# Exercice

*Cet exercice porte sur les bases de données, la programmation de base en Python et l'algorithmique.*

La compagnie de danse *L'air de l'art* fait des présentations de spectacle. Les danseurs et danseuses d'un spectacle se déplacent donc dans une camionnette avec le matériel nécessaire.

Pour gérer la logistique, Jessica, la secrétaire de la compagnie, a mis en place une base de données.

## Partie A

La base de données permet à Jessica de relier les danseurs aux différentes chorégraphies et représentations. Pour simplifier, seules les villes seront mentionnées dans les adresses.

La base de données se schématise comme suit :

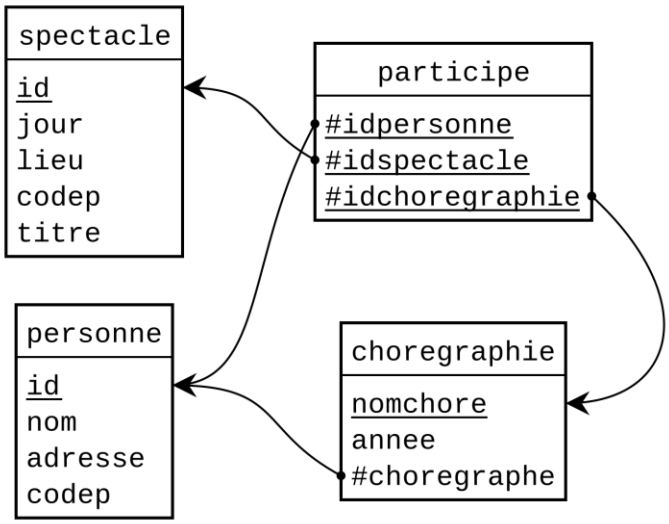



Figure 1. Schéma relationnel de la base

- La table `personne` est composée d'un identifiant `id`, d'un `nom`, d'une `adresse` et du code postal associé `codep`.
- La table `spectacle` est composée d'un identifiant `id`, d'un `jour` de représentation (au format `AAAA-MM-JJ`), d'une adresse du `lieu`, de son code postal associé `codep` et du titre de spectacle `titre`.
- La table `choregraphie` est composée d'un `nomchore` unique, d'une `annee` de création et de l'identifiant de son `choregraphe`.

- La table `participe` associe une personne qui danse `idpersonne`, un spectacle `idspectacle` et une chorégraphie `idchoregraphie`. La clef primaire est le triplet `idpersonne`, `idspectacle` et `idchoregraphie`.
- Les clefs primaires sont soulignées et les clefs étrangères sont précédées du symbole «#».

On pourra utiliser les mots clefs du langage SQL suivants : `SELECT, DISTINCT, FROM, WHERE, JOIN … ON, UPDATE … SET, DELETE, INSERT INTO … VALUES`.

La fonction d'agrégation `COUNT()` permet de compter le nombre d'enregistrements dans une table. Par exemple, la requête SQL ci-dessous permet de connaitre le nombre de lignes dans la table `ma_table` :

```
SELECT COUNT(*) FROM ma_table;
```

Un code postal est composé de cinq chiffres.

1. Écrire une requête qui ajoute la chorégraphie 'Tout autour' de Rachid Ouramdane (`id` n°9) datant de 2015 à la table `choregraphie`.
2. Écrire une requête qui affiche le nombre de chorégraphies de 1983.
3. Décrire par une phrase le résultat obtenu lors de l'exécution de la requête ci-dessous :

```
SELECT choregraphie.nomchore, personne.nom
FROM choregraphie
JOIN personne
ON choregraphie.choregraphe = personne.id;
```

4. Écrire une requête qui affiche le nom des danseurs ayant dansé dans le spectacle ayant pour titre 'Tout autour'.
5. Préciser les précautions à prendre pour supprimer un spectacle de la table `spectacle`. On ne demande pas d'écrire la requête.

## Partie B

Jessica a en charge d'amener les costumes et accessoires pour les représentations. Afin de minimiser les allers-retours, elle essaie d'optimiser le chargement de son véhicule à l'aide d'un programme.

Pour chaque représentation, elle a récupéré les informations utiles sous forme d'un dictionnaire de dictionnaires. Les sous-dictionnaires contiennent le nombre d'éléments nécessaires, le volume unitaire ainsi que le poids unitaire. Par exemple :

```
dmateriel = {
    'costume_A': {'nb': 3, 'volume':2, 'poids':1},
    'costume_B': {'nb': 1, 'volume':6, 'poids':2},
```

```
    'chaise_bois': {'nb': 40, 'volume':50, 'poids':4},
    'chapeau_A': {'nb': 2, 'volume':3, 'poids':1},
    'chapeau_B': {'nb': 4, 'volume':7, 'poids':1},
}
```

6. Recopier et compléter la ligne 3 du code ci-dessous afin que `n` soit égal au nombre total d'éléments nécessaires dans le dictionnaire `dmateriel`.

