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?

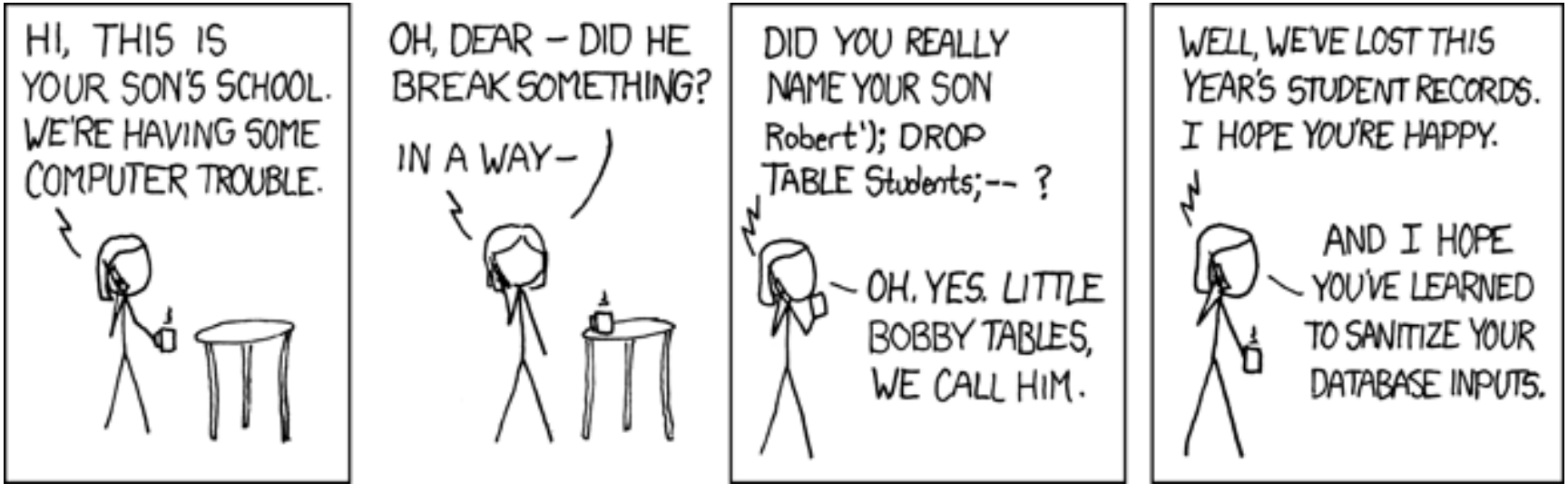
```
eleve (nom, annee_entree, ecole, classe_sup,
       classe_spe, classe_spe2)
```

Question

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

Bases de données 2: plusieurs tables

Informatique pour tous



Rappels

Question 12 Dans une base de données, on souhaite sélectionner les trois champs “id”, “question” et “reponse” de tous les enregistrements d’une table nommée “QCM”. Quelle(s) requête(s) SQL peut-on utiliser ?

A) SELECT * FROM QCM

B) SELECT in QCM ALL id,question,reponse

C) SELECT id,question,reponse FROM QCM

D) SELECT QCM WHERE id,question,reponse

Projection

Étant donné une table R, une **projection** de R revient à ne conserver que certains attributs:

R		
A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_2	c_3

$\pi_{(A,B)}(R)$	
A	B
a_1	b_1
a_2	b_2

Projection

Étant donné une table R, une **projection** de R revient à ne conserver que certains attributs:

R		
A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_2	c_3

$\pi_{(A,B)}(R)$	
A	B
a_1	b_1
a_2	b_2

En SQL: `SELECT DISTINCT A, B FROM R;`

Sélection

Étant donné une table R, une **sélection** $\sigma_{\text{condition}}(R)$ de R permet d'obtenir les enregistrements vérifiant une condition:

R		
A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_2	c_3

$\sigma_{A=a_1}(R)$		
A	B	C
a_1	b_1	c_1
a_1	b_1	c_2

Sélection

Étant donné une table R, une **sélection** $\sigma_{\text{condition}}(R)$ de R permet d'obtenir les enregistrements vérifiant une condition:

R		
A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_2	c_3

$\sigma_{A=a_1}(R)$		
A	B	C
a_1	b_1	c_1
a_1	b_1	c_2

En SQL: `SELECT * FROM R WHERE A = a1;`

Exemple

Expliquer ce que donne l'expression suivante:

$$\pi_{nom,rayon}(\ \sigma_{etoile='Soleil'}(planete)\ )$$

Dans la table: planete (nom, rayon, poids, etoile)

Union

Si R_1 et R_2 sont des tables ayant le **même schéma relationnel**, $R_1 \cup R_2$ contient les enregistrements dans R_1 ou R_2:

R_1		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

R_2		
A	B	C
a_1	b_1	c_1
a_3	b_3	c_3

$R_1 \cup R_2$		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3

En SQL: `SELECT * FROM R1 UNION SELECT * FROM R2;`

Exemple

Étant donné des tables `eleve_hugo`, `eleve_haag`, `eleve_pergaud`, il est possible d'obtenir tous les élèves de CPGE de Besançon:

```
SELECT * FROM eleve_hugo
UNION SELECT * FROM eleve_haag
UNION SELECT * FROM eleve_pergaud;
```

Différence

Si R_1 et R_2 sont des tables ayant le **même schéma relationnel**, $R_1 - R_2$ contient les enregistrements dans R_1 mais pas dans R_2:

R_1		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

R_2		
A	B	C
a_1	b_1	c_1
a_3	b_3	c_3

$R_1 - R_2$		
A	B	C
a_2	b_2	c_2

En SQL: `SELECT * FROM R1 MINUS SELECT * FROM R2;`

Intersection

Si R_1 et R_2 sont des tables ayant le **même schéma relationnel**, $R_1 \cap R_2$ contient les enregistrements à la fois dans R_1 et R_2:

R_1		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

R_2		
A	B	C
a_1	b_1	c_1
a_3	b_3	c_3

$R_1 \cap R_2$		
A	B	C
a_1	b_1	c_1

En SQL: `SELECT * FROM R1 INTERSECT SELECT * FROM R2;`

Produit cartésien

On peut réaliser le **produit cartésien** $R_1 \times R_2$ de deux tables:

R_1		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

R_2	
D	E
d_1	e_1
d_2	e_2

$R_1 \times R_2$				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_1	b_1	c_1	d_2	e_2
a_2	b_2	c_2	d_1	e_1
a_2	b_2	c_2	d_2	e_2

