Metro Ticket Generating System

in ServiceNow

Final Report

1. INTRODUCTION

1.1 Project Overview

The **Metro Ticket Generating System** is a ServiceNow-based solution designed to modernize the urban transit lifecycle by replacing manual workflows with digital automation. By leveraging the **Service Catalog** for centralized intake and **Flow Designer** for orchestration, the project streamlines ticketing and smart card management, significantly reducing turnaround time.

1.2 Purpose

- Automate request intake, station routing, and ticket generation in QR code.
- Standardize digital ticketing and smart card recharge workflows.
- Improve visibility, reporting, and the overall commuter experience through real-time fare previews.

2. IDEATION PHASE

2.1 Problem Statement

Manual handling of metro ticketing causes:

- Delays in ticket fulfilment and long queues.
- Errors due to incomplete or inconsistent passenger information.
- Lack of standardized tracking for smart card recharges.
- Poor visibility for commuters regarding their journey status.

2.2 Empathy Map Canvas

Stakeholder	Think & Feel	See & Hear	Say & Do	Pain Points
Commuters	Frustrated with ticket delays	System inconsistencies	Submits ticket requests	Long wait times, no updates
IT Admins	Wants data integrity	Complex configurations	Maintains workflows	Manual interventions, data errors

Stakeholder	Think & Feel	See & Hear	Say & Do	Pain Points
Transit Team	Needs compliance	Missing info	Executes station tasks	Human errors, inconsistent data

3. REQUIREMENT ANALYSIS

3.1 Customer Journey Map

1. **Commuter** logs in via the Service Portal.
2. Selects the **"Book a Metro Ticket"** catalog item.
3. Fills a dynamic form with auto-populated fare fields based on station selection.
4. **Flow Designer** triggers the workflow immediately upon submission.
5. A record is created in the **Metro Database** (u_metro_station_s_details).
6. A unique **QR Ticket** is rendered via an SP Modal popup for immediate use.

3.2 Solution Requirements

- **Service Catalog**: Creation of the “Book a Metro Ticket” item with dynamic forms.
- **Workflow Automation**: Flow Designer logic to handle data storage and fulfilment.
- **Custom Table**: A backend repository (u_metro_station_s_details) for journey and recharge records are stored.
- **Security**: Role-based access control and ACLs to protect passenger data.

3.3 Data Flow Diagram

Service Portal Form > Flow Designer Trigger > Get Catalog Variables > Create Record in Metro Database > Generate QR Code > Ticket Fulfilled.

3.4 Technology Stack

- **Platform**: ServiceNow PDI.
- **Tools**: Flow Designer, Catalog UI Policies, and Client Scripts.
- **Database**: Custom Table (u_metro_station_s_details).
- **Languages**: Minimal scripting (Client Scripts for auto-populating fare and generating QR).

4. PROJECT DESIGN

4.1 Problem Solution Fit

Manual request handling was replaced by an automated workflow, ensuring streamlined station routing, real-time fare accuracy, and uncompromised data integrity.

4.2 Proposed Solution

- **Custom Table**: u_metro_station_s_details to store station names, card numbers, and recharge amounts.
- **Flow Designer**: Automates task creation and record updates upon ticket submission.
- **Dynamic Forms**: Service Portal forms featuring auto-populated fare fields and real-time station selection.

5. SOURCE CODE: CATALOG CLIENT SCRIPTS

5.1 Script: Automatic Price/Fare Calculation

This onChange script updates the "Amount for Single Journey" variable dynamically based on the number of passengers and journey logic.

