

Gestion des données

Manipuler les données en SQL

HEI 2021 / 2022

Rappel

- La dernière fois, nous avons vu comment définir la structure d'une base de données :
 - Création de tables
 - Création de relations avec les clés étrangères
 - LDD : Langage de définition des données
- La finalité est de manipuler des données concrètes.
 - LMD : langage de manipulation des données

Les différents types de requêtes

- Les différentes actions de manipulation des données sont les suivantes :
 - Créer de nouveaux objets (Create)
 - Lire les objets existants (Read)
 - Modifier des objets existants (Update)
 - Supprimer des objets existants (Delete)
- Différentes requêtes SQL vont permettre de réaliser ces actions.

Création de nouveaux objets

La requête INSERT INTO

Liste des colonnes

- Dans la suite du cours on suppose existante la table person suivante :

```
CREATE TABLE person (  
  id          INT AUTO_INCREMENT PRIMARY KEY,  
  nom         VARCHAR (50) NOT NULL,  
  prenom      VARCHAR (50) NOT NULL,  
  date_naissance DATE NULL);
```

INSERT INTO

- La requête **INSERT INTO** permet d'ajouter une nouvelle ligne dans une table.
- Sa syntaxe est la suivante :

INSERT INTO Nom de la table **person** (Nom des colonnes **id, nom, prenom, date_naissance**)
VALUES (Valeurs des attributs **1, 'DUPONT', 'Aurélie', str_to_date('1995-09-13', '%Y-%m-%d')**)

Liste des colonnes

- Il n'est pas obligatoire d'indiquer toutes les colonnes de la table au moment de l'insertion des données.
- Les colonnes non renseignées auront pour valeur :
 - Leur valeur par défaut si elle existe
 - Une valeur générée automatiquement si colonne en AUTO_INCREMENT
 - La valeur « null » sinon

Liste des colonnes exemple

- Il n'est pas obligatoire d'indiquer toutes les colonnes de la table au moment de l'insertion des données.
- **INSERT INTO** person (nom, prenom) **VALUES**
 ('DUPONT', 'Aurélie') ,
 ('DUPONT', 'Thomas');
- Ici le champ date_naissance prendra la valeur NULL et id sera généré avec la valeur max(id+1)

Ajouter plusieurs lignes

- Il est possible d'insérer plusieurs lignes en une seule requête en précisant plusieurs ensemble de valeurs, séparés par une virgule.

```
INSERT INTO person(nom, prenom, date_naissance)  
VALUES  
    ('DUPONT', 'Aurélie', '1995-09-13'),  
    ('DUPONT', 'Thomas', '1996-03-17');
```

Écriture simplifiée

- Il est possible d'écrire une requête d'insert sans préciser le nom des colonnes :
- **INSERT INTO** person **VALUES**
 (1, 'DUPONT', 'Aurélie', '1995-09-13') ,
 (2, 'DUPONT', 'Thomas', '1996-03-17');
- /!\ : bien respecter l'ordre des colonnes (les même que lors du create table)

Insertion depuis un résultat

- Il est possible d'écrire une requête d'insert à partir du résultat d'un SELECT

```
INSERT INTO person (nom, prenom) SELECT nom, prenom  
FROM otherPerson;
```

