Bases de données (2/4)

SQL avant la théorie

Stéphane Gonnord, Laurent Jouhet

Lycée du parc - Lyon

Plan

Rappels : Structured Query Language

- Base de donnée : ensemble de tables/relations.
  - Communes
  - Départements
  - Régions

Rappels : Structured Query Language

- Base de donnée : ensemble de tables/relations.
  - Communes
  - Départements
  - Régions
- Table/relation : ensemble de *n*-uplets/lignes/tuples ayant tous la même structure/le même schéma.

...	...	...	...
1198	1	Joyeux	223
...	...	...	...
69123	69	Lyon	484344
...	...	...	...

Rappels : Structured Query Language

- Base de donnée : ensemble de tables/relations.
  - Communes
  - Départements
  - Régions
- Table/relation : ensemble de *n*-uplets/lignes/tuples ayant tous la même structure/le même schéma.

...	...	...	...
1198	1	Joyeux	223
...	...	...	...
69123	69	Lyon	484344
...	...	...	...

- Schéma relationnel : décrit la structure des tables d'une base

`commune(id:entier, dep:texte, nom:texte, pop:entier)`

Autres exemples

- Prénoms
  - Schéma relationnel :

```
(prenom:texte, classe:entier, sexe:texte)
```

Autres exemples

- Prénoms
  - Schéma relationnel :

    `(prenom:texte, classe:entier, sexe:texte)`
  - Exemples de ligne :

    ('Pierre', 843, 'M')
    ('Olivier', 842, 'M')

Autres exemples

- Prénoms
  - Schéma relationnel :

    `(prenom:texte, classe:entier, sexe:texte)`
  - Exemples de ligne :

    ('Pierre', 843, 'M')
    ('Olivier', 842, 'M')
- Communes/départements/régions
  - Schéma relationnel (3 tables) :
    - `communes(id:texte, dep:texte, nom:texte, pop:entier)`
    - `departements(id:texte, reg:entier, nom:texte)`
    - `regions(id:entier, nom:texte)`

Autres exemples

- Prénoms
  - Schéma relationnel :

    `(prenom:texte, classe:entier, sexe:texte)`
  - Exemples de ligne :

    ('Pierre', 843, 'M')
    ('Olivier', 842, 'M')
- Communes/départements/régions
  - Schéma relationnel (3 tables) :
    - `communes(id:texte, dep:texte, nom:texte, pop:entier)`
    - `departements(id:texte, reg:entier, nom:texte)`
    - `regions(id:entier, nom:texte)`
  - Exemples de lignes/tuples :
    - ('2A041', '2A', 'Bonifacio', 2955)
    - ('2A', 94, 'Corse-du-Sud')
    - (94, 'Corse')

Communes, régions et départements

commune			
id	dep	nom	pop
...	...	...	...
69023	69	Lyon	484344
...	...	...	...
2BO50	2B	Calvi	5394
...	...	...	...

departement		
id	reg	nom
...	...	...
69	82	Rhône
...	...	...
2B	94	Haute-Corse
...	...	...

region	
id	nom
...	...
82	Auvergne-Rhône-Alpes
...	...
94	Corse
...	...

Élèves, colleurs et colles

eleves

id	nom	prenom
0	Lyons	Jacques-Louis
1	Laurent	Jean
...	...	...

profs

id	nom	prenom
0	Théron	Pierre
1	Brun	Jules
...	...	...

Élèves, colleurs et colles

eleves

id	nom	prenom
0	Lyons	Jacques-Louis
1	Laurent	Jean
...	...	...

profs

id	nom	prenom
0	Théron	Pierre
1	Brun	Jules
...	...	...

colles

prof	eleve	semaine	note
2	8	1	16
1	0	6	19
...	...	...	...

SQL, DDL, DML : ? ? ?

- SQL : Structured Query Language (« sequel »).

SQL, DDL, DML : ? ? ?

- SQL : Structured Query Language (« sequel »).
- DDL : Data Definition Language. `CREATE`, `DROP`. *Pour créer/supprimer une table dans une BD, selon un schéma relationnel.*

SQL, DDL, DML : ? ? ?

- SQL : Structured Query Language (« sequel »).
- DDL : Data Definition Language. `CREATE`, `DROP`. *Pour créer/supprimer une table dans une BD, selon un schéma relationnel.*
- DML : Data Manipulation Language. `INSERT`, `UPDATE`, `DELETE`. *Pour alimenter les tables d'une BD : nouvelle ligne, suppression, modification.*

SQL, DDL, DML : ? ? ?