Source Code

```
function onChange(control, oldValue, newValue, isLoading) {
    if (isLoading || newValue == '') {
        return;
    }
    var source = g_form.getValue('starting_from');
    var destination = g_form.getValue('going_to');
   var numPassengers = parseInt(g_form.getValue('no_of_passengers'))
|| 0;
    // Check if passengers is 0 or empty to prevent errors
    if (numPassengers <= 0) {
        alert('Please enter the number of passengers first.');
        return;
    }
```

```
    // newValue == 1 (Single Journey), newValue == 2 (Return Journey)
    // Logic for Ameerpet to L B nagar
        if (source == "9659786293c23ad09a12f8eddd03d6cf" &&
destination == 'a0793c6693823ad09a12f8eddd03d6f8') {
        updateFare(50);
    }
    // Logic for Ameerpet to Miyapur
      else if (source == "9659786293c23ad09a12f8eddd03d6cf" &&
destination == '7989fc6293c23ad09a12f8eddd03d656') {
        updateFare(60);
    }
    // Logic for Ameerpet to Nagole
      else if (source == "9659786293c23ad09a12f8eddd03d6cf" &&
destination == '0d99342a93823ad09a12f8eddd03d62f') {
        updateFare(55);
    }
      else if (source == "9659786293c23ad09a12f8eddd03d6cf" &&
destination == '2b99b0a293c23ad09a12f8eddd03d674') {
        updateFare(40);
    }
    // Helper function to set values based on journey type
```

```
    function updateFare(basePrice) {
        if (newValue == '1') { // Single
            g_form.setValue('amount_for_single_journey', numPassengers * basePrice);
            g_form.clearValue('amount_including_return');
        } else if (newValue == '2') { // Return
            g_form.clearValue('amount_for_single_journey');
            // Fixed the math: 2 * (passengers * price)
                        g_form.setValue('amount_including_return', 2 * (numPassengers * basePrice));
        }
    }
}
```

5.2 Script: QR Code Generation (onSubmit)

This script captures the record sys_id and renders a scannable QR ticket via an spModal popup.

Source Code

```
function onSubmit() {
    var work = g_form.getValue('what_do_you_want_to_do_today');
    if (work == '2') {
```

```
var sysId = g_form.getUniqueValue();
// First submit → create record
if (!sysId) {
    return true;
}
//  DATA THAT WILL BE STORED INSIDE QR
var qrData =
    'Metro Ticket' + '\n' +
    'Request ID: ' + sysId;
var qrURL =
    'https://api.qrserver.com/v1/create-qr-code/?size=320x320&margin=10&data=' +
    encodeURIComponent(qrData);
spModal.open({
    title: 'Scan Your QR Here',
    message:
        '<div style="display:flex;justify-content:center;align-items:center;padding:20px;">' +
            '<div style="border:3px solid #e0e0e0;padding:15px;border-radius:6px;">' +
```

```
                    '<img src="' + qrURL + '"
style="width:280px;height:280px;" />' +
                '</div>' +
            '</div>',
        buttons: [] ,    // no OK / Cancel
        size: 'md'});
         return false;
    }
   return true;}
```

5.3 Catalog Client Script Configuration for Field Validation

Used Catalog client Script for the quantity field Validation, when the user enters characters instead of numbers.

Source Code

```
function onChange(control, oldValue, newValue, isLoading) {
  if (isLoading || newValue == '') {
    return;
  }
   if (isNaN(newValue)) {
      alert('please enter the values in number format only');}}
```

6. FLOW DESIGN

6.1 Backend Automation Logic

The "Metro Project" flow is the central engine of the system. It is designed with a "Low-Code First" approach, utilizing ServiceNow's native Flow Designer to manage data integrity and record fulfilment.

- **Trigger:** The process initiates immediately upon the submission of the "Book a Metro Ticket" Service Catalog item.
- **Action 1:** Get Catalog Variables: The flow retrieves user-provided data, such as station references, passenger counts, and smart card details.
- **Action 2:** Create Record: The system maps these variables directly to the u_metro_station_s_details custom table to maintain a persistent audit trail of the transaction.

6.2 Workflow Execution and Validation

To ensure reliability, every flow execution is tracked via Execution Details, providing a transparent view of the automation stages.

