

DUT SRC – IUT de Marne-la-Vallée
12/03/2014
M2203 – Bases de données

Cours 4

Le langage SQL et ses fonctionnalités avancées

Sources

- Cours de Tony Grandame à l'IUT de Marne-la-Vallée en 2010-2011

- Cours de Mathieu Mangeot, IUT de Savoie

<http://jibiki.univ-savoie.fr/~mangeot/Cours/BasesDeDonnees.pdf>

- Cours de Fabrice Meuzeret, IUT de Troyes

<http://195.83.128.55/~fmeuzeret/vrac/>

- Livre de Laurent Audibert : *Bases de données - de la modélisation au SQL*

Version partielle sur :

<http://laurent-audibert.developpez.com/Cours-BD/html/index.php>

Plan du cours 4 – Le langage SQL

- Résumé des épisodes précédents
- Langage de manipulation des données
- SQL avancé : les jointures
- SQL avancé : les groupements
- SQL avancé : les transactions
- SQL avancé : l'intégrité référentielle

Plan

- Résumé des épisodes précédents
- Langage de manipulation des données
- SQL avancé : les jointures
- SQL avancé : les groupements
- SQL avancé : les transactions
- SQL avancé : l'intégrité référentielle

Langage de création de données

- Création de base : `CREATE DATABASE`
- Suppression de base : `DROP DATABASE`
- Modification de base : `ALTER DATABASE`
- Création de table : `CREATE TABLE`
- Renommage de table : `RENAME TABLE`
- Suppression de table : `DROP TABLE`

Langage de modification de données

- Ajout de données : `INSERT`
- Modification de données : `UPDATE`
- Suppression de données : `DELETE`
- Consultation de données : `SELECT`
- Critères de sélection de données : `WHERE`

Plan

- Résumé des épisodes précédents
- Langage de manipulation des données
- SQL avancé : les jointures
- SQL avancé : les groupements
- SQL avancé : les transactions
- SQL avancé : l'intégrité référentielle

Langage de manipulation des données - SELECT

Le `WHERE` permet de préciser les critères de recherche et d'associer les tables entre elles.

Tous les opérateurs `=`, `<=>`, `<`, `>`, `!=`, `>=`, `<=`, `<>`, `BETWEEN`, `IN`, `NOT IN`, `IS NULL`, `IS NOT NULL`, ... sont supportés.

Pour chercher des données contenues dans une table ainsi que dans une autre table liées par le biais d'une clé étrangère, indispensable de préciser l'égalité entre les 2 champs.

Langage de manipulation des données - SELECT

Le `WHERE` permet de préciser les critères de recherche et d'associer les tables entre elles.

Tous les opérateurs `=`, `<=>`, `<`, `>`, `!=`, `>=`, `<=`, `<>`, `BETWEEN`, `IN`, `NOT IN`, `IS NULL`, `IS NOT NULL`, ... sont supportés.

Pour chercher des données contenues dans une table ainsi que dans une autre table liées par le biais d'une clé étrangère, indispensable de préciser l'égalité entre les 2 champs.

Attention : si toutes les tables listées dans la clause `FROM` ne sont pas associées dans la clause `WHERE`, le moteur effectuera un produit cartésien des tables non liées.

Ainsi si 3 tables de 500, 1000, et 2500 lignes sont appelées dans le `FROM` sans association dans la clause `WHERE`, le résultat sera de :

$500 * 1000 * 2500 = 1\ 250\ 000\ 000$ lignes.

Langage de manipulation des données - SELECT

Lire des données dans une ou plusieurs tables :

```
SELECT [DISTINCT] select_expression, ...
FROM table_references
    [WHERE where_definition]
    [ORDER BY {unsigned_integer | nom_de_colonne}
            [ASC | DESC] , ...]
    [LIMIT [offset,] lignes]
```

`select_expression` indique la colonne à lire, une constante, ou une valeur calculée.

Le `DISTINCT` permet de ne lire que des valeurs distinctes.

Le `FROM` permet de lister les tables à utiliser dans la recherche des données.

