

Notion d'ORM

Table des matières

| | |
|---|-----------|
| I. Symétrie modèle de données / schéma physique de données | 3 |
| II. Exercice : Quiz | 6 |
| III. Utilisation d'objets SQLAlchemy et Python | 7 |
| IV. Exercice : Quiz | 10 |
| V. Essentiel | 11 |
| VI. Auto-évaluation | 11 |
| A. Exercice | 11 |
| B. Test | 12 |
| Solutions des exercices | 13 |

I. Symétrie modèle de données / schéma physique de données

Contexte

Le but de l'Object Relational Mapping ou O.R.M est de faciliter la manipulation de données stockées dans un Système de Gestion de Base de Données Relationnelles (SGBD) au sein des langages de programmation-objet.

En effet, la plupart des applications nécessitent le stockage d'un grand nombre de données et ont donc recours aux bases de données. Cependant, les opérations les plus basiques d'ajout, lecture, mise à jour et suppression sont loin de la trivialité que souhaiterait le développeur.

Afin de répondre à ce besoin de facilitation, l'O.R.M offre une couche d'abstraction qui réalise la traduction des données extraites de la base de données vers un objet propre au langage de programmation. Le développeur travaille ainsi uniquement avec des objets sans se soucier du stockage sous-jacent des données.

Structurer une base de données avec SQL

Pour tirer parti de la puissance de SQL, vous devrez appliquer une certaine normalisation de la base de données aux données du fichier `author_book_publisher.csv`. Pour ce faire, vous allez séparer les auteurs, les livres et les éditeurs dans des tables de base de données distinctes.

Conceptuellement, les données sont stockées dans la base de données dans des structures de table bidimensionnelles. Chaque table se compose de lignes d'**enregistrements** et chaque enregistrement se compose de colonnes ou de **champs** contenant des données.

Les données contenues dans les champs sont de types prédéfinis, notamment du texte, des entiers, des flottants, etc. Les fichiers CSV sont différents, car tous les champs sont du texte et doivent être analysés par un programme pour qu'un type de données leur soit attribué.

Chaque enregistrement de la table a une **clé primaire** définie pour donner à un enregistrement un identifiant unique. La clé primaire est similaire à la clé d'un dictionnaire Python. Le moteur de base de données lui-même génère souvent la clé primaire sous la forme d'une valeur entière incrémentée pour chaque enregistrement inséré dans la table de base de données.

Bien que la clé primaire soit souvent générée automatiquement par le moteur de base de données, elle n'est pas obligée de l'être. Si les données stockées dans un champ sont uniques parmi toutes les autres données de la table de ce champ, il peut s'agir de la clé primaire. Par exemple, un tableau contenant des données sur les livres pourrait utiliser le SBB du livre comme clé primaire.

Méthode Création des tables avec SQL

Voici comment créer les trois tables représentant les auteurs, les livres et les éditeurs dans le fichier CSV à l'aide d'instructions SQL :

```
CREATE TABLE author (  
    author_id INTEGER NOT NULL PRIMARY KEY,  
    first_name VARCHAR,  
    last_name VARCHAR  
);  
  
CREATE TABLE book (  
    book_id INTEGER NOT NULL PRIMARY KEY,  
    author_id INTEGER REFERENCES author,  
    title VARCHAR  
);  
  
CREATE TABLE publisher (  
    publisher_id INTEGER NOT NULL PRIMARY KEY,  
    name VARCHAR  
);
```

Notez qu'il n'y a aucune opération sur les fichiers, aucune variable créée et aucune structure pour les contenir. Les instructions décrivent uniquement le résultat souhaité : la création d'une table avec des attributs particuliers. Le moteur de base de données détermine comment procéder. On peut par exemple utiliser le moteur de stockage InnoDB avec INNODB "CREATE TABLE t1 (a INT, b CHAR (20), PRIMARY KEY (a)) ENGINE=InnoDB;"

