R207
Sources de données

Manuel Munier

UPPA STEE - IUT des Pays de l'Adour - Département RT
manuel.munier@univ-pau.fr
https://munier.perso.univ-pau.fr/teaching/butrt-r207/

2025-2026

Plan du cours

Plan du cours

Ce que dit le PPN

- Objectifs du module
  - ▷ Introduction aux systèmes de gestion de base de données
  - ▷ Stockage et codage de l'information en fonction des données et de leur usage
  - ▷ Langages et outils spécifiques pour l'accès aux données
- Compétences visées (entre autres)
  - → Comprendre l'architecture des systèmes numériques
  - → Choisir les mécanismes de gestion de données adaptés
  - → Lire, exécuter, corriger et modifier un programme
- Pré-requis
  - ▷ R107 : Fondamentaux de la programmation
  - ▷ R109 : Introduction aux technologies Web

Ce que dit le PPN

- Contenus
  - ▷ Stockage et accès aux données
    - Système de gestion de données (relationnel/non relationnel)
    - Structuration des données : fichiers (CSV, JSON), exemples de sources ouvertes (open data), web scraping
    - Sensibilisation à la réglementation française et internationale (CNIL, RGPD)
  - ▷ Base de données relationnelles
    - Schéma relationnel d'une base de données
    - Sensibilisation aux contraintes d'intégrité
    - Création de tables simples
    - Interrogation de données, ajout et modification de données
  - ▷ Lecture d'une documentation technique (UML, diagramme de classes)
- Volume horaire : ~20h
  - → Cours 3h / TD 4h30 / TP 12h *(y compris exam & éval TP ! ! ! ☹)*

Ce que l'on va (essayer) de faire...

- Contenu de ce module
  - ✓ BdD & SQL
  - ✓ requêtes en Python sur des tableaux ?
  - ✓ utilisation JSON en Python
  - ✓ accès à des données Open Data ?
    (sous forme de "recette de cuisine")
- **Ce que nous ne ferons pas ! ! !**
  - ✗ Algorithmique : boucles, tests, tableaux, fonctions,...
  - ✗ Ligne de commande : gestion fichiers & répertoires,...

⇒ `if (lacunes>0) then do_révisions([R107,R109]);`

Pourquoi s'intéresser aux données ?

- Sur de petits projets
  - peu de données $\rightsquigarrow$ de "simples" tableaux suffisent
  - peu de composants logiciels $\rightsquigarrow$ échanges $\equiv$ passage de paramètres & return dans les fonctions
  - et parfois... on structure "un peu" $\Rightarrow$ tuples, tableaux à plusieurs dimensions, etc.
- Mais dans la vraie vie...
  - **beaucoup** plus de données
  - des données **structurées** : enregistrements, listes, dictionnaires, objets, etc.
  - différentes **technologies** selon les usages
  - traitements plus **complexes**

Plan du cours

Déroulement

- Très peu de temps (2 cours d'1h30 ☹), donc :
  - pas le temps de faire de la théorie ⇒ apprentissage par la pratique, sur des exemples
  - on va se concentrer sur les fondamentaux :
    - **BdD** ⇝ tables, langage SQL, contraintes, schéma relationnel
    - d'autres façons de stocker des données ⇝ formats CSV, **JSON**
    - utilisation de données disponible sur Internet (ex : **Open Data**)
    - **aspects juridiques**, car on ne peut pas faire ce que l'on veut avec les données !

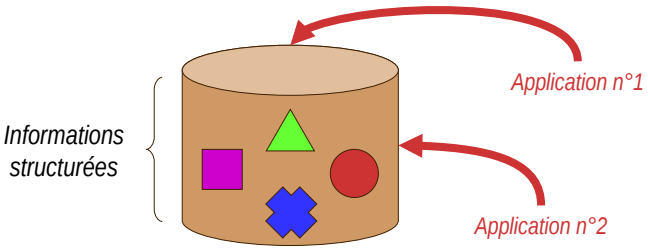
Déroulement

1. SQL
   - intérêt d'utiliser une BdD relationnelle
   - structure & schéma E-A (schéma conceptuel) / tables (schéma physique)
   - langage de requête $\rightsquigarrow$ SQL
2. Utiliser SQL depuis un langage de programmation (ex : Python)
3. Une autre façon de stocker des données "moins rigide" $\rightsquigarrow$ le format JSON
   - utiliser JSON en Python
   - accéder à des données en Open Data
4. Droit & numérique $\rightsquigarrow$ CNIL, RGPD, DGA,...

Plan du cours

Introduction : définition d'une BdD

- Définition informelle d'une BdD
  - Une **base de données** (BdD) est un ensemble **structuré** d'informations **persistantes** partagées par plusieurs **applications** d'une même **entreprise**.



Introduction : définition d'une BdD

- Ensemble structuré
  - ▷ les données ont une structure qui a été définie (fixée) une fois pour toutes
  - ▷ cette structure (organisation) doit donc être définie en fonction de l'exploitation ultérieure de ces données
- Plusieurs applications
  - ▷ les utilisateurs, au travers de plusieurs applications, se partagent ces données mais peuvent avoir des intérêts différents

Introduction : définition d'une BdD

- Entreprise
  - ▷ les applications ne sont pas complètement indépendantes les unes des autres : elles appartiennent à la même entreprise
  - ▷ au sens large : université, banque, PME/PMI,...
- Informations persistantes
  - ▷ les données sont conservées de manière permanente (persistance) et elles sont disponibles pour chaque application, sans qu'il soit nécessaire de les réintroduire dans le système à chaque fois

Introduction : exemple

- Données
  - liste des étudiants inscrits,
  - liste des cours,
  - liste des enseignants,
  - emplois du temps,
  - relevés de notes,...
- Applications
  - gestion des inscriptions,
  - planning des salles,
  - jurys d'examens,...
- Entreprise
  - université

Introduction : exploitation d'une BdD

- On peut imaginer que les données sont stockées dans des fichiers (ou des tableaux)
- Mais les informations stockées ne sont pas les seules informations accessibles !
  - « Pourcel suit le cours de BdD en RT »
  - « Munier est l'enseignant de BdD en RT »
- On peut en déduire l'information
  - « Munier enseigne les BdD à Pourcel »

Introduction : exploitation d'une BdD

- Une BdD contient à la fois
  - des informations représentant des objets du monde extérieur
  - des liens sémantiques entre ces objets

- Exploiter une BdD c'est savoir
  - insérer de nouvelles informations
  - extraire, parmi toutes ces informations, celles dont on a besoin
  - manipuler les relations entre ces informations ("croiser les données")

Introduction : objectifs d'une BdD

- Une approche BdD nous apporte
  - ▷ intégration
    - toutes les données d'une entreprise sont placées dans un entrepôt commun à toutes les applications qui y puisent les données les concernant
  - ▷ flexibilité
    - données indépendantes d'une application particulière
    - SGBD ⇒ indépendance vis-à-vis du support physique
  - ▷ disponibilité
    - persistance, réseau, performances du serveur,...
  - ▷ sécurité
    - pannes, confidentialité, accès concurrents,...

Introduction : survol des bases de données

- Formation express
  - 1er TD et TP après seulement 1h30 de cours
  - dès les 1ers exercices il faut savoir interroger une BdD
- Que va-t-on voir ?
  - comment sont structurées les données (niveau conceptuel)
  - comment elles sont représentées dans la BdD (niveau physique $\rightsquigarrow$ tables)
  - comment exploiter une BdD $\rightsquigarrow$ langage de requêtes (SQL)

Plan du cours

Conception d'une BdD

- Formalisme utilisé ≡ modèles entités-associations (dû à Chen)
  - ou *modèle individus-relations*
  - ou *modèle individuel*

- But : décrire le réel perçu à l'aide de ce formalisme
  - ▷ à partir d'un cahier des charges
  - ▷ par analyse d'un système existant
  - ▷ etc.

