Liceu Fesch Aiacciu
NSI
Classe de Terminale

FÉVRIER

2025

BASES DE DONNÉES

Stéphane
GAREDDU

Qu'est-ce qu'une BDD ?

- Une Base de Données (ou BDD) admet plusieurs définitions
- C'est avant tout une collection de données (pouvant être « physique », donc sur papier !)
- On s'intéressera bien entendu exclusivement aux BDD informatisées

Qu'est-ce qu'une BDD ?

- Plus formellement :

> Une BDD est un ensemble structuré de données, cohérentes entre elles, modélisant le monde réel

Qu'est-ce qu'une BDD ?

- Une BDD s'incarne dans un Système de Gestion de Bases de Données (SGBD), ou en anglais : DataBase Management System (DBMS)
- Pour ce cours, on utilisera MySQL

Principes

- L'idée directrice est qu'un SGBD doit pouvoir gérer les données (création, accès, modification, suppression) indépendamment de leur représentation « interne » sur la ou les machines (fichiers, pointeurs, recherche, tri…)

Besoins

- Les opérations de gestion de données sont souvent appelées CRUD : Create, Read, Update, Delete
- Trois besoins principaux sont inhérents aux SGBD :

Description des données

- **Besoin n°1 : Les données doivent pouvoir être décrites**

→ On utilise pour cela un LDD (Langage de Description de Données) ou DDL, qui décrit la structure des données

Accès aux données

- **Besoin n°2 : Les données doivent rester intègres**, c'est-à-dire que seules les personnes autorisées à y avoir accès ne doivent y avoir accès (!)

→ On utilise pour cela un LCD (Langage de Contrôle de Données) ou DCL, qui contrôle les accès

Manipulation des données

▪**Besoin n°3 : on doit pouvoir manipuler les données, questionner notre base de données** sans pour autant dire de quelle manière, mais juste ce que l'on veut faire.

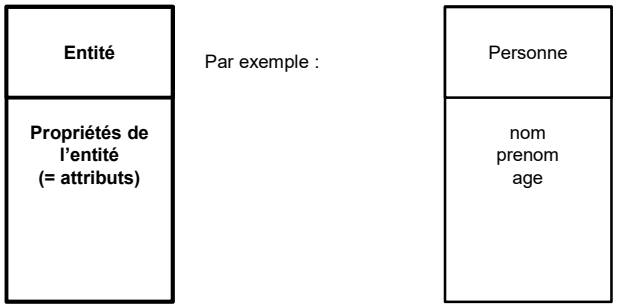
→ On va utiliser des requêtes grâce à un Langage de Manipulation des Données (LMD) ou DML en anglais

Autres besoins

▪Outre ces trois besoins principaux précédents, on a aussi besoin de :
Gérer les accès concurrents (partage)
Gérer les aspects de sécurité (journalisation des évènements, …etc)
Bénéficier de bonnes performances (arbres, hachage, « grappes »…)
Voir ACID : les 4 propriétés des transactions au sein d'une BDD

Modèle Entité-Association

Il est basé sur des entités et des associations (→ scoop !)

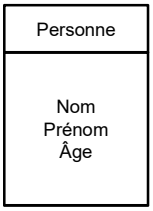


Modèle Entité-Association

- Dans cet exemple, Personne est une entité
- Elle commence par une majuscule
- Ce qu'on vient de voir est un modèle de données
- Il décrit la structuration de la base et non pas son contenu
- Ma BDD sera capable de gérer plusieurs occurrences de mon entité (entité = table)
- Attention : ces termes sont très importants, assurez-vous de bien les maîtriser

Modèle Entité-Association

- Exemple : au niveau du modèle des données j'ai ça



- Au niveau du contenu j'ai ça (par exemple)

nom	prenom	age
GAREDDU	Stéphane	22
PUZZA	Maguy	25

Notion de clef

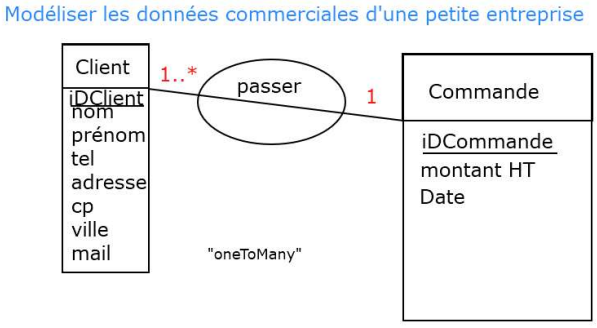
- Chaque occurrence d'une entité doit pouvoir être identifiée de manière unique
- On utilise pour cela un attribut spécial appelé clef primaire
- Exemples : n° de sécu (personne), n° pièce d'identité (personne), référence de produit (produit), n° de facture (facture), adresse e-mail (adresse e-mail)…
- Exercice : dites si les attributs suivants peuvent être des clefs (à l'oral)

Notion d'association

- Comme son nom l'indique, le modèle entité-relation est basé sur des entités…et des relations
- Elles se représentent dans un MCD via un trait entre deux tables, sur lequel on écrit le nom de la relation dans un cercle plein (infinitif ou 3e pers. sing.)
- En pratique elles symboliseront une référence dans une entité à une autre entité

MCD

Exemple de modèle conceptuel de données :



MCD

Modèle Conceptuel de Données

Trois cardinalités possibles

- Il existe trois grands types d'associations possibles, ayant des conséquences au niveau de la représentation des données
- One to One (un à un)
- One to Many, Many to One
- Many to Many
- On les décrit à l'aide d'un verbe (généralement à l'infinitif) et de cardinalités

Trois cardinalités possibles

- Notation :
- 0 ou 1 : 0..1 ou 0,1
- Exactement 1 : 1..1 ou 1,1 ou 1
- 0 ou plusieurs : 0..* ou 0,N ou *
- 1 ou plusieurs : 1..* ou 1,N
- On peut dans certains cas avoir des entiers dans les cardinalités mais on évite en général

Trois cardinalités possibles

- La cardinalité décrit le nombre minimal et maximal d'interventions d'une entité dans une association.
Attention : cardinalité minimale ≤ cardinalité maximale
- On doit toujours faire figurer les cardinalités des associations sur chaque « patte » de la relation, i.e. chaque entité participant à la relation doit posséder une cardinalité
- Les cardinalités décrivent « combien de fois » une entité peut participer à une relation

Trois cardinalités possibles

- Interprétation :
- Une cardinalité minimale de 0 indique qu'il est possible que l'entité ne soit impliquée dans aucune association (cas du client qui peut exister sans jamais avoir passé de commande)
- Inversement, une cardinalité minimale de 1 implique que l'entité, si elle existe, est forcément impliquée dans une association

One to One

- On doit toujours se poser la question suivante lorsqu'on est face à ce type de relation :

« puis-je fusionner les deux tables (entités) pour n'en faire qu'une ? »

One to One

- Fusionner les tables n'est pas toujours opportun, par exemple pour modéliser une relation de mariage
- Dans ce cas, on est quand même face à deux entités distinctes, malgré la cardinalité de la relation qui reste 1:1

One to Many, Many to One

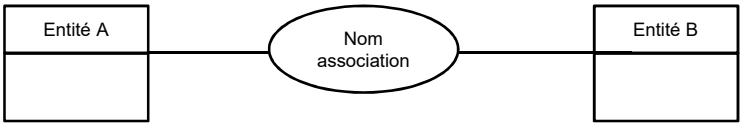
- Elle est utilisée pour exprimer une relation hiérarchique, ou « possédant-possédé », par exemple un client peut passer plusieurs commandes tandis qu'une commande n'est passée par un et un seul client
- Autre exemple : un dossier peut contenir plusieurs fichiers
- « Un seul possédant possède *n* objets »

Many to Many

- Ce type d'association exprime le fait qu'il existe plusieurs combinaisons possibles
- Par exemple des auteurs et des livres : un auteur peut écrire un ou plusieurs livres, un livre peut être écrit par un ou plusieurs auteurs
- En pratique, dans la base, chaque occurrence de ce type de relation sera un enregistrement différent

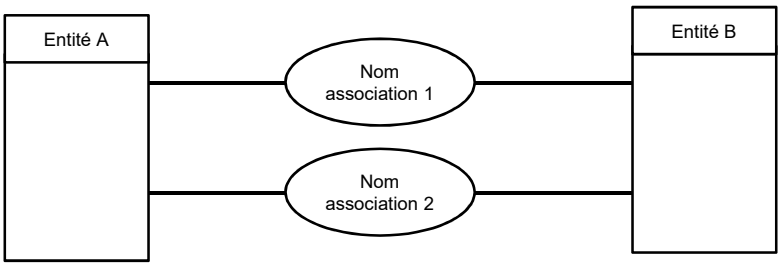
Association binaire

- L'association la plus simple est l'association binaire, c'est celle que l'on devra toujours essayer d'obtenir lorsque cela est possible



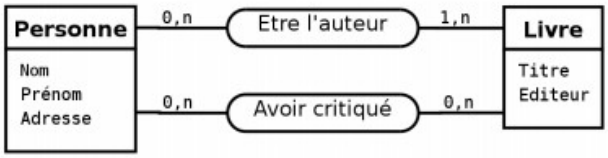
Autres types d'associations

- On peut trouver des associations spécifiques, par exemple:
- L'association multiple



Autres types d'associations

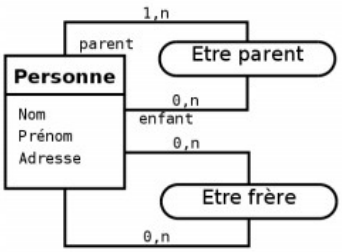
- Je vous renvoie à l'exemple de l'excellent cours de Laurent Audibert



Autres types d'associations

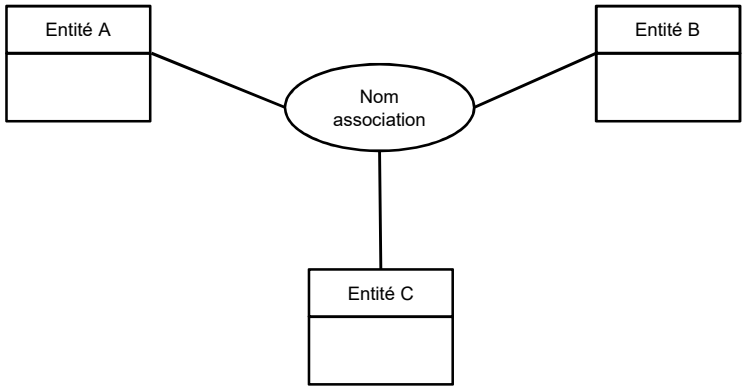
- L'association réflexive (une entité associée à elle-même, par exemple une Personne avec l'association « être parent »)

- Exemple plus « tordu », une association réflexive ET multiple, (encore merci M. Audibert !)



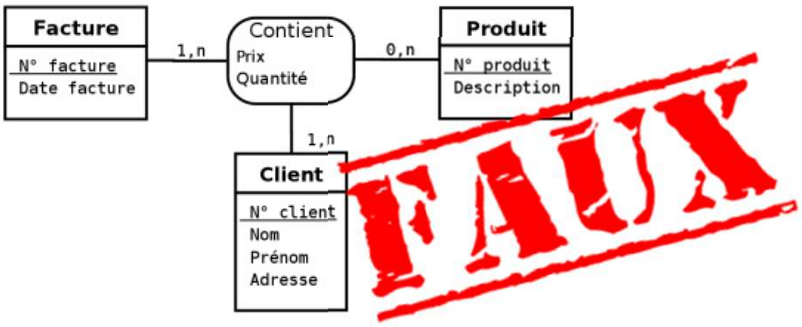
Autres types d'associations

- L'association n-aire (par exemple ternaire)



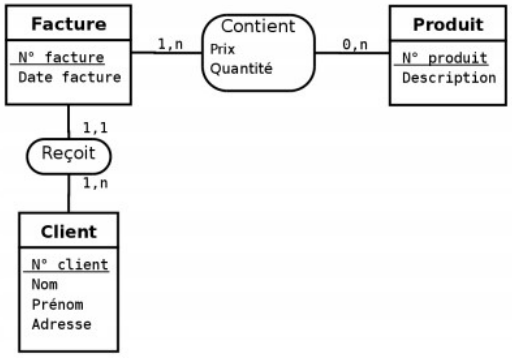
Autres types d'associations

- Ce dernier cas de figure n'est pas pratique, il peut même conduire à des erreurs :



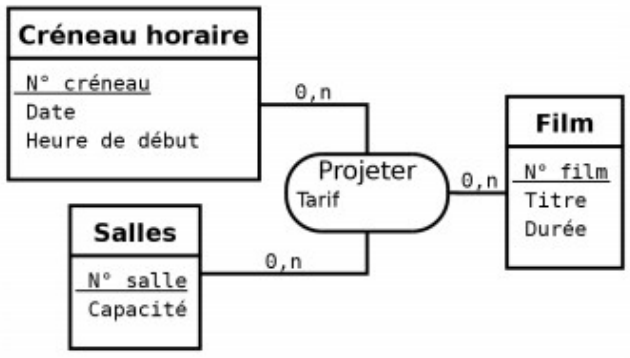
Autres types d'associations

Le mieux est de tenter de se ramener à un ensemble d'associations binaires.



