

**UNIVERSITI TUNKU ABDUL RAHMAN**

**Faculty of Information and Communication Technology (FICT)**

**UCCD2303 DATABASE TECHNOLOGY /**

**UCCD2203 DATABASE SYSTEMS**

**Group Assignment Mark sheet**

February 2025 Trimester

|  |  |
| --- | --- |
| Group Number (e.g. G999): | G033 |
| Group leader to provide the OneDrive folder hyperlink (editor mode) which contains the zip file and a group presentation video in mp4 format: | <https://utarict-my.sharepoint.com/:f:/g/personal/cziwei0112_1utar_my/EixEK6iRv7xCl0q_yD87brsBT6OU-QdmBtoE17aZT4HOdA?e=SJuEW8> |
| Group leader name: | Hii Zi Wei |
| Group leader email address: | cziwei0112@1utar.my |
| Submission date (dd-Mon-yy): | 29-Apr-25 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Member Contribution Declaration Sheet** | | | | | |
|  | | **Member 1** | **Member 2** | **Member 3** | **Member 4** |
| **Member Name** (ascending order, group leader with \* ): | | **Chia Yue Sheng** | **Hii Zi Wei** | **Lee Hien Leong** | **Teh Bee Ling** |
| Student ID (e.g. 2299999) | | **2204673** | **2204587** | **2204958** | **2204237** |
| Program CS / IA, IB / DE: | | **CS** | **CS** | **CS** | **CS** |
| **Member’s signature** | |  | **A signature of a person  Description automatically generated** | A close-up of a signature  AI-generated content may be incorrect. |  |
|  | | Contribution of each member (Total 100%) | | | |
|  | | **%** | **%** | **%** | **%** |
| PART 1: (Group Assessment – 50 marls) | |  |  |  |  |
| **1.** | Scope of Work (5 marks). |  |  |  |  |
| **2.** | ER model (10 marks). |  |  |  |  |
| **3.** | Redesign and EER (10 marks). |  |  |  |  |
| **4.** | Data Dictionary (10 marks) |  |  |  |  |
| **5.** | Tables and records (5 marks) |  |  |  |  |
| **6.** | Script (10 marks) |  |  |  |  |

Note:

* Maximum 4 members per group.
* The assignment grouping can be from different tutorial groups but must be from the same programme except with approval i.e. CS student work with CS student
* Programme e.g. IA/IB/DE/CS/CN/CT
* All members should attach their individual signature confirming that the report is not plagiarized
* For assignment answer submission include the Mark sheet, Table of Contents. Members’ contributions.

|  |  |  |  |
| --- | --- | --- | --- |
| **Assignment Mark Sheet** | |  |  |
| **PART 1: (Group Assessment – 50 marls)** | | Allocated marks | **Given Marks** |
| **1.** | **Scope of Work (5 marks)**  Analyse requirements study (briefly explain the requirements/ office / business rules in the system).  PLEASE INCLUDE ANY ASSUMPTIONS THAT YOU MAKE. | 5 |  |
| **2.** | **ER model** (**10 marks**)  You are required to design an ER diagram for the case study given, identify entities, identify relationships, identify associate attribute and determine keys.  Check your ERD with the transaction requirements stated in the case. | 10 |  |
| **3.** | **Redesign and EER** (**10 marks**)  Redesign your ER diagram with the new requirements and extending the ERD to EER model, if any. | 10 |  |
| **4.** | **Data Dictionary** (**10 marks**)  Based on EER diagram that you created in part 3, create a data dictionary for the solution. (Make sure the data types (Oracle) selected are appropriate) | 10 |  |
| **5.** | **Tables and records** (**5 marks**)  Create all relations in ERD and insert the necessary records (Minimum 5 record for each table) | 5 |  |
| **6.** | **Script** (**10 marks**)  You are required to submit the SQL schema script with proper codes. Should include Integrity and referential integrity constraints.  **Softcopy:** *Include the scripts in the submission, screenshots are not required.* | 10 |  |

**PART 2: (Individual Assessment – 50 marks)**

(Fill in all your group members name and ID according to the same sequence on the cover page)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Member |  | 1 | 2 | 3 | 4 |
| **Student Name: (to fill)** |  | **CHIA YUE SHENG** | **HII ZI WEI** | **LEE HIEN LEONG** | **TEH BEE LING** |
| **Student ID; (to fill)** |  | **2204673** | **2204587** | **2204958** | **2204237** |
| Two Queries (10 marks) | 10 |  |  |  |  |
| Two Stored Procedure (10 marks) | 10 |  |  |  |  |
| Two Functions (10 marks) | 10 |  |  |  |  |
| Presentation (20 marks) | 20 |  |  |  |  |
| Total Individual Assessment (50 marks) |  |  |  |  |  |
| Group Assessment (50 marls) |  |  |  |  |  |
| **Total marks (100 marks)** |  |  |  |  |  |

|  |
| --- |
| **Part 2 Individual Report within the group report.**  Include all members’ individual SQL and PL/SQL answers into the group report, screenshots are not required. |
| **Part 2 Individual Presentation videos in ONE group video**   1. Group members are required to present two SQL queries, two PL/SQL stored procedures, and two PL/SQL functions. 2. Every group has to present their assignment work and explain each group member contribution towards the completion of the assignment. 3. Group member must present his/her work as individual for 3-5 minutes each person and compiled by the group leader as a single presentation video. 4. Each member record individually then group leader compiles as a single group video. 5. Group leader to compile one group presentation video to be uploaded into OneDrive and provide the video hyperlink. 6. For Part 2 answers**:**  * Indicate the operations performed and include an explanations on how the user can use the corresponding SQL commands. * No duplication or similar answers should be listed among the group members. |

|  |  |  |
| --- | --- | --- |
| No. | Name  (in ascending order)  Group leader with \* | A short description of members’ contributions |
| 1. | Chia Yue Sheng | * Design and draw the ERD and EERD. * Handle Personal Script 4. * Handle reporting in TABLES AND RECORDS. * Handle Formatting issues. |
| 2. | Hii Zi Wei \* | * Leader: Distributing the scope of work. Handle the DBMS design. * Handle Group Script. * Handle Personal Script 1. * Handle Business Rules and Assumptions. * Revise the Whole Report. |
| 3. | Lee Hien Leong | * Handle Personal Script 2. * Handle users, roles, and privileges. |
| 4. | Teh Bee Ling | * Handle Personal Script 3. * Handle reporting in the INTRODUCTION AND DATA DICTIONARY. |

Contents

[**List of Figures** 6](#_Toc196835443)

[**List of Table** 7](#_Toc196835444)

[**List of Abbreviations** 8](#_Toc196835445)

[1.0 Scope of Work 9](#_Toc196835446)

[1.1. Introduction 9](#_Toc196835447)

[1.2. List of Entities 15](#_Toc196835448)

[1.3. List of Entities and Attributes in Database Design 15](#_Toc196835449)

[1.4. Assumptions & Business Rules 19](#_Toc196835450)

[1.5. User, Role, and Privileges 35](#_Toc196835451)

[2.0 ER Model 37](#_Toc196835452)

[3.0 Redesign and Enhance ER Model 38](#_Toc196835453)

[4.0 Data Dictionary 39](#_Toc196835454)

[5.0 Table and Record 45](#_Toc196835455)

[6.0 Group SQL Script 52](#_Toc196835456)

[7.0 Individual SQL Script 73](#_Toc196835457)

[7.1. Chia Yue Sheng – Personal Script 4 73](#_Toc196835458)

[7.2. Hii Zi Wei – Personal Script 1 100](#_Toc196835459)

[7.3. Lee Hien Leong - Personal Script 2 132](#_Toc196835460)

[7.4. Teh Bee Ling - Personal Script 3 159](#_Toc196835461)

**List of Figures**

Figure 2.1 - ER Model 37

Figure 3.1 - Enhance ER Model 38

**List of Table**

Table 1.1 - List of Entities 15

Table 1.2 - List of entities and attributes 18

Table 1.3- Structure of membership tier 20

Table 1.4 - Credentials of Users 35

Table 1.5 - Privileges of roles 36

Table 4.1 - Data Dictionary of SALE 39

Table 4.2 - Data Dictionary of CUSTOMERS 40

Table 4.3 - Data Dictionary of MENU\_ITEMS 40

Table 4.4 - Data Dictionary of PROMOTIONS 41

Table 4.5 - Data Dictionary of ORDERS 42

Table 4.6 - Data Dictionary of POINT\_TRANSACTIONS 42

Table 4.7 - Data Dictionary of MEMBERSHIP\_HISTORY 43

Table 4.8 - Data Dictionary of CUSTOMER\_PROMOTIONS 43

Table 4.9 - Data Dictionary of REDEMPTIONS 43

Table 4.10 - Data Dictionary of SET\_MEAL\_COMPONENTS 44

Table 4.11 - Data Dictionary of ORDER\_ITEMS 44

Table 4.12 - Data Dictionary of CUSTOMER\_ACTIVITY 45

Table 5.1 - Records of MEMBERSHIP\_TIER 45

Table 5.2 - Records of CUSTOMERS 46

Table 5.3 - Records of MENU\_ITEMS 47

Table 5.4 - Records of PROMOTIONS 48

Table 5.5 - Records of ORDERS 48

Table 5.6 - Records of POINT\_TRANSACTIONS 49

Table 5.7 - Records of MEMBERSHIP\_HISTORY 49

Table 5.8 - Records of CUSTOMER\_PROMOTIONS 50

Table 5.9 - Records of REDEMTIONS 50

Table 5.10 - Records of SET\_MEAL\_COMPONENTS 50

Table 5.11 - Records of ORDER\_ITEMS 51

Table 5.12 - Records of CUSTOMER\_ACTIVITY 51

**List of Abbreviations**

|  |  |
| --- | --- |
| **Abbreviation** | **Full Form** |
| PK | Primary Key |
| FK | Foreign Key |
| CC | Check Condition |
| NN | Not Null |
| DF | Default Constraint |
| UK | Unique Constraint |

# Scope of Work

## Introduction

The project aims to develop a Promotion Database Management System (PDBMS) for The Foodie, a fast-food restaurant specializing in burgers and fried chicken. The Foodie currently manages promotions manually, leading to inefficiencies in tracking redemptions, missed customer engagement opportunities, and difficulty analyzing promotion effectiveness. The PDBMS addresses these challenges by automating the entire promotional lifecycle. In today's competitive fast-food market, efficient promotional management is essential to drive customer loyalty, increase sales, and maintain a competitive edge. Recognizing the limitations of manual promotion tracking — such as missed opportunities and inefficient operations — The Foodie aims to implement an automated, intelligent Promotion Database Management System (PDBMS) to streamline processes, optimize marketing effectiveness, and enhance customer engagement.

The proposed PDBMS is designed to manage a variety of promotional activities within the restaurant, including vouchers, set meal promotions, loyalty programs, and seasonal campaigns. The system focuses exclusively on promotions and does not handle payment processing, inventory management, or employee scheduling. The system is built on Oracle Database using PL/SQL, with triggers and procedures for real-time automation. The overarching goal of the system is to automate and streamline the creation, tracking, and redemption of promotions, ultimately enhancing customer engagement, driving sales, and improving overall business operations. The functionalities outlined are expected to work in concert to provide a comprehensive solution for managing promotional efforts. By automating the promotion process, the system will reduce manual work, improve operational efficiency, and ensure the timely execution of promotional campaigns. Additionally, the PDBMS will offer valuable insights into the effectiveness of each promotion.

The workflow of the PDBMS is encouraging a loyalty program as a base promotion, which manages the membership tier and points system. The membership tier provides promotion in terms of different levels of benefits, like discounts according to the different tier levels. The points system is accumulated to upgrade a tier and enjoy higher benefits, or it can be used for redemption in other promotion types. Hence, we have promotion in voucher assignments, either auto-assigned or by points redemption. Then, the voucher can be used in ordering, and set meal ordering is offering a promotion with lower prices compared to à la carte. Next, the seasonal item can only be redeemed using points in the point balance of each customer once they have reached the minimum spend per receipt.

Primary users include restaurant staff (for managing promotions), marketing teams (for analytics), and customers (via redemption interfaces).

The scope of work for this project encompasses the following functionalities:

* **Loyalty Points System:**

Implement a tiered membership system where customers earn loyalty points with each purchase. Customers can use the points to upgrade to a higher tier, which has higher benefits, or redeem other vouchers or seasonal items. The system will also automate point allocation based on customer activity and track point expiration to ensure an efficient loyalty process. Hence, it is important to manage the renewal, points management, and tier upgrading of the loyalty points system to keep loyal members.

* **Voucher Management:**

Create, manage, and distribute digital or physical vouchers offering discounts, such as 10% off or 20% off, for a set duration. These vouchers can be automatically assigned based on customer tier or manually distributed during specific promotions to enhance customer engagement and increase sales. Besides, it can also be redeemed by members using points.

* **Tier-Based Membership:**

Monitor and manage customer membership tiers such as Bronze, Silver, Gold, Platinum, and Diamond. Customers will be upgraded based on loyalty points accumulated, and renewed once the expiry date is reached to extend the lifespan. If the membership is not being renewed, then it will be cancelled. If the expiry points in the point balance have not been used before expiry, it will be deducted from the point balance. Special promotional offers will be linked to specific tiers. If the points are deducted and cause the points below the tier required points criteria, a downgrade will occur automatically.

* **Seasonal Item Promotions:**

Managing and tracking limited-time offers on specific seasonal items can be either seasonal drinks, special meals, toy sets, or combinations, such as Pumpkin Spice Latte or Hello Kitty Happy Meal, which is a Hello Kitty Toy plus a Happy Meal, which can only be redeemed using points. These promotions will be linked to specific menu items and track the minimum spend of the customer in a single receipt. Hence, only achieving a specific minimum spend will get the redemption requirement.

* **Promotion Redemption:**

Allow customers to redeem their accumulated rewards, such as discounts, free items, or special offers, based on points accumulated or specific promotional criteria. With these rules, promotion types including seasonal item redemption, voucher redemption, membership tier benefiting discounts, member accumulated points and set meal bundle deal at lower prices.

* **Points Expiry and Bonus Points:**

Automatically manage the expiry of customer loyalty points based on tier policies. Customers will also be rewarded with bonus points for special occasions, such as birthdays.

* **Customer Promotion History:**

Maintain a detailed history of all customer interactions with promotions, including points earned, vouchers redeemed, and discounts applied.

* **Personalized Suggestion:**

To enhance customer engagement and encourage repeat visits, the PDBMS will support the creation of personalized promotions tailored to individual customer behaviors and preferences. By analyzing customer activity such as purchase history, redemption patterns, membership tier status, and periods of inactivity, the system can trigger targeted offers, such as bonus points, exclusive vouchers, or special seasonal item deals. For example, a customer who has minimum points left to reach a new tier enjoys massive benefits, the system will prompt them and provide suggestions on how to strengthen customer loyalty and increase customer lifetime value. This optimizes the effectiveness of marketing campaigns through data-driven targeting.

* **Real-Time Analytics and Reporting:**

Provide insights into the effectiveness of each promotion, including data on the number of redemptions, customer participation, and the total value of discounts applied. The insight is being used by the staff to plan their promotion planning and can also be used as a function to alert the customer on either the renewal or the points needed to upgrade to the next tier, as a motivating factor to extend the loyalty program. If the customer is motivated to be a loyal member, they will use their points for other promotions, redemption, and spend in the merchants.

* **Integration with Point-of-Sale System:**

Integrate the promotion system with the restaurant’s POS system to ensure that promotions are automatically applied at checkout. This includes applying discounts, vouchers, and other redemption offers when customers make their purchases. It will also check the minimum spend of a single order to allow the seasonal item redemption.

* **Customer Activity Tracking:**

Track various customer activities such as logins, orders, redemptions, and tier changes. This data is used to personalize customer experiences, trigger specific promotions, and monitor engagement with the loyalty program.

**User Roles and Access Privileges:**

Various user roles with distinct access privileges have been established to ensure the efficient operation of Promotion Database Management System (PDBMS). Each role is tailored to support specific business functions and enforce access control:

1. **Promotion Administrator (promo\_admin\_role):**

Has full system and object privileges, enabling complete management of all system functionalities and configurations. Granted full system and object privileges, this role allows comprehensive control over all database functionalities, including user management, configuration, and all promotional activities.

1. **Promotion Manager (promo\_manager\_role):**

Can view and update customer information, view membership tiers, and create/manage promotions with select, insert, and update privileges on relevant tables. Responsible for managing promotions and customer records, with privileges to select, insert, and update data in customer and promotion-related tables. This role also has read access to membership tiers and redemption data to support decision-making.

1. **Voucher Administrator (voucher\_admin\_role):**

Handles promotion and voucher-related processes with full write access to promotions, customer promotions, and redemptions, plus read access to customer data. Manages the lifecycle of promotional vouchers and redemptions. This role has full access to modify promotion, customer promotion, and redemption data, and can view customer details for accurate processing.

1. **Membership Administrator (membership\_admin\_role):**

Manages all aspects of the membership system including customer records, tier configurations, membership history, and point transactions. Oversees the loyalty membership program, including maintaining customer records, configuring membership tiers, tracking historical changes, and managing point transactions.

1. **Promotion Reporter (promo\_report\_role):**

Has read-only access to all relevant tables for performance evaluation and reporting purposes, including memberships, customers, promotions, redemptions, and transaction history. Designed for analytics and reporting purposes, this role has read-only access to all major entities within the system, including customer data, promotion details, membership activity, redemptions, and points history.

## List of Entities

|  |  |
| --- | --- |
|  | **Entity** |
| **1.** | MEMBERSHIP\_TIERS |
| **2.** | CUSTOMERS |
| **3.** | MENU\_ITEMS |
| **4.** | PROMOTIONS |
| **5.** | ORDERS |
| **6.** | POINT\_TRANSACTIONS |
| **7.** | MEMBERSHIP\_HISTORY |
| **8.** | CUSTOMER\_PROMOTIONS |
| **9.** | REDEMPTIONS |
| **10.** | SET\_MEAL\_COMPONENTS |
| **11.** | ORDER\_ITEMS |
| **12.** | CUSTOMER\_ACTIVITY |

Table 1.1 - List of Entities

## List of Entities and Attributes in Database Design

|  |  |
| --- | --- |
| **Entity** | **Attributes** |
| **MEMBERSHIP\_TIERS** | * **PK** tier\_id * tier\_name * points\_required * discount\_percentage * renewal\_fee * birthday\_bonus\_point * points\_expiry\_months * base\_earn\_rate * created\_date |
| **CUSTOMERS** | * **PK** customer\_id * name * email * phone * **FK** tier\_id **references** MEMBERSHIP TIERS (tier\_id) * points\_balance * last\_points\_earned\_date * points\_expiry\_date * membership\_expiry\_date * is\_member * membership\_status * last\_visit\_date * join\_date * birth\_date * last\_renewal\_date * renewal\_count |
| **MENU\_ITEMS** | * **PK** item\_id * item\_type * name * description * base\_price * is\_active * valid\_from * valid\_to * current\_stock * initial\_stock |
| **PROMOTIONS** | * **PK** promotion\_id * promotion\_type * name * valid\_from * valid\_to * discount\_value * voucher\_code * is\_auto\_assign * points\_required * min\_spend * **FK** set\_meal\_id **references** MENU\_ITEMS (item\_id) * **FK** applicable\_tier\_id **references** MEMBERSHIP\_TIERS (tier\_id) |
| **ORDERS** | * **PK** order\_id * **FK** customer\_id **references** CUSTOMERS (customer\_id) * order\_date * total\_amount * discount\_amount * final\_amount * payment\_method * status |
| **POINT\_TRANSACTIONS** | * **PK** transaction\_id * **FK** customer\_id **references** CUSTOMERS (customer\_id) * **FK** order\_id **references** ORDERS (order\_id) * **FK** promotion\_id **references** PROMOTIONS (promotion\_id) * points\_amount * transaction\_type * transaction\_date * description * expiry\_date |
| **MEMBERSHIP\_HISTORY** | * **PK** history\_id * **FK** customer\_id **references** CUSTOMERS (customer\_id) * **FK** old\_tier\_id **references** MEMBERSHIP\_TIERS (tier\_id) * **FK** new\_tier\_id **references** MEMBERSHIP\_TIERS (tier\_id) * change\_date * change\_reason * changed\_by |
| **CUSTOMER\_PROMOTIONS** | * **PK, FK** customer\_id **references** CUSTOMERS (customer\_id) * **PK, FK** promotion\_id **references** PROMOTIONS (promotion\_id) * date\_acquired * is\_used * used\_date * acquisition\_method |
| **REDEMPTIONS** | * **PK** redemption\_id * **FK** customer\_id **references** CUSTOMERS (customer\_id) * **FK** promotion\_id **references** PROMOTIONS (promotion\_id) * **FK** order\_id **references** ORDERS (order\_id) * points\_used * redemption\_date * redemption\_status * item\_id |
| **SET\_MEAL\_COMPONENTS** | * **PK, FK** set\_meal\_id **references** MENU\_ITEMS (item\_id) * **PK, FK** component\_id **references** MENU\_ITEMS (item\_id) * quantity |
| **ORDER\_ITEMS** | * **PK** order\_item\_id * **FK** order\_id **references** ORDERS (order\_id) * **FK** item\_id referencesMENU\_ITEMS (item\_id) * quantity * price * discount\_applied |
| **CUSTOMER\_ACTIVITY** | * **PK** activity\_id * **FK** customer\_id **references** CUSTOMERS (customer\_id) * activity\_type * activity\_date * ip\_address * details |

Table 1.2 - List of entities and attributes

## Assumptions & Business Rules

**1. MEMBERSHIP\_TIERS**

**Assumption 1:**

Assume that the tier progression is linear (Bronze→Silver→Gold→Platinum→Diamond).

**Business Rule 1**:

Each tier must have a unique name and id, a non-negative points requirement, and a discount percentage (0-100%). Each tier has different benefits, and different points requirement.

**Assumption 2:**

Assume that the customers will actively pursue tier upgrades for better benefits.

**Business Rule 2**:

Higher tiers must have greater benefits (discounts/points) than lower tiers. The points\_required for each tier must be non-negative and strictly increasing as the tier level increases. Membership benefits associated with a tier take effect immediately once the member reaches that tier.

**Assumption 3:**

Assume that a member’s current tier is determined solely by the number of accumulated points, and upgrades occur automatically when a member meets the minimum point threshold for the next tier.

**Business Rule 3:**

Each tier must offer a discount percentage between 0% and 100%, and the value must increase for higher tiers. (Diamond > Platinum > Gold > Silver > Bronze).

**Assumption 4:**

Assume that the following table defines the detailed structure and benefits of each membership tier:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tier** | **Points Required** | **Discount (%)** | **Birthday Bonus Points** | **Base Earn Rate** |
| Bronze | 0 | 5 | 50 | 1.0 |
| Silver | 500 | 10 | 100 | 1.2 |
| Gold | 1500 | 15 | 150 | 1.5 |
| Platinum | 3000 | 20 | 200 | 2.0 |
| Diamond | 5000 | 25 | 250 | 2.5 |

Table 1.3- Structure of membership tier

**Assumption 5:**

Assume that all benefit values (discount, bonus points, earn rate) increase progressively with each higher tier and are never equal or lower than those of a preceding tier.

**Business Rule 5:**

* The birthday bonus points provided by each tier must increase with each level.
* The base earn rate (points earned per RM spent) must also increase as the tier increases.
* Tier benefits (discount, bonus points, earn rate) are exclusive and cumulative within a tier, and cannot be shared or combined across multiple tiers.
* A member’s tier must be automatically recalculated whenever their points balance changes, and the system must assign the highest tier for which they qualify.
* Membership benefits associated with a tier take effect immediately once the member reaches that tier.

**Assumption 6:**

Assume that members can be downgraded to a lower tier if their points balance falls below the threshold required for the current tier.

**Business Rule 6:**

The system must automatically recalculate and update the membership tier whenever a customer’s points balance changes after points deduction due to either expiration or redemption.

**2. CUSTOMERS**

**Assumption 1:**

Assume that each customer is uniquely identified by their customer ID (customer\_id).

**Business Rule 1:**

The customer\_id must be unique for each customer and is the primary key in the CUSTOMERS table.

**Assumption 2:**

Assume that all customers belong to a membership tier (tier\_id), which is based on accumulated points, starting at Bronze.

**Business Rule 2:**

The tier\_id must reference an existing tier in the MEMBERSHIP\_TIERS table.

**Assumption 3:**

Customers will accumulate points through transactions, and the points balance may change over time.

**Business Rule 3:**

The points\_balance must always be non-negative.

**Assumption 4:**

Assume that PENDING status requires manual activation by staff.

**Business Rule 4:**

The membership\_status must be ACTIVE/CANCELLED/PENDING with valid dates.

**Assumption 5:**

Assume that customers have a birth\_date to support birthday-related promotions, and bonus points are automatically awarded based on the birthday\_bonus\_points associated with their membership tier.

**Business Rule 5:**

The system must automatically assign birthday bonus points to customers whose birthday falls on the current day through a trigger.

* The trg\_birthday\_check trigger, which runs after a user log in, checks if the customer’s birthday matches the current date. If so, and if they are eligible for birthday bonus points (defined in the MEMBERSHIP\_TIERS table), the points are automatically awarded. The points are credited to the customer’s POINT\_TRANSACTIONS record as a BONUS transaction, with details like description, transaction date, and expiry date. A corresponding entry is also made in the CUSTOMER\_ACTIVITY table to log the birthday activity.

**Assumption 6:**

Assume that members are notified of their tier upgrades and any associated benefits via email or mobile notifications.

**Business Rule 6:**

Members must be able to view their current points balance, membership tier, and any associated benefits at any time through their account interface.

**Assumption 7:**

Assume that members are alerted to the upgrade tier.

**Business Rule 7:**

Members will be receiving message or other platform notifications on their current membership status and measures to achieve next tier.

**3. MENU\_ITEMS**

**Assumption 1:**

Assume that customers can choose from menu items from either one of the following types: A\_LA\_CARTE, SET\_MEAL, INGREDIENT, or SEASONAL.

**Business Rule 1:**

Menu items must belong to one of the following types: A\_LA\_CARTE, SET\_MEAL, INGREDIENT, or SEASONAL. Each item type has distinct handling requirements, with SEASONAL items subject to additional promotional and validity constraints, which are defined by their valid\_from and valid\_to dates.

**Assumption 2:**

Assume that for a period of time, seasonal items are available on the menu.

**Business Rule 2:**

The valid\_to date must be greater than or equal to valid\_from to ensure the item is only available during the specified period. This ensures that seasonal items are available within a valid time range for promotions.

**Assumption 3:**

Assume that menu items will be updated regularly based on availability, seasonality, and promotions.

**Business Rule 3:**

Seasonal items must be flagged as ‘active’ or ‘inactive’ to prevent confusion during promotions and ensure they are not mistakenly included in regular menus after the promotion period ends.

**4. PROMOTIONS**

**Assumption 1:**

Assume that the promotion contains single inheritance disjoint of VOUCHER, SEASONAL ITEM, and SET\_MEAL.

**Business Rule 1:**

promotion\_type must be one of VOUCHER, SEASONAL, SET\_MEAL

**Assumption 2:**

Assume that a significant portion of promotions are tier-specific, meaning they apply only to certain membership tiers.

**Business Rule 2:**

* The applicable\_tier\_id is NOT NULL.
* Seasonal promotions must be redeemed using points (points\_required IS NOT NULL).
* Vouchers can be either: Redeemed using points (points\_required IS NOT NULL), or

Auto-assigned by the system (is\_auto\_assign = 'Y').

**Assumption 3:**

Assume that the other portion of promotions are open to all tiers

**Business Rule 3:**

* The applicable\_tier\_id is NULL.
* Set Meal promotions does not need a applicable\_tier\_id, minimum point required, or a minimum spend.
* Some voucher promotions are auto-assigned to all tiers.

**Assumption 4:**

Assume that the voucher has two types of assignments: Redemption via member points or Automatic system assignment. All vouchers are unique and single-use, each with a distinct voucher code and fixed discount value.

**Business Rule 4:**

* Each voucher must have a unique voucher\_code.
* Every voucher must specify a fixed discount\_value.

**Assumption 5:**

Assume that the Seasonal item promotions require point redemption. They are triggered only when a minimum spend is reached in a single transaction. Must run within a valid date range without overlapping conflicting offers.

**Business Rule 5:**

* valid\_from and valid\_to dates must define a non-negative range (valid\_to >= valid\_from).
* Seasonal promotions should not overlap with conflicting offers (enforced via application logic).

**5. ORDERS**

**Assumption 1:**

Assume that each order may have an optional discount applied (defaults to 0 if not specified), and hould not be manually editable

**Business Rule 1:**

Final amount must equal (Total amount - Discount amount) and be ≥ 0, enforced by CHECK constraint. Total order is enforced by: The trg\_order\_totals trigger (updates totals after any item change). total\_amount = Σ (price × quantity) of all items which attribute in ORDER\_ITEM

**Assumption 2**:

The business only accepts payments via CASH, CARD, or ONLINE (no other payment methods are allowed).

**Business Rule 2:**

All orders must have a valid payment method (CASH, CARD, or ONLINE), enforced by CHECK constraint.

**Assumption 3**:

An order cannot exist without a valid customer (orphaned orders are prohibited).

**Business Rule 3**:

Every order must be associated with an existing customer record, enforced by FOREIGN KEY constraint.

**Assumption 4:**

Orders are considered "COMPLETED" by default unless explicitly marked as "CANCELLED" or "REFUNDED" later.

**Business Rule 4:**

Order status defaults to COMPLETED if not specified, with only valid statuses (COMPLETED, CANCELLED, REFUNDED) allowed.

**Assumption 5:**

Assume that the system will validate the customer's eligibility for promotions or discounts at the point of order creation based on their membership tier and accumulated points.

**Business Rule 5:**

If an order is cancelled or refunded, the system must reverse any applied promotions or point redemptions and adjust the customer's points balance accordingly.

**6. POINT\_TRANSACTIONS**

**Assumption 1:**

Customer loyalty points expire after 12 months of inactivity.

**Business Rule 1:**

* System must track and automatically process point expirations via EXPIRY transactions.
* Points cannot expire before 12 months of inactivity.

**Assumption 2:**

Points transactions may or may not be associated with orders or promotions.

**Business Rule 2:**

* All transactions must be associated with a customer (customer\_id NOT NULL).
* Order and promotion associations are optional.
* Deleting a customer automatically deletes all their point transactions (CASCADE).

**Assumption 3**:

Points balance can only decrease through specific transaction types.

**Business Rule 3:**

- Valid transaction types:

* PURCHASE (add points from order)
* REDEMPTION (deduct points for rewards)
* BONUS (manual point additions)
* ADJUSTMENT (manual corrections)
* EXPIRY (automatic deductions)
* REDEMPTION/EXPIRY must deduct points (points\_amount ≤ 0)
* All other types must add points (points\_amount ≥ 0)

**Assumption 4**:

Assume that customers will be notified about upcoming point expirations 30 days in advance.

**Business Rule 4:**

Manual adjustments to point balances can be performed by authorized staff members but must be logged and reviewed for compliance.

**7. MEMBERSHIP\_HISTORY**

**Assumption 1:**

Tier changes can only occur for specific, predefined business reasons.

**Business Rule 1:**

Valid change reasons: SIGNUP (initial membership)

RENEWAL (periodic renewal)

UPGRADE (tier improvement)

DOWNGRADE (tier reduction)

ADMIN (manual adjustment)

AUTO (system-triggered change)

(Enforced by CHECK constraint)

**Assumption 2:**

Membership tier changes should be permanently audited for compliance and customer service purposes.

**Business Rule 2:**

Every tier change must record:

* Previous tier (optional for new signups)
* New tier (mandatory)
* Timestamp of change
* Reason for change
* Who initiated the change

**Assumption 3:**

All membership changes must be tied to a valid customer.

**Business Rule 3:**

customer\_id is mandatory (NOT NULL)

Deleting a customer automatically deletes their history (CASCADE)

**8. CUSTOMER\_PROMOTIONS**

**Assumption 1:**

Promotions can be acquired through multiple channels but must be tracked per customer.

**Business Rule 1:**

Each promotion assignment is unique per customer (PRIMARY KEY (customer\_id, promotion\_id)).

Deleting a customer/promotion automatically revokes access (via ON DELETE CASCADE).

**Assumption 2:**

Promotions can be obtained automatically or manually, but the source must be documented.

**Business Rule 2:**

Valid acquisition methods:

* AUTO\_TIER (auto-assigned based on membership tier
* POINT\_REDEEM (redeemed with loyalty points)
* MANUAL (staff-assigned exception)
* BIRTHDAY (special birthday reward)

(Enforced by CHECK constraint)

**Assumption 3:**

Unused promotions should not clutter historical records.

**Business Rule 3:**

Promotions are unused by default (is\_used DEFAULT 'N').

When used (is\_used = 'Y'), the used\_date becomes mandatory (chk\_used\_date constraint).

**Assumption 4:**

The system must track when promotions are granted and (if applicable) when they are used.

**Business Rule 4:**

date\_acquired defaults to the current date (SYSDATE).

used\_date is optional until the promotion is marked as used.

**9. REDEMPTIONS**

**Assumption 1:**

Customers can redeem promotions by spending loyalty points, and each redemption must be tracked.

**Business Rule1:**

Every redemption must be associated with:

* A valid customer (customer\_id NOT NULL, with ON DELETE CASCADE)
* A valid promotion (promotion\_id NOT NULL, with ON DELETE CASCADE)
* Redemptions are assigned a unique ID (redemption\_id via sequence).

**Assumption 2:**

Redemptions may or may not be tied to a specific order or menu item.

**Business Rule 2:**

* order\_id is optional (redemptions can occur without an immediate order).
* item\_id is optional (not all promotions apply to specific menu items).
* Deleting an order or menu item cascades to related redemptions.

**Assumption 3:**

Redemptions should not use negative points.

**Business Rule 3:**

points\_used must be ≥ 0 (CHECK constraint). Points are deducted from the customer's balance upon redemption (enforced by application logic).

**Assumption 4:**

Redemptions are typically completed immediately but can be cancelled.

**Business Rule 4:**

redemption\_status defaults to COMPLETED. Valid status:

* COMPLETED (successful redemption)
* CANCELLED (reversed redemption)

(Enforced by CHECK constraint)

**Assumption 5:**

The system must track when redemptions occur.

**Business Rule 5:**

redemption\_date defaults to the current timestamp (SYSTIMESTAMP).

**10. SET\_MEAL\_COMPONENTS**

**Assumption 1:**

Set meals are composed of multiple menu items, and these relationships must be explicitly defined.

**Business Rule 1:**

Each set meal is defined by its components through set\_meal\_id and component\_id (both referencing MENU\_ITEMS.item\_id). Deleting a menu item automatically removes its associations (ON DELETE CASCADE).

**Assumption 2:**

A set meal cannot include itself as a component (to prevent recursive loops).

**Business Rule 2:**

A menu item cannot be its own component (no\_self\_reference CHECK constraint).

**Assumption 3:**

Components must have a positive quantity.

**Business Rule 3:**

quantity defaults to 1 and must be > 0 (CHECK constraint). Quantity cannot be NULL (NOT NULL).

**Assumption 4:**

The combination of a set meal and its component must be unique.

**Business Rule 4:**

The primary key (set\_meal\_id, component\_id) ensures no duplicate components per set meal.

**11. ORDER\_ITEMS**

**Assumption 1:**

Every order contains one or more items, each with its own pricing and quantity details.

**Business Rule 1:**

Each order item must be associated with:

* A valid parent order (order\_id NOT NULL, with ON DELETE CASCADE)
* A valid menu item (item\_id NOT NULL, with ON DELETE CASCADE)
* Items are assigned a unique ID (order\_item\_id via sequence).

**Assumption 2:**

Order items should have reasonable, non-negative quantities and prices.

**Business Rule 2:**

* quantity must be > 0 (minimum 1, enforced by CHECK constraint)
* price must be ≥ 0 (non-negative, enforced by CHECK constraint)
* discount\_applied must be ≥ 0 (non-negative, enforced by CHECK constraint)

**Assumption 3:**

Default values should simplify order entry for common cases.

**Business Rule 3:**

* quantity defaults to 1 (most common case)
* discount\_applied defaults to 0 (no discount unless specified)

**Assumption 4:**

Prices should be captured at time of ordering (not rely on current menu prices).

**Business Rule 4:**

price is required (NOT NULL) to maintain historical accuracy. Price represents the actual charged price, not necessarily current menu price

**12. CUSTOMER\_ACTIVITY**

**Assumption 1:**

Customer interactions with the system should be tracked for analytics and security purposes.

**Business Rule 1:**

Every activity record must be tied to a valid customer (customer\_id NOT NULL with ON DELETE CASCADE). Activities are assigned a unique ID (activity\_id via sequence)

**Assumption 2:**

Only specific types of customer activities should be logged.

**Business Rule 2:**

Valid activity types:

* LOGIN (authentication events)
* ORDER (purchase activities)
* REDEMPTION (loyalty reward usage)
* TIER\_CHANGE (membership tier updates)

(Enforced by CHECK constraint)

**Assumption 3:**

Activities should be timestamped automatically.

**Business Rule 3:**

activity\_date defaults to current system timestamp (SYSTIMESTAMP). Records should be immutable after creation

**OTHER ASSUMPTIONS:**

(Cross-Table Business Rules Aligned with Each Promotion Types/Main Functions)

**1. Membership Management**

**Assumption:**

Assume that a member tier is being upgraded.

**Business Rule:**

It requires a minimum loyalty points threshold, an active customer status, and no pending downgrades. Upgrade notification will be given before and after to the member.

**Assumption:**

Assume that member points are expiring.

**Business Rule:**

Point expiration handling must:

* Process quarterly batches (bulk operation).
* Notify customers 30 days prior.
* Check the downgrade needed after deduction of point or not.

**Assumption:**

Assume that member extend the memberships lifespan.

**Business Rule:**

Automated renewal alert sent to customer before expiration.

Automated renewal will be conduct once renewal fee is paid.

**2. Seasonal Item Promotion**

**Assumption:**

Assume that a member redeems a seasonal item.

**Business Rule:**

Seasonal redemptions require:

* Valid seasonal menu item
* Valid order (contains the setmeal)
* Valid date range (start\_date ≤ redemption\_date ≤ end\_date)
* Customer tier eligibility
* Minimum spend in an order
* Enough point to redeem

**Assumption:**

Assume that the seasonal item has a lifecycle on promotion.

**Business Rule:**

Promotion lifecycle automation:

* Auto-activate before start\_time.
* Auto-expire at end\_time and 1-day grace period.

**3. Order Management**

**Assumption:**

Assume that the customer ordered a set meal.

**Business Rule:**

* Set meal pricing formula must applied:

Final Price = MAX( (Σ component\_prices × 0.7), minimum\_profit\_price

* Check voucher usage availability.

**Assumption:**

Assume that the customer order à la carte.

**Business Rule:**

À la carte orders must:

* Contain ≥1 active menu item.
* Respect current availability flags.
* Validate custom modifications.
* Check voucher usage availability.

**4. Voucher System**

**Assumption:**

Assume that the voucher is auto assigned.

**Business Rule:**

Bulk voucher assignments must:

* Filter by the active status and tier eligibility.
* Prevent duplicate allocations.
* Limit 5 vouchers/customer/month.

**Assumption:**

Assume that the voucher is redeemed by a member.

