

Cours

“Bases de données”

3° année (MIS)

Antoine Cornuéjols

www.lri.fr/~antoine
antoine.cornuejols@agroparistech.fr

Langage de manipulation des données

I. *La partie LMD de SQL*

- I.1. Les requêtes de base
- I.2. Insertions, suppressions, mises à jour
- I.3. Autres fonctionnalités

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

SELECT-FROM-WHERE

```
SELECT    <liste d'attributs>
FROM      <liste de tables>
WHERE     <condition> ;
```

- <liste d'attributs> : liste des noms des attributs dont la requête doit extraire la valeur
- <liste de tables> : liste des noms des relations nécessaires pour traiter la requête
- <condition> : expression conditionnelle (booléenne) identifiant les tuples à récupérer par la requête

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

SELECT-FROM-WHERE

```
SELECT    <liste d'attributs>
FROM      <liste de tables>
WHERE     <condition> ;
```

```
SELECT Date_Naiss, Adresse
FROM EMPLOYÉ
WHERE PRENOM='Bernard' AND NOM='Schmidt' ;
```

Requête **SELECT** = opération de SELECTION-PROJECTION

- **SELECT** spécifie les *attributs de la projection*
- **WHERE** spécifie la *condition de sélection*

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

SELECT-FROM-WHERE (syntaxe générale)

```
SELECT      nom_table, nom_colonnes*
FROM        nom_table*
[WHERE      conditions_de_sélection_sur_lignes*]
[GROUP BY   nom_colonne_de_regroupement*]
[HAVING     conditions_de_sélection_sur_groupe*]
[ORDER BY   nom_colonne_tri*]
```

* = plusieurs occurrences possibles ; [] = optionnel

5

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

Chercher la date de naissance et l'adresse des employés dont le nom est 'John B. Smith'

```
SELECT Date_Naiss, Adresse
FROM EMPLOYÉ
WHERE PRENOM='Bernard' AND NOM='Schmidt' ;
```

ATTENTION : La clause SELECT peut ramener des tuples dupliqués.

```
SELECT DISTINCT auteur
FROM Livre;
```

6

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

Chercher dans la table LIVRE le titre et le genre des ouvrage écrits par DUMAS :

```
SELECT titre, genre FROM Livre WHERE auteur = 'Dumas';
```

Chercher les ouvrages non écrits par Dumas et postérieurs à 1835 :

```
SELECT * FROM Livre WHERE auteur <> 'Dumas' AND année > 1835;
```

Chercher le nom (et lieu de naissance) des auteurs nés entre 1802 et 1850 :

```
SELECT auteur, lieu FROM Ecrivain WHERE né_en BETWEEN 1802 AND 1850;
```

Chercher les auteurs dont le nom commence par B :

```
SELECT * FROM Evrivain WHERE auteur LIKE 'B%';
```

7

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

Chercher les auteurs dont le nom contient un A en deuxième position :

```
SELECT * FROM Evrivain WHERE auteur LIKE '_A%';
```

Chercher les ouvrages (auteur, titre, année) publiés en 1839, 1866 ou 1857 :

```
SELECT auteur, titre, année FROM Livre WHERE année IN
(1839,1866,1857);
```

Chercher les ouvrages pour lesquels on dispose de l'année de parution

```
SELECT auteur, titre, année FROM Livre WHERE année NOT NULL;
```

8

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

Expression des restrictions

Restriction = combinaison booléenne (or, and, not) de conditions élémentaires portant sur les colonnes d'une table.

Prédicats de restriction : permettent la comparaison d'une valeur portée par une colonne à une valeur constante.

