Bases de Données

Grégoire Roumache

Avril 2020

1 Configuration SQL Server

- Activer SQL Server:
  1. Dans la barre de recherche Windows, taper: *services.msc*.
  2. Aller à *SQL Server (sqlexpress)* et activer le service. Le reste peut être désactivé.
- Créer une base de données:
  1. Ouvrir *SQL Server Management Studio*.
  2. Dans le menu de gauche, en-dessous du nom de l'ordinateur, click-droit sur *Databases*.
  3. Clicker sur *New Database*.
  4. Donner un nom à la base de données dans *Database name*.
  5. Dans le menu de gauche, clicker sur *Options*.
  6. Configurer *Collation* (*Classement* en français) sur *Latin_ General_ 100_ CI_ AS*.

     – CI = case insensitive
     – AS = accent sensitive

     Remarque: *French_ CI_ AS* revient à peu près au même.

2 Introduction

- Qu'est-ce qu'une donnée ?

  l'enregistrement dans un code donné d'un objet, un texte, un concept, un fait, etc.

- Donnée $\neq$ Information ?

  l'information est le sens qu'on donne à une donnée

- Une base de données, c'est:
  – une **collection de données en relation**
  – **indépendante** des applications
  – **sans redondance** inutile
  – **partageable** entre plusieurs utilisateurs
  – on **accéde** à son contenu **en lui posant une question**
- SGBD = système de gestion de bases de données

  Le SGBD (système de gestion de bases de données) permet de:
  – **créer** et **supprimer** des fichiers
  – **insérer**, **effacer** et **modifier** des enregistrements

- **rechercher** des données

- Que doit gérer un SGBD ?
  - Accès optimal à toutes les données
  - Traitement simultané des données
  - Validité et cohérence des données
  - Sécurité des données
  - Sauvegarde et récupération

3 Schéma conceptuel

3.1 Entité-relation

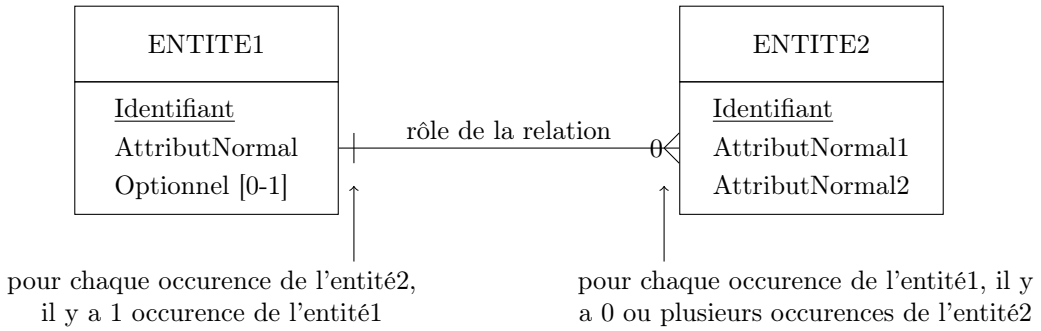
- but du schéma conceptuel = **modéliser** un système. On y définit les concepts et les liens qui les unissent
- entité = quelque chose lié au système que l'on veut modéliser (ex: personne, réservation, etc.)
- attribut = caractéristique d'une entité
- relation = lien entre les entités
- occurence =

3.2 Notation

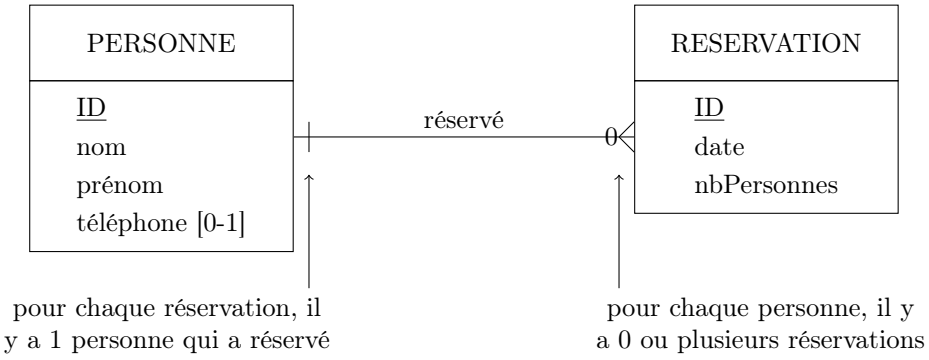
- Notation d'une entité:
  - rectangle divisé en 2 parties,
  - 1ère partie = nom de l'entité, en **majuscule**, au **singulier**,
  - 2ème partie = les attributs, en **CamelCase**, au **singulier**.
- Notation d'un attribut:
  - un attribut **souligné** est un **identifiant**,
  - un attribut suivi de **[0-1]** est un attribut **optionnel**.
- Notation des relations:
  - une relation est simplement une ligne qui relie les entités,
  - elle a un **rôle** qui est écrit **au milieu de la ligne**,
  - elle a aussi des **cardinalités à ses extrémités**.

