Documentation for VetCare

1 Database

1.1 Overview of Entities

User: Represents users of the system, including pet owners

- UserID (Primary Key)
- Name
- Password
- Email
- PhoneNumber
- Address

Vet: Represents veterinarians working at various clinics

- VetID (Primary Key)
- Name
- Password
- Email
- ClinicID (Foreign Key to **Clinic**)
- PhoneNumber
- Address

Pet: Represents pets owned by users

- PetID (Primary Key)
- OwnerID (Foreign Key to **User**)
- Name
- Species
- Breed
- Age
- MedicalHistoryID (Foreign Key to **MedicalHistory**)

Appointment: Tracks appointments scheduled for pets with veterinarians

- AppointmentID (Primary Key)
- UserID (Foreign Key to **User**)
- PetID (Foreign Key to **Pet**)
- ClinicID (Foreign Key to **Clinic**)

- AppointmentDate
- Status (scheduled, completed, or cancelled)
- GeneralNotes
- Fees (base fixed fee, e.g. $84.95 for one appointment)
- VetID (Foreign Key to **Vet**)

MedicalHistory: Stores the medical history of pets

- MedicalHistoryID (Primary Key)
- PetID (Foreign Key to **Pet**)
- ChronicConditions
- Allergies
- Notes (Updated by VetInCharge)
- LastVaccinationDate
- LastTreatmentDate
- LastPrescriptionDate

VaccinationRecord: Contains records of vaccinations administered to pets

- VaccinationID (Primary Key)
- PetID (Foreign Key to **Pet**)
- VaccineName
- DateAdministered
- NextDueDate
- Status
- Notes
- VetID (Foreign Key to **Vet**)

TreatmentPlan: Stores treatment plans for pets

- TreatmentPlanID (Primary Key)
- PetID (Foreign Key to **Pet**)
- Diagnosis
- Description
- DateAdministered
- EndDate
- Notes
- VetID (Foreign Key to **Vet**)

Prescription: Records prescriptions given to pets

- PrescriptionID (Primary Key)
- PetID (Foreign Key to **Pet**)
- VetID (Foreign Key to **Vet**)
- MedicineID (Foreign Key to **Medicine**)
- Instructions
- DosageQuantity
- DateAdministered
- ExpiryDate
- RepeatsLeft
- RenewalDate

Medicine: Stores information about medicines used in prescriptions

- MedicineID (Primary Key)
- Name
- Description
- Strength
- SideEffects
- Cost

EducationalResource: Contains educational resources available to users

- ResourceID (Primary Key)
- Title
- ResourceType
- Author
- PublishDate
- Category
- Content
- Description

Clinic: Represents veterinary clinics.

- ClinicID (Primary Key)
- Name
- Address
- PhoneNumber
- Email

SavedResources: Tracks educational resources saved by users

- SavedResourceID (Primary Key)
- UserID (Foreign Key to **User**)
- ResourceID (Foreign Key to **EducationalResource**)
- SavedAt

LatestTrends: Contains the latest trends and information in veterinary care

- TrendID (Primary Key)
- Title
- Description
- Author
- PublishDate
- TrendCategory

1.2 Relationships

- **User ↔ Pet**: One-to-Many
  - A user can own multiple pets, but each pet has only one owner.
- **Vet ↔ Clinic**: Many-to-One
  - A vet works at one clinic, but a clinic can have multiple vets.
- **Pet ↔ MedicalHistory**: One-to-One
  - Each pet has one instance of medical history, created when significant medical events occur, such as a treatment plan, vaccine, or prescription being administered.
- **Pet ↔ Appointment**: One-to-Many
  - A pet can have multiple appointments
- **MedicalHistory ↔ VaccinationRecord/TreatmentPlan/Prescription**: One-to-Many
  - A medical history can have multiple associated records.
- **User ↔ Appointment**: One-to-Many
  - A user can book multiple appointments for their pets.
- **Appointment ↔ Vet**: Many-to-One
  - Each appointment is associated with one vet.

1.3 Basic Querying Information

This section is intended to serve as a basic guide on querying the VetCare database.