Insertions

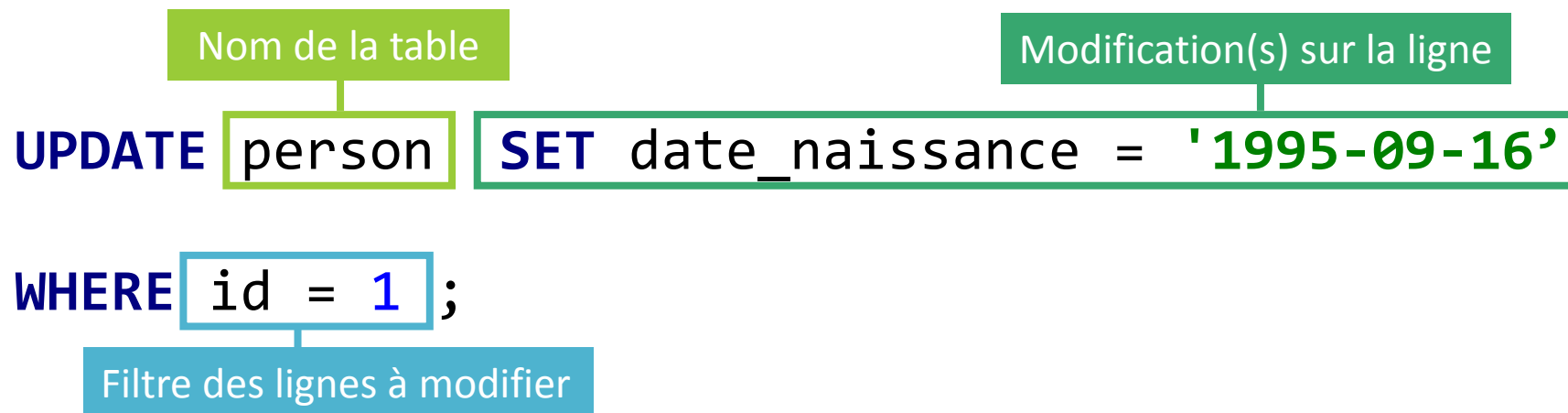
- Attention aux contraintes lors des insertions:
 - Clés primaires et contrainte UNIQUE (pas de doublons)
 - Clés étrangères : respecter l'ordre des insert (la valeur référencée doit exister)
 - Contrainte NOT NULL

Modification des données

La requête UPDATE

UPDATE

- La requête **UPDATE** permet de modifier les informations d'une ou plusieurs lignes.
- Sa syntaxe est la suivante :



The diagram illustrates the SQL UPDATE syntax with three components highlighted by colored boxes and labels:

- Nom de la table** (green box) points to the `person` table name in the `UPDATE person` clause.
- Modification(s) sur la ligne** (green box) points to the `SET date_naissance = '1995-09-16'` clause.
- Filtre des lignes à modifier** (blue box) points to the `WHERE id = 1` clause.

```
UPDATE person SET date_naissance = '1995-09-16' WHERE id = 1;
```

Mise à jour de plusieurs colonnes

- Pour mettre à jour plusieurs colonnes, il suffit de lister les différentes colonnes avec leur nouvelle valeur, séparées par des caractères « , ».

```
UPDATE person
  SET nom = UPPER(nom) ,
      date_naissance = '1995-09-16'
 WHERE prenom = 'Aurélie';
```

Suppression de données

La requête DELETE

DELETE

- La requête **DELETE** permet de supprimer une ou plusieurs lignes.
- Sa syntaxe est la suivante :

DELETE FROM Nom de la table **WHERE** Filtre des lignes à supprimer ;



- /!\ Si la clause **WHERE** n'est pas spécifiée toutes les données de la table seront supprimées !

Répercussion

- Il est possible de préciser l'action à effectuer lors de la suppression :

```
ALTER table nom_table ADD CONSTRAINT fk_col_ref
```

```
FOREIGN KEY (colonne)
```

```
REFERENCES table_ref(col_ref)
```

```
ON {DELETE/UPDATE} {RESTRICT/NO ACTION/SET NULL/CASCADE};
```

RESTRICT/NO ACTION : comportement par défaut (génère une erreur)

SET NULL : positionne la fk à NULL

CASCADE : supprime/met à jour les lignes liées

Lecture des données

La requête SELECT

SELECT

- La requête SELECT permet de lire les données d'une table.
- Sa syntaxe est la suivante :

SELECT nom, prenom **FROM** person ;

Nom de la table

Informations à récupérer



	nom	prenom
▶	DUPONT	Aurélie
	DUPONT	Thomas

SELECT *

- Il est possible de remplacer la liste des colonnes à récupérer par une « * » pour récupérer toutes les colonnes :

SELECT * FROM person ;




	id	nom	prenom	date_naissance
▶	1	DUPONT	Aurélie	1995-09-13
	2	DUPONT	Thomas	1996-03-17