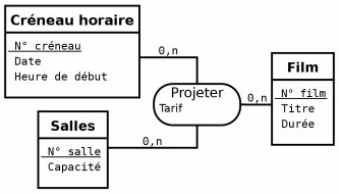
Autres types d'associations

- Exercice :



- Piste : créer une entité à partir de l'association avec « toutes ses pattes à 1 »

Correction : étape 1



Projection

N°Projection
Tarif

Salle

N°Film
Titre
Durée

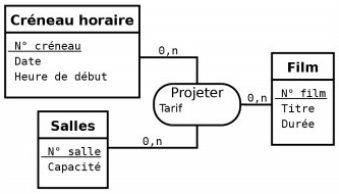
Créneau horaire

N°Créneau
Date
Heure début

Salle

N°salle
Capacité

Correction : étape 2



Projection

N°Projection
Tarif

Salle

N°Film
Titre
Durée

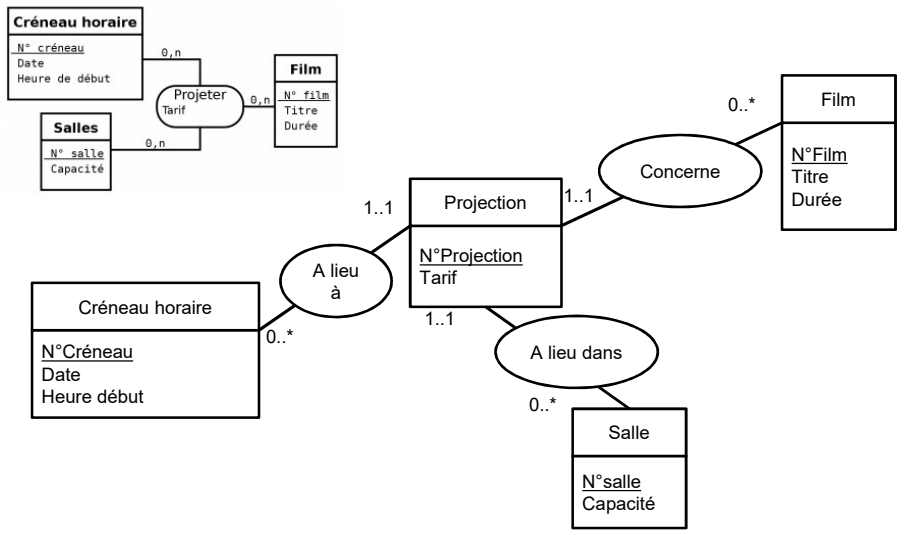
Créneau horaire

N°Créneau
Date
Heure début

Salle

N°salle
Capacité

Correction : étape 3



Un point sur le MCD

- Jusqu'à présent, nous avons parlé de comment représenter les données de manière conceptuelle, via le MCD.
- Le choix des cardinalités, surtout les cardinalités max, aura des conséquences directes pour la suite
- La suite c'est la normalisation, c'est-à-dire passer du MCD au MLD, donc au schéma relationnel

Un point sur les attributs

- Attention : nous ne parlons pas encore ici de leur format
- Un attribut est une propriété particulière d'une entité (il aura concrètement des valeurs ensuite)
- Plusieurs entités ne doivent jamais partager le même attribut (redondance !)
- Un attribut est une donnée de base (pas calculée à partir d'autres attributs)
- Les attributs doivent respecter une certaine cohésion

Un point sur les attributs

- Il est préférable de « découper » au maximum les attributs, les rendre « atomiques », par exemple un attribut adresse, sous forme de chaîne de caractères : « 18 cours Napoléon, 20090 Ajaccio » doit plutôt être décomposé en 4 attributs distincts et « indivisibles » : numéro, nom de rue, code postal, ville

Démarche générale pour créer un MCD

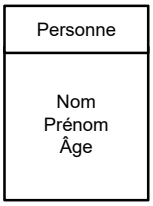
1. Établir ce qu'on appelle un dictionnaire de données (i.e. faire un recensement de toutes les propriétés qu'on peut trouver dans le système)
2. Pour chaque propriété, préciser : son nom, sa définition, d'où elle vient (on vous la donne ? Elle est calculée ?)
3. Grouper de manière logique ces propriétés, ce qui donnera une ébauche des entités
4. Définir les entités
5. Définir les associations et les cardinalités

Démarche générale pour créer un MCD

- À ce stade, j'ai concentré mes efforts sur les concepts, donc pour les attributs je ne me suis pas trop étendu (à part pour parler des clefs)
- Sachez qu'un attribut a un format, et donc que nous devrons choisir au mieux ce format
- Pour le moment on continue comme ça encore un peu…

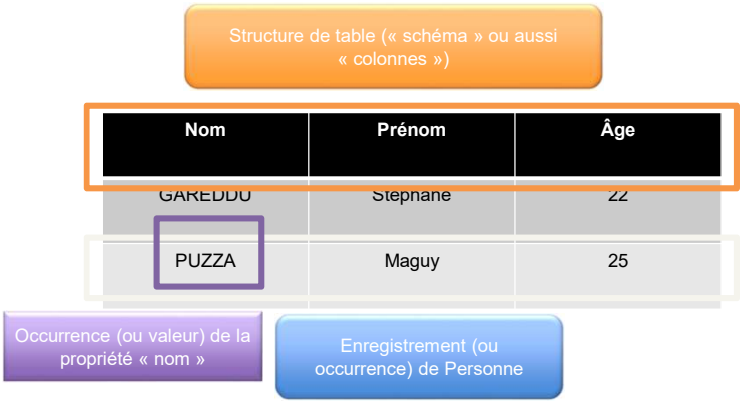
Incarnation d'un MCD dans la BDD

- On l'a dit précédemment, un MCD représente une structuration des données via des entités (qui deviennent des tables) et leurs attributs (qui deviennent des champs prenant des valeurs), exemple avec cette table orpheline (pédagogique, sans clef primaire)



Nom	Prénom	Âge
GAREDDU	Stéphane	22
PUZZA	Maguy	25

Incarnation d'un MCD dans la BDD



Incarnation d'un MCD dans la BDD

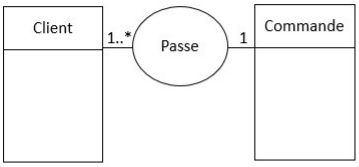
- Ici je suis dans le cas d'une table orpheline, donc le MCD est identique au MLD, ce n'est (pratiquement) jamais le cas dans une base de données !

Du MCD au MLD

- On appelle cela la normalisation
- Elle obéit à certaines règles :
- Une relation de 1 à plusieurs se traduit par la migration d'une clef étrangère de l'entité qui « possède » la cardinalité « plusieurs » vers l'entité qui « possède » la cardinalité 1 (ex client-commande, commande va récupérer la clef primaire de client qui devient clef étrangère)

Du MCD au MLD

MCD : Modèle Conceptuel de Données



Assimilable à une relation contenant-contenu (comme la relation fichier/dossier)

Quand on a une relation de ce type dans le MCD, il y a une migration de clef qui se fait dans le MLD (clef étrangère)

MLD : Modèle Logique de Données (schéma relationnel)

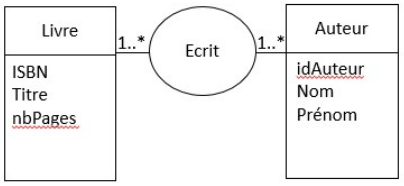
NomDeTable(…attributs…)
Client(iDclient, nom, prénom, adresse, tel, mail)
Commande(numCommande, prixHT, modeLivraison, #iDclient)

Du MCD au MLD

- Une relation de plusieurs à plusieurs va conduire à la création d'une table-association, i.e. l'association devient une entité, avec pour clef les deux clefs primaires des entités partageant la relation (+ des attributs éventuels)

Du MCD au MLD

MCD : Modèle Conceptuel de Données



Assimilable à une relation de maillage

Quand on a une relation de ce type dans le MCD, il y a....

MLD : Modèle Logique de Données (schéma relationnel)

Livre(ISBN, titre, nbPages)
Auteur(idAuteur, nom, prénom)
Ecrire(#ISBN, #idAuteur)

Du MCD au MLD

Exemples d'enregistrements possibles

Enregistrements dans la table Auteur

001	Saint-ex	Antoine
002	Diderot	Denis
003	Voltaire	François
004	Tartampion	jules

Enregistrements dans la table Livre

456456	Le petit Prince	100
666	Encyclopedie	19000

Enregistrements dans la table-association Ecrire

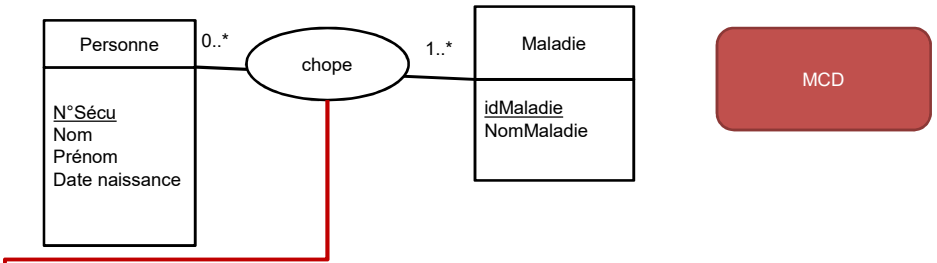
456456	001
666	002
666	003
666	004
564654	001

Du MCD au MLD

- Notre MLD sera un schéma relationnel
- Il n'a pas de représentation graphique, mais textuelle :

Entite(clefPrimaire, att1, att2…attn, #clefEtrangère)

Du MCD au MLD : Exemple



Personne(N°Sécu, Nom, Prénom, Date naissance)
Maladie(idMaladie, NomMaladie)
Malade(#N°Sécu, #idMaladie)

MLD

Du MCD au MLD : Exemple

Personne(N°Sécu, Nom, Prénom, Date naissance)
Maladie(idMaladie, NomMaladie)
Malade(#N°Sécu, #idMaladie)

MLD

Exemples d'enregistrements dans la base

Personne

N°Sécu	Nom	Prénom	Date naissance
123	PUZZA	Maguy	01/01/20
666	ZUMORTU	Amhed	01/01/20
221	TAZEGA	Simon	01/01/20
458	PORTA	Sarah	01/01/20

Maladie

idMaladie	NomMaladie
001	COVID-19
002	COVID-20 S+

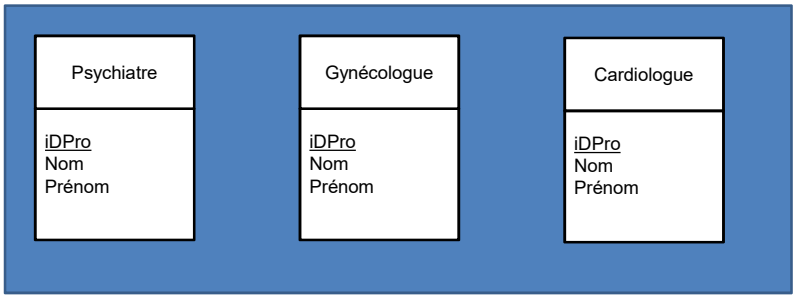
Malade

N°Sécu	IdMaladie
123	001
123	002
458	001
221	001

Du MCD au MLD : Explication de l'exemple

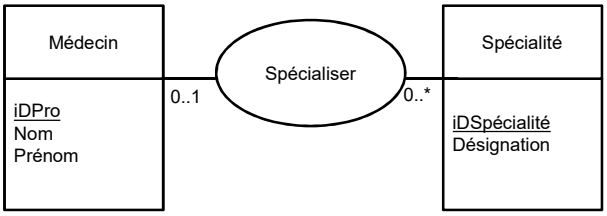
Pour une association (ici, Malade) il faut créer autant d'enregistrements (occurrence de Malade) qu'il y a de valeurs possibles
Exemple : Maguy PUZZA a eu deux maladies, il y a donc deux enregistrements comportant chacun son identifiant et l'identifiant de la maladie qu'elle a eue
N'oubliez pas, les listes n'existent pas (en BDD) !

