COURS DE PROGRAMMATION WEB
CREATION ET GESTION DE BASE DE DONNEES

M. Réné ZEMBA

(226) 64 00 20 53 / 53 90 26 72

zembarene@gmail.com

Ingénieur Statisticien Economiste (ENSAE-Sénégal)

24/01/2025

Année académique 2022-2023

Chapitre IV

SQL

Structured Query Language

24/01/2025

Année académique 2022-2023

Thème 2 : Le langage SQL

Thème 2 : Le langage SQL (suite)

2 Mise à jour de données

- 2.1 Insertion de n-uplets
  - a) 1ère forme : insertion de valeurs
  - b) 2ième forme : insertion dans une relation du résultat d'une sous-requête
- 2.2 Modification des valeurs de n-uplets
- 2.3 Suppression de n-uplets

3 Le langage de description de données

- 3.1 L'administration des tables
  - a) Création d'une table
  - b) Modification du schéma d'une table
  - c) Destruction d'une table
- 3.2 Administration et utilisation des vues
  - a) Création d'une vue
  - b) Utilisation des vues : consultation d'informations
  - c) Destruction d'une vue
- 3.3 L'administration des droits d'accès
  - a) Les utilisateurs d'une base de données
  - b) Octroi et retrait de droits d'usage
    - Octroi de droits
    - Retrait de droits
  - c) Un exemple

4 Exercices

Conclusion sur les langages relationnels

Annexe :

script SQL Oracle de création des 4 relations du Centre de vacances

0 Principe

SQL (Structured Query Language)

Version commerciale de SEQUEL (IBM 74), projet de langage de requêtes pour non informaticiens

Objectif :

Offrir à l'utilisateur les moyens d'exprimer une requête en décrivant seulement les données recherchées au moyen d'une expression logique (assertion) sans détailler le moyen de les trouver:

1- L'utilisateur **qualifie** les données à extraire

2- Le système **construit la procédure d'extraction des données**

SQL présente les données extraites sous forme de **tableaux à 2 dimensions :**

- Les colonnes (**columns**) peuvent être:
  - des attributs de la base de données
  - des expressions calculées à partir des attributs
- Les lignes sont nommées **rows**

Chaque case d'un tableau SQL ne peut contenir au plus **qu'une seule valeur**

Les tableaux SQL sont construits à partir de relations « **table** » et d'autres structures :

vues externes **view**

index **index**

synonymes **synonym**

Langage standard et normalisé de description et de manipulation de données commun

- à la quasi totalité des SGBD relationnels actuels

ORACLE, POSTGRES, SYBASE, INFORMIX, SQL-SERVER, UNIFY, MY-SQL, MariaDB (logiciel libre)…

- et aux SGBD plutôt réservés aux BD personnelles

ACCESS …

SQL fonctionne selon 2 modes:

- interprété, décrit dans ce chapitre,
- intégré à un langage de programmation

SQL se compose :

- d'un Langage de Manipulation de Données (sections 1 et 2)
- et d'un Langage de Définition de Données (section 3)

Notations :

Une table et ses colonnes R(C1, C2, . . ., Cn)

Nom complet d'une colonne C d'une table R: **R . C - le nom de la table peut être omis s'il n'y a aucune ambiguîté**

Les **mots clés** du langage sont en **bleu et gras**

Les termes entre [ ] sont facultatifs

Remarques

On donne ici la syntaxe Oracle de SQL, très proche de la norme internationale.

L'exemple du « Centre de vacances » a été modifié pour tenir compte des contraintes lexicales du langage:

- aucun accent, caractère "espace", signe de ponctuation dans les noms d'objet, excepté le « _ »
- commentaire : texte en vert qui commence par « -- » jusqu'à la fin de la ligne

Les noms des relations et des attributs sont donc ceux d'un MPD, Modèle Physique des Données adapté du MLD sur la Modélisation des données

On reprend les requêtes traitées dans le langage algébrique –Thème 1 ci-dessus

1 Recherche / extraction de données

Une « **requête** » est, le plus souvent, écrite en **une seule « phrase » ou « instruction »**, composée de « **clauses** ». Les données extraites par la requête sont présentées sous la forme d'un tableau.

