Bases de données

Informatique pour tous

Gérer beaucoup de données

Problème: comment gérer de façon efficace un grand nombre de données?

Gérer beaucoup de données

Problème: comment gérer de façon efficace un grand nombre de données?

Exemples:

1. Les astres connus, avec leur taille, poids...
2. Les espèces connues, avec leur taxonomie, famille...
3. Les utilisateurs d'un site web, avec leur mot de passe, préférences...
4. Les élèves d'un lycée, avec leurs notes, options...

Quelques solutions

On peut stocker ces données dans:

1. Un tableur: trop limité.
2. Une liste Python: trop désorganisé.
3. Une **base de donnée**.

Base de donnée

Base de donnée

Une **base de donnée** est un ensemble de **tables**. On peut y extraire des informations à l'aide de **requêtes**.

Base de donnée

Base de donnée

Une **base de donnée** est un ensemble de **tables**. On peut y extraire des informations à l'aide de **requêtes**.

Table

Une **table** est un tableau à 2 dimensions dont les colonnes sont les **attributs** et les lignes les **enregistrements**.

Base de donnée

Base de donnée

Une **base de donnée** est un ensemble de **tables**. On peut y extraire des informations à l'aide de **requêtes**.

Table

Une **table** est un tableau à 2 dimensions dont les colonnes sont les **attributs** et les lignes les **enregistrements**.

Exemple: une base de donnée `astre` contient deux tables `planete` et `etoile`.
La table `planete` possède des attributs `nom`, `rayon`, `poids`...
Chaque enregistrement de `planete` correspond aux informations sur une planète.

Base de donnée astre

nom	rayon (km)	poids (kg)	etoile
'Terre'	6400	$6x10^{24}$	'Soleil'
'Jupiter'	70000	$2x10^{27}$	'Soleil'
'Proxima b'	?	?	'Proxima Centauri'
...	...	...	...

Table planete

nom	type	duree_vie (année)	galaxie
'Soleil'	'Naine jaune'	10^{10}	'Voie lactée'
'Proxima Centauri'	'Naine rouge'	?	'Voie lactée'
...	...	...	...

Table etoile

Domaine

Chaque attribut a un **domaine**: l'ensemble des valeurs que peut prendre cet attribut.

Domaine

Chaque attribut a un **domaine**: l'ensemble des valeurs que peut prendre cet attribut.

Dans la table `planete`:

1. `nom` est un attribut ayant pour domaine l'ensemble des chaînes de caractères.
2. `rayon` est un attribut ayant pour domaine $\mathbb{N}$.
3. ...

Relation

Relation

Une **relation** R sur des ensembles E_1, ..., E_n est un sous-ensemble du produit cartésien $E_1 \times ... \times E_n$:

$$R \subseteq E_1 \times ... \times E_n$$

Relation

Relation

Une **relation** R sur des ensembles E_1, ..., E_n est un sous-ensemble du produit cartésien $E_1 \times ... \times E_n$:

$$R \subseteq E_1 \times ... \times E_n$$

Par exemple, on peut voir la divisibilité comme une relation:
$R = \{(a, b),\ a \mid b\} \subseteq \mathbb{N} \times \mathbb{N}$

Relation

Une table peut être vue comme une **relation** au sens mathématique.

Si une table a pour domaines d'attributs D_1, ..., D_n alors l'ensemble R des enregistrements de la table vérifie bien:

$$R \subseteq D_1 \times ... \times D_n$$

On parle parfois de bases de données **relationnelles**.

Clé

Clé

Une **clé** d'une table est un ensemble minimal d'attributs permettant d'identifier de façon unique chaque enregistrement.

Clé

Clé

Une **clé** d'une table est un ensemble minimal d'attributs permettant d'identifier de façon unique chaque enregistrement.
Parmi les clés possibles on en choisit une qu'on nomme **clé primaire**.

Clé

Clé

Une **clé** d'une table est un ensemble minimal d'attributs permettant d'identifier de façon unique chaque enregistrement.
Parmi les clés possibles on en choisit une qu'on nomme **clé primaire**.

nom	type	duree_vie (année)	galaxie
'Soleil'	'Naine jaune'	10^{10}	'Voie lactée'
'Proxima Centauri'	'Naine rouge'	?	'Voie lactée'
'Kepler-22'	'Naine jaune'	?	'Voie lactée'

Table etoile

Clés possibles?