**Business Rule:**

Voucher redemption checks:

* Valid date range
* Minimum basket value
* Non-combinable flag

**Assumption:**

All changes to customer membership status, tier upgrades, points transactions, and promotion redemptions must be logged for audit and compliance purposes. The system must be capable of handling high volumes of concurrent transactions, such as membership renewals, points expiration, and other bulk operations during promotional periods without significant performance degradation.

## User, Role, and Privileges

This section outlines user credentials, including their corresponding passwords and roles within the system. Its purpose is to streamline the process of granting varying levels of access privileges to different users within the database system.

|  |  |  |
| --- | --- | --- |
| **User** | **Password** | **Role** |
| admin\_user | Admin@1234 | promo\_admin\_role |
| manager\_user | Manager@1234 | promo\_manager\_role |
| voucher\_user | Voucher@1234 | voucher\_admin\_role |
| member\_user | Member@1234 | membership\_admin\_role |
| report\_user | Report@1234 | promo\_report\_role |

Table 1.4 - Credentials of Users

Each user has been granted: Their respective role(s) (GRANT role TO user) and CREATE SESSION (allows login).

|  |  |
| --- | --- |
| **Role** | **Privilege** |
| promo\_admin\_role | * ALL privileges on MEMBERSHIP\_TIERS * ALL privileges on CUSTOMERS   ALL privileges on PROMOTIONS   * ALL privileges on CUSTOMER\_PROMOTIONS * ALL privileges on REDEMPTIONS |
| promo\_manager\_role | * SELECT, INSERT, UPDATE on CUSTOMERS * SELECT on MEMBERSHIP\_TIERS * SELECT, INSERT, UPDATE on PROMOTIONS * SELECT on CUSTOMER\_PROMOTIONS * SELECT on REDEMPTIONS |
| voucher\_admin\_role | * SELECT, INSERT, UPDATE on PROMOTIONS * SELECT, INSERT, UPDATE on CUSTOMER\_PROMOTIONS * SELECT, INSERT, UPDATE on REDEMPTIONS * SELECT on CUSTOMERS |
| membership\_admin\_role | * SELECT, INSERT, UPDATE on CUSTOMERS * SELECT, INSERT, UPDATE on MEMBERSHIP\_TIERS * SELECT, INSERT, UPDATE on MEMBERSHIP\_HISTORY * SELECT, INSERT, UPDATE on POINT\_TRANSACTIONS |
| promo\_report\_role | All are (read-only):   * SELECT on MEMBERSHIP\_TIERS * SELECT on CUSTOMERS * SELECT on PROMOTIONS * SELECT on CUSTOMER\_PROMOTIONS * SELECT on REDEMPTIONS * SELECT on MEMBERSHIP\_HISTORY * SELECT on POINT\_TRANSACTIONS |

Table 1.5 - Privileges of roles

Key Notes for the above Table 1.5:

ALL privileges = Full control (SELECT, INSERT, UPDATE, DELETE, etc.)

# 

# ER Model

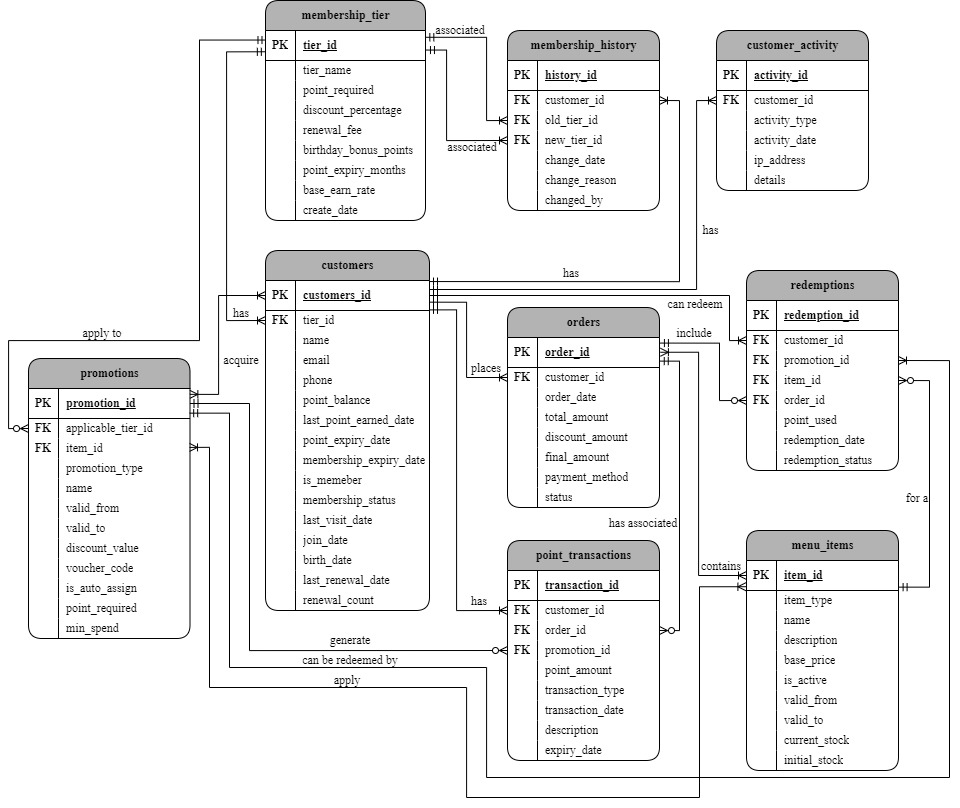


Figure 2.1 - ER Model

# 

# Redesign and Enhance ER Model

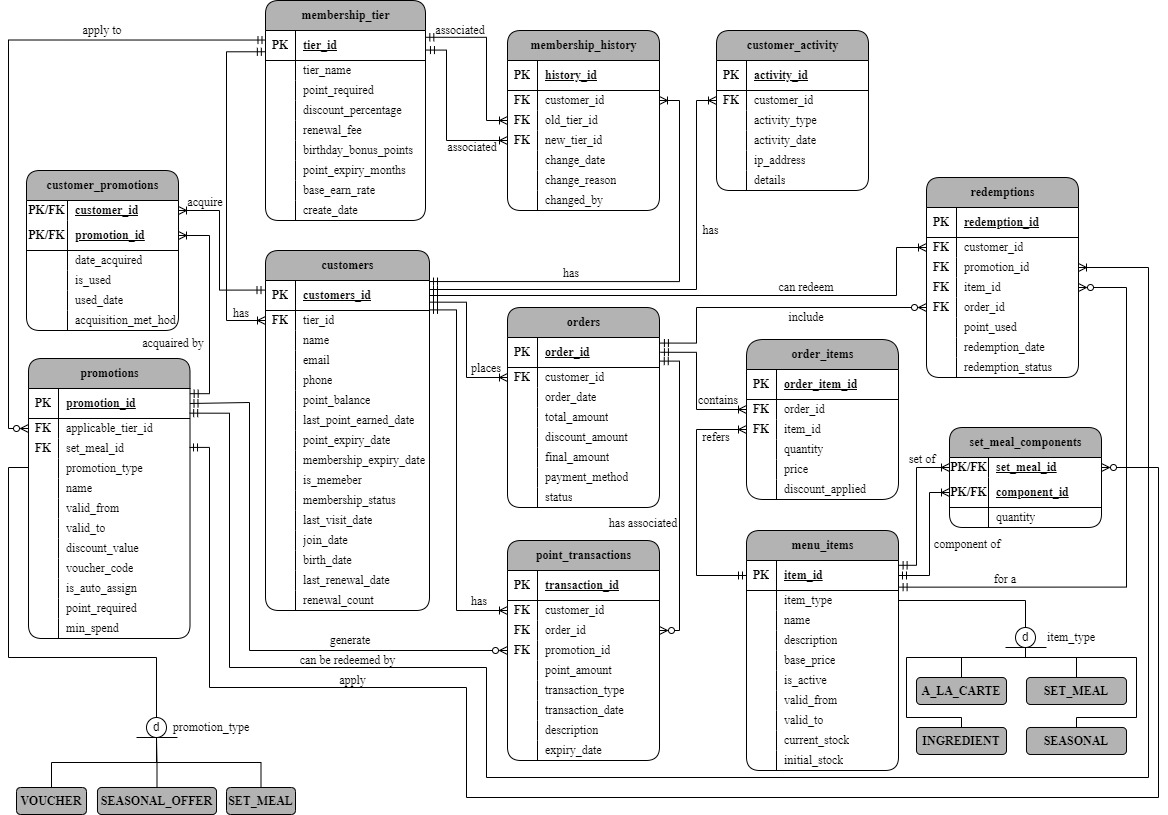


Figure 3.1 - Enhance ER Model

# Data Dictionary

**Table Name: MEMBERSHIP\_TIERS**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute Name** | **Constraints** | **Data Type** | **Data Format** | **Field Size** | **Allowable Values** | **Description** |
| tier\_id | PK, DF | NUMBER | - | - | 1 to 999 | Unique identifier for membership tiers |
| tier\_name | NN, UK | VARCHAR2 | - | 20 | - | Name of the membership tier |
| points\_required | NN, CC | NUMBER | - | - | 0 to 999999 | Points required to qualify for this tier |
| discount\_percentage | NN, CC | NUMBER(5,2) | - | 5 | 0 to 100.00 | Discount percentage for this tier |
| renewal\_fee | DF | NUMBER(10,2) | - | 10 | 0 to 99999999.99 | Renewal fee for the tier |
| birthday\_bonus\_points | DF | NUMBER | - | - | 0 to 999999 | Birthday bonus points given for the customer |
| points\_expiry\_months | DF | NUMBER | - | - | 0 to 999 | Number of months before points expire |
| base\_earn\_rate | DF | NUMBER(5,2) | - | 5 | 0 to 999.99 | Base earning rate for points |
| created\_date | DF | TIMESTAMP | - | - | - | Date and time the tier was created |

Table 4.1 - Data Dictionary of SALE

**Table Name: CUSTOMERS**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute Name** | **Constraints** | **Data Type** | **Data Format** | **Field Size** | **Allowable Values** | **Description** |
| customer\_id | PK, DF | NUMBER | - | - | 1001 to 99999 | Unique customer identifier |
| name | NN | VARCHAR2 | - | 100 | - | Name of the customer |
| email | NN, UK | VARCHAR2 | - | 100 | - | Email address of the customer |
| phone | NN, UK | VARCHAR2 | - | 15 | - | Phone number of the customer |
| tier\_id | NN, FK | NUMBER | - | - | - | Unique identifier for membership tiers |
| points\_balance | DF, CC | NUMBER | - | - | 0 to 999999 | Points balance of the customer |
| last\_points\_earned\_date | - | DATE | YYYY-MM-DD | - | - | Date when the last points were earned |
| points\_expiry\_date | - | DATE | YYYY-MM-DD | - | - | Date when current points will expire |
| membership\_expiry\_date | - | DATE | YYYY-MM-DD | - | - | Date when membership expires |
| is\_member | DF, CC | CHAR | - | 1 | ‘Y’ or ‘N’ | Whether the customer is a member |
| membership\_status | DF, CC | VARCHAR2 | - | 20 | ‘ACTIVE’,  ‘CANCELLED’,  ‘PENDING’ | Status of the customer’s membership |
| last\_visit\_date | - | DATE | YYYY-MM-DD | - | - | Date of last visit |
| join\_date | DF | DATE | YYYY-MM-DD | - | - | Date customer joined |
| birth\_date | - | DATE | YYYY-MM-DD | - | - | Customer’s birth date |
| last\_renewal\_date | - | DATE | YYYY-MM-DD | - | - | Date membership was last renewed |
| renewal\_count | DF | NUMBER | - | - | 0 to 999999 | Count of membership renewals |

Table 4.2 - Data Dictionary of CUSTOMERS

**Table Name: MENU\_ITEMS**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute Name** | **Constraints** | **Data Type** | **Data Format** | **Field Size** | **Allowable Values** | **Description** |
| item\_id | PK, DF | NUMBER | - | - | 1 to 99999 | Unique identifier for item |
| item\_type | NN, CC | VARCHAR2 | - | 20 | ‘A\_LA\_CARTE’,’SET\_MEAL’, ‘INGREDIENT’, ‘SEASONAL’ | Type of menu item |
| name | NN | VARCHAR2 | - | 100 | - | Name of the menu item |
| description | - | VARCHAR2 | - | 200 | - | Description of the menu item |
| base\_price | NN, CC | NUMBER(10,2) | - | 10 | 0 to 99999999.99 | Base price of the item |
| is\_active | DF, CC | CHAR | - | 1 | ‘Y’ or ‘N’ | Whether item is currently available |
| valid\_from | DF | DATE | YYYY-MM-DD | - | - | Date item becomes available |
| valid\_to | - | DATE | YYYY-MM-DD | - | - | Date item becomes unavailable |
| current\_stock | - | NUMBER | - | - | 0 to 99999 | Current stock quantity |
| initial\_stock | - | NUMBER | - | - | 0 to 99999 | Initial stock quantity |

Table 4.3 - Data Dictionary of MENU\_ITEMS

**Table Name: PROMOTIONS**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute Name** | **Constraints** | **Data Type** | **Data Format** | **Field Size** | **Allowable Values** | **Description** |
| promotion\_id | PK, DF | NUMBER | - | - | 1 to 99999 | Unique identifier for promotion |
| promotion\_type | NN, CC | VARCHAR2 | - | 20 | ‘VOUCHER’, ‘SEASONAL’, ‘SET\_MEAL’ | Type of promotion |
| name | NN | VARCHAR2 | - | 100 | - | Name of the promotion |
| valid\_from | NN | DATE | YYYY-MM-DD | - | - | Start date of promotion validity |
| valid\_to | NN | DATE | YYYY-MM-DD | - | - | End date of promotion validity |
| discount\_value | - | NUMBER(10,2) | - | 10 | 0.00 to 99999999.99 | Discount amount |
| voucher\_code | UK | VARCHAR2 | - | 50 | - | Unique voucher code |
| is\_auto\_assign | DF, CC | CHAR | - | 1 | ‘Y’ or ‘N’ | Whether promotion is auto assigned |
| points\_required | - | NUMBER | - | - | 0 to 999999 | Points needed to seasonal promotions |
| min\_spend | - | NUMBER(10,2) | - | 10 | 0 to 99999999.99 | Minimum spend requirement for the promotion |
| set\_meal\_id | FK | NUMBER | - | - | - | Unique identifier for menu items |
| applicable\_tier\_id | FK | NUMBER | - | - | - | Unique identifier for membership tiers |

Table 4.4 - Data Dictionary of PROMOTIONS

**Table Name: ORDERS**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute Name** | **Constraints** | **Data Type** | **Data Format** | **Field Size** | **Allowable Values** | **Description** |
| order\_id | PK, DF | NUMBER | - | - | 5001 to 99999 | Unique identifier for order |
| customer\_id | FK, NN | NUMBER | - | - | - | Unique identifier for customer |
| order\_date | DF | TIMESTAMP | - | - | - | Date and time when the order was placed |
| total\_amount | NN, CC | NUMBER(10,2) | - | 10 | 0 to 99999999.99 | Total order amount before discounts |
| discount\_amount | DF, CC | NUMBER(10,2) | - | 10 | 0 to 99999999.99 | Total discount amount applied |
| final\_amount | NN, CC | NUMBER(10,2) | - | 10 | 0 to 99999999.99 | Final amount after applying discounts |
| payment\_method | CC | VARCHAR2 | - | 50 | ‘CASH’, ‘CARD’, ‘ONLINE’ | Method used for payment |
| status | DF, CC | VARCHAR2 | - | 20 | ‘COMPLETED’, ‘CANCELLED’, ‘REFUNDED’ | Current order status |

Table 4.5 - Data Dictionary of ORDERS

**Table Name: POINT\_TRANSACTIONS**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute Name** | **Constraints** | **Data Type** | **Data Format** | **Field Size** | **Allowable Values** | **Description** |
| transaction\_id | PK, DF | NUMBER | - | - | 1 to 99999 | Unique identifier for transaction |
| customer\_id | FK, NN | NUMBER | - | - | - | Unique identifier for customer |
| order\_id | FK | NUMBER | - | - | - | Unique identifier for order |
| promotion\_id | FK | NUMBER | - | - | - | Unique identifier for promotion |
| points\_amount | NN | NUMBER | - | - | - | Points amount involved in the transaction |
| transaction\_type | NN, CC | VARCHAR2 | - | 20 | ‘PURCHASE’, ‘REDEMPTION’, ‘BONUS’, ‘ADJUSTMENT’, ‘EXPIRY’ | Type of transaction |
| transaction\_date | DF, NN | TIMESTAMP | - | - | - | Date and time of transaction |
| description | - | VARCHAR2 | - | 200 | - | Description of transaction |
| expiry\_date | - | DATE | YYYY-MM-DD | - | - | Date when points will expire |

Table 4.6 - Data Dictionary of POINT\_TRANSACTIONS

**Table Name: MEMBERSHIP\_HISTORY**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute Name** | **Constraints** | **Data Type** | **Data Format** | **Field Size** | **Allowable Values** | **Description** |
| history\_id | PK, DF | NUMBER | - | - | 1 to 99999 | Unique identifier for history |
| customer\_id | FK, NN | NUMBER | - | - | - | Unique identifier for customer |
| old\_tier\_id | FK | NUMBER | - | - | - | Unique identifier for membership tiers |
| new\_tier\_id | FK, NN | NUMBER | - | - | - | Unique identifier for membership tiers |
| change\_date | DF | TIMESTAMP | - | - | - | Date and time when the tier change occurred |
| change\_reason | CC | VARCHAR2 | - | 100 | ‘SIGNUP’, ‘RENEWAL’, ‘UPGRADE’, ‘DOWNGRADE’, ‘ADMIN, AUTO’ | Reason for the tier change |
| changed\_by | DF | VARCHAR2 | - | 30 | - | User who performed the tier change |

Table 4.7 - Data Dictionary of MEMBERSHIP\_HISTORY

**Table Name: CUSTOMER\_PROMOTIONS**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute Name** | **Constraints** | **Data Type** | **Data Format** | **Field Size** | **Allowable Values** | **Description** |
| customer\_id | NN, FK, PK | NUMBER | - | - | - | Unique identifier for customer |
| promotion\_id | NN, FK, PK | NUMBER | - | - | - | Unique identifier for promotion |
| date\_acquired | DF | DATE | YYYY-MM-DD | - | - | Date when the promotion was acquired |
| is\_used | DF, CC | CHAR | - | 1 | ‘Y’ or ‘N’ | Whether the promotion has been used |
| used\_date | - | DATE | YYYY-MM-DD | - | - | Date when the promotion was used |
| acquisition\_method | CC | VARCHAR2 | - | 20 | ‘AUTO\_TIER’, ‘POINT\_REDEEM’, ‘MANUAL’, ‘BIRTHDAY’ | Method by which the promotion was acquired |

Table 4.8 - Data Dictionary of CUSTOMER\_PROMOTIONS

**Table Name: REDEMPTIONS**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute Name** | **Constraints** | **Data Type** | **Data Format** | **Field Size** | **Allowable Values** | **Description** |
| redemption\_id | PK, DF | NUMBER | - | - | 1 to 99999 | Unique identifier for redemption |
| customer\_id | FK, NN | NUMBER | - | - | - | Unique identifier for customer |
| promotion\_id | FK, NN | NUMBER | - | - | - | Unique identifier for promotion |
| order\_id | FK | NUMBER | - | - | - | Unique identifier for order |
| points\_used | NN, CC | NUMBER | - | - | 0 to 999999 | Points used in the redemption |
| redemption\_date | DF | TIMESTAMP | - | - | - | Date and time when the redemption occurred |
| redemption\_status | DF, CC | VARCHAR2 | - | 20 | ‘COMPLETED’, ‘CANCELLED’ | Status of redemption |
| item\_id | FK | NUMBER | - | - | - | Unique identifier for item |

Table 4.9 - Data Dictionary of REDEMPTIONS

**Table Name: SET\_MEAL\_COMPONENTS**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute Name** | **Constraints** | **Data Type** | **Data Format** | **Field Size** | **Allowable Values** | **Description** |
| set\_meal\_id | NN, FK, PK | NUMBER | - | - | - | Unique identifier for set meal |
| component\_id | NN, FK, PK | NUMBER | - | - | - | Unique identifier for component |
| quantity | DF, NN, CC | NUMBER | - | - | 1 to 9999 | Quantity of the component in the set meal |

Table 4.10 - Data Dictionary of SET\_MEAL\_COMPONENTS

**Table Name: ORDER\_ITEMS**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute Name** | **Constraints** | **Data Type** | **Data Format** | **Field Size** | **Allowable Values** | **Description** |
| order\_item\_id | PK, DF | NUMBER | - | - | 1 to 99999 | Unique identifier for each item in an order |
| order\_id | FK, NN | NUMBER | - | - | - | Unique identifier for order |
| item\_id | FK, NN | NUMBER | - | - | - | Unique identifier for item |
| quantity | DF, CC | NUMBER | - | - | 1 to 9999 | Quantity of this item ordered |
| price | NN, CC | NUMBER(10,2) | - | 10 | 0 to 99999999.99 | Price per unit at time of order |
| discount\_applied | DF, CC | NUMBER(10,2) | - | 10 | 0 to 99999999.99 | Discount amount applied per unit |

Table 4.11 - Data Dictionary of ORDER\_ITEMS

**Table Name: CUSTOMER\_ACTIVITY**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attribute Name** | **Constraints** | **Data Type** | **Data Format** | **Field Size** | **Allowable Values** | **Description** |
| activity\_id | PK, DF | NUMBER | - | - | - | Unique identifier for activity |
| customer\_id | FK | NUMBER | - | - | - | Unique identifier for customer |
| activity\_type | CC | VARCHAR2 | - | 30 | ‘LOGIN’,  ‘ORDER’, ‘SEASONAL\_REDEMPTION’, ‘SEASONAL\_EXPIRATION’, ‘VOUCHER\_REDEMPTION’, ‘VOUCHER\_GENERATION’, ‘TIER\_CHANGE’, ‘RENEWAL, EXPIRATION’, ‘VOUCHER\_ASSIGNMENT’ | Type of activity performed by the customer |
| activity\_date | DF | TIMESTAMP | - | - | - | Timestamp when the activity occurred |
| ip\_address | - | VARCHAR2 | - | 45 | - | IP address from which the activity was performed |
| details | - | VARCHAR2 | - | 4000 | - | Detailed description of the activity |

Table 4.12 - Data Dictionary of CUSTOMER\_ACTIVITY

# Table and Record

**Table Name: MEMBERSHIP\_TIER**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TIER\_ID** | **TIER\_ NAME** | **POINTS\_ REQUIRED** | **DISCOUNT\_ PERCENTAGE** | **RENEWAL\_FEE** | **BIRTHDAY\_ BONUS\_POINTS** | **POINTS\_EXPIRY\_MONTHS** | **BASE\_EARN\_RATE** | **CREATED\_DATE** |
| 1 | Bronze | 0 | 5.00 | 10.00 | 50 | 12 | 1.0 | [current timestamp] |
| 2 | Silver | 500 | 10.00 | 15.00 | 100 | 12 | 1.2 | [current timestamp] |
| 3 | Gold | 1500 | 15.00 | 20.00 | 150 | 12 | 1.5 | [current timestamp] |
| 4 | Platinum | 3000 | 20.00 | 25.00 | 200 | 12 | 2.0 | [current timestamp] |
| 5 | Diamond | 5000 | 25.00 | 30.00 | 250 | 12 | 2.5 | [current timestamp] |

Table 5.1 - Records of MEMBERSHIP\_TIER

**Table Name: CUSTOMERS**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CUSTOMER\_ID** | **NAME** | **EMAIL** | **PHONE** | **TIER\_ID** | **POINTS\_BALANCE** | **LAST\_POINTS\_EARNED\_DATE** | **POINTS\_EXPIRY\_DATE** | **MEMBERSHIP\_EXPIRY\_DATE** | **IS\_MEMBER** | **MEMBERSHIP\_STATUS** | **LAST\_VISIT\_DATE** | **JOIN\_DATE** | **BIRTH\_DATE** | **LAST\_RENEWAL\_DATE** | **RENEWAL\_COUNT** |
| 1001 | John Smith | johnsmith@example.com | 012-3456789 | 1 | 300 | 2024-12-01 | 2025-12-01 | 2025-12-01 | Y | ACTIVE | 2025-04-01 | [today] | 1990-04-07 | NULL | 0 |
| 1002 | Emily Johnson | emilyj@example.com | 013-2233445 | 2 | 750 | 2025-03-21 | 2026-03-21 | 2026-03-21 | Y | ACTIVE | 2025-03-15 | [today] | 1988-07-12 | NULL | 0 |
| 1003 | Michael Lee | mikelee@example.com | 014-5566778 | 3 | 1000 | 2025-01-10 | 2026-01-10 | 2026-01-10 | Y | ACTIVE | 2025-01-11 | [today] | 1995-11-30 | NULL | 0 |
| 1004 | Sarah Tan | saraht@example.com | 015-9988776 | 4 | 2000 | 2025-02-28 | 2026-02-28 | 2026-02-28 | Y | ACTIVE | 2025-02-28 | [today] | 1992-03-18 | NULL | 0 |
| 1005 | David Wong | davidw@example.com | 016-1122334 | 5 | 3200 | 2025-03-05 | 2026-03-05 | 2026-03-05 | Y | ACTIVE | 2025-03-05 | [today] | 1985-05-25 | NULL | 0 |
| 1006 | Test User for G001 | test.user@example.com | 012-9999999 | 1 | 500 | 2025-03-01 | 2025-12-31 | 2025-03-01 | Y | ACTIVE | 2025-04-01 | [today] | 1990-01-01 | NULL | 0 |

Table 5.2 - Records of CUSTOMERS

**Table Name: MENU\_ITEMS**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ITEM \_ID** | **ITEM\_TYPE** | **NAME** | **DESCRIPTION** | **BASE\_**  **PRICE** | **IS\_**  **ACTIVE** | **VALID\_**  **FROM** | **VALID\_**  **TO** | **CURRENT\_ STOCK** | **INITIAL\_ STOCK** |
| 1 | A\_LA\_CARTE | Cheeseburger | Classic beef burger with cheese | 8.90 | Y | NULL | NULL | NULL | NULL |
| 2 | A\_LA\_CARTE | Chicken Nuggets | 6-piece nuggets | 7.00 | Y | NULL | NULL | NULL | NULL |
| 3 | A\_LA\_CARTE | Garden Salad | Fresh greens with dressing | 5.50 | Y | NULL | NULL | NULL | NULL |
| 4 | A\_LA\_CARTE | Chocolate Shake | Creamy chocolate milkshake | 4.50 | Y | NULL | NULL | NULL | NULL |
| 5 | SET\_MEAL | Family Combo | Includes 2 burgers, 2 fries, 2 drinks | 24.90 | Y | NULL | NULL | NULL | NULL |
| 6 | SET\_MEAL | Value Meal | Burger, fries, and drink | 12.50 | Y | NULL | NULL | NULL | NULL |
| 7 | INGREDIENT | Fries | Crispy golden fries | 3.50 | Y | NULL | NULL | 5000 | 5000 |
| 8 | INGREDIENT | Soft Drink | Regular fountain drink | 2.50 | Y | NULL | NULL | 3000 | 3000 |
| 9 | SEASONAL | Hello Kitty Happy Meal | Toy + Kids Meal (Limited Edition) | 12.99 | Y | 2025-10-15 | 2025-11-30 | 1000 | 1000 |
| 10 | SEASONAL | Pumpkin Spice Latte | Seasonal autumn drink | 5.90 | Y | 2025-09-15 | 2025-11-15 | 200 | 200 |
| 11 | SEASONAL | Festival Mooncake Set | Premium mooncakes (Mid-Autumn Special) | 18.50 | Y | 2025-09-01 | 2025-09-30 | 500 | 500 |
| 12 | SEASONAL | Summer BBQ Burger | Special summer edition burger | 9.99 | Y | 2025-06-01 | 2025-08-31 | 800 | 800 |
| 13 | SEASONAL | Winter Hot Chocolate | Rich hot chocolate with marshmallows | 4.99 | Y | 2025-12-01 | 2026-02-28 | 300 | 300 |

Table 5.3 - Records of MENU\_ITEMS

**Table Name: PROMOTIONS**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **PROMOTION\_ID** | **PROMOTION\_TYPE** | **NAME** | **VALID\_FROM** | **VALID\_ TO** | **DISCOUNT\_VALUE** | **VOUCHER\_CODE** | **IS\_AUTO\_ASSIGN** | **POINTS\_ REQUIRED** | **MIN\_ SPEND** | **SET\_MEAL\_ID** | **APPLICABLE\_TIER\_ID** |
| 1 | VOUCHER | 10% Off | [today] | [today+30] | 10.00 | DISC10 | Y | NULL | NULL | NULL | 1 |
| 2 | VOUCHER | 20% Off | [today] | [today+30] | 20.00 | DISC20 | N | NULL | NULL | NULL | 3 |
| 3 | VOUCHER | 5% Off | [today] | [today+15] | 5.00 | DISC5 | Y | NULL | NULL | NULL | 1 |
| 4 | SEASONAL | Hello Kitty Redemption | 2025-10-15 | 2025-11-30 | NULL | NULL | N | 500 | 20 | 9 | NULL |
| 5 | SEASONAL | Pumpkin Spice Latte Redemption | 2025-09-01 | 2025-09-30 | NULL | NULL | Y | 300 | 30 | 10 | 3 |
| 6 | SET\_MEAL | Family Combo Discount | [today] | [today+60] | 15.00 | NULL | N | NULL | 50 | 5 | NULL |

Table 5.4 - Records of PROMOTIONS

**Table Name: ORDERS**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ORDER\_ ID** | **CUSTOMER\_ ID** | **ORDER\_ DATE** | **TOTAL\_ AMOUNT** | **DISCOUNT\_AMOUNT** | **FINAL\_AMOUNT** | **PAYMENT\_METHOD** | **STATUS** |
| 5001 | 1001 | [current time] | 50.00 | 5.00 | 45.00 | CARD | COMPLETED |
| 5002 | 1002 | [current time] | 35.00 | 0.00 | 35.00 | CASH | COMPLETED |
| 5003 | 1003 | [current time] | 60.00 | 10.00 | 50.00 | ONLINE | COMPLETED |
| 5004 | 1004 | [current time] | 80.00 | 15.00 | 65.00 | CARD | COMPLETED |
| 5005 | 1005 | [current time] | 25.00 | 0.00 | 25.00 | CASH | COMPLETED |
| 5006 | 1003 | [current time] | 32.99 | 0.00 | 32.99 | CARD | COMPLETED |
| 5007 | 1004 | [current time] | 55.50 | 18.50 | 37.00 | ONLINE | COMPLETED |

Table 5.5 - Records of ORDERS

**Table Name: POINT\_TRANSACTIONS**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TRANSACTION\_ID** | **CUSTOMER\_ID** | **ORDER\_ID** | **PROMOTION\_ID** | **POINTS\_AMOUNT** | **TRANSACTION\_TYPE** | **TRANSACTION\_DATE** | **DESCRIPTION** | **EXPIRY\_DATE** |
| 1 | 1006 | NULL | NULL | 100 | BONUS | [current time] | Expired test points | 2024-12-31 |
| 2 | 1006 | NULL | NULL | 200 | PURCHASE | [current time] | Points expiring soon | [today+15] |
| 3 | 1006 | NULL | NULL | 300 | BONUS | [current time] | Mid-term expiring points | [today+3mo] |
| 4 | 1006 | NULL | NULL | 400 | PURCHASE | [current time] | Standard expiry points | [today+1yr] |
| 5 | 1006 | NULL | NULL | -150 | REDEMPTION | [current time] | Test redemption points | NULL |

Table 5.6 - Records of POINT\_TRANSACTIONS

**Table Name: MEMBERSHIP\_HISTORY**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **HISTORY\_ID** | **CUSTOMER\_ID** | **OLD\_TIER\_ID** | **NEW\_TIER\_ID** | **CHANGE\_DATE** | **CHANGE\_REASON** | **CHANGED\_BY** |
| 1 | 1001 | NULL | 1 | 2025-01-15 09:30:45 | SIGNUP | NULL |
| 2 | 1002 | NULL | 1 | 2025-02-20 14:15:22 | SIGNUP | SYSTEM |
| 3 | 1003 | 1 | 2 | 2025-03-10 10:05:33 | ADMIN | NULL |
| 4 | 1004 | NULL | 1 | 2025-04-05 11:20:18 | SIGNUP | JSMITH |
| 5 | 1005 | 1 | 3 | 2025-05-12 16:45:07 | ADMIN | NULL |

Table 5.7 - Records of MEMBERSHIP\_HISTORY

**Table Name: CUSTOMER\_PROMOTIONS**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CUSTOMER\_ID** | **PROMOTION\_ID** | **DATE\_ACQUIRED** | **IS\_USED** | **USED\_DATE** | **ACQUISITION\_METHOD** |
| 1001 | 1 | [today] | N | NULL | AUTO\_TIER |
| 1002 | 2 | [today] | Y | [today] | POINT\_REDEEM |
| 1003 | 3 | [today] | N | NULL | MANUAL |
| 1004 | 6 | [today] | N | NULL | BIRTHDAY |
| 1005 | 4 | [today] | Y | [today] | AUTO\_TIER |

Table 5.8 - Records of CUSTOMER\_PROMOTIONS

**Table Name: REDEMPTIONS**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **REDEMPTION\_ID** | **CUSTOMER\_ID** | **PROMOTION\_ID** | **ORDER\_ID** | **POINTS\_USED** | **REDEMPTION\_ DATE** | **REDEMPTION\_ STATUS** | **ITEM\_ID** |
| 1 | 1001 | 1 | 5001 | 100 | [current time] | COMPLETED | NULL |
| 2 | 1002 | 2 | 5002 | 300 | [current time] | COMPLETED | NULL |
| 3 | 1003 | 3 | 5003 | 500 | [current time] | COMPLETED | NULL |
| 4 | 1004 | 6 | 5007 | 300 | [current time] | COMPLETED | 10 |
| 5 | 1005 | 4 | NULL | 200 | [current time] | COMPLETED | 9 |

Table 5.9 - Records of REDEMTIONS

**Table Name: SET\_MEAL\_COMPONENTS**

|  |  |  |
| --- | --- | --- |
| **SET\_MEAL\_ID** | **COMPONENT\_ID** | **QUANTITY** |
| 5 | 1 | 2 |
| 5 | 7 | 2 |
| 5 | 8 | 2 |
| 6 | 1 | 1 |
| 6 | 7 | 1 |
| 6 | 8 | 1 |

Table 5.10 - Records of SET\_MEAL\_COMPONENTS

**Table Name: ORDER\_ITEMS**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ORDER\_ITEM\_ID** | **ORDER\_ID** | **ITEM\_ID** | **QUANTITY** | **PRICE** | **DISCOUNT\_APPLIED** |
| 1 | 5001 | 1 | 1 | 8.90 | 0 |
| 2 | 5002 | 3 | 2 | 7.00 | 0 |
| 3 | 5003 | 2 | 1 | 24.90 | 5.00 |
| 4 | 5004 | 5 | 1 | 7.00 | 1.00 |
| 5 | 5005 | 4 | 1 | 6.90 | 0 |
| 6 | 5006 | 9 | 1 | 12.99 | 0 |
| 7 | 5006 | 1 | 2 | 8.90 | 0 |
| 8 | 5007 | 10 | 1 | 18.50 | 18.50 |
| 9 | 5007 | 2 | 1 | 24.90 | 0 |

Table 5.11 - Records of ORDER\_ITEMS

**Table Name: CUSTOMER\_ACTIVITY**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ACTIVITY\_ID** | **CUSTOMER\_ID** | **ACTIVITY\_TYPE** | **ACTIVITY\_DATE** | **IP\_ADDRESS** | **DETAILS** |
| 1 | 1001 | LOGIN | [current time] | 192.168.1.10 | Logged in from mobile |
| 2 | 1002 | ORDER | [current time] | 192.168.1.11 | Ordered cheeseburger combo |
| 3 | 1003 | VOUCHER\_REDEMPTION | [current time] | 192.168.1.12 | Redeemed voucher DISC20 |
| 4 | 1004 | TIER\_CHANGE | [current time] | 192.168.1.13 | Upgraded to Platinum |
| 5 | 1005 | LOGIN | [current time] | 192.168.1.14 | Web login |
| 6 | 1003 | SEASONAL\_REDEMPTION | [current time] | 192.168.1.12 | Redeemed Hello Kitty |
| 7 | 1004 | SEASONAL\_REDEMPTION | [current time] | 192.168.1.13 | Redeemed Mooncake set |

Table 5.12 - Records of CUSTOMER\_ACTIVITY

# Group SQL Script

*/\**

*COURSE CODE: UCCD2303 Database Technology*

*PROGRAMME: CS*

*GROUP NUMBER: G033*

*GROUP LEADER NAME & EMAIL: HII ZI WEI cziwei0112@1utar.my*

*MEMBER 2 NAME: CHIA YUE SHENG*

*MEMBER 3 NAME: LEE HIEN LEONG*

*MEMBER 4 NAME: TEH BEE LING*

*Submission date and time (DD-MON-YY): 29 APRIL 2025*

*\*/*

*-- Execution Order Mandatory*

*/\**

*1. groupscript.sql - Base schema*

*2. personal\_script\_1.sql - Membership system*

*3. personal\_script\_2.sql - Voucher system (depends on 1e)*

*4. personal\_script\_3.sql - setmeal system*

*5. personal\_script\_4.sql - seasonalsystem \*/*

*-- Alter session*

ALTER SESSION SET "\_oracle\_script" = true;

*-- Clean up*

*-- Drop Tables (in reverse dependency order)*

DROP TABLE CUSTOMER\_ACTIVITY CASCADE CONSTRAINTS;

DROP TABLE ORDER\_ITEMS CASCADE CONSTRAINTS;

DROP TABLE ORDERS CASCADE CONSTRAINTS;

DROP TABLE REDEMPTIONS CASCADE CONSTRAINTS;

DROP TABLE SET\_MEAL\_COMPONENTS CASCADE CONSTRAINTS;

DROP TABLE CUSTOMER\_PROMOTIONS CASCADE CONSTRAINTS;

DROP TABLE PROMOTIONS CASCADE CONSTRAINTS;

DROP TABLE POINT\_TRANSACTIONS CASCADE CONSTRAINTS;

DROP TABLE MEMBERSHIP\_HISTORY CASCADE CONSTRAINTS;

DROP TABLE MENU\_ITEMS CASCADE CONSTRAINTS;

DROP TABLE CUSTOMERS CASCADE CONSTRAINTS;

DROP TABLE MEMBERSHIP\_TIERS CASCADE CONSTRAINTS;

*-- Drop Sequences*

DROP SEQUENCE activity\_seq;

DROP SEQUENCE order\_item\_seq;

DROP SEQUENCE order\_seq;

DROP SEQUENCE redemption\_seq;

DROP SEQUENCE menu\_item\_seq;

DROP SEQUENCE tier\_seq;

DROP SEQUENCE customer\_seq;

DROP SEQUENCE promo\_seq;

DROP SEQUENCE history\_seq;

*-- Drop Users*

DROP USER admin\_user;

DROP USER manager\_user;

DROP USER voucher\_user;

DROP USER member\_user;

DROP USER report\_user;

