Bases de données

Plan

Éléments de cours

- Tables (ou relation) ☐
- Colonnes (ou attributs) ☐
- Lignes (ou enregistrements) ☐
- Clé ☐
- Clé primaire ☐
- Entité et association ☐
- Clé étrangère ☐
- Projeter : `SELECT ... FROM` ☐
- Filtrer : `WHERE` ☐
- Ordonner : `ORDER BY` ☐
- Renommer : `AS` ☐
- Éviter les doublons : `DISTINCT` ☐
- Limiter le nombre de résultats : `LIMIT` et `OFFSET` ☐
- Calculer une propriété globale sur une table : `MIN`, `MAX`, `SUM`, `AVG` ou `COUNT` ☐
- Grouper ces informations : `GROUP BY` ☐
- Filtrer les agrégats : `HAVING` ☐
- Combiner les lignes de tables : `UNION`, `INTERSECT` et `EXCEPT` ☐
- Coupler les colonnes de tables : produit cartésien ☐
- Lever une ambiguïté sur le nom des attributs ☐
- Concaténer des informations : `JOIN ... ON` ☐
- Autojointures ☐

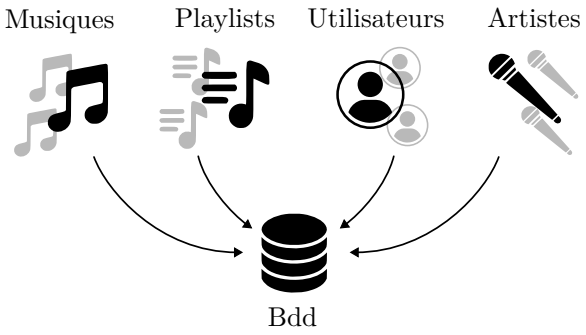
Capacités exigibles

- Vocabulaire des bases de données : tables ou relations, attributs ou colonnes, domaine, schéma de tables, enregistrements ou lignes, types de données.
- Clé primaire.
- Entités et associations, clé étrangère : cas concrets d'associations $1 - 1$, $1 - *$, $* - *$. Séparation d'une association $* - *$ en deux associations $1 - *$.
- Requêtes `SELECT` avec simple clause `WHERE` (sélection), projection, renommage `AS` : opérateurs `+`, `-`, `*`, `/`, `=`, `<>`, `<`, `<=`, `>`, `>=`, `AND`, `OR`, `NOT`.
- Utilisation des mots-clés `DISTINCT`, `LIMIT`, `OFFSET`, `ORDER BY`.
- Opérateurs ensemblistes `UNION`, `INTERSECT` et `EXCEPT`, produit cartésien.
- Jointures internes $\texttt{T}_1$ `JOIN` $\texttt{T}_n$ `...` `JOIN` $\texttt{T}_n$ `ON` Φ. Autojointure.
- Agrégation avec les fonctions `MIN`, `MAX`, `SUM`, `AVG` et `COUNT`, y compris avec `GROUP BY`.
- Filtrage des agrégats avec `HAVING` : remarquer la différence avec `WHERE`.

Intérêt et motivation

Il existe aujourd'hui de très nombreux sites internet stockant du contenu : Netflix, Spotify, Instagram, Marmitton, Air Bnb, Blablacar... Chaque plateforme a besoin d'enregistrer des données, dans ce qu'on appelle une base de donnée (Bdd).

Exemple



Une plateforme de musique a besoin de stocker en mémoire bien sûr les informations relatives aux musiques qu'elle contient, mais aussi celles de utilisateurs, des différentes playlists ou encore des artistes (et bien d'autres encore !).

But

L'enjeu est de pouvoir répondre aux problématiques suivantes :

1. Comment minimiser l'espace occupé ?
2. Comment ajouter / modifier / supprimer simplement une donnée ?
3. Comment extraire facilement des données ?

Exemple

Dans le cas de notre plateforme de musique,

1. Une musique stockée dans deux playlists différentes, ne doit pas être simplement dédoublée ! Elle prendrait bêtement deux fois plus de place.
2. Un artiste souhaite changer de nom, il faut modifier toutes ses musiques.
3. Un utilisateur veut faire une recherche par style (rap, rock, pop...)

