DATABASE

SPECIFICATIONS

Database Design and Optimisation
Ming Hsuan Chen
s4055813@student.rmit.edu.au

School of Computing Technologies, RMIT University

March 2025

Revision Sheet

Release No.	Date	Revision Description
Rev. 0	14/3/2025	Database Specifications Template and Checklist
Rev. 1	21/3/2025	Add the draft of ERD and describe all entities
Rev. 2	22/3/2025	Update the ERD v1 and roughly finish the relational schema in 3NF
Rev. 3	25/3/2025	Update the ERD v2 (add the RealTimeLocation entity)
Rev. 4	26/3/2025	Update the ERD v3 (aad is_free_zone contribute)
Rev. 5	28/3/2025	Finalize relational schema and normalize all tables to 3NF
Rev. 6	12/4/2025	Add table descriptions and define primary/foreign keys
Rev. 7	12/4/2025	Implement partitioning plan and initial index structure
Rev. 8	17/4/2025	Complete stored procedure touchOn() with error handling
Rev. 9	17/4/2025	Estimate storage size and update scalability section
Rev. 10	17/4/2025	Write explanation for partitioning and how to add partitions
Rev. 11	19/4/2025	Finalize error handling section with TRY-CATCH and IF-ELSE logic
Rev. 12	20/4/2025	Add system environment details and support software used
Rev. 13	20/4/2025	Review full report for grammar, structure, and final polish

DATABASE SPECIFICATIONS

TABLE OF CONTENTS

Page #

1.0 GENERAL INFORMATION 1-1
1.1 Purpose 1-1
1.2 Scope 1-1
1.3 System Overview 1-1
1.4 Project References 1-2
1.5 Acronyms and Abbreviations 1-2
1.6 Points of Contact 1-3
1.6.1 Information 1-3
1.6.2 Coordination 1-3
2.0 DATABASE IDENTIFICATION AND DESCRIPTION 2-1
2.1 Naming Conventions 2-1
2.2 Database Identification 2-1
2.3 Systems Using the Database 2-2
2.4 Relationship to Other Databases 2-2
2.5 Schema Information 2-2
2.5.1 Description 2-2
2.5.2 Physical Design 2-5
2.5.3 Physical Structure 2-5
2.6 Data Dictionary 2-6
2.7 Special Instructions 2-7
3.0 DATABASE ADMINISTRATIVE INFORMATION 3-1
3.1 Responsibility 3-1
3.2 System Information 3-1
3.2.1 Database Management System (DBMS) Configuration 3-1
3.2.2 Hardware Configuration 3-1
3.2.3 Database Software Utilities 3-1
3.2.4 Support Software Available for Maintaining Database 3-1
3.2.5 Security 3-2
3.3 Storage Requirements 3-2
3.4 Recovery 3-2
3.5 Partition/File Information 3-2
3.5.1 Content 3-3
3.5.2 Description 3-3
3.5.3 Partition/File Interdependencies 3-3
3.6 Database Interfaces 3-3
3.6.1 Description of Operational Implications 3-4
3.6.2 Description of Data Transfer Requirements 3-4
3.6.3 Description of Formats of Data 3-4
3.7 Error Handling 3-4
4.0 APPENDIX-DATA DICTIONARY 4-2

1.0 GENERAL INFORMATION

1.0 GENERAL INFORMATION

1.1 Purpose

This document explains the design and structure of the Myki Card Management database. The system simulates the use of transit cards in Victoria, recording basic information about each MyKi card, passenger swipes, trip status, payment method, and balance changes.

Because this information grows over time and is mostly related to time, place, and money, the database needs to be designed for speed and storage efficiency. The main purpose of this report is to illustrate how I can improve the performance of data query and storage through rational table design, field setting, primary and secondary key associations, as well as Index and Partition.

Overall, this document will clearly introduce the design logic and structure of the entire database, and explain how I balanced data integrity and system performance during the design process.