1.1 La structure de base d'une requête – le « bloc de qualification »

Select	Ci, Cj, . . ., Cn	→	liste de colonnes
From	R	→	nom de la table
Where	expression logique ;	→	assertion : expression que vérifient les n-uplets de R qualifiés

analogue à :

une **restriction** algébrique - clause **Where**

+

une **projection** algébrique sur une liste de colonnes - clause **Select**.

Les résultats sont présentés sous la forme d'un **tableau** qui ressemble à une relation sans clé primaire.

Une colonne Ci du tableau résultat est

- soit un attribut de R
- soit une expression calculée à partir d'attributs de R

C1	C2	Cn	*en-tête*
----------	---------- …	----------	
*****	******	*****	*lignes du résultat*
*****	******	*****	
xxx lignes traitées			commentaire

Le résultat peut contenir des lignes identiques contrairement à la projection algébrique

Exemple: requête 4

"Quels sont les dates et les heures des séances de tennis ?"

```
Select  date_seance, heure_debut, heure_fin
From    Seance
Where nom_activite = 'tennis'; -- le caractère « ' » est utilisé pour les
                               -- valeurs d'une donnée textuelle
```

Exemple: requête 3

"Quels sont les noms des Toulousains nés après 95 ?"

```
Select     nom
From       Client
where   (adresse = 'toulouse')
          and (date_de_naissance >= '01/01/95'); -- les dates peuvent être
                                                 -- traitées comme des textes
```

a) Elimination des lignes identiques

```
Select distinct     C1, C2, …
```

élimine les lignes identiques dans le tableau résultat - comme les projections algébriques

b) Formes simplifiées

- projection algébrique

```
Select distinct     Ci, Cj, . . ., Cn
From                R ;
```

Si le mot clé **distinct** est présent, le résultat ne contient qu'un seul exemplaire des lignes, comme pour une projection algébrique. Sinon, le tableau résultat contient la table R en entier limitée aux seules colonnes Ci, Cj, . . ., Cn

Exemple: requête 1

"Quelle est la liste des noms et adresses des clients?"

```
Select      nom, adresse
From        Client ;
```

- restriction algébrique

```
Select      *           -- toutes les colonnes
From        R
Where       expression logique ;
```

Exemple : requête 2

"Quelles sont les propriétés des activités qui coûtent moins de 80 €?"

```
Select      *
From        Activite
Where       cout < 80 ;
```

1.2 Produit cartésien et jointure algébriques

Recherche et édition de colonnes issues de relations différentes

Select colonnes de R1, …, colonnes de Rk
From R1, R2, … , Rk
Where expression logique ;

interprétation
Soient r1, r2, … , rk des n-uplets respectifs de R1, R2, … Rk
Une combinaison (r1, r2, …, rk) est « qualifiée » si elle satisfait l'expression logique

tableau résultat:
projection de (r1, r2, . . . , rk)
sur colonnes de R1, …, colonnes de Rk
La requête est semblable à la combinaison de requêtes élémentaires :

Produit Cartésien + Restriction + Projection

Exemple : requête 13
"Quelle est l'évaluation détaillée de la séance n° 1 ?"

```
Select  Client.n_client, Client.nom, Inscription.appreciation   -- Projection
From    Client, Inscription                                    -- Produit
Where   (Client.n_client = Inscription.n_client)               -- Critère de jointure
  and   (Inscription.n_seance = 1) ;                           -- Restriction
```

1.3 Les alias

Dans une requête, un **alias** est un nom symbolique qui remplace une table, une vue ou une colonne

a) Alias de table ou vue

Variable "de parcours" d'une table ou vue :

Type de la variable = type d'un n-uplet de cette table ou vue

Exemple : requête 13 alias

```
Select   C.n_client, C.nom, I.appreciation -- Projection
From     Client C, Inscription I           -- Produit
Where    (C.n_client = I.n_client)         -- Critère de jointure
  and    (I.n_seance =1) ;                 -- Restriction
```

C et I sont 2 alias.