Conception d'une BdD : modèle E-A (entités)

- Une **entité** est un objet physique ou abstrait ayant une existence propre et pouvant être différencié par rapport aux autres objets
  - objets physiques
    - personne, voiture,...
  - objet abstrait
    - « vol `#IJ509` pour Metz partant de Bordeaux »
  - contre-exemple
    - un « livre de BdD écrit en 1975 » n'est pas une entité car il n'est pas unique

Conception d'une BdD : modèle E-A (entités)

- Une entité est décrite par l'ensemble de ses propriétés appelées **attributs**
  - ne pas confondre le nom de l'attribut et sa valeur
    - `NumVol` → nom
    - `#IJ509` → valeur
  - les valeurs doivent appartenir à un ensemble de valeurs (domaine)
    - `D1 = {Pourcel,Cousy,Burgy} = NomEtudiant`
    - `D2 = {BdD,Java,SE,Réseaux} = NomCours`
    - `D3 = {x | x ∈ [0..20]} = Note`

Conception d'une BdD : modèle E-A (entités)

- Les 3 entités suivantes ont la même structure (i.e. les mêmes attributs)
  - `(C1,Java,Munier,12,10.5,15)`
  - `(C2,Réseaux,Bascou,24,30,30)`
  - `(C3,TransNum,Baillot,12,18,24)`
- On dit qu'elles appartiennent à une même classe d'entité ou **type d'entité**
- Ici la classe `Cours` est caractérisée par ses 6 attributs `NumCours`, `NomCours`, `NomProf`, `NbHC`, `NbHTD` et `NbHTP`

Conception d'une BdD : modèle E-A (entités)

- Représentation graphique du type d'entité `Cours`

Cours
NumCours: entier
NomCours: chaîne NomProf : chaîne NbHC : réel NbHTD : réel NbHTP : réel

- ▷ **identifiant** (ou **clé**) = attribut(s) permettant d'identifier de manière unique une entité
- ▷ en effet, l'attribut `NomCours` ne suffit pas ; il peut exister du SE en 1ère et en 2nde année
- ▷ si aucun attribut ne convient, il suffit de créer artificiellement un identifiant unique pour chaque entité (ex : `R207`)

Conception d'une BdD : modèle E-A (entités)

- Type d'entité `Etudiant`

Etudiant
NumEtud : entier
Nom : chaîne **Prenom : chaîne** **Adresse : chaîne** **DateNais: date** **Sexe : {M,F}**

- ▷ le couple d'attributs `(Nom,Prenom)` ne peut pas servir de clé car ce n'est pas suffisant pour identifier de manière unique un étudiant $\rightsquigarrow$ cas des homonymes

$\Rightarrow$ on crée un attribut `NumEtud` qui servira d'identifiant

Conception d'une BdD : modèle E-A (entités)

- Au niveau physique, un type d'entité sera représenté par une table
  - chaque **colonne** correspond à un **attribut**
  - chaque **ligne** représente une **entité** de ce type

Cours

NumCours	NomCours	NomProf	NbHC	NbHTD	NbHTP
C1	Java	Munier	12	10,5	15
C2	Réseaux	Bascou	24	30	30
C3	TransNum	Baillot	12	18	24

Etudiant

NumEtud	Nom	Prenom	Adresse	DateNais	Sexe
E1	Cousy	Cécile	?	?	F
E2	Pourcel	Mathieu	?	?	M
E3	Burgy	Laurent	?	?	M

Conception d'une BdD : modèle E-A (entités)

- Certains liens entre les informations n'existent pas encore
  - lien entre les entités `Cours` et `Etudiant` pour indiquer que tel étudiant suit tel cours
- Il nous faut donc enrichir ce modèle pour prendre en compte ces informations mettant en relation plusieurs entités

Conception d'une BdD : modèle E-A (associations)

- Une **association d'entités** est un regroupement de deux ou plusieurs entités pour représenter une réalité de l'organisation

- Soit les 2 entités suivantes :
  - `(C1,Java,Munier,12,10.5,15)`
  - `(E1,Cousy,Cécile,?,?,F)`

- L'association ci-dessous exprime le fait que l'étudiant(e) `E1` a suivi le cours `C1` et a obtenu la note de `18.5`
  - `(E1,C1,18.5)`

Conception d'une BdD : modèle E-A (associations)

- Un type d'association (d'entités) est un sous-ensemble d'un produit cartésien d'entités. Il permet de représenter les informations n'ayant de sens que par rapport à l'association de certains types d'entités.

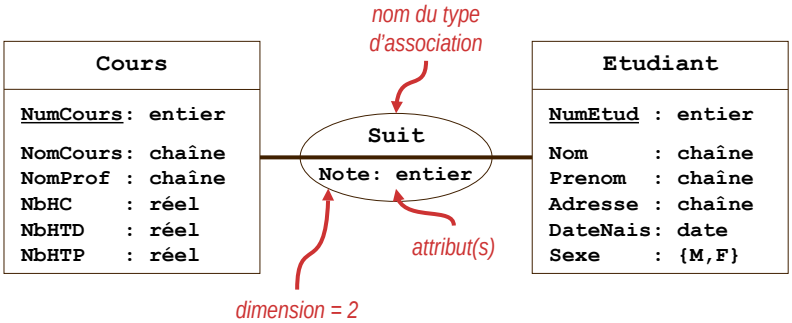
- Bref, c'est un **lien sémantique** entre plusieurs entités ☺

Conception d'une BdD : modèle E-A (associations)

- Propriétés d'un type d'association
  - ▷ attributs
    - au minimum les identifiants des types d'entités reliés
    - éventuellement des attributs spécifiques (ex : `Note`)
  - ▷ identifiant
    - concaténation des identifiants des types d'entités reliés (ici le couple `(NumEtud,NumCours)`)
    - NB : les identifiants des types d'association doivent eux aussi être uniques
  - ▷ dimension
    - nombre de types d'entités reliés (généralement 2 pour simplifier)

Conception d'une BdD : modèle E-A (associations)

- Représentation graphique du type d'association `Suit`



Conception d'une BdD : modèle E-A (associations)

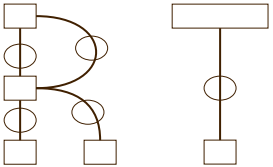
- Au niveau physique, un type d'association sera également représenté par une table
  - chaque **colonne** correspond à un **attribut**
  - chaque **ligne** représente une **association** de ce type entre 2 entités

Suit

NumEtud	NumCours	Note
E1	C1	18,5
E1	C2	15
E2	C1	9,5

Conception d'une BdD : modèle E-A

- Il y a encore beaucoup d'autres choses à dire sur le modèle entités-associations... mais pas assez de temps dans ce module R207 ☹
  - notations supplémentaires
  - extensions au modèle E-A
  - comment le concevoir intelligemment
  - optimisations (limiter les redondances,...)