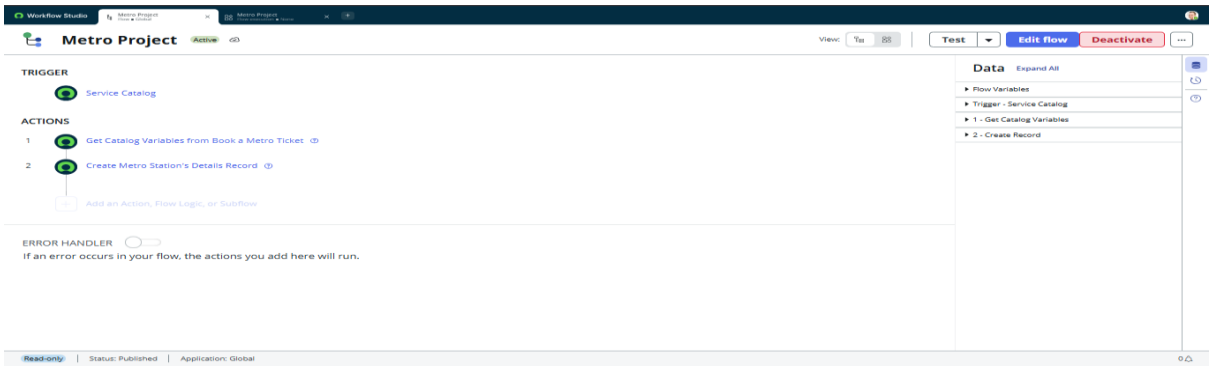


Figure 6.2.1: Flow Designer Workflow Architecture

A comprehensive view of the "Metro Project" flow logic, demonstrating the sequence from the initial Service Catalog trigger to the final creation of the database record.

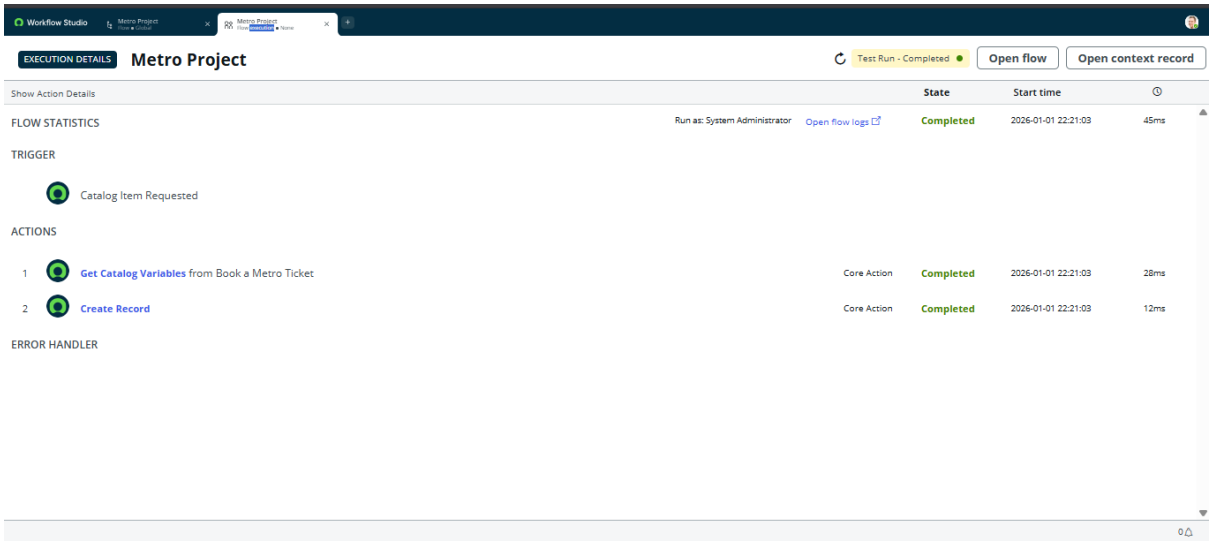


Figure 6.2.2: Flow Execution Success Log

A technical screenshot of the Flow Designer Execution Details showing a "Completed" status. This validates that the system successfully captured catalog variables and populated the backend table without errors.

7. RESULTS (OUTPUT SCREENSHOTS)

The following screenshots validate the dynamic behaviour of the Service Catalog, the successful generation of digital tickets, and the integrity of the backend database storage.

