Résumé SGBD I

✓ **Pour créer une base de données :**

```
CREATE DATABASE BASE_1
ON PRIMARY
( NAME = BASE_1_dat,
FILENAME = 'C:\program files\Microsoft SQL
Server\mssql\data\base_1dat.mdf',
SIZE = 10,
MAXSIZE = 50,
FILEGROWTH = 5 %)
LOG ON
( NAME = BASE_1_log,
FILENAME = 'C:\program files\Microsoft SQL
Server\mssql\data\base_1log.ldf',
SIZE = 5MB,
MAXSIZE = 25MB,
FILEGROWTH = 5MB )
```

✓ **Pour supprimer une base de données :**

```
DROP DATABASE BASE_1
```

✓ **Pour utiliser la base de données :**

```
USE BASE_1
```

✓ **Pour renomer une base de données :**

```
Sp_renamedb 'BASE_1', 'BASE_2'
```

✓ **Pour créer une table :**

```
CREATE TABLE nom_table
(nom_column datatype [DEFAULT expr],...)
```

Exemple :

```
CREATE TABLE clients
 (
 clt_num CHAR(8) PRIMARY KEY, -- cle primaire
 clt_nom VARCHAR(64) NOT NULL, -- vide interdit
 clt_ca INT DEFAULT 0 -- valeur par defaut
 )
```

✓ **Pour modifier une table :**

➢ **Pour ajouter une colonne :**

```
ALTER TABLE nom_table
 ADD nom_colonne type_donnée
```

➢ **Pour retirer une colonne :**

```
ALTER TABLE nom_table
 DROP COLUMN nom_colonne
```

➢ **Pour reconvertir le type de données d'une colonne :**

```
 ALTER TABLE nom_table
 ALTER COLUMN nom_colonne type_donnée [NULL/NOT NULL]
```

➢ **pour connaître les colonnes d'une base :**

```
use test
select * from sys.columns
```

✓ **Pour supprimer une table :**

```
DROP TABLE nom_table
```


Préparé par : ECH-CHEIKH Safaa

- ✓ **Pour vider une table (Supprimer toutes les lignes d'une table et libèrre l'espace occupé par la table.)**

```
TRUNCATE  TABLE   nom_table ;
```

- ✓ **Pour supprimer juste les lignes d'une table :**

```
DELETE TABLE nom_table ;
```

- ✓ **Pour renommer un objet(table,colonne,PS,Trigger,...) :**

```
Sp_rename 'ancien_nom', 'nouveau_nom', Object
```

On peut insérer des commentaires de deux façons :

- sur une ligne, à partir de deux tirets -- ;
- dans un bloc délimité par /* et par */.

Les chaînes de caractères sont délimitées par une quotte ' et si la chaîne contient elle-même une apostrophe ', il suffit de doubler la quotte ''.

- ✓ **Types de données en SQL :**

bit entier dont la valeur est 1 ou 0.
tinyint entier dont la valeur est comprise entre 0 et 255.
smallint entier dont la valeur est comprise entre -2^{15} (-32 768) et $2^{15} - 1$ (32 767).
int entier dont la valeur est comprise entre -2^{31} (-2 147 483 648) et $2^{31} - 1$ (2 147 483 647).

---

decimal Données numériques fixes de précision et d'échelle comprises entre $-10^{38} -1$ et $10^{38} -1$.
numeric Synonyme de **decimal**.

smallmoney Valeurs de données monétaires comprises entre - 214 748,3648 et +214 748,3647, avec une précision d'un dix-millième d'unité monétaire.
money Valeurs de données monétaires comprises entre -2^{63} (-922 337 203 685 477,580 8) et $2^{63} - 1$ (+922 337 203 685 477,580 7), avec une précision d'un dix-millième d'unité monétaire.

---

real Données numériques de précision en virgule flottante comprises entre -3.40E + 38 et 3.40E + 38.
float Données numériques de précision en virgule flottante comprises entre -1.79E + 308 et 1.79E + 308.

