THE UNIVERSITY OF QUEENSLAND
AUSTRALIA

# INFS2200/7903 PROJECT ASSIGNMENT 1

| | |
|---|---|
| **Due Date:** | 4:00 PM 8th September 2023 |
| **Total Marks:** | 30 marks for 10% |
| **What to submit:** | A single SQL script file |
| **Where to submit:** | Electronic submission via Blackboard |

The goal of the project assignment is to gain practical experience in applying several database management concepts and techniques by using the Oracle DBMS. In particular, this assignment mainly focuses on ensuing database semantics using various integrity constraints.

Your main task is to first populate your database with appropriate data, then design, implement, and test the appropriate queries to perform the tasks explained in the next sections.

You must work on this project **individually**. Academic integrity policies apply. Please refer to *3.60.04 Student Integrity and Misconduct* of the University Policy for more information.

**Roadmap:** Section 1 describes the database schema for the assignment and provides instructions on downloading the script file needed to create and populate the database. Section 2 describes the tasks to be completed for this assignment. Finally, Section 3 explains the submission guidelines and marking scheme.

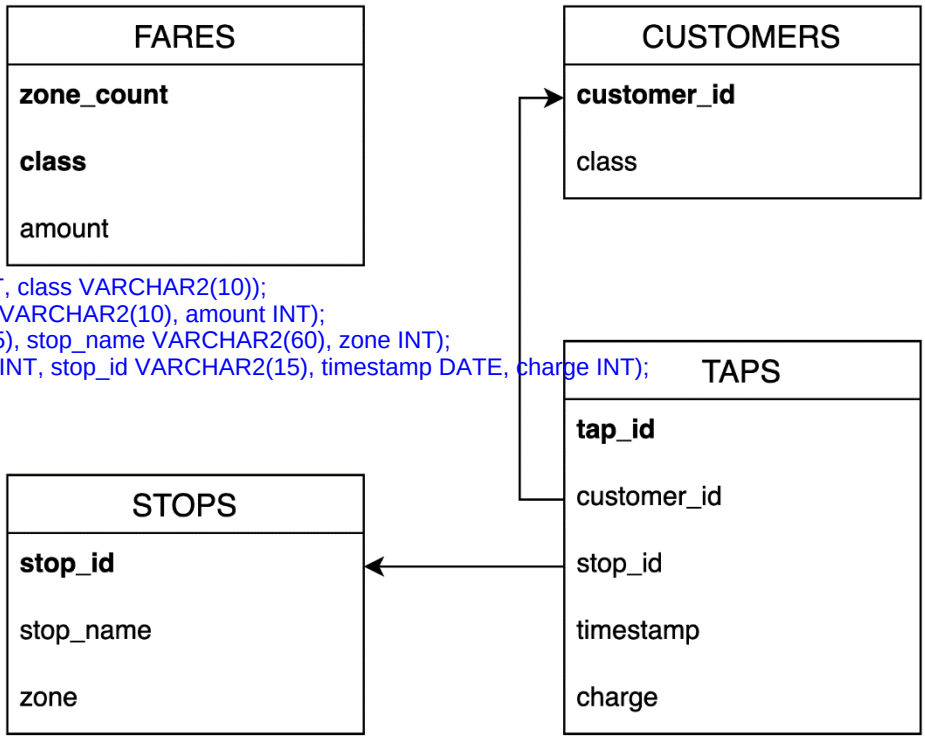
Enjoy the project!

## Section 1 — the TRAVEL database

The `TRAVEL` database captures information about public transport users' journeys and their fares. The database includes four tables: `CUSTOMER`, `FARES`, `STOPS` and `TAPS`.

Users travel around the network, "tapping on" with their fare card when they enter a station or vehicle, and "tapping off" when they exit a station or vehicle.

- **`CUSTOMERS`** stores each customer's fare class (e.g., 'Adult' or 'Concession') and their fare card ID.
- **`FARES`** stores the price in cents, for each combination of class and the number of zones travelled.
- **`STOPS`** stores information about each stop on the network, particularly its name and fare zone.
- **`TAPS`** stores the travel history of each customer, in the form of the stops they have tapped on or off at, and the fares they were charged along the way.




CUSTOMERS (customer_id INT, class VARCHAR2(10));
FARES (zone_count INT, class VARCHAR2(10), amount INT);
STOPS (stop_id VARCHAR2(15), stop_name VARCHAR2(60), zone INT);
TAPS (tap_id INT, customer_id INT, stop_id VARCHAR2(15), timestamp DATE, charge INT);

Please go to Blackboard and download the supplementary script file for this assignment, **`TravelDB.sql`**.

