# PROG8850 – Assignment 1: Database Automation and Scripting

Name: Jithin Jyothi  
Instructor: Prof. Richard Hildred

GitHub link: <https://github.com/JithinJyothi95/Assignment1-DbAutomation>

# Question 1.1: Understanding Database Automation

Database automation is the process of using scripts and tools to execute routine database tasks such as backups, updates, deployments, and monitoring without manual intervention. In today's fast-paced, data-driven environments, automation is critical for ensuring data consistency, reducing errors, and supporting scalability.  
  
When managing large volumes of data, manual processes can become error-prone, time-consuming, and inefficient. Automation helps streamline these operations securely and repeatedly, providing a stable foundation for modern software systems.

# Question 1.2: Benefits of Database Automation

* Reduced Errors: Automation reduces the chance of human errors like missed commands or typos.
* Increased Reliability: Scheduled scripts ensure consistent execution, making database operations dependable.
* Faster Deployments: Changes can be pushed automatically to production using CI/CD pipelines.
* Cost Efficiency: Less manual work means less time spent by developers and DBAs.
* Real-World Example:  
  Companies like Amazon and Netflix use automated CI/CD pipelines to manage their databases across multiple environments. Tools like Jenkins, GitHub Actions, and Ansible help these organizations update and backup their databases with confidence.

# Question 2.1: Python Script for Database Backup Automation

* To make each backup unique, I used a timestamp in the filename like `mysql\_backup\_20250611\_153000.sql`.
* I verified that the script successfully creates `.sql` backup files and tested it manually with `mysqldump`.
* The script checks for errors and prints appropriate messages, making it easy to debug.

# Question 2.2: Python Script for Database Change Deployment

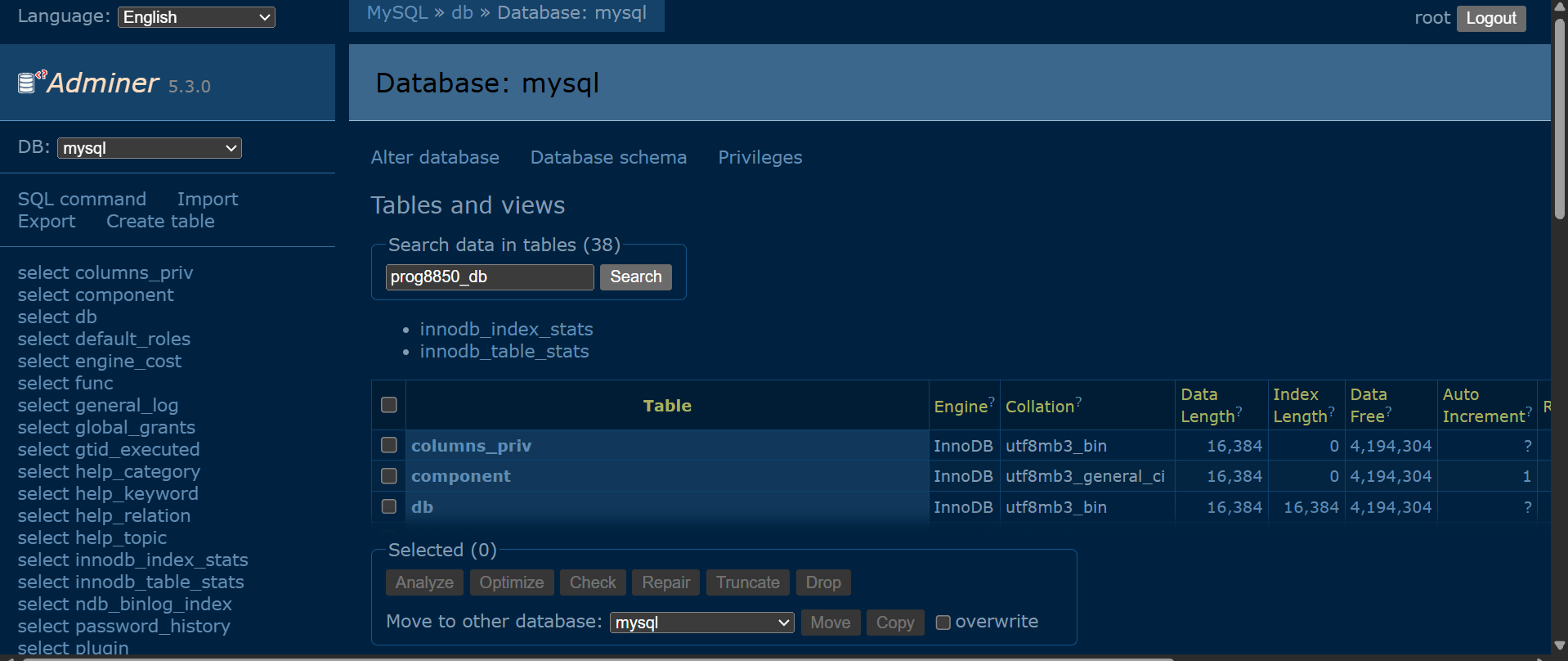
* In `**deploy\_changes\_script.py**`, I used the `**mysql.connector**` Python library to connect directly to MySQL.
* The script checks whether the target database `prog8850\_db` exists and creates it if it does not.
* Then, it opens the `**schema\_changes.sql**` file and executes the SQL commands to create or modify tables.
* I added exception handling to provide clear error messages if anything goes wrong. This script can be used in CI/CD workflows to deploy schema changes automatically.