Conseil : factoriser les entités



Conseil : factoriser les entités

- Ici on a 3 entités similaires qui varient par une spécialité
- On peut donc factoriser ça en une seule entité, liée à au plus une spécialité



Conseil : factoriser les entités

- Au niveau du schéma relationnel, cela se traduirait par une migration de la clef de Spécialité vers Médecin, cette dernière pouvant être nulle (dans ce cas le médecin n'a pas de spécialité, il est considéré comme généraliste)

Conseil : factoriser les entités

- Un traitement plus fin consisterait à créer, malgré la relation qui n'est pas une relation many-to-many, une table intermédiaire contenant un identifiant pro de médecin et un code de spécialité
- On veillerait à n'y inscrire un enregistrement que dans le cas ou l'identifiant pro du médecin n'y figurerait pas encore

Exercice

1. Modéliser via un MCD une base de données selon les spécifications client suivantes : « je voudrais pour mon entreprise de vente en ligne stocker mes produits et les clients qui les ont commandés »
2. Faire le MLD correspondant (schéma relationnel)
3. « Peupler » les tables ainsi créées avec des valeurs fictives

Exercice

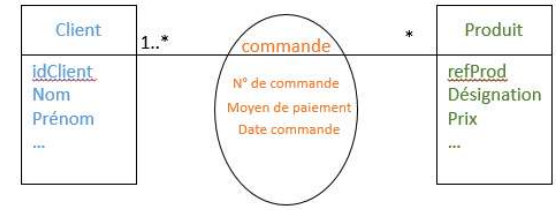
Dictionnaire de données

- Designation produit: Nom du produit donné par le vendeur
- poids: Poids en Kg donné par le fournisseur
- reference: Ref du produit (entier)
- date commande: Date de validation de la commande
- descriptif: Texte donné par le vendeur
- limite stock dispo: Nombre donné par le vendeur
- IdClient: Donné par le vendeur / générer automatiquement
- Mode paiement: Choisi par le client / proposé par le vendeur
- Adresse: Donné par le client
- Nom: Donné par le client
- Prénom: Donné par le client
- Mail: Donné par le client
- Tel: Donné par le client
- Numéro commande: Généré automatiquement
- Marque: Donné par le vendeur
- Civilité: Donné par le client
- Fournisseur: Donné par le vendeur
- prix: Donné par le vendeur
- TVA: Volée par l'état

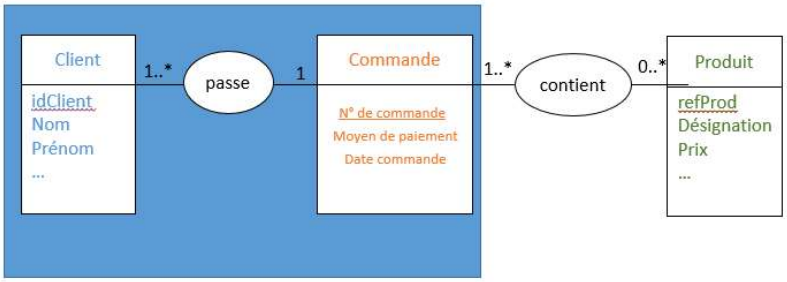
Exercice

MCD

Possibilité 1 (semi-fausse) :



Possibilité 2 :



Exercice

Schéma relationnel

- Client(idClient, nom, prénom,…)
- Commande(N°Commande, date, moyen_paiement, #idClient)
- Produit(refProd, désignation, prix, stock)
- CommandeProduit(#N°Commande, #refProd, quantité)

Exercice

Exemple d'enregistrements

Client

idCLient	Nom	Prénom
001	PUZZA	Maguy
002	TAZEGA	Simon

Produit

refProd	Désignation	Prix	Stock
1789	Chaussettes	10	8
1790	Pantalon	59	75
1791	Vodka	20	80000
1792	Chocolat	5	160

Commande

N°Commande	Date	Moyen paiement	idClient
153	22/01	CB	001
154	14/02	CB	001
155	16/02	Chèque	002

ProduitCommande

N°Commande	RefProd	Quantité
153	1791	5
153	1792	10
155	1791	8
154	1791	5
154	1792	2

Questions

1. Y a-t-il une commande sans produits ?
2. Y a-t-il des produits commandés dans au moins une commande ?
3. Y a-t-il des produits jamais commandés ?
4. Y a-t-il des clients n'ayant pas passé de commande ?
5. Y a-t-il des commandes avec un seul produit ?
6. Y a-t-il des clients ayant passé plusieurs commandes ?

Questions

Question bonus : existe-t-il un produit commandé plusieurs fois par le même client ?

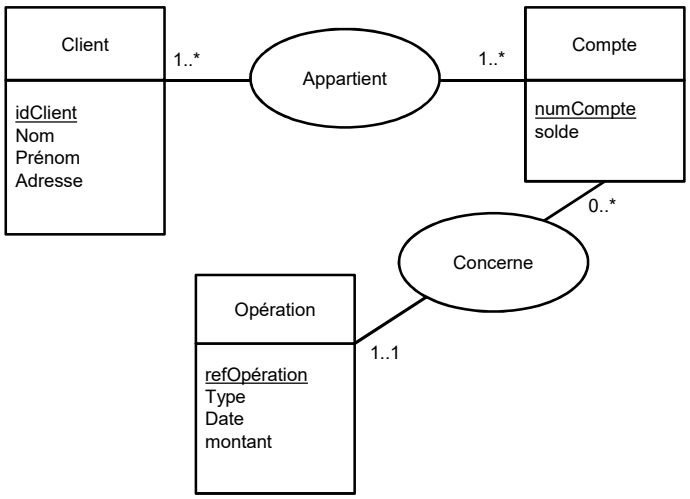
Exercice

- Une banque désire posséder un SGBD pour suivre ses clients. Elle désire ainsi stocker les coordonnées de chaque client (nom, prénom adresse), et les comptes dont elle dispose ainsi que leur solde (sachant par ailleurs que
  certains comptes ont plusieurs bénéficiaires)
- On stockera également les opérations relatives à ces comptes (retrait et dépôt, avec leur date et le montant).
- Faire le MCD
- Faire le MLD

Exercice

Une banque désire posséder un SGBD pour suivre ses clients. Elle désire ainsi stocker les coordonnées de chaque client (nom, prénom adresse), et les comptes dont elle dispose ainsi que leur solde (sachant par ailleurs que certains comptes ont plusieurs bénéficiaires). On stockera également les opérations relatives à ces comptes (retrait et dépôt, avec leur date et le montant).

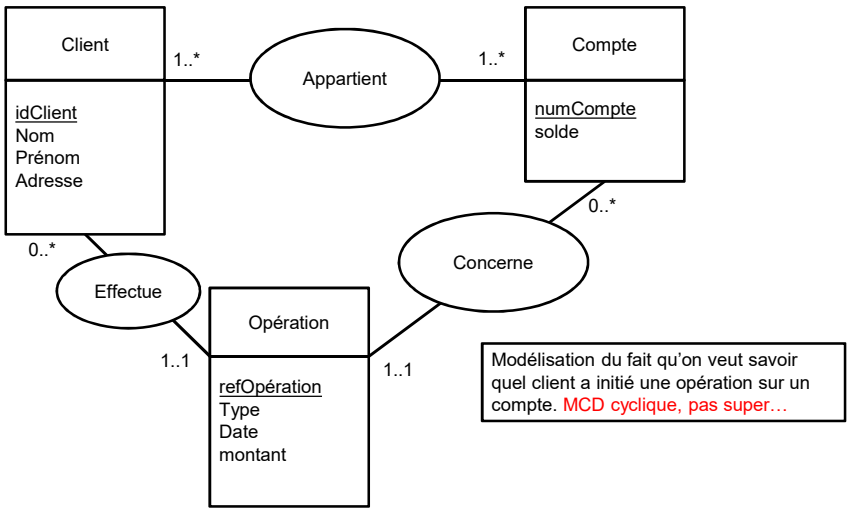
Exercice



Exercice

- Client(idClient, Nom, Prénom, Adresse)
- Compte(numCompte, solde)
- Operation(refOpération, Type, Date, montant, #numCompte)
- Appartient(#idClient, #numCompte)

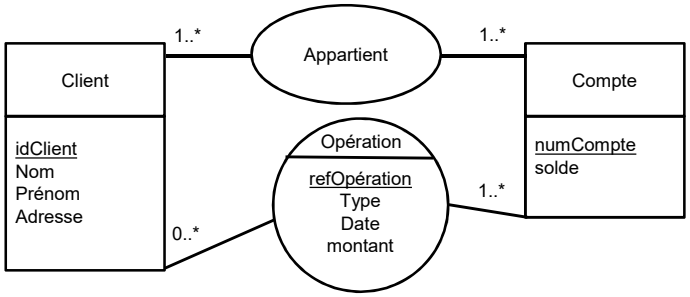
Exercice : solution alternative



Exercice : solution alternative

- Client(idClient, Nom, Prénom, Adresse)
- Compte(numCompte, solde)
- Operation(refOpération, Type, Date, montant, #numCompte, #idClient)
- Appartient(#idClient, #numCompte)

Exercice : solution alternative 2



Exercice : solution alternative 2

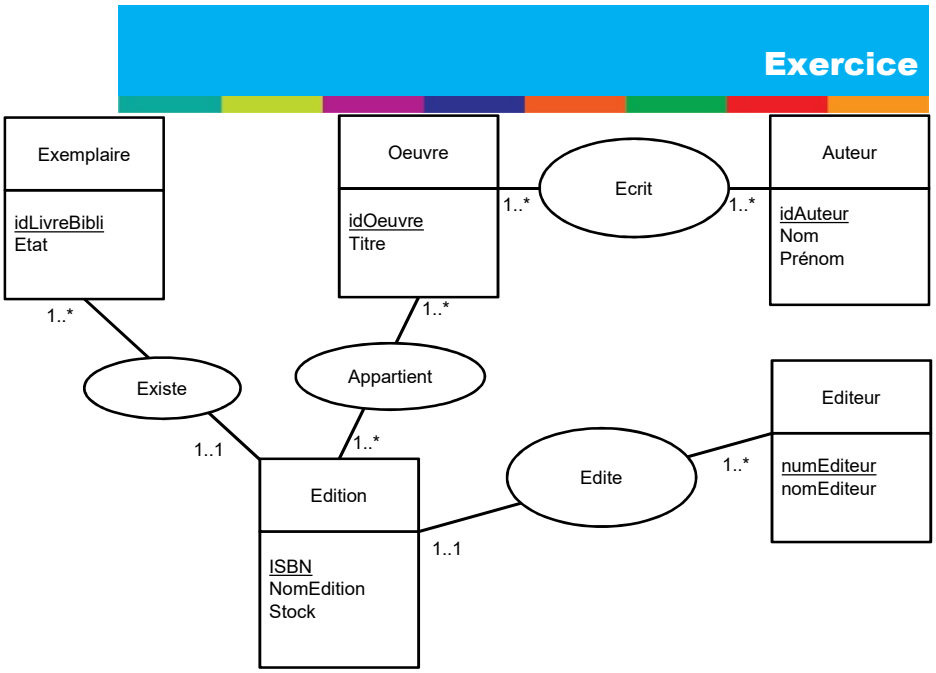
- Client(idClient, Nom, Prénom, Adresse)
- Compte(numCompte, solde)
- Operation(refOpération, Type, Date, montant, #numCompte, #idClient)
- Appartient(#idClient, #numCompte)

Même schéma relationnel que la solution alternative précédente, mais MCD différents

Exercice

Un libraire gère des œuvres littéraires :
- Une œuvre est une création littéraire
- Une œuvre a au moins un auteur et est dans une édition (un livre)
- Une édition possède un ISBN unique et a un unique éditeur. Elle peut contenir plusieurs œuvres.
- On veut mémoriser pour chaque édition le nombre d'exemplaires en stock et pour chaque exemplaire son état

Exercice



Rappels

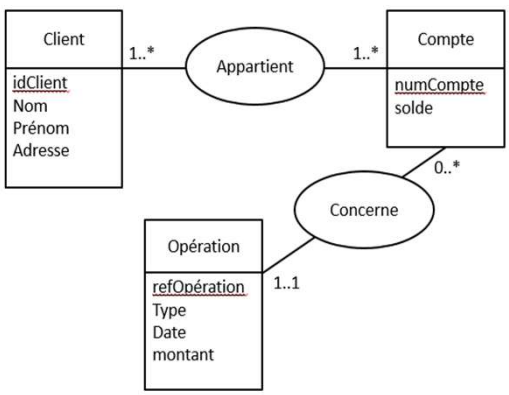
- On a vu que les bases de données relationnelles pouvaient être implémentées dans un SGBD
- Un serveur de bases de données peut contenir plusieurs bases de données (voir photo)

