

Bases de données avancées

Du relationnel à l'objet-relationnel, persistance objet, bases de données orientées objets et bases de données distribuées

SELMANE Samir

Département Informatique — Mila

UEF1 · 6 crédits · Coefficient 3

Plan du module

N°	Chapitre	Sections	Poids
1	Rappel sur les bases de données relationnelles	Modèle, algèbre, normalisation, SQL	10 %
2	Le modèle objet-relationnel et SQL3	Modèle OR · SQL3	25 %
3	Persistance des données dans les applications objet	Mapping semi-transparent · transparent	25 %
4	Bases de données orientées objets	Manifeste, ODMG, ODL, OQL	20 %
5	Bases de données distribuées	Fragmentation, catalogue · Mise à jour	20 %

Volume hebdomadaire : Cours 1h30 · TD 1h30 · TP 1h30

Modalités d'évaluation

60 %

Examen final

40 %

Contrôle continu

Détail du contrôle continu (sur 40)

Élément	Points
Assiduité	2
Participation	4
Interrogation (1)	7
Travaux pratiques	20
Exposés	7
Total	40

01

Rappel sur les bases de données relationnelles

10 % du module · Modèle relationnel, algèbre, normalisation, SQL et limites du modèle

Le modèle relationnel

Relation : table à deux dimensions, sous-ensemble du produit cartésien de domaines.

Attribut / domaine : colonne nommée et ensemble de ses valeurs atomiques.

Tuple : une ligne de la relation.

Clé primaire : attribut(s) qui identifient un tuple de façon unique.

Clé étrangère : attribut qui référence la clé primaire d'une autre relation.

Schéma

ETUDIANT(NumEt, Nom, Ville, #CodeSpec)

SPECIALITE(CodeSpec, Libelle)

NumEt	Nom	Ville	CodeSpec
101	Amine	Mila	STIC
102	Sara	Constantine	STIC
103	Yacine	Mila	RSD

L'algèbre relationnelle

Opérateur	Symbole	Rôle	Exemple
Sélection	σ	garde les tuples qui vérifient une condition	$\sigma \text{ Ville='Mila' (ETUDIANT)}$
Projection	π	garde certains attributs	$\pi \text{ Nom (ETUDIANT)}$
Produit cartésien	$\times$	combine chaque tuple de R avec chaque tuple de S	$\text{ETUDIANT} \times \text{SPECIALITE}$
Jointure	$\bowtie$	produit suivi d'une sélection	$\text{ETUDIANT} \bowtie \text{SPECIALITE}$
Union	$\cup$	tuples de R ou de S (schémas compatibles)	$\text{ETU_MILA} \cup \text{ETU_CNE}$
Différence	$-$	tuples de R absents de S	$\text{INSCRITS} - \text{ADMIS}$
Intersection	$\cap$	tuples communs à R et S	$\text{INSCRITS} \cap \text{BOURSIERS}$
Division	$\div$	« pour tous » : liés à tous les tuples de S	$\text{INSCRIPTION} \div \text{MODULE}$

Dépendances fonctionnelles et normalisation

Dépendance fonctionnelle $X \rightarrow Y$: deux tuples qui ont la même valeur de X ont la même valeur de Y. La normalisation décompose une relation pour éliminer la redondance et les anomalies de mise à jour.

1FN

Tous les attributs sont atomiques : pas de listes ni de groupes répétitifs.

2FN

1FN, et tout attribut non clé dépend de toute la clé, pas d'une partie.

3FN

2FN, et aucun attribut non clé ne dépend d'un autre attribut non clé.

BCNF

Toute DF non triviale $X \rightarrow Y$ a pour source X une clé candidate.

À retenir : la 1FN interdit les valeurs structurées. C'est cette contrainte que le modèle objet-relationnel va lever (chapitre 2).

Exemple : première forme normale (1FN)

Règle : chaque attribut contient une seule valeur atomique. Pas de liste, pas de groupe répétitif.

Avant : pas en 1FN

NumEt	Nom	Telephones
101	Amine	0555 11 22, 0661 33 44
102	Sara	0770 55 66

Problème : impossible de chercher ou modifier un numéro sans découper la chaîne.



Après : en 1FN

ETUDIANT (NumEt, Nom)
TELEPHONE (#NumEt, Tel)

NumEt	Tel
101	0555 11 22
101	0661 33 44
102	0770 55 66

Exemple : deuxième forme normale (2FN)

Règle : 1FN, et aucun attribut non clé ne dépend d'une **partie** seulement de la clé.