# CI/CD Integration and Screenshots:

I successfully tested the deployment using both:

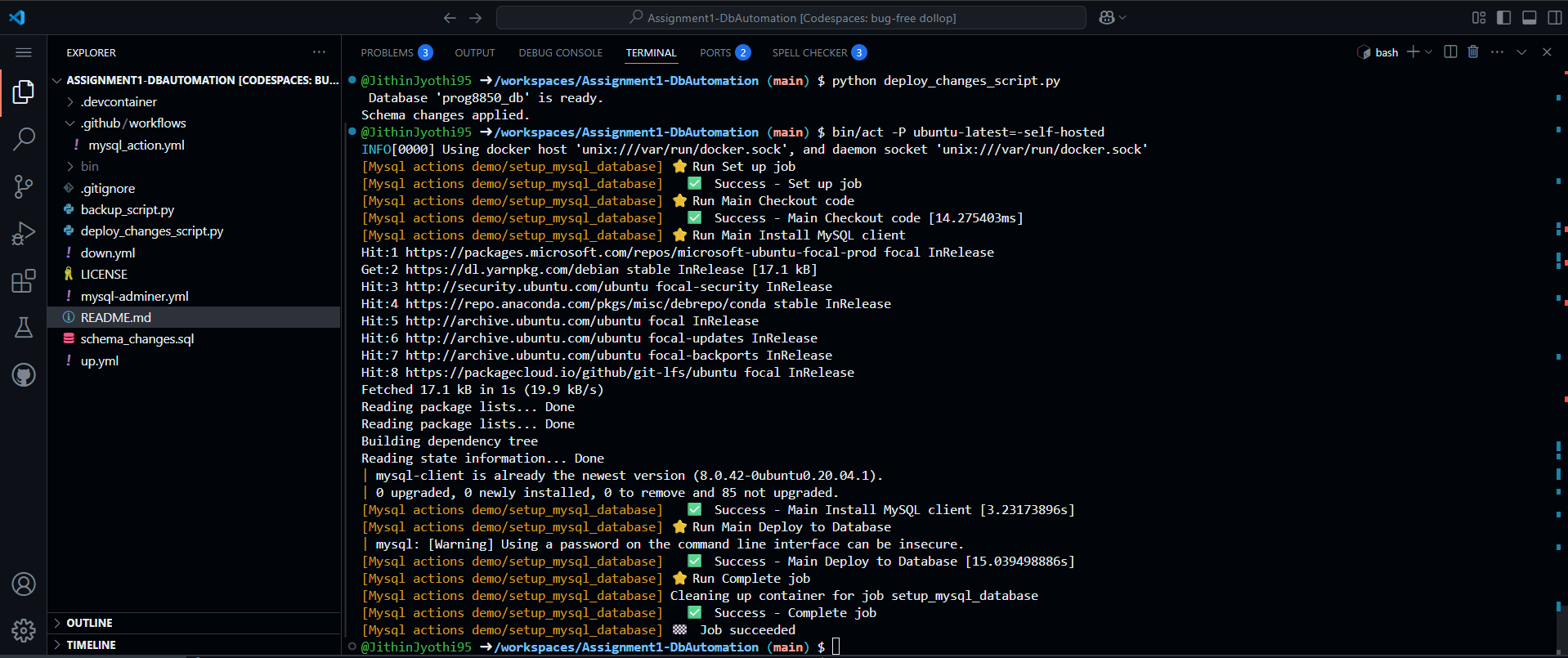
* Local environment with `act -P ubuntu-latest=-self-hosted`
* GitHub-hosted Actions using a MySQL service container
* Below are the screenshots of the working pipeline, Adminer UI, and terminal confirmations.

