Chapitre 3.

SQL

SQL

- SQL (Structured Query Language) est l'une des raisons principales du succès du modèle relationnel
- SQL est devenu un standard supporté par tous les SGBD relationnels
- Standardisé par un effort conjoint de l'ANSI et de l'ISO en 1986 (SQL1). Révisé en 1992 (SQL2), puis en 1999 (SQL3).
- D'autres révisions ont suivi en 2003 et en 2006 pour introduire des aspects liés à XML et aux objets notamment.
- SQL est à la fois un **LDD** et un **LMD** ; il permet la définition, l'interrogation et la modification de la BD. Il permet également de définir des **vues** et des **autorisations**.

Définition et types de données

CREATE SCHEMA ENTREPRISE **AUTHORIZATION** 'SMI5'

- Un schéma est identifié par un **nom** et un **propriétaire** (AUTHORIZATION)
- Il est composé de : tables, vues, contraintes, domaines, autorisations, etc.

Définition et types de données

CREATE TABLE ENTREPRISE.EMPLOYE…

Ou simplement

CREATE TABLE EMPLOYE…

- La commande CREATE TABLE permet de définir une nouvelle relation en lui affectant un nom et en spécifiant ses attributs.
- On spécifie le nom d'attribut, son type de donnée (domaine), ses contraintes (NOT NULL par exemple)
- On peut également définir les contraintes de clé et d'intégrité référentielle

Définition et types de données : Exemple

```
CREATE TABLE EMPLOYE
(Matricule CHAR(9) NOT NULL,
Nom VARCHAR(15) NOT NULL,
Prenom VARCHAR(15) NOT NULL,
DNaiss DATE,
Adresse VARCHAR(30),
Sexe CHAR,
Salaire DECIMAL(10,2),
Super_Mat CHAR(9),
Dnum INT NOT NULL,
PRIMARY KEY (Matricule),
FOREIGN KEY (Super_Mat) REFERENCES EMPLOYE(Matricule),
FOREIGN KEY (Dnum) REFERENCES DEPARTEMENT(Dnumero) );
```

Domaines et types de données

- **Numérique** : Entier (INT et SMALLINT), réel (FLOAT et DOUBLE PRECISION), DECIMAL*(i,j)* où *i* est la précision (le nombre de chiffres) et où *j* est le nombre de chiffres après la virgule.
- **Chaîne de caractères** : de longueur fixe (CHAR(n) où n est le nombre de caractères) ou variable (VARCHAR(n) où n est le nombre maximal de caractères)
- **Chaîne de bits** : BIT(n)
- **Booléen** : Vrai ou Faux (En SQL, il peut être aussi INCONNU)
- **DATE** : de 10 positions AAAA-MM-JJ

Définition des contraintes : Attributs

- **NOT NULL**: Cette contrainte spécifie qu'un attribut ne peut être NULL. Elle est implicite pour les attributs de la clé primaire.
- **DEFAULT** : Valeur prise à défaut d'une valeur donnée explicitement
  - Syntaxe : **DEFAULT** <value>
- **CHECK**: restriction des valeurs du domaine ou de l'attribut
  - Note DECIMAL(4,2) NOT NULL **CHECK** (Note >=0 AND Note <=20);
  - **CHECK** peut être également spécifiée lors de la définition d'un domaine
  - **CREATE DOMAIN** D_Note **AS** DECIMAL(4,2) **CHECK** (D_Note >=0 AND D_Note <=20);

Définition des contraintes : Clés et intégrité référentielle

- **PRIMARY KEY** : Spécifier qu'un ou plusieurs attributs forment la clé **primaire** de la relation
  - `MATRICULE CHAR(9) PRIMARY KEY;`
- **UNIQUE** : Spécifier qu'un ou plusieurs attributs forment une clé **secondaire** de la relation
  - `DNom VARCHAR(20) UNIQUE;`
- **FOREIGN KEY** : Spécifie une clé étrangère