On peut penser à l'utilisation de tableur :

Titre	Album	Artiste	Année	Style
Can you hear the music	Oppenheimer (OST)	Ludwig Göransson	2023	Neo-Classical
Day Break	Canon	Overwerk	2015	Electro-symphonic
L'odeur de l'essence	Civilisation	Orelsan	2021	French Rap
Knigths of Cydonia	Black Holes and Revelations	Muse	2006	Pop Rock
Touch	Random Access Memories	Daft Punk	2013	French Touch

Mais pour répondre aux exigences citées, on voit tout de suite les limites de cette solution.

Le moyen de plus efficace pour traiter des données est l'utilisation de **bases de données relationnelles**, avec lesquelles on peut interagir via le langage SQL (Structured Query Language)

I Structure d'une base de données relationnelle

A Tables, colonnes et lignes

Définition : Tables (ou relation)

Une base de données peut contenir plusieurs **tables** (ou **relations**), stockant des objets de même nature.

Définition : Colonnes (ou attributs)

Une table contient des **colonnes** (ou **attributs**) indiquant les types de données à stocker pour chaque objet.

Définition : Lignes (ou enregistrements)

Une table contient aussi des **lignes** (ou **enregistrements**), représentant chacune un objet donné et les informations (attributs) qui lui sont relatives.

✔ Exemple

Une plateforme de covoiturage peut avoir une table stockant les trajets renseignés et une autre pour les comptes enregistrés :

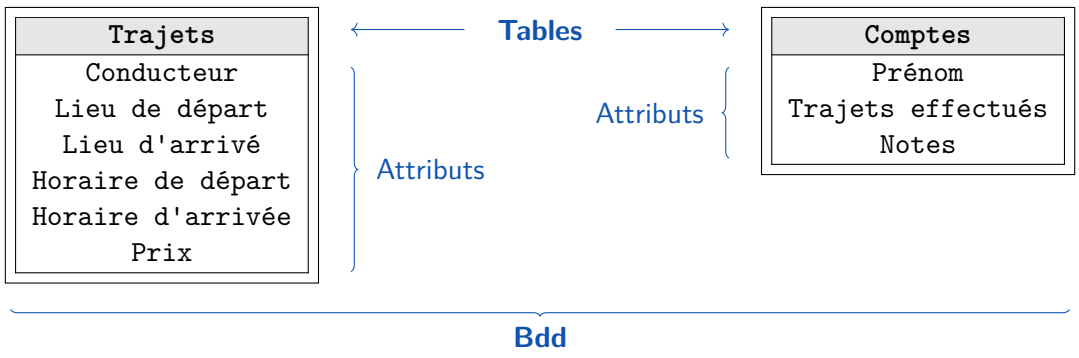
Table des trajets

Conducteur	Lieu de départ	Lieu d'arrivée	Horaire de départ	Horaire d'arrivée	Prix
Sasha	Lyon	Paris	10h30	15h00	32€
Sam	Marseille	Montpellier	16h00	17h50	15€
Alix	Nantes	Brest	14h20	17h30	28€

Table des comptes

Prénom	Trajets effectués	Notes
Camille	14	4.8/5
Sofiane	25	4.5/5
Charlie	7	4.1/5

Finalement, la structure de la base de données se résume ainsi :



B Clé et clé primaire

Définition : Clé

Dans une table donnée, une clé est un attribut ou un groupe d'attributs permettant d'identifier **de manière unique** une ligne particulière.

Définition : Clé primaire

Parmi les clés possibles d'une table donnée, on en choisit une qu'on appellera alors **clé primaire**.

Exemple

En France, l'assurance maladie enregistre les personnes ayant droit à une protection sociale.

Nom	Prénom	Date de naissance	Département de naissance	Numéro de sécurité sociale
Michel	David	01/04/1975	44	1 75 04 44 429 781 27
Michel	David	01/04/1975	57	1 75 04 57 398 174 49
Michel	David	18/10/1987	44	1 87 10 44 105 865 53

1. Pour l'exemple ci-dessus, dire si les groupes d'attributs suivants constituent des clés ou non :
   - {Nom, Prénom}
   - {Nom, Prénom, Date de naissance}
   - {Nom, Prénom, Date de naissance, Département de naissance}
   - {Numéro de sécurité sociale}
2. Quel choix de clé primaire feriez-vous ?

Remarque

En pratique on préfère quasi-systématiquement créer un attribut spécifiquement voué à servir de clé primaire. Il contient uniquement un entier, s'incrémentant à chaque nouvel enregistrement.

Exemple

Dans un site de recettes de cuisine, on peut imaginer la table suivante :

id	Recette	Temps	Difficulté
0	Taboulé à la libanaise	30 min	Facile
1	Forêt noire	1h30	Moyenne
2	Kouign Amann	2h	Difficile
3	Fondue de poireaux	1h	Facile

Ici clairement, l'attribut **id** va servir de clé primaire.