    une cardinalité exprime le nombre d'occurences d'une entité par rapport à une autre entité

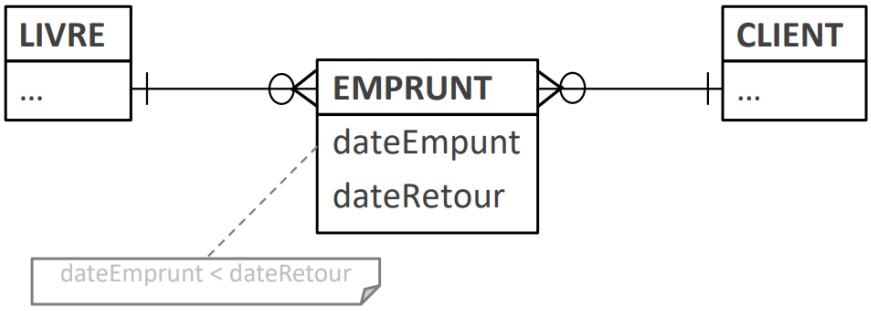
Exemple (1):



Exemple (2):



Remarque: on peut ajouter des "commentaires" dans le schéma, ce sont les **contraintes d'intégrité**. Exemple:



3.3 Méthode pour créer le schéma

1. Identifier les **entités** et leur **attributs**.
2. Souligner les attributs servant d'**identifiants** et ajouter [0-1] aux **attributs optionnels**.
3. Identifier les **relations** entre les entités.
4. Ajouter les **cardinalités**.
5. Ajouter les **contraintes d'intégrité** (si nécessaire).
6. **Vérifier** que le schéma est correct et le **normaliser** si besoin.

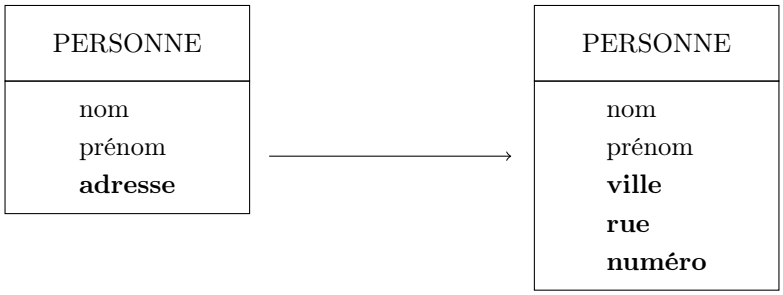
3.4 Normalisation

- normalisation = ensemble de bonnes pratiques
- La nomalisation permet d'éviter:
  - les contre-performances
  - la redondance d'information
  - les anomalies transactionnelles
- La normalisation permet d'améliorer:
  - les mises à jours
  - la maintenance
  - l'évolution
- Il y a 3 formes de normalisation:

1. **1ère forme**:
   - tous les attributs dépendent de l'identifiant,
   - tous les attributs ont une valeur atomique.
2. **2ème forme**: un attribut ne peut pas dépendre d'une seule partie de l'identifiant.
3. **3ème forme**: aucun attribut ne peut dépendre d'un autre attribut à l'execption de l'identifiant.