---

smalldatetime Données de date et d'heure comprise entre le 1er janvier 1900 et le 6 juin 2079, avec une précision d'une minute.
datetime Données de date et d'heure comprises entre le 1er janvier 1753 et le 31 décembre 9999, avec une précision de trois centièmes de seconde ou de 3,33 millisecondes.

---

char chaîne de caractères de longueur fixe d'un maximum de 8 000 caractères.
varchar chaîne de caractères de longueur variable d'un maximum de 8 000 caractères.
text texte de longueur variable ne pouvant pas dépasser $2^{31} - 1$ (2 147 483 647) caractères.

---

binary données binaires de longueur fixe ne pouvant pas dépasser 8 000 octets.
varbinary Données binaires de longueur variable ne pouvant pas dépasser 8 000 octets.
image Données binaires de longueur variable ne pouvant pas dépasser $2^{31} - 1$ (2 147 483 647) octets.

✓ **Pour créer un nouveau type de données :**

```
sp_addtype nom_de_type, type_physique [,
contrainte_de_valeur_NULL]
```

✓ **Pour ajouter un enregistrement :**

```
INSERT INTO       NomTable
[Liste des colonnes]
VALUES            Liste_des_valeurs
[SELECT …FROM]
```

Exemples :

```
INSERT INTO Etudiant (num, nom, note)
VALUES (5, 'Albert', 12) ;

INSERT INTO Etudiant VALUES (3, 'Paul', '15/08/1985',13.25);

INSERT INTO Etudiant (num, nom)
SELECT num, nom
FROM Personnes
WHERE nom like 'n%' ;
```

✓ **Pour mettre à jour des enregistrements:**

```
UPDATE table          --(la table dans laquelle met `a jour)
SET colonne1 = ..., colonne2 = ...
                      --(les colonnes que l'on met `a jour)
FROM tables           --(les tables de la clause WHERE)
WHERE conditions      --(les lignes `a mettre `a jour)
```

✓ **Pour supprimer des enregistrements :**

```
DELETE      Nom_Table
[FROM]      Eventuellement depuis une sélection
WHERE Expression de la condition
```

✓ **Utiliser les clause CASE et WHILE :**

⇨ CASE :

```
CASE
WHEN expression booléenne THEN
... une instruction ou un bloc
WHEN expression booléenne THEN
... une instruction ou un bloc
... d'autres WHEN ... THEN
ELSE
... une instruction ou un bloc
END
```

⇨ WHILE boucle conditionnelle :

```
WHILE expression booléenne
... une instruction ou un bloc
```

✓ **Pour effectuer une requête de selection :**

```
SELECT colonnes
FROM tables
WHERE condition1 AND/OR condition2
```

➢ Dans les conditions WHERE (reliées entre elles par OR ou AND) on peut utiliser :

⇨ =, <> et tous les opérateurs de comparaison :

```
WHERE nom <> 'Razibus'
```

⇨ une plage de valeurs (bornes incluses) :

```
WHERE salaire BETWEEN 10000 AND 100000
```

⇨ une liste de valeurs :

```
WHERE clt nom IN ('Razibus', 'Fricotin', 'Mironton')
```

⇨ un filtre :

```
WHERE clt nom LIKE 'R%'        -- commencant par R
                               -- % remplace toute s´erie
                               -- de caractères (y compris
                               -- vide)
            LIKE 'R zibus'     -- remplace un caract`ere
            LIKE '%[M-R]'      -- finissant par M,N,O,P,Q ou R
            LIKE '%[^FMR]%'    -- ne contenant ni F ni M ni R
```

- Remarque : on dispose évidemment de :

```
NOT BETWEEN ... AND ...
```

```
NOT IN ...
NOT LIKE ...
```

Par ailleurs, on peut :

⇨ intituler les colonnes (si l'intitulé contient des espaces ou des accents, le délimiter avec des crochets [ ]) :

```
SELECT cmd num AS [numéro de commande], cmd date AS date
```