1.2 Scope

The scope of this database system is to simulate the MyKi public transportation card operation. The system is mainly responsible for managing the following information:

- Basic information of each passenger
- Status and balance of each MyKi card
- Detailed records of each card swipe (Touch On / Touch Off)
- Start and end time, location and completion status of each journey
- Type of ticket used (MyKi Money or MyKi Pass) and payment result

The system focuses on recording and tracking all data related to “card action” and does not cover other administrative, vehicle scheduling or route management functions. The data mainly focuses on the records generated when passengers use the card to travel on public transportation, and supports subsequent inquiries and processing.

1.3 System Overview

This database system is designed to simulate the basic functions of a transit card swipe system, which records passenger card information, swipe behavior, transaction history and trip status. The core objective of the system is to ensure that every swipe is quickly queried and correctly recorded, and that fares can be accurately determined based on different circumstances (e.g., whether a regular ticket is available, whether it is within two hours).

- **Responsible organization:**
  Individual project for ISYS1102 – Database Applications, RMIT University
- **System name:** Myki Card Management System
- **System code:** MYKI-2025
- **System category:** Major application
- **Operational status:** Under development
- **System environment and special conditions:**
  This database was built using Azure Data Studio and Microsoft SQL Server. I used SQL commands to create tables, run queries, and use T-SQL to create the stored procedure. To keep the system fast when the data grows, I used partitioning and added indexes.

1.4 Project References

A list of the references that were used in preparation of this document:

- Public Transport Victoria (PTV)
- Course material from ISYS1102 Database Applications – RMIT University
- Course material from ISYS1055 Database Concepts – RMIT University

1.5 Acronyms and Abbreviations

A list of the acronyms and abbreviations used in this document:

Acronym	Full Term	Explanation
SQL	Structured Query Language	A language we use to work with data in a database.
T-SQL	Transact-SQL	A Microsoft version of SQL used in SQL Server.
DB	Database	A place to store and manage data.
PK	Primary Key	A column that makes each row in a table unique.
FK	Foreign Key	A column that connects two tables together.
ID	Identifier	A number or code used to identify something.
TXN	Transaction	One action in the system, like a card tap or payment.
ERD	Entity-Relationship Diagram	A diagram that shows how all the tables are related to each other.
PTV	Public Transport Victoria	The organization that manages public transport in Victoria, Australia.
dob	Date of Birth	The day someone was born

1.6 Points of Contact

1.6.1 Information

If someone has a problem with the database, this is the place to contact:

Type	Name	Email
Developer	Ming Hsuan Chen	S4055813@student.rimt.edu.au

1.6.2 Coordination

This database project was designed and completed by me. There was no collaboration with other teams or departments. Therefore, no extra coordination was needed during the development process.

2.0 DATABASE IDENTIFICATION AND DESCRIPTION

2.0 DATABASE IDENTIFICATION AND DESCRIPTION

2.1 Naming Conventions

The naming conventions help to keep the database organized and easy to understand.
The Myki Card Management System follows clear naming conventions to make the database easy to read and understand.

Logical Naming Conventions:
- Table names use PascalCase. For example, *Passenger, MykiTransaction, Station.*
- Column names use snake_case. For example, *passenger_id, station_name, txn_status*.
- Names are short, clear, and in plain English.
- No spaces or special characters are used in names.

Physical Naming Conventions:
- Primary keys end with ***_id*** or ***_no***.
- Underline indicates a primary key, and an asterisk (*) indicates a foreign key
- Foreign keys match the primary key of the related table.
- All table names are singular nouns.

These conventions help to keep the database structure simple and consistent.

2.2 Database Identification

The database is called MykiCardDB, and it contains 9 tables that are related to each other, and each table has a specific function. The details are as follow

Table	Purpose
Passenger	Stores basic passenger information, such as name, date of birth, and contact details.
MyKiCard	Stores information about each card, including balance, expiry date, card type, and the passenger who owns it.
MykiPass	If a card has a valid pass, it is recorded here. Includes pass type, valid zones, and active period.
MykiTransaction	Records each card transaction, including transaction time, transaction status, payment method, and amount.
Journey	Tracks the journey status, such as touch on / off time, destination, and if the journey is completed.
TouchEvent	Stores detailed records of every card scan, including time, event type, status, and fare.
Scanner	Stores information about scanner device, such as its location, whether it is in a station or on a vehicle, and its zone.
Station	Stores station details, including the name and type (e.g., train station or bus stop).
Vehicle	If a scanner is installed on a vehicle, this table records which vehicle it is, its type, and route name.