7.1 Dynamic Form Interface (User Selection Path)

The Service Catalog form utilizes **UI Policies** and **Catalog Client Scripts** to adjust the interface based on the user's intent, ensuring a clean and focused experience.

- **None (Default State)**: Upon initial load, the form displays only the primary question "What do you want to do Today?" to prevent user confusion.

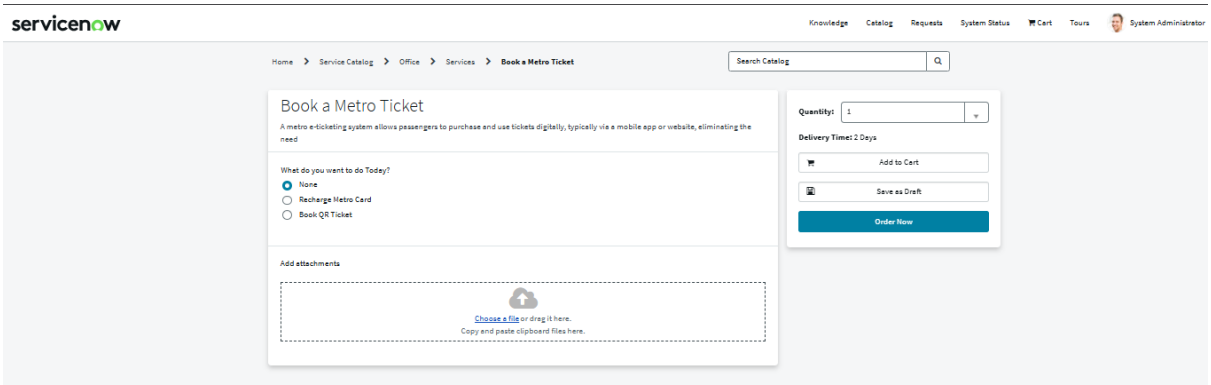


Figure 7.1: Default State (None Selected)

- Shows the initial form layout where journey-specific and card-specific variables are hidden to provide a clean user interface and reduce initial input complexity.

- **Book QR Ticket Path**: Selecting this option reveals journey-specific variables.

- **Station References**: Users select "Starting From" (e.g., Ameerpet) and "Going To" (e.g., L B nagar).
- **Auto-Calculated Fare**: Based on the selection and passenger count, the "Amount for Single Journey" auto-populates (e.g., 120) in real-time.

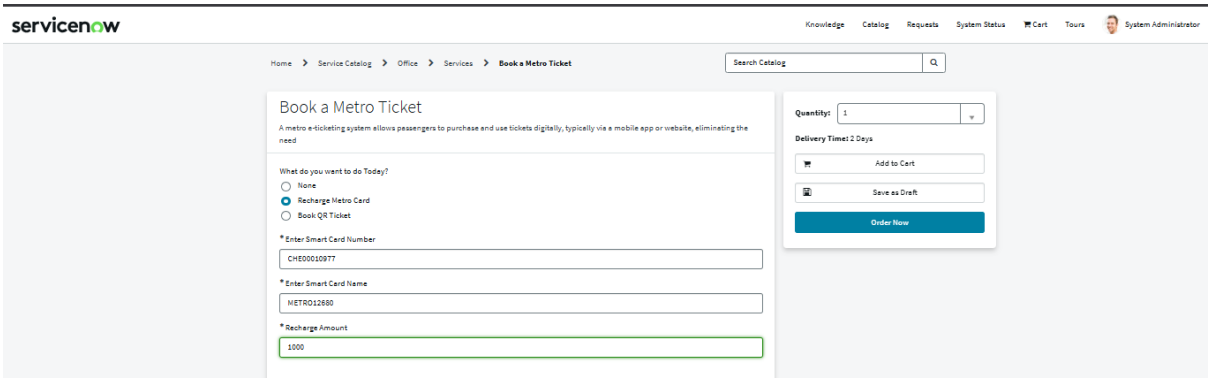


Figure 7.2: Dynamic UI for Smart Card Recharge

- Demonstrates the dynamic reveal of the **Smart Card Number**, **Smart Card Name**, and **Recharge Amount** variables. These fields appear only when "Recharge Metro Card" is selected, while journey station fields remain hidden.

