Réalisée par : Karima BELABED

Encadrée par : Mme Ibtissam CHETOUANI

Résumé SGBD1 et SGBD2

SGBD1

L'instruction CREATE DATABASE est utilisée pour créer une nouvelle base de données SQL :

```
create database Nom_Base_de_Données;
```

L'instruction DROP DATABASE est utilisée pour supprimer une base de données SQL existante :

```
drop database Nom_Base_de_Données;
```

L'instruction CREATE TABLE est utilisée pour créer une nouvelle table dans une base de données :

```
create table Nom_Table(
    column1 int primary key,
    column2 varchar(50),
    column3 date
)
```

L'instruction DROP TABLE est utilisée pour supprimer une table existante dans une base de données :

```
drop table Nom_Table
```

L'instruction TRUNCATE TABLE est utilisée pour supprimer les données dans une table, mais pas la table elle-même :

```
truncate table Nom_Table
```

L'instruction ALTER TABLE est utilisée pour ajouter, supprimer ou modifier des colonnes dans une table existante.

L'instruction ALTER TABLE est également utilisée pour ajouter et supprimer diverses contraintes sur une table existante.

Pour ajouter une colonne dans une table, utilisez la syntaxe suivante :

```
alter table Nom_Table
add Nom_Colonne int
```

Pour supprimer une colonne dans une table, utilisez la syntaxe suivante :

```
alter table Nom_Table
drop column Nom_Colonne
```

Pour modifier le type de données d'une colonne dans une table, utilisez la syntaxe suivante :

```
alter table Nom_Table
alter column Nom_Colonne varchar(50)
```

Les contraintes SQL sont utilisées pour spécifier des règles pour les données d'une table.

Des contraintes peuvent être spécifiées lorsque la table est crée avec l'instruction CREATE TABLE ou après la création de la table avec l'instruction ALTER TABLE.

Les contraintes suivantes sont couramment utilisées dans SQL :

NOT NULL : Garantit qu'une colonne ne peut pas avoir une valeur NULL.

UNIQUE : Vérifie que toute les valeurs d'une colonne sont différentes.

PRIMARY KEY : Une combinaison d'un NOT NULL et UNIQUE. Identifie de manière unique chaque ligne d'une table.

TOUCHE ETRANGERE : Identifie de manière une ligne / un enregistrement dans une table.

CHECK : Vérifie que toutes les valeurs d'une colonne satisfont à une condition spécifique.

DEFAULT : Définit une valeur par défaut pour une colonne lorsqu'une valeur n'est spécifique.

INDEX : Utilisé pour créer et extraire les données de la base de données très rapidement.

Exemples :

```
create table Personne(
    P_ID int not null primary key identity,
    Nom varchar(40),
    Prenom varchar(40),
    Age int check(age>=18),
    Ville varchar(50) default'Taourirt'
)

create table Stagiaire(
    S_ID int not null primary key,
    S_Filiere varchar(55),
    P_ID int foreign key references  Personne(P_ID)
)
```

Pour créer une contrainte UNIQUE sur la colonne « P_ID » lorsque la table et déjà créée :

```
alter table Personne
add unique(P_ID)
```

Pour nommer une contrainte UNIQUE et pour définir une contrainte UNIQUE sur plusieurs colonnes :

```
alter table Personne
add constraint UC_Personne unique(P_ID,nom)
```

Pour supprimer une contrainte UNIQUE :

```
alter table Personne
Drop constraint UC_Personne
```

Pour créer une contrainte PRIMARY KEY sur la colonne « P_ID » lorsque la table est déjà créée :

```
alter table Personne
add primary key(P_ID)
```

Pour autoriser la dénomination d'une contrainte PRIMARY KEY et pour définir une contrainte PRIMARY KEY sur plusieurs colonnes :

```
alter table Personne
add constraint PK_Personne primary key(P_ID,Nom)
```

Pour supprimer contrainte PRIMARY KEY :

```
alter table Personne
drop constraint PK_Personne
```

Pour créer une contrainte FOREIGN KEY dans la colonne « P_ID » lorsque la table stagiaire est déjà créée :

```
alter table Stagiaire
add foreign key(P_ID) references Personne(P_ID)
```