2.3 Systems Using the Database

This database was developed using Azure Data Studio on a macOS device. However, the database itself is hosted on a remote Microsoft SQL Server environment provided by RMIT University.

Below are the system details:

- **Development Tool:** Azure Data Studio
- **Tool Version:** 1.46.1 (on macOS)
- **Database Platform:** Microsoft SQL Server
- **SQL Server Version:** SQL Server 2022 Standard Edition (16.0.4165.4)
- **Host System:** Windows Server 2019 Datacenter (64-bit, Hypervisor)
- **IAS System Code:** N/A

This setup was used for writing, executing, and testing all SQL queries and stored procedures in this project.

2.4 Relationship to Other Databases

This database is designed as a independent system and does not connect or integrate with other databases.

2.5 Schema Information

Describe the overall structure in the schema or other global definition of the database.

2.5.1 Description

This database consists of nine main relational tables, each serving a specific purpose in tracking and managing the MyKi transport card system.

- **Schema Name:** myki_system
- **File Type and Structure:** All data is stored in relational database tables.
- **Description Language:** SQL (Structured Query Language)
- **Naming Convention:**
    1. Table names use **PascalCase** (e.g., TouchEvent, Transaction)
    2. Column names use **snake_case** (e.g., card_no, txn_time)

3. Names are short but meaningful for clarity and consistency.

- **Relational Database Schema:**

Passenger (passenger_id, passenger_name, dob, email, phone_num, address)
MyKiCard (card_no, myki_money_balance, expiry_date, card_type, passenger_id)*
MykiPass (pass_id, pass_type, zone_coverage, start_date, end_date, status, card_no)*
MykiTransaction (txn_no, txn_type, txn_time, txn_status, payment_type, amount, card_no, scanner_id*)*
Scanner (scanner_id, scanner_type, gps_latitude, gps_longitude, zone, vehicle_id, station_id*)*
Station (station_id, station_name, station_type)
Vehicle (vehicle_id, vehicle_route_name, vehicle_type)
Journey (journey_id, touch_on_time, touch_on_station_id, touch_off_time, touch_off_station_id, fare_charged, is_complete, fare_type, txn_no, scanner_id*)*
TouchEvent (event_id, event_type, event_time, event_status, fare_charged, card_no, scanner_id*, journey_id*, txn_no*)*

- **Table Design Details:**

1. **Passenger**:
    - Stores basic passenger information, such as name, date of birth, email, phone number, and address.
    - This table is linked to MyKiCard to show which cards belong to which passenger.

2. **MyKiCard:**
    - Stores card details such as balance, expiry date, and card type.
    - One passenger can have multiple cards.

3. **MykiPass:**
    - Stores information about travel passes added to a card, such as pass type, valid zones, start and end dates, and current status.
    - A card can have multiple passes during its lifetime.

4. **MykiTransaction:**
    - Stores each transaction record, such as touch on, touch off, top-ups, or failed attempts. Includes details like time, status, amount, and payment method.
    - All journeys and events are linked to transactions in this table.

5. **Journey:**
    - Tracks the start and end of a trip. Includes touch on and touch off time, stations, fare charged, and if the journey is complete.

- A journey starts with a Touch On. If there is no Touch Off, the journey is marked as incomplete. This follows how the real MyKi system works.

6. TouchEvent:

- Stores every card scan, including event time, type (Touch n or off), result, fare charged, and related data.
- Special Feature: This table uses partitioning by event_time to improve performance when querying by date.

7. Station:

- Stores station information, such as station name and station type (e.g., train station, tram stop).
- Used by the Scanner and Journey tables to track travel locations.