Plan du cours

Exploitation d'une BdD

- Pour accéder à une base de données, on distingue 2 outils
  - LDD (Langage de Définition des Données)
    - création du schéma : tables, index,...
    - gestion des droits d'accès
  - LMD (Langage de Manipulation des Données)
    - manipulation des informations stockées $\rightsquigarrow$ insertion, modification, suppression
    - recherche d'informations $\rightsquigarrow$ requêtes
    - statistiques sur ces informations $\rightsquigarrow$ opérateurs

Langage SQL

- **SQL** ≡ Structured Query Language
  - langage de requête structuré
  - conçu par IBM en 1974, normalisé en 1986
  - norme SQL3 définie en 1999 ; actuellement SQL:2011

- SQL est à la fois un LDD et un LMD

- Dans cette initiation nous allons nous contenter de voir comment interroger une BdD à l'aide de l'instruction `select` (forme simplifiée)

Langage SQL : instruction `select`

- Construction de base d'une requête SQL

```
select a1,...,an
from T1,...,Tn
where B
```

- avec :
  - les a_i sont des attributs et représentent le résultat attendu de la requête
  - les T_i indiquent quelles sont les tables concernées par cette requête $\rightsquigarrow$ où vont être récupérées les informations
  - B est un prédicat (condition booléenne) sur les a_i

Langage SQL : instruction `select`

- Remarques
  - on peut éliminer les tuples en double dans le résultat en faisant précéder la liste des attributs citée dans le `select` par le mot clé `distinct`

    ```
    select distinct a1,...,an
    from T1,...,Tn
    where B
    ```

  - `*` est une convention qui inclut tous les attributs des tables citées dans le `from`
  - la clause `where` est facultative

Langage SQL : instruction select

Etudiant

NumEtud	Nom	Prenom	Adresse	DateNais	Sexe
E1	Cousy	Cécile	?	?	F
E2	Pourcel	Mathieu	?	?	M
E3	Burgy	Laurent	?	?	M

Cours

NumCours	NomCours	NomProf	NbHC	NbHTD	NbHTP
C1	Java	Munier	12	10,5	15
C2	Réseaux	Bascou	24	30	30
C3	TransNum	Baillot	12	18	24

Suit

NumEtud	NumCours	Note
E1	C1	18,5
E1	C2	15
E2	C1	9,5

Langage SQL : instruction `select`

Etudiant

NumEtud	Nom	Prenom	Adresse	DateNais	Sexe
E1	Cousy	Cécile	?	?	F
E2	Pourcel	Mathieu	?	?	M
E3	Burgy	Laurent	?	?	M

Cours

NumCours	NomCours	NomProf	NbHC	NbHTD	NbHTP
C1	Java	Munier	12	10,5	15
C2	Réseaux	Bascou	24	30	30
C3	TransNum	Baillot	12	18	24

Suit

NumEtud	NumCours	Note
E1	C1	18,5
E1	C2	15
E2	C1	9,5

requête SQL

```
select NumEtud,Nom,Prenom,Sexe
from Etudiant
where Sexe='M'
```

résultat

```
NumEtud  Nom      Prenom  Sexe
-------- -------- ------- -----
E2       Pourcel  Mathieu M
E3       Burgy    Laurent M

2 ligne(s) selectionnee(s)
```

Langage SQL : instruction select

Etudiant

NumEtud	Nom	Prenom	Adresse	DateNais	Sexe
E1	Cousy	Cécile	?	?	F
E2	Pourcel	Mathieu	?	?	M
E3	Burgy	Laurent	?	?	M

Cours

NumCours	NomCours	NomProf	NbHC	NbHTD	NbHTP
C1	Java	Munier	12	10,5	15
C2	Réseaux	Bascou	24	30	30
C3	TransNum	Baillot	12	18	24

Suit

NumEtud	NumCours	Note
E1	C1	18,5
E1	C2	15
E2	C1	9,5

« algorithme »

Exécution pas-à-pas de la requête `select`...

requête SQL

```
select Nom,Prenom,NomCours,NomProf,Note
from Etudiant,Cours,Suit
where (Suit.NumEtud = Etudiant.NumEtud)
  and (Suit.NumCours = Cours.NumCours)
```

résultat

```
Nom      Prenom  NomCours   NomProf    Note
-------  ------- ---------  ---------  -----
Cousy    Cécile  Java       Munier     18.5
Cousy    Cécile  Réseaux    Bascou     15
Pourcel  Mathieu Java       Munier     9.5

3 ligne(s) selectionnee(s)
```

Langage SQL : instruction `select`

Etudiant

NumEtud	Nom	Prenom	Adresse	DateNais	Sexe
E1	Cousy	Cécile	?	?	F
E2	Pourcel	Mathieu	?	?	M
E3	Burgy	Laurent	?	?	M

Cours

NumCours	NomCours	NomProf	NbHC	NbHTD	NbHTP
C1	Java	Munier	12	10,5	15
C2	Réseaux	Bascou	24	30	30
C3	TransNum	Baillot	12	18	24

Suit

NumEtud	NumCours	Note
E1	C1	18,5
E1	C2	15
E2	C1	9,5

requête SQL

```
select NomCours,NomProf
from Cours,Suit
where (Suit.NumCours = Cours.NumCours)
```

résultat

```
NomCours  NomProf
--------- ---------
Java      Munier      <- (E1,C1)
Réseaux   Bascou      <- (E1,C2)
Java      Munier      <- (E2,C1)

3 ligne(s) selectionnee(s)
```

Langage SQL : instruction `select`

Etudiant

NumEtud	Nom	Prenom	Adresse	DateNais	Sexe
E1	Cousy	Cécile	?	?	F
E2	Pourcel	Mathieu	?	?	M
E3	Burgy	Laurent	?	?	M

Cours

NumCours	NomCours	NomProf	NbHC	NbHTD	NbHTP
C1	Java	Munier	12	10,5	15
C2	Réseaux	Bascou	24	30	30
C3	TransNum	Baillot	12	18	24

Suit

NumEtud	NumCours	Note
E1	C1	18,5
E1	C2	15
E2	C1	9,5

requête SQL

```
select distinct NomCours,NomProf
from Cours,Suit
where (Suit.NumCours = Cours.NumCours)
```

résultat

```
NomCours  NomProf
--------- ---------
Java      Munier       <- distinct supprime
Réseaux   Bascou          les doublons

2 ligne(s) selectionnee(s)
```

Langage SQL : instruction `select`

- Opérateurs de comparaison
  - `=` `>=` `>` `!=` `<` `<=`
  - `between ... and ...`
  - `in`
  - `is null`
  - `is not null`
  - `like ...`

- Opérateurs booléens
  - `not`
  - `and`
  - `or`

Langage SQL : instruction `select`

- Exemples de conditions
  - la note est comprise entre 8 et 16
    `Note between 8 and 16`
  - le nom du cours est Java, BdD ou Réseaux
    `NomCours in ('Java','BdD','Réseaux')`
  - le prénom commence par la lettre C
    `Prenom like 'C%'`

Langage SQL : instruction select

- On peut trier les tuples retournés par une requête en ajoutant une clause order by

  - recherche de tous les cours réalisés par Munier ou Gallon et les affiche par ordre croissant de NbHC, puis NbHTD, puis NbHTP

```
select *
from Cours
where NomProf in ('Munier','Gallon')
order by NbHC,NbHTD,NbHTP
```

Langage SQL : instruction select

- Le tri peut se faire
  - par ordre croissant (ASC par défaut)
  - par ordre décroissant (DESC)

```
rem Classement au partiel de Java
select Nom,Prenom,Note
from Suit,Etudiant,Cours
where (Suit.NumEtud = Etudiant.NumEtud)
  and (Suit.NumCours = Cours.NumCours)
  and (NomCours = 'Java')
order by Note DESC
```

Conclusion (forme simple du `select`)

- On a vu le strict minimum sur les BdD relationnelles pour avoir une idée de leur fonctionnement
  - ▷ pourquoi utiliser des BdD
  - ▷ comment les concevoir
    - schéma entités-associations
    - notion d'identifiant (clé)
    - entités et associations sont traduites en tables
  - ▷ comment les exploiter
    - un langage de requêtes ⇒ SQL
    - forme de base d'une requête SQL ⇒ `select`

Langage SQL : select "avancé"

- Le **renommage** permet de définir des alias sur les noms des tables
  - quand une même table apparaît plusieurs fois dans une jointure, des sous-requêtes,...
  - quand il y a ambiguïté sur le nom d'un attribut (même attribut dans plusieurs tables)
  - pour simplifier l'écriture des requêtes

