MIAGE-BDAV – Bases de données avancées

Rappel du modèle relationnel et de SQL

THION Romuald

https://romulusfr.github.io/unc-miage-bdav/

Lundi 24 janvier 2022

Plan

Système de Gestion de Bases de Données

SGBD : ensemble d'outils logiciels permettant la création et l'utilisation de bases de données (relationnelles ou pas).
Modèle relationnel : *toutes* les données sont organisées en relations

Le concepteur définit la structure

Il modélise le problème puis traduit sa modélisation dans le modèle relationnel sous forme de tables et de contraintes sur celles-ci.

Le SGBD gère le reste

- la persistance et la journalisation
- la disponibilité
- la concurrence des accès
- la sécurité

Système de Gestion de Bases de Données

Données structurées : le schéma

Défini via le *Langage de Description de Données* (LDD) de SQL

- organisation et type des données
- contraintes d'intégrité
- *e.g.*, `CREATE DATABASE`, `CREATE TABLE`, `ALTER TABLE`, `CREATE INDEX`

Langage déclaratif : manipulation de données

Exprimée via le *Langage de Manipulation de Données* (LMD) de SQL

- Recherche, création, modification et suppression
- On spécifie ce que l'on veut faire pas comment le faire
- *e.g.*, `SELECT ...FROM WHERE`, `INSERT`, `UPDATE`, `DELETE`

Système de Gestion de Bases de Données

Indépendance logique/physique

- Séparation entre le quoi et le comment : principe fondateur du modèle relationnel
- Toutes les interactions passent via la couche externe du SGBD

Interaction avec le SGBD

- Interpréteur ligne de commandes (e.g., `sqlite3`, `psql`)
- IDE (e.g., DBeaver, sqlitebrowser)
- Dans une langage de programmation via les bibliothèques
  - C, C++, Java, Python, PHP, JavaScript . . .
  - e.g., `https://docs.python.org/3/library/sqlite3.html`

Modèle relationnel

Modèle ensembliste

- Les objets sont simples, atomiques :
  - entier, flottants, chaînes de caractères, dates,
  - mais pas de listes, tableaux, structures...
  - ...mais en pratique les SBGDs supportent ces structures
- On utilise des relations (au sens de la théorie des ensembles) pour définir les données et uniquement des relations.
- On se dotte des opérations ensemblistes usuelles :
  - Union ($\cup$), intersection ($\cap$), différence ($-$), produit ($\times$)
- On se dotte d'opérations typiques du modèle relationnel :
  - Renommage (ρ), sélection (σ), jointure ($\bowtie$), jointure ouverte ($\ltimes$, $\rtimes$)

Jeu de données *Stanford*

Apply

sID	cName	major	dec.
123	Stanford	CS	Y
123	Stanford	EE	N
123	Berkeley	CS	Y
123	Cornell	EE	Y
234	Berkeley	biology	N
345	MIT	bioeng.	Y
345	Cornell	bioeng.	N
345	Cornell	CS	Y
345	Cornell	EE	N
678	Stanford	history	Y

Apply

sID	cName	major	dec.
987	Stanford	CS	Y
987	Berkeley	CS	Y
876	Stanford	CS	N
876	MIT	biology	Y
876	MIT	marine bio.	N
765	Stanford	history	Y
765	Cornell	history	N
765	Cornell	psych.	Y
543	MIT	CS	N

Schéma de base de données

Les attributs

- Décrit les données atomiques que l'on veut manipuler, par exemple `titre`, `annee`, `genre`
- Les attributs sont (souvent) typés, par exemple `titre:text`, `annee:int4`

Les relations (entre les attributs)

- Représente les liens entre les données atomiques, par exemple `Student(sID, sName, GPA, sizeHS)`
- Arité d'une relation: nombre d'attributs de cette relation

Schéma de base de données

- L'ensemble des relations composent le schéma, comme par exemple `Student`, `College` et `Apply`

Instances

Instances de relations et de bases de données

- Une instance d'une relation $R(A_1, \ldots, A_n)$ est un sous-ensemble fini du produit cartésien des domaines de ses attributs : un ensemble *fini* de tuples.
- Une instance d'une base de données est un ensemble d'instances, une pour chaque relation du schéma.

En pratique, dans les SGBD on appelle souvent les relations (et leurs instances) des tables.