C Modèle entité-association

Définition : Entité et association

Une **entité** d'une table (ou enregistrement) peut être mise en relation avec d'autres lignes d'une seconde table. On distingue trois cas possibles d'**associations** :

`1-1` : association 1-1 (*one to one*) ;

`1-*` : association 1-plusieurs (*one to many*) ;

`*-*` : association plusieurs-plusieurs (*many to many*) ;

Définition : Clé étrangère

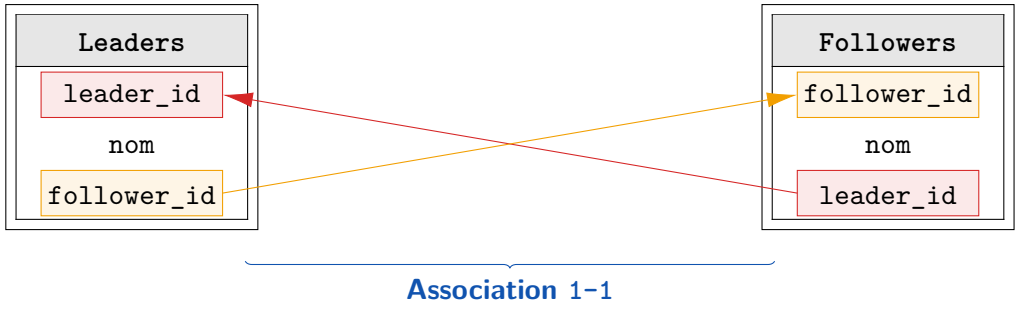
Une clé étrangère d'une table est un attribut permettant d'identifier de **manière unique** une ligne d'une autre table.

Remarque

- On utilise quasi-systématiquement la clé primaire de la seconde table comme clé étrangère de la première.
- Une association `*-*` se décompose en pratique par plusieurs relations `1-*`. Ces relations sont elles-mêmes stockées dans une troisième table faisant la jonction entre les deux.

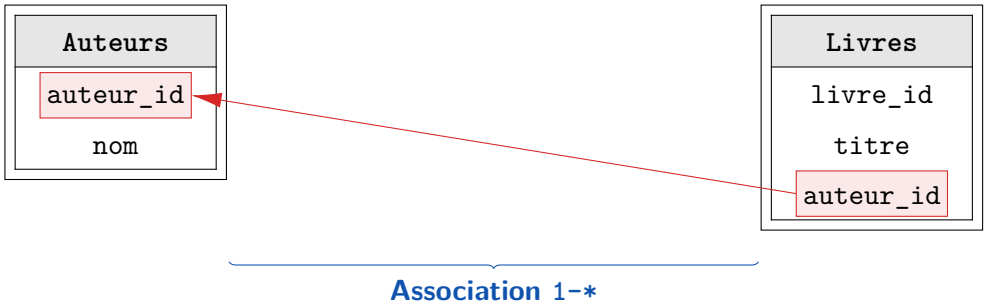
Exemple

- Dans une compétition de danse en couple, chaque participant · e peut guider (*leader*) ou bien suivre (*follower*). On peut donc associer les *leader* avec les *follower* dans une relation `1-1`



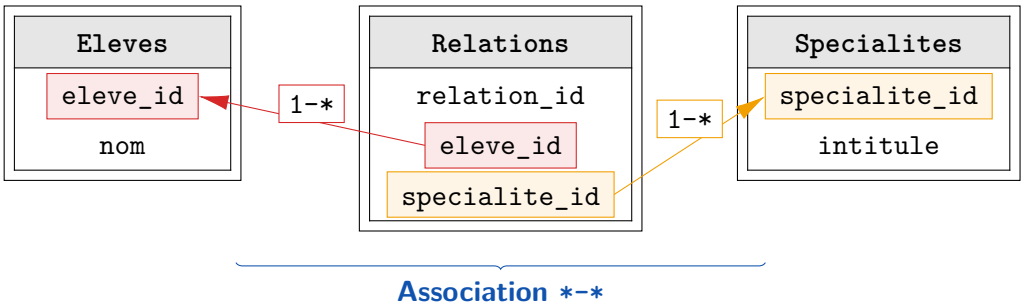
Ici la clé étrangère d'un *leader* permettant d'identifier un *follower* est `follower_id` (qui est aussi la clé primaire des *followers*).