- à l'aide des opérateurs =, <, >, <=, >=
- à l'aide du prédicat d'intervalle **BETWEEN** qui teste si la valeur de l'attribut est compris entre deux valeurs constantes. L'attribut doit porter une valeur numérique ou de type date
- à l'aide du prédicat de comparaison de texte **LIKE** qui teste si un attribut de type chaîne contient une ou plusieurs sous-chaînes. Le caractère souligné remplace un caractère quelconque. Le caractère % remplace une séquence de caractères
- à l'aide du prédicat de test de nullité (attribut non renseigné) **NULL**
- à l'aide du prédicat d'appartenance **IN** qui teste si la valeur de l'attribut appartient à une liste de valeurs

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

Tri du résultat À l'aide de la clause **ORDER BY** suivie de la liste des attributs servant de critère au tri.

Chercher les stations par ordre croissant de tarifs et par ordre alphabétique :

```
SELECT * FROM Station ORDER BY tarif, nomStation;
```

Pour **trier en ordre descendant**, on ajoute le mot-clé **DESC** après la liste des attributs concernés.

```
SELECT * FROM Station ORDER BY tarif DESC, nomStation ASC;
```

10

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

Expression des jointures

Une **jointure sans qualification** (sans clause **WHERE**) est le **produit cartésien**.

Dans une **jointure avec qualification**, le produit cartésien est restreint par une combinaison de prédicats de comparaison.

Il est aussi possible de privilégier une table.

11

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

Exemple :

```
CREATE TABLE Livre
(auteur CHAR (8),
titre CHAR (24),
année SMALLINT,
genre CHAR (8),
prix DECIMAL (5,2),
PRIMARY KEY (auteur, titre));
```

```
CREATE TABLE Ecrivain
(auteur CHAR (8),
né_en SMALLINT,
lieu CHAR (20),
rayon SMALLINT,
PRIMARY KEY (auteur));
```

```
CREATE TABLE Rayon
(rayon SMALLINT,
salle SMALLINT,
PRIMARY KEY (rayon));
```

12

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

```
SELECT A.auteur, B.lieu FROM Livre A, Ecrivain B;
```

forme le produit cartésien des tables Livre et Ecrivain sur les attributs auteur de Livre et lieu de Ecrivain. On peut associer un nom abrégé (alias) aux noms de table pour alléger l'écriture des requêtes.

```
SELECT A.auteur, A.titre, B.rayon FROM Livre A, Ecrivain B WHERE A.auteur = B.auteur;
```

Equi-jointure : liste le titre, l'auteur et le rayon de rangement des ouvrages de la bibliothèque.

13

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

Exemple (suite) :

```
CREATE TABLE Livre
(auteur CHAR (8),
 titre CHAR (24),
 année SMALLINT,
 genre CHAR (8),
 prix DECIMAL (5,2),
 PRIMARY KEY (auteur, titre));
```

```
CREATE TABLE Ecrivain
(auteur CHAR (8),
 né_en SMALLINT,
 lieu CHAR (20),
 rayon SMALLINT,
 PRIMARY KEY (auteur));
```

```
CREATE TABLE Rayon
(rayon SMALLINT,
 salle SMALLINT,
 PRIMARY KEY (rayon));
```

```
CREATE TABLE Ouvrage
(auteur CHAR (8),
 titre CHAR (24),
 né_en SMALLINT,
 salle SMALLINT,
 rayon SMALLINT);
```

14

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

```
SELECT X.auteur, X.titre, Y.né_en, Z.salle, Z.rayon FROM Livre X, Ecrivain Y, Rayon Z WHERE X.auteur = Y.auteur AND Y.rayon = Z.rayon;
```

reconstitue la table OUVRAGE à partir des tables LIVRE, ECRIVAIN et RAYON.

```
SELECT A.auteur, A.titre, A.année, A.genre, A.prix, B.né_en, B.lieu, B.rayon FROM Livre A, Ecrivain B WHERE A.auteur = B.auteur;
```

réalise une **jointure naturelle** (les attributs de jointure ne sont pas inclus dans la relation résultante).

```
SELECT A.auteur, A.titre, A.né_en, B.auteur FROM Ecrivain A, Ecrivain B WHERE A.né_en = B.né_en AND A.auteur > B.auteur;
```

jointure d'une table sur elle-même : liste le nom, la date de naissance des écrivains nés la même année.