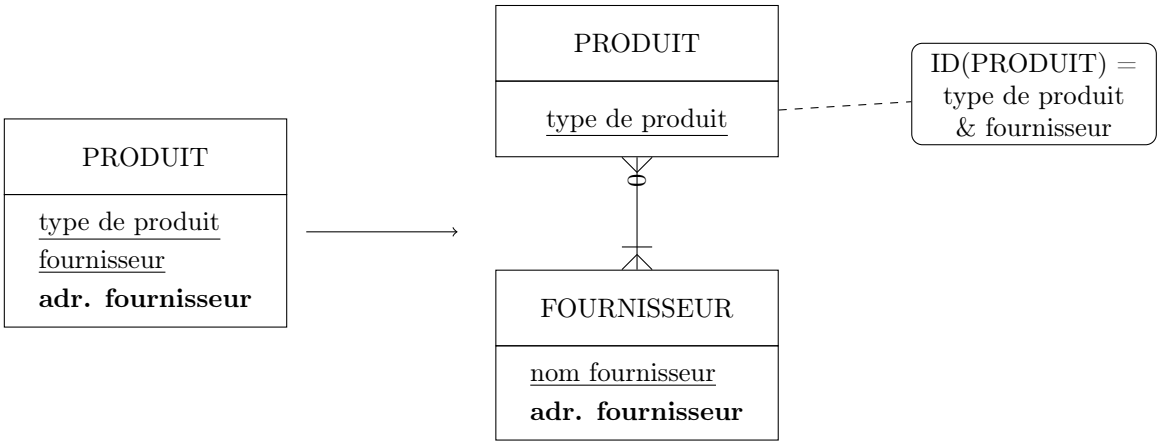
- Exemples de normalisation:

1. 1ère forme:



L'attribut **adresse** n'a pas une valeur atomique donc on la sépare en ses composants.

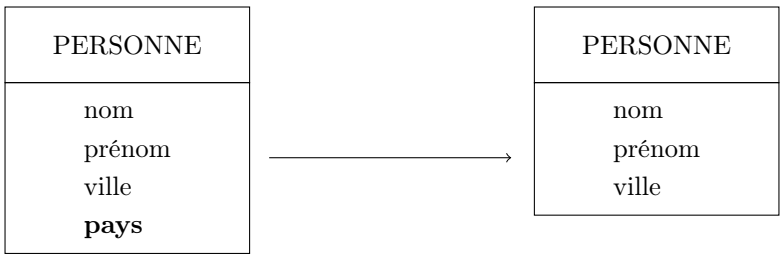
2. 2ème forme:



Ici, l'adresse du fournisseur (**adr. fournisseur**) dépend de l'attribut fournisseur mais pas de l'attribut type de produit.

3. 3ème forme:

Exemple de normalisation:

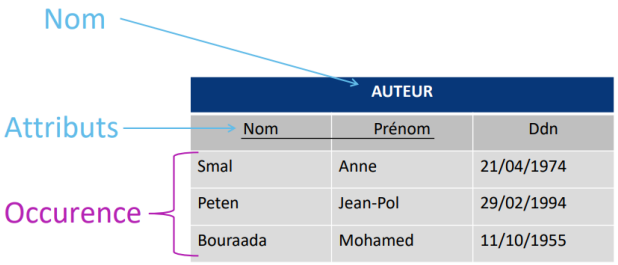


L'attribut **pays** dépend de l'attribut ville qui n'est pas un identifiant. On doit donc le supprimer.

4 Schéma conceptuel

4.1 Table

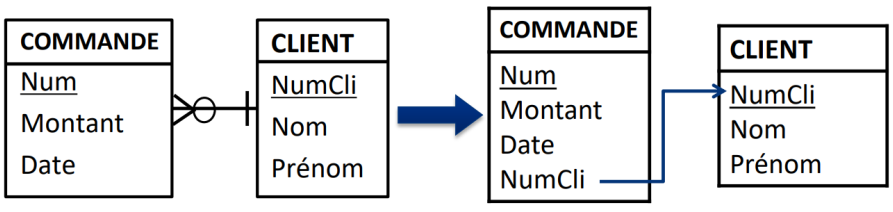
- La **table** dans une base de données de type relationnel correspond à une entité du diagramme ERD.
  - elle est définie par un nom,
  - ses colonnes sont ses attributs,
  - ses lignes sont ses occurences.



- Valeur particulière dans une table: `null`, elle a 3 significations:
  - la valeur de l'attribut est inconnue pour certaines occurrences,
  - l'attribut ne s'applique pas à certaines occurrences,
  - certaines occurrences ne possèdent pas de valeur pour l'attribut.
- **Remarque**: dans une BD relationnelle, l'ordre des lignes et des colonnes n'a pas d'importance.

4.2 Clé primaire

- La **clé primaire** est un attribut qui sert d'*identifiant*.
  - un bon identifiant est un identifiant invariant,
  - aucune clé primaire ne peut être `null`,
  - les identifiants composés sont rarement utilisés (on préfère un identifiant technique).
- Pour stocker les relations entre les entités dans une table, on utilise des **clés étrangères**, c-à-d qu'on ajoute une colonne pour y stocker les clés primaires des autres tables.