En SQL: `SELECT * FROM R1, R2;`

Produit cartésien

On peut réaliser le **produit cartésien** $R_1 \times R_2$ de deux tables:

R_1		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

R_2	
D	E
d_1	e_1
d_2	e_2

$R_1 \times R_2$				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_1	b_1	c_1	d_2	e_2
a_2	b_2	c_2	d_1	e_1
a_2	b_2	c_2	d_2	e_2

En SQL: `SELECT * FROM R1, R2;`

On peut aussi sélectionner seulement certaines colonnes de $R_1 \times R_2$ en écrivant, par exemple, `SELECT A, B FROM R1, R2;`

Exemple

Considérons une base de donnée bibliotheque avec les tables:

1. livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
2. emprunteur (id : INT, nom : CHAR(50))
3. emprunt (id_emprunteur : INT, titre_livre: CHAR(50))

Exemple

Considérons une base de donnée bibliotheque avec les tables:

1. livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
2. emprunteur (id : INT, nom : CHAR(50))
3. emprunt (id_emprunteur : INT, titre_livre: CHAR(50))

Comment obtenir les emprunteurs qui sont aussi auteurs?

Exemple

Considérons une base de donnée bibliotheque avec les tables:

1. livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
2. emprunteur (id : INT, nom : CHAR(50))
3. emprunt (id_emprunteur : INT, titre_livre: CHAR(50))

Comment obtenir les emprunteurs qui sont aussi auteurs?

```
SELECT nom FROM emprunteur, livre
WHERE nom = auteur;
```

Exemple

1. livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
2. emprunteur (id : INT, nom : CHAR(50))
3. emprunt (id_emprunteur : INT, titre_livre: CHAR(50))

Comment obtenir les noms des personnes qui ont emprunté le livre dont le titre est Le Banquet?

Exemple

1. livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
2. emprunteur (id : INT, nom : CHAR(50))
3. emprunt (id_emprunteur : INT, titre_livre: CHAR(50))

Comment obtenir les noms des personnes qui ont emprunté le livre dont le titre est Le Banquet?

```
SELECT nom FROM emprunteur, emprunt
WHERE id = id_emprunteur
AND titre_livre = 'Le Banquet';
```

JOIN

La jointure $R_1 \bowtie_{A=D} R_2$ de deux tables R_1 et R_2 revient à combiner les enregistrements de R_1 et R_2 en identifiant les colonnes A et D:

JOIN

La jointure $R_1 \bowtie_{A=D} R_2$ de deux tables R_1 et R_2 revient à combiner les enregistrements de R_1 et R_2 en identifiant les colonnes A et D:

R_1		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3

R_2	
D	E
a_1	e_1
a_2	e_2

$R_1 \bowtie_{A=D} R_2$			
A	B	C	E
a_1	b_1	c_1	e_1
a_2	b_2	c_2	e_2

En SQL: `SELECT ... FROM R1 JOIN R2 ON A = D;`

Exemple

1. livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
2. emprunteur (id : INT, nom : CHAR(50))
3. emprunt (id_emprunteur : INT, titre_livre: CHAR(50))

Comment obtenir les noms des personnes qui ont emprunté le livre Le Banquet?

Exemple

1. livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
2. emprunteur (id : INT, nom : CHAR(50))
3. emprunt (id_emprunteur : INT, titre_livre: CHAR(50))

Comment obtenir les noms des personnes qui ont emprunté le livre Le Banquet?

On peut aussi utiliser une jointure:

```
SELECT nom FROM emprunteur
JOIN emprunt ON id = id_emprunteur
WHERE titre_livre = 'Le Banquet';
```

Exemple

1. livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
2. emprunteur (id : INT, nom : CHAR(50))
3. emprunt (id_emprunteur : INT, titre_livre: CHAR(50))

Comment obtenir les titres des livres empruntés par M. Machin?

Exemple

1. livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
2. emprunteur (id : INT, nom : CHAR(50))
3. emprunt (id_emprunteur : INT, titre_livre: CHAR(50))

Comment obtenir les titres des livres empruntés par M. Machin?

```
SELECT titre_livre FROM emprunteur
JOIN emprunt ON id = id_emprunteur
WHERE nom = 'Machin';
```

Exemple

1. livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
2. emprunteur (id : INT, nom : CHAR(50))
3. emprunt (id_emprunteur : INT, titre_livre: CHAR(50))

Comment obtenir les noms des personnes ayant emprunté un livre écrit par Stephen King?

Exemple

1. livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
2. emprunteur (id : INT, nom : CHAR(50))
3. emprunt (id_emprunteur : INT, titre_livre: CHAR(50))

Comment obtenir les noms des personnes ayant emprunté un livre écrit par Stephen King?

```
SELECT nom FROM emprunteur
JOIN emprunt ON id = id_emprunteur
JOIN livre ON titre_livre = titre
WHERE auteur = 'Stephen King';
```

Exemple

1. livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
2. emprunteur (id : INT, nom : CHAR(50))
3. emprunt (id_emprunteur : INT, titre_livre: CHAR(50))

Comment obtenir les plus gros livres empruntés avec leur nombre de pages?

Exemple

1. livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
2. emprunteur (id : INT, nom : CHAR(50))
3. emprunt (id_emprunteur : INT, titre_livre: CHAR(50))

Comment obtenir les plus gros livres empruntés avec leur nombre de pages?

```
SELECT titre, pages FROM livre
JOIN emprunt ON titre_livre = titre
ORDER BY pages DESC;
```

Ambiguïté

1. livre (**id** : INT, titre : CHAR(50), auteur : CHAR(50), pages : INT)
2. emprunteur (**id** : INT, nom : CHAR(50))
3. emprunt (id_emprunteur : INT, titre_livre : CHAR(50))

Problème: comment savoir, dans livre $\times$ emprunteur, à quelle table **id** fait référence?

Ambiguïté

1. livre (**id** INT, titre CHAR(50), auteur CHAR(50), pages INT)
2. emprunteur (**id** INT, nom CHAR(50))
3. emprunt (id_emprunteur INT, titre_livre CHAR(50))

```
SELECT id FROM livre, emprunteur;
```

Résultat:
ERROR 1052 (23000): Column 'id' in field list is ambiguous

Ambiguïté

1. livre (**id** INT, titre CHAR(50), auteur CHAR(50), pages INT)
2. emprunteur (**id** INT, nom CHAR(50))
3. emprunt (id_emprunteur INT, titre_livre CHAR(50))

Solution:

```
SELECT livre.id FROM livre, emprunteur;
```

Ambiguïté

1. livre (**id** INT, titre CHAR(50), auteur CHAR(50), pages INT)
2. emprunteur (**id** INT, nom CHAR(50))
3. emprunt (id_emprunteur INT, titre_livre CHAR(50))

Problème 2: afficher tous les couples de livres ayant le même nombre de pages.