C et I représentent respectivement un n-uplet de Client et de Inscription

Dans une requête, on peut définir plusieurs alias indépendants pour représenter une table ou vue

Exemple : requête 14

" Quelles sont les séances organisées après la séance n° 1 ?"

```
Select   ApresS1.n_seance
From     Seance  S1, Seance ApresS1
Where    (ApresS1.date  >  S1.date)     -- critère de jointure
  and    (S1.n_seance   =   1);                    -- filtre de restriction
```

S1 et ApresS1 sont 2 alias qui représentent la relation Seance.

S1 permet de traiter la restriction n_seance = 1

ApresS1 permet de traiter la restriction date > "date de la séance 1"

Le "critère de jointure" et le filtre de restriction sont regroupés dans la même clause "**where**" portant sur le produit des 2 relations

b) Alias de colonne

Type de la variable = type de la colonne

Exemple : requête 2 bis

"Quelle sont les caractéristiques des activités qui coûtent moins de 80 €?"

```
Select    nom_activite [as] "l'activité", cout [as] "son coût", prix [as] "et  son prix"
From      Activite ;
```

l'activité	son coût	et son prix
golf	100	120
ski	80	100
tennis	50	60
volley	30	40

Le mot clé **as** est facultatif

Les alias doivent être encadrés par le caractère " et non par '

1.4 Tri des lignes extraites par une requête : la clause "order by"

Introduit un critère de tri sur une ou plusieurs colonnes dans l'ordre croissant "ascending" (valeur par défaut) ou décroissant "descending"

Exemple: requête 1 bis

"Quelle est la liste des noms et adresses des clients classée par ordre alphabétique et dans l'ordre décroissant des dates de naissance ?"

```
Select      *
From        Client
Order by    nom                    asc,
            date_de_naissance      desc;
```

Remarques

- **Order by** est toujours la dernière clause d'une requête car on trie les lignes du résultat de la requête
- **Order by** déclenche le tri des lignes extraites : le critère de tri ne peut donc mentionner que sur des colonnes de la clause "Select"

1.5 Imbrication de requêtes : les sous requêtes

Principe initial

Les colonnes cherchées sont issues d'une seule table ou vue

Les lignes sont qualifiées à partir d'informations provenant d'autres tables ou vues désignées aussi par une requête.

a) L'opérateur "in"

Exemple : requête 6 bis

"Quelles sont les dates des activités de tennis et de ski"

```
Select distinct     date_seance
From                Seance
Where               nom_activite in  ('tennis','ski'); -- énumération de valeurs de référence
```

Exemple : requête 5

"Quels sont les noms des clients inscrits à la séance 3 ?"

```
Select      nom
From        Client
Where       n_client    in
            (Select     n_client                -- sous-requête qui produit les valeurs de référence
             From       Inscription  -- "les n° s des clients inscrits à la séance n° 3"
             Where      n_seance = 3            );
```

Remarque

Un "produit" peut être utilisé à la place d'un opérateur "in"

Exemple : requête 5 bis

"Quels sont les noms des clients inscrits à la séance 3 ?"

```
Select      C.nom
From        Client C, Inscription I
Where       (C.n_client = I.n_client)  -- semblable à une
   and      (I.n_seance  = 3) ;        -- jointure algébrique
```

Remarques

Le choix entre les 2 expressions de la requête 5 dépend de leur lisibilité et des performances de l'interpréteur du langage

Dans tous les systèmes du marché, il existe un **optimiseur de requêtes** qui rend le critère de performance moins fondamental

L'usage de la version avec « in » explicite en apparence la procédure de collecte des données

b) L'opérateur "exists"