Pour autoriser la dénomination d'une contrainte FOREIGN KEY et pour définir une contrainte FOREIGN KEY sur plusieurs colonnes :

```
alter table Stagiaire
add constraint FK_PersonneStagiaire foreign key(P_ID) references Personne(P_ID)
```

Pour supprimer une contrainte FOREIGN KEY :

```
alter table Stagiaire
drop constraint Fk_PersonneStagiaire
```

Pour créer une contrainte CHECK dans la colonne « Age » lorsque la table est déjà créée :

```
alter table Personne
add check(Age>=18)
```

Pour autoriser la dénomination d'une contrainte CHECK sur plusieurs colonnes :

```
alter table Personne
add constraint CHK_Personne check(Age>=18 and Ville='trt')
```

Pour supprimer une contrainte CHECK :

```
alter table Personne
drop constraint CHK_Personne
```

Pour créer une contrainte DEFAULT dans la colonne « Ville » lorsque la table est déjà créée :

```
alter table Personne
add constraint DF_Ville default'Taourirt' for Ville
```

Pour supprimer une contrainte DEFAULT :

```
alter table Personne
drop constraint DF_Ville
```

Pour créer INDEX :

```
create index Nom_Index
on Personne(Nom,Prenom)
```

Pour créer UNIQUE INDEX :

```
create unique index Nom_Index
on Personne(Nom,Prenom)
```

L'instruction DROP INDEX est utilisée pour un index dans une table :

```
drop index Personne.Nom_Index
```

L'instruction SQL suivante définit la colonne « P_ID » comme un champ de clé primaire auto-incrément dans la table « Personne » :

```
create table Personne(
    P_ID int identity(1,1) primary key,
    Nom varchar(40),
    Prenom varchar(40),
    Age int
)
```

Pour insérer un nouvel enregistrement dans le tableau « Personne », nous ne devrons pas spécifier de valeur pour la colonne « P_ID » (une valeur unique sera ajoutée automatiquement) :

```
insert into Personne(Nom,Prenom)
values('BELABED','Karima')
```

L'instruction suivante sélectionne tous les enregistrements dans la table « Personne » :

```
select * from Personne
```

Les commandes SQL les plus importantes :

SELECT : Extrait les données d'une base de données.

UPDATE : Met à jour les données.

DELETE : Supprime les données.

INSERT INTO : Insère de nouvelles données.

CREATE DATABASE : Crée nouvelle base de données.

ALTER DATABASE : Modifie une base de données.

CREATE TABLE : Crée une nouvelle table.

ALTER TABLE : Modifie une table.

DROP TABLE : Supprime une table.

CREATE INDEX : Crée un index (clé de recherche).

DROP INDEX : Supprime un index.

SELECT :

```
select Nom,Prenom from Personne
select * from Personne
```

SELECT DISTINCT : est utilisée pour renvoyer uniquement de valeurs distinctes (différentes).

```
select distinct Nom,Prenom from Personne
select count(distinct Nom) from Personne
```

WHERE : est utilisée pour extraire uniquement les enregistrements qui remplissent une condition spécifiée.

```
select Nom,Prenom
from Personne
where Nom='BELABED'
```

Les opérateurs AND et OR sont utilisées pour filtrer les enregistrements en fonctions de plusieurs conditions.

L'opérateur NOT affiche un enregistrement si la ou les conditions sont fausses.

ORDER BY : trie les enregistrements par ordre croissant par défaut. Pour trier les enregistrements dans l'ordre décroissant, utilisez le mot clé DESC.

```
select Nom,Prenom
from Personne
order by Nom asc, Prenom desc
```

INSERT INTO : est utilisée pour insérer de nouveaux enregistrements.

```
insert into Personne(Nom,Prenom)
values('BELABED','Karima')
```

Un champ avec une valeur NULL est un champ sans valeur.