Le `ORDER BY` permet de trier le résultat de la requête (`ASC` : croissant, `DESC` : décroissant).

Langage de manipulation des données - SELECT

Exemples

On désire lire les noms rangés par ordre alphabétique de toutes les personnes qui se prénomment Lisa.

Personne	
<u>ID</u>	<u>int</u>
Nom	varchar(30)
Prenom	varchar(30)
Adress#	int

Langage de manipulation des données - SELECT

Exemples

On désire lire les noms rangés par ordre alphabétique de toutes les personnes qui se prénomment Lisa.

```
SELECT Nom FROM Personne  
WHERE Prenom = 'Lisa' ORDER BY 1
```

On désire lire tous les noms et prénoms associés dans un champ séparés par un espace.

Personne	
<u>ID</u>	<u>int</u>
Nom	varchar(30)
Prenom	varchar(30)
Adress#	int

Langage de manipulation des données - SELECT

Exemples

On désire lire les noms rangés par ordre alphabétique de toutes les personnes qui se prénomment Lisa.

```
SELECT Nom FROM Personne  
WHERE Prenom = 'Lisa' ORDER BY 1
```

On désire lire tous les noms et prénoms associés dans un champ séparés par un espace.

```
SELECT concat(Nom, ' ', Prenom) as Gens  
FROM Personne ORDER BY 1
```

On désire lire les ID de toutes les personnes ayant une adresse renseignée.

Personne	
<u>ID</u>	<u>int</u>
Nom	varchar(30)
Prenom	varchar(30)
Adress#	int

Langage de manipulation des données - SELECT

Exemples

On désire lire les noms rangés par ordre alphabétique de toutes les personnes qui se prénomment Lisa.

```
SELECT Nom FROM Personne
WHERE Prenom = 'Lisa' ORDER BY 1
```

On désire lire tous les noms et prénoms associés dans un champ séparés par un espace.

```
SELECT concat(Nom, ' ', Prenom) as Gens
FROM Personne ORDER BY 1
```

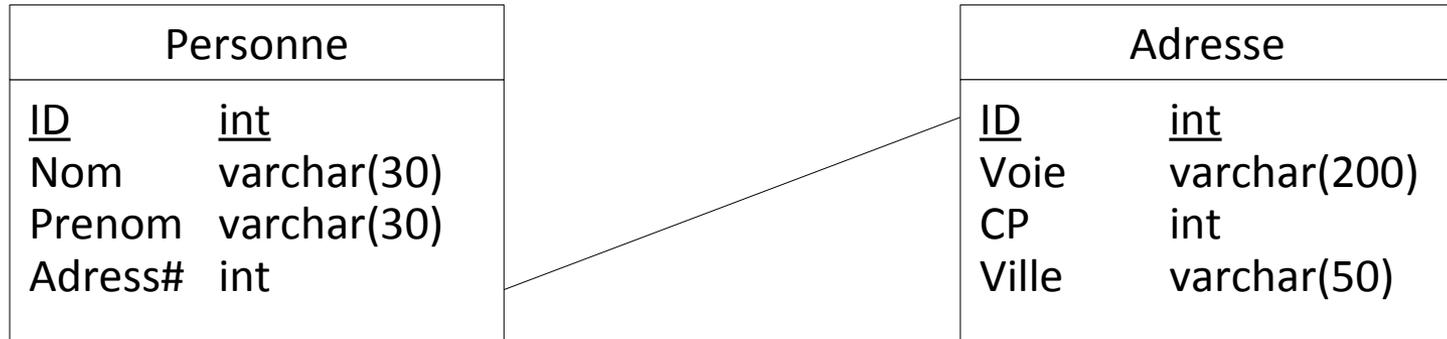
On désire lire les ID de toutes les personnes ayant une adresse renseignée.

```
SELECT ID FROM Personne
WHERE Adress IS NOT NULL
```