Rappels

- Une base de données a pour but la persistance des données, tout en permettant d'effectuer sur ces dernières des opérations de type CRUD
- Il n'y a pas d'aspect temporel dans une base de données, on se fiche de savoir comment les données arrivent, le tout est de décrire comment les stocker et les lier

Rappels

- On utilise le modèle Entité-Relation (il en existe d'autres), ou plus rigoureusement le modèle Entité-Association
- On le décrit avec un MCD, ex :



Rappels

- Un MCD contient des tables et des associations entre elles
- Chaque association comporte obligatoirement des cardinalités, dont le rôle est d'indiquer dans quelle mesure une entité participe à une association
- Une association peut-être binaire (c'est le type que l'on doit toujours rechercher) mais aussi ternaire, etc…

Rappels

- Peupler une base de données signifie créer des enregistrements dans les tables
- Il ne faut donc pas confondre le schéma d'une table (ses colonnes) avec les enregistrements qu'elle contient (ses lignes)

Rappels

- Une table comporte des attributs
- Le premier attribut est important : on l'appelle la clef primaire, il permet d'identifier chaque occurrence (=enregistrement, =tuple) de l'entité de manière unique
- **Aucune clef étrangère ne doit figurer dans le MCD !**

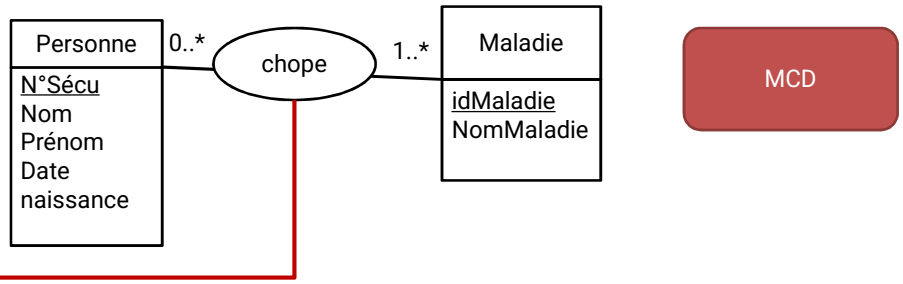
Rappels

- Dans certains cas, on a besoin de deux clefs pour identifier de manière unique un enregistrement (par exemple l'occurrence d'une relation dans une table association)

Rappels

- Une fois que l'on dispose du MCD (établi d'après les besoins du client) on le normalise, c'est-à-dire qu'on s'attache à élaborer la structure de la base de données qui en découle
- Normaliser, c'est créer un Modèle Logique de Données (MLD) en tenant compte des associations et des cardinalités sur ces dernières

Rappels : normalisation



Personne(N°Sécu, Nom, Prénom, Date naissance)
Maladie(idMaladie, NomMaladie)

MLD

Malade(#N°Sécu, #idMaladie)

Rappels

- Il existe des règles de normalisation pour passer du MCD au MLD
- C'est là que vont apparaître des clefs étrangères
- Une fois le MLD créé, on peut le peupler, c'est-à-dire créer des enregistrements respectant la structure (=le schéma) des tables qui découlent des entités et associations du MCD

Rappels : du MCD au MLD

- **Trois règles** indispensables sont à retenir lorsqu'on passe du MCD au MLD :
  - Relation One-to-One : on essaie si possible de fusionner les tables
  - Relation Many-to-One, One-to-Many : l'entité qui peut participer plusieurs fois à l'association « donne » sa clef à l'entité qui n'y participe qu'une fois (clef étrangère)
  - Relation Many-to-Many : création d'une nouvelle table dont la clef est la « concaténation » des deux clefs étrangères

Formes normales

- On appelle Forme Normale *n* (abrégé en FN) par exemple FN2, un type de relation entre les différentes entités
- Toute forme normale au niveau *n+1* doit avant tout être de la forme normale du niveau *n*
- Le but de la normalisation est d'éviter les erreurs, les redondances, les non-sens, donc de respecter les bonnes pratiques

Première forme normale

- Une relation est en première forme normale (1FN, ou 1NF en anglais) si :
  - Chaque entité dispose d'un identifiant unique (clef)
  - Chaque attribut contient une valeur atomique (indivisible) et il est monovalué. Exception notable : les dates (elles ont leur format particulier !)
- Ne pas respecter la 1NF c'est avoir une base de données « FAUSSE » ☹

Première forme normale

- Les colonnes doivent être homogènes en termes de types
- Suis-je en 1NF ?

NomFournisseur	AdresseFournisseur	Produit	Prix
Lebras	10, Rue des Gras - Clermont	Chaise	20
		Table	35
Dupont	86, Rue de la République - Moulins	Bureau	60
Lajoie	26, Rue des Dômes - Vichy	Lit	50
Dupont	39, Rue des Buttes - Moulins	Lampe	18
		Table de chevet	25

Source : S. LAPORTE (sur devsup.free.fr)

Première forme normale

- Suis-je en 1NF ?

IdClient	Nom	Prénom	Commande
0089	TOTO	Jean	1755
0090	TAZEGA	Simon	1769, 1775

*Source : Moi ^^*

Deuxième forme normale

- Elle ne concerne que les tables qui ont plus d'une clef (une relation en 1NF ne possédant qu'une clef est *de facto* en 2NF)
- Une relation est en 2NF si :
  - Elle est en 1NF
  - Tout attribut non-clef dépend de la totalité de la clef (qui est composite, rappelons-le)

Deuxième forme normale

- Autre manière de le dire : Les propriétés d'une entité ne doivent dépendre que de l'identifiant de l'entité et non d'une partie de cet identifiant
- Exemple avec le cours précédent :

CommandeProduit(#N°Commande, #refProd, quantité, date)

- Ici la date dépend seulement du numéro de commande, pas de la référence procuit, donc elle n'a rien à faire là sinon on est pas en 2NF

Troisième forme normale

$R\ (\underline{a1}, \underline{a2}, \underline{a3}, a4, a5, a6)$

- Une relation est en 3NF si :
  - Elle est en 2NF
  - Tout attribut doit dépendre directement de la clef

Eleve(<u>NumEleve,</u> codeLycée, nomLycée)

- Ici, nomLycée dépend de codeLycée (pas bon !)

Eleve(<u>NumEleve,</u> ville, pays)

- Ici, la ville dépend du pays (pas bon !)

Troisième forme normale

- Limitations : elle ne protège pas des redondances, on va plutôt utiliser pour cela la forme normale de Boyce-Codd (à vous de chercher après le cours)
- On va se limiter à ces 3 formes normales pour notre cours, il en existe d'autres (cherchez !)
- Documentez-vous sur les dépendances fonctionnelles

Types de données MySQL

- Afin d'optimiser la gestion de la mémoire, et d'accélérer les traitements, on utilise, comme en programmation, plusieurs types de données
- Ils dépendent du SGBD, mais grosso modo c'est un peu pareil partout (comme en prog !)
- On va pour notre part se concentrer types MySQL

Types de données MySQL

- Sans surprise, on va retrouver des types :
  - Numériques
  - Textuels
- Mais aussi des types dédiés à la gestion des dates et du temps

Types numériques

Utilisé pour les booléens

Data Type	Description	Storage
TINYINT(size)	Allows signed integers -128 to 127 and 0 to 255 unsigned integers.	1 byte
SMALLINT(size)	Allows signed integers from -32768 to 32767 and 0 to 65535 unsigned integers.	2 bytes
MEDIUMINT(size)	Allows signed integers from -8388608 to 8388607 and 0 to 16777215 unsigned integers.	3 bytes
INT(size)	Allows signed integers from -2147483638 to 214747483637 and 0 to 4294967925 unsigned integers.	4 bytes
BIGINT(size)	Allows signed integers from -922337203685477580 8 to 9223372036854775807 and 0 to 18446744073709551615 unsigned integers.	8 bytes
FLOAT(size,d)	Allows small numbers with floating decimal point. The size parameter is used to specify the maximum number of digits, and the d parameter is used to specify the maximum number of digits to the right of the decimal.	4 bytes
DOUBLE(size,d)	Allows large numbers with floating decimal point. The size parameter is used to specify the maximum number of digits, and the d parameter is used to specify the maximum number of digits to the right of the decimal.	8 bytes
DECIMAL(size,d)	Allows storing DOUBLE as a string, so that there is a fixed decimal point. The size parameter is used to specify the maximum number of digits, and the d parameter is used to specify the maximum number of digits to the right of the decimal.	Varies

Source : https://dzone.com

Types numériques

- Attention ce tableau est indicatif seulement
- Les paramètres sont en cours de dépréciation, on tend de plus en plus à utiliser le type tel quel
- Contrairement à ce qu'on pourrait penser, les paramètres ne concernent que la manière d'afficher les nombres, pas de les stocker

Types textuels

- On utilisera essentiellement des VARCHAR(n) où n est le nombre de caractères
- Il y a aussi les CHAR(n)
- Voici la différence :

Value	CHAR(4)	Storage Required	VARCHAR(4)	Storage Required
''	' '	4 bytes	''	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

Source : https://dev.mysql.com

Types textuels

Data Type	Description	Storage
CHAR(size)	Holds up to 255 characters and allows a fixed length string.	(Declared column length of characters * Number of bytes) <= 255
VARCHAR(size)	Holds up to 255 characters and allows a variable length string. If you store characters greater than 55, then the data type will be converted to TEXT type.	• String value(Len) + 1 WHERE column values require 0 – 255 bytes • String value(Len) + 2 bytes WHERE column values may require more than 255 bytes
TINYTEXT	Allows a string with a maximum length of 255 characters	Actual length in bytes of String value(Len) + 1 bytes, where Len < 2^8
TEXT	Allows a string with a maximum length of 65,535 characters	Actual length in bytes of String value(Len) + 2 bytes, where Len < 2^{16}
BLOB	Holds up to 65,535 bytes of data, and is used for Binary Large Objects.	Actual length in bytes of String value(Len) + 2 bytes, where Len < 2^{16}
MEDIUMTEXT	Allows a string with a maximum length of 16,777,215 characters	Actual length in bytes of String value(Len) + 3 bytes, where Len < 2^{24}
MEDIUMBLOB	Holds up to 16,777,215 bytes of data, and is used for Binary Large Objects.	Actual length in bytes of String value(Len) + 3 bytes, where Len < 2^{24}
LONGTEXT	Allows a string with a maximum length of 4,294,967,295 characters	Actual length in bytes of String value(Len) + 4 bytes, where Len < 2^{32}
LONGBLOB	Holds up to 4,294,967,295 bytes of data, and is used for Binary Large Objects.	Actual length in bytes of String value(Len) + 4 bytes, where Len < 2^{32}
ENUM(x,y,z,etc.)	Allows you to enter a list of possible values, with the maximum to be 65535 values. Just in case a value is inserted which is not present in the list, a blank value will be inserted.	1 or 2 bytes, depending on the number of enumeration values (65,535 values maximum)
SET	This data type is similar to ENUM, but SET can have up to 64 list items and can store more than one choice.	1, 2, 3, 4, or 8 bytes, depending on the number of set members (64 members maximum)

Pour info quand même

Source : https://dzone.com

Types de date et de temps

- Ils sont simples à appréhender, la seule « difficulté » réside dans la compréhension du timestamp

Data Type	Description	Storage Required Before MySQL 5.6.4	Storage Required as of MySQL 5.6.4
YEAR()	Holds the value of year either in a two digit or in a four-digit format. Year values in the range (70-99) are converted to (1970-1999), and year values in the range (00-69) are converted to (2000-2069)	1 byte	1 byte
DATE()	Holds the date values in the format: YYYY-MM-DD, where the supported range is (1000-01-01) to (9999-12-31)	3 bytes	3 bytes
TIME()	Holds the time values in the format: HH:MI:SS, where the supported range is (-838:59:59) to (838:59:59)	3 bytes	3 bytes + fractional seconds storage
DATETIME()	A combination of date and time values in the format: YYYY-MM-DD HH:MI:SS, where the supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'	8 bytes	5 bytes + fractional seconds storage
TIMESTAMP()	Holds values which are stored as the number of seconds, with the format (YYYY-MM-DD HH:MI: SS). The supported range is from (1970-01-01 00:00:01) UTC to (2038-01-09 03:14:07) UTC	4 bytes	4 bytes + fractional second storage

