Chapitre 2

Le modèle Relationnel

Le langage SQL (Structured Query Language)

SQL

SQL est le langage informatique standard pour la communication avec les SGBD relationnelles. Il peut être logiquement vu comme étant composé de 3 sous langages: un Langage de Manipulation de Données (LMD), un Langage de Description de Données (LDD) et un Langage de Contrôle de Données (LCD).

Langage de Description de Données (LDD)

- Instructions: CREATE, DROP, ALTER TABLE

Langage de Manipulation de Données (LMD)

- Instructions: SELECT, INSERT, DELETE, UPDATE

Chapitre 2

Modèle Relationnel
Le langage SQL
(Structured Query Language)

Langage de Description de Données LDD

La description des données avec SQL (1)

- **Création de la base de données create database Tp1;**
- **Création de schéma de relations**

La création de schéma de relation consiste à nommer la relation et à en définir les attributs;

- CREATE TABLE nom de relation (nom_attribut1 type_attribut1 [Not NULL| NULL] [nom_attribut2 type_attribut2 [Not NULL| NULL]…)
- type_attribut peut être par exemple: Int: entier signé représenté sur 4 octets; Float: virgule flottante;
- Char (longueur): chaîne de longueur fixe.
- **NOT NULL:** spécifie qu'une colonne ne doit pas contenir de valeurs **NULL** (ou indéfinies, inconnues).
- ***Exemple:*** **CREATE TABLE ETUDIANTS** (NUMERO int NOT NULL, NOM CHAR(10), MOYENNE float)

Contraintes sur les colonnes

PRIMARY KEY

Clé primaire de la table. Pour désigner plus d'une colonne, on l'utilise comme une clause PRIMARY KEY(col1, col2, …) à côté des définitions de colonnes

ماهو الحقل المفتاح ؟

كيف يستغل المفتاح عند ادخال المعلومات ؟

كيف يستغل المفتاح في البحث على المعلومات ؟

NOT NULL

Doit avoir une valeur pour chaque enregistrement

UNIQUE

Chaque enregistrement doit avoir une valeur différente

CHECK *condition*

La valeur doit respecter une condition donnée

Exemple de création de table

```
CREATE TABLE Etudiant
   (
   Matricule INT PRIMARY KEY,
   Nom VARCHAR(30) NOT NULL,
    Prénom VARCHAR(30) NOT NULL,
    DateNais  date,
    Résidence VARCHAR(30),
   Spécialité VARCHAR(10) CHECK (type IN ('Tcommun', 'management', 'comptabilité','autres')
   );
```

Clé etrangère

Une clé étrangère identifie une colonne ou un ensemble de colonnes d'une table comme référençant une colonne ou un ensemble de colonnes d'une autre table (la table référencée). Les colonnes de la table référencée doivent faire partie d'une contrainte de clé primaire ou d'une contrainte d'unicité. La contrainte de clé étrangère garantit que les valeurs de chaque ligne de la table référençant existent dans la table référencée : ainsi une ligne de la table référençant ne peut pas contenir un ensemble de valeurs qui n'existe pas dans la table référencée.

Exemple de création de table

Exemple

categorie (id_categorie, nom_categorie)

objet (id_objet, # id_categorie, nom_objet)

Mysql

```
Create table categorie (id_categorie int primary
  key, nom_categorie varchar(30));

Create table objet (id_objet int primary key,
  id_categorie int, nom_objet var char(30),
  foreign key (id_categorie) references categorie
  (id_categorie));
```

La description des données avec SQL (2)

- **Suppression d'une table : DROP TABLE**
- **DROP TABLE** *<**nom de table**>* ; supprime le contenu de la relation ainsi que sa définition, c. a d. son schéma.

- **Modification de schémas de relations:**
- La commande **ALTER TABLE** permet de modifier le schéma d'une relation en y ajoutant de attributs ou des contraintes, en modifiant la définition d'un attribut, …la variété des modifications dépend des SGBD. Il faut cependant avoir à l'esprit que la modification de schéma de relations pendant la durée de vie d'une base peut avoir des incidences sur les données et sur les programmes existants.

