BDBIO - Manipulation relationnelle - bases du SQL

Fabien Duchateau

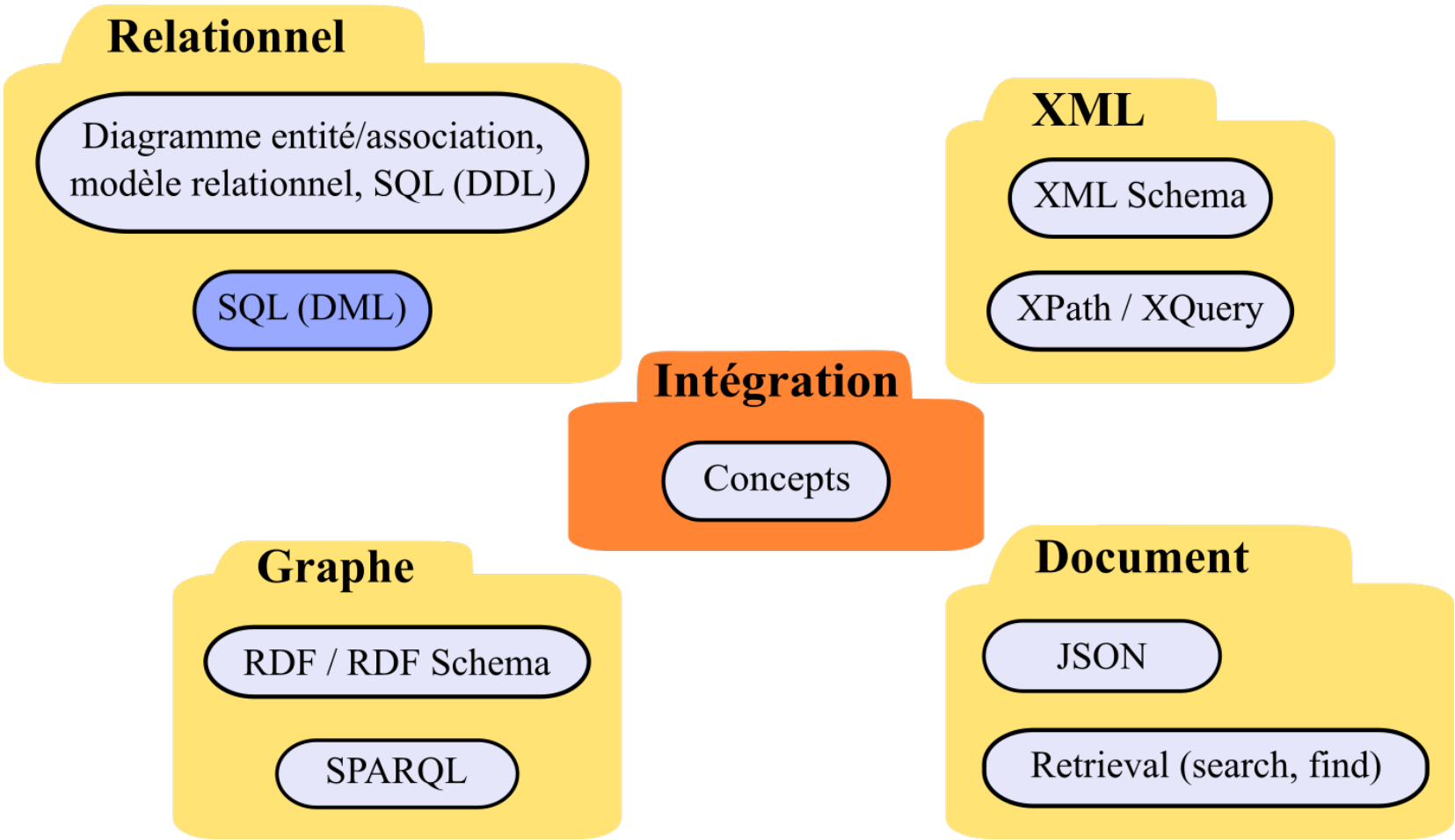
fabien.duchateau [at] univ-lyon1.fr

Université Claude Bernard Lyon 1

2023 - 2024

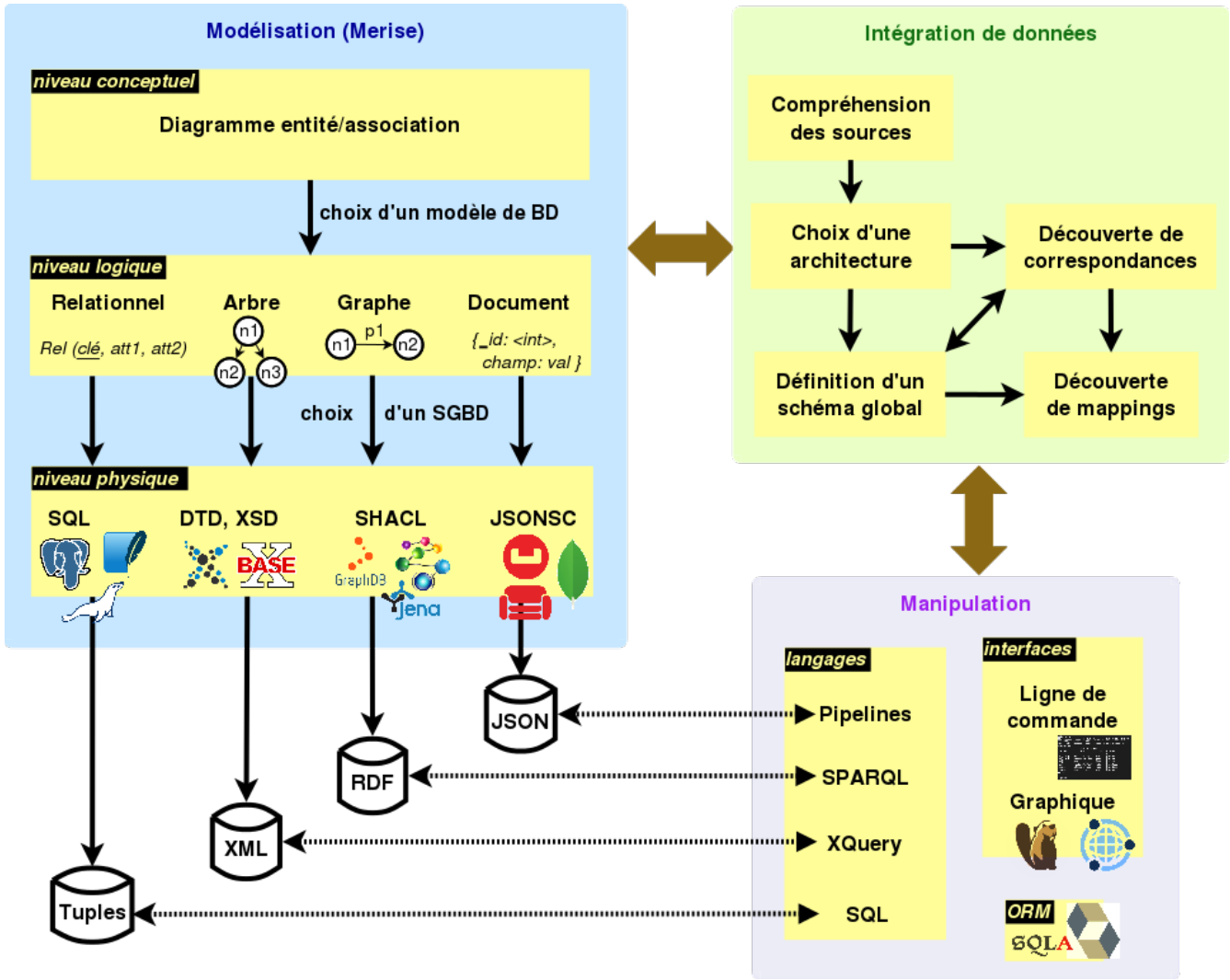
https://perso.liris.cnrs.fr/fabien.duchateau/BDBIO/

Positionnement dans BDBIO



Ces diapositives utilisent le genre féminin (e.g., chercheuse, développeuses) plutôt que l'écriture inclusive (moins accessible, moins concise, et pas totalement inclusive)

Relations entre les notions étudiées en BDBIO



Généralités sur le langage SQL

SQL = Structured Query Language

- Un langage de description de données
- Un langage de manipulation de données (CRUD)
- Un langage pour administrer la base, gérer les contrôles d'accès

SQL comme langage de manipulation de données ⇒ **déclaratif**, i.e., description du résultat escompté

Cours et tutoriels SQL sur http://sql.sh
http://use-the-index-luke.com/
http://modern-sql.com/

Spécificités de SQL

- Des centaines de SGBD utilisent SQL
- Différences entre la théorie (monde ensembliste) et la norme SQL : doublons, tri, valeur non définie, absence de certains opérateurs, etc.
- Différences entre la norme SQL et son implémentation dans les SGBD
- Différences entre les implémentations de SQL (propre à chaque SGBD)

Dans ce cours, la syntaxe SQL est celle implémentée dans le SGBD PostgreSQL

---

https://www.postgresql.org/docs/current/

Syntaxe de base d'une requête SQL

SELECT [**DISTINCT**] att_1 [, att_2 [**AS** att'_2], ...]
FROM nom_table_1 [, nom_table_2 [alias], ...]
[**WHERE** *condition*] **;**

- La clause SELECT liste les attributs du résultat (DISTINCT pour supprimer les n-uplets redondants)
- La clause FROM liste les tables utilisées (éventuellement jointes)
- La clause WHERE filtre les n-uplets selon une condition, qui est une combinaison (AND, OR) d'expressions (=, !=, <, <=, >, >=)

