Numérique et sciences informatiques
Classe de terminale

III Bases de données
2. Le langage SQL

- Qu'est-ce que SQL : un langage d'interrogation structuré *Structured Query Language* ?
- Les ordres SQL se décomposent en :
  * requêtes
  * mises à jour
- SQL n'est pas sensible à la casse.

I Création du schéma des relations et mises à jour des relations

I.1 Créer ou supprimer une relation

- **Créer une relation** :

```
CREATE TABLE nom_relation(nom_attr1 dom1,
                          nom_attr2 dom2, ...) ;
```

- Pour **définir la clé primaire** de la relation :

  * on place l'indication **PRIMARY KEY** après la description d'un attribut

  * ou bien, si la clé primaire est formée de plusieurs attributs, on place à la fin de la relation, après la description des attributs :

```
PRIMARY KEY(nom_attr1, nom_attr2, …)
```

Remarque : en SQL il n'est pas obligatoire de définir une clé primaire lors de la création d'une table. En principe donc une table définie en SQL peut contenir des doublons.

- Pour **définir les clés étrangères**, après la description d'un attribut on ajoute :

```
REFERENCES nom_relation(nom_attr)
```

Remarque : La plupart des SGBR ne supportent pas les clés étrangères composites. Cela rend donc peu utile la possibilité de définir une clé primaire composite.

- Pour **indiquer l'unicité** des valeurs d'un attribut (sans que cet attribut soit une clé primaire) : on ajoute le mot-clé **UNIQUE** après la description de l'attribut.
- Pour **indiquer la non-nullité** de la valeur d'un attribut : on ajoute le mot-clé **NOT NULL** après la description de l'attribut.

Remarque : les valeurs d'une clé primaire sont obligatoirement uniques et non nulles.

- **Ajouter d'autres contraintes utilisateurs** : après la description des attributs de la relation on ajoute **CHECK** suivi d'une condition (une expression qui donne un booléen).
- Pour supprimer une relation :

```
DROP TABLE nom_relation ;
```

I.2 Les domaines de valeur en SQL

• La valeur nulle : `NULL`.

• Les nombres :

`SMALLINT`	sur 16 bits signés
`INT` ou `INTEGER`	sur 32 bits signés
`BIGINT`	sur 64 bits signés

`DECIMAL (n, p)`	nombre décimal signé avec `n` chiffres dont `p` chiffres après la virgule
	---
`REAL`	flottant 32 bits
`DOUBLE PRCESION`	flottant 64 bits

• Les textes :

`CHAR(n)`	chaîne de caractère comportant exactement `n` caractères
`VARCHAR(n)`	chaîne de caractère comportant au plus `n` caractères
`TEXT`	chaîne de caractères de longueur quelconque

• Booléens :

`BOOLEAN`

• Dates et heures

`DATE`
`TIME`
`TIMESTAMP`

I.3 Insertion d'un nouvel enregistrement dans une table

• Pour insérer une nouvelle entité (ou enregistrement) dans la relation, on écrit ;

```
INSERT INTO nom_relation VALUES (val1, va2, ...)[, (...)] ;
```

• Chaque valeur dans le tuple doit correspondre à l' attribut défini <u>à la même place</u> dans l'ordre de création de la relation `CREATE TABLE` .... Cette valeur donc être cohérente avec le domaine de valeur défini dans cet ordre. Le respect des contraintes d'intégrité est effectué à la création de l'enregistrement et l'insertion du nouvel enregistrement est donc rejeté en cas de non-respect de ces contraintes.

<u>II Les requêtes</u>

II.1 Les sélections simples

```
SELECT nom_attr1, nom_attr2, ...
FROM nom_relation
WHERE condition ;
```

ou bien

```
SELECT *
FROM nom_relation WHERE condition ;
```

Remarque : une telle requête génère une table sur laquelle on peut appliquer d'autres requêtes.

II.2 Écrire une condition

• On utilise :

* les opérateurs booléens habituels : **OR**, **AND** et **NOT**

* les opérateurs sur les nombres habituels: :**+**, **-**, *****, **/** et **%**.

* les opérateurs de comparaison habituels : **=**, **<>** ou **!=**, **<**, **<=**, **>**, **>=**.

* un opérateur spécial de comparaison pour des texte : **LIKE** à utiliser avec le symbole `%` qui désigne une chaîne de caractère quelconque.

II.3 Autres paramétrages de la sélection

• Pour obtenir une réponse **sans doublon**, on écrit

```
SELECT DISTINCT nom_attr1, nom_attr2, ...
FROM ... WHERE ...
```

• Pour **trier les résultats** par ordre croissant ou décroissant des valeurs de l'attribut `attribA` on écrit

```
SELECT [DISTINCT]...
FROM ... WHERE ...
ORDER BY attribA [ASC]
```

ou bien pour trier par ordre décroissant :

```
SELECT [DISTINCT]...
FROM ... WHERE ...
ORDER BY attribA DESC
```

II.3 Les sélections avec agrégation

• La syntaxe :

```
SELECT nom_fonction(nom_attr)
FROM nom_relation
WHERE condition ;
```

• Quelques fonctions classiques :

* **COUNT(*)**

* **SUM(**`nom_attr`**)** (où `nom_attr` correspond à des nombres) : affiche la somme des valeurs de l'attribut `nom_attr` pour la sélection ;

* **AVG(**`nom_attr`**)** (où `nom_attr` correspond à des nombres) : affiche la moyenne arithmétique des valeurs de l'attribut `nom_attr` pour la sélection ;

* **MIN(**`nom_attr`**)** : affiche la valeur minimum de l'attribut `nom_attr` pour la sélection ;

* **MAX(**`nom_attr`**)** : affiche la valeur maximum de l'attribut `nom_attr` pour la sélection ;

II.4 Faire des jointures

• La syntaxe :

```
SELECT nom_fonction(nom_attr)
FROM rel1
JOIN rel2
ON égalité
[WHERE condition] ;
```

`égalité` est une égalité entre un attribut de la relation `rel1` et un attribut de la relation `rel2`.

• Un exemple :

```
SELECT * FROM emprunt
JOIN livre ON emprunt.id_ex = livre.id
WHERE emprunt.retour < '2025-01-23' ;
```

Remarque : Dans le contexte d'une jointure, on désigne un attribut d'une table en le préfixant par le nom de la relation auquel il appartient.

III Les ordres de mise à jour

III.1 Modifier un enregistrement

• La syntaxe :

```
UPDATE nom_relation SET atr1 = val1[, SET atr2 = val2, …]
WHERE condition ;
```

III.2 Supprimer un enregistrement

• La syntaxe :

```
DELETE FROM nom_relation
WHERE condition ;
```

Remarque : Attention à respecter les contraintes d'intégrité lors des suppressions d'enregistrement.

III.3 Copier une table

• Récupérer une sélection dans une table temporaire :

```
SELECT ... INTO nouveau_nom FROM ... WHERE … ;
```

• Pour créer une relation `relationB` ayant le même schéma qu'une relation existante `relationA` y compris les contraintes d'intégrité (clé primaire, clés étrangères etc.) :

```
CREATE TABLE relationB (LIKE relationA INCLUDING ALL) ;
```

• Pour recopier toutes les données de la relation `relationA` dans la relation `relationB`, on écrit :

```
INSERT INTO relationB (SELECT * FROM relationA) ;
```