Ambiguïté

1. livre (**id** INT, titre CHAR(50), auteur CHAR(50), pages INT)
2. emprunteur (**id** INT, nom CHAR(50))
3. emprunt (id_emprunteur INT, titre_livre CHAR(50))

Problème 2: afficher tous les couples de livres ayant le même nombre de pages.

```
SELECT titre, titre FROM livre, livre WHERE pages = pages;
```

Ne marche pas du tout!

Ambiguïté

1. livre (**id** INT, titre CHAR(50), auteur CHAR(50), pages INT)
2. emprunteur (**id** INT, nom CHAR(50))
3. emprunt (id_emprunteur INT, titre_livre CHAR(50))

Solution: renommer les tables (temporairement).

Ambiguïté

1. livre (**id** INT, titre CHAR(50), auteur CHAR(50), pages INT)
2. emprunteur (**id** INT, nom CHAR(50))
3. emprunt (id_emprunteur INT, titre_livre CHAR(50))

Solution: renommer les tables (temporairement).

```
SELECT liv1.titre, liv2.titre
FROM livre AS liv1, livre AS liv2
WHERE liv1.pages = liv2.pages;
```

Extrait Mines-Pont 2017

On modélise ici un réseau routier par un ensemble de *croisements* et de *voies* reliant ces croisements. Les voies partent d'un croisement et arrivent à un autre croisement. Ainsi, pour modéliser une route à double sens, on utilise deux voies circulant en sens opposés.
La base de données du réseau routier est constituée des relations suivantes :

- Croisement(<u>id</u>, longitude, latitude)
- Voie(<u>id</u>, longueur, id_croisement_debut, id_croisement_fin)

Dans la suite on considère c l'identifiant (`id`) d'un croisement donné.

❏ **Q26** – Écrire la requête SQL qui renvoie les identifiants des croisements atteignables en utilisant une seule voie à partir du croisement ayant l'identifiant c.

❏ **Q27** – Écrire la requête SQL qui renvoie les longitudes et latitudes des croisements atteignables en utilisant une seule voie, à partir du croisement c.

❏ **Q28** – Que renvoie la requête SQL suivante ?

```
1  SELECT V2.id_croisement_fin
2  FROM   Voie as V1
3  JOIN   Voie as V2
4  ON     V1.id_croisement_fin = V2.id_croisement_debut
5  WHERE  V1.id_croisement_debut = c
```

Bases de données 3: fonctions d'agrégation

Informatique pour tous

Fonctions d'agrégations

Fonctions s'appliquant sur un attribut a:

- `MAX(a)`: maximum de a parmi les enregistrements
- `MIN(a)`: minimum de a parmi les enregistrements
- `SUM(a)`: somme de a parmi les enregistrements
- `AVG(a)`: moyenne de a parmi les enregistrements
- `COUNT(*)`: nombre total d'enregistrements

Exemples

1. livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
2. emprunteur (id : INT, nom : CHAR(50))
3. emprunt (id_emprunteur : INT, titre_livre: CHAR(50))

Question

Comment obtenir le nombre moyen de pages d'un livre?

Exemples

1. livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
2. emprunteur (id : INT, nom : CHAR(50))
3. emprunt (id_emprunteur : INT, titre_livre: CHAR(50))

Question

Comment obtenir le nombre moyen de pages d'un livre?

```
SELECT AVG(pages) FROM livre;
```

Exemples

1. livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
2. emprunteur (id : INT, nom : CHAR(50))
3. emprunt (id_emprunteur : INT, titre_livre: CHAR(50))

Question

Comment obtenir le nombre de livres empruntés par M. Machin?

Exemples

1. livre (titre : CHAR(50), auteur : CHAR(50), pages : INT)
2. emprunteur (id : INT, nom : CHAR(50))
3. emprunt (id_emprunteur : INT, titre_livre: CHAR(50))

Question

Comment obtenir le nombre de livres empruntés par M. Machin?

```
SELECT COUNT(*) FROM emprunt
JOIN emprunteur ON id = id_emprunteur
WHERE nom = 'Machin';
```

Exemples

Dans `eleve (nom, classe, note, ...)`, comment obtenir la note maximum dans le classe de PC?

Exemples

Dans `eleve (nom, classe, note, ...)`, comment obtenir la note maximum dans le classe de PC?

```
SELECT MAX(note)
FROM eleve
WHERE classe = 'PC';
```

Exemples

Étant données les tables
`planete(nom, rayon, poids, nom_etoile)` et
`etoile(nom, galaxie)` comment obtenir la somme des poids des planètes de la Voie lactée?

Exercice ICNA

Question 13 On dispose d'une table de données dont le schéma de relation est le suivant :

candidats(identifiant, nom, prénom, rang, moyenne, age, adresse),

dont les attributs sont respectivement l'identifiant des candidats, leur nom, leur prénom, leur rang au concours, leur moyenne, leur âge et leur adresse postale. La requête SQL suivante :

```
SELECT COUNT(*) FROM candidats WHERE moyenne>10
```

A) permet de lister tous les candidats de la table candidats.

B) renvoie le nombre de candidats dont la moyenne est supérieure ou égale à 10.

C) permet de regrouper les candidats ayant la même moyenne.

D) provoque une erreur.

Groupes

Par défaut, les fonctions d'agrégations s'appliquent sur tous les enregistrements de la table.

Groupes

Par défaut, les fonctions d'agrégations s'appliquent sur tous les enregistrements de la table.

Il est possible de séparer les enregistrements en groupes avec `GROUP BY`, pour ensuite appliquer une fonction à chaque groupe.

Groupes

```
SELECT ... FROM ... GROUP BY attribut;
```

a pour effet de grouper les résultats par même attribut.
Il y a un résultat affiché pour chaque valeur possible de `attribut`.

Groupes

```
SELECT ... FROM ... GROUP BY attribut;
```

a pour effet de grouper les résultats par même attribut.
Il y a un résultat affiché pour chaque valeur possible de `attribut`.

Les fonctions d'agrégations dans le `SELECT` s'appliquent alors à chaque groupe.