Rappel du jeu de données

ÉLÈVE (idE, *nomE*, *moyenneLycée*, *effectifLycée*)
CANDIDATURE (#*idE*, #*nomU*, département, *décision*)
UNIVERSITÉ (nomU, *ville*, *effectif*)

idE	nomE	moyenneLycée	effectifLycée
123	Ana	19.5	1000
234	Bob	18	1500
345	Chloé	17.5	500
456	Damien	19.5	1000
543	Chloé	17	2000
567	Éléonore	14.5	2000
654	Ana	19.5	1000
678	Farid	19	200
765	Joana	14.5	1500
789	Gisèle	17	800
876	Irène	19.5	400
898	Hector	18.5	800

Table ÉLÈVE

nomU	ville	effectif
INSA	Lyon	36000
UCB	Lyon	15000
UJF	Grenoble	10000
UJM	Saint-Étienne	21000

Table UNIVERSITÉ

idE	nomU	département	décision
123	INSA	informatique	O
123	UCB	électronique	N
123	UCB	informatique	O
123	UJM	électronique	O
234	INSA	biologie	N
345	UJF	bioinformatique	O
345	UJM	bioinformatique	N
345	UJM	électronique	N
345	UJM	informatique	O
543	UJF	informatique	N
678	UCB	histoire	O
765	UCB	histoire	O
765	UJM	histoire	N
765	UJM	psychologie	O
876	UCB	informatique	N
876	UJF	biologie	O
876	UJF	biologie marine	N
898	INSA	informatique	O
898	UCB	informatique	O

Table CANDIDATURE

Dans ces transparents, les tables sont abrégées en *E*, *C* et *U*

Jeu de données francisé inspiré du cours Databases de Stanford

Plan

Syntaxe de base

SELECT att_1, att_2, ...
FROM *nom_table*
WHERE *condition* ;

- La clause SELECT indique les attributs du résultat (projection)
  - `att1, att2, …` peuvent être remplacés par * pour projeter tous les attributs (de toutes les tables du FROM)
  - le mot-clé DISTINCT supprime les doublons
- La clause FROM indique les relations/tables utilisées
- La clause WHERE filtre les instances selon une condition (sélection, en combinant des comparaisons avec AND et OR)

Un exemple de requête SQL

idE	nomE	moyenneLycée	effectifLycée
123	Ana	19.5	1000
234	Bob	18	1500
345	Chloé	17.5	500
456	Damien	19.5	1000
543	Chloé	17	2000
567	Éléonore	14.5	2000
654	Ana	19.5	1000
678	Farid	19	200
765	Joana	14.5	1500
789	Gisèle	17	800
876	Irène	19.5	400
898	Hector	18.5	800

Table Élève

nomU	ville	effectif
INSA	Lyon	36000
UCB	Lyon	15000
UJF	Grenoble	10000
UJM	Saint-Étienne	21000

Table Université

idE	nomU	département	décision
123	INSA	informatique	O
123	UCB	électronique	N
123	UCB	informatique	O
123	UJM	électronique	O
234	INSA	biologie	N
345	UJF	bioinformatique	O
345	UJM	bioinformatique	N
345	UJM	électronique	N
345	UJM	informatique	O
543	UJF	informatique	N
678	UCB	histoire	O
765	UCB	histoire	O
765	UJM	histoire	N
765	UJM	psychologie	O
876	UCB	informatique	N
876	UJF	biologie	O
876	UJF	biologie marine	N
898	INSA	informatique	O
898	UCB	informatique	O

Table Candidature

Informations sur les universités de Lyon avec un effectif inférieur à 25000

SELECT *
FROM Université
WHERE ville = 'Lyon'
AND effectif < 25000 ;

nomU	ville	effectif
UCB	Lyon	15000

Renommage et alias

Mot-clé AS pour renommer un attribut :

SELECT att_1 **AS** att'_1, att_2 **AS** att'_2, ...
FROM *nom_table* **;**

- L'attribut att_1 sera renommé en att'_1, etc.

Alias de table comme moyen de désambiguïsation entre attributs identiques ou pour utiliser deux fois la même table :

SELECT DISTINCT $t.att_1$, att_2, ...
FROM *nom_table* [**AS**] *t*, *nom_table* [**AS**] *t2* **;**

- L'alias de la table est *t*, accès à l'attribut att_1 par $t.att_1$

Exemple de renommage et alias

idE	nomE	moyenneLycée	effectifLycée
123	Ana	19.5	1000
234	Bob	18	1500
345	Chloé	17.5	500
456	Damien	19.5	1000
543	Chloé	17	2000
567	Éléonore	14.5	2000
654	Ana	19.5	1000
678	Farid	19	200
765	Joana	14.5	1500
789	Gisèle	17	800
876	Irène	19.5	400
898	Hector	18.5	800

Table Élève

nomU	ville	effectif
INSA	Lyon	36000
UCB	Lyon	15000
UJF	Grenoble	10000
UJM	Saint-Étienne	21000