8. Vehicle:

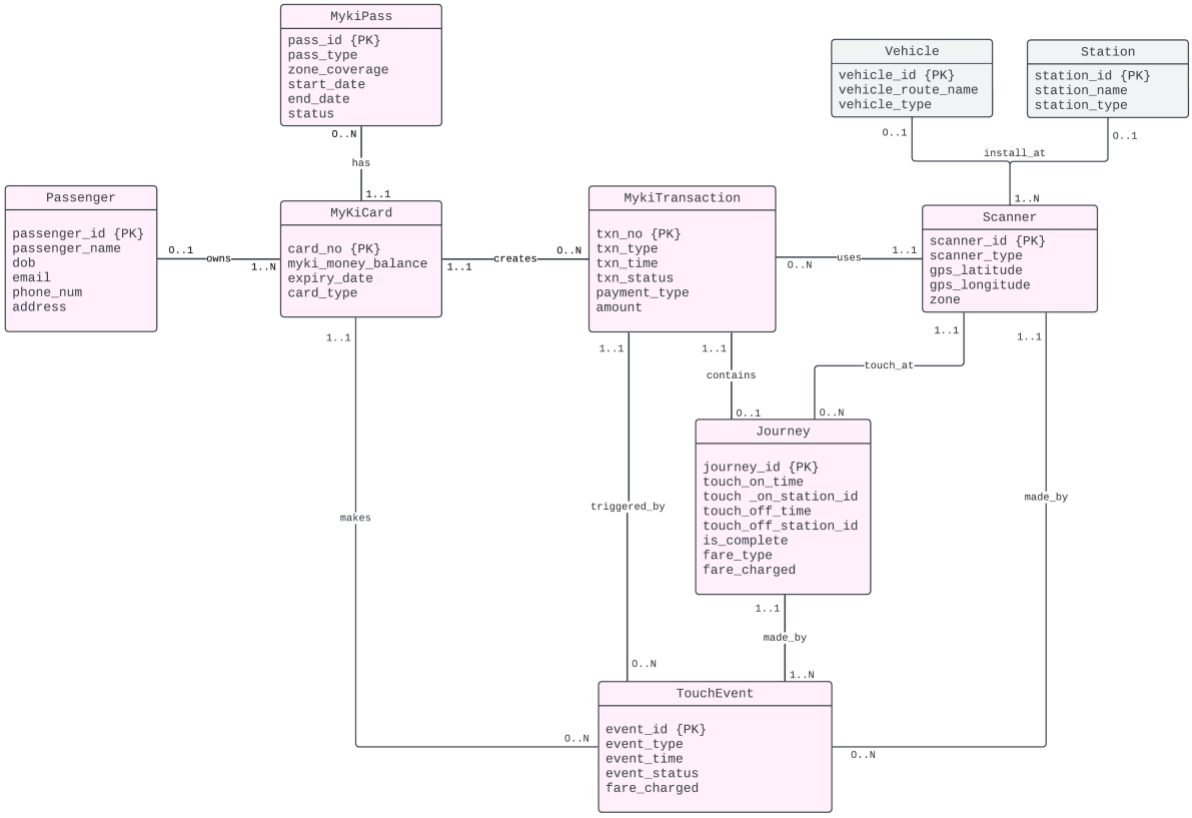
- Stores vehicle details for mobile scanners, including vehicle ID, route name, and type (e.g., bus, tram).
- Connected to Scanner to indicate which vehicle a scanner is on.

9. Scanner:

- Stores data about each scanner device. Includes type (fixed or mobile), GPS location, and zone.
- To simulate real-time location, each scanner is assumed to have built-in GPS. Therefore, a separate real-time table is not used.

2.5.2 Physical Design

Entity-Relationship Diagram:



This ERD shows the primary key and foreign key relationships for each table. This makes the data flow clear at a glance. In addition, TouchEvent is designed as a partitioned table, which makes the querying of data more efficient.

2.5.3 Physical Structure

To make sure the database can keep running smoothly in the long term, I added some optimisation strategies. These help the system stay fast and stable even when the amount of data gets large.

- **Optimisation Methods:** My optimisation design focuses on: using partition on TouchEvent to manage large amounts of data, using indexes to speed up checks on important columns and improving performance for tables that are read often. So I choose partition and index together to help the system perform better.