- **Finding All Treatment Plans Associated with a Pet**
  - `SELECT * FROM TreatmentPlan`
  - `WHERE PetID = {PetID};`
- **Finding All Vaccinations Associated with a Pet**
  - `SELECT * FROM VaccinationRecord`
  - `WHERE PetID = {PetID};`
- **Finding All Prescriptions Associated with a Pet**
  - `SELECT * FROM Prescription`
  - `WHERE PetID = {PetID};`
- **Finding All Past Appointments a User Has Had/Is Upcoming**
  - `SELECT * FROM Appointment`
  - `WHERE UserID = {UserID}`
  - `AND Status = 'completed'/'upcoming'/;`

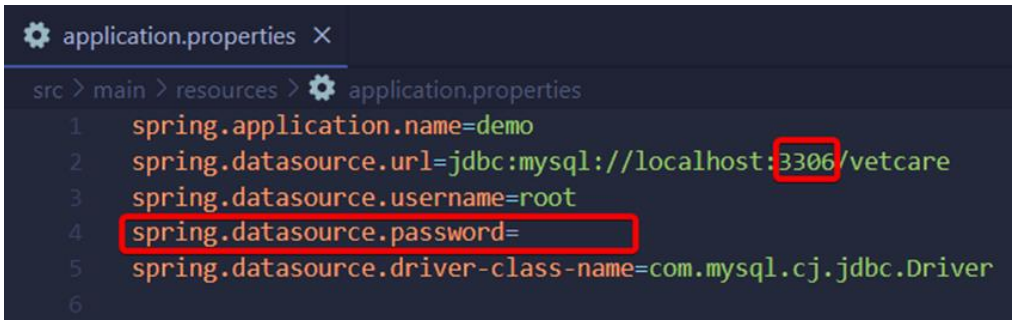
1.4 Usage Points

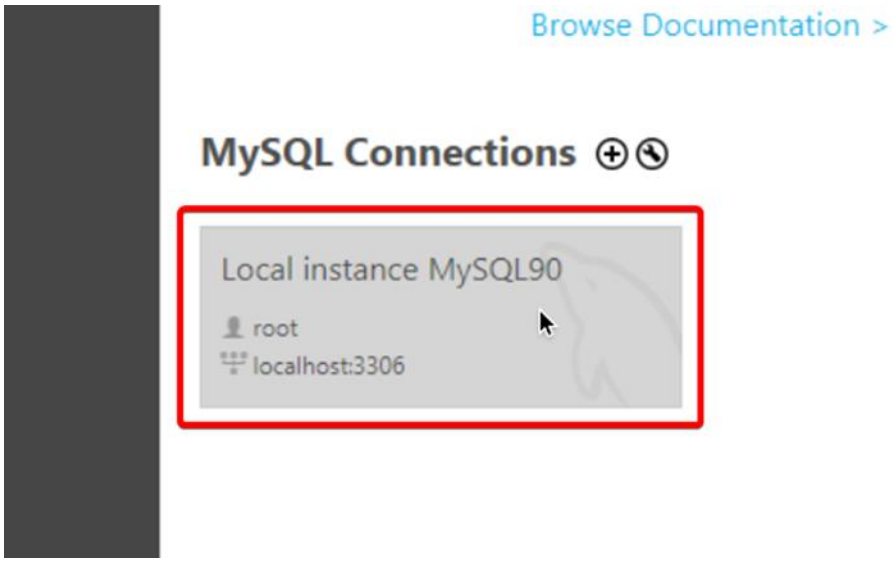
- **Fixed Appointment Fee**
  - Each appointment has a flat, fixed fee stored in the 'Fees' attribute of the 'Appointment' table. Currently, this value sits at A$84.95. Costs of vaccinations and prescriptions are extra and should be added to it if calculating and displaying total costs incurred.
- **Updating Appointment Notes**
  - By default, the 'Notes' attribute of the 'Appointment' table is empty. The vet can and should update notes after every appointment.
  - Appointment notes are intended on being very general notes (e.g., "routine check-up", "treatment plan of {TreatmentPlanID/TreatmentPlanName} prescribed"
- **MedicalHistory Table**
  - A pet can only have one instance of 'MedicalHistory'. This table is created by default, but all fields are empty.