Exercice

- Exercice : on a la base suivante

+ Options

← T →	idPersonne	prenom	ville	dateNaissance
Éditer Copier Supprimer	1	Sacha	Ajaccio	1986-12-05
Éditer Copier Supprimer	2	Maguy	Ajaccio	1990-02-02
Éditer Copier Supprimer	3	Hans	Bastia	1990-06-05
Éditer Copier Supprimer	4	Livia	Corte	1991-02-08
Éditer Copier Supprimer	5	Gertrude	Calvi	1960-11-20
Éditer Copier Supprimer	6	Perle	Ajaccio	1960-08-10
Éditer Copier Supprimer	7	Bégonia	Bastia	1978-01-15
Éditer Copier Supprimer	8	Pétunia	Corte	1998-03-28

Requêtes SQL de base

- Requête de sélection :
  SELECT *attribut1, …,attributN*
  FROM *table;*

  On rencontrera aussi :

  SELECT *
  FROM *table;*

  *-> Ici * remplace toutes les colonnes*

Requêtes SQL de base

- ORDER BY (nomColonne) ASC/DESC (par défaut c'est asc)

Permet de faire un tri, se met toujours à la fin :

```
SELECT nom
FROM eleve
ORDER BY age DESC
```

Requêtes SQL de base

- Exercice : afficher les prénoms et les dates de naissance des personnes inscrites de la plus récente à la plus ancienne

```
SELECT prenom, dateNaissance
FROM personne
ORDER BY idPersonne DESC;
```


Requêtes SQL de base

- Exercice plus difficile : afficher les trois plus vieilles personnes

Tip : utiliser l'instruction LIMIT n, où N est un entier, en fin de requête

```
SELECT prenom
FROM personne
ORDER BY dateNaissance
LIMIT 3
```

Requêtes SQL de base

- Exercice plus difficile : afficher la ville de la personne la plus jeune

```
SELECT ville
FROM personne
ORDER BY dateNaissance DESC
LIMIT 1
```


Requêtes SQL de base

- Exercice plus difficile : sélectionner les prénoms des personnes habitant à Ajaccio
- On introduit la clause WHERE suivie d'une condition (attribut + opérateur tel que LIKE pour une égalité textuelle, sinon =, >, <, <=, >=, <> + valeur) : WHERE attribut operande valeur

- WHERE nom LIKE 'toto'
- WHERE age >= 15
- WHERE montant <> 150

Requêtes SQL de base

- Ici on a besoin d'une condition textuelle, introduite par LIKE suivie d'une chaîne entre guillemets, donc :

```
SELECT nom,prenom
FROM Personne
WHERE nom LIKE 'toto'
```


Requêtes SQL de base

- Correction :

```
SELECT prenom FROM personne
WHERE ville LIKE 'Ajaccio'
```

Exercice : combien de personnes habitent Corte ?

```
SELECT count(*) FROM personne
WHERE ville LIKE 'Corte'
```

Requêtes SQL de base

Exercice : combien de personnes habitent Corte ou Bastia ?

Tip : utiliser le OR pour combiner plusieurs conditions dans un WHERE

```
SELECT count(*) FROM personne
WHERE ville LIKE 'Corte'
OR …
```


Requêtes SQL de base

Exercice : Quels sont les prénoms des personnes nées avant 1970 vivant à Ajaccio ?

```
SELECT prenom
FROM personne
WHERE ville LIKE 'Ajaccio'
AND dateNaissance < '1970-01-01'
```

Exercice

Exercice : on veut modéliser un magasin avec des tables « client » et « commande »

- Créer une base de données

Création de table

```
MariaDB [magasin]> CREATE TABLE client
    -> (
    -> idClient INT NOT NULL AUTO_INCREMENT,
    -> login VARCHAR(40) NOT NULL,
    -> password VARCHAR(40) NOT NULL,
    -> dateNaissance DATE,
    -> ville VARCHAR(30),
    -> PRIMARY KEY (idClient)
    -> );
Query OK, 0 rows affected (0.047 sec)
```

Création de table

- Un champ peut être NULL ou NOT NULL, on appelle cela des contraintes (sur les colonnes)
- La majeure partie du temps on choisit NOT NULL
- Dans ce cas, si aucune valeur n'est spécifiée lors d'une insertion, le champ prendra la valeur par défaut donnée par l'utilisateur, ou la valeur par défaut prévue par le SGBD le cas échéant

Création de table

- Un champ avec la conctrainte AUTO_INCREMENT ne peut qu'exister une seule fois dans une table et s'applique à la PRIMARY KEY, on s'assure ainsi de son unicité

Vérification

```
MariaDB [magasin]> SHOW TABLES;
+-------------------+
| Tables_in_magasin |
+-------------------+
| client            |
+-------------------+
1 row in set (0.001 sec)

MariaDB [magasin]> DESCRIBE client;
+---------------+-------------+------+-----+---------+----------------+
| Field         | Type        | Null | Key | Default | Extra          |
+---------------+-------------+------+-----+---------+----------------+
| idClient      | int(11)     | NO   | PRI | NULL    | auto_increment |
| login         | varchar(40) | NO   |     | NULL    |                |
| password      | varchar(40) | NO   |     | NULL    |                |
| dateNaissance | date        | YES  |     | NULL    |                |
| ville         | varchar(30) | YES  |     | NULL    |                |
+---------------+-------------+------+-----+---------+----------------+
```


Insertion d'un enregistrement

```
MariaDB [magasin]> INSERT INTO client
    -> VALUES
    -> (
    -> '',
    -> 'fafa',
    -> 'Admin1337@!',
    -> '1982-07-08',
    -> 'Ajaccio',
    -> '20090'
    -> );
Query OK, 1 row affected, 1 warning (0.009 sec)

MariaDB [magasin]> SELECT * FROM client
    -> ;
+----------+-------+------------+---------------+---------+-------+
| idClient | login | password   | dateNaissance | ville   | cp    |
+----------+-------+------------+---------------+---------+-------+
|        1 | fafa  | Admin1337@! | 1982-07-08    | Ajaccio | 20090 |
+----------+-------+------------+---------------+---------+-------+
1 row in set (0.000 sec)
```

On doit renseigner toutes les colonnes (champs)

Insertion d'un enregistrement

```
MariaDB [magasin]> INSERT INTO client
    -> (login, password, dateNaissance, ville, cp)
    -> VALUES
    -> (
    -> 'totobichoco',
    -> '!@p#to',
    -> '1974-08-08',
    -> 'Bastia',
    -> '20200'
    -> );
Query OK, 1 row affected (0.005 sec)

MariaDB [magasin]> SELECT * FROM client
    -> ;
+----------+-------------+-------------+---------------+---------+-------+
| idClient | login       | password    | dateNaissance | ville   | cp    |
+----------+-------------+-------------+---------------+---------+-------+
|        1 | fafa        | Admin1337@! | 1982-07-08    | Ajaccio | 20090 |
|        2 | totobichoco | !@p#to      | 1974-08-08    | Bastia  | 20200 |
+----------+-------------+-------------+---------------+---------+-------+
2 rows in set (0.001 sec)
```

Spécification des champs que l'on veut renseigner

Insertion d'un enregistrement

- Lorsqu'on ne spécifie rien après le nom de la table, il faut donner les valeurs des tuples pour chaque colonne dans l'ordre (y compris la clef : on lui donne comme valeur la chaîne vide, l'AUTO_INCREMENT se charge de la générer)
- Sinon comme dans la capture précédente on spécifie entre parenthèses les colonnes que l'on veut renseigner

Suppression d'un enregistrement

```
MariaDB [magasin]> DELETE FROM client
    -> WHERE idClient=8
    -> ;
Query OK, 1 row affected (0.006 sec)

MariaDB [magasin]> SELECT * FROM client;
+----------+-------------+-----------------+---------------+---------+-------+
| idClient | login       | password        | dateNaissance | ville   | cp    |
+----------+-------------+-----------------+---------------+---------+-------+
|        1 | fafa        | Admin1337@!     | 1982-07-08    | Ajaccio | 20090 |
|        2 | totobichoco | !@p#to          | 1974-08-08    | Bastia  | 20200 |
|        3 | machinTruc  | shfjqs          | 1980-02-07    | Calvi   | 20300 |
|        4 | troucMouche | sqdsq@!sdDD     | 1990-02-07    | Calvi   | 20300 |
|        5 | Michael     | sqdsdsqsq@!sdDD | 1990-02-07    | Bastia  | 20200 |
|        6 | Francky     | sMMp@-          | 1987-09-11    | Ajaccio | 20090 |
|        7 | Nath        | sMgdMxdfp@-     | 1989-12-31    | Ajaccio | 20090 |
+----------+-------------+-----------------+---------------+---------+-------+
7 rows in set (0.001 sec)
```

Client 8 supprimé

Jointures

- On les a vues de manière intuitive et/ou empirique, maintenant on va formaliser tout ça
- Une jointure, quelle qu'elle soit, est utilisée pour retrouver des informations partagées sur plusieurs tables
- En effet, la quête de la non-redondance des données passe par le fait qu'une donnée n'est présente qu'à un seul « endroit » (une colonne dans une entité précise)

Jointures

- Pour ces exemples, nous allons utiliser la base « jointures » (jointures.sql)
- Vous devez donc la télécharger et travailler dessus en même temps que vous lisez ce cours, sinon ça ne sert à rien ☺
- Il existe 7 types de jointures, elles reposent toutes sur la théorie mathématique des ensembles (union, intersection, différence, etc…)
- Vous ne devrez connaître que les INNER JOIN

Jointures

Parmi ces 7 types, on trouve 2 grandes catégories :

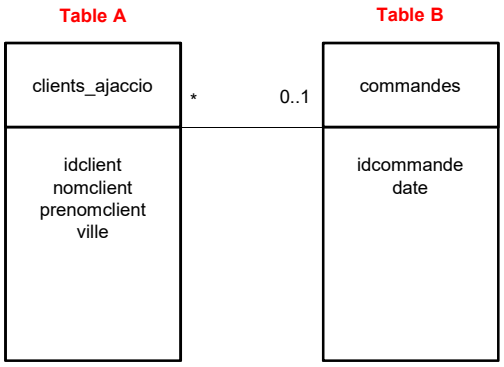
- Les INNER JOIN (jointures internes) : à connaître
- Les OUTER JOIN (jointures externes)
- Schématiquement : les premières utilisent l'intersection des tables A et B et s'occupent exclusivement des tuples résultant de cette intersection, tandis que les secondes utilisent les tuples des deux tables A et B (union)

Jointures

- L'autre particularité, c'est que j'ai ajouté une contrainte de clef étrangère pour la table commandes. La clef étrangère idclient possède une contrainte ON DELETE SET NULL
- Ca signifie que si un client est supprimé, le idclient correspondant sera NULL dans la table commandes

Jointures

Avant de commencer, un mot sur les tables qu'on va utiliser pour nos premières jointures



Note : je ne mets pas le schéma relationnel, on l'a assez fait. Il est évident que du fait des cardinalités, il y a migration de la clef primaire de la table clients_ajaccio vers la table commandes. La particularité ici c'est qu'un client peut ne pas avoir passé de commandes, et qu'une commande peut avoir été passée par un client n'existant plus dans la base de données…

Jointures

Vue d'ensemble :

```
MariaDB [jointures]> DESCRIBE clients_ajaccio;
+--------------+-------------+------+-----+---------+----------------+
| Field        | Type        | Null | Key | Default | Extra          |
+--------------+-------------+------+-----+---------+----------------+
| idclient     | int(4)      | NO   | PRI | NULL    | auto_increment |
| nomclient    | varchar(40) | NO   |     | NULL    |                |
| prenomclient | varchar(40) | NO   |     | NULL    |                |
| ville        | varchar(50) | NO   |     | NULL    |                |
+--------------+-------------+------+-----+---------+----------------+
4 rows in set (0.016 sec)
```

```
MariaDB [jointures]> DESCRIBE commandes;
+--------------+---------+------+-----+---------+----------------+
| Field        | Type    | Null | Key | Default | Extra          |
+--------------+---------+------+-----+---------+----------------+
| idcommande   | int(11) | NO   | PRI | NULL    | auto_increment |
| datecommande | date    | NO   |     | NULL    |                |
| idclient     | int(11) | YES  | MUL | NULL    |                |
+--------------+---------+------+-----+---------+----------------+
3 rows in set (0.015 sec)
```

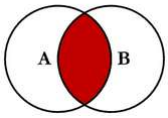

Jointures

```
MariaDB [jointures]> SELECT * FROM clients_ajaccio;
+----------+-----------+--------------+--------------+
| idclient | nomclient | prenomclient | ville        |
+----------+-----------+--------------+--------------+
|        1 | PUZZA     | Maguy        | Ajaccio      |
|        2 | ZUMORTU   | Amhed        | Ajaccio      |
|        3 | PORTA     | Sarah        | Ajaccio      |
|        5 | KOLIK     | Al           | Bastia       |
|        6 | TAZEGA    | Simon        | Bastia       |
|        7 | TALU      | Jean         | Corte        |
|        8 | RIEN      | Jean-S┼®     | Bastelicaccia |
+----------+-----------+--------------+--------------+
7 rows in set (0.000 sec)
```

```
MariaDB [jointures]> SELECT * FROM commandes;
+------------+--------------+----------+
| idcommande | datecommande | idclient |
+------------+--------------+----------+
|          1 | 2020-11-04   |        1 |
|          2 | 2020-11-04   |        1 |
|          3 | 2020-10-05   |        5 |
|          4 | 2020-12-20   |        1 |
|          5 | 2020-12-05   |        2 |
|          6 | 2020-12-08   |        2 |
|          7 | 2020-05-23   |        1 |
|          8 | 2020-07-08   |        4 |
|          9 | 2020-01-03   |        8 |
+------------+--------------+----------+
9 rows in set (0.000 sec)
```

INNER JOIN

- La première jointure qu'on a vue
- La plus intuitive
- Elle nous donne l'intersection de deux ensembles
- Graphiquement :
- Elle permet de répondre à la question : quels sont les clients qui ont passé une commande ?