Personne	
<u>ID</u>	<u>int</u>
Nom	varchar(30)
Prenom	varchar(30)
Adress#	int

Langage de manipulation des données - SELECT

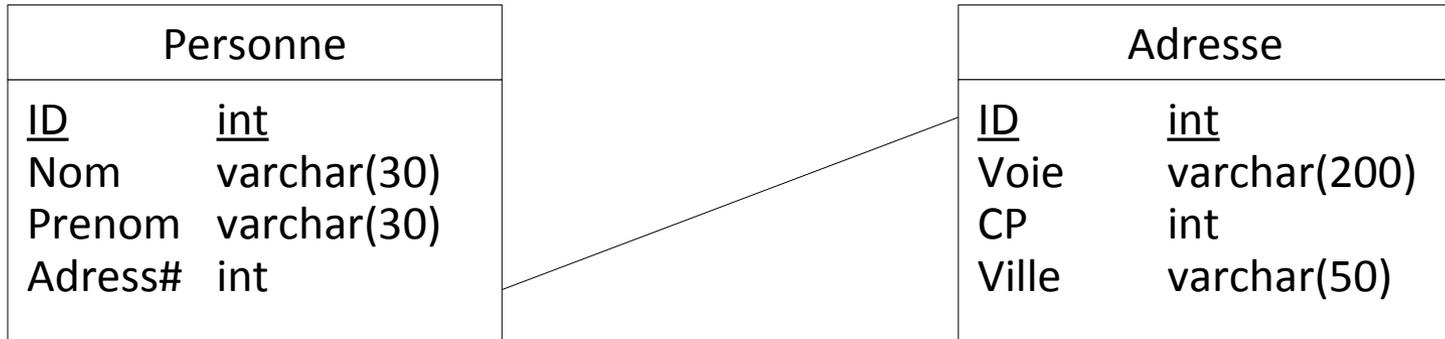
Exemple



Sélectionner le nom et l'adresse des personnes dont le nom commence par Simps :

Langage de manipulation des données - SELECT

Exemple



Sélectionner le nom et l'adresse des personnes dont le nom commence par Simps :

```
SELECT Personne.Nom, Adresse.Voie
FROM Personne, Adresse
WHERE Personne.Adress = Adresse.ID
AND Personne.Nom LIKE 'Simps%'
```

Langage de manipulation des données

Lien entre requêtes

Il est possible d'insérer dans une table des données issues d'une autre requête.

```
INSERT [INTO] tbl_name [(col_name, ...)]  
    SELECT ...
```

Il est possible de mettre à jour des données en fonction de données d'autres tables :

```
UPDATE tbl_name [, tbl_name ...]  
    SET col_name1=expr1 [, col_name2=expr2 ...]  
    [WHERE where_definition]
```

C'est toujours la table dont le nom est accolé au mot UPDATE qui est mise à jour.

Plan

- Résumé des épisodes précédents
- Langage de manipulation des données
- **SQL avancé : les jointures**
- SQL avancé : les groupements
- SQL avancé : les transactions
- SQL avancé : l'intégrité référentielle

Jointures

Utilisation des jointures

→ Sélectionner les données se trouvant dans plusieurs tables.

→ Préciser les données sur lesquelles travailler lors d'un :

- Select (lecture)
- Update (mise à jour)
- Delete (suppression)

Jointures

Utilisation des jointures

→ Sélectionner les données se trouvant dans plusieurs tables.

→ Préciser les données sur lesquelles travailler lors d'un :

- Select (lecture)
- Update (mise à jour)
- Delete (suppression)