Théorie VS pratique

Dans un modèle ensembliste, théoriquement

- Les valeurs des attributs sont toujours définieset uniques
- Pas de doublons ($\{x, x\} = \{x\}$)
- L'ordre des tuples n'est pas important ($\{x, y\} = \{y, x\}$)

En pratique, les choses sont différentes

- Possibilité de valeur non définie (logique tri-valuée) (`NULL`)
- Possibilités de doublons (`DISTINCT`)
- Possibilité d'ordonner le résultat des requêtes (`ORDER BY`)

Standards ISO/IEC 9075

- SQL-86 : 1 version
- SQL-89 : contraintes d'intégrité
- SQL-1992 (SQL-2) : triggers, CTE
- SQL-1999 (SQL-3) : RE, types non scalaires
- SQL-2003 : ajout SQL/XML, *windows functions*
- SQL-2006 : XML (suite)
- SQL-2008 : `FETCH`
- SQL-2011
- SQL-2016 : JSON
- SQL-2019

Langage SQL – SELECT

Structured Query Language : plusieurs facettes

- des opérations de définition de données
- des opérations de manipulation de données
- et aussi pour la sécurité, les sessions, les transactions, ...

Strutcture générale d'une requête SQL de sélection

```
SELECT [DISTINCT] att1, att2, *, ...
FROM table1 [JOIN table2 ON condition]...
WHERE condition
GROUP BY att1,
HAVING condition
ORDER BY att1 (DESC/ASC), att2 (DESC/ASC)...
LIMIT nombre OFFSET decalage
```

Langage SQL – SELECT

Remarque importante

Les syntaxes SQL données ici sont *partielles* : toujours se référer à la documentation de référence du SGBD utilisé :

- SQLite3 https://www.sqlite.org/lang.html
- PostgreSQL https://www.postgresql.org/docs/current/sql.html

SELECT

Prototype

```
SELECT att1,att2...
FROM MaTable;
```

SELECT : la projection

- Récupérer les valeurs de `MaTable` en ne gardant que les attributs `att1,att2...`
- On peut remplacer `att1,att2...` par `*` pour utiliser tous les attributs
- Le point virgule `;` est utilisé pour marquer la fin des commandes SQL (†)

SELECT

Noms et moyennes des étudiants

```
SELECT sName, GPA
FROM Student;
```

L'intégralité des étudiants

```
SELECT *
FROM Student;
```

WHERE

Prototype

```
SELECT att1 , att2 ...
FROM MaTable
WHERE condition ;
```

WHERE : la sélection

- La clause WHERE spécifie les lignes à sélectionner grâce à la condition
- Spécification des conditions :
  - Opérateurs booleéns, fonctions, prédicats
  - Nombreuses primitives fournies par les SGBDs

WHERE

Expression des conditions

- Comparaisons (`=`, `!=`, `<`, `<=`, `>`, `>=`)
- Entre un attribut et une constante ou un autre attribut
- Différents types de données pour les constantes :
  - nombres: `1`, `1980`, `1.5`
  - chaînes de caractères: `'Martin'`,`'directeur'`
  - dates: `'1980-06-18'`
- Le formattage des date peut varier d'un SGBD à l'autre

Combinaison d'expressions

- Le et logique ($\wedge$) : `AND`
- le ou logique ($\vee$) : `OR`
- le non logique ($\neg$) : `NOT`

WHERE

Exemple

```
SELECT sName
FROM Student
WHERE GPA > 3.6;
```

Exemple

```
SELECT sName, GPA
FROM Student
WHERE GPA > 3.6 or GPA < 2.9;
```

WHERE

Opérateur IN

- Syntaxe : *attribut* IN *liste de valeurs*
- Permet d'éviter une répétition de OR

Exemple

```
SELECT cName, state
FROM College
WHERE state IN ('CA', 'NY', 'WY');

SELECT cName, state
FROM College
WHERE state = 'CA' OR state = 'NY' OR state = 'WY';
```

WHERE

Opérateur BETWEEN

- Syntaxe : *attribut* BETWEEN *mininum* AND *maximum*
- Sucre pour les conditions de la forme $l \leq x \leq u$

Exemple

```
SELECT sName, GPA
FROM Student
WHERE GPA BETWEEN 2.0 AND 3.0;

SELECT sName, GPA
FROM Student
WHERE GPA >= 2.0 AND GPA <= 3.0;
```

FROM

Prototype

```
SELECT att1, att2...
FROM MaTable1, MaTable2, MaTable3...
WHERE condition;
```