Clause WHERE

- Le mot clé **WHERE** permet d'introduire un filtre à notre requête de lecture. Toutes les lignes pour lesquelles la condition indiquée est évaluée à « true » sont renvoyées.

```
SELECT * FROM person WHERE prenom='Thomas'
```



	id	nom	prenom	date_naissance
▶	2	DUPONT	Thomas	1996-03-17

Combiner les conditions

- Les mots clés **AND** et **OR** permettent de combiner plusieurs conditions. Les règles habituelles de l'algèbre booléenne s'appliquent.

```
SELECT * FROM person
  WHERE prenom = 'Thomas' AND 2000 < YEAR(date_naissance)
      OR YEAR(date_naissance) <= 2000;
```

```
SELECT * FROM person
  WHERE prenom = 'Thomas'
      AND ( 2000 < YEAR(date_naissance) OR YEAR(date_naissance) <= 2000 );
```

Comparaison de 2 colonnes

- Il n'est pas seulement possible de comparer la valeur d'une colonne avec une valeur fixe, il est également possible de comparer directement 2 colonnes entre elles.

```
SELECT * FROM person WHERE prenom < nom
```



	id	nom	prenom	date_naissance
▶	1	DUPONT	Aurélie	1995-09-13

Opérateurs de comparaison

Opérateurs de comparaison


- Les opérateurs de comparaison classiques existent en SQL :

Type	Opérateur	Exemple
Egalité	=	prenom = 'Thomas'
Inégalité	!=	prenom != 'Thomas'
	<>	prenom <> 'Thomas'
Comparaison	<	age > 50
	<=	date_naissance < '2000-01-01'
	>	prenom <= 'M'
	>=	
Nullité	IS NULL	date_naissance IS NULL
Non Nullité	IS NOT NULL	date_naissance IS NOT NULL

Négation

- L'opérateur NOT permet d'inverser une comparaison.

```
SELECT * FROM person WHERE NOT prenom='Thomas'
```



	id	nom	prenom	date_naissance
	1	DUPONT	Aurélie	1995-09-13

Opérateur BETWEEN

- L'opérateur BETWEEN permet de vérifier qu'une valeur est comprise dans un intervalle. Les deux bornes sont incluses.

```
SELECT * FROM person  
WHERE YEAR(date_naissance) BETWEEN 1995 AND 2000;
```

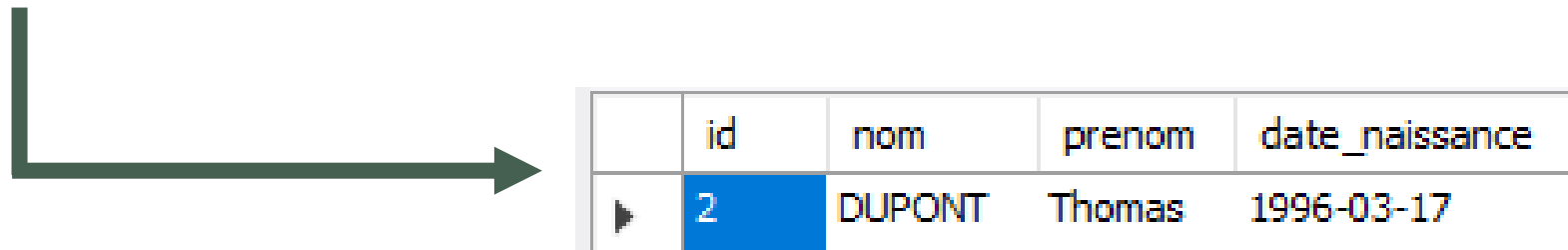


	id	nom	prenom	date_naissance
▶	1	DUPONT	Aurélie	1995-09-13
	2	DUPONT	Thomas	1996-03-17

Opérateur IN

- L'opérateur **IN** permet de vérifier qu'une valeur est contenue dans une liste.

```
SELECT * FROM person  
WHERE prenom IN ('Thomas', 'Nicolas', 'Paul');
```



A diagram showing a query result table. A large L-shaped arrow points from the SQL query above to the table below. The table has five columns: an empty column, 'id', 'nom', 'prenom', and 'date_naissance'. The first row of data has the values 2, DUPONT, Thomas, and 1996-03-17. The 'id' cell containing '2' is highlighted in blue.