Table Université

idE	nomU	département	décision
123	INSA	informatique	O
123	UCB	électronique	N
123	UCB	informatique	O
123	UJM	électronique	O
234	INSA	biologie	N
345	UJF	bioinformatique	O
345	UJM	bioinformatique	N
345	UJM	électronique	N
345	UJM	informatique	O
543	UJF	informatique	N
678	UCB	histoire	O
765	UCB	histoire	O
765	UJM	histoire	N
765	UJM	psychologie	O
876	UCB	informatique	N
876	UJF	biologie	O
876	UJF	biologie marine	N
898	INSA	informatique	O
898	UCB	informatique	O

Table Candidature

Le nom et la ville des universités, avec renommage et alias

SELECT u.nomU **AS** nom-univ, u.ville **AS** ville-univ
FROM Université u ;

nom-univ	ville-univ
INSA	Lyon
UCB	Lyon
UJF	Grenoble
UJM	Saint-Étienne

Plan

Syntaxe du produit cartésien

SELECT att_1, att_2, ...
FROM nom_table_1, nom_table_2, ... ;

- Plusieurs tables séparées par des virgules = produit cartésien entre ces différentes tables
- Si la requête utilise un attribut *att* présent dans plusieurs tables, on doit l'écrire *nom_table.att* ou utiliser un alias de table *alias.att*
- Le produit cartésien a peu d'intérêt pour l'interrogation, mais sert d'opération de base pour les jointures

Exemple de produit cartésien

idE	nomE	moyenneLycée	effectifLycée
123	Ana	19.5	1000
234	Bob	18	1500
345	Chloé	17.5	500
456	Damien	19.5	1000
543	Chloé	17	2000
567	Éléonore	14.5	2000
654	Ana	19.5	1000
678	Farid	19	200
765	Joana	14.5	1500
789	Gisèle	17	800
876	Irène	19.5	400
898	Hector	18.5	800

Table Élève

nomU	ville	effectif
INSA	Lyon	36000
UCB	Lyon	15000
UJF	Grenoble	10000
UJM	Saint-Étienne	21000

Table Université

idE	nomU	département	décision
123	INSA	informatique	O
123	UCB	électronique	N
123	UCB	informatique	O
123	UJM	électronique	O
234	INSA	biologie	N
345	UJF	bioinformatique	O
345	UJM	bioinformatique	N
345	UJM	électronique	N
345	UJM	informatique	O
543	UJF	informatique	N
678	UCB	histoire	O
765	UCB	histoire	O
765	UJM	histoire	N
765	UJM	psychologie	O
876	UCB	informatique	N
876	UJF	biologie	O
876	UJF	biologie marine	N
898	INSA	informatique	O
898	UCB	informatique	O

Table Candidature

Les paires de nom d'université et de nom d'élève

SELECT nomU, nomE
FROM Université, Élève ;

nomU	nomE
INSA	Ana
UCB	Ana
UJF	Ana
UJM	Ana
INSA	Bob
...	...

Total de 48 tuples (12×4)

Syntaxe de jointure interne

La **jointure interne** est fréquemment utilisée : seuls les tuples (de chacune des tables jointes) qui respectent la condition de jointure sont "assemblés" en un tuple résultat

SELECT att_1, att_2, ...
FROM nom_table_1 **INNER JOIN** nom_table_2
ON $nom\_table_1.att_x \ \Theta \ nom\_table_2.att_x$
[**WHERE** *autres_conditions*] **;**

- La condition de jointure $nom\_table_1.att_x \ \Theta \ nom\_table_2.att_x$ s'exprime avec le mot-clé ON, avec Θ un opérateur parmi $=$, $\neq$, $<$, $>$, $\leq$, $\geq$, LIKE, ...
- Les autres conditions s'expriment dans le WHERE

Exemple de jointure interne

idE	nomE	moyenneLycée	effectifLycée
123	Ana	19.5	1000
234	Bob	18	1500
345	Chloé	17.5	500
456	Damien	19.5	1000
543	Chloé	17	2000
567	Éléonore	14.5	2000
654	Ana	19.5	1000
678	Farid	19	200
765	Joana	14.5	1500
789	Gisèle	17	800
876	Irène	19.5	400
898	Hector	18.5	800

Table Élève

nomU	ville	effectif
INSA	Lyon	36000
UCB	Lyon	15000
UJF	Grenoble	10000
UJM	Saint-Étienne	21000

Table Université

idE	nomU	département	décision
123	INSA	informatique	O
123	UCB	électronique	N
123	UCB	informatique	O
123	UJM	électronique	O
234	INSA	biologie	N
345	UJF	bioinformatique	O
345	UJM	bioinformatique	N
345	UJM	électronique	N
345	UJM	informatique	O
543	UJF	informatique	N
678	UCB	histoire	O
765	UCB	histoire	O
765	UJM	histoire	N
765	UJM	psychologie	O
876	UCB	informatique	N
876	UJF	biologie	O
876	UJF	biologie marine	N
898	INSA	informatique	O
898	UCB	informatique	O