## 1. Adminer UI – prog8850\_db Table Found



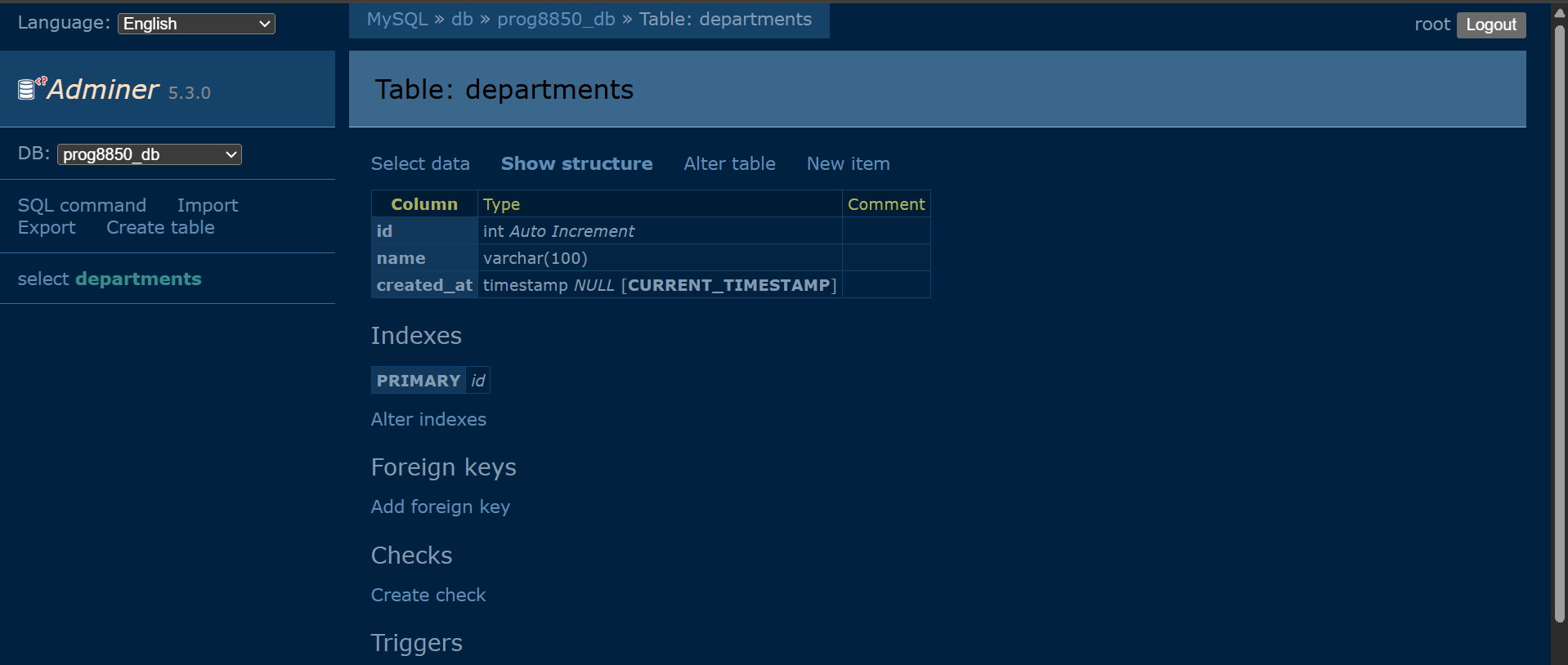
This confirms that the database exists after script execution.

