Chapitre 7 : SQL Avancé Oracle ©

INF3080 BASES DE DONNÉES (SGBD)

Guy Francoeur

3 septembre 2019

UQÀM | Département d'informatique

Matériel et droits

- Les droits de lecture sont accordés aux étudiants inscrits au cours INF3080-030 A2019 uniquement;
- Aucun droit pédagogique ou reproduction n'est accordé sans autorisation;

Table des matières

1. Au dernier cours

2. SQL Avancé
   - la clause WHERE
   - la clause IN
   - la clause BETWEEN
   - la clause LIKE
   - la clause GROUP BY
   - la clause HAVING
   - la clause ORDER BY
   - la clause DISTINCT

Table des matières

1. Au dernier cours

2. SQL Avancé
   - la clause WHERE
   - la clause IN
   - la clause BETWEEN
   - la clause LIKE
   - la clause GROUP BY
   - la clause HAVING
   - la clause ORDER BY
   - la clause DISTINCT

Retour sur les devoirs

- Questions, précisions, ...

résumé

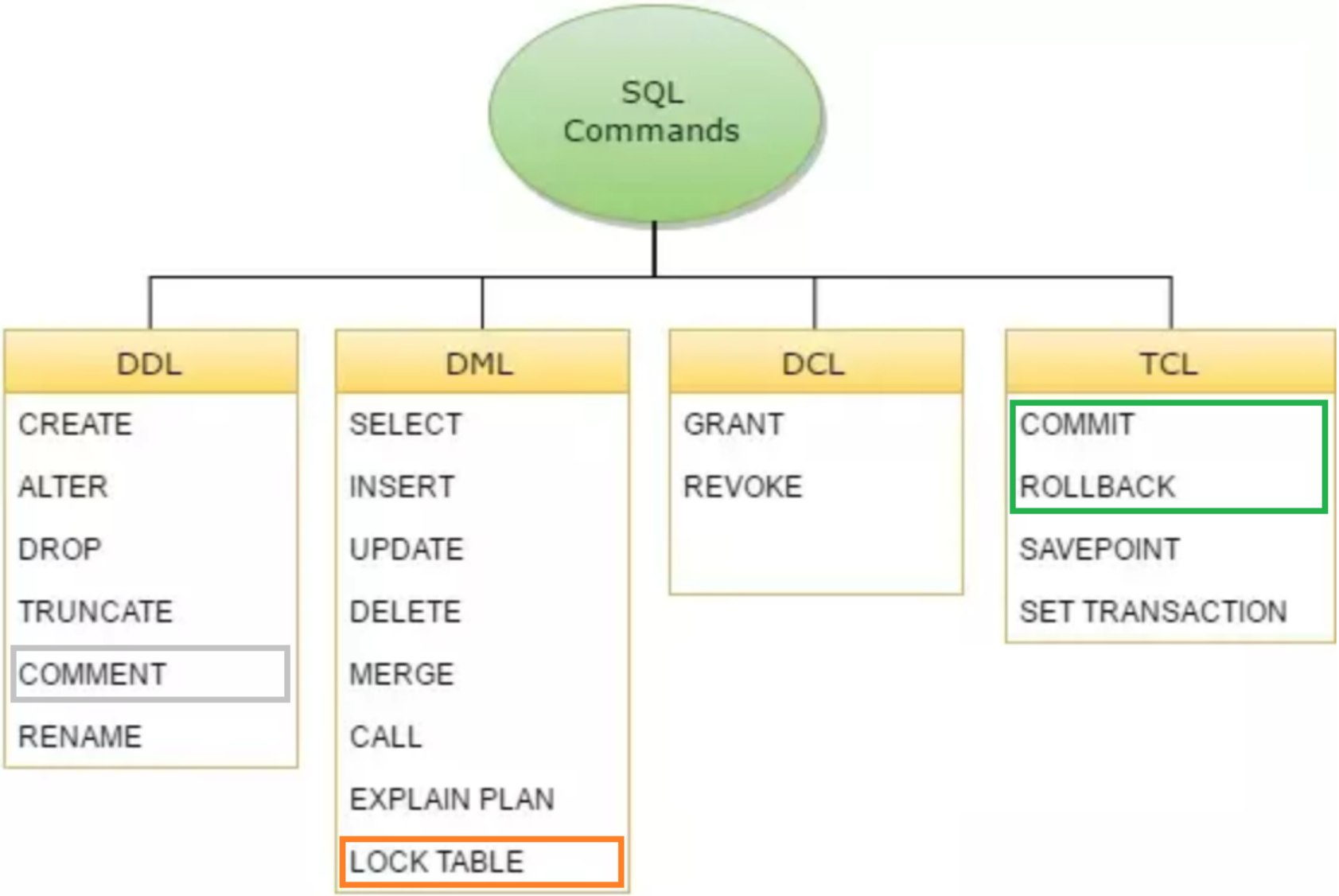

SQL
Commands
DDL
CREATE
ALTER
DROP
TRUNCATE
COMMENT
RENAME
DML
SELECT
INSERT
UPDATE
DELETE
MERGE
CALL
EXPLAIN PLAN
LOCK TABLE
DCL
GRANT
REVOKE
TCL
COMMIT
ROLLBACK
SAVEPOINT
SET TRANSACTION

Table des matières

1. Au dernier cours

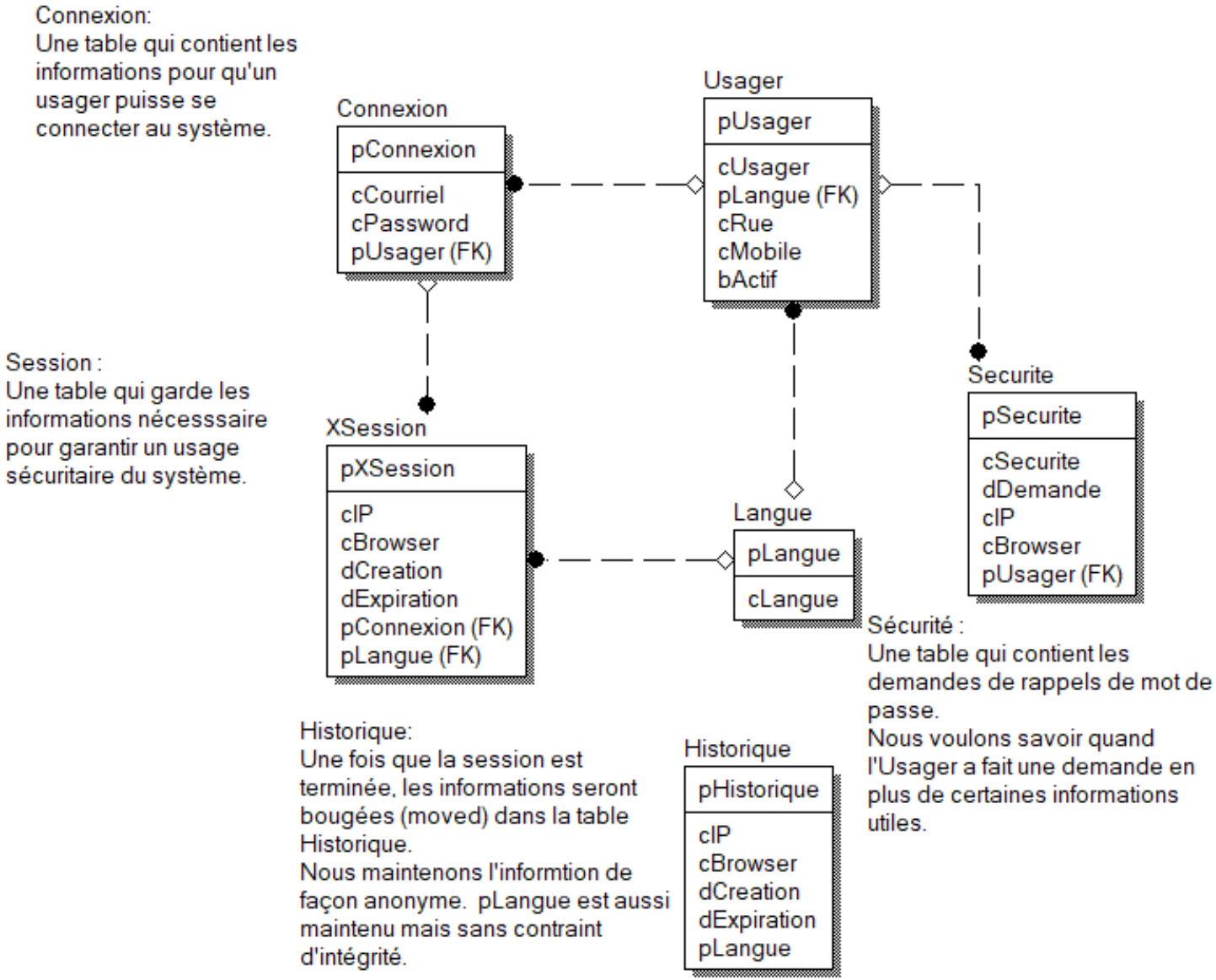
2. SQL Avancé
   - Contraintes
   - Jointures
     - la clause WHERE
     - la clause IN
     - la clause BETWEEN
     - la clause LIKE
   - groupement
     - la clause GROUP BY
     - la clause HAVING
     - la clause ORDER BY
   - sous-requêtes
   - sans doublons
     - la clause DISTINCT