Avant : pas en 2FN

INSCRIPTION(NumEt, CodeMod, Note, TitreMod)

DF : NumEt, CodeMod $\rightarrow$ Note

DF partielle : CodeMod $\rightarrow$ TitreMod

Problème : le titre d'un module est répété pour chaque étudiant inscrit. Le changer oblige à modifier toutes ces lignes.



Après : en 2FN

INSCRIPTION(#NumEt, #CodeMod, Note)

MODULE(CodeMod, TitreMod)

On sort l'attribut qui dépend d'une partie de la clé dans une nouvelle relation, avec cette partie comme clé.

Remarque : une relation dont la clé a un seul attribut est toujours en 2FN si elle est en 1FN.

Exemple : troisième forme normale (3FN)

Règle : 2FN, et aucun attribut non clé ne dépend d'un autre attribut non clé (pas de DF transitive).

Avant : pas en 3FN

ETUDIANT(NumEt, Nom, CodeSpec, LibelleSpec)

DF : NumEt $\rightarrow$ CodeSpec

DF transitive : CodeSpec $\rightarrow$ LibelleSpec

Problèmes : libellé répété ; on ne peut pas créer une spécialité sans étudiant ; supprimer le dernier étudiant fait perdre le libellé.



Après : en 3FN

ETUDIANT(NumEt, Nom, #CodeSpec)

SPECIALITE(CodeSpec, LibelleSpec)

L'attribut intermédiaire (CodeSpec) devient la clé d'une nouvelle relation et reste clé étrangère dans ETUDIANT.

La décomposition est sans perte : ETUDIANT $\bowtie$ SPECIALITE redonne la relation de départ.

Exemple : forme normale de Boyce-Codd (BCNF)

Règle : pour toute DF non triviale $X \rightarrow Y$, X est une clé candidate. Hypothèse : chaque enseignant enseigne un seul module.

Avant : en 3FN, pas en BCNF

SUIVI(NumEt, Module, Enseignant)

DF : NumEt, Module $\rightarrow$ Enseignant

DF : Enseignant $\rightarrow$ Module

3FN car Module fait partie de la clé. Mais Enseignant n'est pas une clé : le module de chaque enseignant est répété.



Après : en BCNF

ENSEIGNANT(Enseignant, Module)

SUIVI(NumEt, #Enseignant)

Décomposition sans perte d'information.

Prix à payer : la DF « NumEt, Module $\rightarrow$ Enseignant » n'est plus vérifiable dans une seule relation. La BCNF ne préserve pas toujours les DF ; la 3FN, si.

SQL et transactions

LDD : CREATE, ALTER, DROP (schéma)

LMD : SELECT, INSERT, UPDATE, DELETE (données)

LCD : GRANT, REVOKE (droits)

Transactions : COMMIT, ROLLBACK

A

Atomicité

C

Cohérence

I

Isolation

D

Durabilité

```
-- Étudiants admis avec leur spécialité
SELECT e.Nom, s.Libelle
FROM   Etudiant e
JOIN   Specialite s
      ON e.CodeSpec = s.CodeSpec
WHERE  e.Moyenne >= 10
ORDER BY e.Nom;
```

```
-- Nombre d'étudiants par spécialité
SELECT CodeSpec, COUNT(*)
FROM   Etudiant
GROUP BY CodeSpec
HAVING COUNT(*) > 20;
```

LDD : définir le schéma

```
CREATE TABLE Etudiant (  
  NumEt      INT PRIMARY KEY,  
  Nom        VARCHAR(30) NOT NULL,  
  Ville      VARCHAR(30) DEFAULT 'Mila',  
  Moyenne    DECIMAL(4,2)  
              CHECK (Moyenne BETWEEN 0 AND 20),  
  CodeSpec   VARCHAR(10)  
              REFERENCES Specialite(CodeSpec)  
);  
ALTER TABLE Etudiant ADD Email VARCHAR(50) UNIQUE;  
ALTER TABLE Etudiant DROP COLUMN Email;  
CREATE INDEX idx_nom ON Etudiant(Nom);  
CREATE VIEW Admis AS  
  SELECT NumEt, Nom FROM Etudiant WHERE Moyenne >= 10;  
DROP TABLE Etudiant;
```

CREATE : table, vue, index, séquence

ALTER : ajouter, modifier ou supprimer une colonne ou une contrainte

DROP : supprimer un objet et ses données

Contraintes : PRIMARY KEY, NOT NULL, UNIQUE, DEFAULT, CHECK, FOREIGN KEY / REFERENCES

ON DELETE CASCADE ou SET NULL fixe l'effet de la suppression d'une ligne référencée.