INNER JOIN

- Intuitivement : on utilise la clause WHERE (je ne vous l'ai pas dit mais c'est désuet…et un peu déprécié). J'ai utilisé les alias pour plus de clarté

```
MariaDB [jointures]> SELECT prenomclient AS prenom, nomclient AS nom, idcommande AS numcommande
    -> FROM clients_ajaccio, commandes
    -> WHERE commandes.idclient = clients_ajaccio.idclient;
+----------+---------+-------------+
| prenom   | nom     | numcommande |
+----------+---------+-------------+
| Maguy    | PUZZA   |           1 |
| Maguy    | PUZZA   |           2 |
| Maguy    | PUZZA   |           4 |
| Maguy    | PUZZA   |           7 |
| Amhed    | ZUMORTU |           5 |
| Amhed    | ZUMORTU |           6 |
| Al       | KOLIK   |           3 |
| Jean-S®  | RIEN    |           9 |
+----------+---------+-------------+
8 rows in set (0.000 sec)
```

INNER JOIN

- Mais le mieux c'est quand même d'utiliser le mot clef JOIN, c'est plus clair qu'un WHERE. En l'occurrence, INNER JOIN
- Version complète (le GROUP BY n'a rien à voir avec la jointure, il me permet juste de me débarrasser des doublons car il y a des clients qui ont passé plusieurs commandes !)

INNER JOIN

- Sans doublons

```
MariaDB [jointures]> SELECT prenomclient AS prenom, nomclient AS nom, idcommande AS numcommande
    -> FROM clients_ajaccio INNER JOIN commandes ON (commandes.idclient = clients_ajaccio.idclient
    -> GROUP BY clients_ajaccio.idclient;
+----------+---------+-------------+
| prenom   | nom     | numcommande |
+----------+---------+-------------+
| Maguy    | PUZZA   |           1 |
| Amhed    | ZUMORTU |           5 |
| Al       | KOLIK   |           3 |
| Jean-S┼® | RIEN    |           9 |
+----------+---------+-------------+
4 rows in set (0.013 sec)
```

- Attention : la jointure interne INNER JOIN est une intersection : de ce fait elle ne renvoie pas les commandes qui ont été passées par un client n'existant plus, et ne renvoie pas non plus les clients n'ayant rien commandé !

INNER JOIN

- Quand il n'y a pas d'ambiguité (c'est le cas ici) concernant les clefs communes, on peut simplifier avec le mot-clef USING

```
MariaDB [jointures]> SELECT DISTINCT prenomclient AS prenom, nomclient AS nom, idcommande AS numcommande
    -> FROM clients_ajaccio INNER JOIN commandes USING (idclient)
    -> GROUP BY idclient;
+----------+---------+-------------+
| prenom   | nom     | numcommande |
+----------+---------+-------------+
| Maguy    | PUZZA   |           1 |
| Amhed    | ZUMORTU |           5 |
| Al       | KOLIK   |           3 |
| Jean-S®  | RIEN    |           9 |
+----------+---------+-------------+
4 rows in set (0.003 sec)
```


INNER JOIN

- On peut aussi utiliser NATURAL JOIN, c'est un type d'INNER JOIN particulier, on parlera plus longuement de la différence en cours. Retenez que dans ce cas simple elle peut tout à fait peut se substituer à INNER JOIN (encore une fois, faisable car les colonnes ont le même nom !)

```
MariaDB [jointures]> SELECT DISTINCT prenomclient AS prenom, nomclient AS nom, idcommande AS numcommande
    -> FROM clients_ajaccio NATURAL JOIN commandes
    -> GROUP BY idclient;
+----------+---------+-------------+
| prenom   | nom     | numcommande |
+----------+---------+-------------+
| Maguy    | PUZZA   |           1 |
| Amhed    | ZUMORTU |           5 |
| Al       | KOLIK   |           3 |
| Jean-S®  | RIEN    |           9 |
+----------+---------+-------------+
4 rows in set (0.001 sec)
```

INNER JOIN

- Si je fais une INNER JOIN avec USING, ça équivaut dans notre cas à une NATURAL JOIN, car la colonne commune ne figure qu'une fois dans les résultats (voir les deux diapos précédentes)
- Par contre si je fais une INNER JOIN avec une correspondance explicite des colonnes (en utilisant ON) il y aura deux fois la colonne idclient si je fais un SELECT * (voir exemple sur diapo suivante)

INNER JOIN

```
MariaDB [jointures]> SELECT * FROM clients_ajaccio INNER JOIN commandes
    -> ON commandes.idclient = clients_ajaccio.idclient;
+----------+----------+-------------+--------------+------------+--------------+----------+
| idclient | nomclient | prenomclient | ville        | idcommande | datecommande | idclient |
+----------+----------+-------------+--------------+------------+--------------+----------+
|        1 | PUZZA    | Maguy       | Ajaccio      |          1 | 2020-11-04   |        1 |
|        1 | PUZZA    | Maguy       | Ajaccio      |          2 | 2020-11-04   |        1 |
|        5 | KOLIK    | Al          | Bastia       |          3 | 2020-10-05   |        5 |
|        1 | PUZZA    | Maguy       | Ajaccio      |          4 | 2020-12-20   |        1 |
|        2 | ZUMORTU  | Amhed       | Ajaccio      |          5 | 2020-12-05   |        2 |
|        2 | ZUMORTU  | Amhed       | Ajaccio      |          6 | 2020-12-08   |        2 |
|        1 | PUZZA    | Maguy       | Ajaccio      |          7 | 2020-05-23   |        1 |
|        8 | RIEN     | Jean-S┼®    | Bastelicaccia |          9 | 2020-01-03   |        8 |
+----------+----------+-------------+--------------+------------+--------------+----------+
8 rows in set (0.009 sec)

MariaDB [jointures]> SELECT * FROM clients_ajaccio NATURAL JOIN commandes;
+----------+----------+-------------+--------------+------------+--------------+
| idclient | nomclient | prenomclient | ville        | idcommande | datecommande |
+----------+----------+-------------+--------------+------------+--------------+
|        1 | PUZZA    | Maguy       | Ajaccio      |          1 | 2020-11-04   |
|        1 | PUZZA    | Maguy       | Ajaccio      |          2 | 2020-11-04   |
|        5 | KOLIK    | Al          | Bastia       |          3 | 2020-10-05   |
|        1 | PUZZA    | Maguy       | Ajaccio      |          4 | 2020-12-20   |
|        2 | ZUMORTU  | Amhed       | Ajaccio      |          5 | 2020-12-05   |
|        2 | ZUMORTU  | Amhed       | Ajaccio      |          6 | 2020-12-08   |
|        1 | PUZZA    | Maguy       | Ajaccio      |          7 | 2020-05-23   |
|        8 | RIEN     | Jean-S┼®    | Bastelicaccia |          9 | 2020-01-03   |
+----------+----------+-------------+--------------+------------+--------------+
8 rows in set (0.001 sec)
```

INNER JOIN

- Même si c'est pratique, on préfère éviter les NATURAL JOIN en environnement pro car elles ne sont pas à proprement parler un type de jointures spécifique…
- Les INNER JOIN sont intuitives, mais peuvent ne pas suffire à obtenir toutes les informations que l'on veut ☺
- ATTENTION : Toutes les jointures qui suivent sont hors programme mais leur étude permet une compréhension bien plus fine de la puissance de MySQL
- Les diapos grisées sont donc facultatives

OUTER JOIN

- Et c'est là que les jointures externes, OUTER JOIN, interviennent…
- Exemple : deux tables sont liées par une association « one to many ». Je veux afficher les enregistrements de ces deux tables.
- Un SELECT simple sur les deux tables renverrait un produit cartésien (toutes les combinaisons possibles !) -> Faites le test

OUTER JOIN

- Graphiquement : il y a plusieurs représentations possibles, selon les cas. Nous les verrons au fur et à mesure

OUTER JOIN

- Je vais utiliser les OUTER JOIN. La première que nous allons voir est la LEFT (OUTER) JOIN. Si on a deux tables A et B, elle permet de retourner tous les enregistrements de A (plus précisément : au moins une fois chacun), même ceux qui ne sont pas dans B.
- Exemple : je veux tous les clients, même ceux qui n'ont pas passé de commande

OUTER JOIN

- Je vais utiliser les OUTER JOIN. La première que nous allons voir est la LEFT (OUTER) JOIN. Si on a deux tables A et B, elle permet de retourner tous les enregistrements de A, même ceux qui ne sont pas dans B.
- En d'autres termes, je renvoie tous les enregistrements de la table de gauche même ceux qui n'ont aucune correspondance dans celle de droite !
- Exemple : je veux tous les clients, même ceux qui n'ont jamais passé de commande

OUTER JOIN

```
MariaDB [jointures]> SELECT * FROM clients_ajaccio
    -> LEFT OUTER JOIN commandes USING(idclient);
+----------+-----------+-------------+--------------+------------+--------------+
| idclient | nomclient | prenomclient | ville        | idcommande | datecommande |
+----------+-----------+-------------+--------------+------------+--------------+
|        1 | PUZZA     | Maguy       | Ajaccio      |          1 | 2020-11-04   |
|        1 | PUZZA     | Maguy       | Ajaccio      |          2 | 2020-11-04   |
|        1 | PUZZA     | Maguy       | Ajaccio      |          4 | 2020-12-20   |
|        1 | PUZZA     | Maguy       | Ajaccio      |          7 | 2020-05-23   |
|        2 | ZUMORTU   | Amhed       | Ajaccio      |          5 | 2020-12-05   |
|        2 | ZUMORTU   | Amhed       | Ajaccio      |          6 | 2020-12-08   |
|        3 | PORTA     | Sarah       | Ajaccio      |       NULL | NULL         |
|        5 | KOLIK     | Al          | Bastia       |          3 | 2020-10-05   |
|        6 | TAZEGA    | Simon       | Bastia       |       NULL | NULL         |
|        7 | TALU      | Jean        | Corte        |       NULL | NULL         |
|        8 | RIEN      | Jean-S┼®    | Bastelicaccia |          9 | 2020-01-03   |
+----------+-----------+-------------+--------------+------------+--------------+
11 rows in set (0.004 sec)
```

OUTER JOIN

- Explication : chaque tuple distinct a été renvoyé. Un client qui a passé trois commandes apparaîtra sur trois lignes différentes du résultat. Un client qui n'a passé aucune commande aura NULL dans la colonne idcommande. Ce NULL n'est pas une chaîne de caractères ! J'ai utilisé USING car les noms des colonnes correspondent mais je pourrais tout aussi bien utiliser ON (comme dans INNER JOIN) qui est plus général

OUTER JOIN

- Bien entendu je peux juste sélectionner tous mes clients, ce qui *ressemblerait* à :

```
SELECT * FROM clients_ajaccio
```

- Sauf que nous disposons grâce au LEFT OUTER JOIN d'une information importante : les clients qui n'ont pas passé de commande !