    - If a user wishes to see the medical history for their pet that has no significant medical events (such as the administration of a vaccine, treatment plan, or prescription), the system should explicitly state “No significant medical history has been recorded for this pet”.
    - When a significant medical event occurs, the ‘Notes’ field should be updated in the ‘MedicalHistory’ table, along with the ‘GeneralNotes’ field in the ‘Appointment’ table.
- **TreatmentPlan/VaccinationRecord/Prescription Tables**
    - Whenever a new entry is created, an auto-incrementing primary key is generated. The backend logic of the system should also update ‘LastVaccinationDate’, ‘LastTreatmentDate’, and/or ‘LastPrescriptionDate’ in the ‘MedicalHistory’ table with a timestamp of when any of these events occurred.
- **More Detailed Queries**
    - When more detailed information is required, such as all specific vaccine dates for a pet or all prescriptions that a pet has been administered in the past, the developer will need to query the relevant table and then filter by ‘PetID’, as seen in section 1.3.
- **Limitations**
    - If medical information history is needed such that a user wishes to see the last vaccination date/prescription date/treatment date **along** with the specific vaccination/prescription/treatment that was administered, multiple and complex SQL queries must be executed.

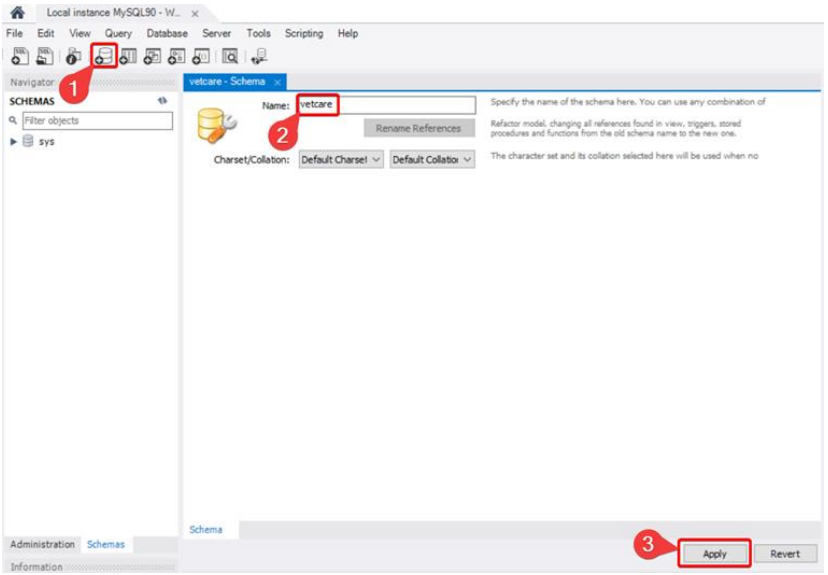
1.5 Setting Up the MySQL Database Instance

For development and testing, VetCare will use a local instance of MySQL running on each developer’s machine. This is also known as a file-based setup, like what could be achieved with SQLite or H2 Database.

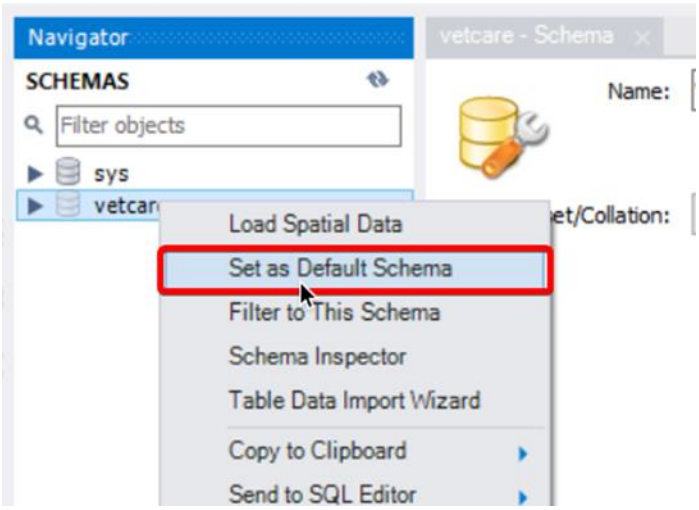
- **Step 1: Setup MySQL Database**
  - Download and install MySQL Server 8.x
  - Ensure the server is running on port 3306 and the root password is blank. This can be done by using the defaults in MySQL configurator, and then running `SET PASSWORD FOR root@localhost='';` in MySQL CLI
  - Alternatively, you can change the port and root password in src/main/resources/application.properties
    - 

- **Step 2: Configure MySQL Database**
  - Download and install MySQL Workbench 8.x
  - Connect to the local database instance
    - 