    1. **Partitioning:** Faster queries for TouchEvent

        The TouchEvent table records every card scan. Over time, this table will have a lot of data.If we query from thousands of records every time, it will be slow. So I used partitioning by

event_time. Each month is stored in its own section. This way, when we query card scans from a specific time period, SQL Server only searches the matching partition, not the whole table.

Example:

```
CREATE PARTITION FUNCTION pf_TouchEventRange (DATETIME)
AS RANGE RIGHT FOR VALUES (
'2025-01-01', '2025-02-01', '2025-03-01', '2025-04-01'
);
```

If we only search for data in April, the system will only look in the April partition. This makes the query much faster.

2. **Indexing**: Faster queries for important columns

- Check if Myki Pass is valid:

  ```
  CREATE INDEX idx_pass_validity ON MykiPass(card_no, status, start_date,
  end_date);
  ```

  This helps the system quickly check if a card has a valid pass when Touch On happens.

- Find the most recent Touch On:

  ```
  CREATE INDEX idx_touch_recent ON TouchEvent(card_no, event_time,
  event_type);
  ```

  This helps the system find the last scan time, to check if it was within two hours.

- Check if the last journey is completed:

  ```
  CREATE INDEX idx_journey_txn ON Journey(txn_no, is_complete, touch_on_time);
  ```

  This prevents the user from starting a new trip if the last one is not finished.

- Check card status and balance:

  ```
  CREATE INDEX idx_card_balance_check ON MyKiCard(card_no);
  ```

  This speeds up checking if the card is still valid and has enough balance. Even though *card_no* is already a primary key, this index clearly shows it is used often in queries.

2.6 Data Dictionary

The data dictionary lists all the tables and fields used in the system. It includes the field name, data type, and a short description of what the data means. The full data dictionary is attached at the end of this document as Appendix.

2.7 Special Instructions

This system is designed to simulate the Myki card process, especially the Touch On step.
Every time a user scans their card, the system automatically checks and processes the action.

- *txn_status* and *event_status*: automatically determined by the system. For example, if the card is expired or the balance is not enough, these two fields will show Failed and vice versa for Success.

- *txn_time* and *event_time:* automatically recorded by the system at the moment the transaction occurs.

The logic of these fields is written in the Stored Procedure *touchOn().* When the user touches on an action, this procedure will automatically handle it for us, no manual operation is needed.

The overall goal is to automate the entire system, minimize errors, and make checking or tracking information clearer.

3.0 DATABASE ADMINISTRATIVE INFORMATION

3.0 DATABASE ADMINISTRATIVE INFORMATION

3.1 Responsibility

Since this is an individual assignment, I was responsible for all parts of the database. I acted as the database administrator, system administrator, and security administrator during the development and testing.

3.2 System Information

Document the Database Management System configuration, hardware configuration, database software utilities, and any support software used.

3.2.1 Database Management System (DBMS) Configuration

The database is hosted on a remote Microsoft SQL Server 2022, running on Windows Server 2019 Datacenter. I used Azure Data Studio on macOS to connect to the server and manage the database using T-SQL commands.

3.2.2 Hardware Configuration

I used Azure Data Studio on my Mac to connect to the RMIT SQL Server. The database runs on a remote server managed by the university.

3.2.3 Database Software Utilities

I used **Azure Data Studio** to manage and test the database.

It helped me with the following:

- Writing and running T-SQL queries
- Creating and modifying database tables
- Inserting and editing data in tables
- Viewing the results of SQL queries

This tool was useful for both development and testing tasks.

3.2.4 Support Software Available for Maintaining Database

Besides Azure Data Studio, I also used the following tools to help with this project:

- **Lucidchart (web-based, latest version as of April 2025)**
  I used it to draw the ERD using UML notation.

- **Microsoft Word (Version 16.96):**
  I used Word to write this report, including all the sections and design explanations.

3.2.5 Security

This is a standalone and simulated system. There is no access control or encryption. Only one person is using the system, so login and user roles are not needed.

3.3 Storage Requirements

This database is made to simulate a public transport card system. In the future, I expect to store many TouchEvent records, along with passenger, card, transaction, and pass data. That is, there may be more than 10,000 records. This is not very large, but if the system checks every row in the table each time, it can still become slow.