- Une librairie utilise une base de données, contenant (entre autres) une table pour les écrivain · es, et une autre pour les livres. Chaque auteur · e peut avoir écrit plusieurs livres (mais chaque livre n'est écrit que par une personne). On peut donc associer ces tables dans une relation `1-*` :



  La clé étrangère d'un livre permettant d'identifier son auteur est `auteur_id`.

- On stocke les spécialités choisies par des élèves d'une classe de première générale. Chacun · e peut en choisir plusieurs ; et réciproquement, chaque matière peut être sélectionnées par plusieurs élèves. On associe donc les élèves et les spécialités par une relation `*-*` :



- Pour des exemples plus concrets, voir la partie sur les jointures (sous-partie C).

Remarque

Dans les exemples précédents, les clés étrangères ont les mêmes noms que les attributs correspondant dans la table ciblée. Ceci est une recommandation par souci de clarté, mais ce n'est pas une condition nécessaire.

II Extraire des données d'une table en SQL

Dans cette section on prendra l'exemple d'une base de données contenant une table des résultats de participants à une course :

Table : resultats

dossard	nom	prenom	categorie	temps
1	Kadiri	Naïm	M	66
2	Rivard	Benjamin	M	58
3	Binet	Joanna	F	64
4	Vaskova	Marketa	F	69
5	Grondin	Zoé	F	55
6	Souplet	Alexis	M	79
7	Charpentier	Pierre	M	75
8	Cotuand	Francis	M	81
9	Fouquet	Aline	F	78
10	Arana Baez	Elea	F	78

A Fonctions de base : `SELECT ... FROM ... WHERE ... ORDER BY`

Projeter : `SELECT ... FROM` ♥

Pour extraire les informations qui nous intéressent pour chaque objet d'une table, on utilise les syntaxes suivantes :

- Sélectionner un attribut d'une table

```
SELECT attribut
FROM table
```

- Sélectionner plusieurs attributs d'une table

```
SELECT attribut1, attribut2 ...
FROM table
```

- Sélectionner tous les attributs d'une table

```
SELECT *
FROM table
```

Filtrer : `WHERE` ♥

Pour sélectionner uniquement certaines lignes de la table, on peut ajouter une condition après le mot-clé `WHERE` :

```
SELECT attributs
FROM table
WHERE condition
```

La condition peut faire intervenir

- des noms de colonnes ;
- des opérateurs mathématiques : `+`, `-`, `*`, `/` ;
- des opérateurs de comparaison : `=`, `<>` (pour $\neq$), `<`, `<=`, `>`, `>=` ;
- des opérateurs logiques : `AND`, `OR`, `NOT` ;

Ordonner : ORDER BY ♥

On peut ensuite ordonner les lignes selon un ou plusieurs attributs grâce au mot-clé ORDER BY :

```
SELECT attributs
FROM table
WHERE condition
ORDER BY attributs
```

⚠ Il y a bien un espace entre ORDER et BY !

✔ Exemple

› On peut extraire les participantes ayant en les triant par ordre croissant de temps, et par ordre alphabétique des noms en cas d'égalité :

```
SELECT nom, temps
FROM resultats
WHERE categorie == F
ORDER BY temps, nom
```

⟶ résultat

```
+-------------+-------+
| nom         | temps |
+-------------+-------+
| Grondin     | 55    |
| Binet       | 64    |
| Vaskova     | 69    |
| Arana Baez  | 78    |
| Fouquet     | 78    |
+-------------+-------+
```

› On peut extraire les participants ayant mis entre 1h10 et 1h20 et les trier par catégorie :

```
SELECT *
FROM resultats
WHERE temps > 70 AND temps < 80
ORDER BY categorie
```

↓ résultat

```
+---------+-------------+----------+-----------+-------+
| dossard | nom         | prenom   | categorie | temps |
+---------+-------------+----------+-----------+-------+
| 9       | Fouquet     | Aline    | F         | 78    |
| 10      | Arana Baez  | Elea     | F         | 78    |
| 6       | Souplet     | Alexis   | M         | 79    |
| 7       | Charpentier | Pierre   | M         | 75    |
+---------+-------------+----------+-----------+-------+
```

B Options supplémentaires : AS, DISTINCT, LIMIT et OFFSET

Renommer : AS ♥

Par souci de clarté, on peut renommer les attributs ou bien les tables sélectionnés avec le mot-clé AS :

- Renommer un attribut :

```
SELECT attribut1 AS mon_nom_attribut1, ...
FROM table
```

- Renommer une table :

```
SELECT attributs
FROM table AS mon_nom_table
```

Remarque

Renommer une table ne sert à rien tant qu'on ne travaille que sur une seule. L'utilité de cette fonction apparaîtra lors de l'étude des jonctions (cf. partie III).