⇨ n'afficher que des résultats distincts :

```
SELECT DISTINCT(colonne)
```

⇨ n'afficher que les premiers résultats :

```
SELECT TOP 50 ...
```

✓ **Requêtes intégrants plusieurs tables :**

- **Méthode ensembliste :**

```
SELECT liste d'attributs
FROM table1
WHERE attribut de jointure
IN    (SELECT attribut de jointure
FROM table2
WHERE condition)
```

- **Méthode prédicative :**

⇨ **Jointure interne :**

```
SELECT cmd_num, cmd_date
FROM commandes
JOIN clients ON cmd_clt = clt_num -- condition de jointure
WHERE clt_nom = 'Razibus' -- condition de selection
```

⇨ **Jointure externe :**

Imaginons maintenant que l'on dispose de la table clients plus qui contient les colonnes clt_num, clt_adresse, clt_email, clt_telephone et que l'on veuille afficher la liste complète des clients ainsi que leur renseignements complémentaires s'ils existent.
La première idée consiste à effectuer la requête suivante :

```
SELECT a.clt_nom, b.clt_adresse, b.clt_email, b.clt_telephone
FROM clients AS a JOIN clients_plus AS b
ON a.clt_num = b.clt_num
```

Problème : ne s'affichent que les clients ayant des informations complémentaires. La solution consiste à rendre facultative la jointure avec la table de droite.

```
SELECT a.clt_nom, b.clt_adresse, b.clt_email, b.clt_telephone
FROM clients AS a LEFT JOIN clients_plus AS b
ON a.clt_num = b.clt_num
-- jointure facultative gauche
```

Attention : si c'est la table de droite qui est facultative, alors il s'agit d'une jointure externe gauche.
Autre cas : on veut la liste des adresses électroniques de la table clients plus mais parfois il n'y a aucun client de rattaché.

```
SELECT b.clt_email, a.clt_nom
FROM clients AS a RIGHT JOIN clients_plus AS b
ON a.clt_num = b.clt_num
-- jointure facultative droite
```

Dernier cas : on veut la liste des clients dont on a soit le nom, soit l'e-mail.

```
SELECT a.clt_nom, b.clt_email
FROM clients AS a FULL OUTER JOIN clients_plus AS b
ON a.clt_num = b.clt_num
-- jointure facultative dans les deux sens
```

On peut utiliser les opérateurs **SOME**, **ANY**, ou **ALL** pour comparer la valeur d'une expression avec les valeurs d'un attribut d'une requête interne.
Les articles dont le prix est supérieur à tous les articles blancs :

```
SELECT art_nom
FROM articles
WHERE art_prix > ALL ( SELECT art_prix
FROM articles
WHERE art_couleur = 'blanc' )
```

Les articles dont le prix est supérieur à l'un des articles blancs :

```
SELECT art_nom
FROM articles
WHERE art_prix > ANY ( SELECT art_prix
FROM articles
WHERE art_couleur = 'blanc' )
```

Tous les articles mais seulement s'il en existe un de blanc :

```
SELECT art_nom
FROM articles
WHERE EXISTS ( SELECT art_num
FROM articles
WHERE art_couleur = 'blanc' )
```

⇨ **Auto jointure :**

Afficher les noms des personnes ayant la même note.

```
SELECT Etudiant.nom, Etudiant2.note
FROM Etudiant INNER JOIN Etudiant AS Etudiant2 ON
Etudiant.note = Etudiant2.note
WHERE Etudiant.n°<> Etudiant2.n° ;
```

✓ **Fonctions d'agrégat :**

```
COUNT   -- dénombrement
SUM
AVG     -- moyenne
VAR     -- variance
STDEV   -- écart-type
MIN
MAX
```