	id	nom	prenom	date_naissance
▶	2	DUPONT	Thomas	1996-03-17

Opérateur LIKE

- L'opérateur **LIKE** permet d'effectuer des comparaisons simples avec un motif (pattern). Il est particulièrement utile pour vérifier si une chaîne de caractères commence, contient ou finit par une certaine valeur.
 - Le caractère spécial % est un joker qui permet d'accepter toute suite de caractères.

```
SELECT * FROM person WHERE prenom LIKE 'Aur%'
```



	id	nom	prenom	date_naissance
	1	DUPONT	Aurélie	1995-09-13

Opérateur REGEXP

- L'opérateur REGEXP permet d'effectuer des comparaisons de motif complexes en utilisant des expressions régulières.

```
SELECT * FROM person  
      WHERE telephone REGEXP '^0[67]([-\. ]?[0-9]{2}){4}$'
```

- Le détail de l'implémentation des expressions régulières dans MariaDB est décrit dans la documentation :

<https://mariadb.com/kb/en/library/regular-expressions-overview>

Fonctions et opérations

Fonctions et opérations

- MariaDB propose un grand nombre de fonctions et opérations permettant de manipuler les données.
 - La plupart du temps, les paramètres des fonctions peuvent être soit des valeurs de colonne, soit des valeurs scalaires fixes.

```
SELECT CONCAT( 'Son prénom est ', prenom) FROM person;
```



	CONCAT('Son prénom est ', prenom)
	Son prénom est Aurélie
▶	Son prénom est Thomas

Fonctions utilisées comme filtre

- Les fonctions peuvent être utilisées dans la clause **WHERE** d'une requête.

```
SELECT * FROM person  
WHERE YEAR(date_naissance) BETWEEN 1995 AND 2000;
```

Fonctions dans la sélection des informations

- Les fonctions peuvent être utilisées dans la sélection des informations à remonter par la requête.

```
SELECT prenom, YEAR(date_naissance) FROM person;
```



	prenom	YEAR(date_naissance)
▶	Thomas	1996
	Aurélié	1995

Opérateurs mathématiques

- Les opérateurs mathématiques classiques sont utilisables avec les types numériques :

Opération	Opérateur
Addition	+
Soustraction	-
Multiplication	*
Division	/
Modulo	%

Fonction mathématique

- Diverses fonctions mathématiques sont disponibles :

Opération	Fonction	Exemple
Valeur absolue	ABS(x)	ABS(-1.12) -> 1.12
Puissance	POW(x, n)	POW(2, 3) -> 8
Arrondi	ROUND(x,n)	ROUND(1.58, 1) -> 1.6
Partie entière	FLOOR(x)	FLOOR(1.58) -> 1
	CEIL(x)	CEIL(1.58) -> 2
Trigonométrie	PI(), SIN(x), COS(x), TAN(x), ...	
Autres fonctions usuelles	SQRT(x), EXP(x), LN(x), ...	

Fonctions sur les dates et heures

- Les fonctions sur les dates peuvent être séparées en 3 catégories :
 - Génération de date
 - Extraction d'une partie de la date
 - Opération sur la date

Générer une nouvelle date

- Il est possible de générer une nouvelle date à la date actuelle. 3 fonctions existent suivant si on souhaite avoir la date ou l'heure :

CURRENT_DATE() -- 2020-09-15

CURRENT_TIME() -- 08:57:24

NOW() -- 2020-09-15 08:57:24

Extraction d'une partie d'une date

	Fonction	Exemple	Résultat
Année	YEAR(d)	YEAR ('2020-09-15 11:05:23')	2020
Mois	MONTH(d)	MONTH ('2020-09-15 11:05:23')	9
Jour dans le mois	DAY(d)	DAY ('2020-09-15 11:05:23')	15
Heure	HOUR(t)	HOUR ('2020-09-15 11:05:23')	11
Minute	MINUTE(t)	MINUTE ('2020-09-15 11:05:23')	5
Seconde	SECONDE(t)	SECOND ('2020-09-15 11:05:23')	23
Jour dans l'année	DAYOFYEAR(d)	DAYOFYEAR ('2020-09-15 11:05:23')	259
Semaine dans l'année	WEEK(d)	WEEK ('2020-09-15 11:05:23')	37
Jour dans la semaine	DAYOFWEEK(d)	DAYOFWEEK ('2020-09-15 11:05:23')	3