Table Candidature

Les élèves qui ont candidaté dans une université grenobloise

SELECT *E*.idE, nomE
FROM *E* **INNER JOIN** *C* **ON** *E*.idE = *C*.idE
INNER JOIN *U* **ON** *C*.nomU = *U*.nomU
WHERE *U*.ville = 'Grenoble' **;**

idE	nomE
345	Chloé
543	Chloé
876	Irène
876	Irène

Syntaxe de jointure naturelle

La **jointure naturelle** est une jointure interne avec condition d'égalité entre tous les attributs portant le même libellé

SELECT att_1, att_2, ...
FROM nom_table_1 **NATURAL JOIN** nom_table_2
[**WHERE** $autres\_conditions$] **;**

- Soient att_{c_1}, ..., att_{c_k} les attributs communs des tables nom_table_1 et nom_table_2
- Les tuples de nom_table_1 et nom_table_2 qui possèdent des valeurs égales sur tous leurs attributs communs att_{c_1}, ..., att_{c_k} sont "assemblés" en un tuple résultat

Exemple de jointure naturelle

idE	nomE	moyenneLycée	effectifLycée
123	Ana	19.5	1000
234	Bob	18	1500
345	Chloé	17.5	500
456	Damien	19.5	1000
543	Chloé	17	2000
567	Éléonore	14.5	2000
654	Ana	19.5	1000
678	Farid	19	200
765	Joana	14.5	1500
789	Gisèle	17	800
876	Irène	19.5	400
898	Hector	18.5	800

Table Élève

nomU	ville	effectif
INSA	Lyon	36000
UCB	Lyon	15000
UJF	Grenoble	10000
UJM	Saint-Étienne	21000

Table Université

idE	nomU	département	décision
123	INSA	informatique	O
123	UCB	électronique	N
123	UCB	informatique	O
123	UJM	électronique	O
234	INSA	biologie	N
345	UJF	bioinformatique	O
345	UJM	bioinformatique	N
345	UJM	électronique	N
345	UJM	informatique	O
543	UJF	informatique	N
678	UCB	histoire	O
765	UCB	histoire	O
765	UJM	histoire	N
765	UJM	psychologie	O
876	UCB	informatique	N
876	UJF	biologie	O
876	UJF	biologie marine	N
898	INSA	informatique	O
898	UCB	informatique	O

Table Candidature

Les élèves avec plus de 19 de moyenne qui ont candidaté

SELECT DISTINCT *E*.idE, nomE, moyenneLycee, nomU
FROM *E* **NATURAL JOIN** *C*
WHERE moyenneLycee $>$ 19 **;**

idE	nomE	moyenneLycee	nomU
123	Ana	19.5	UJM
123	Ana	19.5	INSA
123	Ana	19.5	UCB
876	Irène	19.5	UJF
876	Irène	19.5	UCB

Jointure externe

SELECT att_1, att_2, ...
FROM nom_table_1 < **LEFT** | **RIGHT** >
[**OUTER**] **JOIN** nom_table_2
ON $nom\_table_1.att_x \Theta nom\_table_2.att_x$
[**WHERE** *autres_conditions*] **;**

- Une requête avec jointure externe retourne :
  - les tuples qui satisfont la condition de la jointure (comme une jointure interne)
  - mais aussi les tuples de la table de gauche (**LEFT**) ou de celle de droite (**RIGHT**) qui ne la satisfont pas

Exemple de jointure externe

idE	nomE	moyenneLycée	effectifLycée
123	Ana	19.5	1000
234	Bob	18	1500
345	Chloé	17.5	500
456	Damien	19.5	1000
543	Chloé	17	2000
567	Éléonore	14.5	2000
654	Ana	19.5	1000
678	Farid	19	200
765	Joana	14.5	1500
789	Gisèle	17	800
876	Irène	19.5	400
898	Hector	18.5	800

Table Élève

nomU	ville	effectif
INSA	Lyon	36000
UCB	Lyon	15000
UJF	Grenoble	10000
UJM	Saint-Étienne	21000

Table Université

idE	nomU	département	décision
123	INSA	informatique	O
123	UCB	électronique	N
123	UCB	informatique	O
123	UJM	électronique	O
234	INSA	biologie	N
345	UJF	bioinformatique	O
345	UJM	bioinformatique	N
345	UJM	électronique	N
345	UJM	informatique	O
543	UJF	informatique	N
678	UCB	histoire	O
765	UCB	histoire	O
765	UJM	histoire	N
765	UJM	psychologie	O
876	UCB	informatique	N
876	UJF	biologie	O
876	UJF	biologie marine	N
898	INSA	informatique	O
898	UCB	informatique	O

Table Candidature

Les élèves avec une moyenne supérieure à 19 et les éventuelles universités où ils/elles ont candidaté