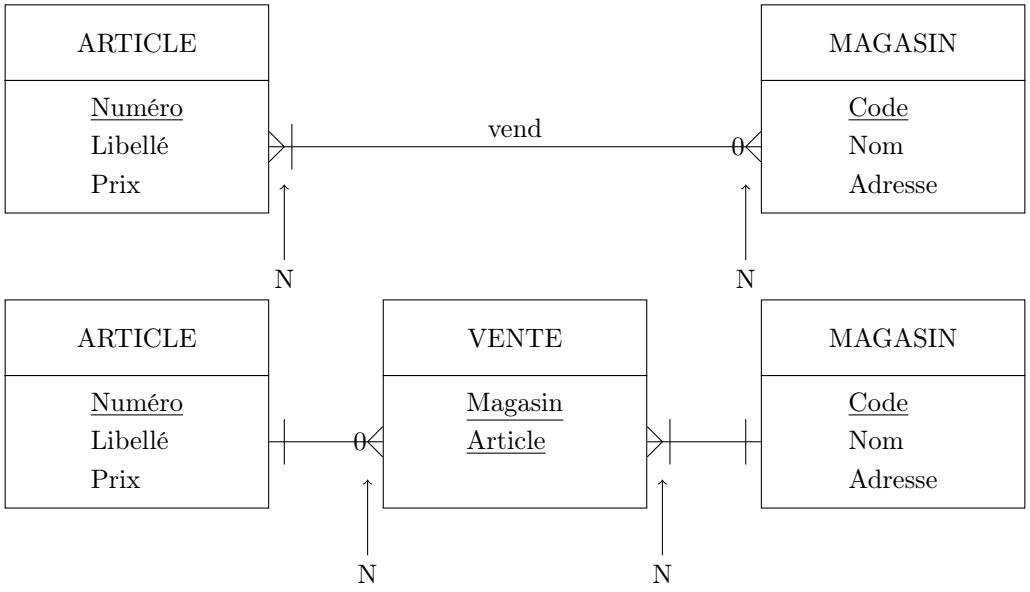
COMMANDE			
Num	Montant	Date	NumCli
1384	125,5	01/04/2019	12
1385	130	12/10/2019	65
1386	160	8/1/2020	65

CLIENT		
NumCli	Nom	Prénom
12	Smal	Anne
65	Peten	Jean-Pol
28	Bouraada	Mohamed

- Où placer la clé étrangère ? On le fait en fonction des relations entre les entités:

- relation 1-N $\Longrightarrow$ clé du côté N
- relation 1-1 $\Longrightarrow$ clé du côté 1
- relation 1-(0,1) $\Longrightarrow$ clé du côté (0,1)
- relation N-N $\Longrightarrow$ on crée une nouvelle entité qui représente la relation

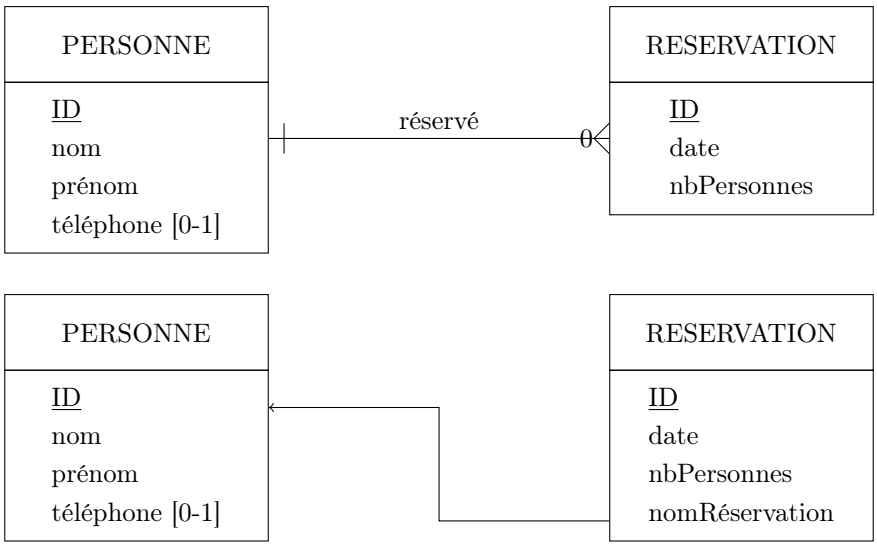
Exemple relation N-N:



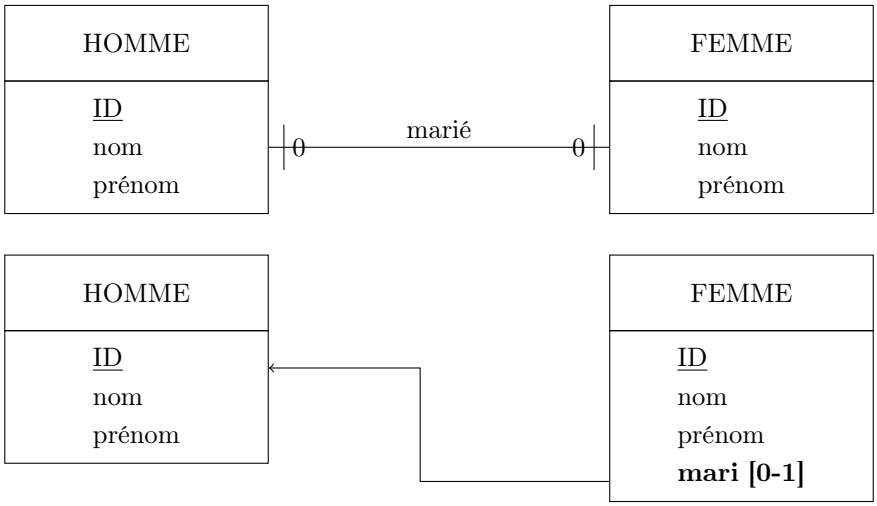
4.3 Schéma relationnel

Pour convertir un schéma entité-relation en schéma relationnel, on ajoute les clés étrangères dans les entités et on trace une flèche qui part de la clé étrangère vers la clé primaire qu'elle représente.

Exemple:



Remarque: dans certains cas, il faut ajouter **[0-1]** à côté de la clé étrangère si nécessaire.
Exemple:



5 SQL

5.1 Base

- Commande pour créer une base de données: `CREATE DATABASE <nom_db>;`
- En SQL, la valeur `null` est une valeur spéciale. Par exemple, quand on crée une occurence et qu'on ne donne pas de valeur à un attribut optionnel, il prendra alors la valeur `null`.
- Liste des types les plus communs:
  - `int` = entier
  - `float` = nombre décimal,
  - `varchar` = string/chaîne de caractères de taille variable
  - `varchar2` = string pour laquelle: `""` == `null`
  - `date`, défaut = 1900-01-01
  - `datetime`, défaut = 1900-01-01 00:00:00
- Créer une table:

ENTITE1
Id
Attribut
Option [0-1]
Relation

```
CREATE TABLE ENTITE1
(
   Id int PRIMARY KEY,
   Attribut varchar(100) NOT NULL,
   Option varchar(20) NULL,
   Relation int NULL FOREIGN KEY REFERENCES ENTITE2(Id),
);
```

5.2 Contraintes

- Liste de contraintes qu'on peut donner aux attributs:
  - `NULL` = attribut optionnel,
  - `NOT NULL` = attribut obligatoire,
  - `PRIMARY KEY` = l'attribut est une clé primaire,
  - `FOREIGN KEY` = l'attribut est une clé étrangère,
  - `UNIQUE` = la valeur de l'attribut doit être différente pour chaque occurence,

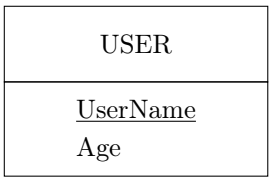
- `DEFAULT <valeur>` = donne une valeur par défaut,
- `CHECK <condition>` = la valeur de l'attribut doit vérifier la condition donnée,

Exemple avec `check`:

```
CREATE TABLE USER (
    UserName varchar(63) PRIMARY KEY,
    Age int CHECK (Age>=18),
);
```

- Normalement, on ajoute les *contraintes* après avoir déclarer l'attribut (ex: null, unique, etc.). Mais ce n'est pas obligatoire, on peut aussi ajouter des contraintes par après avec: `CONSTRAINT <nom_contrainte> <contrainte>`.
  Exemples:
  - `CONSTRAINT <nom_contrainte1> UNIQUE email`,
  - `CONSTRAINT <nom_contrainte2> CHECK (Age>=18 AND Pays='Belgium')`,
  - `CONSTRAINT <nom_contrainte3> NOT NULL prenom`,
  - `CONSTRAINT <nom_contrainte4> PRIMARY KEY Id`,
  - `CONSTRAINT <nom_contrainte5> FOREIGN KEY IdVille REFERENCES VILLE (IdVille)`,
  - `CONSTRAINT <nom6> FOREIGN KEY (IdPart1, IdPart2) REFERENCES ENTITE (IdPart1, IdPart2)`,