*-- Drop Roles*

DROP ROLE promo\_admin\_role;

DROP ROLE promo\_manager\_role;

DROP ROLE voucher\_admin\_role;

DROP ROLE membership\_admin\_role;

DROP ROLE promo\_report\_role;

*-- Drop Trigger*

DROP TRIGGER trg\_birthday\_check;

DROP TRIGGER trg\_sync\_point\_balance;

DROP TRIGGER trg\_membership\_history;

DROP TRIGGER trg\_auto\_assign\_vouchers;

DROP TRIGGER trg\_order\_totals;

DROP TRIGGER trg\_promo\_usage;

*-- Create roles*

CREATE ROLE promo\_admin\_role;

CREATE ROLE promo\_manager\_role;

CREATE ROLE voucher\_admin\_role;

CREATE ROLE membership\_admin\_role;

CREATE ROLE promo\_report\_role;

*-- Create users*

CREATE USER admin\_user IDENTIFIED BY "Admin@1234" DEFAULT TABLESPACE users;

CREATE USER manager\_user IDENTIFIED BY "Manager@1234" DEFAULT TABLESPACE users;

CREATE USER voucher\_user IDENTIFIED BY "Voucher@1234" DEFAULT TABLESPACE users;

CREATE USER member\_user IDENTIFIED BY "Member@1234" DEFAULT TABLESPACE users;

CREATE USER report\_user IDENTIFIED BY "Report@1234" DEFAULT TABLESPACE users;

*-- Grant roles to users*

GRANT promo\_admin\_role TO admin\_user;

GRANT promo\_manager\_role TO manager\_user;

GRANT voucher\_admin\_role TO voucher\_user;

GRANT membership\_admin\_role TO member\_user;

GRANT promo\_report\_role TO report\_user;

*--Grant CREATE SESSION privilege to all users*

GRANT CREATE SESSION TO admin\_user, manager\_user, voucher\_user, member\_user, report\_user;

*-- Create sequences*

CREATE SEQUENCE tier\_seq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE customer\_seq START WITH 1001 INCREMENT BY 1;

CREATE SEQUENCE menu\_item\_seq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE promo\_seq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE history\_seq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE order\_seq START WITH 5001 INCREMENT BY 1;

CREATE SEQUENCE order\_item\_seq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE redemption\_seq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE activity\_seq START WITH 1 INCREMENT BY 1;

*-- Create tables with inline constraints*

CREATE TABLE MEMBERSHIP\_TIERS (

    tier\_id NUMBER DEFAULT tier\_seq.NEXTVAL PRIMARY KEY,

    tier\_name VARCHAR2(20) NOT NULL UNIQUE,

    points\_required NUMBER NOT NULL CHECK (points\_required >= 0),

    discount\_percentage NUMBER(5,2) NOT NULL CHECK (discount\_percentage BETWEEN 0 AND 100),

    renewal\_fee NUMBER(10,2) DEFAULT 0,

    birthday\_bonus\_points NUMBER DEFAULT 0,

    points\_expiry\_months NUMBER DEFAULT 12,

    base\_earn\_rate NUMBER(5,2) DEFAULT 1.0,

    created\_date TIMESTAMP DEFAULT SYSTIMESTAMP

);

CREATE TABLE CUSTOMERS (

    customer\_id NUMBER DEFAULT customer\_seq.NEXTVAL PRIMARY KEY,

    name VARCHAR2(100) NOT NULL,

    email VARCHAR2(100) NOT NULL UNIQUE,

    phone VARCHAR2(15) NOT NULL UNIQUE,

    tier\_id NUMBER NOT NULL CONSTRAINT fk\_customer\_tier REFERENCES MEMBERSHIP\_TIERS(tier\_id) ON DELETE CASCADE,

    points\_balance NUMBER DEFAULT 0 CHECK (points\_balance >= 0),

    last\_points\_earned\_date DATE,

    points\_expiry\_date DATE,

    membership\_expiry\_date DATE,

    is\_member CHAR(1) DEFAULT 'N' CHECK (is\_member IN ('Y','N')),

    membership\_status VARCHAR2(20) DEFAULT 'ACTIVE' CHECK (membership\_status IN ('ACTIVE','CANCELLED','PENDING')),

    last\_visit\_date DATE,

    join\_date DATE DEFAULT SYSDATE,

    birth\_date DATE,

    last\_renewal\_date DATE,

    renewal\_count NUMBER DEFAULT 0

);

CREATE TABLE MENU\_ITEMS (

    item\_id NUMBER DEFAULT menu\_item\_seq.NEXTVAL PRIMARY KEY,

    item\_type VARCHAR2(20) NOT NULL CHECK (item\_type IN ('A\_LA\_CARTE','SET\_MEAL','INGREDIENT','SEASONAL')),

    name VARCHAR2(100) NOT NULL,

    description VARCHAR2(200),

    base\_price NUMBER(10,2) NOT NULL CHECK (base\_price >= 0),

    is\_active CHAR(1) DEFAULT 'Y' CHECK (is\_active IN ('Y','N')),

    valid\_from DATE DEFAULT SYSDATE,

    valid\_to DATE,

    current\_stock NUMBER,

    initial\_stock NUMBER,

    CONSTRAINT chk\_menu\_dates CHECK (valid\_to IS NULL OR valid\_to >= valid\_from)

);

CREATE TABLE PROMOTIONS (

    promotion\_id NUMBER DEFAULT promo\_seq.NEXTVAL PRIMARY KEY,

    promotion\_type VARCHAR2(20) NOT NULL CHECK (promotion\_type IN ('VOUCHER','SEASONAL','SET\_MEAL')),

    name VARCHAR2(100) NOT NULL,

    valid\_from DATE NOT NULL,

    valid\_to DATE NOT NULL,

    discount\_value NUMBER(10,2),

    voucher\_code VARCHAR2(50) UNIQUE,

    is\_auto\_assign CHAR(1) DEFAULT 'N' CHECK (is\_auto\_assign IN ('Y','N')),

    points\_required NUMBER,

    min\_spend NUMBER(10,2),

    set\_meal\_id NUMBER CONSTRAINT fk\_promo\_meal REFERENCES MENU\_ITEMS(item\_id) ON DELETE CASCADE,

    applicable\_tier\_id NUMBER CONSTRAINT fk\_promo\_tier REFERENCES MEMBERSHIP\_TIERS(tier\_id) ON DELETE CASCADE,

    CONSTRAINT chk\_disjoint\_promo\_attrs CHECK (

        (promotion\_type = 'VOUCHER' AND discount\_value IS NOT NULL) OR

        (promotion\_type = 'SEASONAL' AND points\_required IS NOT NULL) OR

        (promotion\_type = 'SET\_MEAL' AND set\_meal\_id IS NOT NULL)

    ),

    CONSTRAINT chk\_valid\_dates CHECK (valid\_to >= valid\_from)

);

CREATE TABLE ORDERS (

    order\_id NUMBER DEFAULT order\_seq.NEXTVAL PRIMARY KEY,

    customer\_id NUMBER NOT NULL CONSTRAINT fk\_order\_customer REFERENCES CUSTOMERS(customer\_id) ON DELETE CASCADE,

    order\_date TIMESTAMP DEFAULT SYSTIMESTAMP,

    total\_amount NUMBER(10,2) NOT NULL CHECK (total\_amount >= 0),

    discount\_amount NUMBER(10,2) DEFAULT 0 CHECK (discount\_amount >= 0),

    final\_amount NUMBER(10,2) NOT NULL CHECK (final\_amount >= 0),

    payment\_method VARCHAR2(50) CHECK (payment\_method IN ('CASH','CARD','ONLINE')),

    status VARCHAR2(20) DEFAULT 'COMPLETED' CHECK (status IN ('COMPLETED','CANCELLED','REFUNDED')),

    CONSTRAINT chk\_order\_amounts CHECK (final\_amount = total\_amount - discount\_amount)

);

CREATE TABLE POINT\_TRANSACTIONS (

    transaction\_id NUMBER DEFAULT history\_seq.NEXTVAL PRIMARY KEY,

    customer\_id NUMBER NOT NULL CONSTRAINT fk\_pt\_customer REFERENCES CUSTOMERS(customer\_id) ON DELETE CASCADE,

    order\_id NUMBER CONSTRAINT fk\_pt\_order REFERENCES ORDERS(order\_id) ON DELETE CASCADE,

    promotion\_id NUMBER CONSTRAINT fk\_pt\_promotion REFERENCES PROMOTIONS(promotion\_id) ON DELETE CASCADE,

    points\_amount NUMBER NOT NULL,

    transaction\_type VARCHAR2(20) NOT NULL CHECK (transaction\_type IN

        ('PURCHASE','REDEMPTION','BONUS','ADJUSTMENT','EXPIRY')),

    transaction\_date TIMESTAMP DEFAULT SYSTIMESTAMP NOT NULL,

    description VARCHAR2(200),

    expiry\_date DATE,

    CONSTRAINT chk\_points\_amount CHECK (

        (transaction\_type IN ('REDEMPTION','EXPIRY') AND points\_amount <= 0) OR

        (transaction\_type NOT IN ('REDEMPTION','EXPIRY') AND points\_amount >= 0)

    )

);

CREATE TABLE MEMBERSHIP\_HISTORY (

    history\_id NUMBER DEFAULT history\_seq.NEXTVAL PRIMARY KEY,

    customer\_id NUMBER NOT NULL CONSTRAINT fk\_mh\_customer REFERENCES CUSTOMERS(customer\_id) ON DELETE CASCADE,

    old\_tier\_id NUMBER CONSTRAINT fk\_mh\_old\_tier REFERENCES MEMBERSHIP\_TIERS(tier\_id) ON DELETE CASCADE,

    new\_tier\_id NUMBER NOT NULL CONSTRAINT fk\_mh\_new\_tier REFERENCES MEMBERSHIP\_TIERS(tier\_id) ON DELETE CASCADE,

    change\_date TIMESTAMP DEFAULT SYSTIMESTAMP,

    change\_reason VARCHAR2(100) CHECK (change\_reason IN ('SIGNUP','RENEWAL','UPGRADE','DOWNGRADE','ADMIN','AUTO')),

    changed\_by VARCHAR2(30) DEFAULT USER

);

CREATE TABLE CUSTOMER\_PROMOTIONS (

    customer\_id NUMBER NOT NULL CONSTRAINT fk\_cp\_customer REFERENCES CUSTOMERS(customer\_id) ON DELETE CASCADE,

    promotion\_id NUMBER NOT NULL CONSTRAINT fk\_cp\_promotion REFERENCES PROMOTIONS(promotion\_id) ON DELETE CASCADE,

    date\_acquired DATE DEFAULT SYSDATE,

    is\_used CHAR(1) DEFAULT 'N' CHECK (is\_used IN ('Y','N')),

    used\_date DATE,

    acquisition\_method VARCHAR2(20) CHECK (acquisition\_method IN ('AUTO\_TIER','POINT\_REDEEM','MANUAL','BIRTHDAY')),

    PRIMARY KEY (customer\_id, promotion\_id),

    CONSTRAINT chk\_used\_date CHECK (is\_used = 'N' OR used\_date IS NOT NULL)

);

CREATE TABLE REDEMPTIONS (

    redemption\_id NUMBER DEFAULT redemption\_seq.NEXTVAL PRIMARY KEY,

    customer\_id NUMBER NOT NULL CONSTRAINT fk\_redemption\_customer REFERENCES CUSTOMERS(customer\_id) ON DELETE CASCADE,

    promotion\_id NUMBER NOT NULL CONSTRAINT fk\_redemption\_promotion REFERENCES PROMOTIONS(promotion\_id) ON DELETE CASCADE,

    order\_id NUMBER CONSTRAINT fk\_redemption\_order REFERENCES ORDERS(order\_id) ON DELETE CASCADE,

    points\_used NUMBER NOT NULL CHECK (points\_used >= 0),

    redemption\_date TIMESTAMP DEFAULT SYSTIMESTAMP,

    redemption\_status VARCHAR2(20) DEFAULT 'COMPLETED' CHECK (redemption\_status IN ('COMPLETED','CANCELLED')),

    item\_id NUMBER CONSTRAINT fk\_redemption\_item REFERENCES MENU\_ITEMS(item\_id) ON DELETE CASCADE

);

CREATE TABLE SET\_MEAL\_COMPONENTS (

    set\_meal\_id NUMBER NOT NULL CONSTRAINT fk\_smc\_meal REFERENCES MENU\_ITEMS(item\_id) ON DELETE CASCADE,

    component\_id NUMBER NOT NULL CONSTRAINT fk\_smc\_component REFERENCES MENU\_ITEMS(item\_id) ON DELETE CASCADE,

    quantity NUMBER DEFAULT 1 NOT NULL CHECK (quantity > 0),

    PRIMARY KEY (set\_meal\_id, component\_id),

    CONSTRAINT no\_self\_reference CHECK (set\_meal\_id != component\_id)

);

CREATE TABLE ORDER\_ITEMS (

    order\_item\_id NUMBER DEFAULT order\_item\_seq.NEXTVAL PRIMARY KEY,

    order\_id NUMBER NOT NULL CONSTRAINT fk\_oi\_order REFERENCES ORDERS(order\_id) ON DELETE CASCADE,

    item\_id NUMBER NOT NULL CONSTRAINT fk\_oi\_item REFERENCES MENU\_ITEMS(item\_id) ON DELETE CASCADE,

    quantity NUMBER DEFAULT 1 CHECK (quantity > 0),

    price NUMBER(10,2) NOT NULL CHECK (price >= 0),

    discount\_applied NUMBER(10,2) DEFAULT 0 CHECK (discount\_applied >= 0)

);

CREATE TABLE CUSTOMER\_ACTIVITY (

    activity\_id NUMBER DEFAULT activity\_seq.NEXTVAL PRIMARY KEY,

    customer\_id NUMBER CONSTRAINT fk\_ca\_customer REFERENCES CUSTOMERS(customer\_id) ON DELETE CASCADE,

    activity\_type VARCHAR2(30) CHECK (activity\_type IN ('LOGIN', 'ORDER','SEASONAL\_REDEMPTION','SEASONAL\_EXPIRATION', 'VOUCHER\_REDEMPTION','VOUCHER\_GENERATION', 'TIER\_CHANGE', 'RENEWAL', 'EXPIRATION', 'VOUCHER\_ASSIGNMENT')),

    activity\_date TIMESTAMP DEFAULT SYSTIMESTAMP,

    ip\_address VARCHAR2(45),

    details VARCHAR2(4000)

);

CREATE OR REPLACE PROCEDURE LOG\_ACTIVITY(

    p\_customer\_id   IN NUMBER,

    p\_activity\_type IN VARCHAR2,

    p\_details       IN VARCHAR2,

    p\_ip\_address    IN VARCHAR2 DEFAULT NULL

) IS

    PRAGMA AUTONOMOUS\_TRANSACTION;

BEGIN

    INSERT INTO CUSTOMER\_ACTIVITY (

        customer\_id,

        activity\_type,

        activity\_date,

        ip\_address,

        details

    ) VALUES (

        p\_customer\_id,

        p\_activity\_type,

        SYSTIMESTAMP,

        p\_ip\_address,  *-- Will be NULL if not passed*

        p\_details

    );

    COMMIT;

END;

/

*-- Insert data*

*-- MEMBERSHIP\_TIERS*

INSERT INTO MEMBERSHIP\_TIERS (tier\_name, points\_required, discount\_percentage, renewal\_fee, birthday\_bonus\_points, points\_expiry\_months, base\_earn\_rate)

VALUES ('Bronze', 0, 5.00, 10.00, 50, 12, 1.0);

INSERT INTO MEMBERSHIP\_TIERS (tier\_name, points\_required, discount\_percentage, renewal\_fee, birthday\_bonus\_points, points\_expiry\_months, base\_earn\_rate)

VALUES ('Silver', 500, 10.00, 15.00, 100, 12, 1.2);

INSERT INTO MEMBERSHIP\_TIERS (tier\_name, points\_required, discount\_percentage, renewal\_fee, birthday\_bonus\_points, points\_expiry\_months, base\_earn\_rate)

VALUES ('Gold', 1500, 15.00, 20.00, 150, 12, 1.5);

INSERT INTO MEMBERSHIP\_TIERS (tier\_name, points\_required, discount\_percentage, renewal\_fee, birthday\_bonus\_points, points\_expiry\_months, base\_earn\_rate)

VALUES ('Platinum', 3000, 20.00, 25.00, 200, 12, 2.0);

INSERT INTO MEMBERSHIP\_TIERS (tier\_name, points\_required, discount\_percentage, renewal\_fee, birthday\_bonus\_points, points\_expiry\_months, base\_earn\_rate)

VALUES ('Diamond', 5000, 25.00, 30.00, 250, 12, 2.5);

*-- MENU\_ITEMS*

INSERT INTO MENU\_ITEMS (item\_type, name, description, base\_price, is\_active, valid\_from, valid\_to, current\_stock, initial\_stock)

VALUES ('A\_LA\_CARTE', 'Cheeseburger', 'Classic beef burger with cheese', 8.90, 'Y', NULL, NULL, NULL, NULL);

INSERT INTO MENU\_ITEMS (item\_type, name, description, base\_price, is\_active, valid\_from, valid\_to, current\_stock, initial\_stock)

VALUES ('A\_LA\_CARTE', 'Chicken Nuggets', '6-piece nuggets', 7.00, 'Y', NULL, NULL, NULL, NULL);

INSERT INTO MENU\_ITEMS (item\_type, name, description, base\_price, is\_active, valid\_from, valid\_to, current\_stock, initial\_stock)

VALUES ('A\_LA\_CARTE', 'Garden Salad', 'Fresh greens with dressing', 5.50, 'Y', NULL, NULL, NULL, NULL);

INSERT INTO MENU\_ITEMS (item\_type, name, description, base\_price, is\_active, valid\_from, valid\_to, current\_stock, initial\_stock)

VALUES ('A\_LA\_CARTE', 'Chocolate Shake', 'Creamy chocolate milkshake', 4.50, 'Y', NULL, NULL, NULL, NULL);

INSERT INTO MENU\_ITEMS (item\_type, name, description, base\_price, is\_active, valid\_from, valid\_to, current\_stock, initial\_stock)

VALUES ('SET\_MEAL', 'Family Combo', 'Includes 2 burgers, 2 fries, and 2 drinks', 24.90, 'Y', NULL, NULL, NULL, NULL);

INSERT INTO MENU\_ITEMS (item\_type, name, description, base\_price, is\_active, valid\_from, valid\_to, current\_stock, initial\_stock)

VALUES ('SET\_MEAL', 'Value Meal', 'Burger, fries, and drink', 12.50, 'Y', NULL, NULL, NULL, NULL);

INSERT INTO MENU\_ITEMS (item\_type, name, description, base\_price, is\_active, valid\_from, valid\_to, current\_stock, initial\_stock)

VALUES ('INGREDIENT', 'Fries', 'Crispy golden fries', 3.50, 'Y', NULL, NULL, 5000, 5000);

INSERT INTO MENU\_ITEMS (item\_type, name, description, base\_price, is\_active, valid\_from, valid\_to, current\_stock, initial\_stock)

VALUES ('INGREDIENT', 'Soft Drink', 'Regular fountain drink', 2.50, 'Y', NULL, NULL, 3000, 3000);

INSERT INTO MENU\_ITEMS (item\_type, name, description, base\_price, is\_active, valid\_from, valid\_to, current\_stock, initial\_stock)

VALUES ('SEASONAL', 'Hello Kitty Happy Meal', 'Toy + Kids Meal (Limited Edition)', 12.99, 'Y', DATE '2025-10-15', DATE '2025-11-30', 1000, 1000);

INSERT INTO MENU\_ITEMS (item\_type, name, description, base\_price, is\_active, valid\_from, valid\_to, current\_stock, initial\_stock)

VALUES ('SEASONAL', 'Pumpkin Spice Latte', 'Seasonal autumn drink', 5.90, 'Y', DATE '2025-09-15', DATE '2025-11-15', 200, 200);

INSERT INTO MENU\_ITEMS (item\_type, name, description, base\_price, is\_active, valid\_from, valid\_to, current\_stock, initial\_stock)

VALUES ('SEASONAL', 'Festival Mooncake Set', 'Premium mooncakes (Mid-Autumn Special)', 18.50, 'Y', DATE '2025-09-01', DATE '2025-09-30', 500, 500);

INSERT INTO MENU\_ITEMS (item\_type, name, description, base\_price, is\_active, valid\_from, valid\_to, current\_stock, initial\_stock)

VALUES ('SEASONAL', 'Summer BBQ Burger', 'Special summer edition burger', 9.99, 'Y', DATE '2025-06-01', DATE '2025-08-31', 800, 800);

INSERT INTO MENU\_ITEMS (item\_type, name, description, base\_price, is\_active, valid\_from, valid\_to, current\_stock, initial\_stock)

VALUES ('SEASONAL', 'Winter Hot Chocolate', 'Rich hot chocolate with marshmallows', 4.99, 'Y', DATE '2025-12-01', DATE '2026-02-28', 300, 300);

*-- SET\_MEAL\_COMPONENTS*

INSERT INTO SET\_MEAL\_COMPONENTS VALUES (5, 1, 2);

INSERT INTO SET\_MEAL\_COMPONENTS VALUES (5, 7, 2);

INSERT INTO SET\_MEAL\_COMPONENTS VALUES (5, 8, 2);

INSERT INTO SET\_MEAL\_COMPONENTS VALUES (6, 1, 1);

INSERT INTO SET\_MEAL\_COMPONENTS VALUES (6, 7, 1);

INSERT INTO SET\_MEAL\_COMPONENTS VALUES (6, 8, 1);

*-- CUSTOMERS*

INSERT INTO CUSTOMERS (name, email, phone, tier\_id, points\_balance, last\_points\_earned\_date, points\_expiry\_date, membership\_expiry\_date, is\_member, membership\_status, last\_visit\_date, birth\_date, last\_renewal\_date, renewal\_count)

VALUES ('John Smith', 'johnsmith@example.com', '012-3456789', 1, 300, DATE '2024-12-01', DATE '2025-12-01', DATE '2025-12-01', 'Y', 'ACTIVE', DATE '2025-04-01', DATE '1990-04-07', NULL, 0);

INSERT INTO CUSTOMERS (name, email, phone, tier\_id, points\_balance, last\_points\_earned\_date, points\_expiry\_date, membership\_expiry\_date, is\_member, membership\_status, last\_visit\_date, birth\_date, last\_renewal\_date, renewal\_count)

VALUES ('Emily Johnson', 'emilyj@example.com', '013-2233445', 2, 750, DATE '2025-03-21', DATE '2026-03-21', DATE '2026-03-21', 'Y', 'ACTIVE', DATE '2025-03-15', DATE '1988-07-12', NULL, 0);

INSERT INTO CUSTOMERS (name, email, phone, tier\_id, points\_balance, last\_points\_earned\_date, points\_expiry\_date, membership\_expiry\_date, is\_member, membership\_status, last\_visit\_date, birth\_date, last\_renewal\_date, renewal\_count)

VALUES ('Michael Lee', 'mikelee@example.com', '014-5566778', 3, 1000, DATE '2025-01-10', DATE '2026-01-10', DATE '2026-01-10', 'Y', 'ACTIVE', DATE '2025-01-11', DATE '1995-11-30', NULL, 0);

INSERT INTO CUSTOMERS (name, email, phone, tier\_id, points\_balance, last\_points\_earned\_date, points\_expiry\_date, membership\_expiry\_date, is\_member, membership\_status, last\_visit\_date, birth\_date, last\_renewal\_date, renewal\_count)

VALUES ('Sarah Tan', 'saraht@example.com', '015-9988776', 4, 2000, DATE '2025-02-28', DATE '2026-02-28', DATE '2026-02-28', 'Y', 'ACTIVE', DATE '2025-02-28', DATE '1992-03-18', NULL, 0);

INSERT INTO CUSTOMERS (name, email, phone, tier\_id, points\_balance, last\_points\_earned\_date, points\_expiry\_date, membership\_expiry\_date, is\_member, membership\_status, last\_visit\_date, birth\_date, last\_renewal\_date, renewal\_count)

VALUES ('David Wong', 'davidw@example.com', '016-1122334', 5, 3200, DATE '2025-03-05', DATE '2026-03-05', DATE '2026-03-05', 'Y', 'ACTIVE', DATE '2025-03-05', DATE '1985-05-25', NULL, 0);

INSERT INTO CUSTOMERS (name, email, phone, tier\_id, points\_balance,last\_points\_earned\_date, points\_expiry\_date, membership\_expiry\_date,is\_member, membership\_status, last\_visit\_date, birth\_date,last\_renewal\_date, renewal\_count)

VALUES ('Test User for G001', 'test.user@example.com',  '012-9999999', 1, 500,DATE '2025-03-01',  DATE '2025-12-31',  DATE '2025-03-01',  'Y', 'ACTIVE',  DATE '2025-04-01', DATE '1990-01-01', NULL, 0);

*-- PROMOTIONS*

INSERT INTO PROMOTIONS (promotion\_type, name, valid\_from, valid\_to, discount\_value, voucher\_code, is\_auto\_assign, points\_required, min\_spend, set\_meal\_id, applicable\_tier\_id)

VALUES ('VOUCHER', '10% Off', SYSDATE, SYSDATE+30, 10.00, 'DISC10', 'Y', NULL, NULL, NULL, 1);

INSERT INTO PROMOTIONS (promotion\_type, name, valid\_from, valid\_to, discount\_value, voucher\_code, is\_auto\_assign, points\_required, min\_spend, set\_meal\_id, applicable\_tier\_id)

VALUES ('VOUCHER', '20% Off', SYSDATE, SYSDATE+30, 20.00, 'DISC20', 'N', NULL, NULL, NULL, 3);

INSERT INTO PROMOTIONS (promotion\_type, name, valid\_from, valid\_to, discount\_value, voucher\_code, is\_auto\_assign, points\_required, min\_spend, set\_meal\_id, applicable\_tier\_id)

VALUES ('VOUCHER', '5% Off', SYSDATE, SYSDATE+15, 5.00, 'DISC5', 'Y', NULL, NULL, NULL, 1);

INSERT INTO PROMOTIONS (promotion\_type, name, valid\_from, valid\_to, discount\_value, voucher\_code, is\_auto\_assign, points\_required, min\_spend, set\_meal\_id, applicable\_tier\_id)

VALUES ('SEASONAL', 'Hello Kitty Redemption', DATE '2025-10-15', DATE '2025-11-30', NULL, NULL, 'N', 500, 20, 9, NULL);

INSERT INTO PROMOTIONS (promotion\_type, name, valid\_from, valid\_to, discount\_value, voucher\_code, is\_auto\_assign, points\_required, min\_spend, set\_meal\_id, applicable\_tier\_id)

VALUES ('SEASONAL', 'Pumpkin Spice Latte Redemption', DATE '2025-09-01', DATE '2025-09-30', NULL, NULL, 'Y', 300, 30, 10, 3);

INSERT INTO PROMOTIONS (promotion\_type, name, valid\_from, valid\_to, discount\_value, voucher\_code, is\_auto\_assign, points\_required, min\_spend, set\_meal\_id, applicable\_tier\_id)

VALUES ('SET\_MEAL', 'Family Combo Discount', SYSDATE, SYSDATE+60, 15.00, NULL, 'N', NULL, 50, 5, NULL);

*-- ORDERS*

INSERT INTO ORDERS (customer\_id, order\_date, total\_amount, discount\_amount, final\_amount, payment\_method, status)

VALUES (1001, SYSDATE, 50.00, 5.00, 45.00, 'CARD', 'COMPLETED');

INSERT INTO ORDERS (customer\_id, order\_date, total\_amount, discount\_amount, final\_amount, payment\_method, status)

VALUES (1002, SYSDATE, 35.00, 0.00, 35.00, 'CASH', 'COMPLETED');

INSERT INTO ORDERS (customer\_id, order\_date, total\_amount, discount\_amount, final\_amount, payment\_method, status)

VALUES (1003, SYSDATE, 60.00, 10.00, 50.00, 'ONLINE', 'COMPLETED');

INSERT INTO ORDERS (customer\_id, order\_date, total\_amount, discount\_amount, final\_amount, payment\_method, status)

VALUES (1004, SYSDATE, 80.00, 15.00, 65.00, 'CARD', 'COMPLETED');

INSERT INTO ORDERS (customer\_id, order\_date, total\_amount, discount\_amount, final\_amount, payment\_method, status)

VALUES (1005, SYSDATE, 25.00, 0.00, 25.00, 'CASH', 'COMPLETED');

INSERT INTO ORDERS (customer\_id, order\_date, total\_amount, discount\_amount, final\_amount, payment\_method, status)

VALUES (1003, SYSDATE, 32.99, 0.00, 32.99, 'CARD', 'COMPLETED');

INSERT INTO ORDERS (customer\_id, order\_date, total\_amount, discount\_amount, final\_amount, payment\_method, status)

VALUES (1004, SYSDATE, 55.50, 18.50, 37.00, 'ONLINE', 'COMPLETED');

*-- POINT\_TRANSACTIONS*

*-- 1. Points that already expired (for testing expired point scenarios)*

INSERT INTO POINT\_TRANSACTIONS (customer\_id, points\_amount, transaction\_type, description, expiry\_date)

VALUES (1006, 100, 'BONUS', 'Expired test points', DATE '2024-12-31');

*-- 2. Points expiring soon (within 1 month)*

INSERT INTO POINT\_TRANSACTIONS (customer\_id, points\_amount, transaction\_type, description, expiry\_date)

VALUES (1006, 200, 'PURCHASE', 'Points expiring soon', SYSDATE + 15);

*-- 3. Points expiring in 3 months*

INSERT INTO POINT\_TRANSACTIONS (customer\_id, points\_amount, transaction\_type, description, expiry\_date)

VALUES (1006, 300, 'BONUS', 'Mid-term expiring points', ADD\_MONTHS(SYSDATE, 3));

*-- 4. Points expiring in 1 year (standard)*

INSERT INTO POINT\_TRANSACTIONS (customer\_id, points\_amount, transaction\_type, description, expiry\_date)

VALUES (1006, 400, 'PURCHASE', 'Standard expiry points', ADD\_MONTHS(SYSDATE, 12));

*-- 5. Negative points (redemption) - doesn't need expiry*

INSERT INTO POINT\_TRANSACTIONS (customer\_id, points\_amount, transaction\_type, description)

VALUES (1006, -150, 'REDEMPTION', 'Test redemption points');

*-- MEMBERSHIP\_HISTORY*

INSERT INTO MEMBERSHIP\_HISTORY (customer\_id, old\_tier\_id, new\_tier\_id, change\_reason)

VALUES (1001, NULL, 1, 'SIGNUP');

INSERT INTO MEMBERSHIP\_HISTORY (customer\_id, old\_tier\_id, new\_tier\_id, change\_reason, changed\_by)

VALUES (1002, NULL, 1, 'SIGNUP', 'SYSTEM');

INSERT INTO MEMBERSHIP\_HISTORY (customer\_id, old\_tier\_id, new\_tier\_id, change\_reason)

VALUES (1003, 1, 2, 'ADMIN');

INSERT INTO MEMBERSHIP\_HISTORY (customer\_id, old\_tier\_id, new\_tier\_id, change\_reason, changed\_by)

VALUES (1004, NULL, 1, 'SIGNUP', 'JSMITH');

INSERT INTO MEMBERSHIP\_HISTORY (customer\_id, old\_tier\_id, new\_tier\_id, change\_reason)

VALUES (1005, 1, 3, 'ADMIN');

*-- CUSTOMER\_PROMOTIONS*

INSERT INTO CUSTOMER\_PROMOTIONS (customer\_id, promotion\_id, date\_acquired, is\_used, used\_date, acquisition\_method)

VALUES (1001, 1, SYSDATE, 'N', NULL, 'AUTO\_TIER');

INSERT INTO CUSTOMER\_PROMOTIONS (customer\_id, promotion\_id, date\_acquired, is\_used, used\_date, acquisition\_method)

VALUES (1002, 2, SYSDATE, 'Y', SYSDATE, 'POINT\_REDEEM');

INSERT INTO CUSTOMER\_PROMOTIONS (customer\_id, promotion\_id, date\_acquired, is\_used, used\_date, acquisition\_method)

VALUES (1003, 3, SYSDATE, 'N', NULL, 'MANUAL');

INSERT INTO CUSTOMER\_PROMOTIONS (customer\_id, promotion\_id, date\_acquired, is\_used, used\_date, acquisition\_method)

VALUES (1004, 6, SYSDATE, 'N', NULL, 'BIRTHDAY');

INSERT INTO CUSTOMER\_PROMOTIONS (customer\_id, promotion\_id, date\_acquired, is\_used, used\_date, acquisition\_method)

VALUES (1005, 4, SYSDATE, 'Y', SYSDATE, 'AUTO\_TIER');

*-- ORDER\_ITEMS*

INSERT INTO ORDER\_ITEMS (order\_id, item\_id, quantity, price, discount\_applied)

VALUES (5001, 1, 1, 8.90, 0);

INSERT INTO ORDER\_ITEMS (order\_id, item\_id, quantity, price, discount\_applied)

VALUES (5002, 3, 2, 7.00, 0);

INSERT INTO ORDER\_ITEMS (order\_id, item\_id, quantity, price, discount\_applied)

VALUES (5003, 2, 1, 24.90, 5.00);

INSERT INTO ORDER\_ITEMS (order\_id, item\_id, quantity, price, discount\_applied)

VALUES (5004, 5, 1, 7.00, 1.00);

INSERT INTO ORDER\_ITEMS (order\_id, item\_id, quantity, price, discount\_applied)

VALUES (5005, 4, 1, 6.90, 0);

INSERT INTO ORDER\_ITEMS (order\_id, item\_id, quantity, price, discount\_applied)

VALUES (5006, 9, 1, 12.99, 0);

INSERT INTO ORDER\_ITEMS (order\_id, item\_id, quantity, price, discount\_applied)

VALUES (5006, 1, 2, 8.90, 0);

INSERT INTO ORDER\_ITEMS (order\_id, item\_id, quantity, price, discount\_applied)

VALUES (5007, 10, 1, 18.50, 18.50);

INSERT INTO ORDER\_ITEMS (order\_id, item\_id, quantity, price, discount\_applied)

VALUES (5007, 2, 1, 24.90, 0);

*-- REDEMPTIONS*

INSERT INTO REDEMPTIONS (customer\_id, promotion\_id, order\_id, points\_used, redemption\_status, item\_id)

VALUES (1001, 1, 5001, 100, 'COMPLETED', NULL);

INSERT INTO REDEMPTIONS (customer\_id, promotion\_id, order\_id, points\_used, redemption\_status, item\_id)

VALUES (1002, 2, 5002, 300, 'COMPLETED', NULL);

INSERT INTO REDEMPTIONS (customer\_id, promotion\_id, order\_id, points\_used, redemption\_status, item\_id)

VALUES (1003, 3, 5003, 500, 'COMPLETED', NULL);

INSERT INTO REDEMPTIONS (customer\_id, promotion\_id, order\_id, points\_used, redemption\_status, item\_id)

VALUES (1004, 6, 5007, 300, 'COMPLETED', 10);

INSERT INTO REDEMPTIONS (customer\_id, promotion\_id, points\_used, redemption\_status, item\_id)

VALUES (1005, 4, 200, 'COMPLETED', 9);

*-- CUSTOMER\_ACTIVITY*

INSERT INTO CUSTOMER\_ACTIVITY (customer\_id, activity\_type, ip\_address, details)

VALUES (1001, 'LOGIN', '192.168.1.10', 'Logged in from mobile');

INSERT INTO CUSTOMER\_ACTIVITY (customer\_id, activity\_type, ip\_address, details)

VALUES (1002, 'ORDER', '192.168.1.11', 'Ordered cheeseburger combo');

INSERT INTO CUSTOMER\_ACTIVITY (customer\_id, activity\_type, ip\_address, details)

VALUES (1003, 'VOUCHER\_REDEMPTION', '192.168.1.12', 'Redeemed voucher DISC20');

INSERT INTO CUSTOMER\_ACTIVITY (customer\_id, activity\_type, ip\_address, details)

VALUES (1004, 'TIER\_CHANGE', '192.168.1.13', 'Upgraded to Platinum');

INSERT INTO CUSTOMER\_ACTIVITY (customer\_id, activity\_type, ip\_address, details)

VALUES (1005, 'LOGIN', '192.168.1.14', 'Web login');

INSERT INTO CUSTOMER\_ACTIVITY (customer\_id, activity\_type, ip\_address, details)

VALUES (1003, 'SEASONAL\_REDEMPTION', '192.168.1.12', 'Redeemed Hello Kitty');

INSERT INTO CUSTOMER\_ACTIVITY (customer\_id, activity\_type, ip\_address, details)

VALUES (1004, 'SEASONAL\_REDEMPTION', '192.168.1.13', 'Redeemed Mooncake set');

*-- Create Trigger*

CREATE OR REPLACE TRIGGER trg\_sync\_point\_balance

FOR INSERT OR UPDATE OR DELETE ON POINT\_TRANSACTIONS

COMPOUND TRIGGER

*-- Type declarations for batch processing*

    TYPE customer\_rec\_type IS RECORD (

        points\_change NUMBER,

        needs\_expiry\_recalc BOOLEAN

    );

    TYPE customer\_map\_type IS TABLE OF customer\_rec\_type INDEX BY PLS\_INTEGER;

    v\_customer\_data customer\_map\_type;

*-- After each row - collect changes*

    AFTER EACH ROW IS

    BEGIN

*-- Initialize record for this customer if not exists*

        IF NOT v\_customer\_data.EXISTS(

            CASE WHEN INSERTING OR UPDATING THEN :NEW.customer\_id ELSE :OLD.customer\_id END

        ) THEN

            v\_customer\_data(

                CASE WHEN INSERTING OR UPDATING THEN :NEW.customer\_id ELSE :OLD.customer\_id END

            ) := customer\_rec\_type(0, FALSE);

        END IF;

*-- Handle points change*

        IF INSERTING THEN

            v\_customer\_data(:NEW.customer\_id).points\_change :=

                v\_customer\_data(:NEW.customer\_id).points\_change + :NEW.points\_amount;

*-- Mark if we need to recalculate expiry date*

            IF :NEW.points\_amount > 0 AND

               :NEW.transaction\_type IN ('PURCHASE','BONUS') AND

               (:NEW.expiry\_date IS NOT NULL) THEN

                v\_customer\_data(:NEW.customer\_id).needs\_expiry\_recalc := TRUE;

            END IF;

        ELSIF UPDATING THEN

            v\_customer\_data(:NEW.customer\_id).points\_change :=

                v\_customer\_data(:NEW.customer\_id).points\_change +

                (:NEW.points\_amount - NVL(:OLD.points\_amount, 0));

*-- Mark if we need to recalculate expiry date*

            IF (:NEW.points\_amount > 0 AND

                :NEW.transaction\_type IN ('PURCHASE','BONUS') AND

                (:NEW.expiry\_date IS NOT NULL)) OR

               (:OLD.points\_amount > 0 AND

                :OLD.transaction\_type IN ('PURCHASE','BONUS') AND

                (:OLD.expiry\_date IS NOT NULL)) THEN

                v\_customer\_data(:NEW.customer\_id).needs\_expiry\_recalc := TRUE;

            END IF;

        ELSE *-- DELETING*

            v\_customer\_data(:OLD.customer\_id).points\_change :=

                v\_customer\_data(:OLD.customer\_id).points\_change - :OLD.points\_amount;