LMD : insérer, modifier, supprimer

```
INSERT INTO Etudiant (NumEt, Nom, Ville, CodeSpec)
VALUES (104, 'Lina', 'Mila', 'STIC');
INSERT INTO Archive SELECT * FROM Etudiant WHERE Moyenne < 5;
```

INSERT : une ligne avec VALUES, ou plusieurs lignes issues d'un SELECT.

```
UPDATE Etudiant
SET     Moyenne = Moyenne + 0.5, Ville = UPPER(Ville)
WHERE   CodeSpec = 'STIC';
```

UPDATE : modifie les colonnes des lignes qui vérifient WHERE.

```
DELETE FROM Etudiant WHERE NumEt = 104;
DELETE FROM Inscription
WHERE   NumEt NOT IN (SELECT NumEt FROM Etudiant);
```

DELETE : supprime des lignes. Sans WHERE, la table est vidée.

LCD et contrôle des transactions

Droits d'accès (LCD)

```
GRANT SELECT, UPDATE (Moyenne)
  ON Etudiant TO scholarite;
GRANT SELECT ON Admis TO PUBLIC;
REVOKE UPDATE ON Etudiant
  FROM scholarite;
CREATE ROLE enseignant;
GRANT enseignant TO ali;
```

WITH GRANT OPTION permet de transmettre le droit reçu.

Transactions (TCL)

```
START TRANSACTION;
UPDATE Compte SET Solde = Solde - 500
WHERE Id = 1;
SAVEPOINT s1;
UPDATE Compte SET Solde = Solde + 500
WHERE Id = 2;
COMMIT; -- ou ROLLBACK [TO s1]
```

COMMIT rend les changements durables ; ROLLBACK les annule.

Structure de la requête SELECT

- ⑤ SELECT [DISTINCT] colonnes, expressions
- ① FROM tables [JOIN ... ON ...]
- ② WHERE condition sur les lignes
- ③ GROUP BY colonnes de regroupement
- ④ HAVING condition sur les groupes
- ⑥ ORDER BY colonnes [ASC | DESC]
- ⑦ FETCH FIRST n ROWS ONLY -- ou LIMIT n

Seuls SELECT et FROM sont obligatoires.

Les numéros donnent l'**ordre d'évaluation** logique, différent de l'ordre d'écriture.

Conséquence : un alias défini dans SELECT n'est pas utilisable dans WHERE, mais l'est dans ORDER BY.

Schéma utilisé dans les exemples

ETUDIANT(NumEt, Nom, Ville, Moyenne, #CodeSpec) · SPECIALITE(CodeSpec, Libelle)
MODULE(CodeMod, Titre, Coef) · INSCRIPTION(#NumEt, #CodeMod, Note)

SELECT : projection, DISTINCT, WHERE, ORDER BY

Élément	Exemple	Sens
DISTINCT	SELECT DISTINCT Ville FROM Etudiant;	supprime les doublons
Alias, calcul	SELECT Nom, Moyenne * 5 AS Sur100 FROM Etudiant;	colonne calculée renommée
Comparaison	WHERE Moyenne >= 10 AND Ville <> 'Mila'	=, <>, <, >, AND, OR, NOT
BETWEEN	WHERE Moyenne BETWEEN 10 AND 12	intervalle, bornes incluses
IN	WHERE Ville IN ('Mila', 'Jijel')	appartenance à une liste
LIKE	WHERE Nom LIKE 'A%' OR Nom LIKE '_ara'	% : suite de caractères ; _ : un caractère
IS NULL	WHERE Moyenne IS NULL	jamais « = NULL »
ORDER BY	ORDER BY Moyenne DESC, Nom ASC	tri sur plusieurs clés