- SQL : Structured Query Language (« sequel »).
- DDL : Data Definition Language. `CREATE`, `DROP`. *Pour créer/supprimer une table dans une BD, selon un schéma relationnel.*
- DML : Data Manipulation Language. `INSERT`, `UPDATE`, `DELETE`. *Pour alimenter les tables d'une BD : nouvelle ligne, suppression, modification.*
- Interrogation des données : `SELECT`. *Pour faire une requête.*

Format général : à connaître !

```
SELECT <attributs>, <calculs d'agrégats>
FROM <tables>
    JOIN ... ON ...
WHERE <conditions>
GROUP BY <attributs>
HAVING <conditions>
ORDER BY <attribut> ASC, <attribut> DESC
```

Dans telles tables, tu prends les lignes vérifiant telles conditions ; tu les groupes selon tels critères. Dans les groupes, tu vas juste prendre ceux dont telle moyenne (par exemple) sur tel attribut vérifie telle condition.

Format général : à connaître !

```
SELECT <attributs>, <calculs d'agrégats>
FROM <tables>
    JOIN ... ON ...
WHERE <conditions>
GROUP BY <attributs>
HAVING <conditions>
ORDER BY <attribut> ASC, <attribut> DESC
```

Dans telles tables, tu prends les lignes vérifiant telles conditions ; tu les groupes selon tels critères. Dans les groupes, tu vas juste prendre ceux dont telle moyenne (par exemple) sur tel attribut vérifie telle condition.
Ah et puis tu vas me donner le résultat sous forme triée selon tels attributs !

Un exemple musclé

Ouch !

```
SELECT eleve, eleves.nom, COUNT(*) AS plantages
FROM eleves
    JOIN colles ON eleve = ide
WHERE note < 8
GROUP BY eleve
HAVING plantages = 5
ORDER BY plantages
```

Alors, que vient-on de demander ?

Un exemple musclé

Ouch !

```
SELECT eleve, eleves.nom, COUNT(*) AS plantages
FROM eleves
    JOIN colles ON eleve = ide
WHERE note < 8
GROUP BY eleve
HAVING plantages = 5
ORDER BY plantages
```

Alors, que vient-on de demander ?

Don't panic... on va commencer par des choses plus simples !

Sélection, projection

- Requête de base :

```
SELECT <tels attributs>
FROM <telle table>
WHERE <telle(s) condition(s)>
```

Sélection, projection

- Requête de base :

```
SELECT <tels attributs>
FROM <telle table>
WHERE <telle(s) condition(s)>
```

- Exemples :
  - ```
    SELECT *                                -- tous les attributs
    FROM communes
    ```
    ```

Sélection, projection

- Requête de base :

```
SELECT <tels attributs>
FROM <telle table>
WHERE <telle(s) condition(s)>
```

- Exemples :
  - ```
    SELECT *                          -- tous les attributs
    FROM communes
    ```
  - ```
    SELECT nom, pop
    FROM communes
    ```

Sélection, projection

- Requête de base :

```
SELECT <tels attributs>
FROM <telle table>
WHERE <telle(s) condition(s)>
```

- Exemples :
  - `SELECT *` `-- tous les attributs`
    `FROM communes`
  - `SELECT nom, pop`
    `FROM communes`
  - `SELECT nom, pop`
    `FROM communes`
    `WHERE pop > 100000`

Sélection, projection

- Requête de base :

