projets qui ont un budget supérieur au budget d'au moins un des projets impliquant l'employé numéro 34888

```
SELECT *
FROM Projet
WHERE budget > ANY
      (SELECT budget
      FROM Role R, Projet P
      WHERE R.nump = P.num
      AND nume = 34888)
```

13

Accès aux données

- Forme de conditions
  - IS (NOT) NULL, Donner la liste des employés sans superviseurs

```
SELECT *
FROM Employe
WHERE sup IS NULL
```

  - EXISTS, Donner les noms des employés qui ne sont impliqués dans aucun projet

```
SELECT nom
FROM Employe E
WHERE NOT EXISTS
      (SELECT *
      FROM Projet P
      WHERE E.mat = P.nume)
```

14

Accès aux données

- Clause GROUP BY
  - Forme

GROUP BY <liste_attributs>

- Exemple
  - Donner les noms des employés regroupés par projet

SELECT nom
FROM Employe E, Role R
WHERE E.mat = R.nume
GROUP BY R.nump

Accès aux données

- Clause HAVING
  - Forme

HAVING <condition>

  - Exemple

Donner les numéros de projets dans lesquels interviennent au moins dix employés autres que le responsable

SELECT nump
FROM Role
WHERE role_emp != ‘responsable’
GROUP BY nump
HAVING COUNT(nume) > 9

Accès aux données

- Clause ORDER BY
  - Forme

    ORDER BY {<attribut> [ASC | DESC]}$^{+}$
  - Exemple

    Donner la liste des projets dans lesquels participe l'employé 33549 par ordre décroissant de budget

```
SELECT *
FROM Projet
WHERE nume = 33549
ORDER BY budget DESC
```

17

Accès aux données

- Unions
  - Forme

    <requete> UNION <requete>
  - Exemple, liste des cadres

```
SELECT *
FROM Employe
WHERE mat IN ( SELECT dir
               FROM Departement)
UNION
SELECT *
FROM Employe
WHERE mat = ANY (   SELECT nume
                    FROM Role
                    WHERE role_emp = 'responsable')
```

18

Manipulation des données

- Insertion
  - Forme

    INSERT INTO <table> [(<liste_colonnes>)]

    {VALUES (<liste_valeurs>) | <requete>}
  - Exemple 1

```
INSERT INTO Employe (mat, nom, adresse, dept, sup)
VALUES (34098, 'Gilles', '3456, Gaspe, Mtl', 456, 68735)
```

  - Exemple 2

```
INSERT INTO Directeur
SELECT *
FROM Employe
WHERE mat IN ( SELECT dir
               FROM Departement)
```


Manipulation des données

- Mise à jour
  - Forme

    UPDATE <table>

    SET {<nom_colonne> = <expression>}$^{+}$

    WHERE {<condition> | CURRENT OF <nom_curseur>}
  - Exemple 1

```
UPDATE Employe
SET adresse = '3456, Gaspe, Mtl'
WHERE mat = 34098
```

  - Exemple 2

```
UPDATE Projet
SET budget = budget + 1000
WHERE num IN (SELECT nump
              FROM Role
              WHERE nume = 34098
              AND role_emp = ' responsable ')
```

Manipulation des données

- Suppression
  - Forme

    DELETE FROM <table>

    [WHERE <condition>]
  - Exemple 1

```
DELETE FROM Employe
WHERE mat = 34098
```

  - Exemple 2

```
DELETE FROM Projet
WHERE num IN ( SELECT nump
               FROM Role
               WHERE nume = 68754)
```


SQL intégré

- Introduction
  - SQL peut être intégré dans un langage hôte (C, COBOL, PL/1, PASCAL, JAVA, etc.)
  - Conflit (opération ensembliste vs. opération séquentielle)
    - traitement de plusieurs n-uplets
  - Étude du cas du langage C

SQL intégré

- Principes
  - Tout instruction SQL commence par l'expression EXEC SQL pour la distinguer des autres instructions du langage hôte
  - Différents types d'instructions
    - déclarations
    - connexion
    - traitement

SQL intégré

- Déclarations (variables de communication)
  - Elle se fait dans la DECLARE SECTION. Celle ci commence par l'ordre

EXEC SQL BEGIN DECLARE SECTION;

et se termine par

EXEC SQL END DECLARE SECTION;

  - Exemple

```
EXEC SQL BEGIN DECLARE SECTION;
int pempno ;
char pname[11];
int pdeptno;
EXEC SQL END DECLARE SECTION;
```

SQL intégré

- Déclarations (variables de communication)
  - Utilisation dans SQL

    EXEC SQL SELECT DEPTO, ENAME

    INTO :pdeptno, :pname FROM EMP

    WHERE EMPNO = :pempno;
    - Variables précédées de ":" pour les distinguer des noms des attributs
  - Utilisation dans C

    strcpy(pname,"Martin") ;
  - Les types possibles pour ces variables sont ceux compatibles avec ORACLE (entiers, réels, chaînes de caractères)

SQL intégré

- Connexion
  - La connexion à une base ORACLE se fait par l'ordre SQL :

    EXEC SQL CONNECT :username IDENTIFIED BY :password;
    - username et password sont des variables déclarées dans la section déclaration
  - Exemple

    EXEC SQL BEGIN DECLARE SECTION;

    VARCHAR username[20];

    VARCHAR password[20];

    EXEC SQL END DECLARE SECTION;

    EXEC SQL INCLUDE sqlca.h;