***Database constraints:*** Table 1 lists all the constraints that should be created on the `TRAVEL` database.

*Note:* Constraints 7-12 must also make sure that their values are not null.

| No. | Constraint Name | Table.Column | Description |
|---|---|---|---|
| 1 | PK_CUST_ID | CUSTOMERS.customer_id | customer_id is the primary key of CUSTOMER |
| 2 | PK_STOP_ID | STOPS.stop_id | stop_id is the primary key of STOPS |
| 3 | PK_TAP_ID | TAPS.tap_id | tap_id is the primary key of TAPS |
| 4 | PK_FARES | FARES.zone_count and FARES.class_name | zone_count and class name are the (composite) primary key of FARES |
| 5 | FK_STOP_ID | TAPS.stop_id | TAPS.stop_id refers to STOPS |
| 6 | FK_CUST_ID | TAPS.customer_id | TAPS.customer_id refers to CUSTOMERS |
| 7 | CK_FARE_CLASS | FARES.class_name | class_name must be one of the following values: 'Adult', 'Concession' |
| 8 | CK_CUST_CLASS | CUSTOMERS.class_name | class_name must be one of the following values: 'Adult', 'Concession' |
| 9 | CK_FARE_AMOUNT | FARES.amount | amount must be a positive value |
| 10 | CK_STOP_ZONE | STOPS.zone | zone must be between 1 and 8 inclusive |
| 11 | CK_FARE_ZONE | FARES.zone_count | zone_count must be between 1 and 8 inclusive |
| 12 | CK_TAP_CHARGE | TAPS.charge | charge must be a non-negative value |
| 13 | CK_TIMESTAMP | TAPS.timestamp | timestamp must not be empty (not null) |
| 14 | CK_CUST_ID | TAPS.customer_id | customer_id must not be empty (not null) |
| 15 | CK_STOP_ID | TAPS.stop_id | stop_id must not be empty (not null) |

***Table 1:*** *Database constraints*

## Section 2 — Assignment Tasks

***Create and Populate Database:*** You need to execute the script file `TravelDB.sql` to create and populate your database before working on the following tasks. Wait until you see the message "Commit complete." It should take several seconds. The script will also attempt to drop related tables, sequences, and triggers.

## Task 1 — Constraints

### 1.1 — Constraint discovery

After running the script file, you will notice that only some of the constraints listed in Table 1 were created. Write an SQL statement to find out which constraints have been created on the four tables `CUSTOMER`, `FARES`, `STOPS` and `TAPS`.

### 1.2 — Constraint creation

Write the SQL statements to create all the missing constraints from Table 1.

## Task 2 — Triggers

### 2.1 — Sequence object

Assume that `tap_id` should be automatically populated when a new tap is made by a customer. Write an SQL statement to create a sequence object to generate values for this column. The sequence, named `TAP_ID_SEQ`, should start from 3,000 and increment by 1.

```
CREATE SEQUENCE "TAP_ID_SEQ"
INCREMENT BY 1 START WITH 3000;
```

### 2.2 — Sequence insertion

Write an SQL statement to create an Oracle trigger called `BI_TAP_ID` that binds the sequence object `TAP_ID_SEQ` to the `tap_id` column, i.e., the trigger populates values of `TAP_ID_SEQ` to the `tap_id` column when a new tap is made.

```
CREATE OR REPLACE TRIGGER "BI_TAP_ID" BEFORE
  INSERT ON "TAPS"
  FOR EACH ROW
BEGIN
   SELECT "TAP_ID_SEQ".NEXTVAL INTO :NEW.tap_id FROM DUAL;
END;
/
```

### 2.3 — Basic Fares

Write an Oracle trigger called `BI_BASIC_FARES` which calculates the fare for each trip a customer makes and fills out `Taps.charge` accordingly. A trip consists of a tap-on followed by a tap-off.

Fares are charged when tapping off, according to both the customer's fare `class` and the number of zones they have travelled through. The number of zones is calculated by taking the absolute difference of the tap-on and tap-off zone numbers, and then adding one. (When a customer taps on, a $0.00 charge is recorded.)