✓ **Fonctions intégrés :**

- **Fonctions de traitement de chaines :**

LEFT, RIGHT	Extraire des caractères à gauche ou à droite
UPPER,LOWER	Mettre en majuscules ou minuscules
LTRIM,RTRIM	Suppression des espaces à gauche ou à droite
SUBSTRING	Extraction d'une sous chaîne
REVERSE	Inversion d'une chaîne (miroir…)
LEN	Longueur d'une chaîne
ASCII	Valeur ascii d'un caractère
NCHAR	Renvoie le caractère Unicode fonction de la valeur donnée
REPLACE	Remplacement d'une occurrence de chaîne par une autre

- **Fonctions de manipulation de dates :**

DATEADD	Ajout d'un intervalle de temps à une date
DATEDIFF	Intervalle de temps entre deux dates
DATEPART	Extraction d'une partie de date
DATENAME	Chaîne représentant une partie de date
DAY,MONTH,YEAR	Renvoie d'une partie de date
GETDATE,GETUTCDATE	Date du système

- **Fonctions de conversion :**

Cast et Convert

Exemples :

```
SELECT CONVERT(DATETIME,GETDATE(),102) AS "Date au format
américain"

SELECT CONVERT(Decimal(10,3),sum((UnitPrice * 1- Discount) *
Quantity)) AS "TOTAL CA Net"
FROM [Order Details]

SELECT Upper(Substring(Nom,1,1)) + Substring(Nom,2,Len(Nom)-
1)
FROM PILOTE

SELECT REPLACE(VILLE,'Toulouse','Ville Rose')
FROM PILOTE
WHERE VILLE LIKE 'TOUL%'

SELECT DATEADD(DAY,3,DateNaissance) as "Date + 3 jours",
DateNaissance
```

```
FROM Pilote

SELECT NOM,DATEDIFF(DAY,DateNaissance,GETDATE()) as "Nombre
jours depuis Naissance"
FROM Pilote

SELECT DATEPART(MONTH,DateNaissance),
DATENAME(MONTH,DateNaissance)
FROM PILOTE
```

- **Fonctions mathématiques**

A savoir : ABS (valeur absolue), CEILING (partie entière +1), COS, EXP, FLOOR (partie entière), LOG (logarithme neperien), LOG10, PI, POWER, SIGN, SIN, SQRT, SQUARE et TAN.

✓ **Pour définir une contrainte :**

On peut définir les contraintes au moment de la création de la table ou par la suite.

```
CREATE TABLE [schema.]table
          (column datatype [DEFAULT expr]
            [column_constraint],
            …
            [table_constraint]);
```

⇨ **Contrainte au niveau colonne**

```
column [CONSTRAINT constraint_name] constraint_type,
```

⇨ **Contrainte au niveau table**

```
column,...
  [CONSTRAINT constraint_name] constraint_type
  (column, ...),
```

Les contraintes :

- **DEFAULT :** affecter un nom de contrainte à une valeur par défaut.

```
ALTER TABLE assures
ADD CONSTRAINT def_date_entree
DEFAULT GETDATE() FOR date_entree
```

- **NOT NULL et NULL :** La contrainte **NOT NULL** interdit la présence de valeurs **NULL** dans la colonne à laquelle elle s'applique. Par défaut, les colonnes qui ne sont pas associées à la contrainte **NOT NULL** peuvent contenir des valeurs **NULL**.

- **PRIMARY KEY** : crée une clé primaire pour la table

```
ALTER TABLE ouvrages
ADD CONSTRAINT pk_ouvrages
PRIMARY KEY (isbn, no_copie)
```

- **IDENTITY** : Permet d'incrémenter automatiquement la valeur des clés primaires par incrémentation.

```
[IdContact] [int] IDENTITY (1, 1)
```

- **UNIQUE** : UNIQUE exige que chaque valeur dans une colonne ou dans un ensemble de colonnes soit unique, c'est-à-dire qu'elle n'existe pas dans plusieurs lignes pour la colonne ou l'ensemble de colonnes spécifiés. Une table peut comprendre plusieurs contraintes UNIQUE.

```
ALTER TABLE clients
ADD CONSTRAINT un_nom_prenom
UNIQUE (clt_nom, clt_prenom)
```

- **FOREIGN KEY :** contrainte d'intégrité référentielle, désigne une colonne ou une combinaison de colonnes comme étant une clé étrangère et établit une relation avec une clé primaire ou une clé unique de la même table ou d'une table différente.

```
FOREIGN KEY (job_id) REFERENCES jobs(job_id)
```

- FOREIGN KEY définit une colonne de la table détail dans une contrainte de niveau table.
- REFERENCES identifie la table et la colonne dans la table maîtresse.