Une fois que vous avez créé et rempli cette table avec les données d'auteur du fichier author_book_publisher.csv, vous pouvez accéder à l'aide d'instructions SQL. L'instruction suivante (également appelée **requête**) utilise le caractère générique (*) pour obtenir toutes les données de la table author et les générer :

```
SELECT * FROM author;
```

Vous pouvez utiliser l'outil SQLite ³¹ pour interagir avec le fichier author_book_publisher.db de la base de données dans le répertoire project/data : `$ sqlite 3 author_book_publisher.db`

Une fois que l'outil de ligne de commande SQLite est en cours d'exécution avec la base de données ouverte, vous pouvez entrer des commandes SQL.

1 <https://sqlite.org/cli.html>

```
sqlite> SELECT * FROM author;  
1|Isaac|Asimov  
2|Pearl|Buck  
3|Tom|Clancy  
4|Stephen|King  
5|John|Le Carre  
6|Alex|Michaelides  
7|Carol|Shaben  
  
sqlite> .q
```

Le code ci-dessus montre une commande SQL et sa sortie, suivie de la commande .q pour quitter le programme:

Construire des relations

Les **relations** sont une autre caractéristique des systèmes de base de données que vous pourriez trouver encore plus puissante et utile que la persistance et la récupération des données. Les bases de données qui prennent en charge les relations vous permettent de diviser les données en plusieurs tables et d'établir des connexions entre elles.

Les données du fichier author_book_publisher.csv représentent les données et les relations en dupliquant les données. Une base de données gère cela en divisant les données en trois tables author-, book- et publisher- et en établissant des relations entre elles.

On distingue trois types de relations :

- **Relation une-à-une** : les **relations** sont une autre caractéristique des systèmes de base de données que vous pourriez trouver encore plus puissante et utile que la persistance et la récupération des données. Les bases de données qui prennent en charge les relations vous permettent de diviser les données en plusieurs tables et d'établir des connexions entre elles.
- **Relation une-à-plusieurs** : une relation **un-à-plusieurs** ressemble à celle d'un client qui commande des articles en ligne. Un client peut avoir plusieurs commandes, mais chaque commande appartient à un client. La base de données author_book_publisher.db a une relation un-à-plusieurs sous la forme d'auteurs et de livres. Chaque auteur peut écrire plusieurs livres, mais chaque livre est écrit par un auteur.z.
- **Relation plusieurs-à-plusieurs** : des relations plusieurs-à-plusieurs existent dans la base de données author_book_publisher.db entre les auteurs et les éditeurs ainsi qu'entre les livres et les éditeurs. Un auteur peut travailler avec de nombreux éditeurs et un éditeur peut travailler avec plusieurs auteurs. De même, un livre peut être publié par de nombreux éditeurs et un éditeur peut publier plusieurs livres. La gestion de cette situation dans la base de données est plus complexe qu'une relation un-à-plusieurs car la relation va dans les deux sens. Les relations plusieurs-à-plusieurs sont créées par une table d'association agissant comme un pont entre les deux tables liées.

Exercice : Quiz

[solution n°1 p.15]

Question 1

SELECT * FROM table ; signifie que :

- ☐ On sélectionne toutes les lignes de la table « table » (ici)
- ☐ On ne sélectionne aucune les lignes de la table « table » (ici)
- ☐ On sélectionne la première ligne de la table « table » (ici)
- ☐ On sélectionne la dernière ligne de la table « table » (ici)

Question 2

Quelle est la bonne manière de crée une table.

- ☐ CREATE TABLE nom_table(
Valeur 1 typeVal1,
Valeur 2 typeVal2,
Valeur 3 typeVal3
);
- ☐ CREATE TABLE(
Valeur 1 typeVal1,
Valeur 2 typeVal2,
Valeur 3 typeVal3
);
- ☐ CREATE TABLE nom_table{
Valeur 1 typeVal1,
Valeur 2 typeVal2,
Valeur 3 typeVal3
};
- ☐ CREATE TABLE nom_table(
Valeur 1 typeVal1,
Valeur 2 typeVal2,
Valeur 3 typeVal3,
);

Question 3

Quel est le type de relation qui n'est pas cité dans le cours ?