Contrainte - définition

Les contraintes sont des règles qui limitent (ajoutent de la précision) les données qui seront maintenues dans une colonne.

- Clé primaire - *Primary Key*;
- Clé unique - *Unique Key*;
- Clé étrangère - *Referential integrity*;
- Validation - *Check option*;
- Base;

Contrainte - modèle entité association



Contrainte - exemple

```
SQL> CREATE TABLE Langue (
 pLangue                NUMBER NOT NULL ,
 cLangue                VARCHAR2(50) NULL
);

ALTER TABLE Langue
 ADD CONSTRAINT  XPKLangue PRIMARY KEY (pLangue);

CREATE UNIQUE INDEX XPKLangue ON Langue
(pLangue   ASC);
```

```
SQL> CREATE TABLE Langue (
 pLangue                NUMBER NOT NULL ,
 cLangue                VARCHAR2(50) NULL ,
CONSTRAINT XPKLangue PRIMARY KEY (pLangue)
);

CREATE UNIQUE INDEX XPKLangue ON Langue
(pLangue   ASC);
```

Contrainte intégrité et validation - exemple

Version avec ALTER TABLE ... ADD CONSTRAINT

```
SQL> CREATE TABLE Usager (
 pUsager                NUMBER NOT NULL ,
 cUsager                VARCHAR2(50) NULL ,
 pLangue                NUMBER NULL ,
 cRue                   VARCHAR2(50) NULL ,
 cMobile                VARCHAR2(50) NULL ,
 bActif                 NUMERIC(1) NOT NULL CONSTRAINT
  Validation_Rule_234 CHECK (bActif IN (0, 1)),
CONSTRAINT XPKUsager PRIMARY KEY (pUsager)
);

ALTER TABLE Usager
 ADD (CONSTRAINT R_4 FOREIGN KEY (pLangue) REFERENCES Langue (
     pLangue));
```

Contrainte intégrité et validation - exemple

- La contrainte d'intégrité lors de la création de la table.
- La contrainte de validation est associée à une colonne lors de la création de la table.

```
SQL> CREATE TABLE Usager (
 pUsager                NUMBER NOT NULL ,
 cUsager                VARCHAR2(50) NULL ,
 pLangue                NUMBER NULL , -- Attention
 cRue                   VARCHAR2(50) NULL ,
 cMobile                VARCHAR2(50) NULL ,
 bActif                 NUMERIC(1) NOT NULL CONSTRAINT --Attention
  Validation_Rule_234 CHECK (bActif IN (0, 1)),
CONSTRAINT XPKUsager PRIMARY KEY (pUsager),
CONSTRAINT R_4 FOREIGN KEY (pLangue) REFERENCES Langue (pLangue)
);
```

filtres, jointure - la clause WHERE

- Utilisé pour lier des tables;
- Utilisé pour filtrer des données;
- Pourquoi est-il utile d'appliquer un filtre ?

```
SQL> SELECT pLangue FROM Langue WHERE pLangue=1;
```

```
SQL> SELECT u.* FROM Usager u, Langue l
WHERE u.pLangue = l.pLangue
AND pLangue=1;
```

filtres, jointure - la clause IN

La clause la plus simple qui exprime très bien la théorie des ensembles.

```
SQL> SELECT u.*
FROM Usager u
WHERE pLangue IN (2,3);
```

```
SQL> SELECT u.*
FROM Usager u
WHERE pLangue NOT IN (2,3);
```

filtres, jointure - la clause WHERE

Un filtre avec la clause BETWEEN

- Utilisé pour définir des intervalles;
- Utilisé avec les type date ou entier;

```
SQL> SELECT u.cUsager, l.cLangue
FROM Usager u, Langue l
WHERE u.pLangue = l.pLangue
AND l.pLangue BETWEEN 1 AND 3;
```