  Exemple avec la création d'une table:




```
CREATE TABLE USER
(
   UserName varchar(63),
   Age int,
   CONSTRAINT UserName_PK PRIMARY KEY Id,
   CONSTRAINT Age_Majeur CHECK (Age>=18),
);
```

5.3 Modifier une table

- Supprimer une table: `DROP TABLE <nom_table>;`.
- Vider une table: `TRUNCATE TABLE <nom_table>;`.
- Renommer une table: `EXEC sp_rename <nom_table>, <nouveau_nom>;`.
- Renommer une colonne:
  `EXEC sp_rename '<table>.<ancien_nom>', '<nouveau_nom>', 'COLUMN';`

  ```
  EXEC sp_rename 'USER.UserName', 'NomUtilisateur', 'COLUMN';
  ```

- Pour modifier la structure d'une table, on doit utiliser la commande: `ALTER TABLE <nom_table> <instruction>;`. Comme instruction, on peut mettre:
  - `ADD <colonne> <type>;`

    ```
    ALTER TABLE USER ADD DateCreationCompte date;
    ```

  - `ALTER COLUMN <colonne> <type>;`

    ```
    ALTER TABLE USER ALTER COLUMN DateCreationCompte datetime NOT NULL;
    ```

  - `DROP <colonne>; DROP COLUMN <colonne>;`

```
ALTER TABLE USER DROP DateCreationCompte, Age;
```

5.4 Manipulation des données

- Ajouter des données: `INSERT INTO USER (UserName, Age) VALUES ("Greg", 50000);`
- Modifier des données: `UPDATE USER SET Age=852 WHERE UserName="Greg";`
- Supprimer des données: `DELETE FROM USER WHERE UserName="Greg";`
- Obtenir des données: `SELECT <colonnes> FROM <table>;`
  On peut aussi ajouter des conditions avec: `where`, et ordonner les valeurs avec: `order by` (on peut ajouter: `asc`, ou: `desc` pour préciser si on trie par ordre croissant ou décroissant).
- Opérateurs:
  - logiques: $\sim$ (not), &, |, $\wedge$ (xor), NOT, AND, OR
  - math: +, -, *, /, %
  - comparaison: =, >, <, >=, <=, <>, !=, !>, !<
- L'opérateur `LIKE` permet de comparer des string/chaînes de caractères.
  - `SELECT * FROM USER WHERE UserName LIKE 'Gre%';`
    cette commande sélectionne toutes les occurences de `user` dont le nom d'utilisateur commence par *gre* (ex: gre, greg, gregoire, etc.).
  - `SELECT * FROM USER WHERE UserName LIKE 'gre_';`
    cette commande donne les utilisateurs qui commencent par *gre* et qui ont un caractère en plus (ex: greg, grec, etc.).
- Pour filtrer des éléments `null`, on ne peut pas utiliser la comparaison = ou !=. Il faut utiliser `is` ou `is not`.

```
SELECT * FROM USER WHERE Age IS NOT NULL;
```

5.5 Fonctions

- Fonctions simples:

```
UPPER('string')                 ⟹   STRING
LOWER('STRING')                 ⟹   string
CONCAT('str1', 'str2')          ⟹   str1str2
LEN('string')                   ⟹   6
REPLACE('string', 'tri', 'o')   ⟹   song
ROUND(123.321, 2)               ⟹   123.32
LEFT('string', 3)               ⟹   str
LTRIM('string', 3)              ⟹   ing
```

- Fonctions de groupes:

```
AVG   =   moyenne
COUNT =   nombre
MAX   =   maximum
MIN   =   minimum
SUM   =   somme
```

5.6 Groupes

- On crée des groupes avec: `GROUP BY`. Exemple:
  - Table:

	lettre
1	'a'
2	'b'
3	'a'
4	'b'
5	'a'

  - SQL: `SELECT lettre FROM TABLE GROUP BY lettre;`
  - Résultat:

	lettre
1	'a'
2	'b'

- Pour filtrer les groupes, on n'utilise pas `WHERE` mais `HAVING`. Exemple:
  - SQL: `SELECT lettre, COUNT(*) FROM TABLE GROUP BY lettre HAVING COUNT(*) = 3;`
  - Résultat:

	lettre
1	'a'

5.7 Requêtes imbriquées

Opérateurs pour les requêtes imbriquées:

IN	$\Longrightarrow$	vrai si l'élément est dans les valeurs renvoyées par le select imbriqué
ANY	$\Longrightarrow$	vrai si la condition est vérifié pour au moins 1 ligne
ALL	$\Longrightarrow$	vrai si la condition est vérifié pour toutes les lignes
EXISTS	$\Longrightarrow$	vrai si au moins 1 ligne est renvoyée

Exemples:

- Objectif: prendre les id et les noms des clients qui ont une facture.
  - SQL: `SELECT Id, Nom FROM CLIENT WHERE Id IN ( SELECT Client FROM FACTURE );`
  - Résultat:

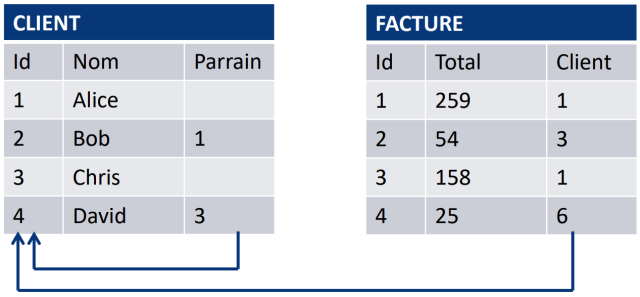
	Id	Nom
1	1	Alice
2	2	Bob
3	3	Chris

- Objectif: ...
  - SQL: `SELECT Id, Nom FROM CLIENT WHERE Id > ALL ( SELECT Client FROM FACTURE );`
  - Résultat:

	Id	Nom
1	4	David

5.8 Jointures

- Les jointures servent à lier des colonnes de tables différentes. Exemple:
  - Tables:



CLIENT		
Id	Nom	Parrain
1	Alice	
2	Bob	1
3	Chris	
4	David	3

FACTURE		
Id	Total	Client
1	259	1
2	54	3
3	158	1
4	25	6

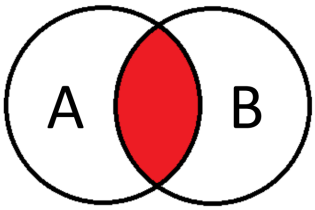
  - SQL: `SELECT * FROM CLIENT C JOIN FACTURE F ON C.Id = F.Client;`
  - Résultat:

	Id	Nom	Parrain	Id	Total	Client
1	1	Alice	NULL	1	259	1
2	3	Chris	NULL	2	54	3
3	1	Alice	NULL	3	158	1
4	2	Bob	1	4	25	2

  - SQL: `SELECT Nom, Total FROM CLIENT C JOIN FACTURE F ON C.Id = F.Client;`
  - Résultat:

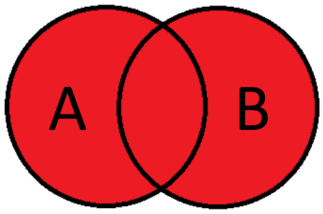
	Nom	Total
1	Alice	259
2	Chris	54
3	Alice	158
4	Bob	25

- Liste des jointures normales:
  - `SELECT C.Id, Nom FROM CLIENT C INNER JOIN FACTURE F ON C.Id = F.Client;`



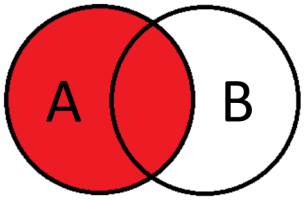
	CLIENT			FACTURE		
	Id	Nom	Parrain	Id	Total	Client
1	1	Alice		1	259	1
2	1	Alice		3	158	1
3	2	Bob	1			
4	3	Chris		2	54	3
5	4	David	3			
6				4	25	6

  - `SELECT C.Id, Nom FROM CLIENT C FULL OUTER JOIN FACTURE F ON C.Id = F.Client;`



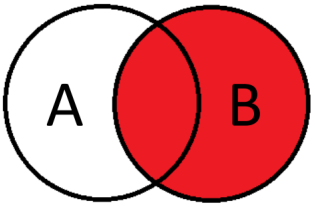
	CLIENT			FACTURE		
	Id	Nom	Parrain	Id	Total	Client
1	1	Alice		1	259	1
2	1	Alice		3	158	1
3	2	Bob	1			
4	3	Chris		2	54	3
5	4	David	3			
6				4	25	6

– `SELECT C.Id, Nom FROM CLIENT C LEFT JOIN FACTURE F ON C.Id = F.Client;`



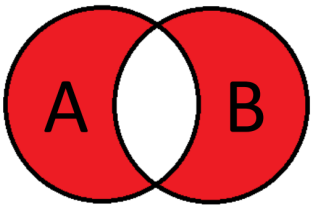
	CLIENT			FACTURE		
	Id	Nom	Parrain	Id	Total	Client
1	1	Alice		1	259	1
2	1	Alice		3	158	1
3	2	Bob	1			
4	3	Chris		2	54	3
5	4	David	3			
6				4	25	6

