School of Computer Science, University of St Andrews, CS3101, 2024-25

# CS3101: Databases

**Assignment:** P2

**Deadline:** Wednesday, 23 April 2025 (Week 12) at 9pm

**Weighting:** 50% of coursework component

MMS is the definitive source for deadline and credit details.

If you feel any part of the specification is ambiguous, you may interpret it in any way that you feel is appropriate, and explain your decisions in your report.

---

## 1 Objective

In this practical, you will extend a given database with appropriate views, triggers, and procedures. You will also implement a command line interface that is capable of querying the database.

**Learning Objectives** The purpose of this practical is to promote awareness of issues involved in using databases and database connections.

## 2 Practical Requirements

You are required to extend a given database and produce a simple command line user interface in line with a railway setting similar to that described in the first practical specification. The relational model for the provided database is given in Sec. 2.1. You will need to load the database, produce specified queries and procedures, enforce appropriate constraints, and develop a user interface for part of the system. Finally, you will compose a single word-processed document in PDF format, showing evidence of your practical work and justifying design and implementation decisions. The code, report, and dump of the database must be submitted to MMS.

### 2.1 Relational Model

**Domains** The relational model uses several domains specific to the scenario, which we define below. Let $\mathbb{N}_n = \{i \mid i \in \mathbb{N} \land i \leq n\}$ be the set of natural numbers less than or equal to $n$. Let Char be the set of characters defined by the ISO basic Latin Alphabet, with Lower and Upper representing the minuscule and majuscule character subsets, respectively. Let String be the set of words over Char extended with spaces.

- An Hour is represented by two digits using the 24-hour clock, e.g. 04 for 4am and 21 for 9pm.

$$\mathsf{Hour} = \mathbb{N}_{24}$$

  $\mathsf{Hour}^\infty$ extends Hour with a notion of infinity, denoted $\omega$.

$$\mathsf{Hour}^\infty = \mathsf{Hour} \cup \{\omega\}$$

- A Minute is similarly represented by two digits.

$$\mathsf{Minute} = \mathbb{N}_{59}$$

  $\mathsf{Minute}^\infty$ extends Minute with a notion of infinity, denoted $\omega$.

$$\mathsf{Minute}^\infty = \mathsf{Minute} \cup \{\omega\}$$

- A Time is represented the concatenation of Hour and Minute, e.g. 1345 for 1:45pm.

$$\mathsf{Time} = \{hm \mid h \in \mathsf{Hour} \land m \in \mathsf{Minute}\}$$

- A StnCode is a three-character string of majuscule characters.

$$\mathsf{StnCode} = \{abc \mid a, b, c \in \mathsf{Upper}\}$$

- A TrainID is a six-digit natural number.

$$\mathsf{TrainID} = \{abcdef \mid a, b, c, d, e, f \in \mathbb{N}_9\}$$

- A Headcode is a four-character string, comprising train class, destination area, and an identifier.

$$\mathsf{Headcode} = \{cdij \mid c, i, j \in \mathbb{N}_9 \land d \in \mathsf{Upper}\}$$

- Company: code for a subset of current train operating companies.

$$\mathsf{Company} = \{\mathsf{VT}, \mathsf{CS}, \mathsf{XC}, \mathsf{SR}, \mathsf{GR}, \mathsf{GW}\}$$

**Locations and Stations**

- $\mathrm{location}(\underline{loc} : \mathsf{String})$
- $\mathrm{station}(\underline{loc} : \Pi_{loc}(\mathrm{location}), code : \mathsf{StnCode})$

Locations are identified by their names. Some locations are stations, which have Computer Reservation System (CRS) codes.

**Trains**

- $\mathrm{train}(uid : \mathsf{TrainID})$
- $\mathrm{coach}(\underline{uid} : \Pi_{uid}(\mathrm{train}), \underline{lid} : \mathsf{Upper})$

A (passenger) train is a unit formed of a locomotive and a set of coaches, and is identified by its Unit ID. Each coach has a letter designation, which is unique in the context of the train.