- Exemple
  - ```
    select C1.nomCours, C2.nomCours
    from Cours C1, Cours C2
    where C1.nomProf=C2.nomProf
    ```
    ```

Langage SQL : `select` "avancé"

- Algèbre relationnelle (sur les ensembles)
  - ▷ dans une base de données
    - les données sont rangées dans des tables
    - une requête est un algorithme dont les paramètres sont des tables de la base
    - le résultat d'une requête est également une table
  - ▷ idée
    - pouvoir utiliser la table résultant d'une requête comme paramètre d'un autre algorithme d'interrogation
  - ▷ NB : on a déjà vu plusieurs opérateurs (de l'algèbre relationnelle)
    - produit cartésien $\rightsquigarrow$ clause `from`
    - sélection (aka "filtre") $\rightsquigarrow$ clause `where`
    - projection $\rightsquigarrow$ clause `select`

Langage SQL : select "avancé"

- Opérateurs ensemblistes
  - ⟨sélection 1⟩ UNION ⟨sélection 2⟩
  - ⟨sélection 1⟩ INTERSECT ⟨sélection 2⟩
  - ⟨sélection 1⟩ MINUS ⟨sélection 2⟩

- Exemple : fournisseurs qui nont livré aucun produit
  - ```
    SELECT numFour FROM Fournisseur
    MINUS
    SELECT numFour FROM Stock
    ```
    ```

Langage SQL : `select` "avancé"

- Remarques sur les opérateurs ensemblistes
  - ▷ les deux sélections sont des requêtes `SELECT` dont le nombre et le type des attributs sélectionnés doivent être identiques $\rightsquigarrow$ les 2 ensembles doivent avoir la même structure
  - ▷ `ALL`, cité après `UNION`, évite l'élimination des tuples en double
  - ▷ dans certaines implémentation de SQL, il se peut que `MINUS` soit noté `EXCEPT`

Langage SQL : select "avancé" ⇝ sous-requêtes

- Opérateurs sur les ensembles utilisables dans les prédicats (ex : `where`)
  - `[not] in (⟨sélection⟩)`
  - `[not] exists (⟨sélection⟩)`
  - `[some|any|all] (⟨sélection⟩)`

- Exemples
  - nom des cours suivis par au moins un étudiant

```
select nomCours from Cours C
where exists(select * from Suit S where C.numCours=S.numCours)
```

  - nom du (des) fournisseur(s) avec la plus forte remise

```
select nomFour from Fournisseur
where remise >= all(select remise from Fournisseur)
```

Langage SQL : `select` "avancé" $\rightsquigarrow$ sous-requêtes

- Avantages à utiliser des sous-requêtes
  - ▷ simplifier l'écriture des requêtes $\rightsquigarrow$ on décompose un problème en sous-problèmes "plus simples", comme avec des fonctions en programmation
  - ▷ optimiser l'exécution de la requête
    - **explosion combinatoire** $\rightsquigarrow$ on diminue le nombre de tables dans le `from` $\Rightarrow$ on limite le produit cartésien $\Rightarrow$ on limite la taille du "big tableau" intermédiaire $\Rightarrow$ moins de lignes à traiter par le `where`
    - **cache** $\rightsquigarrow$ avec un `in` l'interpréteur SQL va mettre en cache le résultat de la sous-requête $\Rightarrow$ évite de recalculer la sous-requête si les tables n'ont pas été modifiées entre temps
    - ***"cut"*** $\rightsquigarrow$ avec un `exists` l'interpréteur SQL peut décider du résultat dès que le `where` de la sous-requête conserve une ligne $\Rightarrow$ il peut interrompre la sous-requête

Langage SQL : `select` "avancé" ⇝ calculs

- Calculs possibles ⇝ tant dans la partie `select` que dans les prédicats (ex : `where`)
  - opérateurs arithmétiques `+`, `-`, `*`, `/`
  - fonctions statistiques applicables sur des groupes de tuples
    - fonctions numériques : `avg`, `sum`, `min`, `max`
    - nombre d'éléments : `count`

- Exemples
  - ▷ totaux des heures de cours, de TD et de TP pour un prof donné

```
select sum(nbHC), sum(nbHTD), sum(nbHTP)
from Cours
where nomProf = 'Munier'
```

Langage SQL : select "avancé" ⇝ calculs

- Exemples
  - nombre de prof différents faisant des cours

```
select count(distinct nomProf)
from Cours
where nbHC > 0
```

  - nombre d'étudiants ayant eu en Java une note supérieure à la moyenne du module

```
select count(*) from Suit
where numCours = 'C1'
  and note >= (select avg(note) from Suit
               where numCours = 'C1')
```

Langage SQL : `select` "avancé" $\rightsquigarrow$ partitions

- Il est possible de faire des calculs pour un ensemble de tuples vérifiant un même critère $\rightsquigarrow$ requêtes avec **partitionnement**

```
select a_1,...,a_p
from T_1,...,T_n
[where B_1]
   group by b_1,...,b_q
   [having B_2]
```

- Remarques
  - la clause `where` porte sur le `from`
  - la clause `having` porte sur le `group by`

Langage SQL : select "avancé" ⇝ partitions

- La clause `group by` permet de réaliser des agrégats
- Un agrégat est un partitionnement horizontal d'une table en sous-tables en fonction des valeurs d'un ou plusieurs attributs de partitionnement
- Exemple : affiche, pour chaque nom de cours, combien d'étudiants suivent ce cours

```
select nomCours, count(*) "nb etud"
from Cours,Suit
where Cours.numCours=Suit.numCours
group by nomCours
```

Langage SQL : select "avancé" ⇝ partitions

- Exemple : donne, grâce à un regroupement des tuples par fournisseur, pour chaque fournisseur
  - le nombre de produits livrés
  - la somme des quantités livrées (tous produits confondus)

```
select numFour, count(numProd), sum(qte) from Stock
group by numFour
```

- Idem, mais uniquement pour les fournisseurs ayant livré au moins 3 produits

NB : le `having` s'applique aux sous-tables du `group by`

```
select numFour, count(numProd), sum(qte) from Stock
group by numFour
  having count(*)>=3
```

Langage SQL : `select` "avancé" ⇝ `inner join`

- Quand on indique plusieurs tables dans le `from`, le comportement par défaut est de faire le produit cartésién

  calcul de **TOUTES** les combinaisons possibles

  ⇒ explosion combinatoire ☹

  ⇒ nécessité de filtrer les lignes "inutiles" dans le `where`

- SQL a introduit un nouvel opérateur ⇝ `inner join`

  ▷ idée ≡ quand on "croise" plusieurs tables, si elles ont des attributs en commun, seules les combinaisons où ces attributs communs ont la même valeur sont calculées

  ⇒ on limite les combinaisons au "strict nécessaire"

Langage SQL : select "avancé" ⇝ inner join

- La requête suivante ...

```
select distinct NomCours,NomProf
from Cours,Suit
where (Suit.NumCours=Cours.NumCours);
```

- ... peut ainsi s'écrire

```
select distinct NomCours,NomProf
from Cours
inner join Suit
   on (Suit.NumCours=Cours.NumCours);
```

Langage SQL : select "avancé" ⇝ inner join

- La requête suivante ...

```
select Nom,Prenom,NomCours,NomProf,Note
from Etudiant,Cours,Suit
where (Suit.NumEtud=Etudiant.NumEtud)
  and (Suit.NumCours=Cours.NumCours);
```

- ... peut ainsi s'écrire

```
select Nom,Prenom,NomCours,NomProf,Note
from Etudiant
inner join Suit on (Suit.NumEtud=Etudiant.NumEtud)
inner join Cours on (Cours.NumCours=Suit.NumCours);
```

Conclusion

- On vient de voir comment "améliorer" nos requêtes SQL
  - définition d'alias sur les tables du `from`
  - utilisation de sous-requêtes
    - `where` ⟨attr⟩ `in` (⟨sous-requête⟩)
    - `where exists` (⟨sous-requête⟩)
  - opérateurs de calcul ⇝ `avg`, `sum`, `min`, `max`, `count`, ...
  - filtres et calculs sur des groupes de lignes ⇝ `group by`... `having`...
  - opérateur `inner join` pour "contrôler" les combinaisons