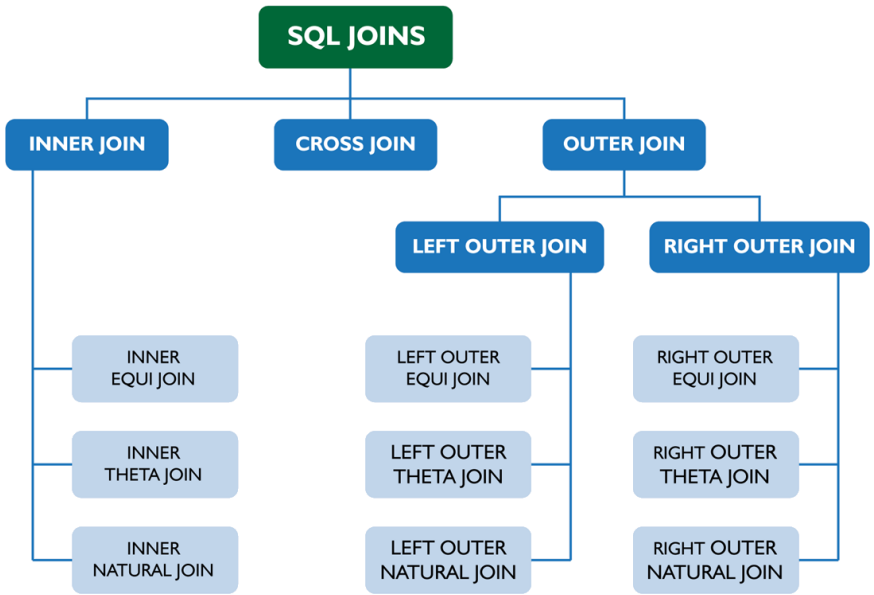
SELECT DISTINCT *E*.idE, nomE, nomU
FROM *E* **LEFT OUTER JOIN** *C*
ON *E*.idE = *C*.idE
WHERE *E*.moyenneLycee > 19 **;**

idE	nomE	nomU
123	Ana	INSA
123	Ana	UCB
123	Ana	UJM
456	Damien	NULL
654	Ana	NULL
876	Irène	UCB
876	Irène	UJF

D'autres types de jointure

Classification selon :

- La condition = naturelle, équi-jointure, θ-jointure
- Les n-uplets conservés dans le résultat = jointure interne, jointures externes (et produit cartésien)
- Les attributs conservés dans le résultat = semi-jointure



http://www.postgresql.org/docs/current/queries-table-expressions.html#QUERIES-FROM

Plan

Syntaxe du tri

Clause ORDER BY pour trier le résultat d'une requête :

SELECT att_1, att_2, ...
FROM *nom_table*
WHERE *condition*
ORDER BY att_i, att_j, ... ;

- Le résultat de la requête est trié selon l'ordre naturel croissant de l'attribut att_i
- En cas d'égalité entre deux lignes au niveau de l'attribut att_i, on utilise l'attribut suivant att_j, etc.
- Le nom d'un attribut peut être suivi par ASC ou DESC pour indiquer un ordre croissant (défaut) ou décroissant

https://www.postgresql.org/docs/current/queries-order.html

Exemple de tri

idE	nomE	moyenneLycée	effectifLycée
123	Ana	19.5	1000
234	Bob	18	1500
345	Chloé	17.5	500
456	Damien	19.5	1000
543	Chloé	17	2000
567	Éléonore	14.5	2000
654	Ana	19.5	1000
678	Farid	19	200
765	Joana	14.5	1500
789	Gisèle	17	800
876	Irène	19.5	400
898	Hector	18.5	800

Table Élève

nomU	ville	effectif
INSA	Lyon	36000
UCB	Lyon	15000
UJF	Grenoble	10000
UJM	Saint-Étienne	21000

Table Université

idE	nomU	département	décision
123	INSA	informatique	O
123	UCB	électronique	N
123	UCB	informatique	O
123	UJM	électronique	O
234	INSA	biologie	N
345	UJF	bioinformatique	O
345	UJM	bioinformatique	N
345	UJM	électronique	N
345	UJM	informatique	O
543	UJF	informatique	N
678	UCB	histoire	O
765	UCB	histoire	O
765	UJM	histoire	N
765	UJM	psychologie	O
876	UCB	informatique	N
876	UJF	biologie	O
876	UJF	biologie marine	N
898	INSA	informatique	O
898	UCB	informatique	O

Table Candidature

Les informations sur les universités, triées par effectif croissant

SELECT *
FROM Université
ORDER BY effectif **ASC** ;

nomU	ville	effectif
UJF	Grenoble	10000
UCB	Lyon	15000
UJM	Saint-Étienne	21000
INSA	Lyon	36000

Exemple de tri (2)

idE	nomE	moyenneLycée	effectifLycée
123	Ana	19.5	1000
234	Bob	18	1500
345	Chloé	17.5	500
456	Damien	19.5	1000
543	Chloé	17	2000
567	Éléonore	14.5	2000
654	Ana	19.5	1000
678	Farid	19	200
765	Joana	14.5	1500
789	Gisèle	17	800
876	Irène	19.5	400
898	Hector	18.5	800