Exemples

```
SELECT Continent, SUM(Population)
FROM Country
GROUP BY Continent;
```

Exemples

```
SELECT Continent, SUM(Population)
FROM Country
GROUP BY Continent;
```

```
+---------------+------------------+
| Continent     | SUM(Population)  |
+---------------+------------------+
| Asia          |       3705025700 |
| Europe        |        730074600 |
| North America |        482993000 |
| Africa        |        784475000 |
| Oceania       |         30401150 |
| Antarctica    |                0 |
| South America |        345780000 |
+---------------+------------------+
```

Remarques

Attention: il ne faut pas afficher un attribut si celui-ci n'est pas le même pour chaque élément d'un groupe.

Remarques

Attention: il ne faut pas afficher un attribut si celui-ci n'est pas le même pour chaque élément d'un groupe.

```
SELECT Continent, Name, SUM(Population)
FROM Country
GROUP BY Continent;
```

Remarques

Attention: il ne faut pas afficher un attribut si celui-ci n'est pas le même pour chaque élément d'un groupe.

```
SELECT Continent, Name, SUM(Population)
FROM Country
GROUP BY Continent;
```

Continent	Name	SUM(Population)
Asia	Afghanistan	3705025700
Europe	Albania	730074600
North America	Aruba	482993000
Africa	Angola	784475000
Oceania	American Samoa	30401150
Antarctica	Antarctica	0
South America	Argentina	345780000

Exemples

Comment afficher la densité de population de chaque continent?

Exemples

Comment afficher la densité de population de chaque continent?

```
SELECT Continent, SUM(Population) / SUM(Surface)
FROM Country
GROUP BY Continent;
```

Exemples

Comment afficher chaque continent trié par ordre décroissant de densité de population?

Exemples

Comment afficher chaque continent trié par ordre décroissant de densité de population?

```
SELECT Continent, SUM(Population)/SUM(Surface) AS densite
FROM Country
GROUP BY Continent
ORDER BY densite DESC;
```

Exemples

Dans `eleve (nom, classe, ...)`, comment afficher chaque classe avec son nombre d'élèves?

Exemples

Dans `eleve (nom, classe, ...)`, comment afficher chaque classe avec son nombre d'élèves?

```
SELECT classe, COUNT(*) FROM eleve GROUP BY classe;
```

```
+---------------+-----------------+
| classe        | COUNT(*)        |
+---------------+-----------------+
| MPSI1         |              41 |
| MPSI2         |              38 |
| PCSI1         |              40 |
| PCSI2         |              37 |
+---------------+-----------------+
```

Exemples

Dans la table `eleve(nom, classe, note, ...)`, comment afficher la moyenne, note maximum et note minimum de chaque classe?

Exemples

Dans la table `eleve(nom, classe, note, ...)`, comment afficher la moyenne, note maximum et note minimum de chaque classe?

```
SELECT classe, AVG(note), MAX(note), MIN(note)
FROM eleve
GROUP BY classe;
```

Exemples

Étant données les tables
`planete(nom, rayon, poids, nom_etoile)` et
`etoile(nom, galaxie)` comment obtenir, pour chaque étoile, le nombre de planètes tournant autour?

HAVING

Lorsque l'on groupe des enregistrements avec `GROUP BY`, on peut afficher seulement les groupes vérifiant une condition avec `HAVING`.

HAVING

Lorsque l'on groupe des enregistrements avec `GROUP BY`, on peut afficher seulement les groupes vérifiant une condition avec `HAVING`.

`WHERE` sert à établir une condition sur les **enregistrements** affichés.
`HAVING` sert à établir une condition sur les **groupes** affichés.

HAVING

Lorsque l'on groupe des enregistrements avec `GROUP BY`, on peut afficher seulement les groupes vérifiant une condition avec `HAVING`.

`WHERE` sert à établir une condition sur les **enregistrements** affichés.
`HAVING` sert à établir une condition sur les **groupes** affichés.

`HAVING` ne peut être utilisé qu'à la suite d'un `GROUP BY`.

Exemples

Dans la table `eleve(nom, classe, ...)`, comment afficher que les classes avec au moins 40 élèves?

Exemples

Dans la table `eleve(nom, classe, ...)`, comment afficher que les classes avec au moins 40 élèves?

```
SELECT classe, COUNT(*)
FROM eleve
GROUP BY classe
HAVING COUNT(*) >= 40;
```

Exemples

Dans `eleve (nom, classe, note, ...)`, comment afficher que les classes dont la moyenne est ≥ 12?

Exemples

Dans `eleve (nom, classe, note, ...)`, comment afficher que les classes dont la moyenne est ≥ 12?

```
SELECT classe
FROM eleve
GROUP BY classe
HAVING AVG(note) >= 12;
```

Exemples

1. livre (titre CHAR(50), auteur CHAR(50), pages INT)
2. emprunteur (id INT, nom CHAR(50))
3. emprunt (id_emprunteur INT, titre_livre CHAR(50))

Comment afficher les noms des personnes ayant emprunté au moins 5 livres?

Exemples

1. livre (titre CHAR(50), auteur CHAR(50), pages INT)
2. emprunteur (id INT, nom CHAR(50))
3. emprunt (id_emprunteur INT, titre_livre CHAR(50))

Comment afficher les noms des personnes ayant emprunté au moins 5 livres?

```
SELECT nom FROM emprunteur
JOIN emprunt ON id = id_emprunteur
GROUP BY nom
HAVING COUNT(*) >= 5;
```

Bases de données 4: SELECT imbriqués

Informatique pour tous

Extrait Mines-Pont 2017

On modélise ici un réseau routier par un ensemble de *croisements* et de *voies* reliant ces croisements. Les voies partent d'un croisement et arrivent à un autre croisement. Ainsi, pour modéliser une route à double sens, on utilise deux voies circulant en sens opposés.
La base de données du réseau routier est constituée des relations suivantes :

- Croisement(<u>id</u>, longitude, latitude)
- Voie(<u>id</u>, longueur, id_croisement_debut, id_croisement_fin)

Dans la suite on considère c l'identifiant (`id`) d'un croisement donné.

❏ **Q26** – Écrire la requête SQL qui renvoie les identifiants des croisements atteignables en utilisant une seule voie à partir du croisement ayant l'identifiant c.

❏ **Q27** – Écrire la requête SQL qui renvoie les longitudes et latitudes des croisements atteignables en utilisant une seule voie, à partir du croisement c.

❏ **Q28** – Que renvoie la requête SQL suivante ?

```
1  SELECT V2.id_croisement_fin
2  FROM   Voie as V1
3  JOIN   Voie as V2
4  ON     V1.id_croisement_fin = V2.id_croisement_debut
5  WHERE  V1.id_croisement_debut = c
```

SELECT imbriqués

Il est possible d'utiliser le résultat d'un `SELECT` à l'intérieur d'un autre `SELECT`, souvent dans un `WHERE` ou `HAVING`.