OUTER JOIN

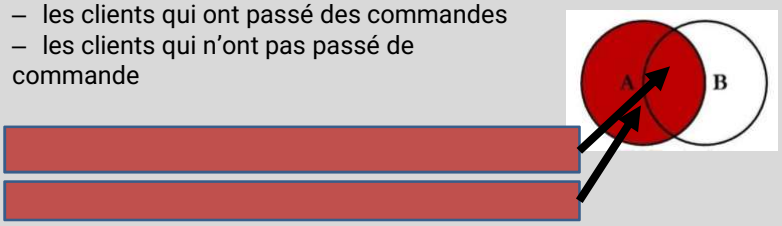
```
MariaDB [jointures]> SELECT * FROM clients_ajaccio
    -> LEFT OUTER JOIN commandes USING(idclient) GROUP BY idclient;
+----------+-----------+--------------+---------------+------------+--------------+
| idclient | nomclient | prenomclient | ville         | idcommande | datecommande |
+----------+-----------+--------------+---------------+------------+--------------+
|        1 | PUZZA     | Maguy        | Ajaccio       |          1 | 2020-11-04   |
|        2 | ZUMORTU   | Amhed        | Ajaccio       |          5 | 2020-12-05   |
|        3 | PORTA     | Sarah        | Ajaccio       |       NULL | NULL         |
|        5 | KOLIK     | Al           | Bastia        |          3 | 2020-10-05   |
|        6 | TAZEGA    | Simon        | Bastia        |       NULL | NULL         |
|        7 | TALU      | Jean         | Corte         |       NULL | NULL         |
|        8 | RIEN      | Jean-S┼®     | Bastelicaccia |          9 | 2020-01-03   |
+----------+-----------+--------------+---------------+------------+--------------+
7 rows in set (0.004 sec)
```

Résultat après groupement : les 4 premières colonnes correspondent à un SELECT * FROM clients_ajaccio

OUTER JOIN

On peut représenter graphiquement ce qui vient de se passer

- J'ai sélectionné tous mes clients en les joignant aux commandes, et j'obtiens l'union de deux ensembles:
  - les clients qui ont passé des commandes
  - les clients qui n'ont pas passé de commande



OUTER JOIN

- Assez facilement, je peux obtenir uniquement les clients qui n'ont jamais passé de commande !
- C'est quoi qui les caractérise ? C'est bien évidemment que l'idcommande, sur la ligne qui leur correspond, est NULL
- On utilise donc le prédicat IS NULL (ou dans certains cas IS NOT NULL) et on le met dans une condition WHERE, c'est tout ☺

OUTER JOIN

```
MariaDB [jointures]> SELECT * FROM clients_ajaccio
    -> LEFT OUTER JOIN commandes USING(idclient)
    -> WHERE idcommande IS NULL;
+----------+-----------+-------------+---------+------------+--------------+
| idclient | nomclient | prenomclient | ville   | idcommande | datecommande |
+----------+-----------+-------------+---------+------------+--------------+
|        3 | PORTA     | Sarah       | Ajaccio |       NULL | NULL         |
|        6 | TAZEGA    | Simon       | Bastia  |       NULL | NULL         |
|        7 | TALU      | Jean        | Corte   |       NULL | NULL         |
+----------+-----------+-------------+---------+------------+--------------+
3 rows in set (0.003 sec)
```

J'ai ici tous les clients qui n'ont jamais passé de commande

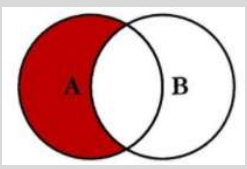
OUTER JOIN

- Bien sûr je peux affiner et enlever ces NULL de mon résultat

```
MariaDB [jointures]> SELECT idclient, prenomclient, nomclient FROM clients_ajaccio
    -> LEFT OUTER JOIN commandes USING(idclient)
    -> WHERE idcommande IS NULL;
+----------+--------------+-----------+
| idclient | prenomclient | nomclient |
+----------+--------------+-----------+
|        3 | Sarah        | PORTA     |
|        6 | Simon        | TAZEGA    |
|        7 | Jean         | TALU      |
+----------+--------------+-----------+
3 rows in set (0.001 sec)
```


OUTER JOIN

- On peut représenter ici aussi graphiquement ce qui vient de se passer
- J'ai sélectionné tous les clients y compris ceux qui n'avaient pas passé de commande grâce à LEFT OUTER JOIN, et ensuite j'ai filtré grâce à un WHERE ceux qui n'avaient jamais passé de commande (idcommande NULL)



OUTER JOIN

- Il est possible d'utiliser le fait que dans le cas de cette LEFT OUTER JOIN, les clients ayant passé une commande ont leur clef qui a migré dans commandes et donc la colonne idcommande n'est pas NULL dans ce cas
- Cela équivaut exactement à faire une INNER JOIN entre les deux tables

OUTER JOIN

```
MariaDB [jointures]> SELECT idclient, prenomclient, nomclient FROM clients_ajaccio
    -> LEFT OUTER JOIN commandes USING(idclient)
    -> WHERE idcommande IS NOT NULL;
+----------+--------------+-----------+
| idclient | prenomclient | nomclient |
+----------+--------------+-----------+
|        1 | Maguy        | PUZZA     |
|        1 | Maguy        | PUZZA     |
|        1 | Maguy        | PUZZA     |
|        1 | Maguy        | PUZZA     |
|        2 | Amhed        | ZUMORTU   |
|        2 | Amhed        | ZUMORTU   |
|        5 | Al           | KOLIK     |
|        8 | Jean-S┆®      | RIEN      |
+----------+--------------+-----------+
8 rows in set (0.006 sec)

MariaDB [jointures]> SELECT idclient, prenomclient, nomclient FROM clients_ajaccio
    -> INNER JOIN commandes USING(idclient)
    -> ;
+----------+--------------+-----------+
| idclient | prenomclient | nomclient |
+----------+--------------+-----------+
|        1 | Maguy        | PUZZA     |
|        1 | Maguy        | PUZZA     |
|        1 | Maguy        | PUZZA     |
|        1 | Maguy        | PUZZA     |
|        2 | Amhed        | ZUMORTU   |
|        2 | Amhed        | ZUMORTU   |
|        5 | Al           | KOLIK     |
|        8 | Jean-S┆®      | RIEN      |
+----------+--------------+-----------+
8 rows in set (0.000 sec)
```

OUTER JOIN

- Dernier exemple : je veux toutes les commandes, même celles ayant été passées par des clients n'existant plus

```
MariaDB [jointures]> SELECT * FROM commandes
    -> LEFT OUTER JOIN clients_ajaccio USING(idclient);
+----------+------------+--------------+-----------+--------------+---------------+
| idclient | idcommande | datecommande | nomclient | prenomclient | ville         |
+----------+------------+--------------+-----------+--------------+---------------+
|        1 |          1 | 2020-11-04   | PUZZA     | Maguy        | Ajaccio       |
|        1 |          2 | 2020-11-04   | PUZZA     | Maguy        | Ajaccio       |
|        5 |          3 | 2020-10-05   | KOLIK     | Al           | Bastia        |
|        1 |          4 | 2020-12-20   | PUZZA     | Maguy        | Ajaccio       |
|        2 |          5 | 2020-12-05   | ZUMORTU   | Amhed        | Ajaccio       |
|        2 |          6 | 2020-12-08   | ZUMORTU   | Amhed        | Ajaccio       |
|        1 |          7 | 2020-05-23   | PUZZA     | Maguy        | Ajaccio       |
|     NULL |          8 | 2020-11-04   | NULL      | NULL         | NULL          |
|        8 |          9 | 2020-01-03   | RIEN      | Jean-S®      | Bastelicaccia |
+----------+------------+--------------+-----------+--------------+---------------+
9 rows in set (0.001 sec)
```


OUTER JOIN

- Un autre type d'OUTER JOIN est le RIGHT OUTER JOIN

C'est en quelque sorte le symétrique de la LEFT OUTER JOIN

- Cette fois-ci, tous les enregistrements de la table de droite (B) seront renvoyés, même ceux n'ayant pas de correspondance dans la table de gauche (A)

OUTER JOIN

- On peut donc répondre encore une fois à la question : quels sont tous les clients, y compris ceux qui n'ont pas passé de commande ?

OUTER JOIN

```
MariaDB [jointures]> SELECT * FROM commandes
    -> RIGHT OUTER JOIN clients_ajaccio USING(idclient);
+----------+-----------+--------------+---------------+------------+--------------+
| idclient | nomclient | prenomclient | ville         | idcommande | datecommande |
+----------+-----------+--------------+---------------+------------+--------------+
|        1 | PUZZA     | Maguy        | Ajaccio       |          1 | 2020-11-04   |
|        1 | PUZZA     | Maguy        | Ajaccio       |          2 | 2020-11-04   |
|        1 | PUZZA     | Maguy        | Ajaccio       |          4 | 2020-12-20   |
|        1 | PUZZA     | Maguy        | Ajaccio       |          7 | 2020-05-23   |
|        2 | ZUMORTU   | Amhed        | Ajaccio       |          5 | 2020-12-05   |
|        2 | ZUMORTU   | Amhed        | Ajaccio       |          6 | 2020-12-08   |
|        3 | PORTA     | Sarah        | Ajaccio       |       NULL | NULL         |
|        5 | KOLIK     | Al           | Bastia        |          3 | 2020-10-05   |
|        6 | TAZEGA    | Simon        | Bastia        |       NULL | NULL         |
|        7 | TALU      | Jean         | Corte         |       NULL | NULL         |
|        8 | RIEN      | Jean-S®      | Bastelicaccia |          9 | 2020-01-03   |
+----------+-----------+--------------+---------------+------------+--------------+
11 rows in set (0.001 sec)
```

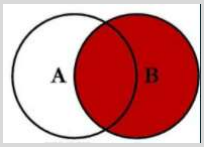
OUTER JOIN

- Ce résultat est identique à celui renvoyé par :

```
MariaDB [jointures]> SELECT * FROM clients_ajaccio
    -> LEFT OUTER JOIN commandes USING(idclient);
+----------+-----------+--------------+---------------+------------+--------------+
| idclient | nomclient | prenomclient | ville         | idcommande | datecommande |
+----------+-----------+--------------+---------------+------------+--------------+
|        1 | PUZZA     | Maguy        | Ajaccio       |          1 | 2020-11-04   |
|        1 | PUZZA     | Maguy        | Ajaccio       |          2 | 2020-11-04   |
|        1 | PUZZA     | Maguy        | Ajaccio       |          4 | 2020-12-20   |
|        1 | PUZZA     | Maguy        | Ajaccio       |          7 | 2020-05-23   |
|        2 | ZUMORTU   | Amhed        | Ajaccio       |          5 | 2020-12-05   |
|        2 | ZUMORTU   | Amhed        | Ajaccio       |          6 | 2020-12-08   |
|        3 | PORTA     | Sarah        | Ajaccio       |       NULL | NULL         |
|        5 | KOLIK     | Al           | Bastia        |          3 | 2020-10-05   |
|        6 | TAZEGA    | Simon        | Bastia        |       NULL | NULL         |
|        7 | TALU      | Jean         | Corte         |       NULL | NULL         |
|        8 | RIEN      | Jean-S┼®     | Bastelicaccia |          9 | 2020-01-03   |
+----------+-----------+--------------+---------------+------------+--------------+
11 rows in set (0.001 sec)
```


OUTER JOIN

- D'où la symétrie existant entre les jointures gauche et droite
- Graphiquement, une RIGHT OUTER JOIN peut se représenter ainsi :



OUTER JOIN

Et si je veux exclure les éléments communs :

```
MariaDB [jointures]> SELECT * FROM commandes
    -> RIGHT OUTER JOIN clients_ajaccio USING(idclient)
    -> WHERE idcommande IS NULL;
+----------+-----------+-------------+---------+------------+--------------+
| idclient | nomclient | prenomclient | ville   | idcommande | datecommande |
+----------+-----------+-------------+---------+------------+--------------+
|        3 | PORTA     | Sarah       | Ajaccio |       NULL | NULL         |
|        6 | TAZEGA    | Simon       | Bastia  |       NULL | NULL         |
|        7 | TALU      | Jean        | Corte   |       NULL | NULL         |
+----------+-----------+-------------+---------+------------+--------------+
3 rows in set (0.001 sec)
```

- On a également obtenu le même résultat auparavant, avec une LEFT OUTER JOIN entre clients_ajaccio et commandes

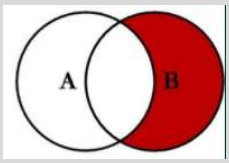
OUTER JOIN

- Comme le montrait l'exemple :

```
MariaDB [jointures]> SELECT * FROM clients_ajaccio
    -> LEFT OUTER JOIN commandes USING(idclient)
    -> WHERE idcommande IS NULL;
+----------+-----------+-------------+---------+------------+--------------+
| idclient | nomclient | prenomclient | ville   | idcommande | datecommande |
+----------+-----------+-------------+---------+------------+--------------+
|        3 | PORTA     | Sarah       | Ajaccio |       NULL | NULL         |
|        6 | TAZEGA    | Simon       | Bastia  |       NULL | NULL         |
|        7 | TALU      | Jean        | Corte   |       NULL | NULL         |
+----------+-----------+-------------+---------+------------+--------------+
3 rows in set (0.003 sec)
```