15

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

```
SELECT A.rayon B.auteur FROM Rayon A, Ecrivain B WHERE A.rayon = B.rayon (+);
```

réalise une jointure dans laquelle la table RAYON est **privilegiée**. Cette jointure affiche la liste de tous les rayons de la bibliothèque et les noms d'auteurs qui y sont rangés si les rayons ne sont pas vides.

16

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

Utilisation de sous-questions

On peut imbriquer des sous-questions au niveau de la clause `WHERE`.

Le résultat d'une sous-question peut être soit une valeur simple, soit un ensemble de valeurs.

Le résultat d'une sous-question est exploité pour :

- tester l'appartenance d'une valeur à une liste de valeurs élaborée dans la sous-question
- vérifier un prédicat de comparaison exprimé à l'aide des quantificateurs de la logique du premier ordre et d'opérateurs de comparaison (`=`, `>`, `<`, `>=`, `<=`).

17

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

```
SELECT auteur, lieu FROM Ecrivain
WHERE auteur IN (SELECT auteur FROM Livre);
```

liste le nom et le lieu de naissance des auteurs de la table `AUTEUR` figurant dans la table `LIVRE`.

Cette requête est équivalente à la requête suivante utilisant une jointure :

```
SELECT auteur, lieu FROM Ecrivain, Livre
WHERE Auteur.auteur = Livre.auteur;
```

Il est possible d'utiliser des variables définies dans un bloc interne au niveau d'un bloc externe. On parle alors de *variable de corrélation*.

La requête suivante liste le nom, le lieu de naissance et le genre des livres écrits par des auteurs de la table `AUTEUR` figurant dans la table `LIVRE`.

```
SELECT auteur, lieu, A.genre FROM Ecrivain WHERE auteur IN (SELECT
auteur FROM Livre A);
```

18

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

Questions quantifiées

On peut comparer le (ou les) attribut(s) se trouvant dans la clause `WHERE` avec l'ensemble de valeurs résultat d'une sous-question à l'aide de quantificateurs.

Le prédicat de comparaison est vrai :

- s'il est vérifié pour tous les éléments de l'ensemble avec le quantificateur `ALL`,
- s'il est vérifié pour au moins un élément de l'ensemble avec le quantificateur `SOME` (ou `ANY`, synonyme).

19

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

```
SELECT titre FROM Livre
WHERE année > ALL (SELECT né_en FROM Ecrivain);
```

liste le titre des ouvrages de la table `LIVRE` écrits après la naissance de tous les auteurs de la table `ECRIVAIN`.

```
SELECT auteur FROM Ecrivain
WHERE Auteur = SOME (SELECT auteur FROM Livre);
```

liste les auteurs de la table `ECRIVAIN` ayant un livre dans la table `LIVRE`.

Remarque : `SOME` est équivalent à `IN`.

20

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

On peut tester si le résultat d'une sous-question est vide ou non. Pour cela, on utilise le quantificateur EXISTS.

EXISTS <sous-question> s'évalue à vrai si le résultat de la sous-question est non vide.

```
SELECT auteur, titre FROM Livre X
WHERE EXISTS (SELECT * FROM Ecrivain WHERE né_en = X.année);
```

liste le nom des auteurs et le titre des ouvrages de la table LIVRE écrits la même année que l'année de naissance d'un des auteurs de la table ECRIVAIN.

21

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

Expression des unions

```
SELECT année, 'parution ', auteur FROM Livre WHERE année >=1825 AND
année <= 1850
UNION
SELECT né_en, 'naissance', auteur FROM Ecrivain WHERE né_en >= 1825
AND né_en <= 1850 ORDER BY 1;
```

liste par ordre chronologique les événements intervenus entre 1825 et 1850 figurant dans la table LIVRE (parution) et dans la table ECRIVAIN.

Remarque : les tables résultantes des selects doivent avoir le même nombre de colonnes et deux colonnes de même rang doivent être de même nature (chaînes de caractères, données numériques ou temporelles).