***Note:*** you can assume that customers always tap on and off correctly.

### 2.4 "Eight then half-price"

The transport department has introduced a frequent user discount: customers who have travelled at least eight times in the last seven days are entitled to half-price travel from the ninth trip onward. Write an Oracle trigger called `BI_EIGHT_HALF` to implement this functionality. (It should do the same thing as `BI_BASIC_FARES` when a customer is not eligible for half-price travel.)

> *Tip:* you can disable a trigger with `ALTER TRIGGER "TRIGGER_NAME" DISABLE;`

### 2.5 — Continuations

For network planning reasons, the transport department has withdrawn "eight then half-price" and instead introduced free continuations when a customer's journey involves multiple trips. Customers are eligible for a continuation fare when their most recent touch-on is less than one hour after their previous touch-off. In this case, they are charged only if their new trip involves new zones.

Write an Oracle trigger called `BI_CONTINUATIONS` to implement this behaviour. (It should do the same thing as `BI_BASIC_FARES` when a customer is not eligible for a continuation fare).

> In the following examples, all four people are in the same fare class; a one-zone fare is $2.00, a two-zone fare is $3.50 and a three-zone fare is $5.00.
>
> ***Example 1:*** Alice catches a bus from her street (in Zone 1), to the city (also in Zone 1). She is charged a one-zone fare ($2.00) when tapping off. A few minutes later, she then transfers onto a different bus which takes her to New Farm Park (also in Zone 1). She is charged $0.00 when she taps off again.
>
> ***Example 2:*** Bob catches a bus from his street (in Zone 3), to the train station (also in Zone 3). He is charged a one-zone fare ($2.00) when tapping off on the bus. A few minutes later, he taps-on to enter the station and catches a train to the city (Zone 1). A three-zone fare would normally be $5.00, but as Bob has already paid a one-zone fare, he is charged the difference ($3.00) when he taps off in the city.
>
> ***Example 3:*** Carol makes the same trip as Alice, except that instead of transferring immediately, she spent some time at a closing-down sale. Since it was more than an hour between tapping off in the city and tapping on again, she is charged a second one-zone fare when she taps off at the park.
>
> ***Example 4:*** Dave has some errands to run today. He first travels from his house (in Zone 2) to Bob's (Zone 3), then within an hour is heading to Alice's (Zone 1), then again within an hour is heading home. He is charged a two-zone fare ($3.50) when he taps off on Bob's street, then the difference between a two-zone and three-zone fare ($1.50) when he taps off at Alice's street, and finally $0.00 when he taps off back home.

## Section 3 — Deliverables and Marking Scheme

The project is due by **4:00PM, 8 September 2023.** Late submissions will be penalised unless you are approved for an extension (refer to Section 5.3 of the ECP).

You are required to turn in a single script file `studentID.sql` *(rename studentID)* that includes all your SQL statements. **Submit your script file on Blackboard** via the upload link *"SQL Script Submission".* Your script file should be in plain text format.

You must make sure that your script file can be executed on the EECS School Lab computers by the "@" command.

***Tip**:* Test this in SQL*Plus by first executing `TravelDB.sql`, then your script.

## Marking Scheme (30 Marks for 10%)

**Generative Artificial Intelligence tools cannot be used in this assignment submition.**

| Task | Marks | Marking Criteria |
|---|---|---|
| 1.1 | 1 | • Write only one SQL query.<br>• Find all the created constraints on the four tables (the result should exclude the constraints on any other tables) |
| 1.2 | 6 | • Write only one SQL statement for creating each constraint.<br>• Each constraint is created with the correct name and semantics. The correctness of the constraints will be tested using several INSERT statements. |
| 2.1 | 2 | • Sequence is created with the correct name and semantics. |
| 2.2 | 3 | • Each trigger is created with the correct name and without compilation error.<br>• The correctness of each trigger will be tested using several INSERT and SELECT statements.<br>• Write only one trigger-creation statement per task — no auxiliary VIEWs or similar should be created. |
| 2.3 | 4 | |
| 2.4 | 6 | |
| 2.5 | 8 | |

***Note**:* the triggers for Tasks 2.3, 2.4 and 2.5 will be tested independently of each other.

## Acknowledgements

Parts of `TravelDB.sql` are derived from the *South East Queensland—general transit feed specification* which is © the State of Queensland and licensed under *CC-BY 4.0.*

https://www.data.qld.gov.au/dataset/general-transit-feed-specification-gtfs-seq
https://creativecommons.org/licenses/by/4.0/