OUTER JOIN

- Graphiquement, cela se représente ainsi :



- Vous l'aurez compris, on peut obtenir là aussi l'intersection de ces ensembles

OUTER JOIN

```
MariaDB [jointures]> SELECT * FROM commandes
    -> RIGHT OUTER JOIN clients_ajaccio USING(idclient)
    -> WHERE idcommande IS NOT NULL;
+----------+-----------+-------------+--------------+------------+--------------+
| idclient | nomclient | prenomclient | ville        | idcommande | datecommande |
+----------+-----------+-------------+--------------+------------+--------------+
|        1 | PUZZA     | Maguy       | Ajaccio      |          1 | 2020-11-04   |
|        1 | PUZZA     | Maguy       | Ajaccio      |          2 | 2020-11-04   |
|        5 | KOLIK     | Al          | Bastia       |          3 | 2020-10-05   |
|        1 | PUZZA     | Maguy       | Ajaccio      |          4 | 2020-12-20   |
|        2 | ZUMORTU   | Amhed       | Ajaccio      |          5 | 2020-12-05   |
|        2 | ZUMORTU   | Amhed       | Ajaccio      |          6 | 2020-12-08   |
|        1 | PUZZA     | Maguy       | Ajaccio      |          7 | 2020-05-23   |
|        8 | RIEN      | Jean-S┼®    | Bastelicaccia |          9 | 2020-01-03   |
+----------+-----------+-------------+--------------+------------+--------------+
8 rows in set (0.004 sec)
```

- La seule différence est que le tri se fait sur idcommande

OUTER JOIN

- Quelques remarques supplémentaires
- On peut être tenté de faire une jointure pour trouver les commandes qui ont été passées par des clients n'existant plus. Pour cela, pas besoin de jointure :

```
MariaDB [jointures]> SELECT * FROM commandes WHERE idclient IS NULL;
+------------+--------------+----------+
| idcommande | datecommande | idclient |
+------------+--------------+----------+
|          8 | 2020-11-04   |     NULL |
+------------+--------------+----------+
1 row in set (0.025 sec)
```

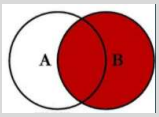

OUTER JOIN

- Si par contre on veut pour chaque commande le nom des clients (même n'existant plus), on doit faire une jointure

```
MariaDB [jointures]> SELECT * FROM clients_ajaccio RIGHT OUTER JOIN commandes USING(idclient);
+----------+------------+--------------+-----------+--------------+----------------+
| idclient | idcommande | datecommande | nomclient | prenomclient | ville          |
+----------+------------+--------------+-----------+--------------+----------------+
|        1 |          1 | 2020-11-04   | PUZZA     | Maguy        | Ajaccio        |
|        1 |          2 | 2020-11-04   | PUZZA     | Maguy        | Ajaccio        |
|        5 |          3 | 2020-10-05   | KOLIK     | Al           | Bastia         |
|        1 |          4 | 2020-12-20   | PUZZA     | Maguy        | Ajaccio        |
|        2 |          5 | 2020-12-05   | ZUMORTU   | Amhed        | Ajaccio        |
|        2 |          6 | 2020-12-08   | ZUMORTU   | Amhed        | Ajaccio        |
|        1 |          7 | 2020-05-23   | PUZZA     | Maguy        | Ajaccio        |
|     NULL |          8 | 2020-11-04   | NULL      | NULL         | NULL           |
|        8 |          9 | 2020-01-03   | RIEN      | Jean-S├®     | Bastelicaccia  |
+----------+------------+--------------+-----------+--------------+----------------+
9 rows in set (0.005 sec)
```

OUTER JOIN

- Graphiquement, on a fait ceci



- Et toute la différence entre les jointures à gauche et à droite est là ! La même requête avec une jointure gauche donnerait tous les clients, même ceux n'ayant jamais commandé

OUTER JOIN

```
MariaDB [jointures]> SELECT * FROM clients_ajaccio LEFT OUTER JOIN commandes USING(idclient);
+----------+-----------+-------------+--------------+------------+--------------+
| idclient | nomclient | prenomclient | ville        | idcommande | datecommande |
+----------+-----------+-------------+--------------+------------+--------------+
|        1 | PUZZA     | Maguy       | Ajaccio      |          1 | 2020-11-04   |
|        1 | PUZZA     | Maguy       | Ajaccio      |          2 | 2020-11-04   |
|        1 | PUZZA     | Maguy       | Ajaccio      |          4 | 2020-12-20   |
|        1 | PUZZA     | Maguy       | Ajaccio      |          7 | 2020-05-23   |
|        2 | ZUMORTU   | Amhed       | Ajaccio      |          5 | 2020-12-05   |
|        2 | ZUMORTU   | Amhed       | Ajaccio      |          6 | 2020-12-08   |
|        3 | PORTA     | Sarah       | Ajaccio      |       NULL | NULL         |
|        5 | KOLIK     | Al          | Bastia       |          3 | 2020-10-05   |
|        6 | TAZEGA    | Simon       | Bastia       |       NULL | NULL         |
|        7 | TALU      | Jean        | Corte        |       NULL | NULL         |
|        8 | RIEN      | Jean-S┼®    | Bastelicaccia |          9 | 2020-01-03   |
+----------+-----------+-------------+--------------+------------+--------------+
11 rows in set (0.000 sec)
```

OUTER JOIN

- Pour terminer sur les right join : on voudrait récupérer uniquement les commandes passées par des clients encore actifs dans la base (non supprimés)

OUTER JOIN

```
MariaDB [jointures]> SELECT * FROM clients_ajaccio RIGHT OUTER JOIN commandes USING(idclient)
    -> WHERE idclient IS NOT NULL;
+----------+------------+--------------+-----------+--------------+---------------+
| idclient | idcommande | datecommande | nomclient | prenomclient | ville         |
+----------+------------+--------------+-----------+--------------+---------------+
|        1 |          1 | 2020-11-04   | PUZZA     | Maguy        | Ajaccio       |
|        1 |          2 | 2020-11-04   | PUZZA     | Maguy        | Ajaccio       |
|        5 |          3 | 2020-10-05   | KOLIK     | Al           | Bastia        |
|        1 |          4 | 2020-12-20   | PUZZA     | Maguy        | Ajaccio       |
|        2 |          5 | 2020-12-05   | ZUMORTU   | Amhed        | Ajaccio       |
|        2 |          6 | 2020-12-08   | ZUMORTU   | Amhed        | Ajaccio       |
|        1 |          7 | 2020-05-23   | PUZZA     | Maguy        | Ajaccio       |
|        8 |          9 | 2020-01-03   | RIEN      | Jean-S®      | Bastelicaccia |
+----------+------------+--------------+-----------+--------------+---------------+
8 rows in set (0.001 sec)
```

OUTER JOIN

- La requête précédente est comme vous l'avez deviné exactement la même que :

```
MariaDB [jointures]> SELECT * FROM commandes LEFT OUTER JOIN clients_ajaccio USING(idclient)
    -> WHERE idclient IS NOT NULL;
+----------+------------+--------------+-----------+--------------+---------------+
| idclient | idcommande | datecommande | nomclient | prenomclient | ville         |
+----------+------------+--------------+-----------+--------------+---------------+
|        1 |          1 | 2020-11-04   | PUZZA     | Maguy        | Ajaccio       |
|        1 |          2 | 2020-11-04   | PUZZA     | Maguy        | Ajaccio       |
|        5 |          3 | 2020-10-05   | KOLIK     | Al           | Bastia        |
|        1 |          4 | 2020-12-20   | PUZZA     | Maguy        | Ajaccio       |
|        2 |          5 | 2020-12-05   | ZUMORTU   | Amhed        | Ajaccio       |
|        2 |          6 | 2020-12-08   | ZUMORTU   | Amhed        | Ajaccio       |
|        1 |          7 | 2020-05-23   | PUZZA     | Maguy        | Ajaccio       |
|        8 |          9 | 2020-01-03   | RIEN      | Jean-S├®     | Bastelicaccia |
+----------+------------+--------------+-----------+--------------+---------------+
8 rows in set (0.005 sec)
```


OUTER JOIN

On veut juste les id de commande, c'est possible aussi ☺

```
MariaDB [jointures]> SELECT idcommande FROM clients_ajaccio RIGHT OUTER JOIN commandes USING(idclient)
    -> WHERE idclient IS NOT NULL;
+------------+
| idcommande |
+------------+
|          1 |
|          2 |
|          4 |
|          7 |
|          5 |
|          6 |
|          3 |
|          9 |
+------------+
8 rows in set (0.003 sec)

MariaDB [jointures]> SELECT idcommande FROM commandes LEFT OUTER JOIN clients_ajaccio USING(idclient)
    -> WHERE idclient IS NOT NULL;
+------------+
| idcommande |
+------------+
|          1 |
|          2 |
|          4 |
|          7 |
|          5 |
|          6 |
|          3 |
|          9 |
+------------+
8 rows in set (0.000 sec)
```

OUTER JOIN

- Ne vous laissez pas abuser par le SELECT : je peux sélectionner n'importe lequel des champs retournés par ma requête, donc

SELECT champ_table_B FROM table_A…

fonctionnera si j'effectue une jointure avec la table B dans ma requête ! On l'a déjà vu pour les INNER JOIN d'ailleurs

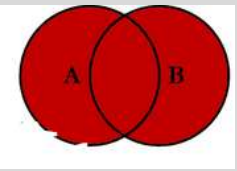
OUTER JOIN

- Enfin, j'ai toujours utilisé OUTER dans mes jointures pour bien fixer les idées, mais il peut être omis, ce que l'on fait couramment

LEFT OUTER JOIN = LEFT JOIN
RIGHT OUTER JOIN = RIGHT JOIN

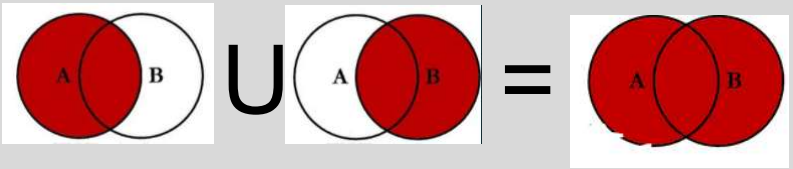
OUTER JOIN

- Enfin, il y nous reste à étudier des jointures externes particulières : les FULL OUTER JOIN
- Non disponibles nativement dans MySQL
- Moins utilisées que les autres
- Fournissent une union de deux tables, soit graphiquement :



OUTER JOIN

- Par exemple, je veux voir toutes les commandes, y compris celles dont le client n'existe plus, et tous les clients, y compris ceux qui n'ont pas passé de commande
- Je vais simuler un FULL OUTER JOIN en faisant une union



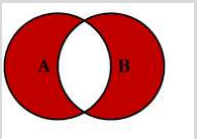
OUTER JOIN

```
MariaDB [jointures]> SELECT idcommande, nomclient FROM commandes
    -> LEFT JOIN clients_ajaccio USING(idclient)
    -> UNION
    -> SELECT idcommande, nomclient FROM commandes
    -> RIGHT JOIN clients_ajaccio USING(idclient);
+------------+-----------+
| idcommande | nomclient |
+------------+-----------+
|          8 | NULL      |
|          1 | PUZZA     |
|          2 | PUZZA     |
|          4 | PUZZA     |
|          7 | PUZZA     |
|          5 | ZUMORTU   |
|          6 | ZUMORTU   |
|          3 | KOLIK     |
|          9 | RIEN      |
|       NULL | PORTA     |
|       NULL | TAZEGA    |
|       NULL | TALU      |
+------------+-----------+
12 rows in set (0.001 sec)
```


OUTER JOIN

- Enfin, je veux aussi sélectionner uniquement les clients n'ayant pas passé de commandes, et les commandes ayant été passées par un client n'existant plus

Graphiquement, ça donnerait ça :



Mathématiquement, ça serait :
A U B / A ⋂ B (« A union B, privé de A inter B »)

OUTER JOIN

```
MariaDB [jointures]> SELECT idcommande, nomclient FROM clients_ajaccio
    -> LEFT JOIN commandes USING(idclient)
    -> WHERE idcommande IS NULL
    -> UNION
    -> SELECT idcommande, nomclient FROM clients_ajaccio
    -> RIGHT JOIN commandes USING(idclient)
    -> WHERE idclient IS NULL;
+------------+-----------+
| idcommande | nomclient |
+------------+-----------+
|       NULL | PORTA     |
|       NULL | TAZEGA    |
|       NULL | TALU      |
|          8 | NULL      |
+------------+-----------+
4 rows in set (0.005 sec)
```


OUTER JOIN

Bonus (attention certaines syntaxes ne sont pas disponibles avec MySQL)