*-- Mark if we need to recalculate expiry date*

            IF :OLD.points\_amount > 0 AND

               :OLD.transaction\_type IN ('PURCHASE','BONUS') AND

               (:OLD.expiry\_date IS NOT NULL) THEN

                v\_customer\_data(:OLD.customer\_id).needs\_expiry\_recalc := TRUE;

            END IF;

        END IF;

    END AFTER EACH ROW;

*-- After statement - process all collected changes*

    AFTER STATEMENT IS

    BEGIN

*-- Process all customers with changes*

        FOR cust\_id IN v\_customer\_data.FIRST..v\_customer\_data.LAST LOOP

            CONTINUE WHEN NOT v\_customer\_data.EXISTS(cust\_id);

*-- Update customer balance*

            UPDATE CUSTOMERS c

            SET points\_balance = points\_balance + v\_customer\_data(cust\_id).points\_change

            WHERE customer\_id = cust\_id;

*-- Recalculate expiry date if needed*

            IF v\_customer\_data(cust\_id).needs\_expiry\_recalc THEN

                UPDATE CUSTOMERS c

                SET points\_expiry\_date = (

                    SELECT MIN(expiry\_date)

                    FROM POINT\_TRANSACTIONS pt

                    WHERE pt.customer\_id = cust\_id

                    AND pt.expiry\_date > SYSDATE

                    AND pt.transaction\_type IN ('PURCHASE','BONUS')

                    AND pt.points\_amount > 0

                )

                WHERE customer\_id = cust\_id;

            END IF;

*-- Tier upgrade/downgrade logic*

            FOR tier\_rec IN (

                SELECT tier\_id

                FROM MEMBERSHIP\_TIERS

                WHERE points\_required <= (

                    SELECT points\_balance

                    FROM CUSTOMERS

                    WHERE customer\_id = cust\_id

                )

                ORDER BY points\_required DESC

                FETCH FIRST 1 ROW ONLY

            ) LOOP

                UPDATE CUSTOMERS

                SET tier\_id = tier\_rec.tier\_id

                WHERE customer\_id = cust\_id

                AND tier\_id != tier\_rec.tier\_id;

            END LOOP;

        END LOOP;

    END AFTER STATEMENT;

END trg\_sync\_point\_balance;

/

*-- Auto-log membership tier changes*

CREATE OR REPLACE TRIGGER trg\_membership\_history

AFTER UPDATE OF tier\_id ON CUSTOMERS

FOR EACH ROW

BEGIN

    INSERT INTO MEMBERSHIP\_HISTORY (

        customer\_id, old\_tier\_id, new\_tier\_id,

        change\_reason, changed\_by

    ) VALUES (

        :NEW.customer\_id, :OLD.tier\_id, :NEW.tier\_id,

        CASE

            WHEN :OLD.tier\_id IS NULL THEN 'SIGNUP'

            WHEN :NEW.tier\_id > :OLD.tier\_id THEN 'UPGRADE'

            WHEN :NEW.tier\_id < :OLD.tier\_id THEN 'DOWNGRADE'

            ELSE 'ADMIN'

        END,

        USER

    );

END;

/

*-- Auto-assign vouchers on tier change*

CREATE OR REPLACE TRIGGER trg\_auto\_assign\_vouchers

AFTER UPDATE OF tier\_id ON CUSTOMERS

FOR EACH ROW

DECLARE

    CURSOR c\_vouchers IS

        SELECT promotion\_id FROM PROMOTIONS

        WHERE is\_auto\_assign = 'Y'

        AND valid\_from <= SYSDATE AND valid\_to >= SYSDATE

        AND (applicable\_tier\_id IS NULL OR applicable\_tier\_id = :NEW.tier\_id);

BEGIN

    FOR v\_rec IN c\_vouchers LOOP

        BEGIN

            INSERT INTO CUSTOMER\_PROMOTIONS (

                customer\_id, promotion\_id, acquisition\_method

            ) VALUES (

                :NEW.customer\_id, v\_rec.promotion\_id,

                CASE

                    WHEN :OLD.tier\_id IS NULL THEN 'AUTO\_TIER'

                    ELSE 'TIER\_UPGRADE'

                END

            );

        EXCEPTION

            WHEN DUP\_VAL\_ON\_INDEX THEN

                NULL; *-- Skip if already assigned*

        END;

    END LOOP;

END;

/

*-- Auto-update order totals*

CREATE OR REPLACE TRIGGER trg\_order\_totals

FOR INSERT OR UPDATE OR DELETE ON ORDER\_ITEMS

COMPOUND TRIGGER

*-- Variables to store order IDs*

    TYPE order\_id\_array IS TABLE OF NUMBER INDEX BY PLS\_INTEGER;

    v\_order\_ids order\_id\_array;

*-- After each row - collect affected order IDs*

    AFTER EACH ROW IS

    BEGIN

        IF INSERTING OR UPDATING THEN

            v\_order\_ids(:NEW.order\_id) := 1;

        ELSE

            v\_order\_ids(:OLD.order\_id) := 1;

        END IF;

    END AFTER EACH ROW;

*-- After statement - process all collected orders*

    AFTER STATEMENT IS

    BEGIN

        FOR i IN v\_order\_ids.FIRST..v\_order\_ids.LAST LOOP

            IF v\_order\_ids.EXISTS(i) THEN

                UPDATE ORDERS o

                SET total\_amount = NVL((

                        SELECT SUM(price \* quantity)

                        FROM ORDER\_ITEMS

                        WHERE order\_id = i

                    ), 0),

                    discount\_amount = NVL((

                        SELECT SUM(discount\_applied \* quantity)

                        FROM ORDER\_ITEMS

                        WHERE order\_id = i

                    ), 0),

                    final\_amount = NVL((

                        SELECT SUM((price - discount\_applied) \* quantity)

                        FROM ORDER\_ITEMS

                        WHERE order\_id = i

                    ), 0)

                WHERE order\_id = i;

            END IF;

        END LOOP;

    END AFTER STATEMENT;

END trg\_order\_totals;

/

*-- Track promotion usage*

CREATE OR REPLACE TRIGGER trg\_promo\_usage

AFTER INSERT ON REDEMPTIONS

FOR EACH ROW

BEGIN

    UPDATE CUSTOMER\_PROMOTIONS

    SET is\_used = 'Y',

        used\_date = SYSDATE

    WHERE customer\_id = :NEW.customer\_id

    AND promotion\_id = :NEW.promotion\_id;

END;

/

*-- Birthday check trigger*

CREATE OR REPLACE TRIGGER trg\_birthday\_check

AFTER LOGON ON DATABASE

DECLARE

    v\_today\_month NUMBER;

    v\_today\_day NUMBER;

BEGIN

*-- Get current month/day*

    v\_today\_month := EXTRACT(MONTH FROM SYSDATE);

    v\_today\_day := EXTRACT(DAY FROM SYSDATE);

*-- Process customers with birthdays today*

    FOR cust IN (

        SELECT c.customer\_id, c.tier\_id, mt.birthday\_bonus\_points

        FROM CUSTOMERS c

        JOIN MEMBERSHIP\_TIERS mt ON c.tier\_id = mt.tier\_id

        WHERE EXTRACT(MONTH FROM c.birth\_date) = v\_today\_month

        AND EXTRACT(DAY FROM c.birth\_date) = v\_today\_day

    ) LOOP

        BEGIN

*-- Only proceed if there are bonus points to award*

            IF cust.birthday\_bonus\_points > 0 THEN

*-- Insert into POINT\_TRANSACTIONS*

                INSERT INTO POINT\_TRANSACTIONS (

                    customer\_id,

                    points\_amount,

                    transaction\_type,

                    description,

                    transaction\_date,

                    expiry\_date

                ) VALUES (

                    cust.customer\_id,

                    cust.birthday\_bonus\_points,

                    'BONUS',

                    'Birthday reward points',

                    SYSDATE,

                    ADD\_MONTHS(SYSDATE, 12)

                );

*-- Insert into CUSTOMER\_ACTIVITY*

                INSERT INTO CUSTOMER\_ACTIVITY (

                    customer\_id,

                    activity\_type,

                    details,

                    activity\_date

                ) VALUES (

                    cust.customer\_id,

                    'BIRTHDAY',

                    'Received ' || cust.birthday\_bonus\_points || ' birthday points',

                    SYSDATE

                );

            END IF;

        EXCEPTION

            WHEN OTHERS THEN

                NULL; *-- Suppress errors to prevent login issues*

        END;

    END LOOP;

EXCEPTION

    WHEN OTHERS THEN

        NULL; *-- Prevent login failure due to trigger error*

END;

/

*-- Grant privileges for promo\_admin\_role*

GRANT ALL ON MEMBERSHIP\_TIERS TO promo\_admin\_role;

GRANT ALL ON CUSTOMERS TO promo\_admin\_role;

GRANT ALL ON PROMOTIONS TO promo\_admin\_role;

GRANT ALL ON CUSTOMER\_PROMOTIONS TO promo\_admin\_role;

GRANT ALL ON REDEMPTIONS TO promo\_admin\_role;

*-- Grant privileges for promo\_manager\_role*

GRANT SELECT, INSERT, UPDATE ON CUSTOMERS TO promo\_manager\_role;

GRANT SELECT ON MEMBERSHIP\_TIERS TO promo\_manager\_role;

GRANT SELECT, INSERT, UPDATE ON PROMOTIONS TO promo\_manager\_role;

GRANT SELECT ON CUSTOMER\_PROMOTIONS TO promo\_manager\_role;

GRANT SELECT ON REDEMPTIONS TO promo\_manager\_role;

*-- Grant privileges for voucher\_admin\_role (now focused on promotions and redemptions)*

GRANT SELECT, INSERT, UPDATE ON PROMOTIONS TO voucher\_admin\_role;

GRANT SELECT, INSERT, UPDATE ON CUSTOMER\_PROMOTIONS TO voucher\_admin\_role;

GRANT SELECT, INSERT, UPDATE ON REDEMPTIONS TO voucher\_admin\_role;

GRANT SELECT ON CUSTOMERS TO voucher\_admin\_role;

*-- Grant privileges for membership\_admin\_role*

GRANT SELECT, INSERT, UPDATE ON CUSTOMERS TO membership\_admin\_role;

GRANT SELECT, INSERT, UPDATE ON MEMBERSHIP\_TIERS TO membership\_admin\_role;

GRANT SELECT, INSERT, UPDATE ON MEMBERSHIP\_HISTORY TO membership\_admin\_role;

GRANT SELECT, INSERT, UPDATE ON POINT\_TRANSACTIONS TO membership\_admin\_role;

*-- Grant privileges for promo\_report\_role (read-only)*

GRANT SELECT ON MEMBERSHIP\_TIERS TO promo\_report\_role;

GRANT SELECT ON CUSTOMERS TO promo\_report\_role;

GRANT SELECT ON PROMOTIONS TO promo\_report\_role;

GRANT SELECT ON CUSTOMER\_PROMOTIONS TO promo\_report\_role;

GRANT SELECT ON REDEMPTIONS TO promo\_report\_role;

GRANT SELECT ON MEMBERSHIP\_HISTORY TO promo\_report\_role;

GRANT SELECT ON POINT\_TRANSACTIONS TO promo\_report\_role;

PROMOTION DBMS WITH:

Personal Script 1, G001– Loyalty Program: Membership Tier and Points Benefits Management

# Loyalty Program is important as the points are used in the following PROMOTION TYPES.

Personal Script 2, G002 – Voucher Promotion Management

Personal Script 3, G003 – Set Meal Promotion Management

Personal Script 4, G004 – Seasonal Item Promotion Management

# Individual SQL Script

## Chia Yue Sheng – Personal Script 4

*/\**

*GROUP NUMBER : G004*

*PROGRAMME : CS*

*STUDENT ID : 22ACB04673*

*STUDENT NAME : CHIA YUE SHENG*

*Submission date and time (DD-MON-YY): 29 April 2025*

*\*/*

*-- Every Error Test Case is checked and place commented.*

*-- Pre-Setup*

SET SERVEROUTPUT ON

SET LINESIZE 300;

SET PAGESIZE 100;

*--------------------------------------------------------------------------------*

*-- QUERY 1: SEASONAL ITEM PERFORMANCE*

*--------------------------------------------------------------------------------*

*-- User Transaction:*

*-- Display a list of all seasonal menu items and thier performance metrics.*

*-- Purpose:*

*-- Track sales and redemptions of seasonal items, to evaluate their popularity,*

*-- revenue contribution, and promotion effectiveness.*

*--------------------------------------------------------------------------------*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('##########################################################');

    DBMS\_OUTPUT.PUT\_LINE('######### QUERY 1: SEASONAL ITEM PERFORMANCE #############');

    DBMS\_OUTPUT.PUT\_LINE('##########################################################');

END;

/

SELECT

    mi.item\_id,

    mi.name,

    mi.description AS season,

    COUNT(oi.order\_item\_id) AS total\_sold,

    SUM(CASE WHEN o.discount\_amount = 0 THEN oi.price ELSE 0 END) AS revenue,

    COUNT(CASE WHEN r.redemption\_id IS NOT NULL THEN 1 END) AS points\_redemptions,

    ROUND(COUNT(oi.order\_item\_id) / GREATEST(1, mi.initial\_stock) \* 100, 2) AS sell\_through\_rate,

    ROUND(AVG(oi.quantity), 1) AS avg\_order\_size,

    ROUND(COUNT(CASE WHEN r.redemption\_id IS NOT NULL AND o.total\_amount >= p.min\_spend THEN 1 END) /

          GREATEST(1, COUNT(r.redemption\_id)) \* 100, 2) AS min\_spend\_met

FROM MENU\_ITEMS mi

JOIN ORDER\_ITEMS oi ON mi.item\_id = oi.item\_id

JOIN ORDERS o ON oi.order\_id = o.order\_id

LEFT JOIN REDEMPTIONS r ON o.order\_id = r.order\_id

LEFT JOIN PROMOTIONS p ON r.promotion\_id = p.promotion\_id AND p.set\_meal\_id = mi.item\_id

WHERE mi.item\_type = 'SEASONAL'

GROUP BY mi.item\_id, mi.name, mi.description, mi.initial\_stock;

*--------------------------------------------------------------------------------*

*-- DEMO TEST (NORMAL CASE): SEASONAL ITEM*

*--------------------------------------------------------------------------------*

*-- Expected Output:*

*-- A List of seasonal menu item with their respective sales performance data*

*-- Each row incliudes :*

*-- Item ID, Item Name, Season/Event, Total units sold, Total revenue. Total number of*

*-- points redemptions, Seel-through rate (%) based on initial stock, average quantity*

*-- per order, percentage of redemptions where minimun spend was achieved.*

*--------------------------------------------------------------------------------*

*-- Example Output:*

*-- | ITEM\_ID |         NAME          | SEASON                            | TOTAL\_SOLD | REVENUE | POINTS\_REDEMPTIONS | SELL\_THROUGH\_RATE | AVG\_ORDER\_SIZE | MIN\_SPEND\_MET |*

*---|---------|-----------------------|-----------------------------------|------------|---------|--------------------|-------------------|----------------|---------------|*

*-- |   10    |Pumpkin Spice Latte    | Seasonal autumn drink             |      1     |    0    |         1          |         .5        |       1        |       0       |*

*-- |    9    |Hello Kitty Happy Meal | Toy + Kids Meal (Limited Edition) |      1     |  12.99  |         0          |         .1        |       1        |       0       |*

*--------------------------------------------------------------------------------------------------------------------------------------------------------------------------*

*--------------------------------------------------------------------------------*

*-- QUERY 2: SEASONAL INVENTORY & REDEMPTION STATUS MONITORING*

*--------------------------------------------------------------------------------*

*-- User Transaction:*

*-- Displays real-time inventory and point redemption eligibility status for all seasonal menu items,*

*-- providing managers with actionable insights for inventory and promotion management.*

*-- Purpose:*

*-- Help monitor stock availability and check if seasonal items are eligible*

*-- for loyalty redemptions during their active period.*

*-- To identify items needing restocking or promotion adjustments, and support decision-making for seasonal menu planning*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('######################################################################');

    DBMS\_OUTPUT.PUT\_LINE('##### QUERY 2: SEASONAL INVENTORY & REDEMPTION STATUS MONITORING #####');

    DBMS\_OUTPUT.PUT\_LINE('######################################################################');

END;

/

SELECT

    mi.item\_id,

    mi.name,

    mi.current\_stock,

    CEIL(mi.valid\_to - SYSDATE) AS days\_left\_in\_season,

    CASE WHEN p.promotion\_id IS NOT NULL THEN 'Y' ELSE 'N' END AS redemption\_allowed,

    NVL(p.min\_spend, 0) AS min\_spend\_required,

    NVL(p.points\_required, 0) AS points\_cost

FROM MENU\_ITEMS mi

LEFT JOIN PROMOTIONS p ON mi.item\_id = p.set\_meal\_id AND SYSDATE BETWEEN p.valid\_from AND p.valid\_to

WHERE mi.item\_type = 'SEASONAL'

ORDER BY mi.valid\_to;

*--------------------------------------------------------------------------------*

*-- DEMO TEST (NORMAL CASE): SEASONAL ITEM*

*--------------------------------------------------------------------------------*

*-- Expected Output:*

*-- Inventory status with days remaining and redemption rules.*

*-- Including: Item ID and name, current stock quantity, days remaining in season, redemtion eligibility,*

*-- minimun spend required, points needs for redemtion.*

*-- Example Output:*

*-- | ITEM\_ID |        NAME             | CURRENT\_STOCK | DAYS\_LEFT\_IN\_SEASON | R | MIN\_SPEND\_REQUIRED | POINTS\_COST |*

*---|---------|-------------------------|---------------|---------------------|---|--------------------|-------------|*

*-- | 12      | Summer BBQ Burger       |     800       |       126           | N |         0          |      0      |*

*-- | 11      | Festival Mooncake Set   |     500       |       156           | N |         0          |      0      |*

*-- | 10      | Pumpkin Spice Latte     |     200       |       202           | N |         0          |      0      |*

*-- | 9       | Hello Kitty Happy Meal  |     1000      |       217           | N |         0          |      0      |*

*-- | 13      | Winter Hot Chocolate    |     300       |       307           | N |         0          |      0      |*

*---------------------------------------------------------------------------------------------------------------------*

*--------------------------------------------------------------------------------*

*-- FUNCTION 1: Check Redemption Eligibility*

*--------------------------------------------------------------------------------*

*-- User Transaction:*

*-- Check if a customer qualifies to redeem a seasonal menu item using loyalty points,*

*-- and provides a immediate feedback on eligibility status with specific rejection reasons.*

*-- Purpose:*

*-- Verify if a member can redeem a seasonal item with points.*

*-- Input Arguments:*

*-- p\_customer\_id   NUMBER   the ID of the customer to evaluate*

*-- p\_setmeal\_id    NUMBER   the ID of the seasonal menu item*

*-- p\_setmeal\_price NUMBER   current price of item in the order*

*-- Return:*

*-- ELIGIBLE or error message (e.g., INSUFFICIENT\_POINTS, MIN\_SPEND\_NOT\_MET).*

*--------------------------------------------------------------------------------*

*-----------------------------------------*

*-- Enhanced FN\_CHECK\_SEASONAL\_REDEMPTION*

*-----------------------------------------*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('############################################################');

    DBMS\_OUTPUT.PUT\_LINE('######### FUNCTION 1: CHECK REDEMPTION ELIGIBILITY #########');

    DBMS\_OUTPUT.PUT\_LINE('############################################################');

END;

/

CREATE OR REPLACE FUNCTION FN\_CHECK\_SEASONAL\_REDEMPTION(

    p\_customer\_id IN NUMBER,

    p\_setmeal\_id IN NUMBER,

    p\_setmeal\_price IN NUMBER

) RETURN VARCHAR2 AS

    v\_promo\_rec PROMOTIONS%ROWTYPE;

    v\_points\_balance NUMBER;

    v\_has\_voucher NUMBER;

BEGIN

*-- Check if seasonal promotion exists*

    BEGIN

        SELECT p.\*

        INTO v\_promo\_rec

        FROM PROMOTIONS p

        WHERE p.set\_meal\_id = p\_setmeal\_id

        AND p.promotion\_type = 'SEASONAL'

        AND p.valid\_from <= SYSDATE

        AND p.valid\_to >= SYSDATE

        AND ROWNUM = 1;

    EXCEPTION

        WHEN NO\_DATA\_FOUND THEN

            RETURN 'NO\_PROMOTION';

    END;

*-- Check points balance*

    BEGIN

        SELECT points\_balance

        INTO v\_points\_balance

        FROM CUSTOMERS

        WHERE customer\_id = p\_customer\_id;

    EXCEPTION

        WHEN NO\_DATA\_FOUND THEN

            RETURN 'INVALID CUSTOMER';

    END;

    IF v\_points\_balance < NVL(v\_promo\_rec.points\_required, 0) THEN

        RETURN 'INSUFFICIENT\_POINTS';

    END IF;

*-- Check min spend*

    IF v\_promo\_rec.min\_spend IS NOT NULL AND p\_setmeal\_price < v\_promo\_rec.min\_spend THEN

        RETURN 'MIN\_SPEND\_NOT\_MET';

    END IF;

*-- Check tier restriction*

    IF v\_promo\_rec.applicable\_tier\_id IS NOT NULL THEN

        DECLARE

            v\_customer\_tier NUMBER;

        BEGIN

            SELECT tier\_id

            INTO v\_customer\_tier

            FROM CUSTOMERS

            WHERE customer\_id = p\_customer\_id;

            IF v\_customer\_tier < v\_promo\_rec.applicable\_tier\_id THEN

                RETURN 'TIER\_RESTRICTED';

            END IF;

        END;

    END IF;

    RETURN 'ELIGIBLE';

EXCEPTION

    WHEN OTHERS THEN

        RETURN 'CHECK\_ERROR';

END;

/

*-- Setup test data*

INSERT INTO PROMOTIONS (promotion\_type, name, valid\_from, valid\_to, discount\_value, voucher\_code, is\_auto\_assign, points\_required, min\_spend, set\_meal\_id, applicable\_tier\_id)

VALUES ('SEASONAL', 'Mooncake Set Redemtion', SYSDATE, SYSDATE+30, NULL, NULL, 'Y', 400, 15, 11, 2);

UPDATE CUSTOMERS SET points\_balance = 300 WHERE customer\_id = 1001;

UPDATE CUSTOMERS SET points\_balance = 750 WHERE customer\_id = 1002;

UPDATE CUSTOMERS SET points\_balance = 1000 WHERE customer\_id = 1003;

UPDATE CUSTOMERS SET points\_balance = 500 WHERE customer\_id = 1006;

*--------------------------------------------------------------------------------*

*-- DEMO TEST 1 (ELIGIBLE CASE): FN\_CHECK\_SEASONAL\_REDEMPTION*

*--------------------------------------------------------------------------------*

*-- Testing with eligible conditions: sufficient points, minimum spend met, and minimum tier achieved.*

*-- Expected Output:*

*-- Status with : ELIGIBLE*

*--------------------------------------------------------------------------------*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('===== TESTING REDEMPTION ELIGIBILITY (ELIGIBLE CONDITION) =====');

    DBMS\_OUTPUT.PUT\_LINE('Testing Status: ' || FN\_CHECK\_SEASONAL\_REDEMPTION(1003, 11, 18.5) || ' to this seasonal promotion');

    DBMS\_OUTPUT.PUT\_LINE('===============================================================');

END;

/

*--------------------------------------------------------------------------------*

*-- DEMO TEST 2 (INELIGIBLE CASE): FN\_CHECK\_SEASONAL\_REDEMPTION*

*--------------------------------------------------------------------------------*

*-- Testing with ineligible conditions: No Seasonal Promotion*

*-- Expected Output:*

*-- Status like : NO\_PROMOTION, INSUFFICIENT\_POINTS, MIN\_SPEND\_NOT\_MET, TIER\_RESTRICTED, INVALID\_CUSTOMER*

*--------------------------------------------------------------------------------*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('============ TESTING WITH A NON-EXISTENT PROMOTION ============');

    DBMS\_OUTPUT.PUT\_LINE('Testing Status: ' || FN\_CHECK\_SEASONAL\_REDEMPTION(1003, 12, 18.5) || ' to this seasonal promotion');

    DBMS\_OUTPUT.PUT\_LINE('===============================================================');

    DBMS\_OUTPUT.PUT\_LINE('-');

    DBMS\_OUTPUT.PUT\_LINE('============ TESTING WITH INSUFFICIENT CUSTOMER POINT =========');

    DBMS\_OUTPUT.PUT\_LINE('Testing Status: ' || FN\_CHECK\_SEASONAL\_REDEMPTION(1001, 11, 18.5) || ' to this seasonal promotion');

    DBMS\_OUTPUT.PUT\_LINE('===============================================================');

    DBMS\_OUTPUT.PUT\_LINE('-');

    DBMS\_OUTPUT.PUT\_LINE('============ TESTING WITH INSUFFICIENT MINIMUN SPEND ==========');

    DBMS\_OUTPUT.PUT\_LINE('Testing Status: ' || FN\_CHECK\_SEASONAL\_REDEMPTION(1003, 11, 12) || ' to this seasonal promotion');

    DBMS\_OUTPUT.PUT\_LINE('===============================================================');

    DBMS\_OUTPUT.PUT\_LINE('-');

    DBMS\_OUTPUT.PUT\_LINE('============ TESTING WITH AN INELIGIBLE TIER ==================');

    DBMS\_OUTPUT.PUT\_LINE('Testing Status: ' || FN\_CHECK\_SEASONAL\_REDEMPTION(1006, 11, 18.5) || ' to this seasonal promotion');

    DBMS\_OUTPUT.PUT\_LINE('===============================================================');

    DBMS\_OUTPUT.PUT\_LINE('-');

    DBMS\_OUTPUT.PUT\_LINE('============ TESTING WITH AN NON-EXISTENT CUSTOMER ============');

    DBMS\_OUTPUT.PUT\_LINE('Testing Status: ' || FN\_CHECK\_SEASONAL\_REDEMPTION(1010, 11, 18.5) || ' to this seasonal promotion');

    DBMS\_OUTPUT.PUT\_LINE('===============================================================');

    DBMS\_OUTPUT.PUT\_LINE('-');

END;

/

ALTER TRIGGER TRG\_SYNC\_POINT\_BALANCE DISABLE;

DELETE FROM PROMOTIONS

WHERE promotion\_type = 'SEASONAL'

  AND name = 'Mooncake Set Redemtion'

  AND set\_meal\_id = 11

  AND applicable\_tier\_id = 2;

ALTER TRIGGER TRG\_SYNC\_POINT\_BALANCE ENABLE;

*--------------------------------------------------------------------------------*

*-- FUNCTION 2: Calculate Seasonal Demand*

*--------------------------------------------------------------------------------*

*-- User Transaction:*

*-- Forecast demand for seasonal items based on historical sales.*

*-- eg.Forecasts how many seasonal items will sell before expiry.Based on past 30-day sales trends.*

*-- Purpose:*

*-- Predict demand for seasonal items based on past trends.*

*-- Assist with inventory planning for seasonal offerings.*

*-- Input Arguments:*

*-- p\_item\_id            NUMBER    ID of the seasonal menu item (from MENU\_ITEMS table)*

*-- p\_days\_remaining     NUMBER    Days remaining until seasonal item expires (valid\_to date)*

*-- Return:*

*-- NUMBER - Predicted quantity needed to meet demand (including 20% buffer)*

*--------------------------------------------------------------------------------*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('############################################################');

    DBMS\_OUTPUT.PUT\_LINE('########### FUNCTION 2: CALCULATE SEASONAL DEMAND ##########');

    DBMS\_OUTPUT.PUT\_LINE('############################################################');

END;

/

CREATE OR REPLACE FUNCTION FN\_PREDICT\_SEASONAL\_DEMAND(

    p\_item\_id IN NUMBER,

    p\_days\_remaining IN NUMBER

) RETURN NUMBER AS

    v\_avg\_daily\_sales NUMBER;

BEGIN

*-- Get historical daily sales rate*

    SELECT AVG(oi.quantity)

    INTO v\_avg\_daily\_sales

    FROM ORDER\_ITEMS oi

    JOIN ORDERS o ON oi.order\_id = o.order\_id

    WHERE oi.item\_id = p\_item\_id

    AND o.order\_date BETWEEN SYSDATE-30 AND SYSDATE;

    IF v\_avg\_daily\_sales IS NULL THEN

        RETURN 0; *-- item no found*

    END IF;

*-- Adjust for remaining season days*

    RETURN CEIL(v\_avg\_daily\_sales \* p\_days\_remaining \* 1.2); *-- +20% buffer*

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        RETURN 0; *-- New item? Default to 0*

END;

/

*--------------------------------------------------------------------------------*

*-- DEMO TEST 1 (NORMAL CASE): FN\_PREDICT\_SEASONAL\_DEMAND*

*--------------------------------------------------------------------------------*

*-- Testing with existing item*

*-- Expected Output: Number of predicted demand for item*

*--------------------------------------------------------------------------------*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('====== TESTING DEMAND PREDICTION (NORMAL CONDITION) ======');

    DBMS\_OUTPUT.PUT\_LINE('Predicted demand for item: ' || FN\_PREDICT\_SEASONAL\_DEMAND(9,15) || ' units');

    DBMS\_OUTPUT.PUT\_LINE('==========================================================');

END;

/

*/\**

*--------------------------------------------------------------------------------*

*-- DEMO TEST 2 (ITEM NOT FOUND): FN\_PREDICT\_SEASONAL\_DEMAND*

*--------------------------------------------------------------------------------*

*-- Testing with no existing item*

*-- Expected Output: 0 (Due to no existing item)*

*--------------------------------------------------------------------------------*

*BEGIN*

*DBMS\_OUTPUT.PUT\_LINE('====== TESTING DEMAND PREDICTION (ITEM NOT FOUND) ========');*

*DBMS\_OUTPUT.PUT\_LINE('Predicted demand for item: ' || FN\_PREDICT\_SEASONAL\_DEMAND(22,15) || ' units');*

*DBMS\_OUTPUT.PUT\_LINE('==========================================================');*

*END;*

*/ \*/*

*--------------------------------------------------------------------------------*

*-- PROCEDURE 1: REDEEM SEASONAL ITEM WITH POINTS*

*--------------------------------------------------------------------------------*

*-- User Transaction:*

*-- Process the redemption of a seasonal item using customer points.*

*-- Purpose:*

*-- Process point redemption for seasonal items.*

*-- Handle the complete seasonal redemption workflow including point deduction.*

*-- Input Arguments:*

*-- p\_customer\_id  VARCHAR2   ID of the redeeming customer*

*-- p\_setmeal\_id   NUMBER     ID of the seasonal menu item to redeem*

*-- p\_order\_id     NUMBER     ID of the order containing the item*

*--------------------------------------------------------------------------------*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('############################################################');

    DBMS\_OUTPUT.PUT\_LINE('####### PROCEDURE 1: REDEEM SEASONAL ITEM WITH POINTS ######');

    DBMS\_OUTPUT.PUT\_LINE('############################################################');

END;

/

CREATE OR REPLACE PROCEDURE PROC\_REDEEM\_SEASONAL\_ITEM(

    p\_customer\_id IN NUMBER,

    p\_setmeal\_id IN NUMBER,

    p\_order\_id IN NUMBER

) AS

    v\_promo\_id NUMBER;

    v\_points\_required NUMBER;

    v\_points\_balance NUMBER;

    v\_order\_item\_exists NUMBER;

    v\_base\_price NUMBER;

    v\_promo\_exists NUMBER := 0;

    v\_customer\_exists NUMBER := 0;

    v\_item\_exists NUMBER := 0;

    v\_order\_exists NUMBER := 0;

BEGIN

*-- 1) Validate customer exists (for FK\_REDEMPTION\_CUSTOMER)*

    SELECT COUNT(\*) INTO v\_customer\_exists

    FROM CUSTOMERS

    WHERE customer\_id = p\_customer\_id;

    IF v\_customer\_exists = 0 THEN

*-- No LOG\_ACTIVITY here because customer doesn't exist*

        RAISE\_APPLICATION\_ERROR(-20053, 'Invalid customer ID');

    END IF;

*-- Now it is safe to log because customer exists*

    LOG\_ACTIVITY(

        p\_customer\_id,

        'SEASONAL\_REDEMPTION',

        'Customer ' || p\_customer\_id || ' validated for seasonal item redemption',

        SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

        );

*-- 2) Validate menu item exists (for FK\_REDEMPTION\_ITEM)*

    SELECT COUNT(\*) INTO v\_item\_exists

    FROM MENU\_ITEMS

    WHERE item\_id = p\_setmeal\_id

    AND item\_type = 'SEASONAL';

    IF v\_item\_exists = 0 THEN

        LOG\_ACTIVITY(

            p\_customer\_id,

            'SEASONAL\_REDEMPTION',

            'Failed redemption attempt - Invalid seasonal menu item ID: ' || p\_setmeal\_id,

            SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

        );

        RAISE\_APPLICATION\_ERROR(-20054, 'Invalid seasonal menu item ID');

    END IF;

*-- Log item validation*

    LOG\_ACTIVITY(

        p\_customer\_id,

        'SEASONAL\_REDEMPTION',

        'Seasonal menu item ' || p\_setmeal\_id || ' validated for customer ' || p\_customer\_id,

        SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

    );

*-- 3) Validate order exists (for FK\_REDEMPTION\_ORDER)*

    SELECT COUNT(\*) INTO v\_order\_exists

    FROM ORDERS

    WHERE order\_id = p\_order\_id

    AND customer\_id = p\_customer\_id;

    IF v\_order\_exists = 0 THEN

        LOG\_ACTIVITY(

            p\_customer\_id,

            'SEASONAL\_REDEMPTION',

            'Failed redemption attempt - Invalid order ID for customer: ' || p\_order\_id,

            SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

        );

        RAISE\_APPLICATION\_ERROR(-20055, 'Invalid order ID for customer');

    END IF;

*-- Log order validation*

    LOG\_ACTIVITY(

        p\_customer\_id,

        'SEASONAL\_REDEMPTION',

        'Order ' || p\_order\_id || ' validated for seasonal item redemption',

        SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

    );

*-- 4) Verify the order contains the setmeal*

    SELECT COUNT(\*) INTO v\_order\_item\_exists

    FROM ORDER\_ITEMS

    WHERE order\_id = p\_order\_id

    AND item\_id = p\_setmeal\_id;

    IF v\_order\_item\_exists = 0 THEN

        LOG\_ACTIVITY(

            p\_customer\_id,

            'SEASONAL\_REDEMPTION',

            'Failed redemption attempt - Order does not contain specified setmeal: ' || p\_setmeal\_id,

            SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

        );

        RAISE\_APPLICATION\_ERROR(-20051, 'Order does not contain the specified setmeal');

    END IF;

*-- Log order item validation*

    LOG\_ACTIVITY(

        p\_customer\_id,

        'SEASONAL\_REDEMPTION',

        'Setmeal ' || p\_setmeal\_id || ' confirmed in order ' || p\_order\_id,

        SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

    );

*-- 5) Check if promotion exists (for FK\_REDEMPTION\_PROMOTION)*

    SELECT COUNT(\*) INTO v\_promo\_exists

    FROM PROMOTIONS

    WHERE set\_meal\_id = p\_setmeal\_id

    AND promotion\_type = 'SEASONAL'

    AND SYSDATE BETWEEN valid\_from AND valid\_to;

    IF v\_promo\_exists = 0 THEN

        LOG\_ACTIVITY(

            p\_customer\_id,

            'SEASONAL\_REDEMPTION',

            'Failed redemption attempt - No valid seasonal promotion found for setmeal: ' || p\_setmeal\_id,

            SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

        );

        RAISE\_APPLICATION\_ERROR(-20050, 'No valid seasonal promotion found for this setmeal');

    END IF;

*-- Log promotion check*

    LOG\_ACTIVITY(

        p\_customer\_id,

        'SEASONAL\_REDEMPTION',

        'Valid seasonal promotion found for setmeal ' || p\_setmeal\_id,

        SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

    );

*-- 6) Find applicable seasonal promotion*

    SELECT promotion\_id, points\_required

    INTO v\_promo\_id, v\_points\_required

    FROM PROMOTIONS

    WHERE set\_meal\_id = p\_setmeal\_id

    AND promotion\_type = 'SEASONAL'

    AND SYSDATE BETWEEN valid\_from AND valid\_to

    AND ROWNUM = 1;

*-- 7) Check customers available points (for points\_used >= 0 constraint)*

    SELECT points\_balance INTO v\_points\_balance

    FROM CUSTOMERS

    WHERE customer\_id = p\_customer\_id;

    IF v\_points\_balance < v\_points\_required THEN

        LOG\_ACTIVITY(

            p\_customer\_id,

            'SEASONAL\_REDEMPTION',

            'Failed redemption attempt - Insufficient points (Balance: ' || v\_points\_balance || ', Required: ' || v\_points\_required || ')',

            SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

        );

        RAISE\_APPLICATION\_ERROR(-20052, 'Insufficient points balance');

    END IF;

*-- Log points check*

    LOG\_ACTIVITY(

        p\_customer\_id,

        'SEASONAL\_REDEMPTION',

        'Customer ' || p\_customer\_id || ' has sufficient points for redemption (Points balance: ' || v\_points\_balance || ')',

        SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

    );

*-- 8) Get base price for the setmeal*

    SELECT base\_price INTO v\_base\_price

    FROM MENU\_ITEMS

    WHERE item\_id = p\_setmeal\_id;

*-- Temporarily disable problematic triggers*

    EXECUTE IMMEDIATE 'ALTER TRIGGER TRG\_AUTO\_ASSIGN\_VOUCHERS DISABLE';

    EXECUTE IMMEDIATE 'ALTER TRIGGER TRG\_SYNC\_POINT\_BALANCE DISABLE';

*-- 9) Record the redemption (meets all constraints)*

    INSERT INTO REDEMPTIONS (

        customer\_id,

        promotion\_id,

        order\_id,

        points\_used,

        redemption\_date,

        redemption\_status,

        item\_id,

        discount\_value

    ) VALUES (

        p\_customer\_id,

        v\_promo\_id,

        p\_order\_id,

        v\_points\_required,

        SYSTIMESTAMP,

        'COMPLETED',

        p\_setmeal\_id,

        v\_base\_price

    );

*-- 10) Deduct points*

    INSERT INTO POINT\_TRANSACTIONS (

        customer\_id,

        order\_id,

        promotion\_id,

        points\_amount,

        transaction\_type,

        description,

        transaction\_date,

        expiry\_date

    ) VALUES (

        p\_customer\_id,

        p\_order\_id,

        v\_promo\_id,

        -v\_points\_required,

        'REDEMPTION',

        'Seasonal item redemption for setmeal ' || p\_setmeal\_id,

        SYSDATE,

        ADD\_MONTHS(SYSDATE, 12)

    );

*-- 11) Apply discount to the order item*

    UPDATE ORDER\_ITEMS

    SET discount\_applied = v\_base\_price

    WHERE order\_id = p\_order\_id

    AND item\_id = p\_setmeal\_id;

*-- Re-enable triggers*

    EXECUTE IMMEDIATE 'ALTER TRIGGER TRG\_AUTO\_ASSIGN\_VOUCHERS ENABLE';