## 2. Terminal – Local CI/CD via act



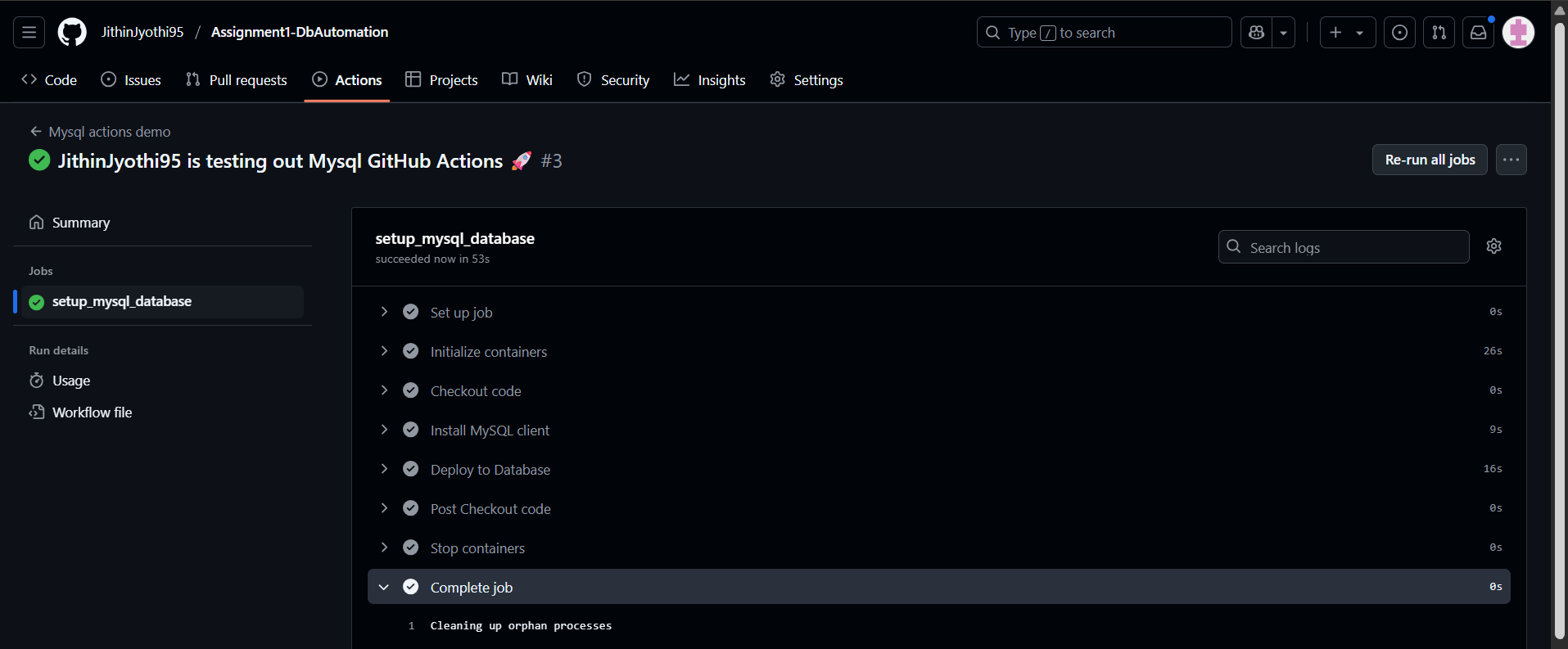
CI/CD executed locally using act in Codespaces.