Plan du cours

Exemple de requête SQL depuis Python

- Documentation (liens web)
  - ▷ `sqlite3` — DB-API 2.0 interface for SQLite databases
- Logique
  1. importer la librairie `sqlite3`
  2. créer un connecteur en lui indiquant les paramètres d'accès à la BdD
     $\rightsquigarrow$ ici le fichier SQLite
  3. créer un objet curseur qui servira à exécuter les requêtes
  4. sur ce curseur, invoquer la méthode `execute` avec la requête SQL à exécuter
     $\rightsquigarrow$ une "simple" chaîne de caractères
  5. récupérer le résultat (sous forme d'itérateur) et le traiter ligne par ligne
  6. fermer "proprement" le connecteur

Exemple de requête SQL depuis Python

exécuter une requête SQL depuis Python

```
import sqlite3 as sql

con = sql.connect("bdd_cours.sqlite")
cur = con.cursor()

cur.execute("select distinct NomCours,NomProf from Cours,Suit where (Suit.NumCours = Cours.NumCours)")
res = cur.fetchall()

for row in res:
  (cours,prof) = tuple(row)
  print("- "+cours+" -> "+prof)

print("fin.")

con.close()
```

Plan du cours

Pourquoi de tels formats ?

- Les bases de données et SQL c'est bien, mais...
  - cela nécessite un SGBD $\rightsquigarrow$ un serveur ("sauf" avec des outils tels que SQLite)
  - au niveau physique, les données sont stockées au format binaire $\rightsquigarrow$ format non lisible par un humain
  - transactions, droits d'accès, ... $\rightsquigarrow$ on n'a pas toujours besoin de tous ces outils

$\Rightarrow$ Besoin de solutions "plus simples" pour stocker efficacement des données

- technologies "plus légères" (RAM, cpu, réseau, ...)
- interopérabilité entre les différents outils
- format lisible par un humain
- utilisation "plus simple" depuis les langages de programmation

Exemples de formats

- CSV ≡ *Comma Separated Values*
  - ▷ « données séparées avec des virgules »
  - ▷ un classique pour échanger des tables au format texte (ex : tableurs)
- XML ≡ *eXtensible Markup Language*
  - ▷ « langage de balisage extensible »
  - ▷ syntaxe très riche, très extensible, très manipulable, très… ⇝ et bien **trop** en fait ☹
- JSON ≡ *JavaScript Object Notation*
  - ▷ fonctionne avec un système de paire clé/valeur
  - ▷ léger, expressif (tableaux, listes, objets,…) ⇝ adoptés par tous les langages

Format CSV

- Définition Wikipédia (➪)
  - *« Un fichier CSV est un fichier texte. Chaque ligne du texte correspond à une ligne du tableau et les virgules correspondent aux séparations entre les colonnes. Les portions de texte séparées par une virgule correspondent ainsi aux contenus des cellules du tableau. »*

Etudiant

NumEtud	Nom	Prenom	Adresse	DateNais	Sexe
E1	Cousy	Cécile	?	?	F
E2	Pourcel	Mathieu	?	?	M
E3	Burgy	Laurent	?	?	M

Cours

NumCours	NomCours	NomProf	NbHC	NbHTD	NbHTP
C1	Java	Munier	12	10,5	15
C2	Réseaux	Bascou	24	30	30
C3	TransNum	Baillot	12	18	24

Suit

NumEtud	NumCours	Note
E1	C1	18,5
E1	C2	15
E2	C1	9,5

table Cours au format CSV

```
NumCours,NomCours,NomProf,NbHC,NbHTD,NbHTP
C1,Java,Munier,12,"10,5",15
C2,Réseaux,Bascou,24,30,30
C3,TransNum,Baillot,12,18,24
```

Format CSV

- Avantages
  - format texte lisible & générable facilement
  - 1ère ligne ≡ noms des colonnes

- Inconvénients
  - que des chaînes de caractères ⇝ pas de types
  - que des tables ⇝ pas d'objets avec des attributs, pas d'arborescence, etc.

Format XML

- Définition Wikipédia (➪)
  - ▷ *« L'objectif initial de XML est de faciliter l'échange automatisé de contenus complexes (arbres, texte enrichi, etc.) entre systèmes d'informations hétérogènes (interopérabilité). Avec ses outils et langages associés, une application XML respecte généralement certains principes*
    - *la structure d'un document XML est définie et validable par un schéma*
    - *un document XML est entièrement transformable dans un autre document XML*

    Sa syntaxe est dite extensible car elle permet de définir différents langages avec pour chacun son vocabulaire et sa grammaire, comme XHTML, XSLT, RSS, SVG,... »

Format XML

- Avantages
  - format très riche, très expressif (ex : .docx ≈ .xml)
  - langage de transformation ⇝ XSLT
  - langage de requête ⇝ XPath
- Inconvénients
  - beaucoup trop verbeux ⇝ "illisible"
  - outils (schémas, espaces de noms, transformations, parcours,...) relativement complexes
  - APIs très énergivores (RAM, cpu)

```xml
<?xml version="1.0"?>
<questionnaire>
 <question>
  Qui était le premier
  empereur romain ?
 </question>
 <réponse>
  Auguste
 </réponse>
 <!-- Note : tu auras besoin
  de plus de questions.-->
</questionnaire>
```

Format JSON

- Origine $\rightsquigarrow$ dérivé de la syntaxe JavaScript (➪)
- Syntaxe
  - ▷ un objet est encadré par des accolades
  - ▷ les clés représentent les descripteurs et les valeurs sont les données
- Exemple :
  - ▷ `{cle_1 : val_1, cle_2 : val_2, cle_3 : val_3}`
  - ▷
```
{
cle_1 : val_1,
cle_2 : val_2,
cle_3 : val_3
}
```

Format JSON

- Exemple de déclaration JSON

format JSON

```
{
  "fruits": [
    { "kiwis": 3,
      "mangues": 4,
      "pommes": null
    },
    { "panier": true },
  ],
  "legumes":
    { "patates": "amandine",
      "figues": "de barbarie",
      "poireaux": false
    }
}
```

équivalent en XML

```
<?xml version="1.0" ?>
<root>
  <fruits>
    <item>
      <kiwis>3</kiwis>
      <mangues>4</mangues>
      <pommes></pommes>
    </item>
    <item>
      <panier>true</panier>
    </item>
  </fruits>
  <legumes>
    <patates>amandine</patates>
    <figues>de barbarie</figues>
    <poireaux>false</poireaux>
  </legumes>
</root>
```

Format JSON

- Avantages
  - format texte lisible & générable facilement
  - syntaxe très expressive : types, objets, listes, tableaux,...
  - parcours facile avec les clés
  - maintenant supporté par tous les langages de programmation
  - outils performants
  - 1 structure de données ≡ 1 chaîne de caractères
    - exemple d'application : *payload* d'un message MQTT en IoT

- Inconvénients
  - ???