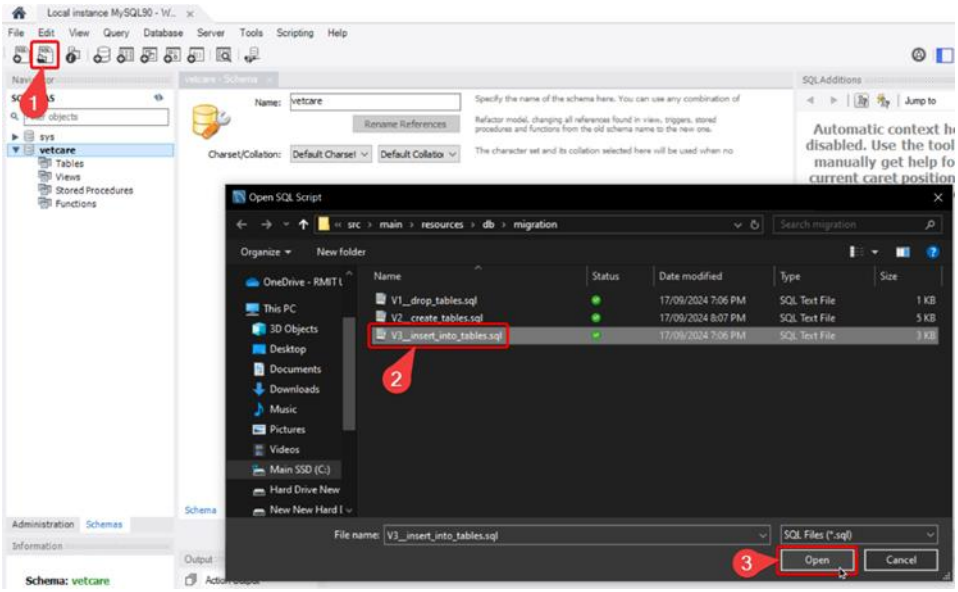
  - Create a new schema named `vetcare` via the toolbar

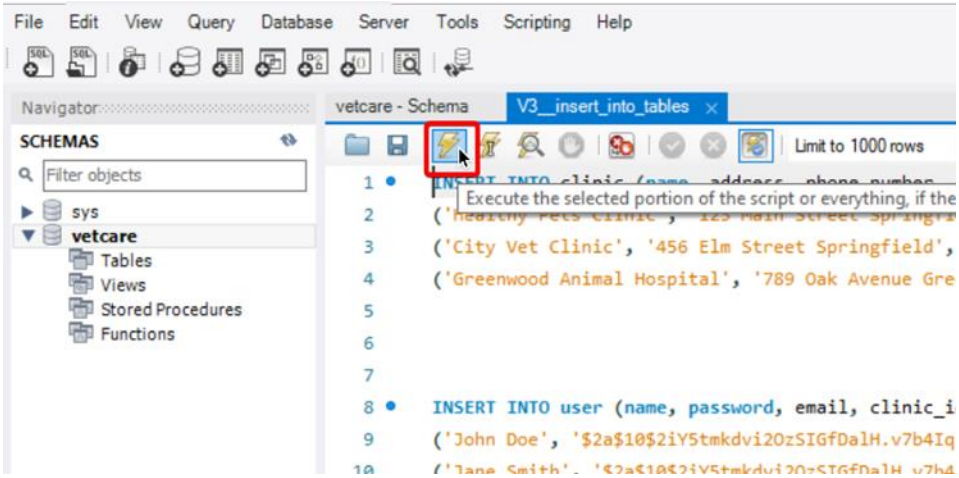


- Select `vetcare` as the default schema



- **Step 3: Run Webserver**
  - To allow flyaway to initialize the database, we need to run the webserver. You can do this through the Maven command `./mvnw spring-boot:run`
  - The webserver should now work. For further testing, stop the server before proceeding
- **Step 4: Use Testing Data if Required**
  - In MySQL Workbench, import the test SQL script in `/src/main/resources/db/migration/V3__insert_into_tables.sql` and execute it

- 

- 

1.6 Using Thymeleaf Layouts

Thymeleaf layouts are stored in the directory `src/main/resources/templates/layout.html`, and contain a header and footer that are applied to any webpage that specifies it.

To use a apply the layout to a webpage, the webpage must contain the property `layout:decorate="~{layout.html}"`, inside of its HTML tag, so that when this page is requested, the elements of the page are placed inside of a container defined by the layout file before being sent to the client.