- **Recharge Metro Card Path**: Selecting this option hides journey fields and displays card-specific variables.
  - **Card Details**: Users enter the "Smart Card Number," "Smart Card Name," and the desired "Recharge Amount".

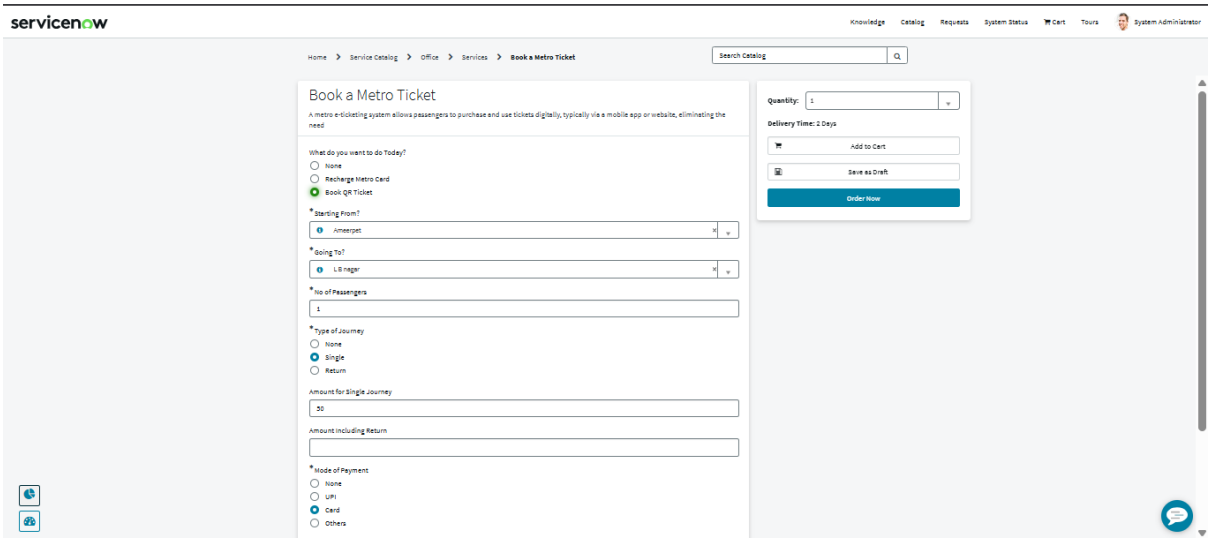


Figure 7.3: Dynamic UI for QR Ticket Booking

- Displays the journey variables, including Starting From, Going To, and No of Passengers. This view highlights the real-time fare calculation (e.g., 100) based on the user's station selection.

7.2 Output Generation: Digital QR Ticket

For commuters booking a journey, the system provides instant fulfilment through a digital interface.

The resulting output of the **Book QR Ticket** path. This shows the spModal popup triggered by the onSubmit script, rendering a unique scannable QR code based on the record's sys_id for immediate passenger use.