- ☐ Relation une à une
- ☐ Relations un-à-plusieurs
- ☐ Relation plusieurs-à-plusieurs
- ☐ Relation plusieurs-à-une

Question 4

Chaque enregistrement de la table a une clé primaire définie pour donner à un enregistrement un identifiant unique.

- ☐ Vrai
- ☐ Faux

Question 5

Lorsque l'on crée une table en SQL il est impératif de donner un type à chaque attribut.

- ☐ Vrai
- ☐ Faux

III. Utilisation d'objets SQLAlchemy et Python

Définition

SQLAlchemy est un puissant kit d'outils d'accès aux bases de données pour Python, son Mappeur Object Relationnel (ORM) étant l'un de ses composants les plus célèbres, et celui discuté et utilisé ici.

Lorsque vous travaillez dans un langage orienté objet comme Python, il est souvent utile de penser en termes d'objets. Il est possible de mapper les résultats renvoyés par les requêtes SQL sur des objets, mais cela va à l'encontre du fonctionnement de la base de données. S'en tenir aux résultats scalaires fournis par SQL va à l'encontre du fonctionnement des développeurs Python. Ce problème est connu sous le nom de discordance d'impédance objet relationnelle.

L'ORM fourni par SQLAlchemy se situe entre la base de données SQLite et votre programme Python et transforme le flux de données entre le moteur de base de données et les objets Python. SQLAlchemy vous permet de penser en termes d'objets tout en conservant les fonctionnalités puissantes d'un moteur de base de données.

Modèle et établissement de connexion

L'un des éléments fondamentaux pour permettre la connexion de SQLAlchemy à une base de données est la création d'un modèle. Le modèle est une classe Python définissant le mappage de données entre les objets Python renvoyés suite à une requête de base de données et les tables de base de données sous-jacentes.

Le diagramme entité-relation affiché précédemment montre des cases reliées par des flèches. Les boîtes sont les tables construites avec les commandes SQL et sont ce que les classes Python vont modéliser. Les flèches sont les relations entre les tables.

Les modèles sont des classes Python héritant d'une classe Base de SQLAlchemy. La classe Base fournit les opérations d'interface entre les instances du modèle et la table de base de données.

Exemple

Vous trouverez ci-dessous le fichier models.py qui crée les modèles pour représenter la base de données author_book_publisher.db :

```
1 from sqlalchemy import Column, Integer, String, ForeignKey, Table
2 from sqlalchemy.orm import relationship, backref
3 from sqlalchemy.ext.declarative import declarative_base
4
5 Base = declarative_base()
6
7 author_publisher = Table(
8     "author_publisher",
9     Base.metadata,
10    Column("author_id", Integer, ForeignKey("author.author_id")),
11    Column("publisher_id", Integer, ForeignKey("publisher.publisher_id")),
12 )
13
14 book_publisher = Table(
15     "book_publisher",
16     Base.metadata,
17    Column("book_id", Integer, ForeignKey("book.book_id")),
18    Column("publisher_id", Integer, ForeignKey("publisher.publisher_id")),
19 )
20
21 class Author(Base):
22     __tablename__ = "author"
23     author_id = Column(Integer, primary_key=True)
24     first_name = Column(String)
25     last_name = Column(String)
26     books = relationship("Book", backref=backref("author"))
27     publishers = relationship(
28         "Publisher", secondary=author_publisher, back_populates="authors"
29     )
30
31 class Book(Base):
32     __tablename__ = "book"
33     book_id = Column(Integer, primary_key=True)
34     author_id = Column(Integer, ForeignKey("author.author_id"))
35     title = Column(String)
36     publishers = relationship(
37         "Publisher", secondary=book_publisher, back_populates="books"
38     )
39
40 class Publisher(Base):
41     __tablename__ = "publisher"
42     publisher_id = Column(Integer, primary_key=True)
43     name = Column(String)
44     authors = relationship(
45         "Author", secondary=author_publisher, back_populates="publishers"
46     )
47     books = relationship(
48         "Book", secondary=book_publisher, back_populates="publishers"
49     )
```