Exemple: requête 11

"Quelles sont les activités organisées au moins une fois ?"

```
Select          nom_activite, cout, prix
From            Activite
Where   exists   -- existe-t-il une séance ?
   ( Select     *
     From       Seance
     Where      Seance.nom_activite  = Activite.nom_activite ) ;
-- pour chaque activité de la table Activite, existe-t-il une séance pour
-- laquelle cette activité est organisée ?
```

La clause Where est une sorte de critère de jointure entre la table Seance de la sous-requête et la table Activite de la requête principale

Exemple : requête 12

"Quelles sont les activités qui ne sont pas organisées ?"

```
Select          nom_activite, cout, prix
From            Activite
Where   not exists  -- cette activité n'est elle pas du tout  organisée ?
   ( Select     *
     From       Seance
     Where      Seance.nom_activite  = Activite.nom_activite );
```

c) Division algébrique

Exemple : requête 10

"Quels sont les clients inscrits à toutes les séances ?"

Il est impossible de traduire directement cet énoncé en une seule requête.

Comme dans le langage algébrique, on transforme cet énoncé en une forme équivalente :

"Quels sont les clients pour lesquels il n'existe aucune séance à laquelle ils ne sont pas inscrits ? "

```
Select          n_client, nom
From            Client
Where   not exists
        ( Select        *               -- les séances
          From          Seance
          Where not exists
                ( Select        *       -- les inscriptions
                  From          Inscription
                  Where         ( Inscription.n_seance = Seance.n_seance)
                        and     ( Inscription.n_client = Client.n_client))) ;
```

La clause Where recherche des n-uplets de la table Inscription liés à la fois à Client et Séance par ses clés étrangères. Si pour 1 client, il n'existe pas de séance à laquelle il n'est pas inscrit, ce client est dans la liste des clients inscrits à toutes les séances

1.6 Opérateurs ensemblistes : Intersect, Union, Minus

Intersection, union, ou différence ensembliste entre les résultats de 2 "Select" construisant 2 tableaux dont les colonnes ont des noms identiques ou qui sont compatibles entre elles

Exemple : requête 6

"Quelles sont les activités qui ont été organisées à la fois le 1er août et le 2 septembre 2021 ?"

```
Select    nom_activite
From      Seance
Where     date_seance = '01/08/2021'
Intersect
Select    nom_activite
From      Seance
Where     date_seance = '02/09/2021' ;
```

Autre solution

```
Select    S1.nom_activite
From      Seance S1, Seance S2
Where     (S1.nom_activite = S2 nom_activite)
    and (S1.date_seance = '01/08/2021')
    and (S2.date_seance = '02/09/2021') ;
```

1.7 Opérateurs de calculs sur des colonnes numériques

Les valeurs d'une colonne construite par une requête peuvent être des expressions calculées à partir d'attributs et des opérateurs:

Count	**(**colonne**)**	compte toutes les valeurs non nulles de la colonne
Count	**(distinct** colonne**)**	compte toutes les valeurs non nulles distinctes de la colonne
count	**(*)**	compte les lignes extraites par une requête
sum	**(**colonne**)**	additionne les valeurs d'une colonne numérique
avg	**(**colonne**)**	calcule la moyenne (average) de valeurs d'une colonne numérique
min	**(**colonne**)**	extrait la plus petite des valeurs d'une colonne numérique
max	**(**colonne**)**	extrait la plus grande des valeurs d'une colonne numérique
var	(colonne)	calcule la variance de la colonne
stddev	(colonne)	calcule la déviation standard (racine de la variance)

Exemple : requête 15

"Quel est le nombre d'inscrits à la séance 3 ?"

```
Select    count (*)
From      Inscription
Where     n_seance = 3  ;
```

Exemple: requête 16

"Quel est la note moyenne attribuée par les clients à la séance 3 ?"

```
Select    avg (appreciation)
From      Inscription
Where     n_seance = 3 ;
```

1.8 Les agrégats

a) Clause Group by

Partitionne une relation et permet de spécifier des critères de recherche relatifs à des n-uplets de chaque partition