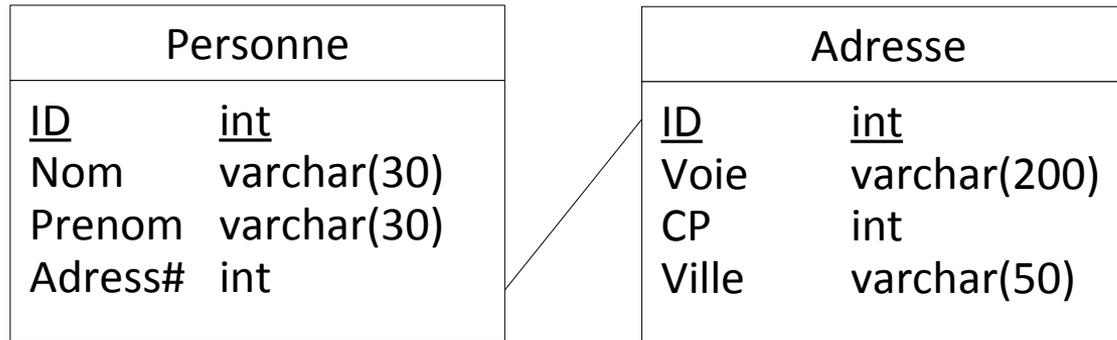
Principe

Une jointure a lieu **entre deux tables**. Elle exprime une correspondance entre deux clés par un **critère d'égalité**.

Si les données à traiter se trouvent **dans trois tables**, la correspondance entre les trois tables s'exprime par **deux égalités**.

Jointures

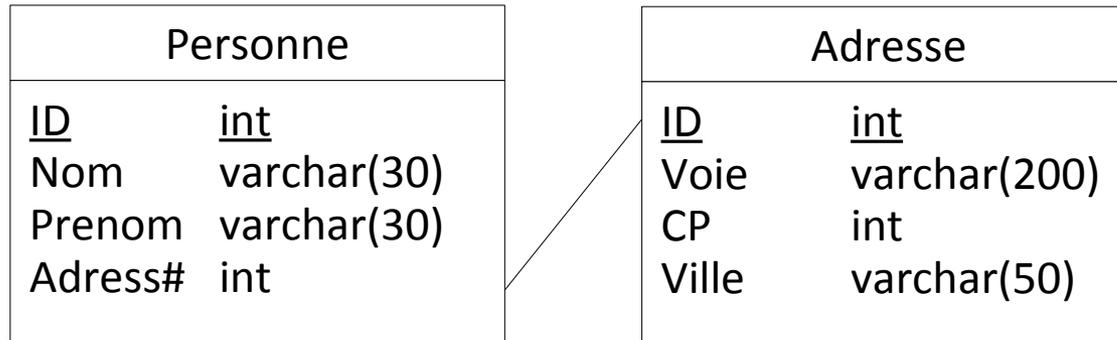
Exemple



Pour lire l'adresse correspondant à la personne, il faut écrire :

Jointures

Exemple

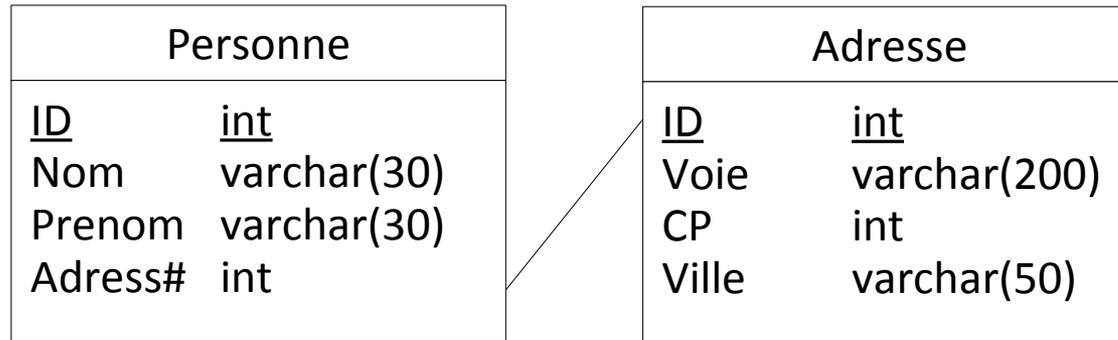


Pour lire l'adresse correspondant à la personne, il faut écrire :

```
SELECT * FROM Personne, Adresse  
WHERE Personne.Adress = Adresse.ID
```

Jointures

Exemple



Pour lire l'adresse correspondant à la personne, il faut écrire :