Clé

Clé

Une **clé** d'une table est un ensemble minimal d'attributs permettant d'identifier de façon unique chaque enregistrement.
Parmi les clés possibles on en choisit une qu'on nomme **clé primaire**.

nom	type	duree_vie (année)	galaxie
'Soleil'	'Naine jaune'	10^{10}	'Voie lactée'
'Proxima Centauri'	'Naine rouge'	?	'Voie lactée'
'Kepler-22'	'Naine jaune'	?	'Voie lactée'

Table etoile

Clés possibles? **nom**

Clé

nom	pays	latitude	longitude
'Hanoï'	'Viêt Nam'	21°	104°
'Valence'	'France'	45°	5°
'Valence'	'Espagne'	39°	0°
'Quito'	'Equateur'	0°	-78°
'Singapour'	'Singapour'	0°	104°
'Valence'	'France'	45°	1°

Table ville

Clés possibles?

Clé

nom	pays	latitude	longitude
'Hanoï'	'Viêt Nam'	21°	104°
'Valence'	'France'	45°	5°
'Valence'	'Espagne'	39°	0°
'Quito'	'Equateur'	0°	-78°
'Singapour'	'Singapour'	0°	104°
'Valence'	'France'	45°	1°

Table ville

Clés possibles?

1. latitude, longitude
2. nom, longitude
3. pays, longitude

Schéma

On peut résumer la structure d'une table par son schéma:

Schéma

Le schéma d'une table est la donnée de ses attributs, des domaines des attributs et de l'éventuelle clé primaire (soulignée), sous la forme:

table (attribut_1**:** type_1, ..., attribut_n**:** type_n)

Schéma

On peut résumer la structure d'une table par son schéma:

Schéma

Le schéma d'une table est la donnée de ses attributs, des domaines des attributs et de l'éventuelle clé primaire (soulignée), sous la forme:

table (attribut_1**:** type_1, ..., attribut_n**:** type_n)

Par exemple, le schéma de la table `ville` avec comme clé primaire (latitude, longitude) est:

ville (nom: chaîne de caractères, pays: chaîne de caractères,
<u>latitude: entier</u>, <u>longitude: entier</u>)

Langage de requêtes

On accède à des informations d'une base de donnée avec un **langage de requêtes**.

Contrairement à un langage de programmation:

1. On ne va pas utiliser de variable, boucle...
2. On se contente de demander ce que l'on veut obtenir, mais il n'y a pas besoin de dire **comment** l'obtenir: la machine se débrouille.

Langage de requêtes / de programmation

Pour trouver la somme des masses des planètes du système solaire:

1. Langage de programmation:

```
Somme = 0
Pour toute planete p:
    Si p tourne autour du Soleil:
        Augmenter Somme du poids de p
```

2. Langage de requête:

```
Obtenir la somme des poids des planetes
qui tournent autour du Soleil
```

SQL

Le langage de requêtes le plus utilisé est **SQL** (**Structured Query Language**).

Il en existe plusieurs implémentations qui varient légèrement:

1. **MySQL**: **open source, gratuit, utilisé dans ce cours**.
2. Oracle Database: propriétaire, payant (40000 la licence...).
3. PostgreSQL: open source, gratuit.

SQL

Quelques règles en SQL:

1. Chaque requête doit être terminée par un point-virgule ;
2. Pas d'indentation obligatoire comme en Python, mais il est conseillé de bien présenter son code
3. Les commandes peuvent être écrites en majuscules ou minuscules
4. Il est conseillé d'écrire les commandes SQL en majuscules et de donner des noms de tables et colonnes en minuscules

Création d'une base de donnée

Pour créer une base de donnée: `CREATE DATABASE nom_base;`
Pour utiliser une base de donnée: `USE nom_base;`

```
CREATE DATABASE cpge;
USE cpge;
```

Création d'une table

Pour créer une table:

```
CREATE TABLE nom_table (
    attribut_1 type_1,
    attribut_2 type_2,
    ...
    attribut_n type_n
);
```

Création d'une table

Pour créer une table:

```
CREATE TABLE nom_table (
    attribut_1 type_1,
    attribut_2 type_2,
    ...
    attribut_n type_n
);
```

On peut ensuite ajouter des enregistrements:

```
INSERT INTO nom_table (attribut_1, ..., attribut_n)
VALUES (valeur_1, ..., valeur_n);
```

Création d'une table

Pour créer une table:

```
CREATE TABLE nom_table (
    attribut_1 type_1,
    attribut_2 type_2,
    ...
    attribut_n type_n
);
```

On peut ensuite ajouter des enregistrements:

```
INSERT INTO nom_table (attribut_1, ..., attribut_n)
VALUES (valeur_1, ..., valeur_n);
```

Il est possible de donner seulement certaines valeurs, les autres prenant alors une valeur par défaut (par exemple `NULL`).

Types en SQL

Les attributs peuvent être de type:

- `INT`: entier
- `CHAR(k)`: chaîne d'au plus k caractères
  Une chaîne de caractère doit être entourée de guillemets ("exemple") ou apostrophes ('exemple')
- `FLOAT`: nombre à virgule
- `BOOLEAN`: booléen (en fait soit 0 soit 1)

Création d'une table

Exemple: je veux créer une table de mes élèves avec leur nom, prénom, classe, option, école intégrée.