Table Élève

nomU	ville	effectif
INSA	Lyon	36000
UCB	Lyon	15000
UJF	Grenoble	10000
UJM	Saint-Étienne	21000

Table Université

idE	nomU	département	décision
123	INSA	informatique	O
123	UCB	électronique	N
123	UCB	informatique	O
123	UJM	électronique	O
234	INSA	biologie	N
345	UJF	bioinformatique	O
345	UJM	bioinformatique	N
345	UJM	électronique	N
345	UJM	informatique	O
543	UJF	informatique	N
678	UCB	histoire	O
765	UCB	histoire	O
765	UJM	histoire	N
765	UJM	psychologie	O
876	UCB	informatique	N
876	UJF	biologie	O
876	UJF	biologie marine	N
898	INSA	informatique	O
898	UCB	informatique	O

Table Candidature

Les informations sur les universités de plus de 12000 étudiant-e-s, avec un tri par ville décroissante puis par effectif croissant

SELECT *
FROM Université
WHERE effectif $>$ 12000
ORDER BY ville **DESC**, effectif **ASC** ;

nomU	ville	effectif
UJM	Saint-Étienne	21000
UCB	Lyon	15000
INSA	Lyon	36000

Syntaxe de la limitation

Clause LIMIT pour conserver un nombre restreint de résultats :

SELECT att_1, att_2, ...
FROM *nom_table*
WHERE *condition*
ORDER BY att_i, att_j, ...
LIMIT n ;

- Les n premières instances (après le tri) sont conservés dans le résultat
- Pas un top-K, qui récupère toutes les instances avec les n meilleures valeurs

Dans la norme SQL, la syntaxe est FETCH FIRST n ROWS ONLY

Exemple de limitation

idE	nomE	moyenneLycée	effectifLycée
123	Ana	19.5	1000
234	Bob	18	1500
345	Chloé	17.5	500
456	Damien	19.5	1000
543	Chloé	17	2000
567	Éléonore	14.5	2000
654	Ana	19.5	1000
678	Farid	19	200
765	Joana	14.5	1500
789	Gisèle	17	800
876	Irène	19.5	400
898	Hector	18.5	800

Table Élève

nomU	ville	effectif
INSA	Lyon	36000
UCB	Lyon	15000
UJF	Grenoble	10000
UJM	Saint-Étienne	21000

Table Université

idE	nomU	département	décision
123	INSA	informatique	O
123	UCB	électronique	N
123	UCB	informatique	O
123	UJM	électronique	O
234	INSA	biologie	N
345	UJF	bioinformatique	O
345	UJM	bioinformatique	N
345	UJM	électronique	N
345	UJM	informatique	O
543	UJF	informatique	N
678	UCB	histoire	O
765	UCB	histoire	O
765	UJM	histoire	N
765	UJM	psychologie	O
876	UCB	informatique	N
876	UJF	biologie	O
876	UJF	biologie marine	N
898	INSA	informatique	O
898	UCB	informatique	O

Table Candidature

Les deux universités qui ont les effectifs les plus élevés

SELECT nomU, effectif
FROM Université
ORDER BY effectif **DESC**
LIMIT 2 ;

nomU	effectif
INSA	36000
UJM	21000

Plan

Pourquoi des fonctions et opérateurs ?

- Manipuler les chaînes de caractères (e.g., recherche de sous-chaine, concaténation)
- Réaliser des calculs mathématiques (e.g., somme, division, moyenne)
- Manipuler des dates (e.g., différence entre dates, formatage)
- Convertir entre types de données (e.g., caractère vers entier)

Se référer à la documentation spécifique au SGBD utilisé !

---

http://www.postgresql.org/docs/current/functions.html
http://mariadb.com/kb/en/mariadb/functions-and-operators/

Comparaison approximative

- *s* LIKE *pat* compare approximativement la chaîne *s* selon le pattern *pat*
- Deux caractères joker pour le pattern *pat* :
  - '_' pour un caractère alphanumérique quelconque
  - '%' pour une chaîne alphanumérique quelconque
- Fonctions plus complexes avec des expression régulières

Exemples de patterns :
'a_ _' ≡ toutes les chaînes de 3 caractères commençant par un 'a'
'a%' ≡ toutes les chaînes commençant par un 'a'
'_n_' ≡ toutes les chaînes de 3 caractères avec un 'n' en deuxième position

---

http://www.postgresql.org/docs/current/functions-matching.html

Exemple de comparaison approximative

idE	nomE	moyenneLycée	effectifLycée
123	Ana	19.5	1000
234	Bob	18	1500
345	Chloé	17.5	500
456	Damien	19.5	1000
543	Chloé	17	2000
567	Éléonore	14.5	2000
654	Ana	19.5	1000
678	Farid	19	200
765	Joana	14.5	1500
789	Gisèle	17	800
876	Irène	19.5	400
898	Hector	18.5	800

Table Élève

nomU	ville	effectif
INSA	Lyon	36000
UCB	Lyon	15000
UJF	Grenoble	10000
UJM	Saint-Étienne	21000