Exemple : requête 17

"Quelle est la liste des séances avec pour chacune le nombre d'inscrits ?"

```
Select      n_seance,  count (*)
From        Inscription
Group by    n_seance
```

interprétation

1 Constitue autant de groupes de n-uplets de Inscription que de valeurs de la colonne "n_seance"

2 Pour chaque groupe, donc pour chaque valeur distincte de n_seance

- calcule le nombre de n-uplets (les inscriptions)
- édite le n_seance et le nombre d'inscriptions

b) Clause Having

Introduit un critère de sélection à appliquer aux n-uplets de chaque groupe et non à la relation toute entière

Exemple : requête 18

"Quelle sont les séances qui ont enregistré au moins 2 inscrits ?

Editer le résultat selon l'appréciation moyenne"

```
Select      n_seance as "n° séance", count (*) as "nombre d'inscrits",
            avg (appreciation) as "note moyenne"
From        Inscription
Group by    n_seance
Having      count (*) >= 2
Order by 3  desc ;
```

Le critère de restriction de la clause "Having" commande de

- compter le nombre d'inscrits dans chaque séance, et de ne retenir que les valeurs de « n° séance » pour lesquelles ce nombre est >= 2
- calculer la moyenne des notes de chacune de ces séances

n° séance	nombre d'inscrits	note moyenne
3	2	12
1	3	11,66666667

Remarque

Dans la clause "Order by 3", "3" désigne la 3ième colonne du "Select", soit la moyenne des notes

Exemple : requête 10 bis

"Quels sont les clients inscrits à toutes les séances ?"

On cherche de manière équivalente les clients inscrits à un nombre de séances égal au nombre total des séances

```
Select          n_client
From            Inscription
Group by        n_client
Having  count (*)                    -- nombre d'inscriptions
=       ( Select count (*)           -- nombre de séances de la table séance
          From   Seance     ) ;
```

Pour chaque groupe, donc pour chaque client, on compare le nombre d'inscriptions avec le nombre de séances organisées.

Remarque

Cette requête donne le même résultat que la forme avec les 2 "not exists", mais ce n'est qu'un cas particulier.

La forme avec les 2 "not exists" est celle qui donne le bon résultat dans tous les cas.

2 Mise à jour de données

Insert, **Update** et **Delete** s'appliquent à 1 relation, et respectent les contraintes d'intégrité de la base de données

2.1 Insertion de n-uplets

a) 1ère forme : insertion de valeurs

syntaxe

Insert
Into R [(Ai, Aj, . . . , Ap)]
Values (vi, vj, . . . , vp) ;
vk constante ou expression calculée

Exemple : requête 7

"Ajouter l'activité 'plongée'"

Insert
Into Activite (nom_activite, cout)
Values ('plongée', 120) ;

Une valeur "vide" (**Null**) est stockée dans le n-uplet pour l'attribut "prix"

b) 2ième forme : insertion dans R du résultat d'une sous-requête

syntaxe

Insert
Into R (A1, A2, …, Ap)
Select C1, C2, …, Cp
From … ; *-- requête SQL complète où chaque colonne Ci est associée à la colonne Ai*

2.2 Modification des valeurs de n-uplets

syntaxe

Update R
Set Ai = vi, Aj = vj, . . . , Ap = vp
Where expression logique ;

vk constante, expression calculée ou valeur **Null**

L'expression logique qualifie les n-uplets de R pour lesquels il faut appliquer la modification

Exemple : requête 9

"Diminuer de 20 % le prix de l'activité tennis"

Update Activite
Set prix = prix * 0.8
Where nom_activite = 'tennis' ;

2.3 Suppression de n-uplets

syntaxe

Delete

From R

Where expression logique ;

Supprime de R tous les n-uplets qualifiés par l'expression logique

Exemple : requête 8

"Supprimer la séance 3"

Delete

From Seance

Where n_seance = 3 ;

Remarques importantes

1- Supprime la séance 3, si elle n'est liée à aucune inscription

2- On ne peut engager une mise à jour que si elle respecte les contraintes (clés primaires, clés étrangères,…) du MLD