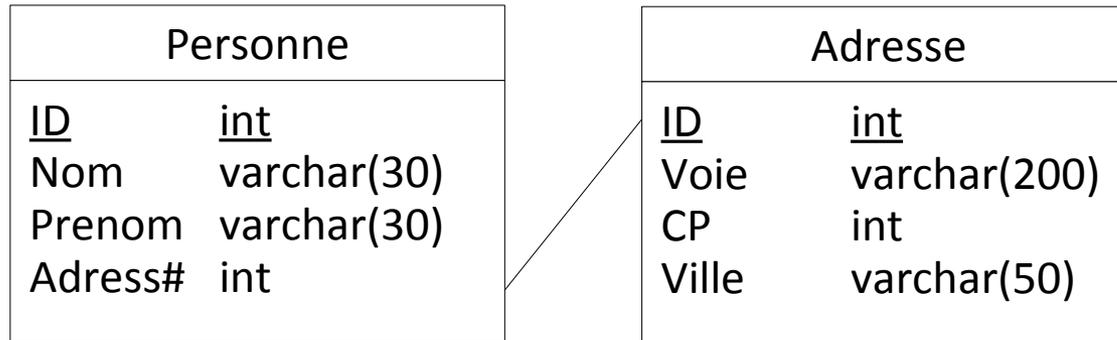
```
SELECT * FROM Personne, Adresse  
WHERE Personne.Adress = Adresse.ID
```

Attention : dans la clause WHERE se mélangent les associations entre les tables et les conditions de sélection des données. Ne pas les confondre !

```
SELECT * FROM Personne, Adresse  
WHERE Personne.Adress = Adresse.ID  
AND Personne.Nom = 'Durand'
```

Jointures

Exemple



Pour lire l'adresse correspondant à la personne, il faut écrire :

```
SELECT * FROM Personne, Adresse
WHERE Personne.Adress = Adresse.ID
```

Attention : dans la clause WHERE se mélangent les associations entre les tables et les conditions de sélection des données. Ne pas les confondre !

```
SELECT * FROM Personne, Adresse
WHERE Personne.Adress = Adresse.ID
AND Personne.Nom = 'Durand'
```

jointure pour associer les deux tables

critère de sélection

Remarque : ordre sans importance

Jointures fermées et ouvertes

Problèmes de la jointure par = :

- Mélange des critères de sélection et des jointures
- Mise en relation des données uniquement quand les deux attributs sont remplis (jointure **fermée**)

Jointures fermées et ouvertes

Problèmes de la jointure par = :

- Mélange des critères de sélection et des jointures
- Mise en relation des données uniquement quand les deux attributs sont remplis (jointure **fermée**)

Exemple :

On désire lire toutes les personnes et accessoirement donner leur adresse si celle-ci est connue.

La requête :

```
SELECT * FROM Personne, Adresse  
WHERE Personne.Adress = Adresse.ID
```

ne retournera pas les personnes n'ayant pas d'adresse référencée.

Jointures fermées et ouvertes

Jointure JOIN

L'association se fait directement entre les tables en précisant les colonnes concernées.

```
SELECT * FROM table1 INNER JOIN table2 ON  
table1.cle_primaire = table2.cle_etrangere
```

Exemple

```
SELECT * FROM Personne, Adresse  
WHERE Personne.Adress = Adresse.ID
```

équivalent à :

```
SELECT * FROM Personne  
INNER JOIN Adresse ON Personne.Adress = Adresse.ID
```

Jointures fermées et ouvertes

Il existe **trois types d'associations** :

- `INNER JOIN` : jointure fermée, les données doivent être à la fois dans les 2 tables
- `LEFT [OUTER] JOIN` : jointure ouverte, on lit les données de la table de gauche en y associant éventuellement celle de la table de droite.
- `RIGHT [OUTER] JOIN` : jointure ouverte, on lit les données de la table de droite en y associant éventuellement celle de la table de gauche.

