Stockage et traitement des données

SQL: ***S**tructured **Q**uery **L**anguage*

Création d'une Base de Données

CREATE DATABASE

sql > **create database** unebase

Connexion à une BD

CONNECT TO (USE)

sql > **connect to** unebase;

BD Bibliothèque

Membre (IdMembre, NomMembre, AdrMembre, CpMembre)
Biblio (IdBiblio, NomBiblio, AdrBiblio, CpBiblio)
Livre (IdLivre, TitreLivre, AuteurLivre, CategLivre, PrixLivreJour, *IdBiblio*)
Emprunt (*IdLivre, IdMembre*, DatEmprunt, DurEmprunt)

Type des attributs
- -IdMembre, CategLivre, DurEmprunt, IdBiblio: entier
- - CategLivre:smallint
- - DatEmprunt: date
- - PrixLivreJour : réel
- - NomMembre, AdrMembre, CpMembre, NomBiblio, AdrBiblio, CpBiblio, IdLivre, TitreLivre, AuteurLivre, IdLivre: chaînes de caractères

Les clés primaires sont soulignées, les clés étrangères sont en *italique*.

DEFINITION DES DONNEES

Définition d'une relation
-> **CREATE TABLE**

- Arguments :

✓ le nom de la relation à créer

✓ une liste des noms et types des attributs + des spécifications de contraintes d'attributs

✓ des spécifications de contraintes de tables

- **Contraintes d'attributs**

Clauses ***PRIMARY KEY***, ***NOT NULL,***
REFERENCES, CHECK (<prédicat>)

```
CREATE TABLE Livre (
        IdLivre INTEGER NOT NULL PRIMARY KEY,
        TitreLivre CHAR(40),
        AuteurLivre CHAR(30) NOT NULL,
        CategLivre SMALLINT DEFAULT 1 CHECK
(CategLivre < 6),
        PrixL INTEGER,
        IdBiblio INTEGER NOT NULL,
        FOREIGN KEY(IdBiblio)
                REFERENCES Biblio(IdBiblio)
        );
```

Possibilité de définir n (>= 0) contraintes et de les nommer

- **Contraintes de table**

Clauses **PRIMARY KEY, FOREIGN KEY, CHECK (<prédicat>)**

--> impliquent des attributs déjà définis dans la table ou appartenant à d'autres tables

--> seule manière pour déclarer une clé étrangère ou une clé primaire multi-attributs

```
Ex      CREATE TABLE Emprunt (
                IdLivre CHAR(8) NOT NULL,
                IdMembre INTEGER NOT NULL,
                DatEmprunt DATE,
                DurEmprunt INTEGER,
                PRIMARY KEY (IdLivre, IdMembre),
                                ...
        );
```

Modification d'une table

ALTER TABLE <nom_de_table>
ADD / DROP
{COLUMN <définition_d'attribut>}
{CONSTRAINT <nom_contrainte> <définition_de_contrainte(s)>}

Ex **ALTER TABLE** Livre
DROP COLUMN CategLivre ;

ALTER TABLE Membre
ADD COLUMN NbL INTEGER ;

Suppression d'une table

Ex **DROP TABLE** Livre;

Clause de contrainte d'attribut : génération automatique de valeurs d'une clé (SQL3)

Syntaxe :

<nom> <type> GENERATED ALWAYS AS IDENTITY
(START WITH <val1>, INCREMENT BY <val2>)

Ex : **CREATE TABLE** Etudiant (
Ide Integer
PRIMARY KEY
GENERATED ALWAYS AS IDENTITY
(START WITH 1, INCREMENT BY 1),
…
);

ECRITURE DANS UNE BASE

INSERT

INSERT … VALUES

Ex : **INSERT INTO** Membre **VALUES**
(2, 'Durand', 'Marseille', '13008');

Les attributs que l'on ne veut pas renseigner doivent être forcés à ***NULL*** (attention aux contraintes existantes)

Un attribut déclaré avec une valeur par défaut V et renseigné avec ***NULL*** est initialisé avec V

On peut n'insérer qu'une partie de tuple dans une relation :

INSERT INTO Membre (IdMembre, NomMembre)
VALUES (12, 'Dupont');

Les attributs non renseignés prennent pour valeur celle existante par défaut

INSERTION avec génération automatique de valeurs
Ex : **INSERT INTO** Etudiant **VALUES**
(DEFAULT, 'Peyo', 'Paul', NULL, '1961-2-25' , 'Aix', 2);