3 Le langage de description de données

Le LDD de **SQL** permet de créer et d'administrer
- les **relations** d'un Modèle Physique de Données construit à partir d'un MLD → **table**
- les **vues externes** construites sur ce MPD → **view**
- Des éléments supplémentaires du MPD comme :

Les **index** → **index**
Les **droits d'accès** (octroi et retrait) sur ces objets → **grant, revoke**

3.1 L'administration des tables

a) Création d'une table

Permet l'enregistrement de la description du schéma d'une relation adapté au MPD de SQL:

- liste des attributs et leur type : numérique, texte, date, …
- attribution possible de valeurs indéterminées "null" à des attributs
- désignation de la clé primaire
- désignation des clés étrangères
- expression de certaines contraintes d'intégrité

syntaxe

Create table relation
(attribut1 type1 **[not null]**,
…
attributn typen **[not null]**,

-- définition de la clé primaire

[**constraint** nom_de_contrainte] **primary key** (attributj, …attributk),

-- définition des clés étrangères

[**constraint** nom_de_contrainte] **foreign key** (attributx)
references relationa (attributa),

…

[**constraint** nom_de_contrainte] **foreign key** (attributy)
references relationz (attributz),

-- définition de contraintes annexes

[**constraint** nom_de_contrainte] **check** (condition sur des attributs)
) ;

Exemple : requête 19

requête de création de la relation "Inscription" (sous Oracle)

```
Create  table       Inscription
(           n_client              number (8) not null,
            n_seance              number (8) not null,
            appreciation                     number (5,2),
            inscription_cle                  primary key      (n_client, n_seance),
            inscription_cle_client           foreign key      (n_clientY
                                             references       Client (n_client),
            inscription_cle_seance           foreign key      (n_seance)
                                             references       Séance (n_seance),
            inscription_appreciation
                                             check            (appreciation < 18
                                                              and appreciation > 5 )
) ;
```

On trouvera en annexe 2 le code SQL de création du MPD de la base de données du centre de vacances et de création des n-uplets qui servent d'exemple au support de cours

b) Modification du schéma d'une table

2 options :

1- ajouter des attributs à une relation

2- changer le type ou l'indétermination possible des attributs

syntaxe

1- **alter table** relation **add** (attribut type [null / not null], …) ;

2- **alter table** relation **modify** (attribut [nouveau type] [null / not null],…) ;

3- **alter table** relation **drop** (attribut);

c) Destruction d'une table

Supprime le schéma et le contenu d'une relation

syntaxe

Drop table relation ;

Remarque

Comme pour les mises à jour, on peut modifier le schéma d'une relation ou détruire une relation si l'opération respecte les contraintes du Modèle Logique des Données

3.2 Administration et utilisation des vues

Vue : définition

Relation "virtuelle"

- résultat d'une requête
- ne contient aucun n-uplet
- utilisable, en mode consultation, comme n'importe quelle relaabletion de la BD

a) Création d'une vue

syntaxe

Create view vue (attribut1, …, attributp)
as **Select** colonne1, …, colonnep
From … ; -- *requête SQL ordinaire*

Un attribut "attribut i" de la vue représente la colonne "colonne i" de même rang de la requête qui "calcule" la vue

Exemple: requête 20

Requête de création de la vue "Planning" :

la vue contient le nom de l'activité, ses séances , le nombre de clients inscrits à chaque séance, et le bénéfice de chaque séance

```
Create view     Planning
                (nom_activite, n_seance, date_s, heure_debut, inscrits,benefice)
As      Select  A.nom_activite, S.n_seance, S.date_s, S.heure_debut,
                        count ( * ), count ( * ) * ( A.prix - A.cout )
        From    Activite A, Seance S, Inscription I
        Where   ( A.nom_activite = S.nom_activite )
          and   ( S.n_seance = I.n_seance )
        Group by A.nom_activite, S.n_seance, S.date_s,S.heure_debut,
                        A.cout, A.prix;
```

b) Utilisation des vues : consultation d'informations

- donner une vision plus lisible d'un schéma, adaptée à un utilisateur
- filtrer les accès à un schéma par la pose de droits d'accès sur une vue
- simplifier une requête complexe en compilant dans une vue une partie de la requête