```
SELECT <tels attributs>
FROM <telle table>
WHERE <telle(s) condition(s)>
```

- Exemples :
  - `SELECT *` `-- tous les attributs`
    `FROM communes`
  - `SELECT nom, pop`
    `FROM communes`
  - `SELECT nom, pop`
    `FROM communes`
    `WHERE pop > 100000`
  - `SELECT *`
    `FROM triangles`
    `WHERE ab = bc AND bc = ac` `-- un seul =`

Opérations ensemblistes

- Union, intersection, différence : Bof...

Opérations ensemblistes

- Union, intersection, différence : Bof...
  - Rarement utile en pratique

Opérations ensemblistes

- Union, intersection, différence : Bof...
  - Rarement utile en pratique
  - Mais au programme !

Opérations ensemblistes

- Union, intersection, différence : Bof...
  - Rarement utile en pratique
  - Mais au programme !
  - Attention à ne pas écrire n'importe quoi (torchons, serviettes...)
- SQL :
  - `SELECT ... FROM ...`
    `UNION`
    `SELECT ... FROM ...`
  - `SELECT ... FROM ... INTERSECT SELECT ... FROM ...`
  - `SELECT ... FROM ... EXCEPT SELECT ... FROM ...`
- Et si les attributs sont différents ?

Opérations ensemblistes

- Union, intersection, différence : Bof...
  - Rarement utile en pratique
  - Mais au programme !
  - Attention à ne pas écrire n'importe quoi (torchons, serviettes...)
- SQL :
  - `SELECT ... FROM ...`
    `UNION`
    `SELECT ... FROM ...`
  - `SELECT ... FROM ... INTERSECT SELECT ... FROM ...`
  - `SELECT ... FROM ... EXCEPT SELECT ... FROM ...`
- Et si les attributs sont différents ? *n'importe quoi*

Opérations ensemblistes

- Union, intersection, différence : Bof...
  - Rarement utile en pratique
  - Mais au programme !
  - Attention à ne pas écrire n'importe quoi (torchons, serviettes...)
- SQL :
  - `SELECT ... FROM ...`
    `UNION`
    `SELECT ... FROM ...`
  - `SELECT ... FROM ... INTERSECT SELECT ... FROM ...`
  - `SELECT ... FROM ... EXCEPT SELECT ... FROM ...`
- Et si les attributs sont différents ? *n'importe quoi*
- Produit cartésien :
  - Définition : comme en maths :

$$A \times B = \{(a, b) \mid a \in A \text{ et } b \in B\}$$

Opérations ensemblistes

- Union, intersection, différence : Bof...
  - Rarement utile en pratique
  - Mais au programme !
  - Attention à ne pas écrire n'importe quoi (torchons, serviettes...)
- SQL :
  - `SELECT ... FROM ...`
    `UNION`
    `SELECT ... FROM ...`
  - `SELECT ... FROM ... INTERSECT SELECT ... FROM ...`
  - `SELECT ... FROM ... EXCEPT SELECT ... FROM ...`
- Et si les attributs sont différents ? *n'importe quoi*
- Produit cartésien :
  - Définition : comme en maths :

$$A \times B = \{(a, b) \mid a \in A \text{ et } b \in B\}$$

  - SQL : pas comme en maths !
    `SELECT * from table1 , table2`

De l'inutilité du produit cartésien

- Deux tables :

commune	
nom	dep
Lyon	69
Calvi	2B
Corte	2B

departement	
id	nom
69	Rhône
2B	Haute-Corse

De l'inutilité du produit cartésien

- Deux tables :

commune	
nom	dep
Lyon	69
Calvi	2B
Corte	2B

departement	
id	nom
69	Rhône
2B	Haute-Corse

- Et leur produit :

commune × departement			
nom	dep	**id**	nom
Lyon	69	69	Rhône
Lyon	69	2B	Haute-Corse
Calvi	2B	69	Rhône
Calvi	2B	2B	Haute-Corse
Corte	2B	69	Rhône
Corte	2B	2B	Haute-Corse

Jointure

- Deux tables :

commune	
nom	dep
Lyon	69
Calvi	2B
Corte	2B

departement	
id	nom
69	Rhône
2B	Haute-Corse

Jointure

- Deux tables :

commune	
nom	dep
Lyon	69
Calvi	2B
Corte	2B

departement	
id	nom
69	Rhône
2B	Haute-Corse

- Et une *jointure* naturelle :

commune ⋈ departement			
nom	dep	**id**	nom
Lyon	69	69	Rhône
Calvi	2B	2B	Haute-Corse
Corte	2B	2B	Haute-Corse

- Une base de données est *pensée* dès le départ autour des jointures de tables.

Jointure

- Formellement : $R_1 \underset{a=b}{\bowtie} R_2 \subset R_1 \times R_2$... (cf. algèbre relationnelle)

Jointure

- Formellement : $R_1 \underset{a=b}{\bowtie} R_2 \subset R_1 \times R_2$... (cf. algèbre relationnelle)
- Et si R_1 et R_2 ont deux attributs de même nom ?

Jointure

- Formellement : $R_1 \underset{a=b}{\bowtie} R_2 \subset R_1 \times R_2$... (cf. algèbre relationnelle)
- Et si R_1 et R_2 ont deux attributs de même nom ?
- En SQL : deux syntaxes équivalentes :
  - ```
    SELECT ...
    FROM table1 JOIN table2
         ON condition
    WHERE...
    ```
  - ```
    SELECT ...
    FROM table1, table2
    WHERE condition AND ...
    ```

Jointure

- Formellement : $R_1 \underset{a=b}{\bowtie} R_2 \subset R_1 \times R_2$... (cf. algèbre relationnelle)
- Et si R_1 et R_2 ont deux attributs de même nom ?
- En SQL : deux syntaxes équivalentes :
  - ```
    SELECT ...
    FROM table1 JOIN table2
         ON condition
    WHERE...
    ```
  - ```
    SELECT ...
    FROM table1, table2
    WHERE condition AND ...
    ```
- SQL, encore :
  - `WHERE table1.foo = table2.bar`
  - ```
    SELECT ...
    FROM table1 JOIN table2 JOIN table3
         ON condition1 AND condition2
    WHERE...
    ```
    ```