Format JSON ⇝ utilisation en Python

- Documentation (liens web)
  - `json` — JSON encoder and decoder
  - Read, Write and Parse JSON using Python
  - `json.dumps()` in Python
  - `json.loads()` in Python
  - Get JSON From URL in Python

- Ce que l'on va voir
  - convertir une structure de données Python en JSON
  - importer des données JSON en Python
  - télécharger des données JSON depuis une URL ⇝ Open Data

Format JSON ⇝ Python → JSON

exemple 1 : exporter des données Python vers JSON

```python
# Programme exportant des données Python vers JSON
import json

# Création de données en Python
cours = [('Java','Munier',12,10.5,15),('Reseaux','Bascou',24,30,30),('TransNum','Baillot',12,18,24)]
etudiants = [('Cousy','Cecile'),('Pourcel','Mathieu'),('Burgy','Laurent')]
data = (cours,etudiants)
print('Données en Python:',data)

# Export des données en JSON
json_string = json.dumps(data)
print('Données en JSON:',json_string)

# Export des données en JSON avec affichage "pretty"
json_pretty_string = json.dumps(data, indent=4)
print('Données en JSON (pretty):\n',json_pretty_string)
```

Format JSON ⇝ Python → JSON

exemple 1 : affichage

```
Données en Python: ([('Java', 'Munier', 12, 10.5, 15), ('Reseaux', 'Bascou', 24, 30, 30), ('TransNum', 'Baillot', 12, 18, 24)], [('Cou
Données en JSON: [[["Java", "Munier", 12, 10.5, 15], ["Reseaux", "Bascou", 24, 30, 30], ["TransNum", "Baillot", 12, 18, 24]], [["Cousy
Données en JSON (pretty):
[
    [
        [
            "Java",
            "Munier",
            12,
            10.5,
            15
        ],
        [
            "Reseaux",
            "Bascou",
            24,
            30,
            30
        ],
        [
            "TransNum",
            "Baillot",
            12,
```

Format JSON ⇝ Python → JSON

exemple 2 : dictionnaire Python vers JSON

```python
# Python program to convert Python to JSON
import json

# Data to be written
dictionary = {
        "id": "007",
        "firstname": "Manuel",
        "lastname": "Munier",
        "department": "RT"
}

# Serializing JSON
json_object = json.dumps(dictionary, indent = 4)
print(json_object)
```

exemple 2 : affichage

```
{
    "id": "007",
    "firstname": "Manuel",
    "lastname": "Munier",
    "department": "RT"
}
```

Source : https://www.geeksforgeeks.org/read-write-and-parse-json-using-python/

Format JSON ⇝ JSON → Python (à partir d'une chaîne de caractères)

exemple 3 : string JSON vers Python

```python
# Python program to convert JSON to Python
import json

# JSON string
employee = '{"id":"007", "name": "Munier", "department":"RT"}'

# Convert string to Python dict
employee_dict = json.loads(employee)
print(employee_dict)

print(employee_dict['name'])
```

exemple 3 : affichage

```
{'id': '007', 'name': 'Munier', 'department': 'RT'}
Munier
```

Source : https://www.geeksforgeeks.org/read-write-and-parse-json-using-python/

Format JSON ⇝ JSON → Python (à partir d'un fichier)

exemple 4 : fichier JSON vers Python

```python
# Python program to read JSON file
import json

# Opening JSON file
f = open('data.json',)
# returns JSON object as a dictionary
data = json.load(f)
# Iterating through the JSON list
for i in data['emp_details']:
    print(i)
# Closing file
f.close()
```

exemple 4 : affichage

```
{'emp_name': 'Munier', 'email': 'munier@univ-pau.fr', 'job': 'chercheur'}
{'emp_name': 'Lespine', 'email': 'lespine@foo.org', 'job': 'enseignant'}
{'emp_name': 'Bats', 'email': 'bats@foo.org', 'job': 'technicien'}
```

Source : https://www.geeksforgeeks.org/read-write-and-parse-json-using-python/

exemple 4 : fichier `data.json`

```
{
    "emp_details": [
        {
            "emp_name": "Munier",
            "email": "munier@univ-pau.fr",
            "job": "chercheur"
        },
        {
            "emp_name": "Lespine",
            "email": "lespine@foo.org",
            "job": "enseignant"
        },
        {
            "emp_name": "Bats",
            "email": "bats@foo.org",
            "job": "technicien"
        }
    ]
}
```

Open Data

- Définition Wikipédia (➪)
  - *Les **données ouvertes** (en anglais : **open data**) sont des données numériques dont l'accès et l'usage sont laissés libres aux usagers, qui peuvent être d'origine privée mais surtout publique, produites notamment par une collectivité ou un établissement public. Elles sont diffusées de manière structurée selon une méthode et une licence ouverte garantissant leur libre accès et leur réutilisation par tous, sans restriction technique, juridique ou financière.*
  - *L'accès aux données vise d'une part à permettre aux citoyens de mieux contrôler l'administration, d'autre part d'exploiter ces données, ce qui implique que ce droit d'accès s'accompagne d'un droit à la réutilisation des données.*
  - *Ces droits d'accès et de réutilisation s'inscrivent dans la pensée qui considère l'information publique comme un **bien commun** (tel que défini par **Elinor Ostrom**) dont la diffusion est d'intérêt public et général.*

Open Data

- En pratique
  - données partagées « librement » sur Internet (accès via une URL)
  - données structurées $\rightsquigarrow$ export en CSV, JSON, XML,...
  - le site donne généralement la description de la structure des données (champs)
- Exemple
  - "Départements et leurs régions" sur `data.gouv.fr`
  - https://www.data.gouv.fr/fr/datasets/departements-et-leurs-regions/
  - jeu de données au format JSON → `departements-region.json`
    https://www.data.gouv.fr/fr/datasets/r/7b4bc207-4e66-49d2-b1a5-26653e369b66

Open Data

115 téléchargements

departements-region.json

json (9.8Ko)

Nom et désignation en clair

1) **num_dep** correspond au numéro du département
2) **dep_name** correspond au nom du département
3) **region_name** correspond à de la région de ce département

URL	https://static.data.gouv.fr/resources/departements-et-leurs-regions/20190815-175403/departements-region.json
URL stable	https://www.data.gouv.fr/fr/datasets/r/7b4bc207-4e66-49d2-b1a5-26653e369b66
Type	Fichier principal
Type MIME	application/json
sha1	14ec056d11c1889f26f294c651cc455aee70a9b7
Créée le	15 août 2019
Modifiée le	15 août 2019
Publiée le	15 août 2019

Open Data ⇝ JSON → Python (à partir d'une URL)

exemple 5 : URL JSON vers Python

```python
# Python program to import JSON open data to Python
import json, requests

# Fetch data from URL
url = requests.get("https://www.data.gouv.fr/fr/datasets/r/7b4bc207-4e66-49d2-b1a5-26653e369b66")
text = url.text

# Process JSON data
data = json.loads(text)
for dep in data:
        print(dep)
```

exemple 5 : affichage

```
{'num_dep': '01', 'dep_name': 'Ain', 'region_name': 'Auvergne-Rhône-Alpes'}
{'num_dep': '02', 'dep_name': 'Aisne', 'region_name': 'Hauts-de-France'}
{'num_dep': '03', 'dep_name': 'Allier', 'region_name': 'Auvergne-Rhône-Alpes'}
{'num_dep': '04', 'dep_name': 'Alpes-de-Haute-Provence', 'region_name': "Provence-Alpes-Côte d'Azur"}
{'num_dep': '05', 'dep_name': 'Hautes-Alpes', 'region_name': "Provence-Alpes-Côte d'Azur"}
```

Source : https://www.delftstack.com/howto/python/python-get-json-from-url/

Open Data : Quel intérêt ?

- Pourquoi télécharger la liste des départements en Open Data ?
  - ces données sont « statiques », i.e. elles ne changent pas souvent
  - on pourrait très bien les télécharger une fois pour toutes et les stocker localement
  - ce sont des **données froides**
- Permet néanmoins d'accéder à une multitude de jeux de données
- C'est une obligation réglementaire pour les données publiques (cf. les administrations)

Open Data : LRN

- LRN, 07/10/2016 $\rightsquigarrow$ directive 2019/1024, 20/06/2019
  - Le premier volet de la loi pour une République numérique (dite "loi Lemaire") vise à favoriser la "circulation des données et du savoir" à travers l'ouverture des données publiques et d'intérêt général, la création d'un service public de la donnée et le libre accès aux écrits de la recherche publique.
  - Avec la LRN, l'ouverture des données publiques ou *open data* franchit un nouveau cap. Elle devient la règle et non plus l'exception. Désormais, les administrations au sens large doivent publier en ligne dans un standard ouvert leurs principaux documents.

