Q.1/ b.

**1. Woman**

| **Column Name** | **Data Type** | **Key Type** |
| --- | --- | --- |
| woman\_id | INT | **PK** |
| name | TEXT |  |
| birth\_date | DATE |  |
| blood\_type | TEXT |  |
| attending\_hospital\_id | INT | **FK → Hospital(hospital\_id)** |

**2. Hospital**

| **Column Name** | **Data Type** | **Key Type** |
| --- | --- | --- |
| hospital\_id | INT | **PK** |
| name | TEXT |  |
| location | TEXT |  |

**3. Professional\_Health\_Analyst**

| **Column Name** | **Data Type** | **Key Type** |
| --- | --- | --- |
| analyst\_id | INT | **PK** |
| assigned\_hospital\_id | INT | **FK → Hospital(hospital\_id)** |

**4. Pregnancy**

| **Column Name** | **Data Type** | **Key Type** |
| --- | --- | --- |
| pregnancy\_id | INT | **PK** |
| woman\_id | INT | **FK → Woman(woman\_id)** |
| baby's\_gender | TEXT |  |
| delivery\_date | DATE |  |
| date\_of\_first\_rejection | DATE |  |
| delivery\_type | TEXT |  |
| number\_of\_checkups | INT |  |
| number\_of\_missed\_checkups | INT |  |
| followed\_by\_hospital\_id | INT | **FK → Hospital(hospital\_id)** |

**5. Checkup**

| **Column Name** | **Data Type** | **Key Type** |
| --- | --- | --- |
| checkup\_id | INT | **PK** |
| pregnancy\_id | INT | **FK → Pregnancy(pregnancy\_id)** |
| analyst\_id | INT | **FK → Professional\_Health\_Analyst(analyst\_id)** |
| hospital\_id | INT | **FK → Hospital(hospital\_id)** |
| username | TEXT |  |
| password | TEXT |  |
| maternal\_mental\_health | TEXT |  |
| weight | DECIMAL |  |
| gestational\_age | INT |  |
| anomaly\_present | BOOLEAN |  |
| fetal\_heart\_rate | INT |  |
| blood\_type | TEXT |  |

Q.9/ Hubert doesn’t appear in the table. When we do:

|  |
| --- |
| cursor.execute("INSERT ...") |

it only changes the temporary transaction — nothing is saved to disk until we call conn.commit().

When your test closes the connection via:

|  |
| --- |
| close\_db\_connexion(cursor, conn) |

without committing, the insertion is rolled back (discarded). Therefore, we don’t see Hubert when listing analysts later.

Q.10 / We get that error:

|  |
| --- |
| An integrity error occurred while insert the analyst: UNIQUE constraint failed: Analyst.username  Impossible to add Hubert ...  analysts in database: [{'username': 'hubert'}] |

SQLite is telling us that it cannot insert a new row into table Analyst, because there’s already a row with the same value of username = 'hubert'.

In other words: it tells us : « You already inserted Hubert before, and since his username is defined as UNIQUE, the database refuses to insert him again. »

Q.12 / When we try to update the password of an analyst who is not in the database, the SQL UPDATE query executes but affects no rows, and our code correctly detects that and returns False — no error, just no change.

Q.14/ We get the following result:

|  |
| --- |
| \* Serving Flask app 'app'  \* Debug mode: off  WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.  \* Running on http://127.0.0.1:5000  Press CTRL+C to quit |

Q.15/ The path of the route to get to all analysts is: /analysts/

The type of HTTP request we can make to access it is : GET

Q.24/ Pour sécuriser l’API, il est nécessaire de vérifier que seules les personnes authentifiées peuvent accéder aux données sensibles. Dans notre projet, nous avons choisi d’utiliser un décorateur @token\_required, qui vérifie que chaque requête HTTP contient un token JWT valide avant d’exécuter la route.

Routes à sécuriser et justification :

1. Routes des analysts (/analysts)
   * GET /analysts/ : retourne la liste de tous les analysts.  
     → Sensible car la liste des utilisateurs est confidentielle.
   * GET /analysts/<analyst\_username> : retourne les informations d’un analyst spécifique.  
     → Sensible car les informations personnelles ne doivent pas être accessibles à n’importe qui.
   * PATCH /analysts/<analyst\_username> : permet de modifier le mot de passe d’un analyst.  
     → Sensible car elle modifie des données critiques.

Fonctionnement du décorateur @token\_required :

* Avant d’exécuter la route, il récupère le token dans l’en-tête Authorization.
* Il vérifie si le token est valide et non expiré.
* Si le token est invalide ou absent, la route n’est jamais exécutée et la requête renvoie une erreur 401.
* Si le token est valide, la route s’exécute normalement et reçoit le username de l’analyst authentifié.

L’ajout de @token\_required sur les routes sensibles assure que seules les personnes authentifiées peuvent accéder ou modifier des informations critiques, garantissant ainsi la sécurité et la confidentialité des données de l’API.