Exemples, utilité des clés

- *Clé* : ensemble d'attributs caractérisant les éléments d'une table.

Exemples, utilité des clés

- *Clé* : ensemble d'attributs caractérisant les éléments d'une table.
- *Clé primaire* : attribut UNIQUE caractérisant les éléments d'une table.

Exemples, utilité des clés

- *Clé* : ensemble d'attributs caractérisant les éléments d'une table.
- *Clé primaire* : attribut UNIQUE caractérisant les éléments d'une table.
- Bien avoir le schéma relationnel devant les yeux.

Exemples, utilité des clés

- *Clé* : ensemble d'attributs caractérisant les éléments d'une table.
- *Clé primaire* : attribut UNIQUE caractérisant les éléments d'une table.
- Bien avoir le schéma relationnel devant les yeux.
- Exemples :
  - `FROM communes JOIN departements -- Ne pas oublier SELECT ON communes.dep = departements.id`

Exemples, utilité des clés

- *Clé* : ensemble d'attributs caractérisant les éléments d'une table.
- *Clé primaire* : attribut UNIQUE caractérisant les éléments d'une table.
- Bien avoir le schéma relationnel devant les yeux.
- Exemples :
  - `FROM communes JOIN departements -- Ne pas oublier SELECT`
    `ON communes.dep = departements.id`
  - `FROM eleves JOIN colles`
    `ON ide = eleve`

Exemples, utilité des clés

- *Clé* : ensemble d'attributs caractérisant les éléments d'une table.
- *Clé primaire* : attribut UNIQUE caractérisant les éléments d'une table.
- Bien avoir le schéma relationnel devant les yeux.
- Exemples :
  - `FROM communes JOIN departements -- Ne pas oublier SELECT`
    `ON communes.dep = departements.id`
  - `FROM eleves JOIN colles`
    `ON ide = eleve`
  - `FROM eleves JOIN colles JOIN profs`
    `ON ide = eleve AND prof = idp`

Exemples, utilité des clés

- *Clé* : ensemble d'attributs caractérisant les éléments d'une table.
- *Clé primaire* : attribut UNIQUE caractérisant les éléments d'une table.
- Bien avoir le schéma relationnel devant les yeux.
- Exemples :
  - `FROM communes JOIN departements -- Ne pas oublier SELECT`
    `ON communes.dep = departements.id`
  - `FROM eleves JOIN colles`
    `ON ide = eleve`
  - `FROM eleves JOIN colles JOIN profs`
    `ON ide = eleve AND prof = idp`
  - `FROM clubs AS c1 JOIN matchs JOIN clubs AS c2`
    `ON c1.idc = eq1 AND c2.idc = eq2`

Agrégats

- Principe :
  1. On regroupe (en général) les lignes suivant des attributs ;
  2. On applique une *fonction d'agrégation* à chacun de ces groupes.
  3. On peut encore une fois sélectionner des lignes avec HAVING

Agrégats

- Principe :
  1. On regroupe (en général) les lignes suivant des attributs ;
  2. On applique une *fonction d'agrégation* à chacun de ces groupes.
  3. On peut encore une fois sélectionner des lignes avec HAVING
- Fonctions : `MIN, MAX, COUNT(...), AVG(...), SUM(...)`

Agrégats

- Principe :
  1. On regroupe (en général) les lignes suivant des attributs ;
  2. On applique une *fonction d'agrégation* à chacun de ces groupes.
  3. On peut encore une fois sélectionner des lignes avec HAVING
- Fonctions : `MIN, MAX, COUNT(...), AVG(...), SUM(...)`
- SQL :

```
SELECT reg, departements.nom, COUNT(*)
FROM departements JOIN regions
     ON reg = regions.id
GROUP BY reg
```