Exemple : requête 21

"Quel est le bénéfice de l'activité "tennis" ?"

```
Select    benefice
From      Planning
Where     nom_activite = 'tennis' ;
```

Remarques

• L'exécution d'une requête qui invoque une vue déclenche le "calcul" de la vue, c'est-à-dire l'exécution de la requête qui définit la vue

• On peut exécuter des mises à jour au travers d'une vue, à condition que:

- la vue soit calculée sur une seule relation
- la requête ne comporte
  - ni clause "group by"
  - ni clause "where" invoquant des attributs d'autres relations
  - ni référence à des fonctions de calcul (count, …)

Contre exemple

On ne peut pas mettre à jour la relation Seance à travers la vue Planning

c) Destruction d'une vue

syntaxe

Drop view vue

Remarque

La destruction d'une vue n'a aucun effet sur la ou les relations sur laquelle elle est bâtie

3.3 L'administration des droits d'accès

a) Les utilisateurs d'une base de données

Ils sont répertoriés par le SGBD, avec:

- un identifiant interne
- un mot de passe
- un nom d'usage, destiné aux autres utilisateurs

Le créateur d'une table, ou vue, est le propriétaire (owner) de cette table

b) Octroi et retrait de droits d'usage

Le propriétaire d'une table peut octroyer et retirer à d'autres utilisateurs le droit de la manipuler

- **Octroi de droits**

syntaxe

Grant [**all** / liste de privilèges]
On table ou vue 1, …, table ou vue n
To [**public** / liste d'utilisateurs]
[With grant option];

"liste de privilèges" = instructions SQL: alter, delete, insert, select, update
"**all**": toutes ces commandes
"**public**": tous les utilisateurs répertoriés
"**with grant option**": droit aux utilisateurs d'octroyer aussi des droits

• **Retrait de droits**

syntaxe

Revoke [**all** / liste de privilèges]
On table ou vue 1, …, table ou vue n
From [**public** / liste d'utilisateurs] ;

c) Un exemple

3 utilisateurs de nom d'usage " Menelas", " Paris " et " Helene « (inspiré de L'Iliade d'Homère

" Menelas" est le propriétaire des table de la BD "Centre de vacances"

• " Menelas " accorde à " Paris " le droit de consulter "Seance", "Inscription" et "Activite"

Grant **Select**
On Seance, Inscription, Activite
To Paris ;

• "Menelas" accorde à "Helene" le droit d'utiliser toutes les instructions SQL sur la table Activite et aussi le droit d'accorder des droits sur cette relation à d'autres utilisateurs

Grant **all**
On Activite
To Helene
With grant option ;

• "Helene" accorde à "Paris" le (nouveau) droit de modifier l'attribut "prix" de la table "Activite" et de supprimer des lignes de cette relation

Grant **update (nom_activite), delete**
On Activite
To Paris ;

• "Menelas" retire ses droits à "Helene"

Revoke **all**
On Activite
From Helene ;

" Helene " ne peut plus utiliser " Activite "

" Paris " perd les droits que lui a accordés " Helene " et ne conserve que les droits de consultation accordés par " Menelas "

Remarques

- Dans tous les SGBD, l'administrateur d'une BD a, par défaut, tous les droits sur toutes les données
- Pour limiter l'accès à des attributs et des lignes d'une table, on utilise les vues externes

Exemple

"Menelas" crée la vue "Activites_possibles" pour rendre publique la table "Activite" sans les prix

```
Create view     Activites_possibles (nom)
        As      Select          nom_activite
                From            Activite ;

Grant   select
On              Activites_possibles
To              public ;
```