Définition des contraintes : Exemple

```
CREATE TABLE EMPLOYE
(Matricule CHAR(9) NOT NULL,
Nom VARCHAR(15) NOT NULL,
Prenom VARCHAR(15) NOT NULL,
DNaiss DATE,
Adresse VARCHAR(30),
Sexe CHAR,
Salaire DECIMAL(10,2),
Super_Mat CHAR(9),
Dnum INT NOT NULL DEFAULT 1,
PRIMARY KEY (Matricule),
FOREIGN KEY (Super_Mat) REFERENCES EMPLOYE(Matricule) ON
DELETE SET NULL ON UPDATE CASCADE,
FOREIGN KEY (Dnum) REFERENCES DEPARTEMENT(Dnumero)
ON DELETE SET DEFAULT ON UPDATE CASCADE);
```

Définition des contraintes : Tuples

- On peut également définir des contraintes CHECK qui portent sur des tuples. Ces contraintes sont insérées à la fin de la clause CREATE TABLE.
- **Exemple** :
  - pour la table DEPARTEMENT

CHECK (DateDebutChef >= DateCreation)

Requêtes de recherche SQL : Syntaxe

SELECT <liste_attributs>

FROM <liste_tables>

WHERE <condition>;

Où

- <liste_attributs> est la liste des noms d'attributs dont les valeurs seront extraites par la requête
- <liste_table> est la liste des noms de relations nécessaires pour l'exécution de la requête
- <conditon> est une expression booléenne qui identifie les tuples à retourner par la requête

- Les opérateurs de comparaison en SQL : =, <, <=, >, >= et <>

Remarque :

• Contrairement au modèle relationnel formel ; les tables SQL ne sont pas des ensembles ; l'existence de lignes identiques (mêmes valeurs pour tous les attributs) est autorisée. Toutefois, les tables SQL pour lesquelles, une clé a été définie sont des ensembles (sans doublons).

Requêtes SQL : Exemple

```
SELECT Adresse, DNaiss
FROM EMPLOYE
WHERE Prenom = 'Ahmed'  AND Nom = 'ALAOUI';
```

- La requête SQL spécifie la liste des attributs (Adresse, et DNaiss) dont les valeurs seront retournés (**Projection**)
- La table source de données (EMPLOYE)
- La condition sur les tuples particuliers à retourner (**Selection**)

Diagramme schéma BD « ENTREPRISE »

EMPLOYE

Matricule	Nom	Prenom	Adresse	DNaiss	Sexe	Salaire	SuperMat	DNum

DEPARTEMENT

DNumero	DNom	ChefMat	Date Creation	DateDebutChef

DEPTLOCALITE

DLNumero	DLocalite

PROJET

PNum	PNom	PLocalite	DNum

TRAVAILLE_SUR

EMat	PNum	Heures

ENFANT

EMat	NomEnfant	Sexe	DateNaiss

Requêtes SQL : Exemples

1. Liste des noms et adresses des employés qui travaillent pour le département informatique

```
SELECT Nom, Adresse
FROM EMPLOYE, DEPARTEMENT
WHERE DNom = 'Informatique'  AND DNum = DNumero;
```

Requêtes SQL : Exemples

2. Liste des numéros projets localisés à Tétouan, le nom du département en charge du projet, le nom et l'adresse du chef de ce département

```
SELECT PNum, Dnom, Nom, Adresse
FROM EMPLOYE, DEPARTEMENT, PROJET
WHERE ChefMat = Matricule AND DNum = DNumero AND Plocalite = 'Tetouan';
```

Requêtes SQL : Attributs ambigus, alias

1. Liste des noms et adresses des employés qui travaillent pour le département informatique (On suppose que l'attribut numéro département est nommé DNumero dans les tables EMPLOYE et DEPARTEMENT. Idem pour l'attribut Nom)

```
SELECT EMPLOYE.Nom, Adresse
FROM EMPLOYE, DEPARTEMENT
WHERE DEPARTEMENT.Nom = 'Informatique'  AND EMPLOYE.DNumero =
DEPARTEMENT.DNumero;
```