FROM : le produit cartésien et la jointure

- Il est possible d'utiliser plusieurs tables dans une requête
- Cela correspond à effectuer un produit cartésien entre les différentes tables
- Si un attribut est présent dans plusieurs tables, on doit l'écrire `nom_table.att`

FROM

Exemple

```
SELECT sName, cName
FROM Student, College
WHERE Student.sizeHS >= 2000;
```

sName	cName
Edward	Stanford
Edward	Berkeley
Edward	MIT
Edward	Cornell

sName	cName
Craig	Stanford
Craig	Berkeley
Craig	MIT
Craig	Cornell

FROM

La jointure, une des opérations fondamentales

Traduction de la jointure

- La jointure (de deux relations) est le sous-ensemble du produit cartésien des relations dont on ne garde *que* les tuples qui satisfont la *condition de jointure*
- La jointure est dite *naturelle*, si la condition est l'égalité entre les attributs communs et que les colonnes en doubles sont supprimées
- Plusieurs façon de la traduire en SQL
  - Soit on utilise la clause `WHERE` pour la *condition de jointure*
  - Soit on utilise la clause `JOIN` et ses variantes (`INNER`, `NATURAL`, `LEFT OUTER`, `RIGHT OUTER`, `FULL OUTER`)

FROM

Exemple : les deux écritures sont logiquement équivalentes

```
-- avec produit cartesien et selection
SELECT sName, cName
FROM Student, Apply
WHERE Student.sID = Apply.sID;

-- avec condition de jointure explicite dans le FROM
-- /!\ forme à préférer /!\
SELECT sName, cName
FROM Student JOIN Apply ON Student.sID = Apply.sID;
```

Attention à l'ambiguïté sur le nom des attributs : il y a deux attribus `sID` dans le résultat !

FROM

Attention à la suppression des colonnes identiques avec `NATURAL JOIN` et la clause `USING`.

Exemple : les deux écritures sont logiquement équivalentes

```
-- avec la syntaxe de la jointure naturelle
SELECT sName, cName
FROM Student NATURAL JOIN Apply;

-- avec USING pour les attributs où tester l'égalité
SELECT sName, cName
FROM Student JOIN Apply USING (sID);
```

FROM

Renommage

- On peut renommer/aliaser localement les tables dans le `FROM`
- Indispensable lorsque l'on veut effectuer des jointures ou des produits cartésiens d'une table avec elle-même

Exemple

```
SELECT Student.sID, sName, GPA, Apply.cName, enrollment
FROM Student, College, Apply
WHERE Apply.sID = Student.sID and Apply.cName = College.cName;

-- equivalent mais avec renommage des relations
SELECT S.sID, S.sName, S.GPA, A.cName, C.enrollment
FROM Student S, College C, Apply A
WHERE A.sID = S.sID and A.cName = C.cName;
```

FROM

Exemple d'auto-jointure

```
SELECT S1.sID, S1.sName, S1.GPA, S2.sID, S2.sName, S2.GPA
FROM Student S1, Student S2
WHERE S1.GPA = S2.GPA;
```

S1.sID	S1.sName	S1.GPA	S2.sID	S2.sName	S2.GPA
654	Amy	3,9	123	Amy	3,9
876	Irene	3,9	123	Amy	3,9
456	Doris	3,9	123	Amy	3,9
123	Amy	3,9	123	Amy	3,9
234	Bob	3,6	234	Bob	3,6
		...			

DISTINCT

DISTINCT = suppression des doublons

- Permet d'*éliminer les doublons* dans le résultat
- Permet de se rapprocher du comportement théorique
- Très (trop ?) utilisé en pratique :
- les doublons peuvent avoir du sens !

Exemple

```
SELECT DISTINCT College.state
FROM College
WHERE enrollment > 15000;
```

DISTINCT

Exemple

Donner (seulement) les noms des étudiants qui ont entre 3,4 et 3,5 de moyenne

```
SELECT Student.sName
FROM Student
WHERE GPA BETWEEN 3.4 AND 3.5;
```

```
SELECT DISTINCT Student.sName
FROM Student
WHERE GPA BETWEEN 3.4 AND 3.5;
```

les noms distincts des étudiants

$\neq$

les noms des étudiants distincts ...