SQL intégré

- Connexion
  - Exemple (suite)

```
main()
{
strcpy(username.arr,"login_oracle"); /* Copie du username*/
username.len = strlen(username.arr);
strcpy(password.arr,"motdepasse_oracle");
password.len = strlen(password.arr);
EXEC SQL CONNECT :username IDENTIFIED BY :password;
…
}
```


SQL intégré

- Traitements
  - Mise à jour
    - "mettre à jour le salaire dans la relation employé"

```
EXEC SQL UPDATE EMP
SET SAL = :salaire
WHERE EMPNO=301;
```

  - Suppression

```
EXEC SQL DELETE FROM EMP WHERE EMPNO = :empno;
```

  - Création d ’une table

```
EXEC SQL CREATE TABLE EMP_TEST
(EMPNO NUMBER, ENAME CHAR(15), JOB CHAR(10));
```

SQL intégré

- Traitements
  - Sélection, cas du INTO
    - S 'applique quand le SELECT retourne un seul n-uplet

```
EXEC SQL SELECT job, sal
INTO :fonction, :salaire
FROM EMP
WHERE empno=301;
```

  - Sélection, utilisation d 'un curseur
    - S 'applique quand le SELECT retourne un ensemble de n-uplets
    - Un curseur est une structure de données contenant tous les n-uplets retournés par la commande SELECT
    - Cette structure se manipule comme un fichier séquentiel

29

SQL intégré

- Traitements
  - Association du curseur à un SELECT

```
EXEC SQL DECLARE C CURSOR FOR
SELECT job, salaire
FROM EMP;
```

  - Ouverture du curseur

```
EXEC SQL OPEN C;
```

    - A l 'ouverture, le premier n-uplet est pointé
  - Fermeture du curseur

```
EXEC SQL CLOSE C;
```

30

SQL intégré

- Traitements
  - Accès aux autres n-uplets de manière séquentielle
    - Se fait par l 'instruction FETCH

```
EXEC SQL FETCH C INTO :fonction, :salaire;
```

    - On ne peut pas reculer dans le curseur
    - Pour accéder de nouveau aux n-uplets, il faut fermer et rouvrir le curseur

SQL intégré

- Commandes dynamiques
  - Il est possible d 'exécuter des commandes SQL inconnues au moment de l' écriture du programme
  - Il existe pour cela quatre méthode
    - Commandes sauf SELECT sans variables (EXECUTE IMMEDIATE)
    - Commandes sauf SELECT avec un nombre de variables fixe (PREPARE, EXECUTE)
    - Commandes avec un nombre de variables variable (PREPARE, DECLARE, OPEN, FETCH)
    - SELECT complètement dynamique

SQL intégré

Commandes dynamiques

– Commandes sans variables

EXEC SQL EXECUTE IMMEDIATE :modif;

– Commandes avec un nombre de variables fixe

EXEC SQL PREPARE S1 FROM :chaîne;
EXEC SQL EXECUTE S1 USING :variable1,:variable2, ...;

SQL intégré

Directives de traitement d'erreur

EXEC SQL WHENEVER [SQLERROR / SQLWARNING / NOT FOUND] [STOP / CONTINUE / GO TO étiquette];

– Ces directives correspondent donc à 3 événements ORACLE :
  - SQLERROR : erreur ORACLE
  - SQLWARNING : "warning" ORACLE
  - NOT FOUND : curseur vide ou fini

SQL intégré

- Directives de traitement d'erreur

  EXEC SQL WHENEVER [SQLERROR / SQLWARNING / NOT FOUND] [STOP / CONTINUE / GO TO étiquette];

  – Les actions possibles sont :
    - STOP : le programme se termine et la transaction est « abortée »,
    - CONTINUE : le programme continue en séquence,
    - GO TO : le programme se branche à l'adresse indiquée.

SQL intégré

- Directives de traitement d'erreur
  – La portée d'une directive WHENEVER va jusqu'à la directive WHENEVER suivante (ou la fin de programme) dans l'ordre du texte source PRO*C (et non pas dans l'ordre d'exécution).

  – L'erreur classique dans la manipulation des SQL WHENEVER est la suivante :

SQL intégré

```
routine a
{ ...
EXEC SQL WHENEVER ERROR GOTO toto;
...
toto : ...
}
routine b
{ ...
EXEC SQL INSERT ...
} /* donc rien est dit dans b pour SQL WHENEVER */
```

– Par conséquent, au sein de la routine b, on garde les dernières directives rencontrées, donc celles de la routine a. Or l'étiquette toto est locale à a et donc inconnue dans b.

37

SQL intégré

- Solutions
  - par exemple avoir systématiquement des étiquettes globales
  - définir localement dans chaque routine les directives d'erreurs.

38

SQL intégré

- Gestion des transactions
  - Une transaction est assimilée à une exécution d'un programme. Le début de transaction est implicite (c'est le début de programme), et la fin est soit implicite
  - (erreur non récupérée par un WHENEVER : annulation, ou fin de programme : validation), soit explicite. Les ordres de fin de transaction explicites sont :

SQL intégré

- Gestion des transactions

```
EXEC SQL COMMIT WORK [RELEASE];
```

  - Valide les mises à jour. L'option RELEASE désalloue toutes les ressources ORACLE et réalise la déconnexion de la base

```
EXEC SQL ROLLBACK WORK [RELEASE];
```

  - Annule les mises à jour

SQL intégré

- Gestion des transactions
  - Pour éviter les problèmes de conflit entre le ROLLBACK et les directives WHENEVER, il est prudent d'utiliser le ROLLBACK comme suit :

```
EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL WHENEVER SQLWARNING CONTINUE;
EXEC SQL ROLLBACK WORK;
```

41