Requêtes SQL : Attributs ambigus, alias

1. Pour chaque employé, extraire le nom et le prénom puis le nom et le prénom de son supérieur hiérarchique (Ambiguïté due au fait qu'on se réfère à deux reprises à la même table)

```
SELECT E.Prenom, E.Nom, S.Prenom, S.Nom
FROM EMPLOYE AS E, EMPLOYE AS S
WHERE E.SuperMat = S.Matricule;
```

- E et S sont des **alias** pour la relation EMPLOYE
- On peut renommer la table ainsi que ses attributs dans la clause FROM :

```
EMPLOYE AS E(Pr, Nm, Ad, DN, Sx, Sl, SMat, DN)
```

Requêtes SQL : sans condition (WHERE) et astérisque (*)

```
SELECT Prenom
FROM EMPLOYE
```

- liste tous les prénoms de la table employé (avec des doublons)

```
SELECT Matricule, DNom
FROM EMPLOYE, DEPARTEMENT
```

- liste toutes les combinaisons possibles EMPLOYE et DEPARTEMENT

```
SELECT *
FROM EMPLOYE
WHERE Dnum = 2;
```

- retourne tous les attributs de la table EMPLOYE du département N° 2

Requêtes SQL : Tables vs Ensembles

- SQL manipule des tables avec des doublons
- L'élimination des doublons est une opération coûteuse ; elle nécessiterait un tri préalable des résultats
- il est également possible que l'utilisateur souhaite garder les doublons dans les résultats
- Les fonctions d'agrégation doivent généralement garder les doublons
- **DISTINCT** : Quand on souhaite éliminer les tuples dupliqués

```
SELECT DISTINCT Salaire
FROM EMPLOYE;
```

- **ALL** : retourne tous les tuples même s'ils sont dupliqués. Elle s'applique par défaut.

Requêtes SQL : opérations ensemblistes

- SQL intègre les opérations de base sur les ensembles ; l'union (UNION), l'intersection (INTERSECT) et la différence (EXCEPT).
- Ces opérations s'appliquent aux relations qui ont les mêmes attributs. Ces derniers doivent être dans le même ordre.
- Le résultat des ses opérations est un **ensemble.**

- **Exemple** : Liste des projets sur lesquels a travaillé l'employé de nom 'Alaoui' aussi bien en tant que collaborateur qu'en tant que chef du département en charge de ces projets

```
(SELECT Pnum
FROM EMPLOYE, TRAVAILLE_SUR
WHERE Matricule = EMat AND Nom = 'Alaoui')
UNION
(SELECT Pnum
FROM DEPARTEMENT, EMPLOYE, PROJET
WHERE ChefMat  = Matricule AND Projet.Dnum = Dnumero AND Nom =
'Alaoui');
```

Requêtes SQL : Chaînes de caractères et opérations arithmétiques

- '%' remplace zéro ou plusieurs caractères
- '_' remplace un seul caractère

- **Exemple** : Noms et prénoms des employés qui résident à Tétouan

```
SELECT Nom, Prenom
FROM EMPLOYE
WHERE Adresse like '%Tetouan%';
```

- **Exemple** : Simuler une augmentation de salaire de 10% pour les employés travaillant sur le projet 'Web'

```
SELECT Nom, Prenom, Salaire*1.1 AS Nouv_Salaire
FROM EMPLOYE AS E , PROJET AS P, TRAVAILLE_SUR AS T
WHERE E.Matricule = T.Emat AND P.PNum = T.PNum AND P.PNom = 'Web';
```

Requêtes SQL : Chaînes de caractères et opérations arithmétiques

- Noms et prénoms des employés dont les salaires sont entre 5000 et 7000

```
SELECT Nom, Prenom
FROM EMPLOYE
WHERE Salaire BETWEEN 5000 AND 7000;
```

Requêtes SQL : Ordonner les résultats

• Liste des employés ordonnée par département en ordre décroissant, puis, pour chaque département, par nom employé en ordre croissant.

```
SELECT D.DNom, E.Nom, E.Prenom
FROM DEPARTEMENT, EMPLOYE
WHERE D.DNum = E.DNum
ORDER BY D.DNom DESC,  E.Nom ASC;
```