## 3. Adminer – Table: departments



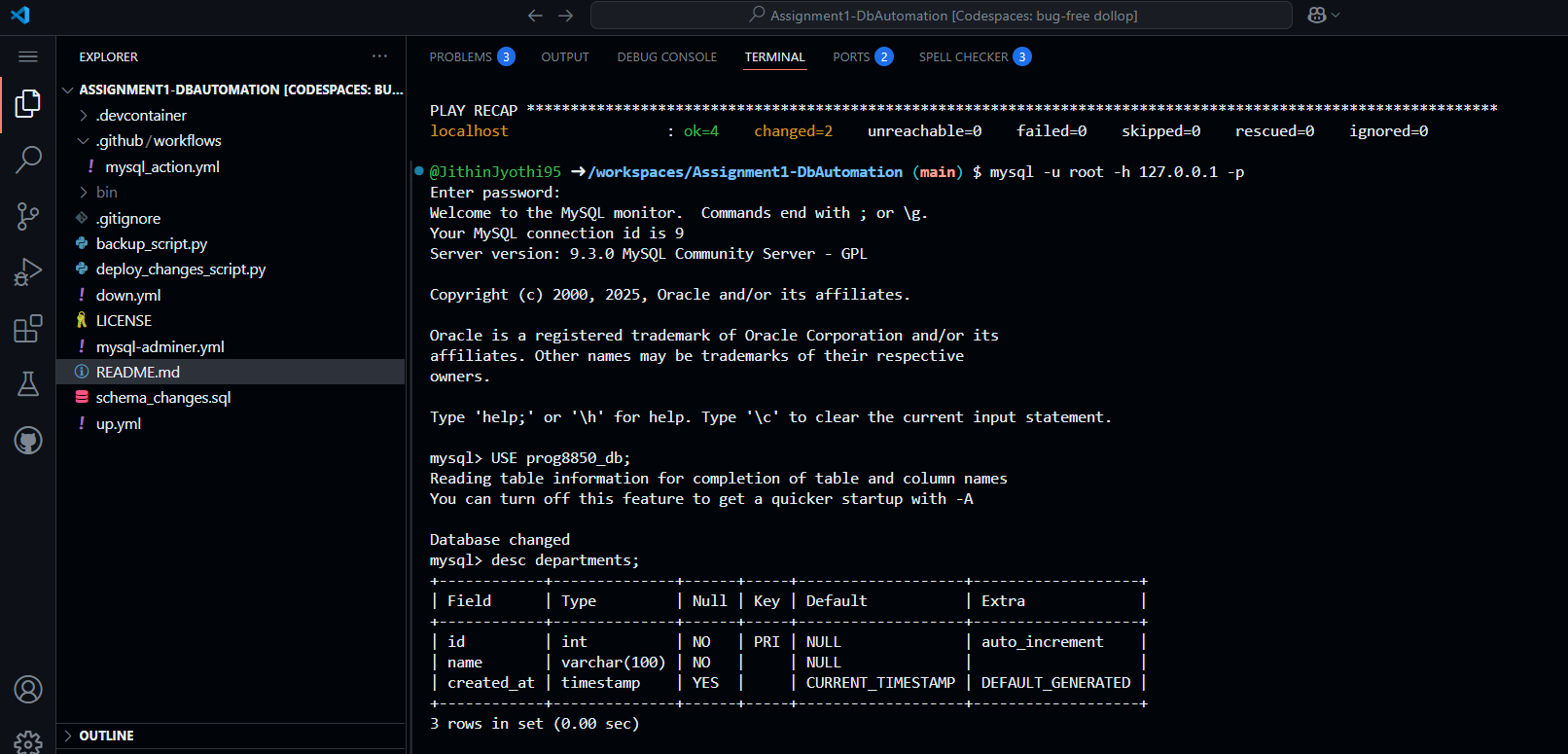
Schema successfully deployed via GitHub Actions or script.

## 4. GitHub Actions – Workflow Passed



CI/CD validation from GitHub-hosted runner with MySQL service.

## 5. MySQL CLI – Verified in Codespaces



Verified schema and data structure manually in CLI.