IS NULL :

```
select Nom from Personne where Prenom is null
```

IS NOT NULL :

```
select Nom from Personne where Prenom is not null
```

UPDATE : est utilisée pour modifier les enregistrements existants dans une table.

```
update Personne
set Nom='BELGACEM',Prenom='Nawal'
where P_ID=2
```

DELETE : est utilisée pour supprimer les enregistrements existants dans une table.

```
delete from Personne
where P_ID=3
```

Pour supprimer tous les enregistrements :

```
delete from Personne
delete * from Personne
```

SELECT TOP : est utilisée pour spécifier le nombre d'enregistrements renvoyé.

```
select top 5 Nom
from Personne
where Ville='Taourirt'
```

La fonction MIN ( ) renvoie la plus petite valeur de la colonne sélectionnée.

```
select min(Age)
from Personne
where Ville='Oujda'
```

La fonction MAX ( ) renvoie la plus grande valeur de la colonne sélectionnée.

```
select max(Age)
from Personne
where Ville='Oujda'
```

La fonction COUNT ( ) renvoie le nombre de lignes correspondant à un critère spécifié.

```
select count(P_ID)
from Personne
where Ville='Taourirt'
```

La fonction AVG ( ) renvoie la moyenne d'une colonne numérique.

```
select avg(Note)
from Stagiaire
where S_filiere='TSDI'
```

La fonction SUM ( ) renvoie la somme totale d'une colonne numérique.

```
select sum(Prix)
from Article
where code=12
```

L'opérateur LIKE est utilisé dans une clause WHERE pour rechercher un modèle spécifié dans une colonne.

Deux jokers sont utilisés conjointement avec l'opérateur LIKE :

% Le signe pourcentage représente zéro, un ou plusieurs caractères.

_ Le trait de soulignement représente un seul caractère.

```
select *
from Personne
where Prenom like 'Karima'
```

L'opérateur IN vous permet de spécifier plusieurs valeurs dans une clause WHERE.

L'opérateur IN est un raccourci pour plusieurs conditions OR.

```
select Nom,Prenom
from Personne
where P_ID in(1,2)

select Nom,Prenom
from Personne
where P_ID in(select P_ID from Stagiaire)
```

L'opérateur BETWEEN sélectionne les valeurs dans une plage donnée. Les valeurs peuvent être des nombres, du texte, ou des dates.

```
select *
from Personne
where Age between 18 and 26
```

Alias SQL sont souvent utilisés pour donner une table ou une colonne, un nom temporaire.

```
select Nom as nom
from Personne
```

INNER JOIN : sélectionne des enregistrements qui ont des valeurs correspondantes dans les deux tables.

```
select Nom,Prenom,Age
from Personne
inner join Stagiaire on Personne.P_ID=Stagiaire.P_ID

select *
from Personne,Stagiaire
where Personne.P_ID=Stagiaire.P_ID
```

SGBD2 (Transact SQL)

Déclarer une variable :

```
declare @Nom_variable int
```

Affecter une valeur à une variable :

```
select @Nom_variable=40
select @Nom_variable=(select P_ID from Personne)
set @Nom_variable=40
set @Nom_variable=(select S_ID from Stagiaire)
```

Afficher les informations :

```
print Elements_A_Afficher
```

La fonction de conversion : CONVERT (type de conversion, valeur à convertir)

La structure alternative : IF…ELSE

```
if P_ID=1
   begin
        Prenom='karima'
   end
else
   begin
        Prenom='Nawal'
   end
```

L'instruction CASE : permet d'effectuer, selon une condition, une valeur à un champ dans une requête SELECT.

Exemple :

```
select NumArt,DesArt,PUArt,'Observation'=
       case
          when QteEnStock=0 then 'Nom Disponible'
          when QteEnStock>SeuilMinimum then 'Disponible'
          else 'à Commander'
       end
from Article
```

La structure répétitive : WHILE condition BEGIN instruction END

Exemple :

```
while ((select avg(PUArt) from Article)<20) and ((select max(PUArt) from Article)<30)
  begin
     update Article set PUArt=PUArt+(PUArt*10)/100
     select * from Article
  end
select avg(PUArt) as moyenne, max(PUArt) as prix élevé from Article
```

L'instruction IF UPDATE renvoie une valeur TRUE ou FALSE pour déterminer si une colonne à été modifié : IF UPDATE (nom colonne) BEGIN…END

L'instruction GOTO renvoie l'exécution du programme vers un point spécifique repéré par une étiquette : GOTO étiquette.