**Services**

- $\text{service}(\underline{hc} : \Pi_{hc}(\text{route}), \underline{dh} : \mathsf{Hour}, \underline{dm} : \mathsf{Minute}, pl : \mathbb{N}, uid : \Pi_{uid}(\text{train}), toc : \mathsf{Company})$
- $\text{route}(\underline{hc} : \mathsf{Headcode}, orig : \Pi_{loc}(\text{station}))$
- $\text{plan}(\underline{hc} : \Pi_{hc}(\text{route}), \underline{frm} : \Pi_{loc}(\text{location}), loc : \Pi_{loc}(\text{location}), ddh : \mathsf{Hour}^{\infty}, ddm : \mathsf{Minute}^{\infty})$
- $\text{stop}(\underline{hc} : \Pi_{hc}(\text{plan}), \underline{frm} : \Pi_{frm}(\text{plan}), \underline{loc} : \Pi_{loc}(\text{plan}), adh : \mathsf{Hour}, adm : \mathsf{Minute}, pl : \mathbb{N})$

A service represents a train that departs its origin station at a specific time and travels a particular route. Each service is operated by a train operating company, is associated with a particular train, and is uniquely identified by its headcode and its departure time. The headcode uniquely identifies the service's route, where a route is comprised of an origin station and a sequence of planned locations through which services will pass, or at which services will stop. The route's destination, i.e. the station at which it terminates, is defined as being the last planned location in the sequence. Services must stop at their destination.

Each planned location is identified by the route of which it is part, and from the location that immediately precedes it in that route. The time taken for a service to depart a planned location from its preceding location (hereinafter *departure differential*) is recorded in hours and minutes. For example, a service departing Edinburgh (EDB) at 1859, will subsequently pass (i.e. depart) Princes St Gardens at 1900, and thence depart from Haymarket (HYM) at 1905. The plan for Princes St Gardens would record a departure differential of 0001, i.e. 00 hours and 01 minute. Similarly, the plan for Haymarket (HYM) would record a departure differential of 0005. A route's destination has a departure differential of infinity. Similarly, a planned location in a route at which services stop additionally records platforms and arrival times via arrival differentials. For example, the above plan for Haymarket (HYM) would additionally record that services arrive in Haymarket (HYM) at 1904 via the arrival differential of 0004.

## 2.2 Details

**The Database** Load the database located on StudRes[1] into MariaDB. Your database implementation must be on the School MariaDB servers. See Sec. 4 for information on how to connect.

**Constraints** Enforce each of the following constraints. You may use checks, triggers, procedures, or any other appropriate mechanism.

1. A service cannot arrive after it departs a station.
2. All locations on a route, barring the destination location, must have a finite departure differential.
3. A train cannot be part of two different services departing at the same time.

**Queries** Specify each of the below queries in SQL and run them against your database. Once you have finalised your queries, define views in the database to represent these queries such that they adhere to the given schemata.

1. Produce a table that lists all the services on which train 170406 operates. The query should return one row per service, and should conform to the following schema.

   $$\text{trainLEV}(hc : \mathsf{Headcode}, orig : \mathsf{Stn}, dep : \mathsf{Time})$$

   Here, $hc$ is the service headcode, $orig$ is the origin location, and $dep$ is the departure time for the service departing its origin. The services should be listed in chronological order.

[1] `https://studres.cs.st-andrews.ac.uk/CS3101/Coursework/CS3101-P2.sql`

2. Produce a table that lists all services that depart Edinburgh (EDB). The query should return one row per service, and should conform to the following schema.

$$\mathrm{scheduleEDB}(hc : \mathsf{Headcode}, dep : \mathsf{Time}, pl : \mathsf{Nat}, dest : \mathsf{Stn}, len : \mathsf{Nat}, toc : \mathsf{Company})$$

Here, $hc$ is the service's headcode and $dep$ is the departure time from its origin station. $pl$ is the platform from which the service will depart Edinburgh, $len$ is the number of coaches the service's train has, and $toc$ is the train operating company. $dest$ should be the next planned location for the service. Alternatively, $dest$ will preferably be the destination station for the service. Services should be listed in ascending order of departure time.