Éviter les doublons : DISTINCT ♥

Parfois, plusieurs lignes peuvent être identiques. Pour retirer tous les doublons, on ajoute le mot-clé DISTINCT :

```
SELECT DISTINCT attributs
FROM table
```

Limiter le nombre de résultats : LIMIT et OFFSET ♥

- Pour limiter le nombre de lignes renvoyées, on utilise le mot-clé LIMIT :

```
SELECT attributs
FROM table
LIMIT nombre_resultats
```

- Pour ignorer les premiers résultats, on utilise le mot-clé OFFSET :

```
SELECT attributs
FROM table
OFFSET nombre_lignes_ignorées
```

Remarque

Il faut respecter un ordre dans la syntaxe. En l'occurrence ici :
`SELECT ... FROM ... WHERE ... ORDER BY ... LIMIT ... OFFSET`

Exemple

› On souhaite afficher les résultats effectués par les participants, en retirant les doublons (deux participantes on mis 78 min) :

```
SELECT DISTINCT temps AS Résutat
FROM resultats
ORDER BY Résultat
```

résultat →

```
+-----------+
| Résultats |
+-----------+
| 55        |
| 58        |
| 64        |
| 66        |
| 69        |
| 75        |
| 78        |
| 79        |
| 81        |
+-----------+
```

On remarque que l'on peut réutiliser l'alias `"Résutats"` dans les fonctions suivantes (ici `ORDER BY`).

› On souhaite extraire la liste des 5 participants, en ignorant les 3 premiers :

```
SELECT DISTINCT dossard, temps
FROM resultats
ORDER BY temps
LIMIT 5
OFFSET 3
```

résultat →

```
+---------+-------+
| dossard | temps |
+---------+-------+
| 1       | 66    |
| 4       | 69    |
| 7       | 75    |
| 9       | 78    |
| 10      | 78    |
+---------+-------+
```

On remarque que cette fois-ci, le mot-clé `DISTINCT` ne retire aucune ligne, car les deux temps de 78 sont distinguables grâce à la colonne `dossard`.

C Agrégation de lignes : MIN, MAX, SUM, AVG, COUNT et GROUP BY

Calculer une propriété globale sur une table : MIN, MAX, SUM, AVG ou COUNT ♥

On peut extraire un unique résultat, synthétisant les informations de toutes les lignes sélectionnées grâce aux mots-clés `MIN`, `MAX`, `SUM`, `AVG` ou `COUNT` :

Extraire le minimum d'une colonne :

```
SELECT MIN(attribut)
FROM table
```

Extraire le maximum d'une colonne :

```
SELECT MAX(attribut)
FROM table
```

Extraire la somme d'une colonne :

```
SELECT SUM(attribut)
FROM table
```

Extraire la moyenne d'une colonne :

```
SELECT AVG(attribut)
FROM table
```

Extraire le nombre d'éléments d'une colonne :

```
SELECT COUNT(attribut)
FROM table
```

Remarque

Ces syntaxes renvoient une unique ligne contenant le résultat final. On parle alors d'**agrégation**, puisque toutes les lignes sont agrégées en une seule.

Grouper ces informations : `GROUP BY` ♥

On peut utiliser ces fonctions en créant des groupes parmi les lignes, grâce au mot-clé `GROUP BY` :

```
SELECT FONCTION(attribut_agrégé)
FROM table
GROUP BY attribut_en_commun
```

Exemple

- On veut obtenir le nombre de participants :

```
SELECT COUNT(dossard)
FROM resultats
```

⟶ résultat

```
+----------------+
| COUNT(dossard) |
+----------------+
| 10             |
+----------------+
```

- On cherche à connaître le nombre de coureurs, celui de coureuses, ainsi que les temps moyen de ces deux catégories :

```
SELECT categorie, COUNT(dossard), AVG(temps)
FROM resultats
GROUP BY categorie
```

↓ résultat

```
+-----------+----------------+------------+
| categorie | COUNT(dossard) | AVG(temps) |
+-----------+----------------+------------+
| F         | 5              | 68.8       |
| M         | 5              | 71.8       |
+-----------+----------------+------------+
```

Remarque

- Il est naturel d'ajouter l'attribut choisit avec `GROUP BY` (noté `attribut_en_commun`) dans la fonction `SELECT`, sinon il est difficile de savoir à quoi correspondent les lignes.
- Il n'y a aucun sens à ajouter d'autres sélecteurs que `attribut_en_commun` ou des fonctions d'agrégation. Sinon la colonne prend une valeur particulière, d'une des lignes ayant été agrégée.