filtres, jointure - la clause WHERE

Un filtre avec la clause LIKE

- Utilisé avec les chaînes de caractères;

```
SQL> SELECT u.cUsager, l.cLangue
FROM Usager u, Langue l
WHERE u.pLangue = l.pLangue
AND cLangue LIKE 'A%';
```

filtres, jointure - la clause WHERE

Sommaire des opérateurs qui sont utilisés

- IN, NOT, LIKE, BETWEEN
- >, <, <=, >=, =, <>
- IS [NOT] NULL, IS [NOT] TRUE | FALSE
- AND, OR, ( )

groupement - la clause GROUP BY

- Utilisé pour grouper les données afin de faire des sommaires;
- Pourquoi est-il utile d'utiliser le groupement ?

```
SQL> SELECT cLangue, sum(1) C1, count(*)
FROM Usager u, Langue l
WHERE u.pLangue = l.pLangue
GROUP BY cLangue;
```

groupement - la clause HAVING

- Utilisé afin d'ajouter un filtre sur une fonction de groupement;
- Joue un rôle similaire à la clause WHERE;

```
SQL> SELECT cLangue, sum(1) C1, count(*)
FROM Usager u, Langue l
WHERE u.pLangue = l.pLangue
GROUP BY cLangue
HAVING count(*) > 1;
```

tri - la clause ORDER BY

- Utilisé pour ordonner notre projection selon un ordre précis;
- Il est possible d'utiliser l'index dans la projection;

```
SQL> SELECT cLangue, sum(bActif), count(*)
FROM Usager u, Langue l
WHERE u.pLangue = l.pLangue
GROUP BY cLangue
HAVING count(*) > 1
ORDER BY 3;
```

```
SQL> SELECT cLangue, sum(bActif), count(*)
FROM Usager u, Langue l
WHERE u.pLangue = l.pLangue
GROUP BY cLangue
HAVING count(*) > 1
ORDER BY cLangue, 3;
```

sous-requêtes - avec IN

Une sous-requête dans l'opérateur IN ( ). La projection de la sous-requête est ce qui sera évalué par IN (...). Ce qui veut dire que les valeurs retournées (projetées) par la sous-requête sont celles prises en compte.

```
SQL> SELECT u.*
FROM Usager u
WHERE pLangue IN (
  SELECT pLangue
  FROM Langue
  WHERE cLangue IN ('Francais','Anglais')
  );
```

sous-requêtes - avancée

Une sous-requête (est ci-bas une projection) qui retourne un nombre filtré de tuples afin de créer une jointure avec une autre entité. La sous-requête produit R1' qui sera utilisée dans une jointure avec R2.

- $R1 = l, R2 = u$
- $R1' = \Pi_{pLangue,cLangue}\ (\sigma_{pLangue=1} R1)$
- $R = \Pi_{cUsager,cLangue}\ (R1' \bowtie_{pLangue,pLangue} R2)$

```
SQL> SELECT u.cUsager, l.cLangue
FROM Usager u,
     (SELECT pLangue, cLangue FROM Langue WHERE pLangue > 1) l
WHERE u.pLangue=l.pLangue;
```

projection - avec la clause DISTINCT

Le DISTINCT joue un rôle similaire au GROUP BY. Son avantage est qu'il est court à écrire. DISTINCT élimine les doublons dans une requête simple sans GROUP BY. Il liste les combinaisons uniques de tuples ceci en évaluant tous les attributs.

```
SQL> SELECT DISTINCT u.bActif, l.cLangue
FROM Usager u,
     (SELECT pLangue, cLangue FROM Langue WHERE pLangue > 1) l
WHERE u.pLangue=l.pLangue
ORDER BY 2,1;
```

Aide mémoire

Structure d'une requête SQL (* optionnel *)

- SELECT < attribut | valeur | fonction > [,]
- FROM < table | view | sous-requête > [,]
- * WHERE condition *
- * GROUP BY attribut *
- * HAVING condition *
- * ORDER BY nombre | attribut *
- ;

```
SELECT < attribut | valeur | fonction > [,]
FROM < table | view | sous-requête > [,]
[ WHERE condition ]
[ GROUP BY attribut ]
[ HAVING condition ]
[ ORDER BY nombre (index colonne) | attribut ] ;
```