SELECT imbriqués

Dans `eleve(nom, note, ...)`, comment trouver le nom de l'élève ayant la note maximum?

SELECT imbriqués

Dans `eleve(nom, note, ...)`, comment trouver le nom de l'élève ayant la note maximum?

```
SELECT nom FROM eleve
WHERE note = (SELECT MAX(note) FROM eleve);
```

Autre solution

Dans `eleve(nom, note, ...)`, comment trouver le nom de l'élève ayant la note maximum, sans utiliser de `SELECT` imbriqué?

Autre solution

Dans `eleve(nom, note, ...)`, comment trouver le nom de l'élève ayant la note maximum, sans utiliser de `SELECT` imbriqué?

```
SELECT nom FROM eleve
ORDER BY note DESC
LIMIT 1;
```

SELECT imbriqués

Dans `Country(name, pib, population, ...)`, comment trouver le nom des pays ayant un PIB par habitant supérieur à la moyenne mondiale?

SELECT imbriqués

Dans `Country(name, pib, population, ...)`, comment trouver le nom des pays ayant un PIB par habitant supérieur à la moyenne mondiale?

```
SELECT name FROM Country
WHERE (pib / population) > (SELECT AVG(pib / population)
                            FROM Country);
```

SELECT imbriqués

Dans `planete(nom, etoile, ...)`, comment trouver les noms des planetes dans le même système que la planète Proxima b?

SELECT imbriqués

Dans `planete(nom, etoile, ...)`, comment trouver les noms des planetes dans le même système que la planète Proxima b?

```
SELECT p1.name FROM planete AS p1
WHERE p1.etoile = (SELECT p2.etoile
FROM planete AS p2 WHERE p2.nom = 'Proxima b');
```

Autre solution

Dans `planete(nom, etoile, ...)`, comment trouver les noms des planetes dans le même système que Proxima b, sans utiliser de `SELECT` imbriqué?

Autre solution

Dans `planete(nom, etoile, ...)`, comment trouver les noms des planetes dans le même système que Proxima b, sans utiliser de `SELECT` imbriqué?

```
SELECT p1.name FROM planete AS p1, planete AS p2
WHERE p1.etoile = p2.etoile AND p2.name = 'Proxima b';
```

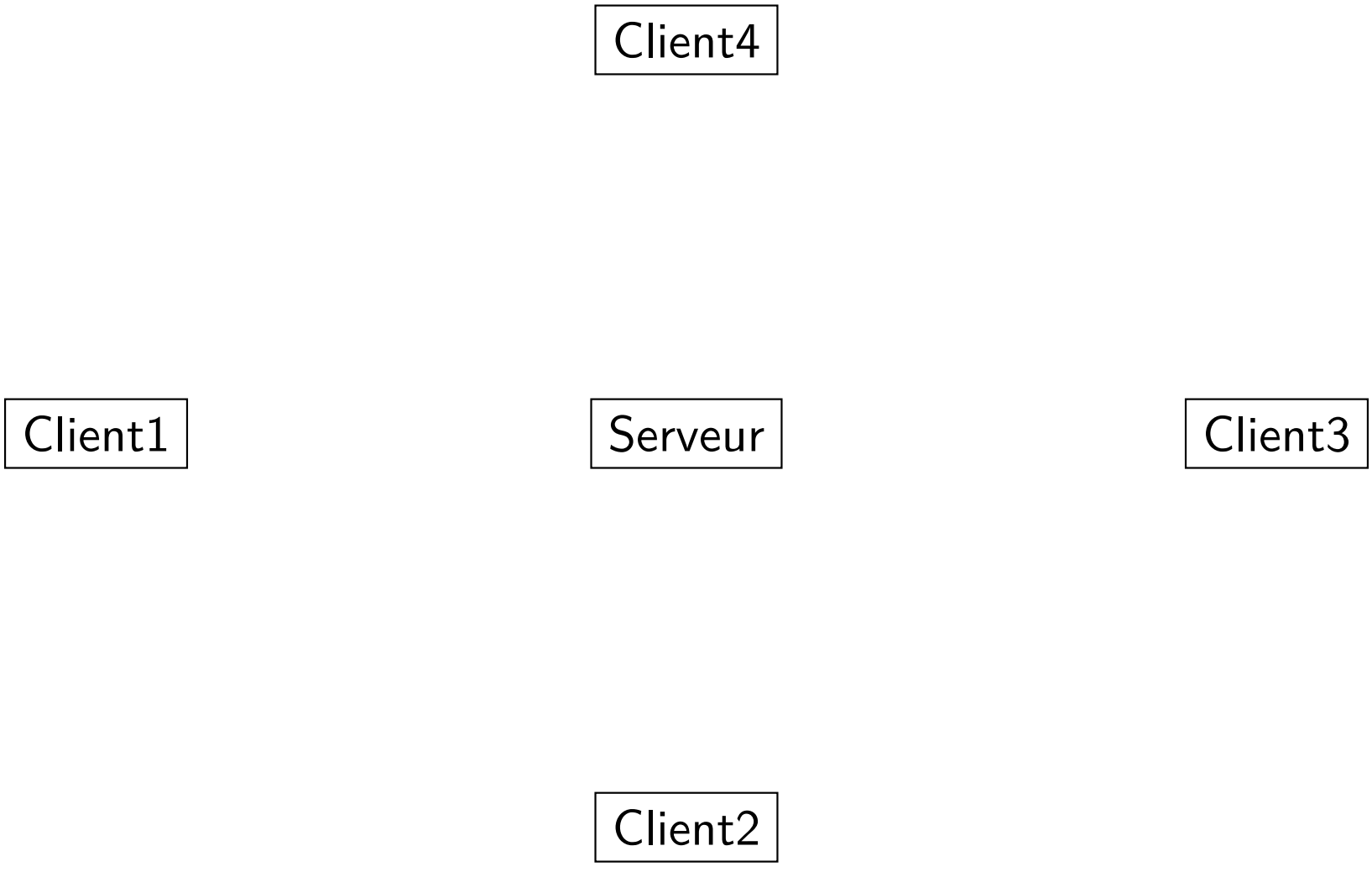
Architecture d'une base de données

Souvent une base de données doit être accessible par de nombreux utilisateurs.
Comment le faire en pratique?