    EXECUTE IMMEDIATE 'ALTER TRIGGER TRG\_SYNC\_POINT\_BALANCE ENABLE';

    COMMIT;

*-- 12) Log success*

    LOG\_ACTIVITY(

        p\_customer\_id,

        'SEASONAL\_REDEMPTION',

        'Successfully redeemed seasonal item ' || p\_setmeal\_id || ' using ' || v\_points\_required || ' points',

        SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

    );

    DBMS\_OUTPUT.PUT\_LINE('Successfully redeemed seasonal item ' || p\_setmeal\_id || ' using ' || v\_points\_required || ' points');

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        ROLLBACK;

*-- Ensure re-enabling of triggers*

        EXECUTE IMMEDIATE 'ALTER TRIGGER TRG\_AUTO\_ASSIGN\_VOUCHERS ENABLE';

        EXECUTE IMMEDIATE 'ALTER TRIGGER TRG\_SYNC\_POINT\_BALANCE ENABLE';

*-- Only log if customer exists*

        BEGIN

            SELECT 1 INTO v\_customer\_exists

            FROM CUSTOMERS

            WHERE customer\_id = p\_customer\_id;

*-- If found, safe to log*

            LOG\_ACTIVITY(

                p\_customer\_id,

                'SEASONAL\_REDEMPTION',

                'Failed redemption attempt - No valid seasonal promotion found for setmeal: ' || p\_setmeal\_id,

                SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

            );

        EXCEPTION

            WHEN NO\_DATA\_FOUND THEN

                NULL; *-- Do not log if customer not found*

        END;

        RAISE\_APPLICATION\_ERROR(-20050, 'No valid seasonal promotion found for this setmeal');

    WHEN OTHERS THEN

        ROLLBACK;

        EXECUTE IMMEDIATE 'ALTER TRIGGER TRG\_AUTO\_ASSIGN\_VOUCHERS ENABLE';

        EXECUTE IMMEDIATE 'ALTER TRIGGER TRG\_SYNC\_POINT\_BALANCE ENABLE';

*-- Only log if customer exists*

        BEGIN

            SELECT 1 INTO v\_customer\_exists

            FROM CUSTOMERS

            WHERE customer\_id = p\_customer\_id;

*-- If found, safe to log*

            LOG\_ACTIVITY(

                p\_customer\_id,

                'SEASONAL\_REDEMPTION',

                'Error redeeming seasonal item: ' || SQLERRM,

                SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

            );

        EXCEPTION

            WHEN NO\_DATA\_FOUND THEN

                NULL; *-- Do not log if customer not found*

        END;

        DBMS\_OUTPUT.PUT\_LINE('Error redeeming seasonal item: ' || SQLERRM);

        RAISE;

END;

/

*-- Set up test environment*

UPDATE CUSTOMERS SET points\_balance = 600 WHERE customer\_id = 1001;

*--------------------------------------------------------------------------------*

*-- DEMO TEST - EXECUTE PROC\_REDEEM\_SEASONAL\_ITEM*

*--------------------------------------------------------------------------------*

*-- Create test order specifically for these tests*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('Adding a record for testing...');

END;

/

INSERT INTO PROMOTIONS (

    promotion\_type, name, valid\_from, valid\_to,

    points\_required, min\_spend, set\_meal\_id

) VALUES (

    'SEASONAL', 'TEST Hello Kitty Promotion',

    SYSDATE-1, SYSDATE+30, 500, 20, 9

);

DECLARE

    v\_test\_order\_id NUMBER;

BEGIN

    INSERT INTO ORDERS (customer\_id, total\_amount, discount\_amount, final\_amount)

    VALUES (1001, 25, 0, 25)

    RETURNING order\_id INTO v\_test\_order\_id;

*-- Add seasonal item to order*

    INSERT INTO ORDER\_ITEMS (order\_id, item\_id, quantity, price)

    VALUES (v\_test\_order\_id, 9, 1, 12.99);

*-- 1. Test successful redemption*

    DBMS\_OUTPUT.PUT\_LINE('====== TESTING SUCCESSFUL REDEMPTION ======');

    BEGIN

        PROC\_REDEEM\_SEASONAL\_ITEM(

            p\_customer\_id => 1001,

            p\_setmeal\_id => 9,

            p\_order\_id => v\_test\_order\_id

        );

        DBMS\_OUTPUT.PUT\_LINE('Success - redemption worked as expected');

        DBMS\_OUTPUT.PUT\_LINE('-');

    EXCEPTION

        WHEN OTHERS THEN

            DBMS\_OUTPUT.PUT\_LINE('UNEXPECTED ERROR: ' || SQLERRM);

    END;

*-- 2. Test invalid customer ID (modified to avoid LOG\_ACTIVITY constraint error)*

    DBMS\_OUTPUT.PUT\_LINE('====== TESTING INVALID CUSTOMER ID ======');

    BEGIN

        PROC\_REDEEM\_SEASONAL\_ITEM(

            p\_customer\_id => 999999,

            p\_setmeal\_id => 9,

            p\_order\_id => v\_test\_order\_id

        );

    EXCEPTION

        WHEN OTHERS THEN

            IF SQLCODE = -20053 THEN

                DBMS\_OUTPUT.PUT\_LINE('Success - got expected error: ' || SQLERRM);

                DBMS\_OUTPUT.PUT\_LINE('-');

            ELSE

                DBMS\_OUTPUT.PUT\_LINE('UNEXPECTED ERROR: ' || SQLERRM);

            END IF;

    END;

*-- 3. Test invalid seasonal item*

    DBMS\_OUTPUT.PUT\_LINE('====== TESTING INVALID SEASONAL ITEM ======');

    BEGIN

        PROC\_REDEEM\_SEASONAL\_ITEM(

            p\_customer\_id => 1001,

            p\_setmeal\_id => 999,

            p\_order\_id => v\_test\_order\_id

        );

    EXCEPTION

        WHEN OTHERS THEN

            IF SQLCODE = -20054 THEN

                DBMS\_OUTPUT.PUT\_LINE('Success - got expected error: ' || SQLERRM);

                DBMS\_OUTPUT.PUT\_LINE('-');

            ELSE

                DBMS\_OUTPUT.PUT\_LINE('UNEXPECTED ERROR: ' || SQLERRM);

            END IF;

    END;

*-- 4. Test order ownership (using a different order)*

    DBMS\_OUTPUT.PUT\_LINE('====== TESTING ORDER OWNERSHIP ======');

    DECLARE

        v\_other\_order NUMBER;

    BEGIN

*-- Create an order that doesn't belong to our customer*

        INSERT INTO ORDERS (customer\_id, total\_amount, discount\_amount, final\_amount)

        VALUES (1002, 25, 0, 25)

        RETURNING order\_id INTO v\_other\_order;

        BEGIN

            PROC\_REDEEM\_SEASONAL\_ITEM(

                p\_customer\_id => 1001,

                p\_setmeal\_id => 9,

                p\_order\_id => v\_other\_order

            );

        EXCEPTION

            WHEN OTHERS THEN

                IF SQLCODE = -20055 THEN

                    DBMS\_OUTPUT.PUT\_LINE('Success - got expected error: ' || SQLERRM);

                    DBMS\_OUTPUT.PUT\_LINE('-');

                ELSE

                    DBMS\_OUTPUT.PUT\_LINE('UNEXPECTED ERROR: ' || SQLERRM);

                END IF;

        END;

    END;

*-- 5. Test missing order item*

    DBMS\_OUTPUT.PUT\_LINE('====== TESTING MISSING ORDER ITEM ======');

    DECLARE

        v\_empty\_order NUMBER;

    BEGIN

*-- Create an order without the seasonal item*

        INSERT INTO ORDERS (customer\_id, total\_amount, discount\_amount, final\_amount)

        VALUES (1001, 25, 0, 25)

        RETURNING order\_id INTO v\_empty\_order;

        BEGIN

            PROC\_REDEEM\_SEASONAL\_ITEM(

                p\_customer\_id => 1001,

                p\_setmeal\_id => 9,

                p\_order\_id => v\_empty\_order

            );

        EXCEPTION

            WHEN OTHERS THEN

                IF SQLCODE = -20051 THEN

                    DBMS\_OUTPUT.PUT\_LINE('Success - got expected error: ' || SQLERRM);

                    DBMS\_OUTPUT.PUT\_LINE('-');

                ELSE

                    DBMS\_OUTPUT.PUT\_LINE('UNEXPECTED ERROR: ' || SQLERRM);

                END IF;

        END;

    END;

*-- 6. Test no valid promotion (fixed date range issue)*

    DBMS\_OUTPUT.PUT\_LINE('====== TESTING NO VALID PROMOTION ======');

    BEGIN

*-- Make promotion dates invalid but maintain valid\_from < valid\_to*

        UPDATE PROMOTIONS

        SET valid\_from = SYSDATE+1, valid\_to = SYSDATE+2

        WHERE set\_meal\_id = 9;

        BEGIN

            PROC\_REDEEM\_SEASONAL\_ITEM(

                p\_customer\_id => 1001,

                p\_setmeal\_id => 9,

                p\_order\_id => v\_test\_order\_id

            );

        EXCEPTION

            WHEN OTHERS THEN

                IF SQLCODE = -20050 THEN

                    DBMS\_OUTPUT.PUT\_LINE('Success - got expected error: ' || SQLERRM);

                    DBMS\_OUTPUT.PUT\_LINE('-');

                ELSE

                    DBMS\_OUTPUT.PUT\_LINE('UNEXPECTED ERROR: ' || SQLERRM);

                END IF;

        END;

*-- Restore promotion*

        UPDATE PROMOTIONS

        SET valid\_from = SYSDATE-1, valid\_to = SYSDATE+30

        WHERE set\_meal\_id = 9;

    END;

*-- 7. Test insufficient points*

    DBMS\_OUTPUT.PUT\_LINE('====== TESTING INSUFFICIENT POINTS ======');

    BEGIN

*-- Set points to 0*

        UPDATE CUSTOMERS SET points\_balance = 0 WHERE customer\_id = 1001;

        BEGIN

            PROC\_REDEEM\_SEASONAL\_ITEM(

                p\_customer\_id => 1001,

                p\_setmeal\_id => 9,

                p\_order\_id => v\_test\_order\_id

            );

        EXCEPTION

            WHEN OTHERS THEN

                IF SQLCODE = -20052 THEN

                    DBMS\_OUTPUT.PUT\_LINE('Success - got expected error: ' || SQLERRM);

                    DBMS\_OUTPUT.PUT\_LINE('-');

                ELSE

                    DBMS\_OUTPUT.PUT\_LINE('UNEXPECTED ERROR: ' || SQLERRM);

                END IF;

        END;

*-- Restore points*

        UPDATE CUSTOMERS SET points\_balance = 600 WHERE customer\_id = 1001;

    END;

*-- 8. Test non-seasonal item*

    DBMS\_OUTPUT.PUT\_LINE('====== TESTING NON-SEASONAL ITEM ======');

    DECLARE

        v\_regular\_item NUMBER;

    BEGIN

        SELECT item\_id INTO v\_regular\_item

        FROM MENU\_ITEMS

        WHERE item\_type != 'SEASONAL' AND ROWNUM = 1;

        BEGIN

            PROC\_REDEEM\_SEASONAL\_ITEM(

                p\_customer\_id => 1001,

                p\_setmeal\_id => v\_regular\_item,

                p\_order\_id => v\_test\_order\_id

            );

        EXCEPTION

            WHEN OTHERS THEN

                IF SQLCODE = -20054 THEN

                    DBMS\_OUTPUT.PUT\_LINE('Success - got expected error: ' || SQLERRM);

                    DBMS\_OUTPUT.PUT\_LINE('-');

                ELSE

                    DBMS\_OUTPUT.PUT\_LINE('UNEXPECTED ERROR: ' || SQLERRM);

                END IF;

        END;

    END;

END;

/

*--VERIFICATION*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('================ Verifty - Point Transaction ==============');

    DBMS\_OUTPUT.PUT\_LINE('==========================================================');

END;

/

SELECT \*

FROM point\_transactions

WHERE customer\_id = 1001

AND transaction\_type = 'REDEMPTION'

ORDER BY transaction\_date DESC;

*-- PROCEDURE 2: AUTO-EXPIRE SEASONAL ITEMS*

*--------------------------------------------------------------------------------*

*-- User Transaction:*

*-- Automatically deactivate seasonal items when their season ends.*

*-- Deactivates expired seasonal items from the menu.Removes linked promotions.*

*-- Purpose:*

*-- Deactivate seasonal items when the season ends.*

*-- Maintain menu integrity by removing expired seasonal items.*

*-- Keeps the menu current (no outdated items).*

*-- Input Arguments:*

*-- none*

*--------------------------------------------------------------------------------*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('############################################################');

    DBMS\_OUTPUT.PUT\_LINE('########## PROCEDURE 2: AUTO-EXPIRE SEASONAL ITEMS #########');

    DBMS\_OUTPUT.PUT\_LINE('############################################################');

END;

/

CREATE OR REPLACE PROCEDURE PROC\_EXPIRE\_SEASONAL\_ITEMS AS

    v\_count NUMBER := 0;

BEGIN

*-- Temporarily disable trigger to prevent point balance sync issues*

    EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_sync\_point\_balance DISABLE';

*-- Deactivate expired seasonal items*

    UPDATE MENU\_ITEMS

    SET is\_active = 'N'

    WHERE item\_type = 'SEASONAL'

    AND valid\_to < SYSDATE

    AND is\_active = 'Y'

    RETURNING COUNT(\*) INTO v\_count;

*-- Log the deactivation of expired seasonal items*

    LOG\_ACTIVITY(

        NULL,  *-- NULL as it's a batch operation, no specific customer involved*

        'SEASONAL\_EXPIRATION',

        'Deactivated ' || v\_count || ' expired seasonal items',

        SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

    );

    DBMS\_OUTPUT.PUT\_LINE('Deactivated ' || v\_count || ' expired seasonal items');

*-- Remove associated promotions*

    DELETE FROM PROMOTIONS

    WHERE set\_meal\_id IN (

        SELECT item\_id FROM MENU\_ITEMS

        WHERE item\_type = 'SEASONAL'

        AND valid\_to < SYSDATE

    )

    RETURNING COUNT(\*) INTO v\_count;

*-- Log the removal of associated promotions*

    LOG\_ACTIVITY(

        NULL,  *-- NULL as it's a batch operation, no specific customer involved*

        'SEASONAL\_EXPIRATION',

        'Removed ' || v\_count || ' associated seasonal promotions',

        SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

    );

    DBMS\_OUTPUT.PUT\_LINE('Removed ' || v\_count || ' associated promotions');

*-- Re-enable trigger*

    EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_sync\_point\_balance ENABLE';

    COMMIT;

EXCEPTION

    WHEN OTHERS THEN

*-- Ensure trigger is re-enabled even if error occurs*

        BEGIN

            EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_sync\_point\_balance ENABLE';

        EXCEPTION

            WHEN OTHERS THEN NULL;

        END;

        ROLLBACK;

        DBMS\_OUTPUT.PUT\_LINE('Error in PROC\_EXPIRE\_SEASONAL\_ITEMS: ' || SQLERRM);

*-- Log failure*

        LOG\_ACTIVITY(

            NULL,  *-- NULL as it's a batch operation*

            'ERROR',

            'Error in PROC\_EXPIRE\_SEASONAL\_ITEMS: ' || SQLERRM,

            SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

        );

        RAISE;

END;

/

*--------------------------------------------------------------------------------*

*-- DEMO TEST (NORMAL CASE): PROC\_EXPIRE\_SEASONAL\_ITEMS*

*--------------------------------------------------------------------------------*

*-- Run the procedure to expire seasonal items and remove associated promotions*

*-- Expected Output:*

*-- Deactivated X expired seasonal items*

*-- Removed X associated promotions*

*--------------------------------------------------------------------------------*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('Adding a record for testing...');

END;

/

INSERT INTO MENU\_ITEMS (item\_type, name, base\_price, valid\_to, is\_active)

VALUES ('SEASONAL', 'Test Seasonal Item', 25.00, SYSDATE + 30, 'Y');

*-- Valid for 30 days*

COMMIT;

*-- Insert a new promotion related to the seasonal item*

INSERT INTO PROMOTIONS (promotion\_type, name, valid\_from, valid\_to, discount\_value, set\_meal\_id)

VALUES ('SET\_MEAL', 'Test Seasonal Promo', SYSDATE, SYSDATE + 30, 5, (SELECT item\_id FROM MENU\_ITEMS WHERE name = 'Test Seasonal Item' AND item\_type = 'SEASONAL'FETCH FIRST 1 ROW ONLY));

*-- 5% discount, valid for 30 days*

COMMIT;

*-- Verify the seasonal item in MENU\_ITEMS*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('============ Verify Seasonal Item in MENU\_ITEMS ==========');

    DBMS\_OUTPUT.PUT\_LINE('==========================================================');

END;

/

SELECT item\_type, name, base\_price, valid\_to, is\_active

FROM MENU\_ITEMS

WHERE name = 'Test Seasonal Item' AND item\_type = 'SEASONAL';

*-- Verify the promotion in PROMOTIONS related to the seasonal item*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('=== Verify the promotion in PROMOTIONS related to the seasonal item ===');

    DBMS\_OUTPUT.PUT\_LINE('=======================================================================');

END;

/

SELECT promotion\_type, name, valid\_from, valid\_to, discount\_value, set\_meal\_id

FROM PROMOTIONS

WHERE set\_meal\_id = (

    SELECT DISTINCT item\_id

    FROM MENU\_ITEMS

    WHERE name = 'Test Seasonal Item' AND item\_type = 'SEASONAL'

);

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('Updating the Item Valid Date for Testing...');

    DBMS\_OUTPUT.PUT\_LINE('Valid Date - 10 days ago, Expire Date - 1 days ago');

    DBMS\_OUTPUT.PUT\_LINE('');

END;

/

*-- If needed, also reset valid\_from before setting valid\_to to expire the item*

UPDATE MENU\_ITEMS

SET valid\_from = SYSDATE - 10, *-- Set the valid\_from date to 10 days ago (for example)*

    valid\_to = SYSDATE - 1  *-- Expire the item by setting valid\_to to yesterday*

WHERE item\_type = 'SEASONAL'

  AND name = 'Test Seasonal Item';

COMMIT;

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('===== TESTING AUTO-EXPIRE SEASONAL ITEMS (Normal Case)====');

    PROC\_EXPIRE\_SEASONAL\_ITEMS;

    DBMS\_OUTPUT.PUT\_LINE('==========================================================');

END;

/

*-- Check if the seasonal item has been deactivated*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('======= Showing Seasonal Item has been DEACTIVATED ======');

    DBMS\_OUTPUT.PUT\_LINE('==========================================================');

END;

/

SELECT item\_type, name, base\_price, valid\_to, is\_active

FROM MENU\_ITEMS WHERE name = 'Test Seasonal Item' AND item\_type = 'SEASONAL';

*-- Check if the associated promotion has been removed*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('===== Showing Associated Promotion has been REMOVED ======');

    DBMS\_OUTPUT.PUT\_LINE('==========================================================');

END;

/

SELECT promotion\_type, name, valid\_from, valid\_to, discount\_value, set\_meal\_id

FROM PROMOTIONS

WHERE set\_meal\_id = (

    SELECT DISTINCT item\_id

    FROM MENU\_ITEMS

    WHERE name = 'Test Seasonal Item' AND item\_type = 'SEASONAL'

);

*-- Verify the procedure is valid*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('================= Showing procedure is VALID =============');

    DBMS\_OUTPUT.PUT\_LINE('==========================================================');

END;

/

SELECT object\_name, status FROM user\_objects

WHERE object\_name = 'PROC\_REDEEM\_SEASONAL\_ITEM';

## Hii Zi Wei – Personal Script 1

*/\**

*GROUP NUMBER : G001*

*PROGRAMME    : CS*

*STUDENT ID   : 22ACB04587*

*STUDENT NAME : Hii Zi Wei*

*Submission date and time (DD-MON-YY): 29 April 2025*

*\*/*

*-- Every Error Test Case is checked and place commented.*

*-- Disabling the trigger is using in the bullk assignments of renewal and point expiration. The procedure are designed as automatic operations.*

*-- Hence, trigger is disable in this case as a bulk insert/update/delete affecting thousands of rows means thousands of trigger executions. A trigger might insert/update other rows that re-fire the same or another trigger.*

SET SERVEROUTPUT ON;

SET LINESIZE 200;

SET PAGESIZE 100;

*--------------------------------------------------------------------------------*

*-- PROCEDURE 1: PROC\_RENEW\_MEMBERSHIPS*

*--------------------------------------------------------------------------------*

*-- User Transaction:*

*-- Renew the memberships of customers whose expiry dates have passed,*

*-- are about to expire, or are due in the coming month, depending on the specified type.*

*-- This allows a staff member or automated system to process bulk renewals,*

*-- updating status, expiry date, and renewal count.*

*-- Purpose:*

*-- To update membership records in batch by extending expiry dates,*

*-- resetting status to ACTIVE, and logging success or failure.*

*-- Input Arguments:*

*-- p\_renewal\_type       VARCHAR2   scope of renewals: 'EXPIRED\_ONLY', 'URGENT', or 'ALL\_DUE'*

*-- p\_extension\_months   NUMBER     number of months to extend the expiry by*

*--------------------------------------------------------------------------------*

CREATE OR REPLACE PROCEDURE PROC\_RENEW\_MEMBERSHIPS (

    p\_renewal\_type IN VARCHAR2 DEFAULT 'EXPIRED\_ONLY',

    p\_extension\_months IN NUMBER DEFAULT 12

) AS

    v\_renewed\_count NUMBER := 0;

    v\_failed\_count NUMBER := 0;

    v\_start\_time TIMESTAMP := SYSTIMESTAMP;

    v\_triggers\_disabled BOOLEAN := FALSE;

BEGIN

*-- Validate parameters*

    IF p\_renewal\_type NOT IN ('EXPIRED\_ONLY', 'URGENT', 'ALL\_DUE') THEN

*-- Log invalid renewal type*

        LOG\_ACTIVITY(

            NULL,  *-- NULL for batch operation*

            'RENEWAL',

            'Invalid renewal type: ' || p\_renewal\_type,

            SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

        );

        RAISE\_APPLICATION\_ERROR(-20001, 'Invalid renewal type. Must be EXPIRED\_ONLY, URGENT, or ALL\_DUE');

    END IF;

    IF p\_extension\_months <= 0 THEN

*-- Log invalid extension months*

        LOG\_ACTIVITY(

            NULL,  *-- NULL for batch operation*

            'RENEWAL',

            'Invalid extension months: ' || p\_extension\_months,

            SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

        );

        RAISE\_APPLICATION\_ERROR(-20002, 'Extension months must be positive');

    END IF;

    BEGIN

        EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_sync\_point\_balance DISABLE';

        EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_membership\_history DISABLE';

        EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_auto\_assign\_vouchers DISABLE';

        v\_triggers\_disabled := TRUE;

    EXCEPTION

        WHEN OTHERS THEN

            DBMS\_OUTPUT.PUT\_LINE('Warning: Could not disable triggers - ' || SUBSTR(SQLERRM, 1, 200));

    END;

    SAVEPOINT before\_renewal\_batch;

    FOR cust IN (

        SELECT c.customer\_id, c.name, mt.renewal\_fee, c.membership\_expiry\_date

        FROM CUSTOMERS c

        JOIN MEMBERSHIP\_TIERS mt ON c.tier\_id = mt.tier\_id

        WHERE c.is\_member = 'Y'

          AND (

              (p\_renewal\_type = 'EXPIRED\_ONLY' AND c.membership\_expiry\_date <= SYSDATE) OR

              (p\_renewal\_type = 'URGENT' AND c.membership\_expiry\_date BETWEEN SYSDATE AND SYSDATE + 7) OR

              (p\_renewal\_type = 'ALL\_DUE' AND c.membership\_expiry\_date <= SYSDATE + 30)

          )

        FOR UPDATE

    ) LOOP

        BEGIN

            SAVEPOINT before\_customer\_renewal;

            DECLARE

                v\_payment\_received BOOLEAN := TRUE;

                v\_new\_expiry\_date DATE := ADD\_MONTHS(GREATEST(cust.membership\_expiry\_date, SYSDATE), p\_extension\_months);

            BEGIN

                IF v\_payment\_received THEN

                    UPDATE CUSTOMERS

                    SET membership\_expiry\_date = v\_new\_expiry\_date,

                        membership\_status = 'ACTIVE',

                        last\_renewal\_date = SYSDATE,

                        renewal\_count = renewal\_count + 1

                    WHERE customer\_id = cust.customer\_id;

                    v\_renewed\_count := v\_renewed\_count + 1;

                ELSE

                    ROLLBACK TO before\_customer\_renewal;

                    UPDATE CUSTOMERS

                    SET membership\_status = 'PENDING\_PAYMENT'

                    WHERE customer\_id = cust.customer\_id;

                    v\_failed\_count := v\_failed\_count + 1;

                END IF;

            END;

        EXCEPTION

            WHEN OTHERS THEN

*-- Log individual errors for each failed renewal*

                LOG\_ACTIVITY(

                    cust.customer\_id,

                    'RENEWAL',

                    'Error renewing customer ' || cust.customer\_id || ': ' || SQLERRM,

                    SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

                );

                ROLLBACK TO before\_customer\_renewal;

                v\_failed\_count := v\_failed\_count + 1;

                DBMS\_OUTPUT.PUT\_LINE('Error renewing customer ' || cust.customer\_id || ': ' || SUBSTR(SQLERRM, 1, 200));

        END;

    END LOOP;

*-- Log batch result*

    LOG\_ACTIVITY(

        NULL,                       *-- use NULL for a batch operation*

        'RENEWAL',                  *-- activity type*

        'Renewed ' || v\_renewed\_count || ' memberships, failed ' || v\_failed\_count,

        SYS\_CONTEXT('USERENV','IP\_ADDRESS')  *-- optional for IP address*

    );

    COMMIT;

    IF v\_triggers\_disabled THEN

        BEGIN

            EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_sync\_point\_balance ENABLE';

            EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_membership\_history ENABLE';

            EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_auto\_assign\_vouchers ENABLE';

        EXCEPTION

            WHEN OTHERS THEN

                DBMS\_OUTPUT.PUT\_LINE('Warning: Could not re-enable triggers - ' || SUBSTR(SQLERRM, 1, 200));

        END;

    END IF;

    DBMS\_OUTPUT.PUT\_LINE('Memberships renewed: ' || v\_renewed\_count);

    DBMS\_OUTPUT.PUT\_LINE('Renewal failures:    ' || v\_failed\_count);

EXCEPTION

    WHEN OTHERS THEN

        IF v\_triggers\_disabled THEN

            BEGIN

                EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_sync\_point\_balance ENABLE';

                EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_membership\_history ENABLE';

                EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_auto\_assign\_vouchers ENABLE';

            EXCEPTION

                WHEN OTHERS THEN NULL;

            END;

        END IF;

        ROLLBACK TO before\_renewal\_batch;

        DBMS\_OUTPUT.PUT\_LINE('Critical Error: ' || SUBSTR(SQLERRM, 1, 200));

        RAISE;

END;

/

*--------------------------------------------------------------------------------*

*-- DEMO TEST 1 (NORMAL CASE): EXECUTE PROC\_RENEW\_MEMBERSHIPS*

*--------------------------------------------------------------------------------*

*-- Expected Output:*

*-- - Memberships renewed: [count]*

*-- - Renewal failures: [count]*

*--------------------------------------------------------------------------------*

BEGIN

*-- First create test data*

    BEGIN

        EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_sync\_point\_balance DISABLE';

        EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_membership\_history DISABLE';

*-- Update some customers to have expired memberships*

        UPDATE CUSTOMERS

        SET membership\_expiry\_date = SYSDATE-10,

            membership\_status = 'ACTIVE'

        WHERE customer\_id IN (1001, 1002);

        EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_sync\_point\_balance ENABLE';

        EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_membership\_history ENABLE';

        COMMIT;

    EXCEPTION

        WHEN OTHERS THEN

            DBMS\_OUTPUT.PUT\_LINE('Error setting up test data: ' || SQLERRM);

    END;

    DBMS\_OUTPUT.PUT\_LINE('=== TESTING NORMAL RENEWAL ===');

    PROC\_RENEW\_MEMBERSHIPS(p\_renewal\_type => 'EXPIRED\_ONLY');

END;

/

*/\*--------------------------------------------------------------------------------*

*-- DEMO TEST 2 (ERROR CASE): EXECUTE PROC\_RENEW\_MEMBERSHIPS*

*--------------------------------------------------------------------------------*

*-- Testing invalid renewal type and negative extension months*

*--------------------------------------------------------------------------------*

*BEGIN*

*DBMS\_OUTPUT.PUT\_LINE('=== TESTING INVALID RENEWAL TYPE ===');*

*BEGIN*

*PROC\_RENEW\_MEMBERSHIPS(p\_renewal\_type => 'INVALID\_TYPE');*

*DBMS\_OUTPUT.PUT\_LINE('ERROR: Should not reach here with invalid type');*

*EXCEPTION*

*WHEN OTHERS THEN*

*DBMS\_OUTPUT.PUT\_LINE('Expected error: ' || SUBSTR(SQLERRM, 1, 200));*

*END;*

*DBMS\_OUTPUT.PUT\_LINE('=== TESTING NEGATIVE EXTENSION MONTHS ===');*

*BEGIN*

*PROC\_RENEW\_MEMBERSHIPS(p\_extension\_months => -5);*

*DBMS\_OUTPUT.PUT\_LINE('ERROR: Should not reach here with negative months');*

*EXCEPTION*

*WHEN OTHERS THEN*

*DBMS\_OUTPUT.PUT\_LINE('Expected error: ' || SUBSTR(SQLERRM, 1, 200));*

*END;*

*END;*

*/ \*/*

*--------------------------------------------------------------------------------*

*-- PROCEDURE 2: PROC\_EXPIRE\_POINTS*

*--------------------------------------------------------------------------------*

*-- User Transaction:*

*-- Expire points that have reached their expiry date and mark them as expired.*

*-- Then evaluate whether affected customers still qualify for their current tier,*

*-- and downgrade their membership tier if necessary.*

*-- Purpose:*

*-- To maintain integrity of point balances and ensure customers are correctly tiered*

*-- based on their active points after expirations.*

*-- Input Arguments:*

*-- None*

*--------------------------------------------------------------------------------*

CREATE OR REPLACE PROCEDURE PROC\_EXPIRE\_POINTS AS

    v\_expired\_count NUMBER := 0;

    v\_customers\_affected NUMBER := 0;

    v\_tier\_downgrades NUMBER := 0;

    v\_start\_time TIMESTAMP := SYSTIMESTAMP;

    v\_batch\_count NUMBER := 0;

    v\_triggers\_disabled BOOLEAN := FALSE;

BEGIN

*-- Disable triggers at start*

    BEGIN

        EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_sync\_point\_balance DISABLE';

        EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_membership\_history DISABLE';

        EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_auto\_assign\_vouchers DISABLE';

        v\_triggers\_disabled := TRUE;

    EXCEPTION

        WHEN OTHERS THEN

            DBMS\_OUTPUT.PUT\_LINE('Warning: Could not disable triggers - ' || SUBSTR(SQLERRM, 1, 200));

    END;

    SAVEPOINT before\_expiration\_process;

    DBMS\_OUTPUT.PUT\_LINE('Starting points expiration at ' || TO\_CHAR(v\_start\_time, 'YYYY-MM-DD HH24:MI:SS'));

*-- Process expiring points*

    FOR exp\_rec IN (

        SELECT pt.transaction\_id, pt.customer\_id, pt.points\_amount,

               c.name, c.tier\_id, c.points\_balance

        FROM POINT\_TRANSACTIONS pt

        JOIN CUSTOMERS c ON pt.customer\_id = c.customer\_id

        WHERE pt.expiry\_date <= TRUNC(SYSDATE)

          AND pt.points\_amount > 0

          AND pt.transaction\_type IN ('PURCHASE', 'BONUS')

          AND c.points\_balance >= pt.points\_amount  *-- Ensure balance won't go negative*

          AND NOT EXISTS (  *-- Check if already expired*

              SELECT 1 FROM POINT\_TRANSACTIONS exp

              WHERE exp.transaction\_type = 'EXPIRY'

              AND exp.description LIKE '%from transaction ' || pt.transaction\_id || '%'

          )

        ORDER BY pt.customer\_id, pt.expiry\_date, pt.transaction\_date

    ) LOOP

        BEGIN

            SAVEPOINT before\_point\_expiration;

*-- First verify the customer still has sufficient points*

            DECLARE

                v\_current\_balance NUMBER;

            BEGIN

                SELECT points\_balance INTO v\_current\_balance

                FROM CUSTOMERS

                WHERE customer\_id = exp\_rec.customer\_id

                FOR UPDATE;

                IF v\_current\_balance < exp\_rec.points\_amount THEN

                    RAISE\_APPLICATION\_ERROR(-20003, 'Insufficient points balance for expiration');

                END IF;

*-- Create the negative transaction*

                INSERT INTO POINT\_TRANSACTIONS (

                    customer\_id,

                    points\_amount,

                    transaction\_type,

                    description,

                    transaction\_date

                ) VALUES (

                    exp\_rec.customer\_id,

                    -exp\_rec.points\_amount,

                    'EXPIRY',

                    'Points expired from transaction ' || exp\_rec.transaction\_id,

                    SYSDATE

                );

*-- Update the customer's points balance to reflect the deduction*

                UPDATE CUSTOMERS

                SET points\_balance = points\_balance - exp\_rec.points\_amount

                WHERE customer\_id = exp\_rec.customer\_id;

*-- Add this after the customer balance update*

                DBMS\_OUTPUT.PUT\_LINE('Updated customer ' || exp\_rec.customer\_id ||

                     ' from ' || v\_current\_balance ||

                     ' to ' || (v\_current\_balance - exp\_rec.points\_amount));

*-- Mark the original transaction as expired*

                UPDATE POINT\_TRANSACTIONS

                SET expiry\_date = NULL,

                    description = description || ' - EXPIRED'

                WHERE transaction\_id = exp\_rec.transaction\_id;

                v\_expired\_count := v\_expired\_count + exp\_rec.points\_amount;

                v\_customers\_affected := v\_customers\_affected + 1;

                v\_batch\_count := v\_batch\_count + 1;

            END;

        EXCEPTION

            WHEN OTHERS THEN

                ROLLBACK TO before\_point\_expiration;

                DBMS\_OUTPUT.PUT\_LINE('Error expiring points for TXN ' || exp\_rec.transaction\_id || ': ' || SUBSTR(SQLERRM, 1, 200));

        END;

    END LOOP;

*-- Process tier downgrades if needed*

    DBMS\_OUTPUT.PUT\_LINE('Starting tier evaluation for ' || v\_customers\_affected || ' affected customers');

    FOR cust IN (

        SELECT DISTINCT pt.customer\_id, c.name

        FROM POINT\_TRANSACTIONS pt

        JOIN CUSTOMERS c ON pt.customer\_id = c.customer\_id

        WHERE pt.transaction\_type = 'EXPIRY'

          AND TRUNC(pt.transaction\_date) = TRUNC(SYSDATE)

    ) LOOP

        BEGIN

            SAVEPOINT before\_tier\_evaluation;

            DECLARE

                v\_current\_tier NUMBER;

                v\_current\_points NUMBER;

                v\_qualified\_tier NUMBER;

                v\_current\_tier\_name VARCHAR2(50);

                v\_new\_tier\_name VARCHAR2(50);

            BEGIN

*-- Get current tier info with FOR UPDATE*

                SELECT c.tier\_id, c.points\_balance, mt.tier\_name

                INTO v\_current\_tier, v\_current\_points, v\_current\_tier\_name

                FROM CUSTOMERS c

                JOIN MEMBERSHIP\_TIERS mt ON c.tier\_id = mt.tier\_id

                WHERE c.customer\_id = cust.customer\_id

                FOR UPDATE;

*-- Find the highest tier the customer qualifies for*

                SELECT MAX(mt.tier\_id), MAX(mt.tier\_name)

                INTO v\_qualified\_tier, v\_new\_tier\_name

                FROM MEMBERSHIP\_TIERS mt

                WHERE mt.points\_required <= v\_current\_points;

*-- Downgrade if needed*

                IF v\_qualified\_tier < v\_current\_tier THEN

                    UPDATE CUSTOMERS

                    SET tier\_id = v\_qualified\_tier

                    WHERE customer\_id = cust.customer\_id;

                    v\_tier\_downgrades := v\_tier\_downgrades + 1;

                    DBMS\_OUTPUT.PUT\_LINE('Downgraded ' || cust.name || ' from ' || v\_current\_tier\_name || ' to ' || v\_new\_tier\_name);

                END IF;

            EXCEPTION

                WHEN NO\_DATA\_FOUND THEN

                    ROLLBACK TO before\_tier\_evaluation;

                    DBMS\_OUTPUT.PUT\_LINE('Tier data not found for ' || cust.name);

            END;

        EXCEPTION

            WHEN OTHERS THEN

                ROLLBACK TO before\_tier\_evaluation;

                DBMS\_OUTPUT.PUT\_LINE('Error evaluating tier for ' || cust.name || ': ' || SUBSTR(SQLERRM, 1, 200));

        END;

    END LOOP;

*-- Log Activity*

    LOG\_ACTIVITY(

        NULL,                         *-- batch operation*

        'EXPIRATION',

        'Expired ' || v\_expired\_count || ' points for ' || v\_customers\_affected

          || ' customers, ' || v\_tier\_downgrades || ' downgrades.',

        SYS\_CONTEXT('USERENV','IP\_ADDRESS')

    );

    COMMIT;

*-- Re-enable triggers if they were disabled*

    IF v\_triggers\_disabled THEN

        BEGIN

            EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_sync\_point\_balance ENABLE';

            EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_membership\_history ENABLE';

            EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_auto\_assign\_vouchers ENABLE';

        EXCEPTION

            WHEN OTHERS THEN

                DBMS\_OUTPUT.PUT\_LINE('Warning: Could not re-enable triggers - ' || SUBSTR(SQLERRM, 1, 200));

        END;

    END IF;

    DBMS\_OUTPUT.PUT\_LINE('----------------------------------');

    DBMS\_OUTPUT.PUT\_LINE('Points expiration completed at ' || TO\_CHAR(SYSTIMESTAMP, 'YYYY-MM-DD HH24:MI:SS'));

    DBMS\_OUTPUT.PUT\_LINE('Total processing time: ' || EXTRACT(SECOND FROM (SYSTIMESTAMP - v\_start\_time)) || ' seconds');

    DBMS\_OUTPUT.PUT\_LINE('Results:');

    DBMS\_OUTPUT.PUT\_LINE('- Points expired: ' || v\_expired\_count);

    DBMS\_OUTPUT.PUT\_LINE('- Customers affected: ' || v\_customers\_affected);

EXCEPTION

    WHEN OTHERS THEN

        ROLLBACK TO before\_expiration\_process;

*-- Ensure triggers are re-enabled if they were disabled*

        IF v\_triggers\_disabled THEN

            BEGIN

                EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_sync\_point\_balance ENABLE';

                EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_membership\_history ENABLE';

                EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_auto\_assign\_vouchers ENABLE';

            EXCEPTION WHEN OTHERS THEN NULL;

            END;

        END IF;

        DBMS\_OUTPUT.PUT\_LINE('CRITICAL ERROR in PROC\_EXPIRE\_POINTS: ' || SUBSTR(SQLERRM, 1, 200));