Modification des données d'une table

UPDATE et DELETE

UPDATE Livre
SET PrixLivreJour = 0.10
WHERE PrixLivreJour = 0.9;

DELETE FROM Emprunt
WHERE DatEmprunt < '2004-6-1';

RECHERCHE DE DONNEES

SELECT … FROM … { WHERE }

La clause ***SELECT*** définit les attributs de la relation résultante

La clause ***FROM*** spécifie les relations sur lesquelles porte la recherche

La clause ***WHERE*** (optionnelle) indique des conditions de restriction

PROJECTION et SELECTION

• Projection: Choisir des colonnes d'une table

```
SELECT TitreLivre, AuteurLivre
    FROM Livre;
```

• Sélection : Choisir un sous-ensemble de tuples d'une table

```
SELECT *
FROM Livre
WHERE AuteurLivre = 'Albert_Camus'
AND TitreLivre='La_peste';
```

Le résultat d'une requête SQL est visualisé sous forme de table mais ne peut être sauvegardé

PROJECTION et SELECTION

Combinaison des deux:

```
SELECT TitreLivre, AuteurLivre
    FROM Livre
    WHERE CategLivre=2;
```


Prédicats

L'expression qui suit la clause ***WHERE*** est un prédicat (une expression logique vraie ou fausse) pouvant comporter

--> les comparateurs usuels (=, <>, ...), ou
--> l'un des mots-clé suivants : ***ALL, ANY, BETWEEN, EXISTS, IN, IS NULL*** et ***LIKE***

```
SELECT IdLivre, TitreLivre, AuteurLivre
    FROM Livre
    WHERE AuteurLivre LIKE '%ar%'
        AND CategLivre BETWEEN 1 AND 3;
```


Fonctions de calcul intégrées

SUM, ***AVG***, ***MIN***, ***MAX***, ***COUNT*** s'appliquent à des attributs numériques, et peuvent se trouver au niveau de la projection (ou dans une clause ***HAVING***)

```
SELECT COUNT(*)
    FROM Livre
    WHERE CategLivre NOT IN (1, 3);
```


Clauses DISTINCT et ORDER BY

✓ **DISTINCT** : élimination des doublons

✓ **ORDER BY** : tri du résultat

```
SELECT DISTINCT NomMembre
    FROM Membre
    WHERE CpMembre='13012'
    ORDER BY NomMembre ASC;
```


Clauses GROUP BY et HAVING

GROUP BY regroupe les données afin d'effectuer des calculs par paquet

"Quel est le nombre de livres pour chaque auteur dans Livre?"

```
SELECT AuteurLivre, COUNT(*)
FROM Livre
GROUP BY AuteurLivre;
```

GROUP BY peut être suivie d'une condition de sélection portant sur certains blocs de la partition

```
SELECT AuteurLivre, COUNT(*)
    FROM Livre
    GROUP BY AuteurLivre
    HAVING COUNT (DISTINCT IdLivre) >= 2;
```

IMPORTANT : tous les attributs (hormis ceux résultats d'un calcul) projetés dans un **SELECT** doivent figurer dans le **GROUP BY**, et inversement

clause CASE … WHEN

Arrangement les données

Afficher le nombre de romans, de nouvelles et de bandes dessinées.

```
SELECT    CASE CategLivre
              WHEN 1 THEN 'Romans'
              WHEN 2 THEN 'Nouvelles'
              WHEN 4 THEN 'Bandes dessinées'
          END AS Categorie,
          COUNT(DISTINCT IdLivre) AS Nombre
FROM Livre
WHERE CategLivre IN (1, 2, 4)
GROUP BY CategLivre ;
```


JOINTURES

Requêtes sur des tables ayant des attributs en commun

✓**Ensembliste**: imbrication de sous-requêtes

✓**Prédicative**: utilisation de variables

« Quels sont les noms des personnes qui empruntent des livres pour une durée supérieure à 8 jours? »

JOINTURES

- Méthode ensembliste :

```
SELECT NomMembre
FROM Membre
WHERE IdMembre IN (
    SELECT IdMembre
    FROM Emprunt
    WHERE DurEmprunt > 8);
```