Création d'une table

Exemple: je veux créer une table de mes élèves avec leur nom, prénom, classe, option, école intégrée.

Il n'y a pas de clé possible! Dans ce cas, on peut créer un attribut qui fera office de clé primaire.
On peut dire quelle est la clé primaire en écrivant:
`PRIMARY KEY (clé)` dans la création de table.

Création d'une table

```sql
CREATE TABLE eleve (
    id INT AUTO_INCREMENT,
    PRIMARY KEY (id),
    nom CHAR(20),
    prenom CHAR(20),
    annee_entree INT,
    option_info BOOLEAN,
    classe_sup CHAR(5),
    classe_spe CHAR(5),
    classe_spe2 CHAR(5),
    ecole CHAR(20)
);
```

Création d'une table

```sql
INSERT INTO eleve (nom, prenom, classe_sup, annee_entree, option_info)
VALUES ('Turing', 'Alan', 'MPSI2', 2015, TRUE),
       ('Gödel', 'Kurt', 'MPSI1', 2015, TRUE),
       (NULL, 'Euclide', 'MPSI2', 2015, FALSE),
       ('Newton', 'Isaac', 'PCSI2', 2015, FALSE),
       ('Curie', 'Marie', 'PCSI1', 2015, FALSE);
```

SELECT

Pour afficher des colonnes d'une table, on utilise:

```
SELECT colonne_1, ..., colonne_n FROM table;
```

SELECT

Par exemple, pour obtenir seulement les noms et prénoms des élèves:

```
SELECT nom, prenom FROM eleve;
```

SELECT

Par exemple, pour obtenir seulement les noms et prénoms des élèves:

```
SELECT nom, prenom FROM eleve;
```

On obtient:

nom	prenom
Turing	Alan
Gödel	Kurt
NULL	Euclide
Newton	Isaac
Curie	Marie

SELECT

Pour afficher la table entière, on peut utiliser * plutôt que donner le nom de chaque colonne:

```
SELECT * FROM eleve;
```

SELECT

Pour afficher la table entière, on peut utiliser * plutôt que donner le nom de chaque colonne:

```
SELECT * FROM eleve;
```

On obtient:

id	nom	prenom	annee_entree	option_info	classe_sup	classe_spe	classe_spe2	ecole
1	Turing	Alan	2015	1	MPSI2	NULL	NULL	NULL
2	Gödel	Kurt	2015	1	MPSI1	NULL	NULL	NULL
3	NULL	Euclide	2015	0	MPSI2	NULL	NULL	NULL
4	Newton	Isaac	2015	0	PCSI2	NULL	NULL	NULL
5	Curie	Marie	2015	0	PCSI1	NULL	NULL	NULL

SELECT

On peut faire des calculs dans les requêtes:

```
SELECT nom, poids / ((4/3)*3.14*POW(rayon, 3))
FROM planete;
```

Que fait cette requête?

SELECT

On peut faire des calculs dans les requêtes:

```
SELECT nom, poids / ((4/3)*3.14*POW(rayon, 3))
FROM planete;
```

Que fait cette requête?
Elle affiche le nom et la densité de chaque planète.

SELECT

On peut faire des calculs dans les requêtes:

```
SELECT nom, poids / ((4/3)*3.14*POW(rayon, 3))
FROM planete;
```

Que fait cette requête?
Elle affiche le nom et la densité de chaque planète.

Exercice

L'attribut `rayon` est en km.
Écrire une requête pour afficher le rayon de chaque planète en mètres.

AS

Il est possible de renommer une colonne avec `AS`:

```
SELECT nom, poids / ((4/3)*3.14*POW(rayon, 3)) AS densite
FROM planete;
```

AS

Il est possible de renommer une colonne avec `AS`:

```
SELECT nom, poids / ((4/3)*3.14*POW(rayon, 3)) AS densite
FROM planete;
```

Utile pour y faire référence!

WHERE

Il est possible de récupérer seulement les enregistrements vérifiant une condition avec `WHERE`:

```
SELECT colonne_1, ..., colonne_n FROM table
WHERE condition;
```

WHERE

Il est possible de récupérer seulement les enregistrements vérifiant une condition avec `WHERE`:

```
SELECT colonne_1, ..., colonne_n FROM table
WHERE condition;
```

Dans `condition` on peut utiliser:

1. = (et non pas ==)
2. <, <=
3. != (ou son équivalent <>)
4. `AND`, `OR`
5. `LIKE`

WHERE

Pour afficher les noms des élèves qui ont été en MPSI2, on écrira:

WHERE

Pour afficher les noms des élèves qui ont été en MPSI2, on écrira:

```sql
SELECT nom FROM eleve WHERE classe_sup = 'MPSI2';
```

Pour afficher les noms des élèves qui sont passés de MPSI à PSI:

WHERE

Pour afficher les noms des élèves qui ont été en MPSI2, on écrira:

```sql
SELECT nom FROM eleve WHERE classe_sup = 'MPSI2';
```

Pour afficher les noms des élèves qui sont passés de MPSI à PSI:

```sql
SELECT nom FROM eleve
WHERE (classe_sup = 'MPSI1' OR classe_sup = 'MPSI2')
      AND classe_spe = 'PSI';
```

Comment afficher les planètes de rayon supérieur à 50000 km?