• Par défaut, l'ordre est croissant (ASC)

Requêtes SQL : Insertion

- L'insertion porte sur un tuple.

```
INSERT INTO EMPLOYE
VALUES (1165, 'Alaoui', 'Ahmed', 'Tetouan', 01/01/1980', 'M',8000,
1120, 2);
```

- On peut préciser les attributs à insérer.

```
INSERT INTO EMPLOYE(Matricule, Nom, Adresse, Prenom)
VALUES (1165, 'Alaoui', 'Tetouan', 'Ahmed');
```

- Dans ce cas les attributs pour lesquelles des valeurs n'ont pas été insérées prennent leurs valeurs par défaut ou sont mis à NULL. Autrement l'insertion est rejetée.

Requêtes SQL : Modification

- **UPDATE**

```
UPDATE EMPLOYE
SET Salaire = Salaire*1.1
WHERE Matricule  = 1165;
```

Requêtes SQL : Suppression

- **DELETE**

✓ **DELETE FROM** EMPLOYE
WHERE Nom = ‘Alaoui’;

✓ **DELETE FROM** EMPLOYE

Requêtes SQL : Table de vérité comparaisons avec NULL

- Les conditions comportent des comparaisons qui peuvent porter sur des valeurs NULL
- Une valeur est NULL si elle n'est pas connue, n'est pas disponible ou n'est pas applicable
- une valeur NULL n'est pas égale à une autre valeur NULL
- Quand une valeur NULL prend part à une comparaison elle est considérée INCONNU

Requêtes SQL : Comparaisons avec NULL

AND	Vrai	Faux	Inconnu
Vrai	Vrai	Faux	Inconnu
Faux	Faux	Faux	Faux
Inconnu	Inconnu	Faux	Inconnu

OR	Vrai	Faux	Inconnu
Vrai	Vrai	Vrai	Vrai
Faux	Vrai	Faux	Inconnu
Inconnu	Vrai	Inconnu	Inconnu

NOT	Vrai
Vrai	Faux
Faux	Vrai
Inconnu	Inconnu

Requêtes SQL : Test attribut NULL

• Comparer la valeur d'un attribut avec NULL se fait à l'aide de **IS** et **IS NOT**.

✓ Liste des employés qui n'ont pas un supérieur hiérarchique

```
SELECT Nom, Prenom
FROM EMPLOYE
WHERE SuperMat IS NULL
```

Imbrication des requêtes SQL

```
(SELECT Pnum
FROM EMPLOYE, TRAVAILLE_SUR
WHERE Matricule = EMat AND Nom =
'Alaoui')
UNION
(SELECT Pnum
FROM DEPARTEMENT, EMPLOYE,
PROJET
WHERE ChefMat  = Matricule AND Dnum =
Dnumero AND Nom = 'Alaoui');
```

Forme imbriquée

```
SELECT Pnum
FROM PROJET
WHERE PNum IN
(SELECT PNum
FROM EMPLOYE,TRAVAILLE_SUR
WHERE Matricule = EMat AND Nom =
'Alaoui')
OR
PNum IN
(SELECT PNum
FROM DEPARTEMENT, EMPLOYE,
PROJET
WHERE ChefMat  = Matricule AND DNum
= Dnumero AND Nom = 'Alaoui');
```

Imbrication des requêtes SQL : Comparaison entre tuples

- SQL permet également la comparaison entre des tuples

- Liste des matricules des employés avec les combinaisons (projet, heures) sur un projet pour lequel travaille l'employé de matricule '1165'

```
SELECT DISTINCT EMat
FROM TRAVAILLE_SUR
WHERE (Pnum, Heures) IN
        (SELECT Pnum, Heures
        FROM TRAVAILLE_SUR
        WHERE Emat = 1165);
```

Imbrication des requêtes SQL : opérateurs arithmétiques

• En plus de **IN**, d'autres opérateurs permettent de comparer une valeur (un nom d'attribut) avec un ensemble de valeurs (requête imbriquée). '**=SOME**' et '**=ANY**' qui sont équivalentes à **IN**.