Open Data : données « dynamiques »

- L'Open Data devient beaucoup plus intéressant avec des données plus « dynamiques », qui changent fréquemment et dont on veut les valeurs les plus récentes
  - exemples : données météo, trafic routier, consommation d'énergie, etc.
  - on parle alors de **données chaudes**
- De nouveaux horizons...
  - IoT, industrie 4.0, edge computing,...
  - smart-cities, smart-buildings, smart-grids,...
  - Big Data, Deep Learning, Machine Learning,...

Plan du cours

CNIL

- La France, pionnière en matière de données personnelles, a très vite réalisé les risques de l'informatique pour les libertés de la personne humaine et a réagi en se dotant depuis 1978 d'une législation traitant de la protection de ces données face aux dangers d'une informatique grandissante : la loi du 6 janvier 1978 dite "informatique et libertés".
- La Commission Nationale de l'Informatique et des Libertés (CNIL) est une autorité administrative indépendante née de l'adoption de la loi précitée.
- La mission générale de la CNIL est, face aux dangers de l'informatique, de protéger la vie privée et les libertés individuelles ou publiques.

CNIL

- La CNIL est une autorité administrative indépendante qui fonctionne avec une dotation du budget de l'État.
- Elle informe et conseille les personnes sur leurs droits : elle reçoit les réclamations, pétitions, plaintes relatives aux traitements de données personnelles et répond par des avis, délibérations ou recommandations.
- Elle régule et recense les fichiers de données personnelles : elle autorise ou non la création des fichiers.
- Elle peut opérer des contrôles dans les entreprises, des enquêtes, des auditions de personnes.
- La CNIL n'est pas un juge. Il faut saisir le juge pour obtenir des dommages-intérêts au titre de la responsabilité civile en cas de préjudice.

Notion de données sensibles

- **Attention** (→ responsables de traitements) : En pratique une confusion est parfois opérée entre :
  - ▷ *"les données qui sont sensibles pour une organisation"* (ex : des données financières pour une entreprise)
  - ▷ le régime juridique des *"données sensibles, au sens de la loi de 1978"*
- Des données sensibles pour une entreprise (l'évolution de son chiffre d'affaire, ses processus d'acquisition de clientèle,...) ou une administration (organisation interne, dossiers politiquement et médiatiquement sensibles,...) ne sont pas nécessairement des données sensibles au sens de la loi de 1978.

Notion de données sensibles

- Cette distinction est importante car seul le traitement de *données sensibles au sens de la loi informatique et libertés* devra répondre au régime juridique décrit ci-après.

- Peu importe qu'une organisation traite des *données qui lui sont sensibles*, dès lors que ces données ne sont pas des données personnelles.

Droits et principes fondamentaux

Loi "informatique et libertés"

La loi de 1978, dite loi "informatique et libertés", instaure de nombreux droits et principes fondamentaux relatifs **"à la protection des personnes physiques à l'égard des traitements de données à caractère personnel"**.

- Principe de **finalité**
  - Les données personnelles ne peuvent être collectées, traitées, conservées ou transmises à des tiers qu'en vue de réaliser des finalités déterminées, légitimes et compatibles entre elles.

Droits et principes fondamentaux

- Principe de **loyauté** et de **transparence**
  - La collecte, le traitement, la conservation des données personnelles et leur transmission éventuelle à des tiers doivent s'effectuer de manière loyale.
  - Cela suppose que les données ne soient pas collectées et traitées à l'insu de la personne concernée et que les personnes soient informées de l'identité et du lieu d'établissement de la personne qui traite ces données, des finalités poursuivies, du caractère obligatoire ou facultatif du traitement des données, des destinataires des informations, ainsi que toute information nécessaire à l'exercice de leurs droits.

Droits et principes fondamentaux

- Principe de la **pertinence** et de l'**exactitude** des données
  - ▷ Les données personnelles faisant l'objet d'un traitement doivent être pertinentes au regard des finalités poursuivies. Elles doivent être exactes et mises à jour.

- Principe du **consentement** pour les traitements de données sensibles
  - ▷ Lorsque des traitements portent sur des données sensibles (religion, opinion politique ou philosophique, appartenance syndicale, origine raciale et ethnique, santé et vie sexuelle), celles-ci ne peuvent être collectées qu'avec le consentement des personnes.

Droits et principes fondamentaux

- Principe d'**accès**, de **rectification** et d'**opposition**
  - Les personnes doivent se voir reconnaître les droits d'accéder, sans subir de coût dissuasif, à toute donnée les concernant, de corriger les données incomplètes ou inexactes et de s'opposer sans avoir à se justifier à l'exploitation de leurs données à des fins commerciales.

- Principe de **sécurité**
  - Le code pénal incrimine le traitement, sans que soient prises les mesures de précaution, et prévoit des sanctions à l'encontre de l'administrateur ne protégeant pas assez efficacement son système (délit de manquement à la sécurité).

Droits et principes fondamentaux

- Principe du **droit à l'oubli**
  - ▷ Le code pénal incrimine le fait, sans l'accord de la CNIL, de conserver une information sous la forme nominative au-delà de la durée prévue à la demande d'avis ou à la déclaration préalable.

- Principe de **protection de la considération et de l'intimité**
  - ▷ Le fait de porter à la connaissance d'un tiers des images portant atteinte à la considération de l'intéressé ou à l'intimité de sa vie privée est condamnable.
  - ▷ Cette divulgation est sanctionnée encore plus sévèrement si elle a été faite par imprudence ou négligence (délit d'atteinte à la considération ou à l'intimité).

Résumé droits & obligations

- Les droits des personnes
  - **Droit à l'information**
    Toute personne a le droit de savoir si elle est fichée et dans quels fichiers elle est recensée.
  - **Droit d'accès**
    [. . .] a le droit d'interroger le responsable d'un fichier pour savoir s'il détient des informations sur elle et d'en obtenir la communication.
  - **Droit de rectification**
    [. . .] a le droit de contrôler l'exactitude des données et de les faire rectifier.
  - **Droit d'opposition**
    [. . .] peut s'opposer pour des motifs légitimes à figurer dans un fichier ou de voir communiquer des informations sur elle à des tiers. Les personnes peuvent saisir la CNIL en cas de difficultés dans l'exercice de leurs droits.

Résumé droits & obligations

- Les obligations des responsables du traitement
  - ▷ **Obligation d'information préalable** des personnes concernées dont on doit obtenir le consentement exprès.
  - ▷ **Obligation d'assurer la sécurité et la confidentialité** des données collectées et traitées.
  - ▷ **Obligation d'une collecte et d'un traitement** ayant une finalité précise et effectués de façon licite et loyale.
  - ▷ **Obligation de déclaration préalable à la CNIL** des traitements informatiques de données personnelles.

Données à caractère personnel

- Points clés
  - Les données sont des données à caractère personnel dès lors qu'elles portent sur une **personne identifiée ou identifiable**, la personne concernée.
  - Une personne est identifiable si des informations complémentaires peuvent être obtenues sans effort déraisonné, permettant l'identification de la personne concernée.
  - L'authentification s'entend du fait de démontrer qu'une certaine personne possède une certaine identité et/ou est autorisée à exercer certaines activités.
  - NB : **identification ≠ authentification**

Données à caractère personnel

- Points clés
  - Il existe des catégories particulières de données, appelées "données sensibles", énumérées dans la directive relative à la protection des données, qui requièrent une protection accrue et, par conséquent, sont soumises à un régime juridique spécial.
  - Les données sont anonymisées si elles ne contiennent plus d'identifiants ; elles sont pseudonymisées si les identifiants sont cryptés.
  - Contrairement aux données anonymisées, les données pseudonymisées sont des données à caractère personnel.