3. Produce a table that lists all the locations through which the 1859 1L27 service departs, and includes both the service's origin and destination locations. The query should return one row per location, and should conform to the following schema.

$$\mathrm{serviceEDBDEE}(loc : \mathsf{Loc}, stn : \mathsf{StnCode}, pl : \mathsf{Nat}, arr : \mathsf{Time}, dep : \mathsf{Time})$$

Here, $loc$ is the origin, destination, or a location from which the service departs. Should $loc$ be a station, $stn$ gives the corresponding station code, and $pl$ gives the corresponding platform. Arrival and departure times from that location are recorded under $arr$ and $dep$, respectively. Locations should be listed in the order that they are visited.

**Procedures** Provide the following functionality using procedures.

1. Provide a procedure for adding a new service. It should be called `proc_new_service`. It should have precisely five inputs: *i)* the origin station, *ii)* the origin platform, *iii)* the departure time from the origin, *iv)* the train used for the service, and *v)* the train operating company. A fresh headcode for the service should be generated automatically.

2. Provide a procedure for adding a planned location to a route. It should be called `proc_add_loc`, and should only permit existing locations to be added to existing routes. It should have precisely six inputs: *i)* the headcode of the route being modified, *ii)* the location being added, *iii)* the location in the route that is to immediately precede the added location, *iv)* the departure differential, *v)* the arrival differential (if any), and *vi)* the platform (if any). If the service is to stop at the location being added, neither the arrival differential nor the platform may be null. The procedure should be able to extend routes by appending locations to the sequence. The procedure will preferably also enable insertion of planned locations mid-route.

**Command Line Interface** Design and implement a simple command line interface for the system, which interacts with your database. It should be implemented using Python and give human-readable error messages. It will be called `rtt.py` and will accept the following arguments.

1. `--schedule <loc>`
   Prints the schedule for a given station. The option should generalise the scheduleEDB view.

2. `--service <hc> <dep>`
   Prints the sequence of stations at which a given service stops. The option should generalise the serviceEDBDEE view.

**Report** Write a report discussing your design and implementation. The document should state and justify any assumptions and design decisions made. It should include:

1. an abstract, stating what was achieved;
2. a section documenting your constraints;
3. a section documenting your queries;
4. a section documenting your procedures; and
5. a section documenting your command line interface utility.

# 3 Deliverables

You are required to submit an archive file (`.zip`) to the P2 slot on MMS that contains your report, the anonymised dump file for your MariaDB database, and your `rtt.py` script. Your report should be in the PDF format.

# 4 MariaDB & Teaching Services

The School provides every user with access to a teaching server running a MariaDB service. To use the service, you must ssh into your teaching server, retrieve your MariaDB credentials, create a database, and select it for use. You can find detailed information about how to do this, and about the MariaDB service more generally, on the School wiki.

`https://wiki.cs.st-andrews.ac.uk/index.php?title=Teaching_Service`

To generate a dump file for a database you must ssh into your teaching server, change to the directory in which you wish to create the dump file, then run the following command.

```
1 mysqldump --routines -p database_name > filename.sql
```

Here, `database_name` should be replaced with the name of the database you want to dump, and `filename.sql` should be similarly replaced with the name of the file you want to dump the database to. You will be prompted to enter your MariaDB password. If successful, the dump file should contain all the SQL DDL statements required to recreate your database, including definitions for tables and any statements to load all data where applicable. You can check this by examining the contents or by loading the dump file into a new database.

**Anonymisation** Your dump file should be anonymised before submission such that your university username does not occur as either part of the filename or within the dump file. This may be achieved with the following command. Other anonymisation methods are possible and permissible.

```
1 sed 's/username/anon/' filename.sql > filename_anonymised.sql
```

# 5 Policies and Guidelines

## 5.1 Marking

See the Generic mark descriptors in the School Student Handbook

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors`

**Lateness** The standard penalty for late submission applies. (Scheme A: 1 mark per 24 hour period, or part thereof.)

### 5.2 Good Academic Practice

The University policy on Good Academic Practice applies.

```
https://www.st-andrews.ac.uk/education/handbook/good-academic-practice/
```