Exemple : si on veut désormais que la suppression d'un client entraîne automatiquement celle de ses commandes, il suffit pour cela de préciser une option lors de la définition de la contrainte clé étrangère dans la table commandes.

```
ALTER TABLE commandes
ADD CONSTRAINT fk_cmd_clt
FOREIGN KEY (cmd_clt) REFERENCES clients
ON DELETE CASCADE
ON UPDATE CASCADE
```

Remarques :

- ➢ de cette façon, la relation entre les deux tables devient non bloquante en suppression et en mise-à-jour;
- ➢ il n'y a pas ON INSERT CASCADE.

- **CHECK** : définit une condition que chaque ligne doit obligatoirement satisfaire. La condition peut utiliser les mêmes **constructions que les conditions d'une requête.**

✓ **Pour ajouter une contrainte dans une table existante :**

```
ALTER TABLE      table
ADD [CONSTRAINT constraint] type (column);
```

✓ **Pour supprimer une contrainte :**

```
ALTER TABLE    table
 DROP  PRIMARY KEY | UNIQUE (column) |
       CONSTRAINT   constraint  [CASCADE];
```

✓ **Pour activer ou désactiver une contrainte :**

```
{ CHECK | NOCHECK} CONSTRAINT
```

✓ **Les vues :**

Une vue est une requête SELECT à laquelle on donne un nom et dont on peut se servir comme s'il s'agissait d'une table.

⇨ Créer une vue :

```
CREATE VIEW VueCommandes       -- nom de la vue
([Nom du client], [Article commandé]) -- nom des colonnes
(plus parlants)
AS
SELECT b.clt_nom, c.art_nom
FROM commandes AS a
JOIN clients AS b ON a.cmd_clt = b.clt_num
JOIN articles AS c ON a.cmd_art = c.art_num
```

Puis on peut l'utiliser comme une table :

```
SELECT [Nom du client]
FROM VueCommandes
WHERE [Article commandé] = 'pinceau'
```

⇨ Modifier une vue :

```
ALTER VIEW VueCommandes
 ( ... ) -- les colonnes
 AS
 ... -- nouvelle requete SELECT
```

⇨ Supprimer une vue :

```
DROP VIEW VueCommandes
```

✓ **Les index :**

Un index stocké séparément de la table, est un ensemble de pointeurs désignant chaque numéro de lignes associées aux données d'une colonne.

⇨ Pour créer un index sur deux champs d'une table:

```
CREATE INDEX nom_index ON nom_table ASC|DESC
```

Exemple:

```
CREATE INDEX Index1
ON Etudiant (num, nom);
```

⇨ Pour créer un index en forçant l'unicité sur un champ :

```
CREATE UNIQUE CLUSTERED INDEX Index2
ON Bus (société);
```

⇨ Pour supprimer un index :

```
DROP INDEX nom_index
```

✓ **Cryptage :**

Lors de la création ou de la modification d'un déclencheur, une vue, une fonction ou une procédure stockée (bref, tout ce qui contient le code SQL destin´e aux utilisateurs), on peut préciser la clause WITH

ENCRYPTION qui permet de crypter le code de ces objets. Cela permet de protéger la propriété intellectuelle des développeurs sous SQL Server.

