# Ejercicios de Repaso

May 8, 2024

## 1 Ejercicios de Repaso BDII

Resuelve los ejercicios propuestos basándote en la BBDD que se indica a continuación. - Colección Juegos:


```
[
 {
     "_id":1,
     "titulo": "The legend of Zelda: Breath of the Wild",
     "desarrollador_id": 101,
     "año": 2017,
     "plataforma": ["Switch", "Wii U"]
 },
 {
     "_id": 2,
     "titulo": "Super Mario Odyssey",
     "desarrollador_id": 101,
     "año": 2017,
     "plataforma": ["Switch"]
 },
  {
     "_id": 3,
     "titulo": "God of War",
     "desarrollador_id": 102,
     "año": 2018,
     "plataforma": ["PS4"]
 },
]
```


- Colección Desarrolladores:


```
[
 {
     "_id": 101,
     "nombre": "Nintendo",
     "pais": "Japón"
 },
 {
     "_id": 102,
```

```
    "nombre": "Santa Monica Studio",
    "pais": "Estados Unidos"
}
]
```

1. Escribe una consulta que obtenga una lista de todos los juegos junto con la información del desarrollador de cada juego.

```
[ ]: # Inicia una operación de agregación en la colección 'juegos'
resultados = db.juegos.aggregate([
    {
        # La operación '$lookup' realiza un join con la colección␣
 ↪'desarrolladores'
        # en base al campo 'desarrollador_id' en 'juegos' y el campo '_id' en␣
 ↪'desarrolladores'
        "$lookup": {
            "from": "desarrolladores",  # La colección con la que hacer el join
            "localField": "desarrollador_id",  # El campo de la colección␣
 ↪'juegos'
            "foreignField": "_id",  # El campo de la colección 'desarrolladores'
            "as": "desarrollador_info"  # El nombre del nuevo campo que␣
 ↪contendrá la información unida
        },

        # La operación '$project' especifica los campos a incluir en los␣
 ↪documentos de salida
        "$project": {
            "_id": 1,  # Incluir el campo '_id'
            "titulo": 1,  # Incluir el campo 'titulo'
            "desarrollador_id": 1,  # Incluir el campo 'desarrollador_id'
            "año": 1,  # Incluir el campo 'año'
            "plataforma": 1,  # Incluir el campo 'plataforma'
        }
    }
])

# Itera sobre los resultados de la operación de agregación e imprime cada␣
 ↪documento
for resultado in resultados:
    print(resultado)
```

2. Escribe una consulta que obtenga el número total de juegos desarrollados por “Nintendo” para la plataforma “Switch”.

```
[ ]: # Este es el inicio de la consulta de agregación en la colección 'juegos'.
db.juegos.aggregate([
```

```
  # La etapa $match filtra los documentos. En este caso, solo los documentos␣
↪donde el campo 'plataforma' es igual a 'Switch' pasan al siguiente paso.
  {
      "$match": {"plataforma": "Switch"}
  },

  # La etapa $lookup realiza una operación de unión con otra colección. En␣
↪este caso, se une la colección 'juegos' con la colección 'desarrolladores'␣
↪en los campos 'desarrollador_id' y '_id'.
  {
      "$lookup":
      {
          from: "desarrolladores",
          localField: "desarrollador_id",
          foreignField: "_id",
          as: "info_desarrollador"
      }
  },

  # La etapa $unwind descompone los campos de array en documentos separados␣
↪para cada elemento. Aquí, se descompone el campo 'info_desarrollador'.
  {"$unwind": "$info_desarrollador" },

  # Otro filtro $match. En este caso, solo los documentos donde el campo␣
↪'info_desarrollador.nombre' es igual a 'Nintendo' pasan al siguiente paso.
  {
      "$match": { "info_desarrollador.nombre": "Nintendo"}
  },

  # La etapa $group agrupa los documentos por algún campo especificado. Aquí,␣
↪los documentos se agrupan por 'info_desarrollador.nombre' y se cuenta el␣
↪número de juegos para cada desarrollador.
  {
      "$group":
          {
              _id: "$info_desarrollador.nombre",
              total_juegos: {"$sum": 1}
          }
  },

  # La etapa $project especifica los campos a incluir en los documentos de␣
↪salida. Aquí, se incluyen los campos 'desarrollador' y 'total_juegos', y se␣
↪excluye el campo '_id'.
  {
      "$project":
          {
```

```
                _id: 0,
                desarrollador: "$_id",
                total_juegos: 1
            }
    }
])
```

### 1.1 Ejercicios de Redis

Para gestionar una clínica veterinaria se dispone de un programa Python que utiliza una BBDD Redis. A continuación se piden una serie de funciones para completar la funcionalidad de dicha aplicación.

La conexión a la base de datos se realiza de la siguiente forma:

```
from happybase import Connection

# Conectar a HBase
conexion = Connection('localhost')

def crear_table_estudiantes(conexion):
    # Crear la tabla 'estudiantes' con la familia de columnas 'datos'
    conexion.create_table(
        'estudiantes',
        {'datos': dict(max_versions=1)}
    )
```

1. Completa las funciones Python que se muestran a continuación, la columna de cursos contiene un string separado por comas.

[ ]:

### 1.2 Ejercicios Neo4j

Disponemos de una BBDD que gestiona una pequeña librería donde se almacenan libros y sus autores que ha sido creada de la siguiente forma:

```
CREATE  (a:Libro {titulo: 'Libro A', autores: 'Autor1, Autor2'}),
        (b:Libro {titulo: 'Libro B', autores: 'Autor2, Autor3'}),
        (c:Libro {titulo: 'Libro C', autores: 'Autor1, Autor3'}),
        (d:Libro {titulo: 'Libro D', autores: 'Autor4, Autor5'}),
        (e:Libro {titulo: 'Libro E', autores: 'Autor5, Autor6'}),
```

[ ]:

1. Crea la consulta para la BBDD Neo4j necesaria para crear los nodos autores y conecte cada autor con los libros que ha escrito.

[ ]:

2. Escribe una consulta que encuentre a los autores que han escrito al menos dos libros y los ordene en orden descendente por el número de libros.