Modification d'une table

Le contenu d'une table peut être modifié par l'instruction ALTER TABLE

ALTER TABLE *nom_table*
***modification1*,**
modification 2…;

Les modifications portent sur des ajouts, suppressions ou remplacements de colonnes avec pour syntaxe :

ADD (*nom_col type_col* [*contrainte*])

DROP *nom_col*

MODIFY *nom_col type_col* [*contrainte*]

Exemple de modification de table

```
ALTER TABLE ETUDIANT
    ADD (Université VARCHAR(30)),
    MODIFY Nom VARCHAR(20) NOT NULL,
    DROP DateNais;
```

Renommage d'une table

Une *table* est renommée par l'instruction

RENAME TABLE

RENAME TABLE *ancien_nom* TO *nouveau_nom*;

Modèle Relationnel
Le langage SQL
(Structured Query Language)

Langage de Manipulation
Données LMD

Instruction SQL de mise à jour

- **INSERT : ajout de tuples**

L'opération d'insertion permet d'ajouter un ou plusieurs tuples à une relation

- **UPDATE: modification de tuples**

UPDATE modifie des valeurs de colonnes dans une ou plusieurs lignes

- **Delete: Suppression de tuples**

La suppression peut concerner tous les tuples d'1 relation ou un sous-ensemble de tuple qui vérifie une condition de selection; il fait noter que delete opère sur le contenu d'une relation, c'est-à-dire que le schéma d'une relation persiste après suppression de tous les tuples de la relation.

Insertion d'enregistrement (TUPLE)

Un *enregistrement* (*tuple* en algèbre relationnelle) s'insère dans une table par l'instruction INSERT INTO

INSERT INTO *nom_table*[*(nom_col1, nom_col2, …)*]
VALUES (*val1*, *val2*, …)
[(*val3*, *val4*, …)];

Exemple :

INSERT INTO ETUDIANT
VALUES (1, ' Hamidi', 'Omar', 19/08/1989, 'alger', 'autre'),
(2, ' Achouri', 'Karim', 12/08/1988, 'Ourgla', 'Tcommun');

Modification d'enregistrement

Un ou plusieurs enregistrements se modifient par l'instruction UPDATE

UPDATE *nom_table*
SET *nom_col1* = *expr1* [, *nom_col2* = *expr2*]...
WHERE *conditions*;

conditions exprime un ensemble de condition pour sélectionner les enregistrements à modifier et les clauses SET indiquent les modifications à effectuer.

Exemple :

```
UPDATE Client
        SET spécialité = 'comptabilité'
        WHERE nom = ' Hamidi ';
```

Suppression d'enregistrement

Un ou plusieurs enregistrements se suppriment par l'instruction DELETE

DELETE FROM *nom_table*
WHERE *conditions*;

conditions exprime un ensemble de condition pour sélectionner (à la manière de l'opérateur relationnel de sélection) les enregistrements à effacer.

Exemple :

```
DELETE FROM Client
        WHERE nom = ' Messoudi';
```

Requête SELECT

Une requête d'interrogation de la BdD s'écrit à l'aide de l'instruction SELECT ….Consultation

SELECT [DISTINCT] *table1.attr1*
[, *table2.attr2*]… | *expr*
FROM *table1* [, *table2*]…
[WHERE *condition*]
[ORDER BY *expr* [DESC]]
[GROUP BY *expr*]
[HAVING *expr*]

Select (1) : Projection

- ***Exemple*** : soit la relation **Produit (<u>code</u>, desig, PU)**, et soit l'expression en algèbre relationnelle suivante: Project $_{Desig, PU}$ (produit), elle se traduit en SQL comme suit:
- **SELECT** desig, PU
- **FROM** *Produit*

- RQ: SQL n'élimine pas les tuples en double à moins que ceci soit explicitement demandé par un mot clé du langage qui est : **DISTINCT** (ou parfois **UNIQUE**) → **SELECT UNIQUE**

Select (2): Sélection

SELECT *

FROM *nom de relation*

WHERE *condition ;*

- " * " dénote toutes les colonnes de la relation.
- ***Exemples:***
- **1)** SELECT $_{(PU<= 250)}$ (produit) → **SELECT** * **FROM** *Produit* **WHERE (**PU<= 250)
- 2) "Quel sont les produits dont le nom commence par 'P' et dont le prix unitaire est inférieur ou égale à 300?"
- **SELECT** * **FROM** *Produit*
- **WHERE (**Desig **Like** "P%") **and** (PU<= 300)

Select (3): Composition de la sélection et de la projection

SELECT *liste d'attributs*

FROM *nom de relation*

WHERE *condition*

- ***Exemple:*** **"**liste contenant le code et la désignation des produits dont leur prix unitaire est supérieur ou égal 500"
  → **SELECT** *code, desig*

  FROM *Produit*

  WHERE (PU>= 500)

