Abstract:

This project implements an extended relational railway database system in MariaDB. The work includes the enforcement of key business constraints through triggers and unique constraints, the construction of complex SQL views for querying service schedules and routes, and the creation of stored procedures to manage dynamic changes in service routing and scheduling. Additionally, a Python-based command line interface has been developed to allow users to interact with the system in a simple and human-readable manner. The implementation emphasizes data integrity, error handling, and usability, ensuring the system behaves predictably under normal and edge-case scenarios.

Constraints:

Three core constraints were identified and implemented to preserve the integrity of the data in the railway scheduling system. The first constraint ensures that a service cannot arrive at a location after it departs. This was enforced using a BEFORE INSERT trigger on the stop table, which checks that the arrival time does not exceed the departure time.

The second constraint ensures that all non-terminal locations in a route have a finite departure differential, represented as a time difference in hours and minutes. A BEFORE INSERT trigger on the plan table verifies that if a location is not the terminal point (i.e., other locations follow it), its departure differential must not be set to ω, represented internally by 99:99.

The third constraint ensures that a train cannot operate on more than one service at the same time. This is enforced using a UNIQUE constraint on the combination of uid, dh, and dm in the service table. This prevents the same train unit from being scheduled for multiple departures at the same hour and minute.

Queries:

Three complex SQL queries were developed to generate views that expose key operational data from the system. These views allow efficient retrieval of scheduling and routing information.

The first view, trainLEV, lists all services operated by a specific train, identified here as train 170406. It returns the headcode, origin station, and formatted departure time for each relevant service. The results are ordered chronologically to reflect actual timetable structure.

The second view, scheduleEDB, shows all services departing from Edinburgh (EDB). It includes the headcode, formatted departure time, platform, destination station, train length (based on coach count), and operating company. The destination is inferred either as the next planned stop or ideally the terminal location of the route, and the results are ordered by departure time.

The third view, serviceEDBDEE, displays the full sequence of locations visited by the service 1L27 departing at 18:59. Each row includes the location name, station code (if applicable), platform, and arrival and departure times. This view captures both through-points and stopping locations, ordered in sequence as they appear on the route.

Procedures:

Two stored procedures were created to enable dynamic modifications to the schedule and route structure.

The proc_new_service procedure allows the insertion of a new service, given an origin station, departure time, platform, train ID, and operating company. It automatically generates a new unique headcode by scanning existing headcodes and selecting the next available increment (e.g., 1A01, 1A02). Input validations ensure the origin station and train exist before proceeding with route and service creation.

The proc_add_loc procedure allows a new location to be added to an existing route. It accepts the headcode of the route, the new location to be inserted, its predecessor location, the departure differential (or ω if it is terminal), and optionally, the arrival differential and platform if the location is a stopping point. The procedure validates the structure of the route to prevent terminal locations being placed mid-route and enforces the presence of a valid location and route. If the location is a stop, it is also inserted into the stop table. All edge cases are handled explicitly, and meaningful error messages are raised if invalid inputs are detected.

Command Line Interface Utility:

The command line interface, implemented as rtt.py, provides two primary functionalities for querying the database.

The first command, --schedule <loc>, generalizes the scheduleEDB view and prints all scheduled services from the given station. The output includes the departure time, destination, train length, and operator in a human-readable format. This function can be used with any valid origin station code, making it versatile for timetable queries.

The second command, --service <hc> <dep>, generalizes the serviceEDBDEE view and prints the full route for a specific service identified by its headcode and departure time. The output displays all locations in order, with arrival and departure times and platform information where applicable. This command is particularly useful for following the path of a given service through the rail network.

The CLI uses the mariadb connector package to establish a connection to the MariaDB server. It includes input validation and informative error messages to ensure usability and robustness.

Tests:

For all the added constraints, queries, procedures and command line interface there are commands to run the tests in src/tests.sql. There are additional comments to state what is being tested.

<u>Conclusion</u>:

This project successfully extends a structured database system for railway operations with sophisticated integrity mechanisms, procedural extensions, and user-level interaction tools. It demonstrates strong practical database design and SQL proficiency, combining triggers, procedures, and views in a coherent and testable architecture. The system not only enforces real-world business rules but also empowers users to explore and maintain complex train services via automated backend logic and an accessible front-end CLI. Testing across all levels confirmed the system's stability, correctness, and flexibility in handling a wide variety of operations.