Agrégats

- Principe :
  1. On regroupe (en général) les lignes suivant des attributs ;
  2. On applique une *fonction d'agrégation* à chacun de ces groupes.
  3. On peut encore une fois sélectionner des lignes avec HAVING
- Fonctions : `MIN, MAX, COUNT(...), AVG(...), SUM(...)`
- SQL :

```
SELECT reg, departements.nom, COUNT(*)
FROM departements JOIN regions
     ON reg = regions.id
GROUP BY reg
```

COUNT(*) va compter le nombre de lignes dans chaque groupe !

reg	nom	COUNT(*)
1	Guadeloupe	1
11	Seine-et-Marne	8
...	...	...

Agrégats

- Condition en amont (`WHERE`) et/ou aval (`HAVING`)

```
SELECT reg, departements.nom, COUNT(*) AS nb_dep
FROM departements JOIN regions
     ON reg = regions.id
GROUP BY reg
HAVING nb_dep >= 5
```

Agrégats

- Condition en amont (`WHERE`) et/ou aval (`HAVING`)

```
SELECT reg, departements.nom, COUNT(*) AS nb_dep
FROM departements JOIN regions
     ON reg = regions.id
GROUP BY reg
HAVING nb_dep >= 5
```

reg	nom	count(*)
11	Seine-et-Marne	8
24	Cher	6
...	...	...

Agrégats

- Condition en amont (`WHERE`) et/ou aval (`HAVING`)

```
SELECT reg, departements.nom, COUNT(*) AS nb_dep
FROM departements JOIN regions
     ON reg = regions.id
GROUP BY reg
HAVING nb_dep >= 5
```

reg	nom	count(*)
11	Seine-et-Marne	8
24	Cher	6
...	...	...

- À retenir :

Penser à projeter (SELECT) les attributs selon lesquels on a groupé (GROUP BY) les lignes.

Exercices

Avec les tables aux schémas relationnels déjà vus, déterminer...

- la note maximale parmi toutes les colles ;

Exercices

Avec les tables aux schémas relationnels déjà vus, déterminer...

- la note maximale parmi toutes les colles ;
- ```
  SELECT MAX(note)
  FROM colles
  ```
- la note maximale parmi toutes les colles de Jaques-Louis Lions ;
  ```

Exercices

Avec les tables aux schémas relationnels déjà vus, déterminer...

- la note maximale parmi toutes les colles ;
- ```
  SELECT MAX(note)
  FROM colles
  ```
- la note maximale parmi toutes les colles de Jaques-Louis Lions ;
- ```
  SELECT MAX(note)
  FROM colles JOIN eleves
       ON ide = eleve
  WHERE nom="Lions"
  ```
- la liste des moyennes des élèves ;

Exercices

Avec les tables aux schémas relationnels déjà vus, déterminer...

- la note maximale parmi toutes les colles ;
- ```
  SELECT MAX(note)
  FROM colles
  ```
- la note maximale parmi toutes les colles de Jaques-Louis Lions ;
- ```
  SELECT MAX(note)
  FROM colles JOIN eleves
       ON ide = eleve
  WHERE nom="Lions"
  ```
- la liste des moyennes des élèves ;
- ```
  SELECT eleve, nom, AVG(note)
  FROM colles JOIN eleves
       ON ide = eleve
  GROUP BY eleve
  ```
  ```

- la liste des départements avec leur nombre de communes ;

- la liste des départements avec leur nombre de communes ;
- ```
  SELECT dep, departements.nom, COUNT(*)
  FROM departements JOIN communes
       ON dep = departements.id
  GROUP BY dep
  ```
  ```

Un algorithme de conception de requêtes

Plusieurs passages sont possibles...

1. `SELECT` : quels attributs (et/ou agrégats) nous intéressent ?
2. `FROM` : issus de quelles tables ?
3. `JOIN ... ON` : si on joint n tables, il y a *a priori* $n-1$ conditions de jointures.
4. `WHERE` : quelles conditions/restrictions en amont ?
5. `GROUP BY` : comment veut-on regrouper les tuples ? *ajouter au* `SELECT`.
6. `HAVING` : restrictions en aval, portant sur les agrégats.
7. Sous-requètes éventuelles, paramétrées ou non ; utilisation de Python...