        DBMS\_OUTPUT.PUT\_LINE('Entire operation rolled back. No changes committed.');

        RAISE;

END;

/

*--------------------------------------------------------------------------------*

*-- DEMO TEST 1 (NORMAL CASE): EXECUTE PROC\_EXPIRE\_POINTS*

*--------------------------------------------------------------------------------*

*-- Insert test point transactions that should expire*

INSERT INTO POINT\_TRANSACTIONS (

    customer\_id, points\_amount, transaction\_type,

    description, transaction\_date, expiry\_date

)

SELECT

    customer\_id,

    100,

    'PURCHASE',

    'Test expiring points',

    SYSDATE-60,

    SYSDATE-1

FROM CUSTOMERS

WHERE customer\_id IN (1001, 1002, 1003);

COMMIT;

*-- Execute the procedure with trigger disabled*

BEGIN

    EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_sync\_point\_balance DISABLE';

    DBMS\_OUTPUT.PUT\_LINE('=== TESTING NORMAL POINT EXPIRATION ===');

    PROC\_EXPIRE\_POINTS;

    EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_sync\_point\_balance ENABLE';

EXCEPTION

    WHEN OTHERS THEN

        BEGIN

            EXECUTE IMMEDIATE 'ALTER TRIGGER trg\_sync\_point\_balance ENABLE';

        EXCEPTION WHEN OTHERS THEN NULL;

        END;

        RAISE;

END;

/

*-- Verify points were expired*

SELECT c.customer\_id, c.name, c.points\_balance,

       (SELECT SUM(points\_amount) FROM POINT\_TRANSACTIONS pt

        WHERE pt.customer\_id = c.customer\_id) AS total\_points

FROM CUSTOMERS c

WHERE customer\_id IN (1001, 1002, 1003);

*-- Verify expired transactions were marked*

SELECT transaction\_id, customer\_id, points\_amount,

       transaction\_type, description, expiry\_date

FROM POINT\_TRANSACTIONS

WHERE description LIKE '%Test expiring points%';

*-- Check for tier downgrades if applicable*

SELECT mh.customer\_id, c.name,

       old\_t.tier\_name AS old\_tier,

       new\_t.tier\_name AS new\_tier,

       mh.change\_date, mh.change\_reason

FROM MEMBERSHIP\_HISTORY mh

JOIN CUSTOMERS c ON mh.customer\_id = c.customer\_id

JOIN MEMBERSHIP\_TIERS old\_t ON mh.old\_tier\_id = old\_t.tier\_id

JOIN MEMBERSHIP\_TIERS new\_t ON mh.new\_tier\_id = new\_t.tier\_id

WHERE mh.change\_date >= TRUNC(SYSDATE)

ORDER BY mh.change\_date DESC;

*--Expected output:*

*/\**

*=== TESTING NORMAL POINT EXPIRATION ===*

*Starting points expiration at 2025-04-28 19:36:50*

*Updated customer 1001 from 400 to 300*

*Updated customer 1002 from 850 to 750*

*Updated customer 1003 from 1100 to 1000*

*Updated customer 1006 from 500 to 400*

*Starting tier evaluation for 4 affected customers*

*----------------------------------*

*Points expiration completed at 2025-04-28 19:36:50*

*Total processing time: .096 seconds*

*Results:*

*- Points expired: 400*

*- Customers affected: 4*

*PL/SQL procedure successfully completed.*

*\*/*

*--Expected Verification*

*/\**

*CUSTOMER\_ID NAME                                                                                             POINTS\_BALANCE TOTAL\_POINTS*

*----------- ---------------------------------------------------------------------------------------------------- -------------- ------------*

*1001 John Smith                                                                                          300     0*

*1002 Emily Johnson                                                                                       750     0*

*1003 Michael Lee                                                                                        1000     0*

*TRANSACTION\_ID CUSTOMER\_ID POINTS\_AMOUNT TRANSACTION\_TYPE*

*-------------- ----------- ------------- --------------------*

*DESCRIPTION*

*--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------*

*EXPIRY\_DA*

*---------*

*6        1001           100 PURCHASE*

*Test expiring points - EXPIRED*

*7        1002           100 PURCHASE*

*Test expiring points - EXPIRED*

*8        1003           100 PURCHASE*

*Test expiring points - EXPIRED*

*CUSTOMER\_ID NAME                                                                                             OLD\_TIER              NEW\_TIER*

*----------- ---------------------------------------------------------------------------------------------------- -------------------- --------------------*

*CHANGE\_DATE                                                                 CHANGE\_REASON*

*--------------------------------------------------------------------------- ----------------------------------------------------------------------------------------------------*

*1003 Michael Lee                                                                                      Gold                  Silver*

*28-APR-25 07.36.50.656000 PM                                                DOWNGRADE*

*1005 David Wong                                                                                       Bronze        Gold*

*28-APR-25 07.36.49.236000 PM                                                ADMIN*

*1003 Michael Lee                                                                                      Bronze        Silver*

*28-APR-25 07.36.49.220000 PM                                                ADMIN*

*\*/*

*/\**

*--------------------------------------------------------------------------------*

*-- DEMO TEST 2 (ERROR CASE): EXECUTE PROC\_EXPIRE\_POINTS*

*--------------------------------------------------------------------------------*

*BEGIN*

*DBMS\_OUTPUT.PUT\_LINE('=== TESTING WITH LOCKED RECORDS ===');*

*DECLARE*

*v\_test\_cust\_id NUMBER;*

*BEGIN*

*-- Lock a customer record TO GET THE ERROR TEST*

*SELECT customer\_id INTO v\_test\_cust\_id FROM CUSTOMERS WHERE ROWNUM = 1 FOR UPDATE;*

*-- Try to expire points while records are locked*

*-- SHOULD BE NO RECORD*

*PROC\_EXPIRE\_POINTS;*

*COMMIT; -- Release the lock*

*EXCEPTION*

*WHEN OTHERS THEN*

*DBMS\_OUTPUT.PUT\_LINE('Expected NOTHING ENTER HERE');*

*ROLLBACK;*

*END;*

*END;*

*/*

*--EXPECTED ERROR:*

*/\**

*It will Skip the locked record, causing no points to be expired.*

*Encounter a lock wait condition or deadlock, which would prevent the procedure from processing the record until the lock is released.*

*\*/*

*-- Expected verification: no record changes*

*-- Verify points were expired*

*SELECT c.customer\_id, c.name, c.points\_balance,*

*(SELECT SUM(points\_amount) FROM POINT\_TRANSACTIONS pt*

*WHERE pt.customer\_id = c.customer\_id) AS total\_points*

*FROM CUSTOMERS c*

*WHERE customer\_id IN (1001, 1002, 1003);*

*-- Verify expired transactions were marked*

*SELECT transaction\_id, customer\_id, points\_amount,*

*transaction\_type, description, expiry\_date*

*FROM POINT\_TRANSACTIONS*

*WHERE description LIKE '%Test expiring points%';*

*-- Check for tier downgrades if applicable*

*SELECT mh.customer\_id, c.name,*

*old\_t.tier\_name AS old\_tier,*

*new\_t.tier\_name AS new\_tier,*

*mh.change\_date, mh.change\_reason*

*FROM MEMBERSHIP\_HISTORY mh*

*JOIN CUSTOMERS c ON mh.customer\_id = c.customer\_id*

*JOIN MEMBERSHIP\_TIERS old\_t ON mh.old\_tier\_id = old\_t.tier\_id*

*JOIN MEMBERSHIP\_TIERS new\_t ON mh.new\_tier\_id = new\_t.tier\_id*

*WHERE mh.change\_date >= TRUNC(SYSDATE)*

*ORDER BY mh.change\_date DESC;*

*\*/*

*--------------------------------------------------------------------------------*

*-- FUNCTION 1: FN\_CALC\_UPGRADE\_VALUE*

*--------------------------------------------------------------------------------*

*-- User Transaction:*

*-- Check how much a customer would benefit from upgrading to the next membership tier,*

*-- including discount percentage increase, points needed, and fee difference.*

*-- User Transaction Type: Personalized Upgrade Recommendation*

*-- Typical User: Customer considering tier benefits or staff advising on upgrades*

*-- Transaction Flow:*

*-- User views tier benefits in profile/checkout*

*-- System calls FN\_CALC\_UPGRADE\_VALUE(customer\_id)*

*-- Returns formatted message like:*

*--"Upgrade to Silver | Discount Increase: +5% | Points Needed: 200 | Renewal Fee Change: +$5"*

*-- Purpose:*

*-- To provide an informative message comparing the customer's current tier with the next one.*

*-- Input Arguments:*

*-- p\_customer\_id  NUMBER   the ID of the customer to evaluate*

*-- Return:*

*-- VARCHAR2  summary string showing potential upgrade benefit or status*

*--------------------------------------------------------------------------------*

CREATE OR REPLACE FUNCTION FN\_CALC\_UPGRADE\_VALUE (

    p\_customer\_id IN NUMBER

) RETURN VARCHAR2 AS

    v\_result VARCHAR2(1000);

BEGIN

    SELECT

        'Upgrade to ' || next\_tier.tier\_name ||

        ' | Discount Increase: +' || (next\_tier.discount\_percentage - current\_tier.discount\_percentage) || '%' ||

        ' | Points Needed: ' || (next\_tier.points\_required - c.points\_balance) ||

        ' | Renewal Fee Change: ' ||

            CASE

                WHEN next\_tier.renewal\_fee > current\_tier.renewal\_fee THEN '+$' || (next\_tier.renewal\_fee - current\_tier.renewal\_fee)

                WHEN next\_tier.renewal\_fee < current\_tier.renewal\_fee THEN '-$' || (current\_tier.renewal\_fee - next\_tier.renewal\_fee)

                ELSE 'No change'

            END

    INTO v\_result

    FROM CUSTOMERS c

    JOIN MEMBERSHIP\_TIERS current\_tier ON c.tier\_id = current\_tier.tier\_id

    JOIN MEMBERSHIP\_TIERS next\_tier ON next\_tier.tier\_id = current\_tier.tier\_id + 1

    WHERE c.customer\_id = p\_customer\_id;

    LOG\_ACTIVITY(

        p\_customer\_id,

        'TIER\_CHANGE',  *-- activity type is TIER\_CHANGE for this case*

        'Calculated upgrade for customer ' || p\_customer\_id || ': ' || v\_result,

        SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')  *-- optional: captures the client IP address*

    );

    RETURN v\_result;

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        RETURN 'At highest tier or invalid customer';

END;

/

*--------------------------------------------------------------------------------*

*-- DEMO TEST 1 (NORMAL CASE): FN\_CALC\_UPGRADE\_VALUE*

*--------------------------------------------------------------------------------*

*-- Expected Output:*

*-- - Descriptive upgrade message with discount, fee, and points gap*

*-- === TESTING UPGRADE CALCULATION ===*

*-- Upgrade to Silver | Discount Increase: +5% | Points Needed: 200 | Renewal Fee Change: +$5*

*--------------------------------------------------------------------------------*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('=== TESTING UPGRADE CALCULATION ===');

    DBMS\_OUTPUT.PUT\_LINE(FN\_CALC\_UPGRADE\_VALUE(1001));

END;

/

*/\*--------------------------------------------------------------------------------*

*-- DEMO TEST 2 (ERROR CASE): FN\_CALC\_UPGRADE\_VALUE*

*--------------------------------------------------------------------------------*

*-- Testing with invalid customer ID and highest tier customer*

*--Expected Output: At highest tier or invalid customer*

*BEGIN*

*DBMS\_OUTPUT.PUT\_LINE('=== TESTING INVALID CUSTOMER ID ===');*

*DBMS\_OUTPUT.PUT\_LINE(FN\_CALC\_UPGRADE\_VALUE(9999));*

*DBMS\_OUTPUT.PUT\_LINE('=== TESTING HIGHEST TIER CUSTOMER ===');*

*DBMS\_OUTPUT.PUT\_LINE(FN\_CALC\_UPGRADE\_VALUE(1005));*

*END;*

*/ \*/*

*--------------------------------------------------------------------------------*

*-- FUNCTION 2: FN\_CHECK\_MEMBERSHIP\_HEALTH*

*--------------------------------------------------------------------------------*

*-- User Transaction:*

*-- Check how close a customer's membership is to expiring or whether it's overdue,*

*-- and return a health status message including tier and last renewal info.*

*-- Purpose:*

*-- To return a readable status about the customer's membership health and urgency.*

*-- User Transaction Type: Customer Self-Service Inquiry*

*-- Typical User: Customer or frontline staff assisting a specific customer*

*-- Transaction Flow:*

*-- Customer accesses their profile (web/mobile/app)*

*-- System calls FN\_CHECK\_MEMBERSHIP\_HEALTH(customer\_id)*

*-- Returns a single human-readable status message like:*

*--"Silver membership: CRITICAL - renew in 3 days"*

*-- Input Arguments:*

*-- p\_customer\_id  NUMBER   the ID of the customer*

*-- Return:*

*-- VARCHAR2  description of the customer's current membership status*

*--------------------------------------------------------------------------------*

CREATE OR REPLACE FUNCTION FN\_CHECK\_MEMBERSHIP\_HEALTH (

    p\_customer\_id IN NUMBER

) RETURN VARCHAR2 AS

    v\_status VARCHAR2(200);

    v\_expiry\_days NUMBER;

    v\_last\_renewal DATE;

    v\_tier\_name VARCHAR2(50);

BEGIN

    SELECT

        FLOOR(membership\_expiry\_date - SYSDATE),

        membership\_status,

        last\_renewal\_date,

        mt.tier\_name

    INTO

        v\_expiry\_days,

        v\_status,

        v\_last\_renewal,

        v\_tier\_name

    FROM CUSTOMERS c

    JOIN MEMBERSHIP\_TIERS mt ON c.tier\_id = mt.tier\_id

    WHERE customer\_id = p\_customer\_id;

    RETURN

        v\_tier\_name || ' membership: ' ||

        CASE

            WHEN v\_status != 'ACTIVE' THEN v\_status || ' (last renewed: ' || TO\_CHAR(v\_last\_renewal, 'DD-MON-YYYY') || ')'

            WHEN v\_expiry\_days <= 0 THEN 'EXPIRED ' || ABS(v\_expiry\_days) || ' days ago'

            WHEN v\_expiry\_days <= 7 THEN 'CRITICAL - renew in ' || v\_expiry\_days || ' days'

            WHEN v\_expiry\_days <= 30 THEN 'Renewal due in ' || v\_expiry\_days || ' days'

            ELSE 'Active (expires in ' || v\_expiry\_days || ' days)'

        END;

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        RETURN 'Customer not found';

    WHEN OTHERS THEN

        RETURN 'Error checking status: ' || SUBSTR(SQLERRM, 1, 100);

END;

/

*--------------------------------------------------------------------------------*

*-- DEMO TEST 1 (NORMAL CASE): FN\_CHECK\_MEMBERSHIP\_HEALTH*

*--------------------------------------------------------------------------------*

*-- Expected Output:*

*-- Status like "Active", "Expired X days ago", "Renewal due in Y days"*

*-- Output: Bronze membership: Active (expires in 364 days)*

*--------------------------------------------------------------------------------*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('=== TESTING MEMBERSHIP HEALTH ===');

    DBMS\_OUTPUT.PUT\_LINE(FN\_CHECK\_MEMBERSHIP\_HEALTH(1001));

END;

/

*/\**

*--------------------------------------------------------------------------------*

*-- DEMO TEST 2 (ERROR CASE): FN\_CHECK\_MEMBERSHIP\_HEALTH*

*--------------------------------------------------------------------------------*

*-- Testing with invalid customer ID and expired membership*

*-- Expected Output: Customer not found*

*BEGIN*

*DBMS\_OUTPUT.PUT\_LINE('=== TESTING INVALID CUSTOMER ID ===');*

*DBMS\_OUTPUT.PUT\_LINE(FN\_CHECK\_MEMBERSHIP\_HEALTH(9999));*

*DBMS\_OUTPUT.PUT\_LINE('=== TESTING EXPIRED MEMBERSHIP ===');*

*-- Test with existing test customer (1016)*

*DBMS\_OUTPUT.PUT\_LINE(FN\_CHECK\_MEMBERSHIP\_HEALTH(1016));*

*END;*

*/*

*\*/*

*--------------------------------------------------------------------------------*

*-- FUNTIONS are different USER TRANSACTIONS with the QUERIES*

*--------------------------------------------------------------------------------*

*/\**

*Scenario 1: HOW THE FUNCTIONS WORK TOGETHER*

*User: Customer John Smith*

*Goal: Check membership status*

*User use Mobile App Calls FN\_CHECK\_MEMBERSHIP\_HEALTH*

*Output: "Bronze membership: URGENT - renew in 12 days"*

*Then, user ask for Upgrade Suggestion Called by FN\_CALC\_UPGRADE\_VALUE:*

*Output: "Upgrade to Silver | Discount Increase: +5% | Points Needed: 200 | Renewal Fee Change: +$5"*

*Scenario 2: HOW QUERY UPCOMING RENEWAL Different from FN\_CHECK\_MEMBERSHIP\_HEALTH*

*Use in Administrative Batch Processing, thus if a Staff act as User: Membership Manager Sarah*

*Goal: Prepare for upcoming renewals*

*Sarah exports this to Excel to plan renewal campaigns. Batch Renewal Processing via called the PROC\_RENEW\_MEMBERSHIPS procedure. Output: "Memberships renewed: 42, Renewal failures: 3"*

*Scenario 3:HOW QUERY TIER COMPARISON Different from FN\_CALC\_UPGRADE\_VALUE*

*Use as Analytical Review (in Management)*

*if a Staff act a User: Marketing Director David*

*Goal: Analyze tier distribution*

*David Run Tier Comparison Query and Uses query results to:*

*1. Identify 1,200 customers near tier thresholds*

*2. Allocate budget for targeted promotional*

*3. Adjust point requirements for underperforming tiers*

*\*/*

*--------------------------------------------------------------------------------*

*-- QUERY 1: UPCOMING RENEWALS*

*--------------------------------------------------------------------------------*

*-- User Transaction:*

*-- Display a list of all customers whose memberships are about to expire within the next*

*-- 30 days, including those already expired. Show customer name, email, tier, expiry date,*

*-- days until expiry, and renewal fee. Useful for membership management or notifying users.*

*-- Purpose:*

*-- To retrieve and classify members into categories like EXPIRED, URGENT, UPCOMING, or ACTIVE*

*-- based on how soon their memberships are expiring.*

*-- User Transaction Type: Administrative Batch Processing*

*-- Typical User: Membership manager or automated renewal system*

*-- Transaction Flow:*

*--Staff runs "Renewals Due" report (or automated job executes)*

*--Query returns tabular data of all expiring memberships*

*--Used to:*

*--Generate renewal notices*

*--Allocate staff for renewal calls*

*--Forecast cash flow from renewal fees*

*--------------------------------------------------------------------------------*

SELECT

    c.customer\_id,

    c.name,

    c.email,

    mt.tier\_name,

    c.membership\_expiry\_date,

    FLOOR(c.membership\_expiry\_date - SYSDATE) AS days\_until\_expiry,

    mt.renewal\_fee,

    CASE

        WHEN c.membership\_expiry\_date <= SYSDATE THEN 'EXPIRED'

        WHEN c.membership\_expiry\_date <= SYSDATE + 7 THEN 'URGENT'

        WHEN c.membership\_expiry\_date <= SYSDATE + 30 THEN 'UPCOMING'

        ELSE 'ACTIVE'

    END AS renewal\_status

FROM

    CUSTOMERS c

JOIN

    MEMBERSHIP\_TIERS mt ON c.tier\_id = mt.tier\_id

WHERE

    c.is\_member = 'Y'

ORDER BY

    c.membership\_expiry\_date;

*--------------------------------------------------------------------------------*

*-- DEMO TEST (NORMAL CASE): UPCOMING RENEWALS*

*--------------------------------------------------------------------------------*

*-- Expected Output:*

*-- - List of customers with expiring memberships within 30 days*

*-- - Includes their tier, fee, and days until expiry*

*--------------------------------------------------------------------------------*

*/\**

*CUSTOMER\_ID NAME*

*----------- ----------------------------------------------------------------------------------------------------*

*EMAIL                                                                                                TIER\_NAME                 MEMBERSHI DAYS\_UNTIL\_EXPIRY RENEWAL\_FEE RENEWAL\_*

*---------------------------------------------------------------------------------------------------- -------------------- --------- ----------------- ----------- --------*

*1003 Michael Lee*

*mikelee@example.com                                                                                  Silver   10-JAN-26                256          15 ACTIVE*

*1004 Sarah Tan*

*saraht@example.com                                                                                   Platinum  28-FEB-26               305          25 ACTIVE*

*1005 David Wong*

*davidw@example.com                                                                                   Diamond  05-MAR-26                310          30 ACTIVE*

*1002 Emily Johnson*

*emilyj@example.com                                                                                   Silver   28-APR-26                364          15 ACTIVE*

*1006 Test User for G001*

*test.user@example.com                                                                                Bronze   28-APR-26                364          10 ACTIVE*

*1001 John Smith*

*johnsmith@example.com                                                                                Bronze   28-APR-26                364          10 ACTIVE*

*6 rows selected.*

*\*/*

*--------------------------------------------------------------------------------*

*-- QUERY 2: TIER COMPARISON FOR A CUSTOMER*

*--------------------------------------------------------------------------------*

*-- User Transaction:*

*-- Retrieve the data from CUSTOMERS and MEMBERSHIP\_TIERS, to compare the current tier of a customer with the next tier to determine upgrade eligibility.*

*-- Includes calculating how many more points are needed, the discount difference, and the change in renewal fee.*

*-- User Transaction Type: Membership Tier Analysis*

*--Typical User: Marketing team or business analysts*

*--Transaction Flow:*

*--Analyst investigates tier distribution*

*--Runs query with specific customer filter*

*--Gets structured data showing:*

*--Current vs next tier metrics*

*--Points gap analysis*

*--Eligibility status flags*

*-- Purpose:*

*-- To provide tier upgrade insight by comparing a customer's current points balance*

*-- with the requirements of the next tier.*

*--------------------------------------------------------------------------------*

SELECT

    c.customer\_id,

    c.name,

    curr.tier\_name AS current\_tier,

    c.points\_balance,

    next.tier\_name AS next\_tier,

    next.points\_required - c.points\_balance AS points\_needed,

    next.discount\_percentage - curr.discount\_percentage AS additional\_discount,

    next.renewal\_fee - curr.renewal\_fee AS fee\_change,

    CASE

        WHEN next.tier\_id IS NULL THEN 'MAX\_TIER'

        WHEN c.points\_balance >= next.points\_required THEN 'ELIGIBLE\_NOW'

        ELSE 'NEEDS\_MORE\_POINTS'

    END AS upgrade\_status

FROM

    CUSTOMERS c

JOIN

    MEMBERSHIP\_TIERS curr ON c.tier\_id = curr.tier\_id

LEFT JOIN

    MEMBERSHIP\_TIERS next ON next.tier\_id = curr.tier\_id + 1

WHERE

    c.customer\_id = 1001;

*--------------------------------------------------------------------------------*

*-- DEMO TEST (NORMAL CASE): TIER COMPARISON*

*--------------------------------------------------------------------------------*

*-- Expected Output:*

*-- - Displays comparison of current vs. next tier, points gap, and potential benefits*

*--------------------------------------------------------------------------------*

*/\**

*CUSTOMER\_ID NAME                                                                                             CURRENT\_TIER          POINTS\_BALANCE NEXT\_TIER            POINTS\_NEEDED*

*----------- ---------------------------------------------------------------------------------------------------- -------------------- -------------- -------------------- -------------*

*ADDITIONAL\_DISCOUNT FEE\_CHANGE UPGRADE\_STATUS*

*------------------- ---------- -----------------*

*1001 John Smith                                                                                       Bronze                   300 Silver                         200*

*5          5 NEEDS\_MORE\_POINTS*

*\*/*

*--------------------------------------------------------------------------------*

*-- TEST CASE 1:  TEST POINT EARNING AND TIER VERIFICATION*

*--------------------------------------------------------------------------------*

*-- User Transaction:*

*-- This test simulates the awarding of additional points to a customer,*

*-- and then checks if the new point balance results in a tier upgrade.*

*-- It verifies that triggers update the customer's tier and assign rewards accordingly.*

*-- Purpose:*

*-- To validate the system behavior when a customer earns enough points to trigger an automatic*

*-- membership tier upgrade and logs this in the membership history.*

*--------------------------------------------------------------------------------*

PROMPT ===== TEST CASE 1   =====

*-- First, let's verify the current tier points requirements*

PROMPT ===== Tier Requirement  =====

SELECT tier\_id, tier\_name, points\_required

FROM MEMBERSHIP\_TIERS

ORDER BY tier\_id;

PROMPT =====  Initial Customer Status  =====

*-- Check the customer's current status before adding point*

*-- Current Status: After the above expiration test case, Customer 1001 starts in the Bronze tier with 300 points and needs 200 more to reach Silver.*

SELECT

    c.customer\_id,

    c.name,

    c.points\_balance,

    mt.tier\_name AS current\_tier,

    mt.points\_required AS current\_tier\_requirement,

    next\_t.tier\_name AS next\_tier,

    next\_t.points\_required AS next\_tier\_requirement,

    next\_t.points\_required - c.points\_balance AS points\_needed

FROM

    CUSTOMERS c

JOIN

    MEMBERSHIP\_TIERS mt ON c.tier\_id = mt.tier\_id

LEFT JOIN

    MEMBERSHIP\_TIERS next\_t ON next\_t.tier\_id = mt.tier\_id + 1

WHERE

    c.customer\_id = 1001;

PROMPT =====  Use Function FN\_CALC\_UPGRADE\_VALUE =====

*-- Current Status: Customer 1001 starts in the Bronze tier with 300 points and needs 200 more to reach Silver.*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('=== TESTING UPGRADE CALCULATION ===');

    DBMS\_OUTPUT.PUT\_LINE(FN\_CALC\_UPGRADE\_VALUE(1001));

END;

/

PROMPT =====  Use Function FN\_CHECK\_MEMBERSHIP\_HEALTH =====

*-- Current Status: Membership is Active, Bronze membership: Active (expires in 365 days)*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('=== TESTING MEMBERSHIP HEALTH ===');

    DBMS\_OUTPUT.PUT\_LINE(FN\_CHECK\_MEMBERSHIP\_HEALTH(1001));

END;

/

*-- Hence, we can continue to Add Point and upgrade the Tier*

*-- If the tier didn't upgrade automatically, we need to:*

*-- 1. Verify the trigger is working*

*-- 2. Manually update if needed*

PROMPT =====  Trigger Status =====

*-- First, let's check if the trigger is enabled and working*

SELECT trigger\_name, status

FROM user\_triggers

WHERE trigger\_name = 'TRG\_SYNC\_POINT\_BALANCE';

*-- If the trigger is disabled, enable it*

ALTER TRIGGER trg\_sync\_point\_balance ENABLE;

PROMPT =====  Add Point =====

*-- Now let's simulate the point addition again with triggers enabled*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('Adding 500 points to customer 1001 with triggers enabled...');

    INSERT INTO POINT\_TRANSACTIONS

      (customer\_id, points\_amount, transaction\_type, description)

    VALUES

      (1001, 500, 'PURCHASE', 'Test points addition - with triggers');

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Successfully added 500 points to customer 1001');

EXCEPTION

    WHEN OTHERS THEN

        DBMS\_OUTPUT.PUT\_LINE('Error adding points: ' || SQLERRM);

        ROLLBACK;

END;

/

PROMPT =====  Checked Customer Status - Tier upgraded =====

*-- Verify the upgrade occurred*

SELECT

    c.customer\_id,

    c.name,

    c.points\_balance,

    mt.tier\_name AS current\_tier,

    (SELECT points\_required FROM MEMBERSHIP\_TIERS WHERE tier\_id = mt.tier\_id + 1) - c.points\_balance AS points\_needed\_for\_next\_tier

FROM

    CUSTOMERS c

JOIN

    MEMBERSHIP\_TIERS mt ON c.tier\_id = mt.tier\_id

WHERE

    customer\_id = 1001;

*/\**

*=====  Checked Customer Status - Tier upgraded =====*

*CUSTOMER\_ID NAME                                                                                             POINTS\_BALANCE CURRENT\_TIER          POINTS\_NEEDED\_FOR\_NEXT\_TIER*

*----------- ---------------------------------------------------------------------------------------------------- -------------- -------------------- ---------------------------*

*1001 John Smith                                                                                          800 Silver                                        700*

*\*/*

PROMPT =====   Membership History Logs =====

*-- Check the membership history for the upgrade*

SELECT mh.change\_date, mh.change\_reason,

       old\_t.tier\_name AS old\_tier,

       new\_t.tier\_name AS new\_tier

FROM MEMBERSHIP\_HISTORY mh

JOIN MEMBERSHIP\_TIERS old\_t ON mh.old\_tier\_id = old\_t.tier\_id

JOIN MEMBERSHIP\_TIERS new\_t ON mh.new\_tier\_id = new\_t.tier\_id

WHERE mh.customer\_id = 1001

ORDER BY mh.change\_date DESC;

*/\*=====   Membership History Logs =====*

*CHANGE\_DATE                                                                 CHANGE\_REASON                    OLD\_TIER*

*--------------------------------------------------------------------------- ---------------------------------------------------------------------------------------------------- --------------------*

*NEW\_TIER*

*--------------------*

*28-APR-25 07.20.20.883000 PM                                                UPGRADE                          Bronze*

*Silver*

*\*/*

*-- If the automatic upgrade still didn't occur, we can manually update the tier*

DECLARE

    v\_new\_tier\_id NUMBER;

    v\_old\_tier\_id NUMBER;

BEGIN

*-- Get the current tier first*

    SELECT tier\_id INTO v\_old\_tier\_id

    FROM CUSTOMERS

    WHERE customer\_id = 1001;

*-- Find the appropriate tier for the customer's current points*

    SELECT MAX(tier\_id) INTO v\_new\_tier\_id

    FROM MEMBERSHIP\_TIERS

    WHERE points\_required <= (SELECT points\_balance FROM CUSTOMERS WHERE customer\_id = 1001);

*-- Only proceed if there's actually a tier change*

    IF v\_new\_tier\_id != v\_old\_tier\_id THEN

*-- Update the customer's tier*

        UPDATE CUSTOMERS

        SET tier\_id = v\_new\_tier\_id

        WHERE customer\_id = 1001;

*-- Log the tier change*

        INSERT INTO MEMBERSHIP\_HISTORY (

            customer\_id,

            old\_tier\_id,

            new\_tier\_id,

            change\_date,

            change\_reason

        )

        VALUES (

            1001,

            v\_old\_tier\_id,

            v\_new\_tier\_id,

            SYSDATE,

            'MANUAL UPGRADE'

        );

        COMMIT;

        DBMS\_OUTPUT.PUT\_LINE('Manually updated customer 1001 from tier ' ||

                            v\_old\_tier\_id || ' to tier ' || v\_new\_tier\_id);

    ELSE

        DBMS\_OUTPUT.PUT\_LINE('No Manully Updated tier needed - customer already at appropriate tier, Automatically Upgraded Tier Successfully.');

    END IF;

EXCEPTION

    WHEN OTHERS THEN

        DBMS\_OUTPUT.PUT\_LINE('Error in manual tier update: ' || SQLERRM);

        ROLLBACK;

END;

/

*-- output:No Manually Updated tier needed - customer already at appropriate tier, Automatically Upgraded Tier Successfully.*

PROMPT => TRIGGER trg\_membership\_history WORKING <=

*-- Final verification*

SELECT

    c.customer\_id,

    c.name,

    c.points\_balance,

    mt.tier\_name AS current\_tier

FROM

    CUSTOMERS c

JOIN

    MEMBERSHIP\_TIERS mt ON c.tier\_id = mt.tier\_id

WHERE

    customer\_id = 1001;

*/\*output:*

*CUSTOMER\_ID NAME                                                                                             POINTS\_BALANCE CURRENT\_TIER*

*----------- ---------------------------------------------------------------------------------------------------- -------------- --------------------*

*1001 John Smith                                                                                          800 Silver*

*\*/*

*/\**

*--------------------------------------------------------------------------------*

*-- TEST POINT EARNING ERROR CASES*

*--------------------------------------------------------------------------------*

*-- Testing invalid point additions and trigger behavior*

*BEGIN*

*DBMS\_OUTPUT.PUT\_LINE('=== TESTING NEGATIVE POINT ADDITION ===');*

*BEGIN*

*INSERT INTO POINT\_TRANSACTIONS*

*(customer\_id, points\_amount, transaction\_type, description)*

*VALUES*

*(1001, -100, 'PURCHASE', 'Invalid negative points');*

*DBMS\_OUTPUT.PUT\_LINE('ERROR: Should not reach here with negative points');*

*ROLLBACK;*

*EXCEPTION*

*WHEN OTHERS THEN*

*DBMS\_OUTPUT.PUT\_LINE('Expected error: ' || SUBSTR(SQLERRM, 1, 200));*

*ROLLBACK;*

*END;*

*DBMS\_OUTPUT.PUT\_LINE('=== TESTING INVALID TRANSACTION TYPE ===');*

*BEGIN*

*INSERT INTO POINT\_TRANSACTIONS*

*(customer\_id, points\_amount, transaction\_type, description)*

*VALUES*

*(1001, 100, 'INVALID\_TYPE', 'Invalid transaction type');*

*DBMS\_OUTPUT.PUT\_LINE('ERROR: Should not reach here with invalid type');*

*ROLLBACK;*

*EXCEPTION*

*WHEN OTHERS THEN*

*DBMS\_OUTPUT.PUT\_LINE('Expected error: ' || SUBSTR(SQLERRM, 1, 200));*

*ROLLBACK;*

*END;*

*DBMS\_OUTPUT.PUT\_LINE('=== TESTING NON-EXISTENT CUSTOMER ===');*

*BEGIN*

*INSERT INTO POINT\_TRANSACTIONS*

*(customer\_id, points\_amount, transaction\_type, description)*

*VALUES*

*(9999, 100, 'PURCHASE', 'Non-existent customer');*

*DBMS\_OUTPUT.PUT\_LINE('ERROR: Should not reach here with invalid customer');*

*ROLLBACK;*

*EXCEPTION*

*WHEN OTHERS THEN*

*DBMS\_OUTPUT.PUT\_LINE('Expected error: ' || SUBSTR(SQLERRM, 1, 200));*

*ROLLBACK;*

*END;*

*END;*

*/*

*/\*expected output: === TESTING NEGATIVE POINT ADDITION ===*

*Expected error: ORA-02290: check constraint (SYSTEM.CHK\_POINTS\_AMOUNT) violated*

*=== TESTING INVALID TRANSACTION TYPE ===*

*Expected error: ORA-02290: check constraint (SYSTEM.SYS\_C0026292) violated*

*=== TESTING NON-EXISTENT CUSTOMER ===*

*Expected error: ORA-02291: integrity constraint (SYSTEM.FK\_PT\_CUSTOMER) violated - parent key not found\*/*

*\*/*

PROMPT ===== END TEST CASE 1 =====

*--------------------------------------------------------------------------------*

*-- FINAL SECTION: SCRIPT COMPLETION REMARKS*

*--------------------------------------------------------------------------------*

*-- This script demonstrates the key components of a Membership Management System,*

*-- including procedures for processing renewals and point expirations, functions*

*-- for upgrade evaluation and health checks, and queries for tier analysis and*

*-- status verification.*

*-- All modules have been tested through realistic demo scenarios. Additional tests*

*-- or enhancements may be added depending on future requirements, such as user-facing*

*-- interfaces or monthly automation jobs.*

*--------------------------------------------------------------------------------*

*-- END OF SCRIPT*

*--------------------------------------------------------------------------------*

## Lee Hien Leong - Personal Script 2

*/\**

*GROUP NUMBER: G002*

*PROGRAMME: CS*

*STUDENT ID: 22ACB04958*

*STUDENT NAME: LEE HIEN LEONG*

*Submission date and time (DD-MON-YY):  29 April 2025*

*\*/*

*-- Every Error Test Case is checked and place commented.*

*-- Pre-Setup*

SET SERVEROUTPUT ON

SET LINESIZE 500;

SET PAGESIZE 500;

*-- Check if trigger exists before trying to disable it*

BEGIN

  EXECUTE IMMEDIATE 'BEGIN

    EXECUTE IMMEDIATE ''ALTER TRIGGER trg\_order\_totals DISABLE'';

  EXCEPTION

    WHEN OTHERS THEN

      IF SQLCODE != -4080 THEN -- ORA-04080: trigger does not exist

        RAISE;

      END IF;

  END;';

END;

/

*-- TEST DATA SETUP*

PROMPT ===== PREPARING TEST DATA =====

*-- Make promotions available to all tiers for testing*

UPDATE PROMOTIONS

SET applicable\_tier\_id = NULL

WHERE promotion\_id IN (1, 3);

*-- Clear existing assignments to test bulk assignment*

DELETE FROM CUSTOMER\_PROMOTIONS

WHERE promotion\_id IN (1, 3)

AND acquisition\_method = 'AUTO\_TIER';

COMMIT;

*--------------------------------------------------------------------------------*

*-- 1. QUERY: List All Assignable Vouchers*

*--------------------------------------------------------------------------------*

*-- User Transaction:*

*-- Displays all vouchers that can be automatically assigned to customers,*

*-- including birthday vouchers and auto-assign promotions.*

*-- Purpose:*

*-- To provide staff with visibility of available vouchers for bulk assignment*

*-- and to verify which promotions are currently active.*

*-- Output Columns:*

*-- promotion\_id      - Unique identifier for the promotion*

*-- name              - Name/description of the voucher*

*-- voucher\_code      - Code customers use to redeem*

*-- assignment\_type   - How voucher is assigned (BIRTHDAY/AUTO\_ASSIGN/MANUAL)*

*-- valid\_from/to     - Date range when voucher is valid*

*-- applicable\_tier   - Which membership tier can use this voucher*

*-- birthday\_bonus\_points - Points awarded for birthday vouchers*

*--------------------------------------------------------------------------------*

PROMPT ===== ASSIGNABLE VOUCHERS =====

SELECT

    p.promotion\_id,

    p.name,

    p.voucher\_code,

    CASE

        WHEN p.name LIKE 'BDAY%' THEN 'BIRTHDAY'

        WHEN p.is\_auto\_assign = 'Y' THEN 'AUTO\_ASSIGN'

        ELSE 'MANUAL'

    END AS assignment\_type,

    p.valid\_from,

    p.valid\_to,

    mt.tier\_name AS applicable\_tier,

    mt.birthday\_bonus\_points

FROM

    PROMOTIONS p

LEFT JOIN

    MEMBERSHIP\_TIERS mt ON p.applicable\_tier\_id = mt.tier\_id

WHERE

    (p.is\_auto\_assign = 'Y' OR p.name LIKE 'BDAY%')

    AND SYSDATE BETWEEN p.valid\_from AND p.valid\_to

ORDER BY

    assignment\_type, p.name;

*--------------------------------------------------------------------------------*

*-- DEMO TEST (NORMAL CASE): LIST ASSIGNABLE VOUCHERS*

*--------------------------------------------------------------------------------*

*-- Expected Output:*

*-- - List of all currently valid vouchers that can be auto-assigned*

*-- - Includes both birthday vouchers and regular auto-assign promotions*

*-- - Shows validity period and applicable tiers*

*--------------------------------------------------------------------------------*