Voici ce qui se passe dans ce module :

- **Ligne 1** importation des classes Column, Integer, String, ForeignKey et les Tables de SQLAlchemy, qui sont utilisés pour aider à définir les attributs du modèle.
- **La ligne 2** importe les objets relationship() et backref, qui sont utilisés pour créer les relations entre les objets.

- **La ligne 3** importe l'objet `declarative_base`, qui connecte le moteur de base de données à la fonctionnalité SQLAlchemy des modèles.
- **La ligne 5** crée la classe `Base`, dont tous les modèles héritent et comment ils obtiennent la fonctionnalité SQLAlchemy ORM.
- **Les lignes 7 à 12** créent le modèle de table d'association `author_publisher`.
- **Les lignes 14 à 19** créent le modèle de table d'association `book_publisher`.
- **Les lignes 21 à 29** définissent le modèle de classe `Author` dans la table de base de données `author`.
- **Les lignes 31 à 38** définissent le modèle de classe `Book` dans la table de base de données `book`.
- **Les lignes 40 à 49** définissent le modèle de classe `Publisher` dans la table de base de données `publisher`.

La description ci-dessus montre le mappage des cinq tables de la base de données.

`author_book_publisher.db`. Mais il passe sous silence certaines fonctionnalités dont l'ORM SQLAlchemy dispose, permettre compris `Table`, `ForeignKey`, `relationship()` et `backref`.

Définition Attribut

Table : la classe `Table` de SQLAlchemy crée une instance unique d'une table mappée ORM dans la base de données. Le premier paramètre est le nom de la table tel que défini dans la base de données, et le second est `Base.metadata`, qui fournit la connexion entre la fonctionnalité SQLAlchemy et le moteur de base de données.

ForeignKey : la classe `ForeignKey` de SQLAlchemy définit une dépendance entre deux champs `Column` dans des tables différentes. Une `ForeignKey` est la façon dont vous rendez SQLAlchemy conscient des relations entre les tables.

Relationship() : avoir un `ForeignKey` définit l'existence de la relation entre les tables, mais pas la collection de livres qu'un auteur peut avoir.

Exemple

```
books = relationship("Book", backref=backref("author"))
```

Le code ci-dessus définit une collection parent-enfant. L'attribut `books` étant au pluriel (ce qui n'est pas une exigence, juste une convention) est une indication qu'il s'agit d'une collection.

Le premier paramètre de `relationship()`, le nom de la classe `Book` (qui n'est pas le nom de la table `book`), est la classe à laquelle l'attribut `books` est lié. `Relationship` informe SQLAlchemy qu'il existe une relation entre les classes `Author` et `Book`. SQLAlchemy trouvera la relation dans la définition `Book` de classe :

```
author_id = Column(Integer, ForeignKey("author.author_id"))
```

Définition

Backref : le paramètre `backref` de la collection `books` de `relationship()` crée un attribut `author` pour chaque instance `Book`. Cet attribut fait référence au parent `Author` auquel l'instance `Book` est liée.

Exemple

Par exemple, si vous exécutez le code Python suivant, une instance Book sera renvoyée à partir de la requête SQLAlchemy. L'instanceBook possède des attributs qui peuvent être utilisés pour imprimer des informations sur le livre :

```
book = session.query(Book).filter_by(Book.title == "The Stand").one_or_none()
print(f"Authors name: {book.author.first_name} {book.author.last_name}")
```

L'existence de l'attribut author dans Book ci-dessus est due à la définition du backref. Un backref peut être très pratique lorsque vous avez besoin de faire référence au parent et que vous possédez instance enfant.