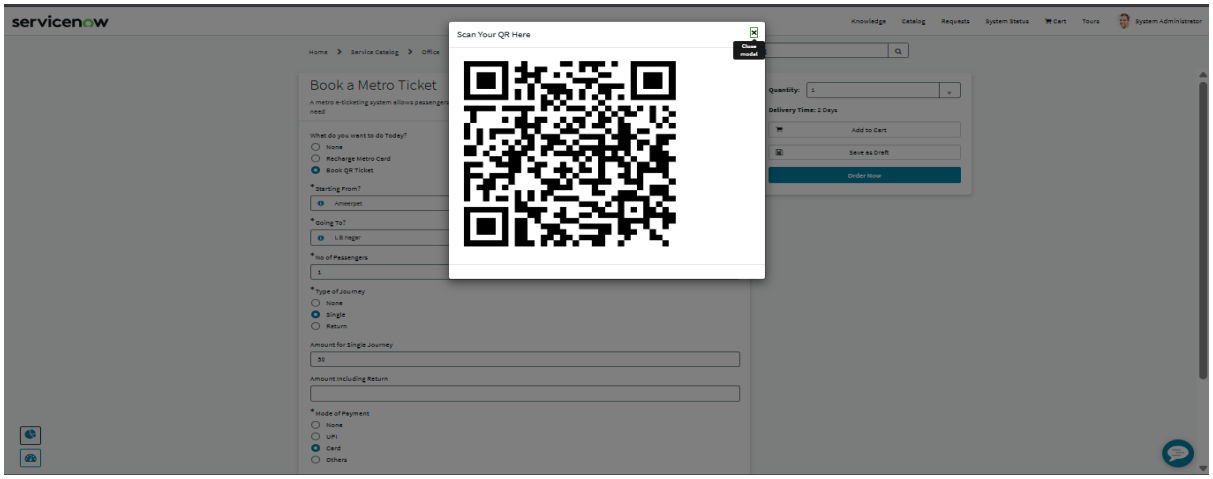


Figure 7.2.1: Instant Digital QR Ticket Generation

7.3 Backend Verification: Data Storage

The system utilizes a structured **Flow Designer** orchestration layer to ensure that every transit transaction is automatically captured and logged within the **u_metro_station_s_details** custom table for rigorous audit and tracking purposes.

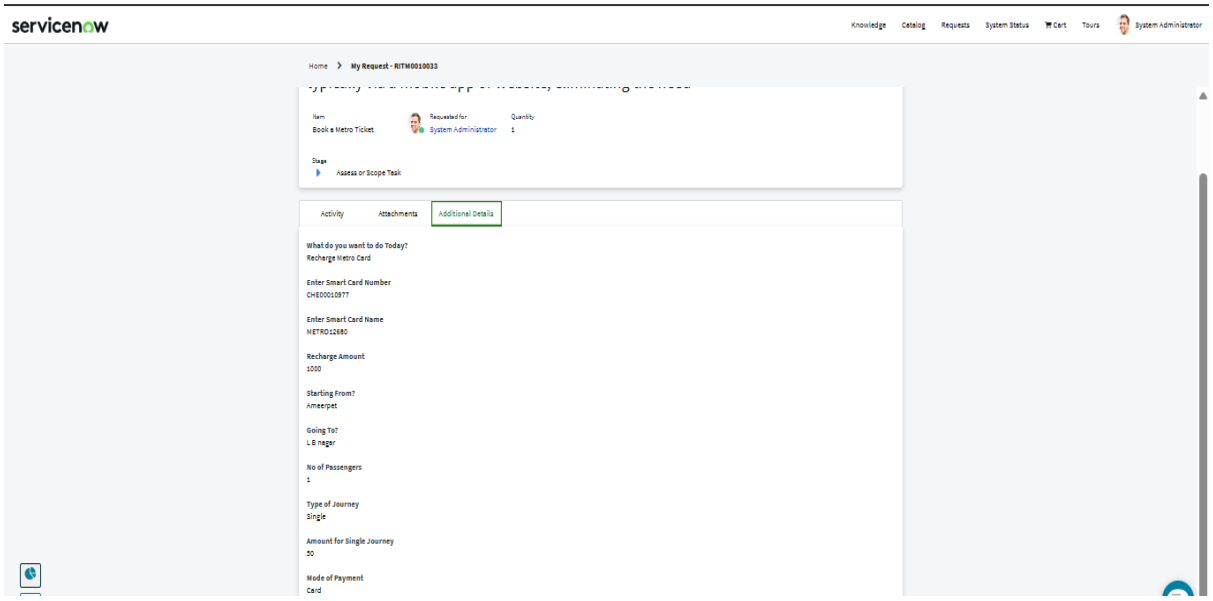
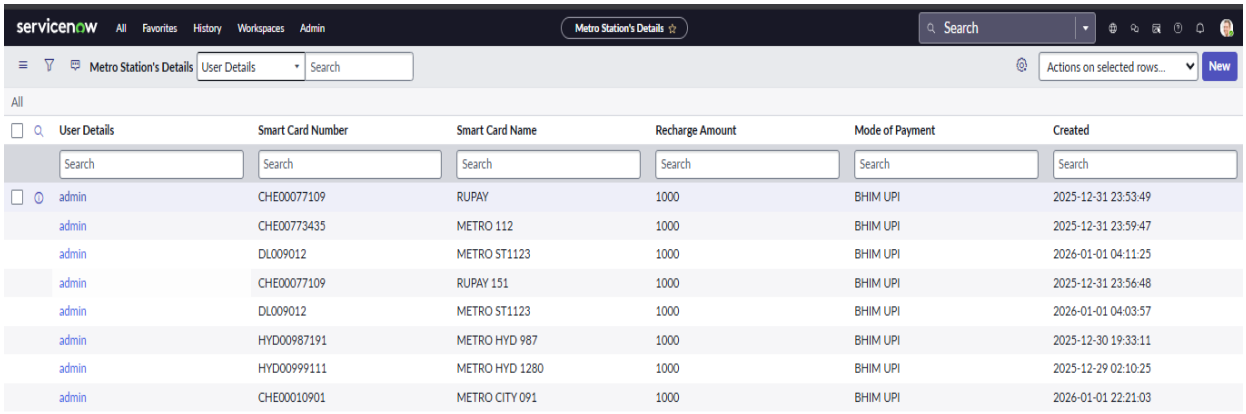