22

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

Agrégations et partitionnements

Toutes les requêtes vues jusqu'à présent pouvaient être considérées comme une suite d'opérations effectuées *tuple à tuple*.

Les fonctionnalités d'*agrégation* de SQL permettent d'exprimer des conditions *sur des groupes de tuples*, et de constituer les résultat par *agrégation de valeurs* au sein de chaque groupe.

La syntaxe SQL fournit donc :

- Le moyen de **partitionner une relation en groupes** selon certains critères.
- Le moyen d'**exprimer des conditions sur ces groupes**.
- Des **fonctions d'agrégation**.

Il existe un groupe par défaut : le relation toute entière. Sans même définir de groupe donc, on peut utiliser les fonctions d'agrégation.

23

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

Fonctions d'agrégation

Ces fonctions s'appliquent à une colonne, en général de type numérique. Ce sont :

- COUNT qui compte le nombre de valeurs *non nulles*.
- MAX et MIN.
- AVG, STDDEV, VARIANCE qui calculent la moyenne, la déviation standard et la variance des valeurs de la colonne.
- SUM qui effectue le cumul.

```
SELECT COUNT nomStation AVG(tarif), MIN(tarif), MAX(tarif)
FROM Station;
```

compte le nombre de stations, les tarifs moyen, minimum et maximum de la table Station.

24

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

Partitionnement

Le partitionnement s'exprime avec l'option GROUP BY

```
SELECT auteur, genre, COUNT(titre)
FROM Livre GROUP BY auteur, genre;
```

liste en les regroupant par auteur et genre, les attributs auteur et genre de la table Livre.

Le résultat pourrait être (autant de groupes que de couples (auteur, genre)) :

No Partition	auteur	genre	count
1	Balzac	Roman	1
2	Dumas	Roman	1
3	Dumas	Théâtre	1
4	Hugo	Poésie	2
5	Hugo	Roman	1

25

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

Il est possible d'effectuer des restrictions sur le résultat du partitionnement i.e. écarter des sous-ensembles. Pour cela, on utilise l'option HAVING.

```
SELECT auteur, genre FROM Livre
GROUP BY auteur, genre HAVING genre <> 'Poésie';
```

liste, en les regroupant par auteur et genre, les attributs auteur et genre en éliminant les groupes dont le genre est "poésie".

Dans l'exemple ci-dessus, élimination du sous-ensemble 4.

26

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

```
SELECT auteur, 'genre :', genre FROM Livre
GROUP BY année <> 1834 HAVING genre <> 'Théâtre'
ORDER BY genre ASC, auteur DESC;
```

liste l'auteur, le libellé "genre" et le genre des ouvrages non publiés en 1834, en les regroupant par genre et par auteur, en éliminant les pièces de théâtre et en triant le résultat par genre croissant et par auteur décroissant.

Remarque : l'option WHERE permet de sélectionner les tuples avant la formation des partitions. L'option HAVING est évaluée ensuite. Elle permet d'effectuer une restriction sur les partitions.

27

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

Mise à jour de tuples

Sélection avec intention de mise à jour

```
SELECT salle, rayon
FROM Ouvrage
WHERE auteur IN ('Hugo', 'Malouf')
FOR UPDATE;
```

...

```
UPDATE Ouvrage
SET salle = 2, rayon = 3
WHERE auteur IN
(SELECT auteur FROM Ecrivain WHERE lieu = 'Paris');
```

met à jour la table OUVRAGE en affectant les valeurs 2 pour la salle et 3 pour le rayon pour les auteurs nés à Paris.

28

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

Suppression de tuples

Suppression avec sous-question :

```
DELETE FROM Ouvrage
WHERE auteur IN
(SELECT auteur FROM Ecrivain WHERE lieu = 'Paris');
```

supprime de la table OUVRAGE les lignes correspondant à des auteurs pour les auteurs nés à Paris.