- **What happens if the system runs for years**

  If one TouchEvent row takes about 150 bytes, and the system adds 10,000 rows per day:

  That is 1.5 MB per day
  In one year: 365 × 1.5 MB = about 547.5 MB

  This is still not very big, but after three or five years, there could be over one million rows. This will make the system slower.

- **What I did to prepare for more data**

  To help the system work well even when the data becomes large, I used two methods:

  1. Partitioning the TouchEvent table:

     The TouchEvent table keeps growing because each scan adds one row. To keep it fast, I used partitioning. I used event_time to split the data by month. This way, when we search for data in a specific time range, the system only looks in that month's data, not the whole table.

  2. Creating indexes:

     I also added indexes to the columns that are used a lot. Indexes help the system find data faster. Without indexes, the system may need to look at every row, which takes more time.

3.4 Recovery

If the system fails, I can restore the database by running the original .sql script. This script includes all table definitions, relationships, and sample data. I can use Azure Data Studio to execute the script and recreate the schema and data.

3.5 Partition/File Information

3.5.1 Content

In this database, there is only one table that is designed as a Partition Table, which is TouchEvent. It's because this table will record every card-touching action, which will generate a lot of data in the future. I use *event_time* as the basis for partitioning, and divide the whole table by month. Each partition still contains all the fields of the TouchEvent, but they are partitioned according to the timestamp of the *event_time*.

3.5.2 Description

This partitioning uses event_time as the condition and splits the data by month using *RANGE RIGHT*. The partition function is defined as:

```
PARTITION FUNCTION pf_TouchEventRange (DATETIME)
AS RANGE RIGHT FOR VALUES (
    '2025-01-01',
    '2025-02-01',
    '2025-03-01',
    '2025-04-01'
);
```

This means:

- Data with event_time before '2025-01-01' goes to the first partition.
- Data between '2025-01-01' and '2025-02-01' goes to the second partition.
- Data between '2025-02-01' and '2025-03-01' goes to the third partition.
- And so on.

This method helps the system search only one partition when looking for data in a specific month. It makes the query faster and more efficient.

3.5.3 Partition/File Interdependencies

Each partition is like a separate section inside the table. They do not affect each other. When we search for card scan records in a specific time range, SQL Server only looks in the matching partition.

For example:

```
SELECT * FROM TouchEvent
WHERE event_time BETWEEN '2025-03-01' AND '2025-03-31'
```

This query only searches in the third partition. It does not read data from other partitions. This design can make queries much faster. It will be very helpful in the future when there are thousands of card scan records.

3.6 Database Interfaces

This database is operated within Azure Data Studio and is not connected to any other external system.

3.6.1 Description of Operational Implications

Since this is a simulation system, there is no connection to other platforms and no user login, so there are no security issues at this time.

3.6.2 Description of Data Transfer Requirements

There is no import or export function at this time. I use *INSERT* to write all the data directly into the table

3.6.3 Description of Formats of Data

I have standardized the format when designing the fields, for example:

- *event_time, start_date* are in *DATETIME* format.
- Amount fields like *amount*, *fare_charged* are in *DECIMAL(6,2),* which can be kept to two decimal places.

3.7 Error Handling

I used both *TRY-CATCH* and *IF-ELSE* to handle different types of errors. This helps the system run smoothly in many situations, avoid writing wrong data, and makes it easier for me to find problems during testing.

1. System Error Handling (Using *TRY-CATCH*)

   I used the *TRY-CATCH* structure in SQL Server to handle system-level errors. The *touchOn()* stored procedure is split into three main steps. Each step is wrapped in its own *TRY-CATCH* block. This design has some advantages:

   - If one step has an error, the whole process will not stop.
   - The system goes into the *CATCH* block and prints an error message using *ERROR_MESSAGE()*.
   - Can easily see which step caused the error.

   These kinds of errors include things like: wrong column names, data type conversion errors, missing tables

2. Logical Error Handling (Using *IF-ELSE* + *PRINT* + *RETURN*)

   I also added logic checks to stop the process when something is not correct, even if it's not a system error. These checks use IF conditions. If a condition is not met, the system shows a message using *PRINT* and stops the process using *RETURN*. This is not an error, it is a normal part of the process control. Here are the conditions I check:

   - Is the card expired?
   - Does the card have a valid Myki Pass?
   - Is there enough balance?
   - Was the last scan within two hours?