• on peut également utiliser les autres opérateurs de comparaison >, >=, <, <= et <> en combinaison avec **ANY**, **SOME** et **ALL**.

✓ Liste des employés qui ont un salaire supérieur à celui de tous les employés du département n°2

```
SELECT Nom, Prenom
FROM EMPLOYE
WHERE Salaire > ALL
                (SELECT Salaire
                FROM EMPLOYE
                WHERE Dnum = 2);
```

Imbrication des requêtes SQL : ambiguïté des noms d'attributs

✓ Liste des employés qui ont des enfants qui ont le même prénom et qui sont du même sexe qu'eux-mêmes

```
SELECT E.Nom, E.Prenom
FROM EMPLOYE AS E
WHERE E.Matricule IN
                    ( SELECT EMat
                    FROM ENFANT AS Enf
                    WHERE E.Prenom=Enf.NomEnfant
                    AND E.Sexe=Enf.Sexe );
```

Imbrication des requêtes SQL : EXISTS

- EXISTS est utilisé pour vérifier si le résultat d'une requête est vide

✓ Liste des employés qui ont des enfants qui ont le même prénom et qui sont du même sexe qu'eux-mêmes

```
SELECT E.Nom, E.Prenom
FROM EMPLOYE E
WHERE EXISTS
            ( SELECT *
            FROM ENFANT Enf
            WHERE E.Matricule = Enf.EMat
            AND E.Prenom=Enf.NomEnfant
            AND E.Sexe=Enf.Sexe );
```

Imbrication des requêtes SQL : EXISTS

✓ Liste des employés qui n'ont pas d'enfants

```
SELECT Nom, Prenom
FROM EMPLOYEE
WHERE NOT EXISTS
                ( SELECT *
                FROM ENFANT
                WHERE Matricule = Emat);
```

Ensembles explicites

✓ Liste des employés qui travaillent sur les projets 1,2 et 3

```
SELECT Emat
FROM TRAVAILLE_SUR
WHERE PNum  IN (1,2,3);
```

« Renommage » des attributs

✓ Noms des employés et de leurs supérieurs immédiats

```
SELECT E.Nom AS NomEmploye, S.Nom As NomSuperieur
FROM EMPLOYE E, EMPLOYE S
WHERE S.Matrciule = E.SuperMat;
```

Jointure et jointure externes

- SQL permet de spécifier directement des jointures entre les tables dans la clause FROM
- Ceci permet de séparer les conditions de jointure des conditions de sélection dans la clause WHERE

✓ **Liste des employés du département INFORMATIQUE**

```
SELECT Nom, Prenom, Adresse
FROM (EMPLOYE JOIN DEPARTEMENT ON  DNumero = DNum)
WHERE DNom = 'INFORMATIQUE';
```

Jointure et jointure externes

- Avec **NATURAL JOIN** (Jointure naturelle) entre deux relations R et S, la condition de jointure est implicite et porte sur tous les attributs de même nom de R et S.

- Si les attributs ne sont pas de même nom, on peut les renommer pour qu'ils coïncident.

✓ **Liste des employés du département INFORMATIQUE**

```
SELECT Nom, Prenom, Adresse
FROM (EMPLOYE NATURAL JOIN DEPARTEMENT)
WHERE DNom = 'INFORMATIQUE';
```

Jointure et jointure externes

- Par défaut, les jointures sont internes, c'est-à-dire que ne sont inclus que les tuples d'une relation pour lesquels il existe des tuples correspondants dans l'autre relation
- Si on souhaite extraire les autres tuples, on a recours à des jointures externes.