Exemples :

```
CREATE VIEW VueCommandes(Client, Article)
WITH ENCRYPTION
AS
...
ALTER PROC InfoDuClient
(@numero INT)
WITH ENCRYPTION
AS
...
```

✓ **Connexions :**

⇨ Création :

IL existe deux façons d'ajouter un nouveau compte de connexion :

- on peut le créer de toute pièce :

```
sp_addlogin
'Paul', -- le login
'luaP', -- le mot de passe
'Northwind' -- la base par defaut
```

- ou bien hériter d'une connexion Windows :
-

```
sp_grantlogin 'STID/Henri'
-- STID etant le nom du domaine

sp_defaultdb 'STID/Henri', 'Northwind'
```

- Il reste ensuite à lui donner accès au serveur :

```
sp_grantdbaccess 'Paul'
```

- On dispose évidemment des procédures :

```
sp_revokedbaccess 'Paul'
sp_droplogin 'Paul'
```

✓ **Rôles :**

Il existe 8 rôles sur serveur dans SQL Server dont :

nom du rôle	droits de ses membres
sysadmin	tous les droits sur le système et toutes les base
securityadmin	gestion des accès à SQL Server
dbcreator	création de bases de données

Pour ajouter et radier un utilisateur à l'un de ces rôles :

```
sp_addsrvrolemember 'Paul', 'dbcreator'
sp_dropsrvrolemember 'Paul', 'dbcreator'
```

Un même utilisateur peut cumuler plusieurs rôles.
Dans chaque base on dispose des rôles suivants :

nom du rôle	droits de ses membres
db_owner	tous les droits sur les objets de la base
db_accessadmin	ajout d'utilisateurs et de rôles
db_datareader	lire le contenu des tables
db_datawriter	insertion, suppression et modification sur toutes les tables
db_ddladmin	création, modification, suppression d'objet
db_securityadmin	gestion des rôles et des autorisations
db_public	à définir

Tous les utilisateurs appartiennent au rôle public et peuvent appartenir à d'autres rôles.
Pour ajouter un rôle et un utilisateur à ce rôle :

```
sp_addrole 'ServiceInformatique'

sp_addrolemember 'ServiceInformatique', 'Henri'

sp_droprolemember 'ServiceMarketing', 'Paul'
```

```
sp_droprole 'ServiceMarketing'
-- possible uniquement s'il ne reste plus aucun membre dans
ce rôle
```

- ✓ **Droits :**
  - ⇨ Sur les instructions :

Exemple : pour autoriser les utilisateurs Paul et Henri à créer des tables et des déclencheurs.

```
GRANT CREATE TABLE, CREATE TRIGGER
TO Paul, Henri
```

Remarque :

Paul et Henri doivent déjà posséder un compte utilisateur sur SQL Server.
Autre exemple : pour empêcher Paul de créer des vues :

```
DENY CREATE VIEW
TO Paul
```

Dernier exemple : pour lever les autorisations et les empêchements de Paul

```
REVOKE CREATE VIEW, CREATE TABLE
TO Paul
```

Remarques :

- REVOKE annule le dernier GRANT ou DENY correspondant ;
- après un REVOKE, SQL Server s'en remet aux autorisations par défaut du rôle dont Paul est membre ;
- on peut utiliser le mot-clé ALL pour désigner toutes les instructions.
  - ⇨ Sur les objets :

Dans une base de données, pour chaque table, chaque colonne et chaque instruction on peut préciser les autorisations.
Exemple : pour autoriser la sélection sur la table clients.

```
GRANT SELECT ON clients
 TO Paul
```

Autre exemple :
Pour empêcher les autres instructions

```
DENY INSERT, UPDATE, DELETE ON clients
TO Paul
```

Dernier exemple : pour autoriser la modification mais seulement du nom de client.

```
GRANT UPDATE(clt_nom) ON clients
TO Paul
```

Remarques :

- en général on n'a pas besoin de descendre jusqu'à la colonne, il est préférable de créer une vue et de donner les droits sur cette vue ;
- (important) il vaut mieux utiliser ALTER ... car les autorisations sur l'objet concerné sont conservées, contrairement à DROP ... suivi de CREATE ...