*--------------------------------------------------------------------------------*

*-- 2. QUERY: Voucher Assignment Status*

*--------------------------------------------------------------------------------*

*-- User Transaction:*

*-- Shows current statistics about voucher assignments including how many have been*

*-- assigned, redeemed, and when assignments occurred.*

*-- Purpose:*

*-- To monitor the effectiveness of voucher campaigns and track redemption rates*

*-- for reporting and analysis.*

*-- Output Columns:*

*-- promotion\_id    - Unique identifier for the promotion*

*-- name            - Name/description of the voucher*

*-- voucher\_code    - Code customers use to redeem*

*-- voucher\_type    - BIRTHDAY or AUTO\_ASSIGN*

*-- assigned\_count  - How many customers received this voucher*

*-- redeemed\_count  - How many have been used*

*-- first/last\_assignment - Date range of assignments*

*--------------------------------------------------------------------------------*

PROMPT ===== VOUCHER ASSIGNMENT STATUS =====

SELECT

    p.promotion\_id,

    p.name,

    p.voucher\_code,

    CASE

        WHEN p.name LIKE 'BDAY%' THEN 'BIRTHDAY'

        ELSE 'AUTO\_ASSIGN'

    END AS voucher\_type,

    COUNT(cp.customer\_id) AS assigned\_count,

    COUNT(CASE WHEN cp.is\_used = 'Y' THEN 1 END) AS redeemed\_count,

    MIN(cp.date\_acquired) AS first\_assignment,

    MAX(cp.date\_acquired) AS last\_assignment

FROM

    PROMOTIONS p

LEFT JOIN

    CUSTOMER\_PROMOTIONS cp ON p.promotion\_id = cp.promotion\_id

WHERE

    (p.is\_auto\_assign = 'Y' OR p.name LIKE 'BDAY%')

GROUP BY

    p.promotion\_id, p.name, p.voucher\_code

ORDER BY

    p.name;

*--------------------------------------------------------------------------------*

*-- DEMO TEST (NORMAL CASE): VOUCHER ASSIGNMENT STATUS*

*--------------------------------------------------------------------------------*

*-- Expected Output:*

*-- - Summary statistics for each voucher type*

*-- - Shows assignment and redemption volumes*

*-- - Indicates time period of assignments*

*--------------------------------------------------------------------------------*

*--------------------------------------------------------------------------------*

*-- 3. PROCEDURE 1: PROC\_BULK\_ASSIGN\_VOUCHERS*

*--------------------------------------------------------------------------------*

*-- User Transaction:*

*-- Automatically assigns eligible vouchers to customers based on their membership tier*

*-- or to all customers if no tier is specified. Handles both auto-assign promotions*

*-- and birthday vouchers in a single batch operation.*

*-- Purpose:*

*-- To efficiently distribute promotional vouchers to qualified customers without*

*-- manual intervention, improving customer engagement and loyalty program benefits.*

*-- Input Arguments:*

*-- p\_voucher\_type  VARCHAR2  type of vouchers to assign: 'AUTO', 'BIRTHDAY', or 'ALL'*

*-- p\_tier\_id       NUMBER    optional tier filter to restrict assignments*

*-- p\_debug\_mode    BOOLEAN   when TRUE, outputs detailed processing information*

*--------------------------------------------------------------------------------*

PROMPT ===== CREATING PROCEDURE =====

CREATE OR REPLACE PROCEDURE PROC\_BULK\_ASSIGN\_VOUCHERS(

    p\_voucher\_type IN VARCHAR2 DEFAULT 'ALL',

    p\_tier\_id IN NUMBER DEFAULT NULL,

    p\_debug\_mode IN BOOLEAN DEFAULT TRUE

) AS

    CURSOR c\_customers IS

        SELECT

            c.customer\_id,

            c.name,

            c.tier\_id,

            mt.tier\_name

        FROM CUSTOMERS c

        JOIN MEMBERSHIP\_TIERS mt ON c.tier\_id = mt.tier\_id

        WHERE c.is\_member = 'Y'

        AND (p\_tier\_id IS NULL OR c.tier\_id = p\_tier\_id);

    v\_new\_assignments NUMBER := 0;

    v\_eligible\_promos NUMBER := 0;

BEGIN

    IF p\_debug\_mode THEN

        DBMS\_OUTPUT.PUT\_LINE('=== BULK VOUCHER ASSIGNMENT STARTED ===');

        DBMS\_OUTPUT.PUT\_LINE('Parameters: Type='||p\_voucher\_type||', Tier='||

                            NVL(TO\_CHAR(p\_tier\_id),'ALL')||', Debug='||

                            CASE WHEN p\_debug\_mode THEN 'ON' ELSE 'OFF' END);

    END IF;

*-- Count eligible promotions*

    SELECT COUNT(\*) INTO v\_eligible\_promos

    FROM PROMOTIONS

    WHERE is\_auto\_assign = 'Y'

    AND SYSDATE BETWEEN valid\_from AND valid\_to;

    IF p\_debug\_mode THEN

        DBMS\_OUTPUT.PUT\_LINE('Found '||v\_eligible\_promos||' eligible promotions');

    END IF;

    FOR cust IN c\_customers LOOP

        IF p\_debug\_mode THEN

            DBMS\_OUTPUT.PUT\_LINE('Processing customer '||cust.customer\_id||

                                ': '||cust.name||' (Tier: '||cust.tier\_name||')');

        END IF;

*-- Process auto-assign vouchers*

        IF p\_voucher\_type IN ('AUTO', 'ALL') THEN

            FOR v\_rec IN (

                SELECT

                    p.promotion\_id,

                    p.name AS promo\_name,

                    p.voucher\_code,

                    p.applicable\_tier\_id

                FROM PROMOTIONS p

                WHERE p.is\_auto\_assign = 'Y'

                AND (p.applicable\_tier\_id IS NULL OR p.applicable\_tier\_id = cust.tier\_id)

                AND SYSDATE BETWEEN p.valid\_from AND p.valid\_to

                AND NOT EXISTS (

                    SELECT 1 FROM CUSTOMER\_PROMOTIONS cp

                    WHERE cp.customer\_id = cust.customer\_id

                    AND cp.promotion\_id = p.promotion\_id

                )

            ) LOOP

                BEGIN

                    INSERT INTO CUSTOMER\_PROMOTIONS (

                        customer\_id, promotion\_id, acquisition\_method, date\_acquired

                    ) VALUES (

                        cust.customer\_id, v\_rec.promotion\_id, 'AUTO\_TIER', SYSDATE

                    );

                    v\_new\_assignments := v\_new\_assignments + 1;

                    IF p\_debug\_mode THEN

                        DBMS\_OUTPUT.PUT\_LINE(' - Assigned '||v\_rec.promo\_name||

                                            ' (ID:'||v\_rec.promotion\_id||')');

                    END IF;

                EXCEPTION

                    WHEN DUP\_VAL\_ON\_INDEX THEN

                        IF p\_debug\_mode THEN

                            DBMS\_OUTPUT.PUT\_LINE(' - Already assigned: '||v\_rec.promo\_name);

                        END IF;

                    WHEN OTHERS THEN

                        IF p\_debug\_mode THEN

                            DBMS\_OUTPUT.PUT\_LINE(' ! Error assigning '||v\_rec.promo\_name||

                                                ': '||SQLERRM);

                        END IF;

                END;

            END LOOP;

        END IF;

    END LOOP;

*-- Audit log the batch assignment result*

    LOG\_ACTIVITY(

      NULL,  *-- batch operation, no single customer*

      'VOUCHER\_ASSIGNMENT',

      'Total new assignments: ' || v\_new\_assignments,

     SYS\_CONTEXT('USERENV','IP\_ADDRESS')

    );

    COMMIT;

    IF p\_debug\_mode THEN

        DBMS\_OUTPUT.PUT\_LINE('=== COMPLETED ===');

        DBMS\_OUTPUT.PUT\_LINE('Total new assignments: '||v\_new\_assignments);

    END IF;

EXCEPTION

    WHEN OTHERS THEN

        ROLLBACK;

        DBMS\_OUTPUT.PUT\_LINE('!!! PROCEDURE FAILED: '||SQLERRM);

*-- Auditlog the failure*

        LOG\_ACTIVITY(

          NULL,          'VOUCHER\_ASSIGNMENT',

         'Procedure failed: ' || SUBSTR(SQLERRM,1,200),

         SYS\_CONTEXT('USERENV','IP\_ADDRESS')

       );

       DBMS\_OUTPUT.PUT\_LINE('!!! PROCEDURE FAILED: '||SQLERRM);

        RAISE;

END;

/

*-----------------------------------------*

*-- UNIT TESTS FOR PROC\_BULK\_ASSIGN\_VOUCHERS*

*-----------------------------------------*

PROMPT === UNIT TESTS: PROC\_BULK\_ASSIGN\_VOUCHERS ===

*-- Test 1: Assign to all tiers (default parameters)*

PROMPT === TEST: ASSIGN TO ALL TIERS ===

DECLARE

    v\_before\_count NUMBER;

    v\_after\_count NUMBER;

BEGIN

    SELECT COUNT(\*) INTO v\_before\_count FROM CUSTOMER\_PROMOTIONS;

    PROC\_BULK\_ASSIGN\_VOUCHERS(p\_debug\_mode => TRUE);

    SELECT COUNT(\*) INTO v\_after\_count FROM CUSTOMER\_PROMOTIONS;

    DBMS\_OUTPUT.PUT\_LINE('Assignments before: ' || v\_before\_count);

    DBMS\_OUTPUT.PUT\_LINE('Assignments after: ' || v\_after\_count);

    DBMS\_OUTPUT.PUT\_LINE('New assignments: ' || (v\_after\_count - v\_before\_count));

END;

/

*-- Test 2: Assign to specific tier only*

PROMPT === TEST: ASSIGN TO SPECIFIC TIER ===

DECLARE

    v\_tier\_id NUMBER := 1; *-- Bronze tier*

    v\_before\_count NUMBER;

    v\_after\_count NUMBER;

BEGIN

    SELECT COUNT(\*) INTO v\_before\_count

    FROM CUSTOMER\_PROMOTIONS cp

    JOIN CUSTOMERS c ON cp.customer\_id = c.customer\_id

    WHERE c.tier\_id = v\_tier\_id;

    PROC\_BULK\_ASSIGN\_VOUCHERS(p\_tier\_id => v\_tier\_id, p\_debug\_mode => TRUE);

    SELECT COUNT(\*) INTO v\_after\_count

    FROM CUSTOMER\_PROMOTIONS cp

    JOIN CUSTOMERS c ON cp.customer\_id = c.customer\_id

    WHERE c.tier\_id = v\_tier\_id;

    DBMS\_OUTPUT.PUT\_LINE('Bronze tier assignments before: ' || v\_before\_count);

    DBMS\_OUTPUT.PUT\_LINE('Bronze tier assignments after: ' || v\_after\_count);

END;

/

*/\* ERROR TEST*

*-- Test 3: Test debug mode off*

*PROMPT === TEST: DEBUG MODE OFF ===*

*BEGIN*

*PROC\_BULK\_ASSIGN\_VOUCHERS(p\_debug\_mode => FALSE);*

*DBMS\_OUTPUT.PUT\_LINE('Executed with debug off - verify no debug output was shown');*

*END;*

*/ \*/*

*-- Test 4: Test assignment type filter*

PROMPT === TEST: AUTO ASSIGN ONLY ===

DECLARE

    v\_before\_count NUMBER;

    v\_after\_count NUMBER;

BEGIN

    SELECT COUNT(\*) INTO v\_before\_count FROM CUSTOMER\_PROMOTIONS;

    PROC\_BULK\_ASSIGN\_VOUCHERS(p\_voucher\_type => 'AUTO', p\_debug\_mode => TRUE);

    SELECT COUNT(\*) INTO v\_after\_count FROM CUSTOMER\_PROMOTIONS;

    DBMS\_OUTPUT.PUT\_LINE('New auto-assign vouchers: ' || (v\_after\_count - v\_before\_count));

END;

/

*-------------------------------------------------------------------------------*

*-- 4. PROCEDURE 2: PROC\_REDEEM\_VOUCHER*

*--------------------------------------------------------------------------------*

*-- User Transaction:*

*-- Processes the redemption of a voucher by a customer, applying discounts to orders*

*-- or deducting points as needed. Validates all redemption requirements before processing.*

*-- Purpose:*

*-- To provide a secure and auditable way for customers to use their vouchers,*

*-- ensuring all business rules are enforced during redemption.*

*-- Input Arguments:*

*-- p\_customer\_id  NUMBER    ID of the customer redeeming the voucher*

*-- p\_voucher\_id   NUMBER    ID of the voucher being redeemed*

*-- p\_order\_id     NUMBER    optional order ID for order-based discounts*

*--------------------------------------------------------------------------------*

CREATE OR REPLACE PROCEDURE PROC\_REDEEM\_VOUCHER(

    p\_customer\_id IN NUMBER,

    p\_voucher\_id IN NUMBER,

    p\_order\_id IN NUMBER DEFAULT NULL

) AS

    v\_discount\_amount NUMBER;

    v\_min\_spend NUMBER;

    v\_order\_total NUMBER := 0;

    v\_points\_required NUMBER;

    v\_customer\_points NUMBER;

    v\_promo\_type VARCHAR2(20);

BEGIN

*-- Get promotion details*

    SELECT p.discount\_value, p.min\_spend, p.points\_required, p.promotion\_type

    INTO v\_discount\_amount, v\_min\_spend, v\_points\_required, v\_promo\_type

    FROM SYSTEM.PROMOTIONS p

    WHERE p.promotion\_id = p\_voucher\_id;

*-- Check if order meets minimum spend (if applicable)*

    IF p\_order\_id IS NOT NULL AND v\_min\_spend IS NOT NULL THEN

        SELECT o.total\_amount INTO v\_order\_total

        FROM SYSTEM.ORDERS o

        WHERE o.order\_id = p\_order\_id;

        IF v\_order\_total < v\_min\_spend THEN

*-- Log error if order does not meet minimum spend*

            LOG\_ACTIVITY(

                p\_customer\_id,

                'VOUCHER\_REDEMPTION',

                'Order ID ' || p\_order\_id || ' does not meet the minimum spend requirement',

                SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

            );

            RAISE\_APPLICATION\_ERROR(-20001, 'Order does not meet minimum spend requirement');

        END IF;

    END IF;

*-- Check points balance if this is a points redemption*

    IF v\_points\_required IS NOT NULL THEN

        SELECT c.points\_balance INTO v\_customer\_points

        FROM SYSTEM.CUSTOMERS c

        WHERE c.customer\_id = p\_customer\_id;

        IF v\_customer\_points < v\_points\_required THEN

*-- Log error if insufficient points for redemption*

            LOG\_ACTIVITY(

                p\_customer\_id,

                'VOUCHER\_REDEMPTION',

                'Insufficient points for redemption. Needed ' || v\_points\_required || ', found ' || v\_customer\_points,

                SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

            );

            RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient points for redemption');

        END IF;

    END IF;

*-- Process the redemption*

    INSERT INTO SYSTEM.REDEMPTIONS (

        customer\_id,

        promotion\_id,

        order\_id,

        points\_used,

        redemption\_date,

        redemption\_status

    ) VALUES (

        p\_customer\_id,

        p\_voucher\_id,

        p\_order\_id,

        NVL(v\_points\_required, 0),

        SYSTIMESTAMP,

        'COMPLETED'

    );

*-- Update order discount if applicable*

    IF p\_order\_id IS NOT NULL AND v\_discount\_amount IS NOT NULL THEN

        UPDATE SYSTEM.ORDERS o

        SET o.discount\_amount = o.discount\_amount + v\_discount\_amount,

            o.final\_amount = o.total\_amount - (o.discount\_amount + v\_discount\_amount)

        WHERE o.order\_id = p\_order\_id;

    END IF;

*-- Deduct points if this was a points redemption*

    IF v\_points\_required IS NOT NULL THEN

        INSERT INTO SYSTEM.POINT\_TRANSACTIONS (

            customer\_id,

            points\_amount,

            transaction\_type,

            description

        ) VALUES (

            p\_customer\_id,

            -v\_points\_required,

            'REDEMPTION',

            'Voucher redemption'

        );

    END IF;

*-- Log successful voucher redemption*

    LOG\_ACTIVITY(

        p\_customer\_id,

        'VOUCHER\_REDEMPTION',

        'Customer ID ' || p\_customer\_id || ' redeemed voucher ID ' || p\_voucher\_id,

        SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

    );

    COMMIT;

EXCEPTION

    WHEN OTHERS THEN

*-- Log failure on any error*

        LOG\_ACTIVITY(

            p\_customer\_id,

            'VOUCHER\_REDEMPTION',

            'Error during redemption for customer ID ' || p\_customer\_id || ': ' || SQLERRM,

            SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

        );

        ROLLBACK;

        RAISE;

END;

/

*-----------------------------------------*

*-- UNIT TESTS FOR PROC\_REDEEM\_VOUCHER*

*-----------------------------------------*

ALTER PROCEDURE SYSTEM.PROC\_REDEEM\_VOUCHER COMPILE;

SHOW ERRORS PROCEDURE SYSTEM.PROC\_REDEEM\_VOUCHER;

PROMPT === UNIT TESTS: PROC\_REDEEM\_VOUCHER ===

*-- Before testing PROC\_REDEEM\_VOUCHER, update customer points*

UPDATE CUSTOMERS SET points\_balance = 1000 WHERE customer\_id = 1001;

UPDATE ORDERS SET total\_amount = 50 WHERE order\_id = 5001;

*-- Test 1: Basic redemption without order*

PROMPT === TEST: BASIC REDEMPTION ===

DECLARE

    v\_test\_customer NUMBER := 1001;

    v\_test\_voucher NUMBER := 1; *-- 10% Off voucher*

    v\_before\_status CHAR(1);

    v\_after\_status CHAR(1);

BEGIN

*-- Reset test data*

    UPDATE CUSTOMER\_PROMOTIONS SET is\_used = 'N'

    WHERE customer\_id = v\_test\_customer AND promotion\_id = v\_test\_voucher;

    COMMIT;

*-- Get before status*

    SELECT is\_used INTO v\_before\_status

    FROM CUSTOMER\_PROMOTIONS

    WHERE customer\_id = v\_test\_customer AND promotion\_id = v\_test\_voucher;

*-- Execute with schema prefix*

    PROC\_REDEEM\_VOUCHER(v\_test\_customer, v\_test\_voucher);

*-- Verify*

    SELECT is\_used INTO v\_after\_status

    FROM CUSTOMER\_PROMOTIONS

    WHERE customer\_id = v\_test\_customer AND promotion\_id = v\_test\_voucher;

    DBMS\_OUTPUT.PUT\_LINE('Status changed from ' || v\_before\_status || ' to ' || v\_after\_status);

END;

/

*-- Test 2: Redemption with order*

PROMPT === TEST: REDEMPTION WITH ORDER ===

DECLARE

    v\_test\_customer NUMBER := 1001;

    v\_test\_voucher NUMBER := 1;

    v\_test\_order NUMBER := 5001;

BEGIN

*-- Reset test data*

    UPDATE CUSTOMER\_PROMOTIONS SET is\_used = 'N'

    WHERE customer\_id = v\_test\_customer AND promotion\_id = v\_test\_voucher;

    COMMIT;

*-- Execute*

    PROC\_REDEEM\_VOUCHER(v\_test\_customer, v\_test\_voucher, v\_test\_order);

*-- Verify redemption record*

    FOR r IN (

        SELECT \* FROM REDEMPTIONS

        WHERE customer\_id = v\_test\_customer

        AND promotion\_id = v\_test\_voucher

        AND order\_id = v\_test\_order

    ) LOOP

        DBMS\_OUTPUT.PUT\_LINE('Redemption ID ' || r.redemption\_id || ' created with order reference');

    END LOOP;

END;

/

*-- For the Points Redemption Test:*

*-- Test 3: Points-based redemption*

UPDATE PROMOTIONS

SET points\_required = 100  *-- Or whatever point cost you want*

WHERE promotion\_id = 2;

PROMPT === TEST: POINTS REDEMPTION ===

DECLARE

    v\_test\_customer NUMBER := 1001;

    v\_test\_voucher NUMBER := 2; *-- Voucher requiring points*

    v\_before\_points NUMBER;

    v\_after\_points NUMBER;

BEGIN

*-- Setup - ensure sufficient points*

    UPDATE CUSTOMERS SET points\_balance = 1000 WHERE customer\_id = v\_test\_customer;

    UPDATE CUSTOMER\_PROMOTIONS SET is\_used = 'N'

    WHERE customer\_id = v\_test\_customer AND promotion\_id = v\_test\_voucher;

    COMMIT;

*-- Get before points*

    SELECT points\_balance INTO v\_before\_points FROM CUSTOMERS

    WHERE customer\_id = v\_test\_customer;

*-- Execute*

    PROC\_REDEEM\_VOUCHER(v\_test\_customer, v\_test\_voucher);

*-- Get after points*

    SELECT points\_balance INTO v\_after\_points FROM CUSTOMERS

    WHERE customer\_id = v\_test\_customer;

    DBMS\_OUTPUT.PUT\_LINE('Points changed from ' || v\_before\_points || ' to ' || v\_after\_points);

EXCEPTION

    WHEN OTHERS THEN

        DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

        ROLLBACK;

END;

/

*-- Test 5: Insufficient points for redemption*

PROMPT === TEST: INSUFFICIENT POINTS ===

DECLARE

    v\_test\_customer NUMBER := 1001;

    v\_test\_voucher NUMBER := 2; *-- Voucher requiring points*

    v\_before\_points NUMBER;

    v\_error\_msg VARCHAR2(4000);

BEGIN

*-- Setup - ensure insufficient points*

    UPDATE CUSTOMERS SET points\_balance = 50 WHERE customer\_id = v\_test\_customer;

    UPDATE CUSTOMER\_PROMOTIONS SET is\_used = 'N'

    WHERE customer\_id = v\_test\_customer AND promotion\_id = v\_test\_voucher;

    COMMIT;

*-- Get before points*

    SELECT points\_balance INTO v\_before\_points FROM CUSTOMERS

    WHERE customer\_id = v\_test\_customer;

*-- Execute (should fail)*

    BEGIN

        PROC\_REDEEM\_VOUCHER(v\_test\_customer, v\_test\_voucher);

        DBMS\_OUTPUT.PUT\_LINE('ERROR: Should have failed for insufficient points');

    EXCEPTION

        WHEN OTHERS THEN

            v\_error\_msg := SQLERRM;

            IF v\_error\_msg LIKE '%20002%' THEN

                DBMS\_OUTPUT.PUT\_LINE('SUCCESS: Failed as expected with: ' || v\_error\_msg);

            ELSE

                DBMS\_OUTPUT.PUT\_LINE('ERROR: Unexpected error: ' || v\_error\_msg);

            END IF;

    END;

*-- Verify points weren't deducted*

    DECLARE

        v\_after\_points NUMBER;

    BEGIN

        SELECT points\_balance INTO v\_after\_points FROM CUSTOMERS

        WHERE customer\_id = v\_test\_customer;

        IF v\_after\_points = v\_before\_points THEN

            DBMS\_OUTPUT.PUT\_LINE('SUCCESS: Points balance unchanged (' || v\_after\_points || ')');

        ELSE

            DBMS\_OUTPUT.PUT\_LINE('ERROR: Points were deducted despite failure');

        END IF;

    END;

EXCEPTION

    WHEN OTHERS THEN

        DBMS\_OUTPUT.PUT\_LINE('Error in test: ' || SQLERRM);

        ROLLBACK;

END;

/

*-- For the Minimum Spend Test:*

*-- Test 4: Minimum spend requirement failure*

PROMPT === TEST: MINIMUM SPEND FAILURE ===

DECLARE

    v\_test\_customer NUMBER := 1001;

    v\_test\_voucher NUMBER := 5; *-- Voucher with minimum spend*

    v\_test\_order NUMBER := 5001;

    v\_error\_msg VARCHAR2(4000);

    v\_original\_amount NUMBER;

BEGIN

*-- Store original amount*

    SELECT total\_amount INTO v\_original\_amount FROM ORDERS WHERE order\_id = v\_test\_order;

*-- Temporarily disable constraint for testing*

    EXECUTE IMMEDIATE 'ALTER TABLE ORDERS DISABLE CONSTRAINT CHK\_ORDER\_AMOUNTS';

*-- Setup order with low amount*

    UPDATE ORDERS SET total\_amount = 10, final\_amount = 10 WHERE order\_id = v\_test\_order;

    UPDATE CUSTOMER\_PROMOTIONS SET is\_used = 'N'

    WHERE customer\_id = v\_test\_customer AND promotion\_id = v\_test\_voucher;

    COMMIT;

*-- Execute (should fail)*

    BEGIN

        PROC\_REDEEM\_VOUCHER(v\_test\_customer, v\_test\_voucher, v\_test\_order);

        DBMS\_OUTPUT.PUT\_LINE('ERROR: Should have failed minimum spend');

    EXCEPTION

        WHEN OTHERS THEN

            v\_error\_msg := SQLERRM;

            DBMS\_OUTPUT.PUT\_LINE('SUCCESS: Failed as expected with: ' || v\_error\_msg);

    END;

*-- Restore original data*

    UPDATE ORDERS

    SET total\_amount = v\_original\_amount,

        final\_amount = v\_original\_amount - discount\_amount

    WHERE order\_id = v\_test\_order;

*-- Re-enable constraint*

    EXECUTE IMMEDIATE 'ALTER TABLE ORDERS ENABLE CONSTRAINT CHK\_ORDER\_AMOUNTS';

    COMMIT;

EXCEPTION

    WHEN OTHERS THEN

*-- Ensure constraint is re-enabled and data restored even if test fails*

        ROLLBACK;

        BEGIN

            EXECUTE IMMEDIATE 'ALTER TABLE ORDERS ENABLE CONSTRAINT CHK\_ORDER\_AMOUNTS';

            UPDATE ORDERS

            SET total\_amount = v\_original\_amount,

                final\_amount = v\_original\_amount - discount\_amount

            WHERE order\_id = v\_test\_order;

            COMMIT;

        EXCEPTION

            WHEN OTHERS THEN NULL;

        END;

        DBMS\_OUTPUT.PUT\_LINE('Error in test cleanup: ' || SQLERRM);

        RAISE;

END;

/

*--------------------------------------------------------------------------------*

*-- 5. FUNCTION 1: FN\_GENERATE\_VOUCHER\_CODE*

*--------------------------------------------------------------------------------*

*-- User Transaction:*

*-- Creates a unique voucher code with optional prefix and specified length,*

*-- using a combination of random characters for security.*

*-- Purpose:*

*-- To generate unpredictable voucher codes that are difficult to guess,*

*-- while allowing for categorization through prefixes.*

*-- Input Arguments:*

*-- p\_prefix  VARCHAR2  optional prefix for the voucher code*

*-- p\_length  NUMBER    total length of the voucher code (default 12)*

*-- Return:*

*-- VARCHAR2  the generated voucher code*

*--------------------------------------------------------------------------------*

CREATE OR REPLACE FUNCTION FN\_GENERATE\_VOUCHER\_CODE(

    p\_prefix IN VARCHAR2 DEFAULT NULL,

    p\_length IN NUMBER DEFAULT 12

) RETURN VARCHAR2 IS

    PRAGMA AUTONOMOUS\_TRANSACTION;  *-- allow COMMIT inside*

    v\_code VARCHAR2(100);

BEGIN

    v\_code := UPPER(NVL(p\_prefix, '')) ||

              DBMS\_RANDOM.STRING('A',

                GREATEST(1, p\_length - NVL(LENGTH(p\_prefix),0))

              );

*-- Audit log: voucher generation*

    LOG\_ACTIVITY(

      NULL,  *-- no specific customer*

      'VOUCHER\_GENERATION',

      'Generated voucher code "' || v\_code ||

      '" with prefix "' || NVL(p\_prefix,'') ||

      '" and length ' || p\_length,

      SYS\_CONTEXT('USERENV','IP\_ADDRESS')

    );

    RETURN v\_code;

EXCEPTION

    WHEN OTHERS THEN

*-- Log failure*

        LOG\_ACTIVITY(

          NULL,

          'VOUCHER\_GENERATION',

          'Error generating voucher code (' || NVL(p\_prefix,'') ||

          ',' || p\_length || '): ' || SQLERRM,

          SYS\_CONTEXT('USERENV','IP\_ADDRESS')

        );

        RETURN 'ERR' || TO\_CHAR(SYSDATE,'DDMMYYYYHH24MISS');

END;

/

*-----------------------------------------*

*-- UNIT TESTS FOR FN\_GENERATE\_VOUCHER\_CODE*

*-----------------------------------------*

PROMPT === UNIT TESTS: FN\_GENERATE\_VOUCHER\_CODE ===

*-- Test 1: Default generation*

PROMPT === TEST: DEFAULT GENERATION ===

DECLARE

    v\_code VARCHAR2(100);

BEGIN

    v\_code := FN\_GENERATE\_VOUCHER\_CODE();

    DBMS\_OUTPUT.PUT\_LINE('Generated code: ' || v\_code);

    IF LENGTH(v\_code) = 12 THEN

        DBMS\_OUTPUT.PUT\_LINE('SUCCESS: Correct default length');

    ELSE

        DBMS\_OUTPUT.PUT\_LINE('ERROR: Incorrect length');

    END IF;

END;

/

*-- Test 2: Custom length with prefix*

PROMPT === TEST: CUSTOM LENGTH WITH PREFIX ===

DECLARE

    v\_code VARCHAR2(100);

    v\_prefix VARCHAR2(10) := 'BDAY';

    v\_length NUMBER := 10;

BEGIN

    v\_code := FN\_GENERATE\_VOUCHER\_CODE(v\_prefix, v\_length);

    DBMS\_OUTPUT.PUT\_LINE('Generated code: ' || v\_code);

    IF v\_code LIKE v\_prefix || '%' AND LENGTH(v\_code) = v\_length THEN

        DBMS\_OUTPUT.PUT\_LINE('SUCCESS: Correct format');

    ELSE

        DBMS\_OUTPUT.PUT\_LINE('ERROR: Incorrect format');

    END IF;

END;

/

*-- Test 3: Short length*

PROMPT === TEST: SHORT LENGTH ===

DECLARE

    v\_code VARCHAR2(100);

BEGIN

    v\_code := FN\_GENERATE\_VOUCHER\_CODE(p\_length => 5);

    DBMS\_OUTPUT.PUT\_LINE('Generated code: ' || v\_code);

    IF LENGTH(v\_code) = 5 THEN

        DBMS\_OUTPUT.PUT\_LINE('SUCCESS: Correct short length');

    ELSE

        DBMS\_OUTPUT.PUT\_LINE('ERROR: Incorrect length');

    END IF;

END;

/

*-- VERIFICATION*

SELECT activity\_id,

       activity\_type,

       activity\_date,

       details

FROM   CUSTOMER\_ACTIVITY

WHERE  activity\_type = 'VOUCHER\_GENERATION'

  AND  activity\_date >= SYSTIMESTAMP - INTERVAL '5' MINUTE

ORDER  BY activity\_date DESC;

*--------------------------------------------------------------------------------*

*-- 6. FUNCTION 2: FN\_CALC\_VOUCHER\_VALUE*

*--------------------------------------------------------------------------------*

*-- User Transaction:*

*-- Calculates the monetary value a voucher would provide for a specific order amount*

*-- and customer, checking all applicable restrictions.*

*-- Purpose:*

*-- To determine the actual discount amount before voucher redemption occurs,*

*-- helping customers and staff understand voucher benefits.*

*-- Input Arguments:*

*-- p\_voucher\_id    NUMBER    ID of the voucher to evaluate*

*-- p\_order\_amount  NUMBER    amount of the order the voucher would apply to*

*-- p\_customer\_id   NUMBER    ID of the customer using the voucher*

*-- Return:*

*-- NUMBER  the calculated discount amount (0 if voucher can't be applied)*

*--------------------------------------------------------------------------------*

CREATE OR REPLACE FUNCTION FN\_CALC\_VOUCHER\_VALUE(

    p\_voucher\_id IN NUMBER,

    p\_order\_amount IN NUMBER,

    p\_customer\_id IN NUMBER

) RETURN NUMBER AS

    v\_discount\_value NUMBER := 0;

    v\_min\_spend NUMBER;

    v\_tier\_id NUMBER;

    v\_applicable\_tier\_id NUMBER;

    v\_promo\_type VARCHAR2(20);

BEGIN

*-- Get voucher details*

    BEGIN

        SELECT

            discount\_value,

            min\_spend,

            applicable\_tier\_id,

            promotion\_type

        INTO

            v\_discount\_value,

            v\_min\_spend,

            v\_applicable\_tier\_id,

            v\_promo\_type

        FROM PROMOTIONS

        WHERE promotion\_id = p\_voucher\_id

        AND SYSDATE BETWEEN valid\_from AND valid\_to

        AND promotion\_type = 'VOUCHER';

*-- Log the voucher being applied*

        LOG\_ACTIVITY(

            p\_customer\_id,  *-- Log it under the customer*

            'VOUCHER\_REDEMPTION',  *-- Activity type*

            'Voucher ID ' || p\_voucher\_id || ' applied for order amount ' || p\_order\_amount,

            SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')  *-- Optionally log IP*

        );

    EXCEPTION

        WHEN NO\_DATA\_FOUND THEN

*-- Log the failure to apply voucher (no matching voucher found)*

            LOG\_ACTIVITY(

                p\_customer\_id,

                'VOUCHER\_REDEMPTION',

                'Voucher ID ' || p\_voucher\_id || ' not found or expired for customer ' || p\_customer\_id,

                SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

            );

            RETURN 0;  *-- Return 0 if voucher not found*

    END;

*-- Check minimum spend*

    IF v\_min\_spend IS NOT NULL AND p\_order\_amount < v\_min\_spend THEN

*-- Log the failure to apply voucher due to minimum spend*

        LOG\_ACTIVITY(

            p\_customer\_id,

            'VOUCHER\_REDEMPTION',

            'Voucher ID ' || p\_voucher\_id || ' not applied due to minimum spend not met for customer ' || p\_customer\_id,

            SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

        );

        RETURN 0;  *-- Return 0 if minimum spend is not met*

    END IF;

*-- Check tier restriction*

    IF v\_applicable\_tier\_id IS NOT NULL THEN

        BEGIN

            SELECT tier\_id INTO v\_tier\_id

            FROM CUSTOMERS

            WHERE customer\_id = p\_customer\_id;

            IF v\_tier\_id != v\_applicable\_tier\_id THEN

*-- Log the failure to apply voucher due to tier mismatch*

                LOG\_ACTIVITY(

                    p\_customer\_id,

                    'VOUCHER\_REDEMPTION',

                    'Voucher ID ' || p\_voucher\_id || ' not applied due to tier mismatch for customer ' || p\_customer\_id,

                    SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

                );

                RETURN 0;  *-- Return 0 if the tier does not match*

            END IF;

        EXCEPTION

            WHEN NO\_DATA\_FOUND THEN

*-- Log failure if customer tier is not found*

                LOG\_ACTIVITY(

                    p\_customer\_id,

                    'VOUCHER\_REDEMPTION',

                    'Voucher ID ' || p\_voucher\_id || ' not applied due to missing tier for customer ' || p\_customer\_id,

                    SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

                );

                RETURN 0;  *-- Return 0 if tier not found*

        END;

    END IF;

*-- Calculate discount*

    IF v\_discount\_value <= 100 THEN *-- Percentage*

        LOG\_ACTIVITY(

            p\_customer\_id,

            'VOUCHER\_REDEMPTION',

            'Voucher ID ' || p\_voucher\_id || ' applied with a discount of ' || v\_discount\_value || '% for customer ' || p\_customer\_id,

            SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

        );

        RETURN ROUND(LEAST(p\_order\_amount \* v\_discount\_value / 100, p\_order\_amount), 2);

    ELSE *-- Fixed amount*

        LOG\_ACTIVITY(

            p\_customer\_id,

            'VOUCHER\_REDEMPTION',

            'Voucher ID ' || p\_voucher\_id || ' applied with a fixed discount of ' || v\_discount\_value || ' for customer ' || p\_customer\_id,

            SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

        );

        RETURN LEAST(v\_discount\_value, p\_order\_amount);

    END IF;

EXCEPTION

    WHEN OTHERS THEN

*-- Log any error that occurs within the function*

        LOG\_ACTIVITY(

            p\_customer\_id,

            'VOUCHER\_REDEMPTION',

            'Error applying voucher ID ' || p\_voucher\_id || ' for customer ' || p\_customer\_id || ': ' || SQLERRM,

            SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

        );

        RETURN 0;  *-- Return 0 in case of any error*

END;

/

*-- Create synonym*

CREATE OR REPLACE PUBLIC SYNONYM FN\_CALC\_VOUCHER\_VALUE FOR FN\_CALC\_VOUCHER\_VALUE;

*-- Grant permissions*

GRANT EXECUTE ON FN\_CALC\_VOUCHER\_VALUE TO PUBLIC;

GRANT EXECUTE ON FN\_CALC\_VOUCHER\_VALUE TO promo\_admin\_role;

GRANT EXECUTE ON FN\_CALC\_VOUCHER\_VALUE TO voucher\_admin\_role;

GRANT EXECUTE ON FN\_CALC\_VOUCHER\_VALUE TO promo\_manager\_role;

*-- VERIFICATION*

SELECT activity\_id,

       customer\_id,

       activity\_type,

       activity\_date,

       details

FROM   CUSTOMER\_ACTIVITY

WHERE  activity\_type = 'VOUCHER\_REDEMPTION'

ORDER  BY activity\_date DESC;

*-----------------------------------------*

*-- TEST CASES WITH VERIFICATION*

*-----------------------------------------*

*-- Before testing PROC\_REDEEM\_VOUCHER, update customer points*

UPDATE CUSTOMERS SET points\_balance = 1000 WHERE customer\_id = 1001;

UPDATE ORDERS SET total\_amount = 50 WHERE order\_id = 5001;

*/\*Current Test Coverage Summary*

*PROC\_BULK\_ASSIGN\_VOUCHERS, Tests bulk assignment to all customers, Verifies count before/after assignment ,Checks for new voucher assignments*

*PROC\_REDEEM\_VOUCHER: Tests voucher redemption for a specific customer, Verifies status change (is\_used flag), Checks redemption record creation, Tests with an existing order ID*

*FN\_CALC\_VOUCHER\_VALUE (indirectly tested through redemption): Tests discount calculation, Tests tier restrictions\*/*

PROMPT ===== TESTING =====

*-- 1. Prepare REDEMPTIONS table*

BEGIN

*-- First try to add the column if it doesn't exist*

    BEGIN

        EXECUTE IMMEDIATE 'ALTER TABLE REDEMPTIONS ADD (discount\_value NUMBER)';

        DBMS\_OUTPUT.PUT\_LINE('Added discount\_value column to REDEMPTIONS');

    EXCEPTION

        WHEN OTHERS THEN

            DBMS\_OUTPUT.PUT\_LINE('discount\_value column already exists');

    END;

*-- Then modify to allow zero points*

    EXECUTE IMMEDIATE 'ALTER TABLE REDEMPTIONS MODIFY (points\_used NUMBER DEFAULT 0)';

    DBMS\_OUTPUT.PUT\_LINE('Modified REDEMPTIONS table to allow zero points');