Exercice : Quiz

[solution n°2 p.16]

Question 1

L'un des éléments fondamentaux pour permettre la connexion de SQLAlchemy à une base de données est la création d'un modèle.

- ☐ Vrai
- ☐ Faux

Question 2

La ForeignKey peut faire référence :

- ☐ À une autre ForeignKey
- ☐ À un attribut de la même table
- ☐ À un attribut d'une autre table
- ☐ À une clé primaire d'une autre table

Question 3

Que prend en charge SQLAlchemy ?

- ☐ CREATE TABLE
- ☐ ALTER TABLE
- ☐ CREATE VIEW

Question 4

SQLAlchemy est un ORM

- ☐ Vrai
- ☐ Faux

Question

Sélectionnez les commandes avec Column qui sont valides.

- ☐ Column(Integer, primary_key = TRUE)
- ☐ Column(Integer)
- ☐ Column(String)
- ☐ Column(primary_key = False)
- ☐ Column(primary_key = True, String)

V. Essentiel

Pour résumer, le but de SQLAlchemy (et de manière plus générale d'un ORM) est de permettre à des développeurs d'interagir avec des moteurs de base de données directement dans le langage qu'ils utilisent pour développer leur application, site internet, etc. Grâce aux ORM il n'y pas (ou peu) besoin de faire des requêtes directement en SQL, c'est le rôle des ORM de produire les requêtes en fonction de comment ils sont utilisés par le développeur.

VI. Auto-évaluation

A. Exercice

Question 1

[solution n°3 p.17]

Créez une table (grâce à create table) Auteur avec comme attribut nom_auteur qui permet d'identifier n'importe quel auteur. L'auteur a aussi un nom, prénom, année de naissance.

```
1 CREATE TABLE Auteur (  
2   nom_auteur VARCHAR(25) PRIMARY KEY,  
3   nom VARCHAR(25),  
4   prenom VARCHAR(25),  
5   annee_de_naissance DATE  
6 ) ;
```

Question 2

[solution n°4 p.18]

Avec la table ci-dessous (la table s'appelle Auteur), donnez la commande qui permet de sélectionner le prénom des auteurs féminins.

| 1 | Nom | Prénom | Genre | Année de naissance |
|----|-------------|------------|-------|--------------------|
| 2 | Demers | Marthe | F | 09 décembre 1987 |
| 3 | Bernier | Courtland | H | 26 octobre 1980 |
| 4 | Bourssa | Normand | H | 13 juillet 1988 |
| 5 | Jalbert | Fantina | F | 07 septembre 1962 |
| 6 | Migneault | Granville | H | 05 juillet 1981 |
| 7 | Drouin | Maurice | H | 27 août 1987 |
| 8 | Vaillancour | Honoré | H | 27 février 1984 |
| 9 | Deshênes | Agate | F | 27 juillet 1980 |
| 10 | Gendron | Alexandrie | F | 26 octobre 1975 |

B. Test

Exercice 1 : Quiz

[solution n°5 p.18]

Question 1

Lorsque le nom d'un attribut commence et / ou finit par id ce dernier devient directement une PRIMARY KEY.

- ☐ Vrai
- ☐ Faux

Question 2

Qu'est-ce qui peut apparaître plusieurs fois dans une table :

- ☐ Une PRIMARY KEY
- ☐ Un attribut
- ☐ Une FOREIGN KEY

Question 3

Quelle est la bonne manière de créer une table :

- ☐ CREATE TABLE nom_table(
Valeur 1 typeVal1,
Valeur 2 typeVal2,
Valeur 3 typeVal3
);
- ☐ CREATE TABLE(
Valeur 1 typeVal1,
Valeur 2 typeVal2,
Valeur 3 typeVal3
);
- ☐ CREATE TABLE nom_table{
Valeur 1 typeVal1,
Valeur 2 typeVal2,
Valeur 3 typeVal3
};
- ☐ CREATE TABLE nom_table(
Valeur 1 typeVal1,
Valeur 2 typeVal2,
Valeur 3 typeVal3,
);

Question 4

Il est possible de créer une table sans nom.