   Avec l'exemple précédent, à la fin de l'exécution du script, `n` doit valoir : $3 + 1 + 40 + 2 + 4 = 50$.

```
1 n = 0
2 for clef in dmateriel:
3     n += dmateriel[...][...]
```

Afin d'obtenir deux tableaux (de type `list`), l'un avec les clefs, l'autre avec les volumes associés, Jessica a écrit la fonction suivante :

```
1 def tab_clefs_tab_volumes(dmatos):
2     tab_clefs = []
3     tab_volumes = []
4     for clef, dico in dmatos.items():
5         tab_clefs.append(...)
6         tab_volumes.append(...)
7     return tab_clefs, tab_volumes
```

L'appel `tab_clefs_tab_volumes(dmateriel)` devra renvoyer le tuple :

```
(['costume_A', 'costume_B', 'chaise_bois', 'chapeau_A',
'chapeau_B'], [2, 6, 50, 3, 7])
```

7. Recopier et compléter les lignes 5 et 6 pour rendre effective cette fonction.

Pour optimiser le volume chargé dans son véhicule, Jessica se dit qu'une bonne stratégie serait de remplir son véhicule du matériel le plus volumineux en premier puis de finir de le remplir par du petit matériel.

8. Préciser à quel type d'algorithme vu en cours cette stratégie fait référence.

Pour le moment, on suppose que la fonction `clefs_triees_selon_volume` prend en paramètre un dictionnaire `dmatos` contenant le matériel et renvoie la liste des clefs de `dmatos` triée dans l'ordre croissant des volumes unitaires associés à chacune des clefs. Ainsi on a :

```
>>> clefs_triees_selon_volume(dmateriel)
['costume_A', 'chapeau_A', 'costume_B', 'chapeau_B',
'chaise_bois']
```

9. Citer un algorithme de tri vu en cours et donner un ordre de grandeur de son coût en temps d'exécution en fonction de la taille `n` de la liste à trier.

Jessica a commencé à coder une fonction `remplissage`, qui prend en paramètres un dictionnaire `dmatos` contenant le matériel, un entier `vmax`, qui renvoie la liste des éléments à charger dans une voiture sans dépasser le volume maximal `vmax` et qui modifie `dmatos` en mettant à jour le nombre d'éléments restants.

Par exemple :

```
>>> remplissage(dmateriel, 160)
['chaise_bois', 'chaise_bois', 'chaise_bois', 'chapeau_B',
'chapeau_A']
>>> print(dmateriel)
{
    'costume_A': {'nb': 3, 'volume':2, 'poids':1},
    'costume_B': {'nb': 1, 'volume':6, 'poids':2},
    'chaise_bois': {'nb': 37, 'volume':50, 'poids':4},
    'chapeau_A': {'nb': 1, 'volume':3, 'poids':1},
    'chapeau_B': {'nb': 3, 'volume':7, 'poids':1},
}
```

10. Recopier et compléter les lignes 6 et de 11 à 15 de la fonction `remplissage` ci-dessous :

```
 1 def remplissage(dmatos, vmax):
 2     vol = 0
 3     clefs_triees = clefs_triees_selon_volume(dmatos)
 4     i = len(clefs_triees)-1
 5     elements = []
 6     while i >= ... and vol < ...:
 7         clef = clefs_triees[i]
 8         if dmatos[clef]['nb'] == 0:
 9             i -= 1
10         elif vol + dmatos[clef]['volume'] <= vmax:
11             vol += ...
12             elements.append(...)
13             dmatos[clef]['nb'] -= 1
14         else:
15             ...
16     return elements
```

On remarquera que la fonction `remplissage` renvoie une liste vide s'il n'y a plus de matériel à charger dans `dmatos` (les valeurs associées à `'nb'` sont toutes égales à 0).

11. Écrire une fonction `nb_voitures` qui prend en paramètres un dictionnaire `dmatos` contenant le matériel, un entier `vmax` et renvoie le nombre de voitures nécessaires pour charger l'ensemble du matériel (on supposera que toutes les voitures ont la même capacité `vmax`). On pourra faire appel à la fonction remplissage, même si le code n'a pas été complété.