Select (4) : Produit cartésien

- 2 eme Exemple
- Soient les deux relations suivantes :
- **EMPLOYES (Num_Emp, Nom_Emp, Fonction, Salaire)**
- **DEPARTEMENTS (Num_Dep, Ville)**

- Le produit cartésien entre deux relations s'exprime comme une jointure sans condition (qualification).
- **SELECT** * **FROM** *liste de noms de relations*
- ***Exemple:*** le produit cartésien des relations EMPLOYES et DEPARTEMENTS s'obtient à l'aide de la requête suivante : **SELECT** * **FROM** EMPLOYES , DEPARTEMENTS

Select (5) : Jointure

- **SELECT** * **FROM** *liste de noms de relations* **WHERE** expression de jointure
- ***Exemples:*** *"Quels sont les numéros et les noms des employés qui travaillent à 'BATNA' ?*
- **SELECT** Num_Emp , Nom_Emp
- **FROM** EMPLOYES , DEPARTEMENTS
- **WHERE (**EMPLOYES•Num_Dept=DEPARTEMENTS•Num_Dept) **AND (**Ville = 'BATNA')

Expression de l'union, de l'intersection et de la différence

- SQL offre des opérateurs ensemblistes qui permettent d'exprimer l'union (UNION), l'intersection (INTERSECT) et la différence (MINUS) à l'aide de requêtes:

< Clause Select > < opérateur ensembliste> <Clause Select>

- Il faut veiller à ce que les opérandes gauche et droit de l'opérateur soit des relations de même schéma ou de schémas compatibles.

Select (6): Les fonctions de groupes

Aperçu général

- SQL offre aussi un certain nombre de fonctions dites fonctions de groupes qui peuvent être utilisées dans l'expression d'une requête. Les plus importantes sont :
- **AVG** : permet de calculer une **MOYENNE**
- **SUM** : permet de calculer une **SOMME**
- **COUNT** : permet de compter des tuples
- **MAX** : permet de calculer un **maximum**
- **MIN** : permet de calculer un **minimum**
- **GROUP BY** : permet de créer des sous-ensembles de tuples
- **HAVING** : permet de tester si une condition est vérifiée par un groupe de tuples

Select (7): Tri

- **Tri du résultat d'une requête : la clause ORDER BY**
- La clause ORDER BY permet de trier le résultat retourné par une requête. L'ordre peut être ascendant (ASC) ou descendant (DESC) et sera fonction du type de ou des attributs qui seront spécifiés comme arguments de cette clause. Les exemples suivants permettent de donner une idée des possibilités de cette clause.
- **Q:** Liste des employés du *département 10 par ordre décroissant de leur salaire?*
- **SELECT** Num_Dept , Nom_Emp, Salaire **FROM** EMPLOYES
- **WHERE** Num_Dept = 10 **ORDER BY** Salaire **DESC**

Modèle Relationnel

Le langage SQL (Structured Query Language)

Exemple de requêtes

Exemples de requêtes

- **EMPLOYES (Num_Emp, Nom_Emp, Fonction, Salaire)**
- **DEPARTEMENTS (Num_Dep, Ville)**

- **Q1:«***Quel est le nom, la fonction et le salaire de(s) l'employé(s) ayant le salaire le plus élevé»*

SELECT Nom_Emp , Fonction , Salaire **FROM** EMPLOYES

WHERE Salaire = **(SELECT MAX** (Salaire) **FROM** EMPLOYES**)**

La sous requête est évalué en premier et permet de calculer le salaire maximum des employés. La requête principale va parcourir les lignes de la table EMPLOYES et comparer le salaire de chaque employé avec le résultat de la sous requête. S'il y a égalité, le nom, la fonction et le salaire de cet employé figureront dans le résultat final.

Exemples de requêtes

- **Q2: «** *Quel est le nombre de fonctions différentes exercées dans le département 30 ?»*

SELECT COUNT (**DISTINCT** Fonction) **FROM** EMPLOYES
WHERE Num_Dept = 30

Le mot clé **DISTINCT** permet de ne compter une même valeur qu'une seule fois. Le nombre retourné sera bien le nombre de valeurs différentes de l'attribut Fonction.