SELECT : les jointures

Type	Exemple	Résultat
Interne	FROM Etudiant e JOIN Specialite s ON e.CodeSpec = s.CodeSpec	lignes qui ont une correspondance
Syntaxe classique	FROM Etudiant e, Specialite s WHERE e.CodeSpec = s.CodeSpec	même résultat
Externe gauche	FROM Specialite s LEFT JOIN Etudiant e ON e.CodeSpec = s.CodeSpec	toutes les spécialités, NULL si pas d'étudiant
Droite, complète	RIGHT JOIN ... ON ... / FULL JOIN ... ON ...	garde la table de droite / les deux
Naturelle	FROM Etudiant NATURAL JOIN Specialite	sur les colonnes de même nom
Auto-jointure	FROM Etudiant a JOIN Etudiant b ON a.Ville = b.Ville AND a.NumEt < b.NumEt	paires d'étudiants de la même ville
Produit	FROM Etudiant CROSS JOIN Module	toutes les combinaisons

SELECT : agrégats, GROUP BY et HAVING

```
-- Agrégats sur toute la table
SELECT COUNT(*), AVG(Moyenne), MAX(Moyenne)
FROM Etudiant;

-- Moyenne des notes par module
SELECT m.Titre, AVG(i.Note) AS MoyNote,
       COUNT(*) AS NbInscrits
FROM Inscription i
JOIN Module m ON i.CodeMod = m.CodeMod
WHERE i.Note IS NOT NULL
GROUP BY m.Titre
HAVING COUNT(*) >= 10
ORDER BY MoyNote DESC;
```

Fonctions : COUNT, SUM, AVG, MIN, MAX

GROUP BY : une ligne de résultat par groupe

WHERE filtre les lignes *avant* le regroupement ;

HAVING filtre les groupes *après*

Toute colonne du SELECT hors agrégat doit être dans GROUP BY

COUNT(*) compte les lignes ; COUNT(col) ignore les NULL ;
COUNT(DISTINCT col) compte les valeurs différentes.

SELECT : sous-requêtes

```
-- Scalaire : au-dessus de la moyenne
SELECT Nom FROM Etudiant
WHERE Moyenne > (SELECT AVG(Moyenne)
                  FROM Etudiant);

-- IN : inscrits au module BDA
SELECT Nom FROM Etudiant
WHERE NumEt IN (SELECT NumEt
                FROM Inscription WHERE CodeMod = 'BDA');

-- ALL : meilleure moyenne
SELECT Nom FROM Etudiant
WHERE Moyenne >= ALL
      (SELECT Moyenne FROM Etudiant);
```

```
-- Corrélée : étudiants sans inscription
SELECT e.Nom FROM Etudiant e
WHERE NOT EXISTS (
    SELECT * FROM Inscription i
    WHERE i.NumEt = e.NumEt);

-- Division : inscrits à TOUS les modules
SELECT e.Nom FROM Etudiant e
WHERE NOT EXISTS (
    SELECT * FROM Module m
    WHERE NOT EXISTS (
        SELECT * FROM Inscription i
        WHERE i.NumEt = e.NumEt
              AND i.CodeMod = m.CodeMod));
```

SELECT : opérateurs ensemblistes et CASE

Opérateur	Exemple	Sens
UNION	SELECT Ville FROM Etudiant UNION SELECT Ville FROM Enseignant	sans doublons (UNION ALL les garde)
INTERSECT	SELECT NumEt FROM Inscription WHERE CodeMod = 'BDA' INTERSECT SELECT NumEt FROM Inscription WHERE CodeMod = 'RES'	inscrits aux deux modules
EXCEPT (MINUS)	SELECT NumEt FROM Etudiant EXCEPT SELECT NumEt FROM Inscription	étudiants sans inscription
CASE	SELECT Nom, CASE WHEN Moyenne >= 10 THEN 'Admis' ELSE 'Ajourné' END AS Resultat FROM Etudiant	valeur conditionnelle
Limitation	ORDER BY Moyenne DESC FETCH FIRST 3 ROWS ONLY	les 3 premiers (LIMIT 3 en MySQL)

Les deux requêtes d'un opérateur ensembliste doivent avoir le même nombre de colonnes, de types compatibles.

Limites du modèle relationnel

Types simples uniquement

La 1FN interdit les attributs structurés ou multivalués (adresse, liste de téléphones).

Pas d'identité d'objet

Un tuple est identifié par la valeur de sa clé, pas par un identifiant indépendant.

Ni héritage ni comportement

Pas de hiérarchie de types et pas de méthodes attachées aux données.

Défaut d'impédance

Les programmes manipulent des objets, la base stocke des tables : il faut convertir.

Réponses étudiées : modèle objet-relationnel (ch. 2), mapping objet-relationnel (ch. 3), bases de données orientées objets (ch. 4).