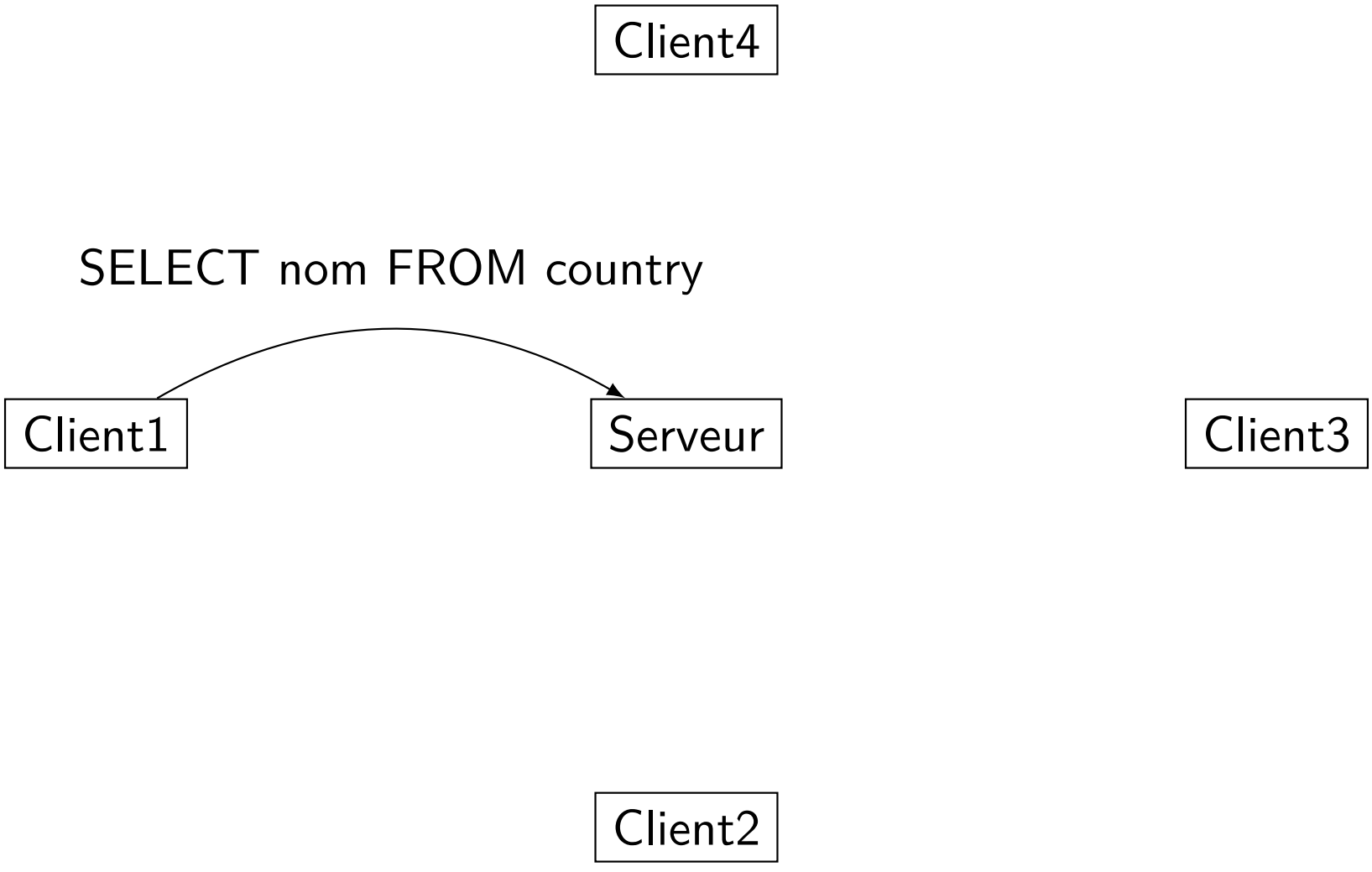
Architecture client-serveur

La méthode la plus simple consiste à avoir un **serveur** hébergeant la base de données.
Des clients peuvent alors s'y connecter et faire des requêtes.