Exemples de requêtes

Manipulation de groupes : la clause GROUP BY

- La clause GROUP BY permet de partitionner les tuples d'une relation de façon à former des groupes de tuples. Chaque groupe de tuples est caractérisé par le fait que les tuples qu'il contient possèdent les mêmes caractéristiques.
- La manipulation des groupes implique donc que les requêtes peuvent comporter des clauses WHERE , GROUP BY, des fonctions de groupes (AVG, SUM, MAX, MIN) et HAVING. La clause WHERE est utilisée pour sélectionner dans une table les lignes (tuples) qui satisfont une condition.
- Elle ne s'applique donc pas aux groupes. La clause HAVING est utilisée pour sélectionner dans une table les groupes de lignes (tuples) qui satisfont une condition. Elle ne s'applique donc pas aux lignes et s'utilise avec une clause GROUP BY.

Exemples de requêtes

- Dans une requête manipulant des fonctions de groupe, il existe un ordre logique d'exécution de la requête et qui est :
- **1-** La clause **WHERE** est appliquée en premier pour qualifier les lignes
- **2-** Les groupes de lignes sont formés (**GROUP BY**)
- **3-** Les fonctions de groupes sont appliquées (**AVG, MIN, MAX**,..)
- **4-** Enfin la clause **HAVING** est appliquée pour choisir les groupes répondant au critère spécifié dans cette clause.

Exemples de requêtes

- **Q3: «** *Quelle est la moyenne des salaires par département? »*

SELECT Num_Dept , **AVG (**Salaire**)**

FROM EMPLOYES

GROUP BY Num_Dept

- **Q4: «** *Quels est pour chaque département le salaire annuel (12 mois) moyen de tous ses employés sauf ceux ayant une fonction de* 'DIRECTEUR' *ou* 'PRESIDENT'*?*

SELECT Num_Dept, **AVG (**Salaire**) * 12 FROM** EMPLOYES

WHERE Fonction **NOT IN** ('DIRECTEUR', 'PRESIDENT')

GROUP BY Num_Dept

Exemples de requêtes

- **Q5:** *Quels sont les numéros des départements ayant plus de 10 employés ?*

SELECT Num_Dept **FROM** EMPLOYES

GROUP BY Num_Dept **HAVING COUNT(*) >** 10

Le **COUNT(*)** compte les tuples des sous-ensembles créés par **GROUP BY**. Les tuples de chaque sous-ensemble crée ont la même valeur de l'attribut qui est utilisé dans le **GROUP BY** (c.a.d. ici Num_Dept). Le résultat de cette requête sera une relation partitionnée en groupes de tuples ayant la même valeur de l'attribut Num_Dept. A chacun de ces groupes sera appliquée la qualification de la clause **HAVING** c'est a dire **COUNT(*) > 10.** Si le groupe satisfait la condition (contient plus de 10 tuples), il sera inclut dans le résultat par le biais de son attribut Num_Dept jouant le rôle de dénominateur commun pour les tuples du groupe.

Exemples de requêtes

- **Q6:«** *Quelles sont les différentes fonctions exercées dans l'ensemble des départements et la moyenne des salaires par fonction ?»*

SELECT Fonction, **AVG (**Salaire**) AS** Moyenne_salaire **FROM** EMPLOYES **GROUP BY** Fonction

Remarque: l'utilisation de *AS* pour donner un nom à la colonne créée à l'aide de la fonction *AVG*

- **Q7: «** *Quelles sont les différentes fonctions exercées dans l'ensemble des départements dont la moyenne des salaires par fonction est supérieure à10.000 ?*

SELECT Fonction , **AVG (**Salaire**) FROM** EMPLOYES

GROUP BY Fonction **HAVING AVG (**Salaire**)** > 10.000

Exemples de requêtes

- **Tri du résultat d'une requête : la clause ORDER BY**

La clause ORDER BY permet de trier le résultat retourné par une requête. L'ordre peut être ascendant (ASC) ou descendant (DESC) et sera fonction du type de ou des attributs qui seront spécifiés comme arguments de cette clause. Les exemples suivants permettent de donner une idée des possibilités de cette clause.

- **Q8:** Liste des employés du *département 10 par ordre décroissant de leur salaire?*

SELECT Num_Dept , Nom_Emp, Salaire **FROM** EMPLOYES
WHERE Num_Dept = 10 **ORDER BY** Salaire **DESC**

Exemples de requêtes

Il est aussi possible de trier le résultat retourné par une requête en fonction de plusieurs attributs.

- **Q9:** Liste des employés triés selon un ordre alphabétique de leur fonction et à l'intérieur de chaque fonction les trier selon un salaire décroissant.

SELECT Nom_Emp, Fonction, Salaire **FROM** EMPLOYES
ORDER BY Fonction **ASC**, Salaire **DESC**