Jointure interne	Jointure Externe
SELECT E.Nom AS NomEmploye S.Nom AS NomSuperieur FROM (EMPLOYE AS E **JOIN** EMPLOYE AS S **ON** S.Matricule = E.SuperMat);	SELECT E.Nom AS NomEmploye S.Nom AS NomSuperieur FROM (EMPLOYE AS E **LEFT OUTER JOIN** EMPLOYE AS S **ON** S.Matricule = E.SuperMat);
Liste des noms d'employés et de leurs supérieurs respectifs. Les employés avec SuperMat NULL ne sont pas retournés	Tous les employés sont inclus dans la liste

Jointure et jointure externes

- Il existe plusieurs manières de spécifier des jointures en SQL :
  - INNER JOIN (équivalent à JOIN)
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
  - FULL JOIN

Fonctions d'agrégats

- Les fonctions d'agrégats permettent de résumer l'information en provenance de plusieurs tuples.
- Les fonctions de groupement permettent de créer des groupes et des sous-groupes.
- Fonctions d'agrégats : COUNT, SUM, MAX, MIN, AVG

✓ Somme, maximum, minimum et moyenne des salaires des employés

```
SELECT SUM(Salaire), Max(Salaire), MIN(Salaire), AVG(Salaire)
FROM EMPLOYE;
```

Fonctions d'agrégats

✓ Le nombre total d'employés

```
SELECT COUNT(*)
FROM EMPLOYE;
```

✓ Le nombre d'employés du département 'INFORMATIQUE'

```
SELECT COUNT(*)
FROM EMPLOYE, DEPARTEMENT
WHERE DNum = DNumero AND Dnom = 'INFORMATIQUE';
```

✓ La liste des employés qui ont plus de deux enfants

```
SELECT Nom, Prenom
FROM EMPLOYE
WHERE (SELECT COUNT(*)
       FROM ENFANT
       WHERE Emat = Matricule) >= 2;
```

Groupement : GROUP BY et HAVING

• Dans plusieurs cas, on souhaite appliquer les fonctions d'agrégat à des groupes de tuples définis à partir de certaines valeurs d'attributs

•La clause GROUP BY permet de réaliser cette opération

✓ Pour chaque département, donner le numéro du département, le nombre d'employés et le salaire moyen de ces employés.

```
SELECT Dnum, COUNT(*), AVG(Salaire)
FROM EMPLOYE
GROUP BY DNum;
```

• Remarque : si l'attribut de groupement est NULL pour certains tuples, ces derniers sont mis dans un groupe séparé.

Groupement : GROUP BY et HAVING

• SQL permet d'exprimer des conditions sur les groupes à l'aide de la clause **HAVING**.

✓ Supposons que l'on souhaite connaître les départements qui comptent parmi leurs employés, au moins 5 avec un salaire supérieur à 10 000.

```
SELECT Dnom, COUNT(*)
FROM EMPLOYE, DEPARTEMENT
WHERE DNum = DNumero AND Salaire > 10000
GROUP BY Dnum
HAVING COUNT(*) > 5;
```

Les assertions

Les **assertions** permettent de définir des contraintes en plus de celles inhérentes au modèle relationnel (Clés, contrainte d'entité et d'intégrité référentielle) qui peuvent être définies à l'aide de CREATE TABLE.

✓ EXEMPLE : Contrainte qui spécifie que le salaire d'un employé affecté à un département ne peut pas être supérieur à celui du chef de ce département.

```
CREATE ASSERTION SALAIRE_CONTRAINTE
CHECK (NOT EXISTS (SELECT *
                   FROM EMPLOYE E, EMPLOYE C,
                   DEPARTEMENT D
                   WHERE E.Salaire > C.Salaire
                   AND E.Dnum = C.Dnum
                   AND D.ChefMAt = C.Matricule));
```

Les déclencheurs (triggers)

CREATE TRIGGER permet de spécifier des actions automatiques à accomplir suite à l'occurrence de certains événements et sous certaines conditions.