- ☐ Vrai
- ☐ Faux

Question 5


Lorsque SELECT est suivi d'une « * » cela signifie que :

- ☐ L'on sélectionne tout d'une table
- ☐ L'on ne sélectionne rien de la table
- ☐ L'on sélectionne uniquement le premier élément de la table
- ☐ L'on sélectionne uniquement le dernier élément de la table

Solutions des exercices


Exercice p. 6 Solution n°1**Question 1**

SELECT * FROM table ; signifie que :

- ☒ On sélectionne toutes les lignes de la table « table » (ici)
 - ☐ On ne sélectionne aucune les lignes de la table « table » (ici)
 - ☐ On sélectionne la première ligne de la table « table » (ici)
 - ☐ On sélectionne la dernière ligne de la table « table » (ici)
-  Entre les mots clés SELECT et FORM, on doit préciser les champs à récupérer. L'étoile signifie que l'on sélectionne tous les champs d'une table. Si l'on souhaite voir apparaît un ou plusieurs champs particuliers il faut remplacer l'étoile par une liste de nom de champs


Question 2

Quelle est la bonne manière de crée une table.

- ☒ CREATE TABLE nom_table(
Valeur 1 typeVal1,
Valeur 2 typeVal2,
Valeur 3 typeVal3
);
 - ☐ CREATE TABLE(
Valeur 1 typeVal1,
Valeur 2 typeVal2,
Valeur 3 typeVal3
);
 - ☐ CREATE TABLE nom_table{
Valeur 1 typeVal1,
Valeur 2 typeVal2,
Valeur 3 typeVal3
};
 - ☐ CREATE TABLE nom_table(
Valeur 1 typeVal1,
Valeur 2 typeVal2,
Valeur 3 typeVal3,
);
-  La bonne manière est la 1, car à la différence de la 4 on ne met pas de virgule sur la dernière valeur, car cela provoque une erreur


Question 3

Quel est le type de relation qui n'est pas cité dans le cours ?

- ☐ Relation une à une
- ☐ Relations un-à-plusieurs
- ☐ Relation plusieurs-à-plusieurs
- ☒ Relation plusieurs-à-une
-  Uniquement la relation plusieurs-à-une n'est pas citée dans le cours


Question 4

Chaque enregistrement de la table a une clé primaire définie pour donner à un enregistrement un identifiant unique.

- ☒ Vrai
- ☐ Faux
-  Comme dit plus haut, c'est un fonctionnement interne des moteurs de base de données, chaque élément d'une table reçoit une clé primaire pour qu'il puisse être différencié d'autres éléments qui auraient les mêmes champs.

Question 5


Lorsque l'on crée une table en SQL il est impératif de donner un type à chaque attribut.

- ☒ Vrai
- ☐ Faux
-  Comme dans d'autres langages de programmation en SQL, il est important de préciser le type de chaque attribut. Sans cette précision, SQL ne saurait pas comment optimiser et gérer son espace mémoire.

Exercice p. 10 Solution n°2

Question 1


L'un des éléments fondamentaux pour permettre la connexion de SQLAlchemy à une base de données est la création d'un modèle.

- ☒ Vrai
- ☐ Faux
-  L'utilisation de SQLAlchemy, ou, de façon plus générale d'un ORM, permet d'interagir avec une base de données via un langage objet. Sans modèle, il n'y a rien sur quoi baser les interactions.