ORDER BY

Prototype

```
SELECT att1 , att2 ...
FROM MaTable1 , MaTable2 ...
WHERE condition
ORDER BY att1 , att2 ... ASC | DESC
```

ORDER BY = tri selon

- Très utilisé : il faut (presque) toujours avoir un `ORDER BY` (†)
- Dans un `ORDER BY` on peut préciser :
  - `ASC` (par défaut) pour indiquer un ordre croissant
  - `DESC` pour indiquer un ordre décroissant
- On peut préciser `NULLS LAST` (par défaut pour `ASC`) ou `NULLS FIRST` (par défaut pour `DESC`)

IS (NOT) NULL

Prototype

```
att2 IS (NOT) NULL
```

IS (NOT) NULL = tester si (non) défini

- Les valeurs non définies sont représentées par `NULL`.
- On peut tester si une valeur n'est pas définie grâce à la condition `IS NULL` (ou au contraire `IS NOT NULL`)
- Très (trop ?) utilisé en pratique . . .
- . . . Attention aux choix concernants `NULL` et à l'intuition

IS (NOT) NULL

```
INSERT INTO Student VALUES (432, 'Kevin', null, 1500);
INSERT INTO Student VALUES (321, 'Lori', null, 2500);
```

Quel est le résultat de cette requête ?

```
SELECT sID, sName, GPA
SELECT Student
WHERE GPA > 3.5 or GPA <= 3.5;
```

Et de celle-ci?

```
SELECT sID, sName
FROM Student;
```

Attention à l'évaluation des attributs en présence de NULL !

GROUP BY

Prototype

```
SELECT att1 , att2 ...
FROM MaTable1 , MaTable2
WHERE conditions
GROUP BY attk , attl ...
ORDER BY atti , attj ... ASC
```

GROUP BY = regroupement de tuples

- Indique de procéder à une répartition du résultat en groupes de n-uplets;
- Deux n-uplets sont dans un groupe s'il ont mêmes valeurs sur les attributs `attk,attl... ...`

GROUP BY

La requête renvoie un seul n-uplet par groupe

Restrictions sur `SELECT` et `ORDER BY`

Le `SELECT` et le `ORDER BY` ne peuvent utiliser que des attributs présents dans le `GROUP BY` :

- Dans un groupe, la valeur pour les attributs du `GROUP BY` est fixe, on peut donc l'utiliser.
- En revanche, la valeur pour les autres attributs peut varier, ce qui rend leur utilisation impossible sans fonction d'agrégation

GROUP BY cname, major

Apply

sID	cName	major	dec.
123	Stanford	CS	Y
987	Stanford	CS	Y
876	Stanford	CS	N
123	Stanford	EE	N
765	Stanford	history	Y
678	Stanford	history	Y
234	Berkeley	biology	N
123	Berkeley	CS	Y
987	Berkeley	CS	Y
345	MIT	bioeng.	Y

Apply

sID	cName	major	dec.
876	MIT	biology	Y
543	MIT	CS	N
876	MIT	marine bio.	N
345	Cornell	bioeng.	N
345	Cornell	CS	Y
345	Cornell	EE	N
123	Cornell	EE	Y
765	Cornell	history	N
765	Cornell	psych.	Y

GROUP BY

Fonctions d'agrégation

- Utilisées dans le `SELECT` et dans le `ORDER BY`
- Utilisables en conjonction avec un `GROUP BY`
- Combine les attributs qui ne font pas partie du `GROUP BY`
- Par exemple, `AVG(e)` donne la moyenne de l'expression `e` pour le groupe considéré

Attention

On ne peut pas les utiliser dans le `WHERE`, car le `WHERE` a lieu avant regroupement (cf. clause `HAVING`)

GROUP BY

Exemple

```
SELECT cName, Round(avg(GPA),1) AS Avg
FROM Student NATURAL JOIN Apply
GROUP BY cName;
```

cName	Avg
Stanford	3,7
Cornell	3,4
Berkeley	3,7
MIT	3,7

GROUP BY

Catalogue des fonctions d'agrégations standard

- `COUNT( [DISTINCT] e)` : le nombre d'occurrences (distinctes) de `e`
  - les n-uplets où `e` vaut `NULL` ne sont pas comptés
  - le mot clefs `DISTINCT` compte le nombre de valeurs *différentes*
  - `*` compte le nombre de n-uplets du groupe.
- `MAX(e)` : La valeur maximale de `e`
- `MIN(e)` : La valeur minimale de `e`
- `SUM(e)` : La somme des valeurs de `e`
- `AVG(e)` : La moyenne de l'évaluation de `e`

GROUP BY

Mot clef DISTINCT

- L'expression e peut être précédée du mot clé DISTINCT pour éliminer les doublons
- Important pour COUNT, SUM, AVG, STDDEV et VARIANCE.