Table Université

idE	nomU	département	décision
123	INSA	informatique	O
123	UCB	électronique	N
123	UCB	informatique	O
123	UJM	électronique	O
234	INSA	biologie	N
345	UJF	bioinformatique	O
345	UJM	bioinformatique	N
345	UJM	électronique	N
345	UJM	informatique	O
543	UJF	informatique	N
678	UCB	histoire	O
765	UCB	histoire	O
765	UJM	histoire	N
765	UJM	psychologie	O
876	UCB	informatique	N
876	UJF	biologie	O
876	UJF	biologie marine	N
898	INSA	informatique	O
898	UCB	informatique	O

Table Candidature

Le nom des élèves qui contient un *'e'*

SELECT nomE
FROM Élève
WHERE nomE **LIKE** '%e%';

nomE
Chloe
Hector
Damien
Eleonore
Gisele
Chloe
Irene

D'autres fonctions

- CONCAT(s_1, s_2, ...) concatène les chaînes s_1, s_2, ...
- LENGTH(s) retourne la longueur de la chaîne s
- SUBSTR(s, *pos*, *len*) extrait de s la sous-chaîne qui début à l'index *pos* et de longueur *len*
- TRUNC(n, *dec*) tronque le nombre n à *dec* décimales
- CURRENT_DATE() au format 'YYYY-MM-DD' et NOW()
- d_1 - d_2 retourne le nombre de jours entre les dates d_1 et d_2
- ...

---

http://www.postgresql.org/docs/current/functions-string.html

http://www.postgresql.org/docs/current/functions-datetime.html

Plan

Opérateurs ensemblistes

Comment utiliser les opérateurs ensemblistes en SQL ?

- Permettent de combiner les résultats de plusieurs SELECT
- Pas de doublons (DISTINCT implicite)
- Les SELECT doivent contenir le même nombre d'attributs
- Les noms des attributs du résultat sont ceux du premier SELECT
  - c'est l'ordre des attributs qui compte
- Un opérateur ensembliste peut inclure un ORDER BY et un LIMIT de manière globale
  - dans ce cas, les deux SELECT sont mis entre parenthèses

Opérateurs ensemblistes (2)

Dans la norme SQL et sous PostgreSQL :

- $\cup$: UNION
- $\cap$: INTERSECT
- $-$: EXCEPT
- $\div$: division non implémentée (possible de la faire avec des différences, ou avec des regroupements, ou avec une double négation)

https://www.postgresql.org/docs/current/queries-union.html

Opérateur union

(**SELECT** att_1, ..., att_n **FROM** ...)
UNION
(**SELECT** att_1, ..., att_n **FROM** ...)
[**ORDER BY** att_i [, att_j, ...]]
[**LIMIT** n] ;

- Deux requêtes SELECT reliées par un opérateur UNION
- Les attributs att_i et att_j sont inclus dans att_1, ..., att_n

Exemple d'union

idE	nomE	moyenneLycée	effectifLycée
123	Ana	19.5	1000
234	Bob	18	1500
345	Chloé	17.5	500
456	Damien	19.5	1000
543	Chloé	17	2000
567	Éléonore	14.5	2000
654	Ana	19.5	1000
678	Farid	19	200
765	Joana	14.5	1500
789	Gisèle	17	800
876	Irène	19.5	400
898	Hector	18.5	800

Table Élève

nomU	ville	effectif
INSA	Lyon	36000
UCB	Lyon	15000
UJF	Grenoble	10000
UJM	Saint-Étienne	21000

Table Université

idE	nomU	département	décision
123	INSA	informatique	O
123	UCB	électronique	N
123	UCB	informatique	O
123	UJM	électronique	O
234	INSA	biologie	N
345	UJF	bioinformatique	O
345	UJM	bioinformatique	N
345	UJM	électronique	N
345	UJM	informatique	O
543	UJF	informatique	N
678	UCB	histoire	O
765	UCB	histoire	O
765	UJM	histoire	N
765	UJM	psychologie	O
876	UCB	informatique	N
876	UJF	biologie	O
876	UJF	biologie marine	N
898	INSA	informatique	O
898	UCB	informatique	O

Table Candidature

Le nom et identifiant des élèves qui ont plus de 18 de moyenne ou qui sont accepté.e.s en informatique

(**SELECT** *E*.idE, nomE **FROM** *E* **WHERE** moyenneLycée $>$ 18)
UNION
(**SELECT** *E*.idE, nomE **FROM** *E* **NATURAL JOIN** *C*
WHERE décision = 'O' **AND** département = 'informatique') ;

idE	nomE
123	Ana
678	Farid
456	Damien
876	Irene
654	Ana
345	Chloe
898	Hector

En résumé

- Jointure (interne) avec INNER JOIN ... ON
- Clauses ORDER BY et LIMIT
- Consulter la documentation pour les fonctions
- Possible d'utiliser des sous-requêtes à la place des opérations ensemblistes