Opération sur les dates

- Il est possible d'effectuer des opérations arithmétiques sur les dates avec les fonctions **DATE_ADD** et **DATE_SUB** :

```
DATE_ADD( '2020-09-15 12:20:29', INTERVAL 1 YEAR)  
-- 2021-09-15 12:20:29
```

```
DATE_SUB( '2020-09-15 12:20:29', INTERVAL 365 DAY)  
-- 2019-09-16 12:20:29
```

Fonctions sur les chaînes de caractères

- Les opération classiques sur les chaines de caractères sont possibles :

	Fonction	Exemple	Résultat
Longueur de la chaîne	CHAR_LENGTH(s)	CHAR_LENGTH ('Aurélie')	7
Concaténation	CONCAT(s1, s2, ...)	CONCAT ('Aurélie', 'Thomas')	AurélieThomas
Passage en minuscule	LOWER(s)	LOWER ('Aurélie')	aurélie
Passage en majuscule	UPPER(s)	UPPER ('Aurélie')	AURÉLIE
Extraction d'une sous-chaine	SUBSTRING(s, pos, len)	SUBSTRING ('Aurélie', 2, 3)	uré
Suppression des espaces	TRIM(s)	TRIM (' Aurélie ')	Aurélie


Alias, Tris et limites

AS, ORDER BY et LIMIT

Alias de colonne

- Il est possible de renommer le nom d'une colonne avec le mot clé **AS**. C'est particulièrement utile dans le cas d'utilisation de fonction.

```
SELECT prenom, YEAR(date_naissance) AS annee_naissance FROM person;
```



	prenom	annee_naissance
▶	Thomas	1996
	Aurélie	1995

Tri des lignes

- Il est possible d'ordonner les lignes résultantes d'une requête **SELECT** avec le mot clé **ORDER BY**. Il est possible de trier sur des chaînes de caractères, sur des dates ou sur des numériques

```
SELECT CONCAT(prenom, ' ', nom) FROM person  
WHERE date_naissance < '2000-01-01'  
ORDER BY prenom;
```



	CONCAT(prenom, ' ', nom)
▶	Aurélie DUPONT
	Thomas DUPONT

Tri ascendant et descendant

- Les mots clé ASC et DESC permettent de préciser le sens de tri. Par défaut, le tri est ascendant.

```
SELECT date_naissance FROM person  
ORDER BY date_naissance DESC;
```



	date_naissance
▶	1996-03-17
	1995-09-13

Tri multiple

- Il est possible d'enchaîner plusieurs tris en les séparant par le caractère « , ».

```
SELECT nom, prenom FROM person  
ORDER BY nom DESC, prenom ASC;
```



	nom	prenom
▶	DUPONT	Aurélie
	DUPONT	Thomas

Restriction du nombre de lignes

- Il est possible de limiter le nombre de lignes renvoyées par une requête avec le mot clé **LIMIT**.
- Il prend 2 paramètres : l'offset et le nombre de lignes à renvoyer. Si l'offset n'est pas renseigné, la valeur 0 est prise.

-- Les 5 premières lignes

```
SELECT * FROM person LIMIT 5;
```

-- 5 lignes à partir de la 3è

```
SELECT * FROM person LIMIT 2,5;
```


Agrégations

La clause GROUP BY

Fonctions d'agrégation

- Les fonctions d'agrégation calculent un résultat à partir des informations de plusieurs lignes. Elles ne peuvent pas être utilisées dans la clause **WHERE** d'une requête.

```
SELECT AVG(taille) AS taille_moyenne FROM person;
```



	taille_moyenne
▶	172.5000

Fonctions d'agrégation

Fonction	Description
COUNT(col)	Nombre de valeurs non vide pour la colonne col
COUNT(*)	Nombre de lignes
SUM(col)	Somme des valeurs de la colonne col
AVG(col)	Moyenne des valeurs de la colonne col
MIN(col)	Valeur minimale pour la colonne col
MAX(col)	Valeur maximale pour la colonne col

DISTINCT

- Le mot clé DISTINCT permet de récupérer toutes les valeurs différentes existantes pour une certaine colonne.

```
SELECT DISTINCT (nom) FROM person;
```



	nom
	THOMAS
	MARTIN
	LEFEVRE
▶	DUPONT

GROUP BY

- La clause GROUP BY permet de regrouper des lignes qui ont la même valeur dans une ou plusieurs colonnes.
- Il est possible de grouper les lignes sur un critère calculé par une fonction. Il est même possible de récupérer une colonne calculée comme critère d'agrégation.

```
SELECT nom, prenom, AVG(note) FROM person GROUP BY nom, prenom;
```