Question 2

La ForeignKey peut faire référence :


- ☐ À une autre ForeignKey
- ☐ À un attribut de la même table
- ☒ À un attribut d'une autre table
- ☒ À une clé primaire d'une autre table

-  La ForeignKey fait référence à un attribut d'une AUTRE table elle peut être une primaryKey, mais pas une autre ForeignKey.

Question 3

Que prend en charge SQLAlchemy ?


- ☒ CREATE TABLE
- ☐ ALTER TABLE
- ☐ CREATE VIEW

-  SQLAlchemy ne prend pas en charge ALTER TABLE et CREATE VIEW

Question 4

SQLAlchemy est un ORM


- ☒ Vrai
- ☐ Faux

-  La fonction principale de SQLAlchemy est de permettre la gestion d'une base de données via python. C'est un package d'ORM en python.

Question

Sélectionnez les commandes avec Column qui sont valides.

- ☒ Column(Integer, primary_key = TRUE)
- ☒ Column(Integer)
- ☒ Column(String)
- ☐ Column(primary_key = False)
- ☐ Column(primary_key = True, String)

-  Seules les trois premières commandes sont justes, car elles ont le bon nombre d'arguments et dans le bon ordre.

p. 11 Solution n°3

Avec ce code, on demande la création d'une table nommée « Auteur » contenant les champs suivants :

- « *nom_auteur* » qui sera une chaîne de caractère permettant d'identifier et de dissocier les différents éléments de notre table
- « *nom* » et « *prenom* » qui seront des chaînes de caractère
- « *annee_de_naissance* » qui sera une date

p. 12 Solution n°4


SELECT prénom FROM Auteur WHERE Genre='F' ;

La requête précédente demande au moteur de récupérer tous les prénoms des éléments dans la table Auteur ayant pour valeur « F » dans la colonne Genre.

Exercice p. 12 Solution n°5


Question 1

Lorsque le nom d'un attribut commence et / ou finit par id ce dernier devient directement une PRIMARY KEY.

- ☐ Vrai
- ☒ Faux
-  Il faut rajouter PRIMARY KEY à côté de la clé primaire.

Question 2

Qu'est-ce qui peut apparaître plusieurs fois dans une table :


- ☐ Une PRIMARY KEY
- ☒ Un attribut
- ☒ Une FOREIGN KEY
-  On peut en effet retrouver plusieurs attributs, plusieurs FOREIGN KEY, mais pas plusieurs PRIMARY KEY.

Question 3

Quelle est la bonne manière de créer une table :

- ☒ CREATE TABLE nom_table(
Valeur 1 typeVal1,
Valeur 2 typeVal2,
Valeur 3 typeVal3
);
- ☐ CREATE TABLE(
Valeur 1 typeVal1,
Valeur 2 typeVal2,
Valeur 3 typeVal3
);
- ☐ CREATE TABLE nom_table{
Valeur 1 typeVal1,
Valeur 2 typeVal2,
Valeur 3 typeVal3
};

☐ CREATE TABLE nom_table(
Valeur 1 typeVal1,
Valeur 2 typeVal2,
Valeur 3 typeVal3,
);


 La bonne manière est la 1, car à la différence de la 4 on ne met pas de virgule sur la dernière valeur, car cela provoque une erreur

Question 4

Il est possible de créer une table sans nom.

☐ Vrai

☒ Faux

 Créer une table sans nom est impossible, car cela crée une erreur.

Question 5


Lorsque SELECT est suivi d'une « * » cela signifie que :

☒ L'on sélectionne tout d'une table

☐ L'on ne sélectionne rien de la table

☐ L'on sélectionne uniquement le premier élément de la table

☐ L'on sélectionne uniquement le dernier élément de la table

 Entre les mots clés SELECT et FROM, on doit préciser les champs à récupérer. L'étoile signifie que l'on sélectionne tous les champs d'une table. Si l'on souhaite voir apparaître un ou plusieurs champs particuliers il faut remplacer l'étoile par une liste de nom de champs.