La transaction :

```
begin Tran[Nom_transaction]
    if PUart=20
    rollback Tran[Nom_transaction]
commit Tran[Nom_transaction]
```

L'instruction RAISERROR affiche un message d'erreur système :

RAISERROR (numéro message | texte message, gravité, état [paramètre 1, paramètre 2, …])

Pour déclarer un curseur :

```
                            static
declare Nom_Curseur cursor keyset for select ...
                            dynamic
```

Pour ouvrir un curseur :

```
open Nom_Curseur
```

Pour lire un enregistrement à partir d'un curseur :

- ✓ Atteindre le premier enregistrement du curseur :

```
fetch first from Nom_Curseur into variable1,variable2, ...
```

- ✓ Atteindre l'enregistrement du curseur suivant celui en cours :

```
fetch next  from Nom_Curseur into variable1,variable2, ...
```

Ou

```
fetch Nom_Curseur into variable1,variable2, ...
```

- ✓ Atteindre l'enregistrement du curseur précédent celui en cours :

```
fetch prior from Nom_Curseur into variable1,variable2, ...
```

- ✓ Atteindre le dernier enregistrement du curseur :

```
fetch last from Nom_Curseur into variable1,variable2, ...
```

- ✓ Atteindre l'enregistrement se trouvant à la position n dans le curseur :

```
fetch absolute n from Nom_Curseur into variable1,variable2, ...
```

- ✓ Atteindre l'enregistrement se trouvant à la position n de la ligne en cours :

```
fetch relative Num_ligne from Nom_Curseur into variable1,variable2, ...
```

La variable système @@FETCH_STATUS est utilisée pour détecter la fin du curseur.

Pour fermer un curseur :

```
close Nom_Curseur
```

Libérer les ressources utilisées par un curseur :

```
deallocate Nom_Curseur
```

Procédure stockée sans paramètre :

CREATE PROCEDURE nom procédure AS instructions

Exécuter : EXEC nom procédure

Procédure stockée avec des paramètres en entrée :

CREATE PROCEDURE nom procédure

Nom paramètre 1 d'entrée type donnée = valeur par défaut,

Nom paramètre 2 d'entrée type donnée = valeur par défaut,…

AS

Instructions

Exécuter : EXEC nom procédure valeur paramètre 1, valeur paramètre 2 …

Procédure stockée avec des paramètres en sortie :

CREATE PROCEDURE nom procédure

Nom paramètre 1 d'entrée type donnée = valeur par défaut,

Nom paramètre 2 d'entrée type donnée = valeur par défaut,…

Nom paramètre 1 de sortie type donnée OUTPUT,

Nom paramètre 2 de sortie type donnée OUTPUT, …

AS

Instructions

Exécuter : DECLARE variable paramètre 1 de sortie type paramètre de sortie

DECLARE variable paramètre 2 de sortie type paramètre de sortie

EXEC nom procédure valeur paramètre 1 d'entrée, valeur paramètre 2 d'entrée …

Nom paramètre 1 de sortie type donnée OUTPUT,

Nom paramètre 2 de sortie type donnée OUTPUT, …

Procédure stockée avec valeur de retour :

CREATE PROCEDURE nom procédure

AS instructions

RETURN valeur de sortie

Exécuter : DECLARE variable de retour Type de variable de retour

EXEC variable de retour=nom procédure

Cryptage d'une procédure stockée :

CREATE PROCEDURE ……WITH ENCRYPRTION AS instructions

Pour supprimer une procédure :

DROP PROCEDURE nom procédure

Pour modifier une procédure :

ALTER PROCEDURE nom procédure AS nouvelles instructions

Fonction qui retourne une valeur scalaire :

CREATE FUNCTION nom fonction (nom paramètre 1 type donnée,..)

RETURNS type de retour

AS instructions

RETURN valeur

Fonction qui retourne une table :

CREATE FUNCTION nom fonction (nom paramètre 1 type donnée,..)

RETURNS nom table Table (champ 1 type 1, …)

AS select …

RETURN

Pour créer un déclencheur (trigger) :

CREATE TRIGGER nom trigger

ON nom table

INSTEAD OF / FOR opération 1, opération 2, …

AS instructions

Pour supprimer un trigger :

DROP TRIGGER nom trigger

Pour modifier un trigger :

ALTER TRIGGER nom trigger

ON nom table

FOR opération 1, opération 2, …

AS nouvelles instructions

__ Bon courage __

__ Karima BELABED __