Voici un contre-exemple pour illustrer ces deux remarques précédentes :

```
SELECT AVG(temps), nom
FROM resultats
GROUP BY categorie
```

⟶ résultat

```
+------------+--------+
| AVG(temps) | nom    |
+------------+--------+
| 68.8       | Binet  |
| 71.8       | Kadiri |
+------------+--------+
```

Filtrer les agrégats : HAVING ♥

De même qu'avec `WHERE`, on peut ajouter des conditions sur les agrégations en utilisant le mot-clé `HAVING` :

```
SELECT FONCTION(attribut_agrégé)
FROM table
GROUP BY attribut_en_commun
HAVING condition
```

Remarque

- Numériquement, les opérateurs `WHERE` et `HAVING` n'agissent pas sur les mêmes objets :

WHERE	≠	**HAVING**
Agit sur des enregistrements.		Agit sur des agrégats.

- Au niveau de la syntaxe `HAVING` s'utilise exactement de la même manière que `WHERE`, mais peut inclure les fonctions d'agrégation : `MIN`, `MAX`, `SUM`, `AVG` ou `COUNT`.

Exemple

```
SELECT categorie, COUNT(dossard), AVG(temps)
FROM resultats
GROUP BY categorie
HAVING AVG(temps) > 70
```

↓ résultat

```
+-----------+-----------------+-------------+
| categorie | COUNT(dossard) | AVG(temps) |
+-----------+-----------------+-------------+
| M         | 5               | 71.8        |
+-----------+-----------------+-------------+
```

III Extraire des données de plusieurs tables en SQL

A Opérations ensemblistes : UNION, INTERSECT et EXCEPT

Combiner les lignes de tables : UNION, INTERSECT et EXCEPT ♥

Lorsque deux sélections possèdent **le même schéma** (c'est-à-dire lorsqu'elles ont **les mêmes attributs**), on peut les combiner de la manière suivante :

- Obtenir l'union de deux tables :

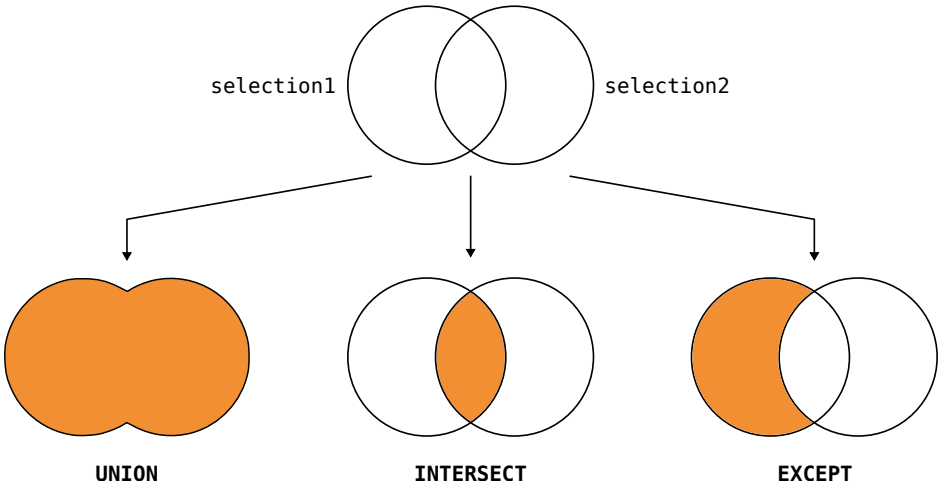
```
SELECT attributs FROM table1
UNION
SELECT attributs FROM table2
```

- Obtenir l'intersection de deux tables :

```
SELECT attributs FROM table1
INTERSECT
SELECT attributs FROM table2
```

- Obtenir la différence de deux tables :

```
SELECT attributs FROM table1
EXCEPT
SELECT attributs FROM table2
```



Remarque

- La différence (avec `EXCEPT`) n'est pas symétrique : on prend les enregistrements de la `selection1` et on leur retire ceux de la `selection2` (cf. schéma ci-dessus).
- Cette syntaxe renvoie une nouvelle sélection, sur laquelle on peut continuer à appliquer des opérateurs ensemblistes :

```
SELECT attributs FROM table1
UNION
SELECT attributs FROM table2
UNION
SELECT attributs FROM table3 ...
```

- Les tables peuvent avoir des schémas différents, mais la sélection doit faire référence aux mêmes attributs dans les deux tables. Sinon on obtient une erreur.
- La fonction `UNION` retire automatiquement les doublons, pas besoin d'utiliser `DISTINCT`.