JOINTURES

Méthode prédicative :

```
SELECT NomMembre
FROM Membre M, Emprunt E
WHERE M.IdMembre = E.IdMembre AND DurEmprunt >8;
```

ou

```
SELECT NomMembre
FROM Membre INNER JOIN Emprunt
    ON Membre.IdMembre = Emprunt.IdMembre
WHERE DurEmprunt>8;
```

- qualifier les noms d'attributs posant conflit

AUTO-JOINTURE

Ex : "Quels sont les noms membres qui ont la même adresse que Dupond?"

```
SELECT M1.NomMembre
FROM Membre M1, Membre M2
WHERE M1.AdrMembre = M2.AdrMembre
    AND M2.NomMembre = 'Dupond';
```

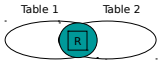
ou:

```
SELECT NomMembre
FROM Membre
WHERE AdrMembre IN (
    SELECT AdrMembre
    FROM Membre
    WHERE NomMembre = 'Dupond');
```

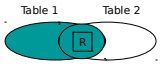
JOINTURES EXTERNES

-> permettent de récupérer les tuples de la jointure interne + certains tuples sans correspondance dans au moins l'une des relations jointes

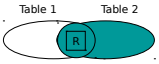
Jointure interne



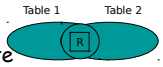
Jointure externe gauche



Jointure externe droite



Jointure externe complète



JOINTURES EXTERNES

Ex : Quels sont les noms des membres du 13ème arrondissement de Marseille, qu'ils aient ou non emprunté un livre, ainsi que le numéro du livre (si livre emprunté)

```
SELECT NomMembre, IdLivre
FROM Membre LEFT OUTER JOIN Emprunt ON
        Membre.IdMembre =  Emprunt.IdMembre
WHERE  CpMembre = '13013' ;
```


JOINTURES EXTERNES

Ex : Donner les noms des auteurs ayant écrit des romans que ces romans aient été empruntés ou non ainsi que la date d'emprunt (si emprunt)

```
SELECT DISTINCT AuteurLivre, DatEmprunt
FROM Emprunt RIGHT OUTER JOIN Livre
      ON Emprunt.IdLivre = Livre.IdLivre
WHERE CategLivre=1;
```


JOINTURES EXTERNES

Ex : Donner les noms des membres et des bibliothèques ainsi que leurs adresses, quelles que soient ces adresses (mais différentes d'Aix_en_Pce)

```
SELECT M.NomMemb AS Nom_M, B.NomBiblio AS Nom_B,
E.adresse AS Adr_E
FROM Membre M FULL OUTER JOIN Biblio B
      ON M.AdrMembre = B.AdrBiblio
WHERE M.AdrMembre <> 'Aix_en_Pce'  ;
```


prédicats supplémentaires

IN '= ANY' (ou '= SOME')

Un prédicat utilisant le mot clé **ANY** est vrai si la liaison est vraie pour au moins un tuple de la relation spécifiée par la commande **SELECT** qui suit **ANY**

L'emploi de **ALL** force la comparaison de chaque valeur des éléments rendus par la sous-requête

«Quel est le numéro des livres empruntés postérieurement à tous les livres d'Albert Camus?«

```
SELECT IdLivre
FROM Emprunt
WHERE DatEmprunt > ALL ( SELECT DatEmprunt
                         FROM Emprunt
                         WHERE IdLivre IN (
                            SELECT IdLivre
                            FROM Livre
                            WHERE              AuteurLivre=
'Albert_Camus'));
```

"Quel est le numéro des livres empruntés postérieurement à un des livres d'Albert Camus? »

```
SELECT IdLivre
FROM Emprunt
WHERE DatEmprunt > ANY ( SELECT DatEmprunt
                         FROM Emprunt
                         WHERE IdLivre IN (
                            SELECT IdLivre
                            FROM Livre
                            WHERE AuteurLivre=
'Albert_Camus'));
```


La requête reformulée devient :

"Pour chaque numéro de membre candidat à la réponse à cette requête, il ne faut pas qu'il y ait de livre écrit par Albert Camus n'ayant pas été emprunté"
La requête SQL correspondante s'écrit :

```
SELECT IdMembre FROM Emprunt E1
WHERE NOT EXISTS (
            SELECT IdLivre
            FROM Livre
            WHERE Autl = 'Albert_Camus'
            AND NOT EXISTS (
                  SELECT *
                  FROM Emprunt E2
                  WHERE E2.IdLivre = Livre.IdLivre
                              AND   E2.AuteurLivre   =
E1.AuteurLivre ) );
```