APPENDIX-DATA DICTIONARY

4.0 APPENDIX-DATA DICTIONARY

Passenger

Column Name	Data Type	Description
passenger_id	INT	Unique ID of the passenger
passenger_name	VARCHAR(100)	Full name of the passenger
dob	DATE	Date of birth
email	VARCHAR(100)	Contact email
phone_num	VARCHAR(20)	Phone number
address	VARCHAR(200)	Address of the passenger

MyKiCard

Column Name	Data Type	Description
card_no	INT	Unique ID of the MyKi card
myki_money_balance	DECIMAL(6,2)	Current stored money on the card
expiry_date	DATE	Card expiry date
card_type	VARCHAR(20)	Type of card (Adult/Concession)
passenger_id	INT	Foreign key to Passenger

MykiPass

Column Name	Data Type	Description
pass_id	INT	Unique ID of the Myki Pass
pass_type	VARCHAR(20)	Type of pass (e.g. 7-day, monthly)
zone_coverage	VARCHAR(20)	Zone covered by this pass
start_date	DATE	Start date of the pass
end_date	DATE	End date of the pass
status	VARCHAR(20)	Pass status (e.g. Active, Expired)
card_no	INT	Foreign key to MyKiCard

MykiTransaction

Column Name	Data Type	Description
txn_no	INT	Unique ID of the transaction
txn_type	VARCHAR(20)	Type of transaction (TouchOn/TouchOff/TopUp)
txn_time	DATETIME	Time of the transaction
txn_status	VARCHAR(20)	Status of transaction (Success/Fail)
payment_type	VARCHAR(20)	Method used (MykiMoney/MykiPass)
amount	DECIMAL(6,2)	Amount charged (if any)
card_no	INT	Foreign key to MyKiCard
scanner_id	INT	Foreign key to Scanner

Journey

Column Name	Data Type	Description
journey_id	INT	Unique ID of the journey

touch_on_time	DATETIME	Start time of journey
touch_on_station_id	INT	Where the journey started
touch_off_time	DATETIME	End time of journey
touch_off_station_id	INT	Where the journey ended
fare_charged	DECIMAL(6,2)	Fare paid
is_complete	BIT	Whether journey was completed
fare_type	VARCHAR(20)	Pass or MykiMoney
txn_no	INT	Foreign key to MykiTransaction
scanner_id	INT	Foreign key to Scanner

TouchEvent

Column Name	Data Type	Description
event_id	INT	Auto-incremented ID for the event
event_time	DATETIME	Time when the touch happened
event_type	VARCHAR(20)	TouchOn / TouchOff
event_status	VARCHAR(20)	Success / Fail
fare_charged	DECIMAL(6,2)	Amount charged for this event
card_no	INT	Foreign key to MyKiCard
scanner_id	INT	Foreign key to Scanner
journey_id	INT	Foreign key to Journey
txn_no	INT	Foreign key to MykiTransaction

Scanner

Column Name	Data Type	Description
scanner_id	INT	Unique ID of the scanner
scanner_type	VARCHAR(20)	OnBoard or Platform
gps_latitude	DECIMAL(9,6)	Latitude of the scanner
gps_longitude	DECIMAL(9,6)	Longitude of the scanner
zone	VARCHAR(20)	Fare zone (e.g. Zone 1, Zone 2)
vehicle_id	INT	Foreign key to Vehicle (if onboard)
station_id	INT	Foreign key to Station (if at station)

Station

Column Name	Data Type	Description
station_id	INT	Unique ID of station
station_name	VARCHAR(100)	Station name
station_type	VARCHAR(20)	Train / Tram / Bus

Vehicle

Column Name	Data Type	Description
vehicle_id	INT	Unique ID of the vehicle
vehicle_route_name	VARCHAR(50)	Name of the route (e.g. Tram 19)
vehicle_type	VARCHAR(20)	Tram / Bus / Train