Exemple

Un site de vente en ligne dispose d'une base de données contenant une table pour les fournisseurs, et d'une table pour les clients :

id	Entreprise	Contact
0	Modern Kitchen	Orlene.Tachel@modernkitchen.com
1	Tune Craft	Thibaut.Chouinard@tunecraft.com
2	Decor Harmony	Raoul.Brault@decorharmony.com
3	Cycle Works	Julius.Rasmussen@cycleworks.com

Fournisseurs

id	Nom	Contact
0	Farah Jawa	Farah.Jawa@chouchou.com
1	Julius Rasmussen	Julius.Rasmussen@cycleworks.com
2	Samuel Cardoso	Samuel.Cardoso@laptcestcool.com
3	Orlene Tachel	Orlene.Tachel@modernkitchen.com

Clients

- On veut récupérer toutes les adresses mails enregistrées sur le site :

```
SELECT Contact FROM Fournisseurs
UNION
SELECT Contact FROM Clients
```

↓ résultat

```
+---------------------------------+
| Contact                         |
+---------------------------------+
| Orlene.Tachel@modernkitchen.com |
| Thibaut.Chouinard@tunecraft.com |
| Raoul.Brault@decorharmony.com   |
| Julius.Rasmussen@cycleworks.com |
| Farah.Jawa@chouchou.com         |
| Samuel.Cardoso@laptcestcool.com |
+---------------------------------+
```

- On veut afficher les contacts des fournisseurs, n'étant pas eux-mêmes clients :

```
SELECT Contact FROM Fournisseurs
EXCEPT
SELECT Contact FROM Clients
```

↓ résultat

```
+---------------------------------+
| Contact                         |
+---------------------------------+
| Thibaut.Chouinard@tunecraft.com |
| Raoul.Brault@decorharmony.com   |
+---------------------------------+
```

- La commande suivante va renvoyer une erreur car on demande de combiner deux sélections incluant des colonnes différentes (Entreprise $\neq$ Nom) :

```
SELECT Contact, Entreprise FROM Fournisseurs
EXCEPT
SELECT Contact, Nom FROM Clients
```

$\longrightarrow$ résultat ERROR

Remarque

Ces opérations ensemblistes permettent de croiser des tables, en couplant leurs **lignes**.

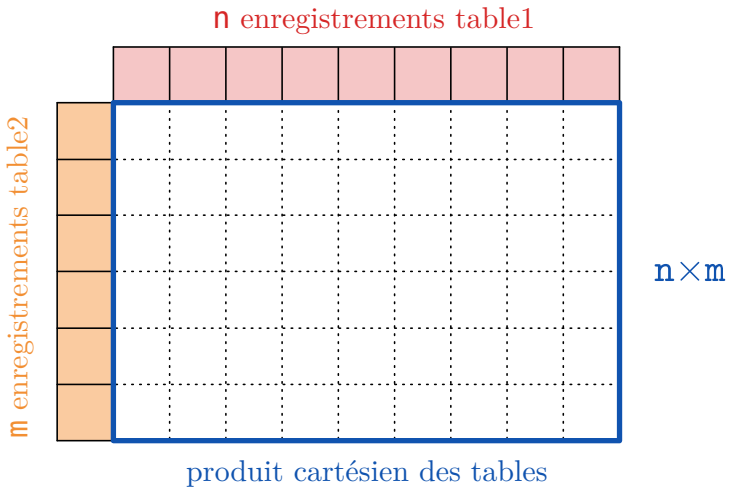
B Produit cartésien

Coupler les colonnes de tables : produit cartésien ♥

On peut aussi coupler deux tables, **par colonnes** avec la syntaxe suivante :

```
SELECT attributs
FROM table1, table2
```

En notant respectivement `n` et `m` les nombres de lignes de `table1` et `table2`, la sélection renvoyée contient `n`×`m` lignes : chaque enregistrement de `table1` est associé à tous les enregistrements de `table2`.