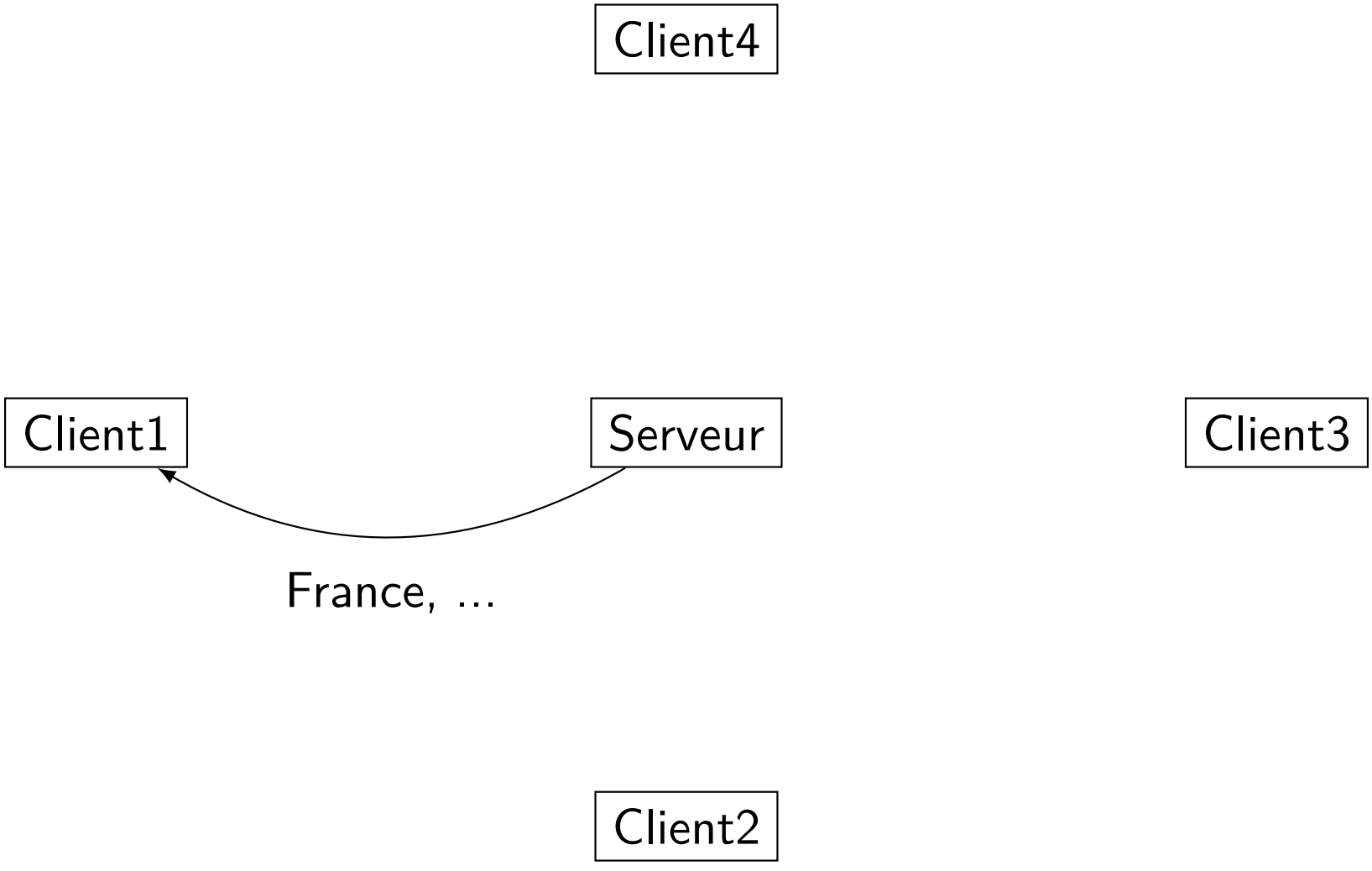
Architecture client-serveur



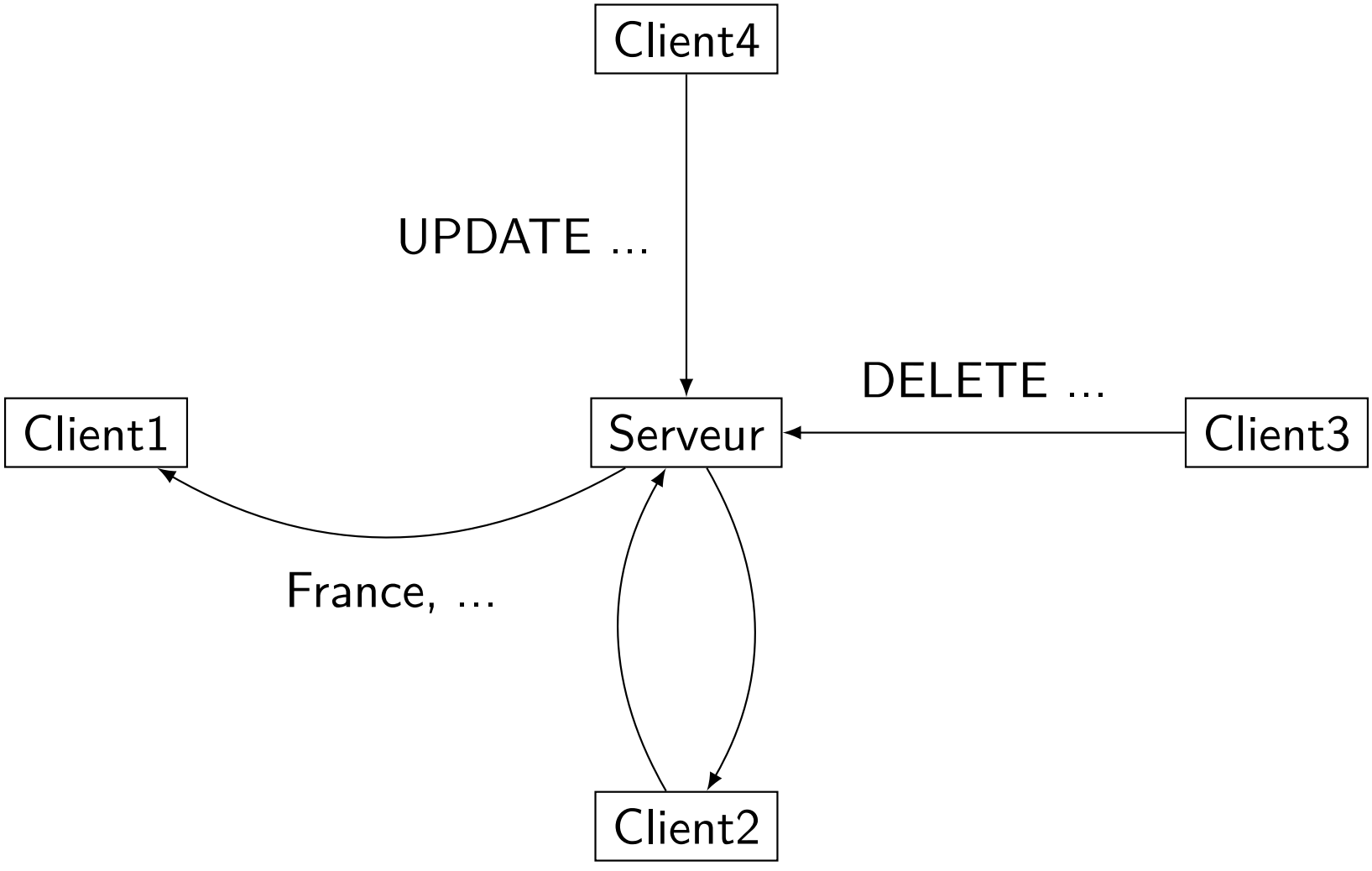
Architecture client-serveur



Architecture client-serveur



Architecture client-serveur



Architecture client-serveur

L'architecture client-serveur est omniprésente, et pas seulement pour les bases de données:

1. Web: consultation de pages web (HTTP)
2. Messagerie électronique (POP, IMAP, SMTP)
3. ...

Architecture client-serveur

Quelques soucis:

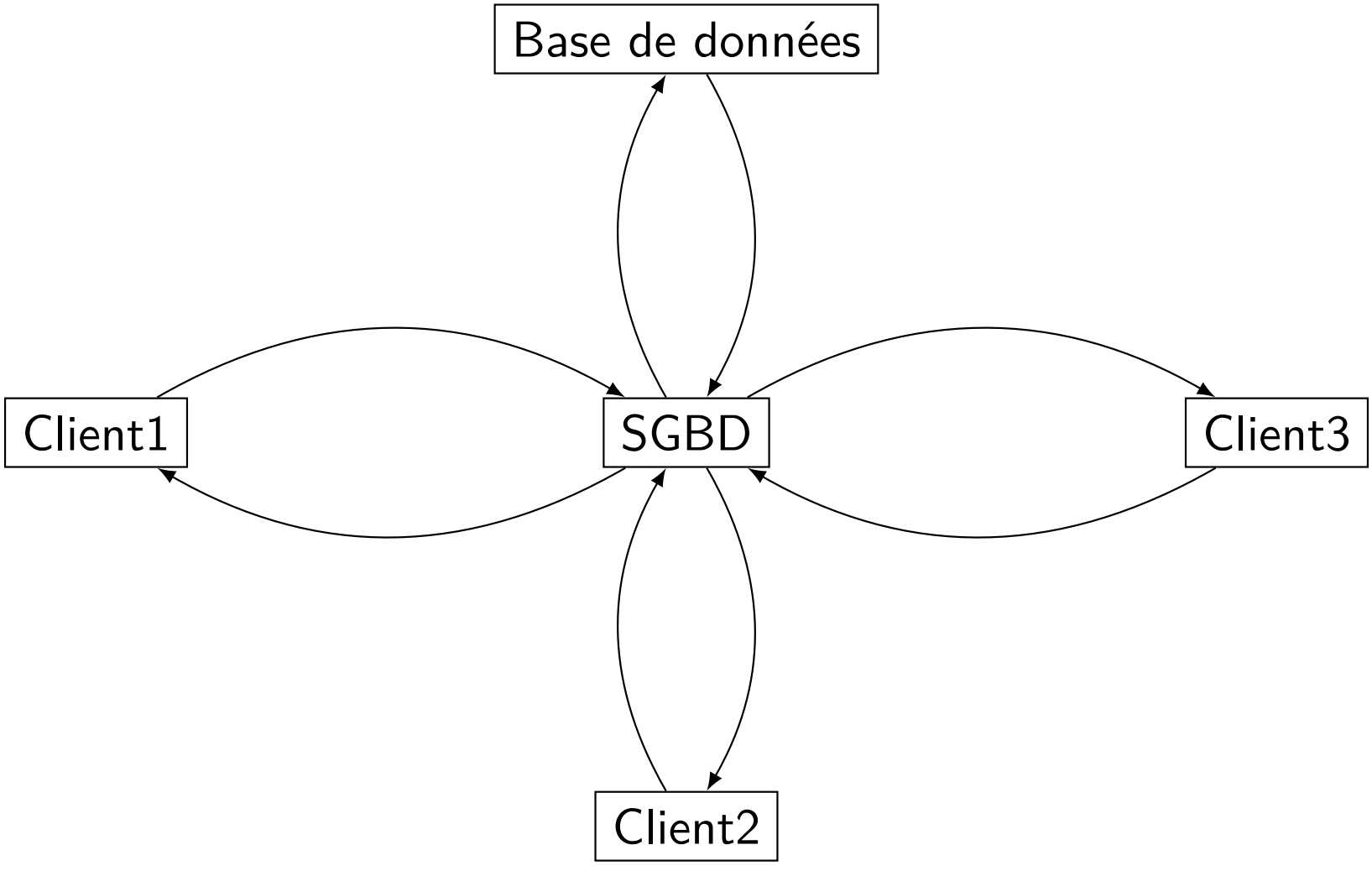
1. Le serveur peut planter au milieu d'une modification
2. Deux personnes peuvent modifier en même temps la base
3. Comment protéger la base de données d'accès non autorisés?

Architecture 3-tiers

L'**architecture 3-tiers** consiste à dupliquer le serveur en:

1. Un serveur contenant la base de données, non accessible par les clients
2. Un **système de gestion de bases de données** (**SGBD**): une interface (souvent graphique) entre clients et base de données, s'occupant de faire les requêtes et renvoyant le résultat aux client.

Architecture 3-tiers



Architecture 3-tiers

Intérêts de l'architecture 3-tiers:

1. Le SGBD est le seul pouvant modifier la base, et vérifie que les requêtes des clients ne sont pas malveillantes.
2. Le client n'a pas besoin de connaître SQL, si le SGBD propose une interface graphique.

Architecture 3-tiers

Souvent, le client communique avec le SGBD via une application web.

Avantage: pas besoin d'installer quoi que ce soit, utilisable sur tout système d'exploitation (Windows, Linux, Mac...).

QCM ICNA

Question 15 :

Pour les questions 15 à 18 on considère une base de données utilisée dans l'agence de location immobilière CHEZ MOI à Toulouse. Cette base de données contient les deux tables suivantes :

APPARTEMENTS(APT_ID,APT_PRO,APT_VILLE,APT_TARIF,APT_SURF)

PROPRIETAIRES(PRO_ID,PRO_NOM,PRO_PRENOM,PRO_ADRESSE,PRO_TEL)

La première regroupe les données sur les appartements : leur identifiant, l´ identifiant de leur propriétaire, la ville, le montant du loyer et la surface.
La deuxième regroupe les données sur les propriétaires des appartements : leur identifiant, le nom, le prénom, leur adresse et le numéro de téléphone.
Les clés primaires sont soulignées.
A quoi servent ces clés ?

QCM ICNA

Question 16 :

Que fait la requête SQL suivante :

```
SELECT * FROM APPARTEMENTS WHERE APT_SURF>40;
```

A) Elle sélectionne tous les champs de la table APPARTEMENTS.

B) Elle sélectionne les données de la table APPARTEMENTS concernant les appartements dont la surface est strictement supérieure à 40.

C) Elle compte le nombre d'appartement de surface supérieure à 40 dans la table APPARTEMENTS.

D) Elle n'est pas valide.

QCM ICNA

Question 17 :

Lesquelles des requêtes suivantes permettent d'obtenir l'identifiant des propriétaires des appartements dont le loyer est le plus élevé ?

A) SELECT APT_PRO FROM APPARTEMENTS WHERE APT_TARIF = (SELECT MAX(APT_TARIF) FROM APPARTEMENTS) ;

B) SELECT APT_PRO FROM APPARTEMENTS WHERE MAX(APT_TARIF) ;

C) SELECT APT_PRO FROM APPARTEMENTS WHERE APT_TARIF = (SELECT MAX(APT_TARIF) FROM PROPIETAIRES) ;

D) SELECT APT_PRO FROM PROPRIETAIRES WITH MAX(APT_TARIF) ;

QCM ICNA

Question 18 :

Quel opérateur est le plus adapté pour obtenir le numéro de téléphone des propriétaires des appartements dont le loyer est le plus élevé.

A) Une jointure.

B) Une division cartésienne.

C) Une intersection.

D) Il est impossible d'obtenir ces données car les tables contenant les numéros de téléphones et le montant des loyers sont distinctes.