WHERE

LIKE permet d'établir une condition sur la forme d'une chaîne de caractères d'un attribut:

```
attribut LIKE motif
```

WHERE

LIKE permet d'établir une condition sur la forme d'une chaîne de caractères d'un attribut:

```
attribut LIKE motif
```

`motif` doit être une chaîne de caractères qui peut contenir:

- %: pour n'importe quelle chaîne de caractères
- _: pour n'importe quel (unique) caractère

WHERE

Que fait la requête suivante?

```sql
SELECT * FROM eleve WHERE ecole LIKE 'Centrale%';
```

WHERE

Que fait la requête suivante?

```sql
SELECT * FROM eleve WHERE ecole LIKE 'Centrale%';
```

Comment avoir les prénoms des élèves qui ont fait une classe étoile?

WHERE

Que fait la requête suivante?

```sql
SELECT * FROM eleve WHERE ecole LIKE 'Centrale%';
```

Comment avoir les prénoms des élèves qui ont fait une classe étoile?

```sql
SELECT prenom FROM eleve WHERE classe_spe LIKE '%*';
```

ORDER BY

`ORDER BY` permet de trier dans l'ordre croissant les enregistrements en fonction d'un attribut. On peut ajouter `DESC` pour trier dans l'ordre décroissant.

Exemples:

```
SELECT nom FROM eleve ORDER BY nom;
```

ORDER BY

`ORDER BY` permet de trier dans l'ordre croissant les enregistrements en fonction d'un attribut. On peut ajouter `DESC` pour trier dans l'ordre décroissant.

Exemples:

```
SELECT nom FROM eleve ORDER BY nom;
```

les noms d'élèves par ordre alphabétique

```
SELECT * FROM planete
WHERE etoile = 'Soleil'
ORDER BY poids DESC;
```

ORDER BY

`ORDER BY` permet de trier dans l'ordre croissant les enregistrements en fonction d'un attribut. On peut ajouter `DESC` pour trier dans l'ordre décroissant.

Exemples:

```
SELECT nom FROM eleve ORDER BY nom;
```

les noms d'élèves par ordre alphabétique

```
SELECT * FROM planete
WHERE etoile = 'Soleil'
ORDER BY poids DESC;
```

les planètes du système solaire de la plus lourde à la plus légère

LIMIT

`LIMIT k` permet de limiter le nombre d'enregistrements aux k premières valeurs. Il est souvent utilisé avec `ORDER BY`.

LIMIT

`LIMIT k` permet de limiter le nombre d'enregistrements aux k premières valeurs. Il est souvent utilisé avec `ORDER BY`.

```sql
SELECT * FROM planete
WHERE etoile = 'Soleil'
ORDER BY poids DESC
LIMIT 3;
```

Donne les 3 planètes les plus lourdes du système solaire.

OFFSET

`OFFSET p` permet d'afficher les enregistrements à partir du $(p+1)$ème. Il est souvent utilisé avec `ORDER BY`.

OFFSET

`OFFSET p` permet d'afficher les enregistrements à partir du $(p+1)$ème. Il est souvent utilisé avec `ORDER BY`.

```sql
SELECT * FROM planete
WHERE etoile = 'Soleil'
ORDER BY poids
LIMIT 1
OFFSET 2;
```

Donne la 3ème planète la plus légère du système solaire.

Récapitulatif

Toutes les commandes optionnelles de SELECT doivent être **écrites dans cet ordre**:

```
SELECT colonne_1, ..., colonne_n
FROM nom_table
WHERE conditions
ORDER BY colonne_i
LIMIT k
OFFSET p;
```

Exercices

`planete (nom, poids, rayon)`

Question

Comment obtenir, dans la table `planete`, la deuxième planète la plus dense connue?

Exercices

`planete (nom, poids, rayon)`

Question

Comment obtenir, dans la table `planete`, la deuxième planète la plus dense connue?

```
eleve (nom, annee_entree, ecole, classe_sup,
       classe_spe, classe_spe2)
```

Question

Comment obtenir, dans la table `eleve`, les noms des 3 derniers élèves entrés dans une ENS en MP*?