✓ EXEMPLE 1 (syntaxe ORACLE) : Garder l'historique des modifications des salaires des employés

```
CREATE TRIGGER SALAIREModif
BEFORE INSERT OR UPDATE OF SALAIRE
ON EMPLOYE
FOR EACH ROW
INSERT INTO SALAIRE_HIST  VALUES (MATRICULE, SYSDATE, SALAIRE)
```

Les déclencheurs (triggers)

✓ EXEMPLE 2 (syntaxe ORACLE) : alerter le supérieur (procédure stockée ALERTER_SUPERIEUR) au cas où le salaire d'un employé dépasse celui de son chef de département (condition) suite à l'insertion ou à la mise à jour d'un employé (événement).

```
CREATE TRIGGER SALAIRE
BEFORE INSERT OR UPDATE OF SALAIRE, SUPERMAT
ON EMPLOYE
FOR EACH ROW
WHEN ( NEW.SALAIRE > ( SELECT SALAIRE FROM EMPLOYE
WHERE MATRICULE = NEW. SUPERMAT) )
ALERTER_SUPERIEUR(NEW.CHEFMAT, NEW.MATRICULE );
```

Les vues(tables virtuelles)

- Une vue est générée à partir d'autres tables. Ces tables peuvent être des tables de base (physique) ou des vues.

- Une vue est une table virtuelle. Elle n'a pas d'existence physique contrairement aux tables de base dont les tuples sont physiquement enregistrés dans la base de données.

- Une vue peut être vue comme une façon de spécifier une table à laquelle on recourt souvent

Les vues(tables virtuelles)

- **Exemple :**

```
CREATE VIEW TRAVAILLE(NomEmp, NomProjet, Heures)
AS SELECT E.Nom, P.PNom, T.Heures
FROM EMPLOYE E, PROJET P, TRAVAILLE_SUR T
WHERE E.MATRICULE=T.EMAT AND T.PNUM=P.PNUM;

SELECT NomEmp
FROM TRAVAILLE
WHERE NomProjet = 'Site Web'
```

Implémentation des vues

-Les vues sont implémentées de deux façons différentes. La première consiste à transformer les requêtes sur les vues en requêtes sur les tables de base. Par exemple, la requête précédente sera transformée en :

```
SELECT E.Nom, P.PNom, T.Heures
FROM EMPLOYE E, PROJET P, TRAVAILLE_SUR T
WHERE E.MATRICULE=T.EMAT AND T.PNUM=P.PNUM AND P.NomProjet ='Site Web';
```

La seconde stratégie consiste à créer des **vues matérialisées**. Une vue matérialisée consiste à créer une table physique qui contient la vue lors de sa première invocation. Cette table sera ensuite mise à jour continuellement.

Modification des vues

-La modification des vues est un problème complexe qui peut être la source de beaucoup d'ambiguïté.

- La modification d'une vue sans agrégats issue d'une seule table se traduit par la modification de la table sous-jacente

- la modification d'une vue basée sur des jointures peut donner lieu à la modification des tables de base de plusieurs façons.

-Exemple :

```
UPDATE TRAVAILLE
SET NomProjet = 'ERP'
WHERE NomEmp = 'Alaoui' AND NomProjet = 'Site Web';
```

Modification des vues

- Voici deux interprétations possibles pour cette modification :

a.
```
UPDATE TRAVAILLE_SUR
 SET PNum = (SELECT PNum
             FROM PROJET
             WHERE Pnom = 'ERP')
 WHERE EMat IN (SELECT Matricule
             FROM EMPLOYE
             WHERE Nom = 'Alaoui')
 AND PNum IN (SELECT PNum
             FROM PROJET
             WHERE PNom ='Site Web');
```

b.
```
UPDATE PROJET
SET PNom = 'ERP'
WHERE PNom = 'Site Web';
```

DROP et ALTER

-Permettent d'ajouter ou de supprimer des tables, des attributs et des contraintes

- **DROP** permet de supprimer des tables, des attributs ou des contraintes.

DROP TABLE ENFANT **CASCADE;**

- CASCADE : tous les éléments qui font référence à la table sont également supprimés (clés étrangères, vues)

DROP TABLE ENFANT **RESTRICT;**

- RESTRICT: la table est supprimée seulement si elle n'est pas référencée par des contraintes.

DROP et ALTER

- ALTER permet de modifier les éléments du schéma.

- Pour une table, ALTER permet d'ajouter ou de supprimer une colonne, changer sa définition, ajouter ou supprimer une contrainte.

ALTER TABLE EMPLOYE ADD COLUMN Fonction CHAR(12);