GROUP BY et fonctions d'agrégation

- Les fonctions d'agrégation utilisées dans une requête avec GROUP BY sont exécutées une fois par groupe.

```
SELECT nom, COUNT(*) as nombre FROM person GROUP BY nom;
```



	nom ▲	nombre
▶	DUPONT	2
	LEFEVRE	1
	MARTIN	1
	THOMAS	1

Clause WHERE dans une agrégation

- Si une clause WHERE existe, elle est appliquée avant que l'agrégation soit faite.

```
SELECT nom, COUNT(*) as nombre FROM person  
WHERE date_naissance > '1992-01-01'  
GROUP BY nom;
```



	nom ▲	nombre
▶	DUPONT	2
	MARTIN	1

HAVING

- La clause HAVING permet d'ajouter un filtre sur les données après l'agrégation.

```
SELECT nom, COUNT(*) as nombre FROM person  
GROUP BY nom  
HAVING nombre > 1;
```



	nom	nombre
▶	DUPONT	2

Ordre d'exécution

- L'ordre d'exécution des clauses d'une requête est le suivant :
- SELECT ...
- FROM ...
- WHERE ...
- GROUP BY ...
- HAVING ...
- ORDER BY ...

Ordre d'exécution

- L'ordre d'exécution des clauses d'une requête est le suivant :
- SELECT ... #6
- FROM ... #1
- WHERE ... #2
- GROUP BY ... #3
- HAVING ... #4
- ORDER BY ... #5

Jointures

Jointures

- Jusqu'à présent, les requêtes **SELECT** effectuées ne permettent de lire les informations que d'une seule table.
- Il est possible de mettre à profit les relations entre les différentes tables afin de manipuler les informations de plusieurs tables en une seule requête.
- C'est ce qu'on appelle faire une **jointure**.

FROM

- Afin de réaliser une jointure entre plusieurs tables, il va être nécessaire de complexifier la clause **FROM** de la requête **SELECT**.

```
SELECT * FROM person WHERE prenom='Thomas'
```

Alias de table

- Il est possible de donner un alias à une table. Cela est particulièrement utile quand on manipule de nombreuses tables dans une seule requête.
 - Cet alias peut être utilisé avant le nom d'une colonne pour préciser quelle table est concernée.

```
SELECT * FROM person p WHERE p.prenom='Thomas'
```



Données exemple

- Dans toute la suite de cette partie sur les jointures, deux tables vont être utilisées avec les données suivantes :

personne		
id	nom	prenom
1	MARTIN	Nicolas
2	DUPONT	Thomas
3	LEFEVRE	Julie

adresse		
id	id_person	ville
1	1	Lille
2	2	Seclin
3	1	Seclin

Produit cartésien

- La jointure la plus simple est un produit cartésien (ou **CROSS JOIN**) entre les lignes des deux tables.
- Il suffit pour cela de lister les deux tables (ou plus) dans la clause **FROM** en les séparant avec le caractère « , ».
- Le résultat sera alors l'ensemble des combinaisons possibles entre les lignes des deux tables.

Produit cartésien

SELECT * **FROM** personne, adresse;

Colonnes de la
table personne

Colonnes de la
table adresse



	id	nom	prenom	id	id_person	ville
▶	1	MARTIN	Nicolas	1	1	Lille
	2	DUPONT	Thomas	1	1	Lille
	3	LEFEVRE	Julie	1	1	Lille
	1	MARTIN	Nicolas	2	2	Sedin
	2	DUPONT	Thomas	2	2	Sedin
	3	LEFEVRE	Julie	2	2	Sedin
	1	MARTIN	Nicolas	3	1	Sedin
	2	DUPONT	Thomas	3	1	Sedin
	3	LEFEVRE	Julie	3	1	Sedin

Filtrer le produit cartésien

- Une fois la jointure effectuée, il est possible de filtrer les informations intéressante dans la clause **WHERE**.

```
SELECT * FROM personne, adresse  
WHERE personne.id = adresse.id_person;
```



	id ▲	nom	prenom	id	id_person	ville
▶	1	MARTIN	Nicolas	1	1	Lille
	1	MARTIN	Nicolas	3	1	Sedin
	2	DUPONT	Thomas	2	2	Sedin