Jointures fermées et ouvertes

Il existe **trois types d'associations** :

- `INNER JOIN` : jointure fermée, les données doivent être à la fois dans les 2 tables
- `LEFT [OUTER] JOIN` : jointure ouverte, on lit les données de la table de gauche en y associant éventuellement celle de la table de droite.
- `RIGHT [OUTER] JOIN` : jointure ouverte, on lit les données de la table de droite en y associant éventuellement celle de la table de gauche.

Exemple

Lire toutes les personnes et accessoirement donner leur adresse si celle-ci est connue.

Jointures fermées et ouvertes

Il existe **trois types d'associations** :

- `INNER JOIN` : jointure fermée, les données doivent être à la fois dans les 2 tables
- `LEFT [OUTER] JOIN` : jointure ouverte, on lit les données de la table de gauche en y associant éventuellement celle de la table de droite.
- `RIGHT [OUTER] JOIN` : jointure ouverte, on lit les données de la table de droite en y associant éventuellement celle de la table de gauche.

Exemple

Lire toutes les personnes et accessoirement donner leur adresse si celle-ci est connue.

```
SELECT * FROM Personne LEFT OUTER JOIN Adresse
```

Les personnes pour lesquelles l'adresse n'est pas connue auront les champs de la table Adresse à NULL.

Jointures fermées et ouvertes

Exemples avec critère

Lire toutes les personnes qui s'appellent Durand et accessoirement donner leur adresse si celle-ci est connue.

Jointures fermées et ouvertes

Exemples avec critère

Lire toutes les personnes qui s'appellent Durand et accessoirement donner leur adresse si celle-ci est connue.

```
SELECT * FROM Personne LEFT JOIN adresse
ON Personne.Adress=Adresse.ID
WHERE Personne.Nom = 'Durand'
```

Plan

- Résumé des épisodes précédents
- Langage de manipulation des données
- SQL avancé : les jointures
- **SQL avancé : les groupements**
- SQL avancé : les transactions
- SQL avancé : l'intégrité référentielle

Groupements

Utilisation des groupements

→ effectuer des opérations sur un ensemble de données :

- MIN (retourne le minimum)
- MAX (retourne le maximum)
- COUNT (retourne le nombre)
- SUM (retourne la somme)

Afin de préciser au moteur SQL que cette opération porte sur une sélection de données, il faut préciser la clause GROUP BY.

Groupements

Utilisation des groupements

→ effectuer des opérations sur un ensemble de données :

- MIN (retourne le minimum)
- MAX (retourne le maximum)
- COUNT (retourne le nombre)
- SUM (retourne la somme)

Afin de préciser au moteur SQL que cette opération porte sur une sélection de données, il faut préciser la clause GROUP BY.

Exemple

Compter le nombre de valeurs de champ1 :

```
SELECT champ2, COUNT(champ1) FROM table1  
GROUP BY champ2
```

Groupements

Pour créer des critères de sélection qui **portent sur un ensemble de données**,
→ associer la clause GROUP BY à la clause HAVING (ou NOT HAVING).

Exemple

Sélectionner les données dont le nombre de répétitions est supérieur à n .

```
SELECT * FROM table1 GROUP BY champ2  
HAVING count(champ2) > n
```

Plan

- Résumé des épisodes précédents
- Langage de manipulation des données
- SQL avancé : les jointures
- SQL avancé : les groupements
- **SQL avancé : les transactions**
- SQL avancé : l'intégrité référentielle

Transactions

En mode classique, les requêtes s'enchaînent. La première peut fonctionner alors que la suivante peut rencontrer une erreur. La base de données contient alors des données dans certaines tables et pas dans d'autres.

Transactions

En mode classique, les requêtes s'enchaînent. La première peut fonctionner alors que la suivante peut rencontrer une erreur. La base de données contient alors des données dans certaines tables et pas dans d'autres.

Pour éviter cela on utilise des **transactions** (blocs de requêtes SQL) :

`START TRANSACTION` (pour ouvrir la transaction)

[Liste de requêtes SQL]

`COMMIT TRANSACTION`

ou

`ROLLBACK TRANSACTION`

Confirme et exécute l'ensemble des requêtes SQL

Annule l'ensemble des requêtes SQL

Transactions

En mode classique, les requêtes s'enchaînent. La première peut fonctionner alors que la suivante peut rencontrer une erreur. La base de données contient alors des données dans certaines tables et pas dans d'autres.