Caractère identifiable d'une personne

- Dans le droit de l'UE, une information contient des données sur une personne si :
  - ▷ une personne est identifiée dans cette information

  ou ▷ si une personne, bien que non identifiée, est décrite dans cette information d'une manière permettant de découvrir qui est la personne concernée en menant d'autres recherches

- Les deux types d'informations sont protégés de la même manière par le droit européen en matière de protection des données.
  - noms non uniques ⇒ date et lieu de naissance, numéros de citoyens,...
  - ère du numérique ⇒ données biométriques (empreintes digitales, photos numériques, aspects rétiniens) pour l'identification des personnes

Authentification

- L'authentification est la procédure par laquelle une personne peut prouver qu'elle possède une certaine identité et/ou est autorisée à faire certaines choses.
  - Par la comparaison de données biométriques (une photo ou les empreintes digitales d'un passeport) avec les données de la personne qui se présente à un contrôle d'immigration.
  - En demandant des informations que seule la personne possédant une certaine identité ou autorisation devrait connaître (numéro d'identification personnel (PIN) ou un mot de passe).
  - En demandant la présentation d'un certain objet qui devrait exclusivement se trouver en la possession de la personne ayant une certaine identité ou autorisation (une carte magnétique spéciale ou la clé d'un coffre en banque).

Authentification

- L'authentification est la procédure par laquelle une personne peut prouver qu'elle possède une certaine identité et/ou est autorisée à faire certaines choses.
  - ▷ Outre les mots de passe ou cartes magnétiques, parfois associés à des codes PIN, les signatures électroniques sont un outil particulièrement utile pour identifier ou authentifier une personne dans des communications électroniques.
- NB : L'authentification ne nécessite pas de stocker les données personnelles (ex : empreinte digitale) sur le serveur, contrairement à l'identification.

Catégories particulières de données à caractère personnel

- Il existe des catégories particulières de données qui, par leur nature, peuvent faire courir un risque aux personnes concernées quand elles font l'objet d'un traitement et requièrent donc une protection accrue :
  - ▷ données à caractère personnel révélant l'origine raciale ou ethnique
  - ▷ données à caractère personnel révélant les opinions politiques, convictions religieuses ou autres convictions
  - ▷ données à caractère personnel relatives à la santé ou à la vie sexuelle

Données anonymisées et pseudonymisées

- Principe de la conservation des données pendant une durée limitée :
  - les données doivent être conservées *"sous une forme permettant l'identification des personnes concernées pendant une durée n'excédant pas celle nécessaire à la réalisation des finalités pour lesquelles elles sont collectées ou pour lesquelles elles sont traitées ultérieurement"*

$\Rightarrow$ Il pourrait être nécessaire d'anonymiser des données si un responsable du traitement souhaite les conserver alors qu'elles ne sont plus d'actualité et qu'elles ne servent plus leur finalité initiale.

Données anonymisées et pseudonymisées

- **Données anonymisées**
  - Des données sont anonymisées si tous les éléments identifiants ont été supprimés d'un ensemble de données à caractère personnel.
  - Les informations ne doivent plus contenir aucun élément qui soit susceptible, au moyen d'un effort raisonnable, de servir à réidentifier la ou les personnes concernées.
  - Lorsque des données ont été correctement anonymisées, elles ne sont plus des données à caractère personnel.

Données anonymisées et pseudonymisées

- **Données pseudonymisées**
  - Les informations personnelles contiennent des identifiants, tels que le nom, la date de naissance, le sexe ou l'adresse.
  - Lorsque des informations personnelles sont pseudonymisées, les identifiants sont remplacés par un pseudonyme.
  - La pseudonymisation est notamment obtenue par cryptage des identifiants figurant dans les données à caractère personnel.
  - Il ne doit pas être possible de relier facilement les données et les identifiants. Pour quiconque ne possède pas la clé de décryptage, les données pseudonymisées peuvent être difficilement identifiables.
  - Le lien avec l'identité demeure sous la forme du pseudonyme associé à la clé de décryptage. Pour toute personne habilitée à utiliser la clé de décryptage, une nouvelle identification est possible aisément ⇒ **données personnelles**

RGPD

- Règlement Général pour la Protection des Données (**RGPD**) le 25 mai 2018
- Rien de révolutionnaire !
  - en France nous avions déjà la loi "informatique et libertés"
  - modifiée par la loi du 6 août 2004 afin de transposer en droit français les dispositions de la directive 95/46/CE
- Dans les grandes lignes, le RGPD cherche à renforcer la responsabilité des sociétés amenées à gérer des informations personnelles. Ses différentes dispositions cherchent donc à assurer la protection de ces données, mais aussi leur traçabilité, et le suivi précis des traitements qui en seront faits.

RGPD

- Le poste de **DPO** (Data Protection Officer) s'inscrit dans le prolongement de ce que la CNIL avait déjà initié avec son Correspondant Informatique et Libertés (**CIL**), avec un niveau de responsabilité similaire mais des prérogatives étendues.

- Ce pilote en interne est censé cartographier l'ensemble des traitements de données personnelles réalisés par l'entreprise pour identifier les carences et proposer une optimisation des processus.

  NB : La CNIL recommande également la création d'une documentation permettant de justifier des mesures entreprises en cas d'enquête de conformité.

RGPD & PIA

- L'Étude d'Impact sur la Vie Privée (**EIVP**) était la traduction utilisée par la CNIL pour le Privacy Impact Assessment (**PIA**).
- L'application du nouveau règlement européen RGPD impose (en partie) la réalisation d'une analyse d'impact relative à la protection des données (ou **DPIA** pour Data Protection Impact Assessment).
- Cette démarche, repose sur une analyse de risques sécurité orientée uniquement sur les risques visant les données personnelles et leurs impacts sur les droits et libertés des personnes concernées par ces données.

RGPD & PIA

- Le DPIA obligatoire uniquement pour les traitements présentant *"un risque élevé pour les droits et libertés des personnes physiques"*
  - → les traitements à grande échelle
  - → la surveillance systématique à grande échelle d'une zone accessible au public (notamment la vidéosurveillance)
  - → les décisions automatiques produisant des effets juridiques (pour des offres de prestations, ou le choix de contractualisation)
  - → le traitement de données sensibles (données de santé, opinions politiques, orientation sexuelle)
  - → l'évaluation ou la notation basée sur des données personnelles, y compris le profilage et la prédiction
  - → le traitement de données biométriques, de données relatives à des condamnations pénales et à des infractions

Partage de données

- Avec l'émergence des environnements connectés, IoT, smart-*, IA & ML le législateur propose de nouveaux règlements :
  - **RGPD** → Règlement Général sur la Protection des Données (25/05/2018)
    https://fr.wikipedia.org/wiki/Règlement_général_sur_la_protection_des_données
  - **LRN** → Loi pour une République Numérique (07/10/2016)
    https://fr.wikipedia.org/wiki/Loi_pour_une_République_numérique
  - **Une stratégie européenne pour les données** (19/02/2020)
    https://eur-lex.europa.eu/legal-content/FR/TXT/?uri=CELEX%3A52020DC0066
  - **DGA** → Data Governance Act (24/09/2023)
    https://eur-lex.europa.eu/legal-content/FR/ALL/?uri=CELEX%3A52020PC0767
  - **Data Act** (11/01/2024)
    https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=COM%3A2022%3A68%3AFIN

Partage de données

- Voire également de nouveaux concepts :
  - **Open Data**
  - **Communs numériques**

- En s'adaptant aux nouveaux usages :
  - **AI Act** → Artificial Intelligence Act (08/2024)
  - **DSA** → Digital Services Act (02/2024)
    fixe un ensemble de règles pour responsabiliser les plateformes numériques et lutter contre la diffusion de contenus illicites ou préjudiciables ou de produits illégaux : attaques racistes, images pédopornographiques, désinformation, vente de drogues ou de contrefaçons... ⇝ succède à la directive dite e-commerce du 8 juin 2000, devenue dépassée