Exemple : quelle différence de sens ?

```
SELECT cName, COUNT (Major)
FROM Apply
GROUP BY cName;

SELECT cName, COUNT (DISTINCT Major)
FROM Apply
GROUP BY cName;
```

GROUP BY

Fonction d'agrégation sans `GROUP BY`

- Provoque la création d'un groupe englobant tous les n-uplets sélectionnés.
- Le `SELECT` ne peut alors contenir que des fonctions d'agrégation.
- Utile pour obtenir des informations sur l'ensemble des lignes sélectionnées.

Exemple

```
SELECT AVG(GPA)
FROM Apply NATURAL JOIN Student
WHERE Major = 'CS';
```

Fonctions

Pour construire des expressions complexes

- Fonctions numériques
- Fonctions sur les chaînes de caractères
- Fonctions sur les dates
- Fonctions de conversion (*a.k.a, cast*)

Expression utilisables

- Dans le `SELECT` : calcul sur les valeurs
- Dans le `WHERE` : pour les conditions booléennes
- Dans le `ORDER BY` : pour trier selon des expressions

Des différences entre les SGBDs. Consulter la documentation !

Fonctions

Opérateur `chaine LIKE pattern`

- Opérateur de *pattern-matching*, où le motif utilise :
  - Un caractère arbitraire (_)
  - Des caractères arbitraires (%)

Exemple

```
SELECT sID, sName
FROM Student
WHERE sName LIKE '%ar%';
```

Quel est le résultat de la requête ? ▸ Jeu de données

Comment rendre la requête insensible à la casse ?

Opérateurs ensemblistes

Opérateurs SQL

- $\cup$: `UNION`
- $\cap$: `INTERSECT`
- $\setminus$: `MINUS` ou `EXCEPT`

Effets

- Permettent de combiner les résultats de plusieurs `SELECT`
- Pas de doublons : utiliser `UNION ALL` pour les obtenir
- Les `SELECT` doivent contenir le même nombre d'attributs, de types compatibles
- Les noms des attributs sont ceux du premier `SELECT`

Opérateurs ensemblistes

Exemple

```
SELECT sID FROM Apply WHERE major = 'CS'
UNION
SELECT sID FROM Apply WHERE major = 'EE';

SELECT sID FROM Apply WHERE major = 'CS'
MINUS
SELECT sID FROM Apply WHERE major = 'EE';

SELECT sID FROM Apply WHERE major = 'CS'
INTERSECT
SELECT sID FROM Apply WHERE major = 'EE';
```

▸ Jeu de données

Langage SQL – INSERT, UPDATE, DELETE

- INSERT : ajout de tuples (à une table)
- UPDATE : modification de tuples
- DELETE : suppression de tuples

Prototype INSERT

```
INSERT INTO table1[(atts)]
VALUES (vals), ..., (vals) | select_stmt
ON CONFLICT ...
RETURNING atts
```

Conseil

- Préciser les attributs dans la table
- La clause RETURNING est très utile pour les valeurs générées côté serveur (AUTOINCREMENT, DEFAULT)
- La clause ON CONFLICT permet de faire des *upserts*

Langage SQL – INSERT, UPDATE, DELETE

Prototype UPDATE et DELETE

```
UPDATE table1
SET
      att1 = expr1 | DEFAULT,
      att2 = expr2 | DEFAULT,
      ...
FROM autre_table, ...
WHERE condition

DELETE FROM table1
WHERE condition
```

Conseils

- La clause `FROM` de `UPDATE` permet des jointures
- Attention à `DELETE FROM table1` !

Conclusion

Le modèle relationnel

- Séparation physique VS logique
- Aspect déclaratif (le quoi, pas de comment)
- Paradigme tout est relation
- Le SGBD assure les fonctions essentielles (et un peu plus)

Quelques pièges classiques

- Test d'égalité sur `NULL` dans les conditions
- Accès aux attributs hors du `GROUP BY`
- Conditions de jointures laxistes

Attention à la mauvaise intuition sur un jeu de donnée spécifique (exemple : attributs uniques) !