Pour éviter cela on utilise des **transactions** (blocs de requêtes SQL) :

START TRANSACTION (pour ouvrir la transaction)

[Liste de requêtes SQL]

COMMIT TRANSACTION

ou

ROLLBACK TRANSACTION

Confirme et exécute l'ensemble des requêtes SQL

Annule l'ensemble des requêtes SQL

Verrou et dead-lock

Lorsqu'une modification sur une table est en cours, les données sont verrouillées en lecture et en écriture.

→ Situation de verrouillages mutuels entre deux transactions : **dead-locks**.

Repérés par le SGDB qui émet un rollback sur l'une des transactions.

Plan

- Résumé des épisodes précédents
- Langage de manipulation des données
- SQL avancé : les jointures
- SQL avancé : les groupements
- SQL avancé : les transactions
- SQL avancé : l'intégrité référentielle

Intégrité référentielle

Rappel :

Dans un modèle physique de données, les tables sont liées entre elles par le biais d'une clé étrangère.

La clé étrangère d'une table permet de lier la table à la clé primaire de l'autre table.

Lors de la sélection des données, l'association entre les deux tables est effectuée grâce à une jointure entre ces deux tables.

Personne

Id	Nom	Prenom	Id_adresse
1	Durand	Marie	4
2	Simpson	Bart	3
3	Dubois	Jean	NULL
4	Simpson	Lisa	3

Adresse

Id	Voie	CP	Ville
1	1, rue Ici	75002	Paris
2	12, rue labas	75015	Paris
3	742 Evergreen Terrasse		Springfield
4	Chemin perdu	66000	Perpignan

Chaque référence à une adresse est respectée.

Intégrité référentielle

Rappel :

Dans un modèle physique de données, les tables sont liées entre elles par le biais d'une clé étrangère.

La clé étrangère d'une table permet de lier la table à la clé primaire de l'autre table.

Lors de la sélection des données, l'association entre les deux tables est effectuée grâce à une jointure entre ces deux tables.

Personne

Id	Nom	Prenom	Id_adresse
1	Durand	Marie	4
2	Simpson	Bart	3
3	Dubois	Jean	NULL
4	Simpson	Lisa	3

Adresse

Id	Voie	CP	Ville
1	1, rue Ici	75002	Paris
2	12, rue labas	75015	Paris
3	742 Evergreen Terrasse		Springfield
4	Chemin perdu	66000	Perpignan

Si on supprime l'occurrence 4 de la table Adresse, il devient impossible de retrouver l'adresse de Marie Durand

Intégrité référentielle

Lors de la création des tables, il est possible de confier ce contrôle à la base de données.

Le fonctionnement désiré devra être précisé lors de la création des clés externes.

Note :

Sur MySQL, cette fonctionnalité n'est disponible qu'avec le moteur des tables : InnoDB.

Lors d'un create table, il faut ajouter une clause

```
FOREIGN KEY (champ1, [..., champN])  
REFERENCES table(champ1, [..., champN])  
ON UPDATE action  
ON DELETE action
```

Intégrité référentielle

Lors d'un create table, il faut ajouter une clause

```
FOREIGN KEY (champ1, [..., champN])
```

```
REFERENCES table(champ1, [..., champN])
```

```
ON UPDATE action
```

```
ON DELETE action
```

Les actions possibles sont :

- **RESTRICT** : si une référence est trouvée, la suppression ou la modification sera interdite.
- **SET NULL** : si une référence est trouvée, la suppression ou la modification aura pour effet en plus de l'action de mettre à jour la référence avec la valeur NULL.
- **CASCADE** : si une référence est trouvée, la suppression ou la modification aura pour effet en plus de l'action d'effectuer la même opération sur les données trouvées.
- **NO ACTION** : pas de contrôle d'intégrité référentielle.