Remarque

Le produit cartésien n'a que peu d'utilité en pratique puisqu'il créer tous les couples possibles entre les lignes des deux tables. Cela n'a souvent pas de sens de construire toutes ces associations.

Lever une ambiguïté sur le nom des attributs ♥

Dans le cas où les tables contiennent des attributs identiques, il faut les différentier dans la commande à l'aide de la syntaxe suivante :

```
SELECT table1.attribut
FROM table1, table2
```

On peut alors en profiter pour renommer les tables pour simplifier l'écriture :

```
SELECT t1.attribut
FROM table1 AS t1, table2
```

C Jointures de tables en SQL : `JOIN ... ON`

Concaténer des informations : `JOIN ... ON`

On peut compléter chaque ligne d'une table en concaténant les informations relatives à ses clés étrangères, en utilisant les mots-clés `JOIN ... ON` :

```
SELECT attributs
FROM table1
JOIN table2
ON table1.clé_etrangère = table2.attribut_correspondant
```

Remarque

- Indiquer le nom des tables suivit d'un point dans la fonction `ON` permet de lever l'ambiguïté dans le cas où un attribut existe dans les deux. Si la clé étrangère et l'attribut correspondant sont les seuls à avoir ce nom, alors cette précision est inutile.
- Il peut être utile de renommer
  - les tables, afin d'alléger la syntaxe ;
  - les attributs, afin de ne pas avoir deux colonnes portant le même nom.

Exemple

La base de données d'une plateforme de musique possède une table pour les titres et une pour les artistes. Chaque musique n'est associée qu'à un artiste (mais un artiste peut avoir écrit plusieurs musiques) : relation 1−∗ ;

id	titre	artiste_id
0	Bohemian Rhapsody	1
1	Another One Bites The Dust	1
2	Dancing Queen	0
3	Mamma Mia	0

musiques

id	nom
0	Abba
1	Queen

artistes

- Pour joindre à chaque musique son artiste, on implémente :

```
SELECT *
FROM musiques
JOIN artistes
ON musiques.artiste_id = artistes.id
```

résultat

id	titre	artiste_id	id	nom
0	Bohemian Rhapsody	1	1	Queen
1	Another One Bites The Dust	1	1	Queen
2	Dancing Queen	0	0	ABBA
3	Mamma Mia	0	0	ABBA

table : `musiques` table : `artistes`

Dans cette sélection, on voit bien l'utilité de renommer les attributs : deux colonnes ont le même nom `id`, alors qu'elles désignent des objets totalement différents.

› Pour afficher les titres et les artistes des musiques, on précise les colonnes souhaitées dans `SELECT` :

```
SELECT titre AS Musique, nom AS Artiste
FROM musiques
JOIN artistes
ON musiques.artiste_id = artistes.id
```

↓ résultat

```
+----------------------------+---------+
| Musique                    | Artiste |
+----------------------------+---------+
| Bohemian Rhapsody          | Queen   |
| Another One Bites The Dust | Queen   |
| Dancing Queen              | ABBA    |
| Mamma Mia                  | ABBA    |
+----------------------------+---------+
```

Autojointures ♥

On peut également concaténer une table avec elle même. Mais il devient alors indispensable de renommer chaque référence à cette table, de la fonction `ON` :

```
SELECT attributs
FROM table AS table1
JOIN table AS table2
ON table1.attribut1 = table2.attribut2
```

✔ Exemple

Une entreprise stocke des informations sur ses employés et notamment, les liens de subordinations qui les relie (pour une personne donnée, qui est son chef direct) :

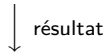
id	nom	chef_id
0	Albert Denis	0
1	Hua Su	0
2	Laila Kristiansen	1
3	Bruno Mancini	1
4	Terence Patel	2
5	Aceline Paré	3

employes

NB : Le PDG "Albert Denis" est considéré comme son propre chef.

On peut alors extraire afficher chaque employé et le nom de son chef direct :

```
SELECT salaries.nom AS Salarié, chefs.nom AS "Chef direct"
FROM employes as salaries
FROM employes as chefs
JOIN salaries.chef_id = chefs.id
```




```
+--------------------+--------------------+
| Salarié            | Chef direct        |
+--------------------+--------------------+
| Albert Denis       | Albert Denis       |
| Hua Su             | Albert Denis       |
| Laila Kristiansen  | Hua Su             |
| Bruno Mancini      | Hua Su             |
| Terence Patel      | Laila Kristiansen  |
| Aceline Paré       | Bruno Mancini      |
+--------------------+--------------------+
```