Figure 7.3.1: Backend Table Verification (Recharge Card): Successful recharges are logged in the u_metro_station_details table.

- **Logged Data**: The table captures the **Mode of Payment** (e.g., BHIM UPI), **Recharge Amount** (e.g., 1000), **Smart Card Number**, and **User Details** (e.g., admin).



User Details	Smart Card Number	Smart Card Name	Recharge Amount	Mode of Payment	Created
admin	CHE00077109	RUPAY	1000	BHIM UPI	2025-12-31 23:53:49
admin	CHE00773435	METRO 112	1000	BHIM UPI	2025-12-31 23:59:47
admin	DL009012	METRO ST1123	1000	BHIM UPI	2026-01-01 04:11:25
admin	CHE00077109	RUPAY 151	1000	BHIM UPI	2025-12-31 23:56:48
admin	DL009012	METRO ST1123	1000	BHIM UPI	2026-01-01 04:03:57
admin	HYD00987191	METRO HYD 987	1000	BHIM UPI	2025-12-30 19:33:11
admin	HYD00999111	METRO HYD 1280	1000	BHIM UPI	2025-12-29 02:10:25
admin	CHE00010901	METRO CITY 091	1000	BHIM UPI	2026-01-01 22:21:03

Figure 7.3.2: Database Schema

- The technical structure of the **Metro Station's Details** table shows the dictionary entries for all custom fields, including **Station Name**, **Smart Card Number**, **Created, Mode of payment** and **Recharge Amount**, ensuring data types are correctly defined for consistency.

8. ADVANTAGES & DISADVANTAGES

8.1 Advantages

- Faster request processing and instant ticket issuance.
- Reduced human error in fare calculation.
- Automated logging and improved visibility for transit authorities.

8.2 Disadvantages

- Dependent on Flow Designer and Service Portal availability.
- Requires initial commuter setup/registration within the ServiceNow platform.

9. CONCLUSION

The **Metro Ticket Generating System** successfully modernizes urban transit by replacing manual workflows with a seamless digital automation framework.. The implementation of dynamic forms and real-time fare calculation significantly improves the commuter experience by reducing station wait times and manual errors. High-quality technical execution is further evidenced by the instant rendering of unique **QR Tickets**, providing passengers with immediate digital credentials. Ultimately, this project delivers a professional-grade, scalable platform that is well-positioned for future enhancements like payment gateway integration and advanced travel analytics.

10. FUTURE SCOPE

- **SMS/Email Integration**: Sending ticket details via SMS for commuters without smartphones.
- **Dashboard Reporting**: Real-time analytics for station traffic and recharge volumes.
- **Payment Gateway**: Direct integration with banking APIs for live ticket purchases.