4 Exercices

A partir de l'exercice 4.3 S.I. de gestion d'une bibliothèque avec les exemplaires et les archives étudié dans la partie 1 sur les Modèle de données, traduire dans le langage SQL :

- les 8 requêtes de l'exercice 10 du thème 1 "Le langage algébrique" p. 21
- les 6 requêtes suivantes

9 Nom du ou des auteurs les plus empruntés de la bibliothèque

10 Titre du ou des ouvrages les plus empruntés de la bibliothèque

11 Noms et adresses des abonnés avec le nombre d'exemplaires empruntés pendant l'année 2020 présentés dans l'ordre décroissant du nombre d'exemplaires

12 Titres des ouvrages qui ont fait l'objet de plus de 100 emprunts en 2020

13 Editeurs qui ont fait l'objet d'au moins 1 emprunt présentés dans l'ordre décroissant du nombre d'emprunts

14 Identifier une information qu'on pourrait trouver dans cette base de données qui pourrait présenter un intérêt si on parvenait à l'extraire : vous indiquerez le texte d'une requête SQL qui permet de l'obtenir

Conclusion sur les langages relationnels

2 types de langages

- **algébrique**

Langage procédural :

Une requête est un enchaînement d'opérations choisi parmi un jeu d'opérateurs de base sur les relations

Il existe des algorithmes d'optimisation de requêtes

- **assertionnel**

Une requête est une description du résultat cherché à l'aide d'une expression logique

Il n'y a plus aucune procédure explicite de recherche

Annexe

```
--
-- script Oracle de création des 4 tables du MPD du Centre de vacances
--
create table            activite
(           nom_activite        varchar2 (20),  -- type texte de 20 caractères max
            cout                number (6,2), -- type décimal de 6 caractères
                                -- dont 2 après la virgule
            prix                number (6,2), -- idem
            primary key         (nom_activite)
) ;
-- Insertion de chaque n-uplet
insert
into activite
values ('golf', 100, 120) ;
insert
into activite
values ('ski', 80, 100) ;
insert
into activite
values ('tennis', 50, 60) ;
insert
into activite
values ('volley', 30, 40) ;
commit ; -- valide les 4 insertions
```

```
create table          client
(          n_client              number (8),
           nom                          varchar2 (20),
           adresse                      varchar2 (20),
           date_de_naissance  date,
           primary key (n_client)
);
insert
into client
values (1, 'pierre', 'toulouse', '30/01/1995') ;
insert
into client
values (2, 'marie', 'toulouse', '02/09/1998') ;
insert
into client
values (3, 'jules', 'toulouse', '05/10/1990') ;
commit ;
```

```
create table        seance
(           n_seance number (6),
            nom_activite        varchar2 (20),
            date_seance         date,
            heure_debut         number (5,2),
            heure_fin           number (5,2),
            primary key (n_seance),
            foreign key (nom_activite)
                        references Activite(nom_activite) -- construction de la clé
                                                          -- étrangère
);
insert
into seance
values (1, 'tennis', '01/08/2021', 10, 12);
insert
into seance
values (2, 'tennis', '02/09/2021', 12, 14);
insert
into seance
values (3, 'ski', '03/09/2021', 10, 12);
insert
into seance
values (4, 'golf', '01/08/2021', 14, 18);
commit;
```

```
Create  table  Inscription
(              n_client                    number (6) not null,
               n_seance                    number (6) not null,
               appreciation   number (5,2),
               primary key                 (n_client , n_seance),
               foreign key                 (n_client) references Client(n_client),
               foreign key    (n_seance) references Seance(n_seance),
               check                       (appreciation <= 18 and appreciation >= 5)
) ;
insert
into inscription
values (1, 1, 12);
insert
into inscription
values (1, 2, 15);
insert
into inscription
values (1, 3, 9);
insert
into inscription
values (1, 4, 15);
insert
into inscription
values (2, 1, 13);
insert
into inscription
values (2, 3, 15);
insert
into inscription
values (3, 1, 10);
commit;
```

La suite est à rechercher pour complément

FIN