– `SELECT C.Id, Nom FROM CLIENT C RIGHT JOIN FACTURE F ON C.Id = F.Client;`



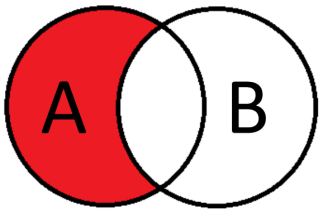
	CLIENT			FACTURE		
	Id	Nom	Parrain	Id	Total	Client
1	1	Alice		1	259	1
2	1	Alice		3	158	1
3	2	Bob	1			
4	3	Chris		2	54	3
5	4	David	3			
6				4	25	6

– `SELECT C.Id, Nom FROM CLIENT C FULL OUTER JOIN FACTURE F ON C.Id = F.Client WHERE C.Id IS NULL OR F.Client IS NULL;`



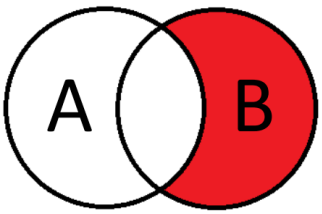
	CLIENT			FACTURE		
	Id	Nom	Parrain	Id	Total	Client
1	1	Alice		1	259	1
2	1	Alice		3	158	1
3	2	Bob	1			
4	3	Chris		2	54	3
5	4	David	3			
6				4	25	6

– `SELECT C.Id, Nom FROM CLIENT C LEFT JOIN FACTURE F ON C.Id = F.Client WHERE F.Client IS NULL;`



	CLIENT			FACTURE		
	Id	Nom	Parrain	Id	Total	Client
1	1	Alice		1	259	1
2	1	Alice		3	158	1
3	2	Bob	1			
4	3	Chris		2	54	3
5	4	David	3			
6				4	25	6

– `SELECT C.Id, Nom FROM CLIENT C RIGHT JOIN FACTURE F ON C.Id = F.Client WHERE C.Id IS NULL;`



	CLIENT			FACTURE		
	Id	Nom	Parrain	Id	Total	Client
1	1	Alice		1	259	1
2	1	Alice		3	158	1
3	2	Bob	1			
4	3	Chris		2	54	3
5	4	David	3			
6				4	25	6

- Jointure sur les colonnes de même nom:
  - SQL: `SELECT colonnes FROM tableA A NATURAL JOIN tableB B;`
  - Tables:

CLIENT

Id_client	Nom	Parrain
1	Alice	
2	Bob	1
3	Chris	
4	David	3

FACTURE

Id	Total	Id_client
1	259	1
2	54	3
3	158	1
4	25	6

- Faire correspondre chaque ligne de A avec chaque ligne de B
  - SQL: `SELECT colonnes FROM tableA A CROSS JOIN tableB B`
  - Tables:

Danseur

Id	Nom
1	Bobby
2	Chris

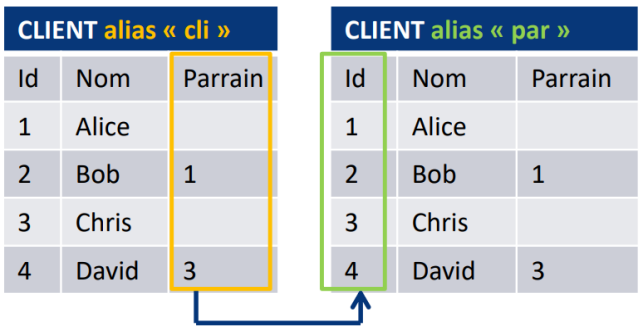
Danseuse

Id	Nom
1	Alice
2	Eve

  - Résultat:

Danseur		Danseuse	
Id	Nom	Id	Nom
1	Bobby	1	Alice
1	Bobby	2	Eve
2	Chris	1	Alice
2	Chris	2	Eve

- Jointure pour une table qui a une clé étrangère vers elle-même:
  - SQL:
```
SELECT cli.Id, cli.Nom, par.Nom AS Parrain
FROM CLIENT cli
INNER JOIN CLIENT par
ON cli.Parrain = par.Id
```
  - Tables:



– Résultats:

Id	Nom	Parrain
2	Bob	Alice
4	David	Chris