EXISTS

Un prédicat **EXISTS** est vrai si le nombre de tuples présents dans la relation résultat correspondant à l'expression de sélection associée à l'opérateur **EXISTS** est non nul

```
SELECT IdMembre
FROM Membre
WHERE NOT EXISTS (
      SELECT * FROM Emprunt
        -- ou SELECT IdMembre FROM Emprunt
                  WHERE   Emprunt.IdMembre   =
Membre.IdMembre);
```

s'interprète par "Quels sont les membres n'ayant pas empruntés de livre? »

Autre formulation possible

On sélectionne les membres ayant emprunté autant de livres écrits par Albert Camus qu'il y a de livre écrits par Albert Camus
Ceci donne (s4_02) :

```
SELECT IdMembre FROM Emprunt
WHERE IdLivre IN ( SELECT IdLivre FROM Livre
                              WHERE   AuteurLivre   =
'Albert_Camus'    )
GROUP BY IdMembre
HAVING COUNT (DISTINCT IdLivre) = (
            SELECT COUNT (*) FROM Livre
            WHERE AuteurLivre = 'Albert_Camus');
```

Cette seconde approche n'est possible que dans certains cas particuliers de requêtes

DIVISION

"Quels sont les numéros des membres ayant empruntés tous les livres d' Albert Camus?"

Traduction en langage (pseudo-) algébrique :

R_{11} ← SELECTION Livre et Livre.AuteurLivre = 'Albert_Camus'
R1 ← PROJECTION R_{11} (IdLivre)
R2 ← PROJECTION Emprunt (IdMembre, IdLivre)
ResDiv ← DIVISION R2 (IdMembre, IdLivre) / R1 (IdLivre)

Pb : en SQL, le quantificateur " ∀" n'existe pas

Artifice : utiliser une double négation
$\forall x\ P(x) \quad \Leftrightarrow \quad \neg(\exists x\ \neg P(x))$

PRODUIT CARTESIEN

```
SELECT *
      FROM Membre, Livre;
```

UNION, INTERSECTION et DIFFERENCE

Ces opérateurs manipulent des tables ayant tous les attributs en commun (même nombre de colonnes, même types, même ordre)

Union : "Quels sont les numéros des livres dont l'auteur est Albert Camus, ainsi que ceux qui ont été empruntés pendant 8 jours ?"

```
SELECT IdLivre FROM Livre
WHERE AuteurLivre = 'Albert_Camus'
UNION
SELECT IdLivre FROM Emprunt
WHERE DurEmprunt=8;
```


Intersection

« Quels sont les numéros des romans ayant été empruntés pendant plus de 8 jours?»

```
SELECT IdLivre FROM Livre
WHERE CategLivre=1
INTERSECT
SELECT IdLivre FROM Emprunt
WHERE DurEmprunt >8;
```


Différence

"Quels sont les codes des bibliothèques qui n'ont pas de livre dont l'auteur est Albert Camus?"

```
SELECT IdBiblio FROM Biblio
EXCEPT
SELECT IdBiblio FROM Livre
WHERE AuteurLivre ='Abert_Camus';
```


INTERSECT et **EXCEPT** peuvent être remplacés par des prédicats **EXISTS** et **NOT EXISTS**

```
SELECT IdLivre FROM Livre
WHERE CategLivre=1 AND EXISTS (
      SELECT IdLivre FROM Emprunt
      WHERE DurEmprunt > 8
            AND Livre.IdLivre = Emprunt.IdLivre);

SELECT IdBiblio FROM Biblio
  WHERE NOT EXISTS (
      SELECT * FROM Livre
      WHERE AuteurLivre='Albert_Camus '
            AND Biblio.IdBiblio = Livre.IdBiblio);
```


ECRITURE DANS UNE BASE

•**INSERT** (2ème forme)

création d'une copie "simplifiée" de Livre

```
CREATE TABLE Copie_Livre
(IdLivre INTEGER, AuteurLivre CHAR(25));

INSERT INTO Copie_Livre
SELECT IdLivre, AuteurLivre FROM Livre;
```

•**UPDATE** et **DELETE**

suppression restrictive de tuples (s4_08)

```
DELETE FROM Livre
WHERE CategLivre = 2 AND IdLivre NOT IN (
            SELECT IdLivre FROM Emprunt);
```