29

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

L'opérateur de division en SQL

Sur l'exemple de la base aérienne, quels sont les pilotes qui pilotent tous les Airbus ?

```
Avion(#av_No, AvNom, ...)
Pilote(#PLNo, ...)
Vol(#VolNo, AvNo, PLNo, ...)
```

```
SELECT DISTINCT PLNo FROM Vol
WHERE av_No IN
(SELECT AvNo FROM Avion WHERE AvNom = 'Airbus')
GROUP BY PLNo
HAVING COUNT (DISTINCT av_No) =
(SELECT COUNT (av_No) FROM Avion WHERE AvNom = 'Airbus');
```

30

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

L'opérateur d'intersection en SQL

On crée d'abord deux tables Livre1 et Livre2 :

```
INSERT INTO Livre1 (SELECT * FROM Livre WHERE année < 1850);
INSERT INTO Livre2 (SELECT * FROM Livre WHERE année >= 1850);
```

```
SELECT A.auteur FROM Livre1 A
WHERE EXISTS
(SELECT B.auteur FROM Livre2 B WHERE A.auteur = B.auteur);
```

liste les auteurs ayant écrit un livre avant 1850 et un livre après 1850.

```
SELECT A.auteur FROM Livre1 A
WHERE NOT EXISTS
(SELECT B.auteur FROM Livre2 B WHERE A.auteur = B.auteur);
```

liste les auteurs ayant écrit un livre avant 1850 et pas après 1850.

31

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

Les vues en SQL

Une vue est une *table virtuelle* qui résulte d'une requête. Ce résultat est ré-évalué à chaque fois que l'on accède à la vue. Une vue est donc *essentiellement une requête à laquelle on a donné un nom*.

```
CREATE VIEW <nom_vue>
AS <requête>
[WITH CHECK OPTION];
```

```
CREATE VIEW ParisCinemas
AS SELECT * FROM Cinema WHERE ville = 'Paris';
```

crée une vue qui ne "contient" que les cinémas parisiens.

32

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

Les vues en SQL

```
CREATE VIEW SimpleParisCinemas
AS SELECT nom, COUNT(*) AS nbSalles
FROM Cinema C, salle S
WHERE ville = 'Paris'
AND C.nom = S.nomCinema
GROUP BY C.nom;
```

restreint la vision des cinémas parisiens à leur nom et à leur nombre de salles

33

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

On peut créer une vue qui regroupe des informations par jointures.

Exemple : on crée une vue `Casting` donnant explicitement les titres des films, leur année et les noms et prénoms des acteurs.

```
CREATE VIEW Casting (film, année, acteur, prenom) AS
SELECT Titre, Annee, Nom, Prenom
FROM Film F, Role R, Artiste A
WHERE F.idFilm = R.idFilm
AND R.idActeur = A.idArtiste;
```

Remarque : on a donné explicitement des noms d'attributs au lieu d'utiliser les attributs de la clause `SELECT`.

```
SELECT acteur, prenom FROM Casting WHERE année = 1997;
```

```
GRANT SELECT ON Casting TO PUBLIC;
```

On peut donner des droits en lecture pour tous sur cette vue.

34

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

Les triggers

Un *trigger* (non évoqué dans SQL2, mais discuté dans SQL3) est une procédure déclenchée par des événements de mise à jour *spécifiés par l'utilisateur* et ne s'exécute que quand une condition est satisfaite.

Les *triggers* peuvent être considérés comme une extension du système de contraintes proposée par la clause `CHECK`. mais à la différence de cette dernière, l'événement déclencheur est explicitement indiqué et l'action n'est pas limitée à la simple alternative acceptation/rejet.

35

Le modèle relationnel

Langage de Manipulation de Données SQL3 (99)

Les triggers

```
CREATE TRIGGER CumulCapacite
AFTER UPDATE ON Salle
FOR EACH ROW
WHEN (new.capacite != old.capacite)
BEGIN
UPDATE Cinema
SET capacite = capacite - :old.capacite + :new.capacite
WHERE nom = :new.nomCinema;
END;
```

Exemple de *trigger* qui maintient la capacité d'un cinéma à chaque mise-à-jour sur la table `Salle`.

36