Jointures explicites

- L'action effectuée dans la requête précédente est appelée **INNER JOIN**. Cela permet de lister les informations qui se retrouvent dans les deux tables.
- Il est possible de l'écrire différemment avec les mots clé **JOIN** et **ON** dans la clause **FROM** de la requête.

```
SELECT * FROM personne  
JOIN adresse ON personne.id = adresse.id_person;
```

Avantage des jointures explicites

- Une jointure explicite permet de bien séparer les deux types de filtre effectués :
 - Filtres techniques liés aux relations entre les tables
 - Filtres « intéressants » liés à l'objectif de la requête

```
SELECT nom, prenom  
FROM personne
```

Filtre technique permettant d'expliquer comment sont
liés une personne et son adresse

```
JOIN adresse ON personne.id = adresse.id_personne  
WHERE ville = 'Seclin'
```

Ce qui m'intéresse vraiment :
lister les habitants de Seclin

Jointures externes

- Une jointure externe (**OUTER JOIN**) permet d'ajouter au résultat de la requête des lignes qui n'ont pas d'équivalent dans l'autre table.
- Il existe deux types de jointure externe suivant si l'on souhaite afficher les informations de la table à gauche ou à droite du mot clé JOIN.

	table1 JOIN table2	table1 LEFT JOIN table2	table1 RIGHT JOIN table2
Correspondance dans les 2 tables	X	X	X
Seulement dans la table 1		X	
Seulement dans la table 2			X

Jointures externes

```
SELECT * FROM personne  
LEFT JOIN adresse ON personne.id = adresse.id_person
```



	id ▲	nom	prenom	id	id_person	ville
▶	1	MARTIN	Nicolas	1	1	Lille
	1	MARTIN	Nicolas	3	1	Sedin
	2	DUPONT	Thomas	2	2	Sedin
	3	LEFEVRE	Julie	NULL	NULL	NULL

Associations de requêtes

Plusieurs SELECT en un

Associations de requêtes

- Il est possible de combiner plusieurs requêtes **SELECT** en une seule. Cela permet d'effectuer des requêtes sinon impossibles ou difficiles à écrire en une fois.
- Si possible, on préfère ne pas utiliser cette fonctionnalité pour des soucis de performance.

UNION

- Le mot clé **UNION** entre deux requêtes **SELECT** permet de combiner le résultat des 2 requêtes.

```
SELECT * FROM personne WHERE prenom='Nicolas'  
UNION  
SELECT * FROM personne WHERE nom='DUPONT'
```

INTERSECT

- Le mot clé **INTERSECT** entre deux requêtes **SELECT** permet d'afficher l'intersection des résultats des 2 requêtes.

```
SELECT * FROM personne WHERE prenom='Nicolas'  
INTERSECT  
SELECT * FROM personne WHERE nom='DUPONT'
```

EXCEPT

- Le mot clé **EXCEPT** entre deux requêtes **SELECT** permet de retirer les résultats de la deuxième des résultats de la première.

```
SELECT * FROM personne WHERE prenom='Nicolas'  
EXCEPT  
SELECT * FROM personne WHERE nom='DUPONT'
```

Sous-requêtes

- Il est enfin possible d'utiliser une requête **SELECT** à l'intérieur d'une autre requête **SELECT**.
- Cela est possible dans la clause **WHERE** :

```
SELECT * FROM personne  
  WHERE taille=(SELECT MAX(taille) FROM personne)
```

Sous-requêtes

- On peut également avoir une sous-requête dans la liste des informations à afficher :

```
SELECT
    nom, prenom,
    (
        SELECT count(*) FROM adresse
        WHERE id_person = personne.id
    ) as nombre_adresses
FROM personne
```

Sous-requêtes

- Dans la clause **FROM**, il est possible d'utiliser le résultat d'une sous requête comme une table virtuelle.

```
SELECT nom, MAX(taille_moy)
FROM
(
    SELECT nom, AVG(taille) AS taille_moy
    FROM personne
    GROUP BY nom
) tailles_moyennes
```