*-- Remove any problematic constraints*

    BEGIN

        EXECUTE IMMEDIATE 'ALTER TABLE REDEMPTIONS DROP CONSTRAINT SYS\_C0010245';

        DBMS\_OUTPUT.PUT\_LINE('Removed constraint SYS\_C0010245');

    EXCEPTION

        WHEN OTHERS THEN

            DBMS\_OUTPUT.PUT\_LINE('No constraint to remove or error removing: ' || SQLERRM);

    END;

EXCEPTION

    WHEN OTHERS THEN

        DBMS\_OUTPUT.PUT\_LINE('Error preparing REDEMPTIONS table: ' || SQLERRM);

END;

/

*-- 2. Test bulk voucher assignment with verification*

*-- Before the bulk assignment test, add:*

DELETE FROM CUSTOMER\_PROMOTIONS

WHERE promotion\_id IN (SELECT promotion\_id FROM PROMOTIONS WHERE is\_auto\_assign = 'Y');

COMMIT;

*-- Add this verification query*

SELECT p.promotion\_id, p.name, COUNT(cp.customer\_id) AS assigned\_count

FROM PROMOTIONS p

LEFT JOIN CUSTOMER\_PROMOTIONS cp ON p.promotion\_id = cp.promotion\_id

WHERE p.is\_auto\_assign = 'Y'

AND SYSDATE BETWEEN p.valid\_from AND p.valid\_to

GROUP BY p.promotion\_id, p.name;

PROMPT === TEST: BULK VOUCHER ASSIGNMENT ===

DECLARE

    v\_before\_count NUMBER;

    v\_after\_count NUMBER;

BEGIN

*-- Get count before assignment*

    SELECT COUNT(\*) INTO v\_before\_count FROM CUSTOMER\_PROMOTIONS;

    DBMS\_OUTPUT.PUT\_LINE('Assignments before: ' || v\_before\_count);

*-- Execute bulk assignment*

    DBMS\_OUTPUT.PUT\_LINE('Executing bulk assignment...');

    BEGIN

        PROC\_BULK\_ASSIGN\_VOUCHERS('ALL');

    EXCEPTION

        WHEN OTHERS THEN

            DBMS\_OUTPUT.PUT\_LINE('Error in bulk assignment: ' || SQLERRM);

*-- Show procedure errors if any*

            FOR err IN (SELECT line, position, text FROM user\_errors WHERE name = 'PROC\_BULK\_ASSIGN\_VOUCHERS') LOOP

                DBMS\_OUTPUT.PUT\_LINE('Procedure error at line ' || err.line || ': ' || err.text);

            END LOOP;

            RAISE;

    END;

*-- Get count after assignment*

    SELECT COUNT(\*) INTO v\_after\_count FROM CUSTOMER\_PROMOTIONS;

    DBMS\_OUTPUT.PUT\_LINE('Assignments after: ' || v\_after\_count);

*-- Verification*

    IF v\_after\_count > v\_before\_count THEN

        DBMS\_OUTPUT.PUT\_LINE('SUCCESS: New voucher assignments created');

    ELSE

        DBMS\_OUTPUT.PUT\_LINE('WARNING: No new voucher assignments created');

    END IF;

END;

/

*-- 3. Test voucher redemption with verification*

PROMPT === TEST: VOUCHER REDEMPTION ===

DECLARE

    v\_test\_customer NUMBER := 1001; *-- Bronze customer*

    v\_test\_voucher NUMBER := 1;    *-- 10% Off voucher*

    v\_test\_order NUMBER := 5001;   *-- Use existing order ID to avoid FK violation*

    v\_before\_status CHAR(1);

    v\_after\_status CHAR(1);

    v\_redemption\_count NUMBER;

BEGIN

*-- Reset test data if needed*

    UPDATE CUSTOMER\_PROMOTIONS

    SET is\_used = 'N', used\_date = NULL

    WHERE customer\_id = v\_test\_customer AND promotion\_id = v\_test\_voucher;

    COMMIT;

*-- Get current status*

    SELECT is\_used INTO v\_before\_status

    FROM CUSTOMER\_PROMOTIONS

    WHERE customer\_id = v\_test\_customer AND promotion\_id = v\_test\_voucher;

*-- Execute redemption*

    DBMS\_OUTPUT.PUT\_LINE('Attempting to redeem voucher...');

    BEGIN

        PROC\_REDEEM\_VOUCHER(v\_test\_customer, v\_test\_voucher, v\_test\_order);

    EXCEPTION

        WHEN OTHERS THEN

            DBMS\_OUTPUT.PUT\_LINE('Redemption failed: ' || SQLERRM);

    END;

*-- Verify status changed*

    SELECT is\_used INTO v\_after\_status

    FROM CUSTOMER\_PROMOTIONS

    WHERE customer\_id = v\_test\_customer AND promotion\_id = v\_test\_voucher;

*-- Check redemption record*

    SELECT COUNT(\*) INTO v\_redemption\_count

    FROM REDEMPTIONS

    WHERE customer\_id = v\_test\_customer AND promotion\_id = v\_test\_voucher;

*-- Output results*

    DBMS\_OUTPUT.PUT\_LINE('Before status: ' || v\_before\_status || ', After status: ' || v\_after\_status);

    DBMS\_OUTPUT.PUT\_LINE('Redemption records found: ' || v\_redemption\_count);

    IF v\_after\_status = 'Y' THEN

        DBMS\_OUTPUT.PUT\_LINE('SUCCESS: Voucher marked as used');

    ELSE

        DBMS\_OUTPUT.PUT\_LINE('WARNING: Voucher not marked as used');

    END IF;

    IF v\_redemption\_count > 0 THEN

        DBMS\_OUTPUT.PUT\_LINE('SUCCESS: Redemption record created');

    ELSE

        DBMS\_OUTPUT.PUT\_LINE('WARNING: No redemption record found');

    END IF;

END;

/

*-- Final verification*

PROMPT ===== FINAL VERIFICATION =====

SELECT \* FROM CUSTOMER\_PROMOTIONS WHERE is\_used = 'Y';

SELECT \* FROM REDEMPTIONS;

SELECT \* FROM POINT\_TRANSACTIONS;

PROMPT ===== SCRIPT COMPLETED =====

*-- Re-enable trigger if it exists*

BEGIN

  EXECUTE IMMEDIATE 'BEGIN

    EXECUTE IMMEDIATE ''ALTER TRIGGER trg\_order\_totals ENABLE'';

  EXCEPTION

    WHEN OTHERS THEN

      IF SQLCODE != -4080 THEN -- ORA-04080: trigger does not exist

        RAISE;

      END IF;

  END;';

END;

/

## Teh Bee Ling - Personal Script 3

*/\**

*GROUP NUMBER : G003*

*PROGRAMME : CS*

*STUDENT ID : 2204237*

*STUDENT NAME : TEH BEE LING*

*Submission date and time (DD-MON-YY): 29 April 2025*

*\*/*

*-- Every Error Test Case is checked and place commented.*

*-- Pre-Setup*

SET SERVEROUTPUT ON

SET LINESIZE 500;

SET PAGESIZE 500;

*--------------------------------------------------*

*-- Create required placeholder functions first*

*--------------------------------------------------*

*-- These minimal implementations enable compilation and testing of main procedures*

*-- Will be replaced with full implementations during development*

*/\**

*Purpose of these placeholders:*

*- Provide minimal implementations to avoid dependency errors*

*- Will be replaced by actual implementations later in the script*

*- Enable testing of the overall script structure first*

*\*/*

*-- Simple placeholder for FN\_CALC\_VOUCHER\_VALUE*

CREATE OR REPLACE FUNCTION FN\_CALC\_VOUCHER\_VALUE(

    p\_voucher\_id IN NUMBER,

    p\_total\_amount IN NUMBER,

    p\_customer\_id IN NUMBER

) RETURN NUMBER AS

BEGIN

    RETURN 0; *-- Default to no discount*

END;

/

*-- Simple placeholder for PROC\_REDEEM\_VOUCHER*

CREATE OR REPLACE PROCEDURE PROC\_REDEEM\_VOUCHER(

    p\_customer\_id IN NUMBER,

    p\_voucher\_id IN NUMBER,

    p\_order\_id IN NUMBER

) AS

BEGIN

    NULL; *-- Do nothing for now*

END;

/

*-- Simple placeholder for FN\_CHECK\_SEASONAL\_REDEMPTION*

CREATE OR REPLACE FUNCTION FN\_CHECK\_SEASONAL\_REDEMPTION(

    p\_customer\_id IN NUMBER,

    p\_setmeal\_id IN NUMBER,

    p\_price IN NUMBER

) RETURN VARCHAR2 AS

BEGIN

    RETURN 'ELIGIBLE'; *-- Default to eligible*

END;

/

*-- Simple placeholder for PROC\_REDEEM\_SEASONAL\_ITEM*

CREATE OR REPLACE PROCEDURE PROC\_REDEEM\_SEASONAL\_ITEM(

    p\_customer\_id IN NUMBER,

    p\_setmeal\_id IN NUMBER,

    p\_order\_id IN NUMBER

) AS

BEGIN

    NULL; *-- Do nothing for now*

END;

/

*--------------------------------------------------------------------------------*

*-- FUNCTION 1: FN\_VALIDATE\_SETMEAL*

*--------------------------------------------------------------------------------*

*-- User Transaction:*

*-- Validate the composition and status of a set meal before order processing.*

*-- Purpose:*

*-- Ensure a set meal has valid active components before allowing orders.*

*-- Input Arguments:*

*-- p\_setmeal\_id       NUMBER   ID of the set meal to validate*

*-- Return:*

*-- VARCHAR2  Description of the set meal validation status with possible values:*

*-- 'VALID'               - Set meal meets all validation criteria*

*-- 'INVALID\_SETMEAL'     - Set meal does not exist or is inactive*

*-- 'NO\_COMPONENTS'       - Set meal has no components defined*

*-- 'INACTIVE\_COMPONENTS' - One or more components are inactive*

*-- 'VALIDATION\_ERROR'    - Unexpected system error occurred*

*--------------------------------------------------------------------------------*

CREATE OR REPLACE FUNCTION FN\_VALIDATE\_SETMEAL(

    p\_setmeal\_id IN NUMBER

) RETURN VARCHAR2 AS

    v\_component\_count NUMBER;

    v\_active\_component\_count NUMBER;

BEGIN

*-- Check set meal exists and is active*

    BEGIN

        SELECT 1

        INTO v\_component\_count

        FROM MENU\_ITEMS

        WHERE item\_id = p\_setmeal\_id

        AND item\_type = 'SET\_MEAL'

        AND is\_active = 'Y';

    EXCEPTION

        WHEN NO\_DATA\_FOUND THEN

            RETURN 'INVALID\_SETMEAL';

    END;

*-- Count all components*

    SELECT COUNT(\*)

    INTO v\_component\_count

    FROM SET\_MEAL\_COMPONENTS

    WHERE set\_meal\_id = p\_setmeal\_id;

*-- Count active components*

    SELECT COUNT(\*)

    INTO v\_active\_component\_count

    FROM SET\_MEAL\_COMPONENTS smc

    JOIN MENU\_ITEMS mi ON smc.component\_id = mi.item\_id

    WHERE smc.set\_meal\_id = p\_setmeal\_id

    AND mi.is\_active = 'Y';

*-- Validation rules*

    IF v\_component\_count = 0 THEN

        RETURN 'NO\_COMPONENTS';

    ELSIF v\_active\_component\_count < v\_component\_count THEN

        RETURN 'INACTIVE\_COMPONENTS';

    ELSE

        RETURN 'VALID';

    END IF;

EXCEPTION

    WHEN OTHERS THEN

        RETURN 'VALIDATION\_ERROR';

END;

/

*-------------------------------------------------------------------*

*-- DEMO TEST 1 (NORMAL CASE): FN\_VALIDATE\_SETMEAL*

*-------------------------------------------------------------------*

*-- Testing Family Combo (ID 5) which has all active components*

*-- Expected Output: 'VALID'*

*-------------------------------------------------------------------*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('=== TESTING WITH VALID SET MEAL ===');

    DBMS\_OUTPUT.PUT\_LINE(FN\_VALIDATE\_SETMEAL(5));

    DBMS\_OUTPUT.PUT\_LINE('Expected Output: VALID');

END;

/

*---------------------------------------------------------------------*

*-- DEMO TEST 2 (Non-existent set meal): FN\_VALIDATE\_SETMEAL*

*---------------------------------------------------------------------*

*-- Testing with non-existent ID 999*

*-- Expected Output: 'INVALID\_SETMEAL'*

*---------------------------------------------------------------------*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('=== TESTING WITH NON-EXISTENT SET MEAL ===');

    DBMS\_OUTPUT.PUT\_LINE(FN\_VALIDATE\_SETMEAL(999));

    DBMS\_OUTPUT.PUT\_LINE('Expected Output: INVALID\_SETMEAL');

END;

/

*-------------------------------------------------------------------------------*

*-- DEMO TEST 3 (No Components): FN\_VALIDATE\_SETMEAL*

*-------------------------------------------------------------------------------*

*-- Expected Output:*

*-- 'NO\_COMPONENTS' because the set meal exists but has no associated components.*

*-------------------------------------------------------------------------------*

UPDATE MENU\_ITEMS SET item\_type = 'SET\_MEAL' WHERE item\_id = 7;

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('=== TESTING WITH NO COMPONENT SET MEAL ===');

    DBMS\_OUTPUT.PUT\_LINE(FN\_VALIDATE\_SETMEAL(7));

    DBMS\_OUTPUT.PUT\_LINE('Expected Output: NO\_COMPONENTS');

END;

/

UPDATE MENU\_ITEMS SET item\_type = 'INGREDIENT' WHERE item\_id = 7;

*---------------------------------------------------------------------------------------------*

*-- DEMO TEST 4 (Inactive Component): FN\_VALIDATE\_SETMEAL*

*---------------------------------------------------------------------------------------------*

*-- Expected Output:*

*-- 'INACTIVE\_COMPONENTS' because the set meal exists but has one or more inactive components.*

*---------------------------------------------------------------------------------------------*

UPDATE MENU\_ITEMS SET is\_active = 'N' WHERE item\_id = 1;

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('=== TESTING WITH INACTIVE COMPONENT SET MEAL ===');

    DBMS\_OUTPUT.PUT\_LINE(FN\_VALIDATE\_SETMEAL(6));

    DBMS\_OUTPUT.PUT\_LINE('Expected: INACTIVE\_COMPONENTS');

END;

/

*-- Restore Cheeseburger component to active status*

UPDATE MENU\_ITEMS SET is\_active = 'Y' WHERE item\_id = 1;

*-------------------------------------------------------------------------------------*

*-- FUNCTION 2: FN\_CALC\_SETMEAL\_PRICE*

*-- (Removed logging functionality)*

*-------------------------------------------------------------------------------------*

*-- User Transaction:*

*-- Calculate the final price of a set meal including standard and tier discounts.*

*-- Purpose:*

*-- To determine the customer specific price for set meals,*

*-- ensures that the final price does not drop below a minimum margin.*

*-- Input Arguments:*

*-- p\_setmeal\_id            NUMBER     ID of the set meal to validate*

*-- p\_customer\_id           NUMBER     ID of the ordering customer*

*-- p\_apply\_tier\_discount   BOOLEAN    Whether to include tier discount*

*-- Return:*

*-- NUMBER the final price of the set meal after applying standard and tier discounts*

*------------------------------------------------------------------------------------*

CREATE OR REPLACE FUNCTION FN\_CALC\_SETMEAL\_PRICE(

    p\_setmeal\_id IN NUMBER,

    p\_customer\_id IN NUMBER,

    p\_apply\_tier\_discount BOOLEAN DEFAULT TRUE

) RETURN NUMBER AS

    v\_component\_cost NUMBER := 0;

    v\_standard\_discount\_pct NUMBER := 15;

    v\_tier\_discount\_pct NUMBER := 0;

    v\_final\_price NUMBER;

BEGIN

*-- Sum component costs*

    SELECT SUM(mi.base\_price \* smc.quantity)

    INTO v\_component\_cost

    FROM SET\_MEAL\_COMPONENTS smc

    JOIN MENU\_ITEMS mi ON smc.component\_id = mi.item\_id

    WHERE smc.set\_meal\_id = p\_setmeal\_id;

*-- Get tier discount if requested*

    IF p\_apply\_tier\_discount THEN

        BEGIN

            SELECT mt.discount\_percentage

            INTO v\_tier\_discount\_pct

            FROM CUSTOMERS c

            JOIN MEMBERSHIP\_TIERS mt ON c.tier\_id = mt.tier\_id

            WHERE c.customer\_id = p\_customer\_id;

        EXCEPTION

            WHEN OTHERS THEN

                v\_tier\_discount\_pct := 0;

        END;

    END IF;

*-- Calculate final price with constraints*

    v\_final\_price := GREATEST(

        v\_component\_cost \* (1 - (LEAST(v\_standard\_discount\_pct + v\_tier\_discount\_pct, 50)/100)),

        v\_component\_cost \* 0.7  *-- Minimum 30% margin*

    );

    RETURN ROUND(v\_final\_price, 2);

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        RAISE\_APPLICATION\_ERROR(-20030, 'Set meal components not found');

END;

/

*---------------------------------------------------------------------*

*-- DEMO TEST 1 (NORMAL CASE): FN\_CALC\_SETMEAL\_PRICE*

*---------------------------------------------------------------------*

*-- Tests standard pricing calculation for Bronze member*

*-- Expected Output:*

*-- Calculated price with 20% total discount (15% standard + 5% tier)*

*----------------------------------------------------------------------*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('=== TESTING Bronze member with tier discount ===');

    DBMS\_OUTPUT.PUT\_LINE('Calculated Price: ' || FN\_CALC\_SETMEAL\_PRICE(5, 1001));

    DBMS\_OUTPUT.PUT\_LINE('Expected Output: Price with 20% discount (15% standard + 5% tier)');

END;

/

*/\**

*---------------------------------------------------------------------*

*-- DEMO TEST 2 (ERROR CASE): FN\_CALC\_SETMEAL\_PRICE*

*---------------------------------------------------------------------*

*-- Tests error handling for non-existent set meal*

*-- Expected Output:*

*-- Error message "Set meal components not found"*

*----------------------------------------------------------------------*

*BEGIN*

*DBMS\_OUTPUT.PUT\_LINE('=== TESTING Invalid set meal ===');*

*BEGIN*

*DBMS\_OUTPUT.PUT\_LINE('Price: ' || FN\_CALC\_SETMEAL\_PRICE(999, 1001));*

*EXCEPTION*

*WHEN OTHERS THEN*

*DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);*

*END;*

*DBMS\_OUTPUT.PUT\_LINE('Expected Output: ORA-20030: Set meal components not found');*

*END;*

*/*

*\*/*

*----------------------------------------------------------------------------------------*

*-- PROCEDURE 1: PROC\_CREATE\_ALA\_CARTE\_ORDER*

*-- (Fixed syntax errors)*

*----------------------------------------------------------------------------------------*

*-- User Transaction:*

*-- Create an order for à la carte menu items.*

*-- Purpose:*

*-- Process customer orders for individual menu items with optional voucher discounts.*

*-- Input Arguments:*

*-- p\_customer\_id       NUMBER             Ordering customer ID*

*-- p\_item\_ids          VARCHAR2           Comma-separated list of item IDs*

*-- p\_payment\_method    VARCHAR2           Payment method used for the order*

*-- p\_voucher\_id        NUMBER,optional    Optional voucher ID for discount*

*----------------------------------------------------------------------------------------*

CREATE OR REPLACE PROCEDURE PROC\_CREATE\_ALA\_CARTE\_ORDER(

    p\_customer\_id    IN NUMBER,

    p\_item\_ids       IN VARCHAR2,

    p\_payment\_method IN VARCHAR2,

    p\_voucher\_id     IN NUMBER DEFAULT NULL

) AS

    v\_order\_id        NUMBER;

    v\_total\_amount    NUMBER := 0;

    v\_discount\_amount NUMBER := 0;

    v\_item\_count      NUMBER := 0;

    v\_item\_price      NUMBER;

    v\_customer\_exists NUMBER;

    v\_order\_type      VARCHAR2(20);

BEGIN

*-- 1) Validate customer*

    SELECT COUNT(\*)

      INTO v\_customer\_exists

      FROM CUSTOMERS

     WHERE customer\_id = p\_customer\_id

       AND is\_member   = 'Y';

    IF v\_customer\_exists = 0 THEN

        RAISE\_APPLICATION\_ERROR(-20010, 'Invalid or non-member customer');

    END IF;

*-- 2) Create order header*

    INSERT INTO ORDERS (

        customer\_id,

        payment\_method,

        status,

        total\_amount,

        discount\_amount,

        final\_amount

    ) VALUES (

        p\_customer\_id,

        p\_payment\_method,

        'COMPLETED',

        0, 0, 0

    )

    RETURNING order\_id INTO v\_order\_id;

*-- 3) Process each item*

    FOR item\_rec IN (

        SELECT TO\_NUMBER(TRIM(REGEXP\_SUBSTR(p\_item\_ids,'[^,]+',1,LEVEL))) AS item\_id

          FROM DUAL

        CONNECT BY REGEXP\_SUBSTR(p\_item\_ids,'[^,]+',1,LEVEL) IS NOT NULL

    ) LOOP

        BEGIN

            SELECT base\_price

              INTO v\_item\_price

              FROM MENU\_ITEMS

             WHERE item\_id    = item\_rec.item\_id

               AND item\_type  = 'A\_LA\_CARTE'

               AND is\_active  = 'Y';

            INSERT INTO ORDER\_ITEMS (

                order\_id, item\_id, quantity, price

            ) VALUES (

                v\_order\_id, item\_rec.item\_id, 1, v\_item\_price

            );

            v\_total\_amount := v\_total\_amount + v\_item\_price;

            v\_item\_count   := v\_item\_count + 1;

        EXCEPTION

            WHEN NO\_DATA\_FOUND THEN

                DBMS\_OUTPUT.PUT\_LINE('Skipped invalid item ID: ' || item\_rec.item\_id);

        END;

    END LOOP;

*-- 4) Ensure at least one item*

    IF v\_item\_count = 0 THEN

        RAISE\_APPLICATION\_ERROR(-20011, 'No valid items were processed');

    END IF;

*-- 5) Determine order type for logging*

    IF INSTR(p\_item\_ids, 'SET\_MEAL') > 0 THEN

        v\_order\_type := 'SET\_MEAL';

    ELSE

        v\_order\_type := 'A\_LA\_CARTE';

    END IF;

*-- 6) Apply voucher (if any)*

    IF p\_voucher\_id IS NOT NULL THEN

        v\_discount\_amount := FN\_CALC\_VOUCHER\_VALUE(p\_voucher\_id, v\_total\_amount, p\_customer\_id);

        IF v\_discount\_amount > 0 THEN

            BEGIN

                PROC\_REDEEM\_VOUCHER(p\_customer\_id, p\_voucher\_id, v\_order\_id);

            EXCEPTION

                WHEN OTHERS THEN

                    v\_discount\_amount := 0;

                    DBMS\_OUTPUT.PUT\_LINE('Voucher redemption failed: ' || SQLERRM);

            END;

        END IF;

    END IF;

*-- 7) Update totals*

    UPDATE ORDERS

       SET total\_amount    = v\_total\_amount,

           discount\_amount = v\_discount\_amount,

           final\_amount    = GREATEST(v\_total\_amount - v\_discount\_amount,0)

     WHERE order\_id = v\_order\_id;

    COMMIT;

*-- 8) Log success*

    LOG\_ACTIVITY(

        p\_customer\_id,

        'ORDER',

        'Created order #' || v\_order\_id

          || ' (' || v\_order\_type || ')'

          || ' items=' || v\_item\_count

          || ', total=' || v\_total\_amount

          || ', discount=' || v\_discount\_amount,

        SYS\_CONTEXT('USERENV','IP\_ADDRESS')

    );

    DBMS\_OUTPUT.PUT\_LINE(

      'Successfully created order #'||v\_order\_id

      ||' with '||v\_item\_count||' items'

    );

EXCEPTION

    WHEN OTHERS THEN

        ROLLBACK;

*-- 9) Log failure*

        LOG\_ACTIVITY(

            p\_customer\_id,

            'ORDER',

            'Error creating order: '||SQLERRM,

            SYS\_CONTEXT('USERENV','IP\_ADDRESS')

        );

        DBMS\_OUTPUT.PUT\_LINE('Error creating order: '||SQLERRM);

        RAISE;

END;

/

*------------------------------------------------------------------------------------*

*-- DEMO TEST 1 (NORMAL CASE): PROC\_CREATE\_ALA\_CARTE\_ORDER*

*------------------------------------------------------------------------------------*

*-- Test for a valid customer and valid à la carte items*

*-- Expected Output:*

*-- Successfully created order, with total amount calculated and voucher applied.*

*------------------------------------------------------------------------------------*

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('=== TESTING VALID ORDER FOR A LA CARTE ===');

    PROC\_CREATE\_ALA\_CARTE\_ORDER(

        p\_customer\_id => 1001,

        p\_item\_ids => '1,2,3',

        p\_payment\_method => 'CARD',

        p\_voucher\_id => NULL

    );

    DBMS\_OUTPUT.PUT\_LINE('Expected Output: Order successfully created for customer 1001');

END;

/

*/\**

*----------------------------------------------------------------------*

*-- DEMO TEST 2 (ERROR CASE): PROC\_CREATE\_ALA\_CARTE\_ORDER*

*----------------------------------------------------------------------*

*-- Test for an invalid customer*

*-- Expected Output:*

*-- Error message "Invalid or non-member customer"*

*----------------------------------------------------------------------*

*BEGIN*

*DBMS\_OUTPUT.PUT\_LINE('=== TESTING INVALID CUSTOMER ID 9999 ===');*

*BEGIN*

*PROC\_CREATE\_ALA\_CARTE\_ORDER(*

*p\_customer\_id => 9999,*

*p\_item\_ids => '1,2',*

*p\_payment\_method => 'CASH',*

*p\_voucher\_id => NULL*

*);*

*EXCEPTION*

*WHEN OTHERS THEN*

*DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);*

*END;*

*DBMS\_OUTPUT.PUT\_LINE('Expected Output: ORA-20010: Invalid or non-member customer');*

*END;*

*/*

*\*/*

*---------------------------------------------------------------------------------------------------------------*

*-- PROCEDURE 2: PROC\_CREATE\_SETMEAL\_ORDER*

*---------------------------------------------------------------------------------------------------------------*

*-- User Transaction:*

*-- Create an order for set meals with validation and optional seasonal redemption.*

*-- Purpose:*

*-- Handle set meal orders including validation, pricing, and special redemptions.*

*-- Input Arguments:*

*-- p\_customer\_id                 NUMBER              Ordering customer ID*

*-- p\_setmeal\_id                  NUMBER              Set meal ID being ordered*

*-- p\_payment\_method              VARCHAR2            Payment method used for the order*

*-- p\_voucher\_id                  NUMBER,optional     Optional voucher ID to apply a discount to order*

*-- p\_apply\_seasonal\_redemption   BOOLEAN,optional    Whether to attempt seosonal redemption for the set meal*

*---------------------------------------------------------------------------------------------------------------*

SET DEFINE OFF;

CREATE OR REPLACE PROCEDURE PROC\_CREATE\_SETMEAL\_ORDER(

    p\_customer\_id IN NUMBER,

    p\_setmeal\_id IN NUMBER,

    p\_payment\_method IN VARCHAR2,

    p\_voucher\_id IN NUMBER DEFAULT NULL,

    p\_apply\_seasonal\_redemption IN BOOLEAN DEFAULT FALSE

) AS

    v\_order\_id          NUMBER;

    v\_setmeal\_price     NUMBER;

    v\_discount\_amount   NUMBER := 0;

    v\_promo\_discount    NUMBER := 0;

    v\_promo\_exists      NUMBER := 0;

    v\_customer\_tier\_id  NUMBER;

    v\_is\_member         CHAR(1);

    v\_base\_price        NUMBER;

    v\_details           VARCHAR2(4000);

    v\_total\_amount      NUMBER := 0;

    v\_final\_amount      NUMBER := 0;

BEGIN

*-- 1) Validate customer exists and is a member*

    BEGIN

        SELECT tier\_id, is\_member

          INTO v\_customer\_tier\_id, v\_is\_member

          FROM CUSTOMERS

         WHERE customer\_id = p\_customer\_id;

        IF v\_is\_member != 'Y' THEN

            RAISE\_APPLICATION\_ERROR(-20010, 'Customer is not a member');

        END IF;

        LOG\_ACTIVITY(

            p\_customer\_id,

            'ORDER',

            'Customer ' || p\_customer\_id || ' validated for set meal order',

            SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

        );

    EXCEPTION

        WHEN NO\_DATA\_FOUND THEN

            RAISE\_APPLICATION\_ERROR(-20011, 'Customer not found');

    END;

*-- 2) Check if promotion exists for set meal, if not create one*

    BEGIN

        SELECT COUNT(\*) INTO v\_promo\_exists

        FROM PROMOTIONS

        WHERE promotion\_type = 'SET\_MEAL'

          AND set\_meal\_id = p\_setmeal\_id

          AND SYSDATE BETWEEN valid\_from AND valid\_to;

        IF v\_promo\_exists = 0 THEN

*-- If no promotion exists, create one automatically*

            INSERT INTO PROMOTIONS (

                promotion\_type, name, valid\_from, valid\_to, discount\_value, set\_meal\_id

            ) VALUES (

                'SET\_MEAL',

                'Auto Promo for Set Meal ' || p\_setmeal\_id,

                SYSDATE,

                SYSDATE + 30,

                5,  *-- default discount value*

                p\_setmeal\_id

            );

            COMMIT;  *-- Commit to save promotion immediately*

            LOG\_ACTIVITY(

                p\_customer\_id,

                'VOUCHER\_ASSIGNMENT',

                'Auto-created promotion for set meal ' || p\_setmeal\_id,

                SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

            );

            DBMS\_OUTPUT.PUT\_LINE('Auto-created promotion for set meal ID ' || p\_setmeal\_id);

        END IF;

    END;

*-- 3) Validate set meal exists and is active*

    BEGIN

        SELECT base\_price

          INTO v\_base\_price

          FROM MENU\_ITEMS

         WHERE item\_id = p\_setmeal\_id

           AND item\_type = 'SET\_MEAL'

           AND is\_active = 'Y';

        LOG\_ACTIVITY(

            p\_customer\_id,

            'ORDER',

            'Set meal ' || p\_setmeal\_id || ' validated for customer ' || p\_customer\_id,

            SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

        );

    EXCEPTION

        WHEN NO\_DATA\_FOUND THEN

            RAISE\_APPLICATION\_ERROR(-20020, 'Set meal not available');

    END;

*-- 4) Calculate the total amount, discount, and final amount*

    v\_total\_amount := v\_base\_price;  *-- Assuming total amount is the base price*

    v\_discount\_amount := 0;  *-- Default discount is 0*

*-- Apply promo discount if found*

    IF v\_promo\_discount IS NOT NULL THEN

        v\_discount\_amount := v\_promo\_discount;

    END IF;

*-- Apply voucher discount if provided*

    IF p\_voucher\_id IS NOT NULL THEN

        v\_discount\_amount := v\_discount\_amount + FN\_CALC\_VOUCHER\_VALUE(

                                p\_voucher\_id, v\_total\_amount, p\_customer\_id

                             );

    END IF;

*-- Ensure the discount does not exceed the total amount*

    IF v\_discount\_amount > v\_total\_amount THEN

        v\_discount\_amount := v\_total\_amount;  *-- Adjust the discount if it exceeds the total amount*

    END IF;

*-- Calculate final amount after applying discount*

    v\_final\_amount := GREATEST(v\_total\_amount - v\_discount\_amount, 0);  *-- Ensure final amount is not negative*

*-- 5) Create order with the calculated amounts*

    INSERT INTO ORDERS(

        customer\_id, payment\_method, status,

        total\_amount, discount\_amount, final\_amount

    ) VALUES (

        p\_customer\_id, p\_payment\_method, 'COMPLETED',

        v\_total\_amount, v\_discount\_amount, v\_final\_amount

    ) RETURNING order\_id INTO v\_order\_id;

*-- Add the setmeal item to the order*

    INSERT INTO ORDER\_ITEMS(

        order\_id, item\_id, quantity, price

    ) VALUES (

        v\_order\_id, p\_setmeal\_id, 1, v\_base\_price

    );

*-- 6) Log successful order creation*

    v\_details := 'Created SET\_MEAL order #' || v\_order\_id ||

                 ' with item ' || p\_setmeal\_id ||

                 ' base price=' || v\_base\_price ||

                 ' promo discount=' || v\_promo\_discount ||

                 ' final price=' || v\_final\_amount;

    LOG\_ACTIVITY(

        p\_customer\_id,

        'ORDER',

        v\_details,

        SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

    );

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Successfully created set meal order #' || v\_order\_id);

EXCEPTION

    WHEN OTHERS THEN

        ROLLBACK;

        v\_details := 'Failed to create set meal order: ' || SQLERRM;

        LOG\_ACTIVITY(

            p\_customer\_id,

            'ORDER',

            v\_details,

            SYS\_CONTEXT('USERENV', 'IP\_ADDRESS')

        );

        DBMS\_OUTPUT.PUT\_LINE('Error creating set meal order: ' || SQLERRM);

        RAISE;

END;

/

*------------------------------------------------------------------------------------------------------*

*-- DEMO TEST 1 (NORMAL CASE): PROC\_CREATE\_SETMEAL\_ORDER*

*------------------------------------------------------------------------------------------------------*

*-- Test for valid customer, valid set meal, and payment method 'CARD'*

*-- Expected Output:*

*-- Successfully created order with voucher applied, final price calculated.*

*------------------------------------------------------------------------------------------------------*

BEGIN

*-- Testing valid set meal order creation*

    DBMS\_OUTPUT.PUT\_LINE('=== TESTING VALID ORDER FOR SET MEAL ===');

*-- Call the procedure to create set meal order*

    PROC\_CREATE\_SETMEAL\_ORDER(

        p\_customer\_id => 1001,  *-- Specify customer ID*

        p\_setmeal\_id => 5,      *-- Specify set meal ID*

        p\_payment\_method => 'CARD',  *-- Payment method (e.g., 'CARD')*

        p\_voucher\_id => 1,      *-- Voucher ID (if applicable)*

        p\_apply\_seasonal\_redemption => FALSE *-- Seasonal redemption flag*

    );

    DBMS\_OUTPUT.PUT\_LINE('Expected Output: Successfully created set meal order with voucher applied');

END;

/

*/\**

*------------------------------------------------------------------------------------------------------*

*-- DEMO TEST 2 (ERROR CASE): PROC\_CREATE\_SETMEAL\_ORDER*

*------------------------------------------------------------------------------------------------------*

*-- Test for an invalid set meal, which does not exist or is inactive*

*-- Expected Output:*

*-- Error message "Set meal not available"*

*------------------------------------------------------------------------------------------------------*

*BEGIN*

*DBMS\_OUTPUT.PUT\_LINE('=== TESTING INVALID SET MEAL ID 999 ===');*

*BEGIN*

*PROC\_CREATE\_SETMEAL\_ORDER(*

*p\_customer\_id => 1001,*

*p\_setmeal\_id => 999,*

*p\_payment\_method => 'CASH',*

*p\_voucher\_id => NULL,*

*p\_apply\_seasonal\_redemption => FALSE*

*);*

*EXCEPTION*

*WHEN OTHERS THEN*

*DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);*

*END;*

*DBMS\_OUTPUT.PUT\_LINE('Expected Output: ORA-20020: Set meal not available');*

*END;*

*/*

*\*/*

*-----------------------------------------------------------------------------------------------------------------*

*-- 6. QUERY: Set Meal Profitability Analysis*

*-----------------------------------------------------------------------------------------------------------------*

*-- User transaction:*

*-- List out all set meals with their component costs, profit margins, and recent sales volume,*

*-- including validation status, ordered by profitability.*

*-- Purpose:*

*-- Analyze the profitability of set meals by comparing component costs to selling prices.*

*-- Compare cost vs. revenue for each set meal that shows gross profit and sales volume last 3 months.*

*-- This query retrieves a detailed profitability analysis for set meals, including the total component*

*-- cost, gross profit, and margin percentages. It also includes the number of sales made for each set*

*-- meal over the last 3 months. The results are ordered by the base margin percentage in descending order.*

*------------------------------------------------------------------------------------------------------------------*

SELECT

    sm.item\_id,

    sm.name,

    sm.base\_price AS selling\_price,

    SUM(mi.base\_price \* smc.quantity) AS component\_cost,

    sm.base\_price - SUM(mi.base\_price \* smc.quantity) AS gross\_profit,

    ROUND((sm.base\_price - SUM(mi.base\_price \* smc.quantity)) / sm.base\_price \* 100, 2) AS base\_margin\_pct,

    ROUND((sm.base\_price\*0.85 - SUM(mi.base\_price \* smc.quantity)) / (sm.base\_price\*0.85) \* 100, 2) AS discounted\_margin\_pct,

    FN\_VALIDATE\_SETMEAL(sm.item\_id) AS validation\_status,

    (SELECT COUNT(\*) FROM ORDER\_ITEMS oi JOIN ORDERS o ON oi.order\_id = o.order\_id

     WHERE oi.item\_id = sm.item\_id AND o.order\_date > ADD\_MONTHS(SYSDATE, -3)) AS sales\_last\_3\_months

FROM MENU\_ITEMS sm

JOIN SET\_MEAL\_COMPONENTS smc ON sm.item\_id = smc.set\_meal\_id

JOIN MENU\_ITEMS mi ON smc.component\_id = mi.item\_id

WHERE sm.item\_type = 'SET\_MEAL'

GROUP BY sm.item\_id, sm.name, sm.base\_price

ORDER BY base\_margin\_pct DESC;

*----------------------------------------------------------------------------------------------*

*-- 7. QUERY: Order Type Comparison*

*----------------------------------------------------------------------------------------------*

*-- User transaction:*

*-- Compare performance metrics between set meals and à la carte orders across customer tiers,*

*-- including order counts, revenue, and customer engagement metrics.*

*-- Purpose:*

*-- To analyze differences in purchasing method between set meals and à la carte items*

*-- by customer membership tier, helping identify which order types drive more value*

*-- from different customer segments.*

*----------------------------------------------------------------------------------------------*

SELECT

    CASE WHEN mi.item\_type = 'SET\_MEAL' THEN 'Setmeal' ELSE 'À La Carte' END AS order\_type,

    mt.tier\_name AS customer\_tier,

    COUNT(DISTINCT oi.order\_id) AS order\_count,

    SUM(oi.price \* oi.quantity) AS total\_revenue,

    ROUND(AVG(oi.price \* oi.quantity), 2) AS avg\_order\_value,

    ROUND(SUM(oi.price \* oi.quantity) / NULLIF(COUNT(DISTINCT o.customer\_id), 0), 2) AS revenue\_per\_customer,

    ROUND(COUNT(DISTINCT oi.order\_id) / NULLIF(COUNT(DISTINCT o.customer\_id), 0), 2) AS orders\_per\_customer

FROM ORDER\_ITEMS oi

JOIN MENU\_ITEMS mi ON oi.item\_id = mi.item\_id

JOIN ORDERS o ON oi.order\_id = o.order\_id

JOIN CUSTOMERS c ON o.customer\_id = c.customer\_id

JOIN MEMBERSHIP\_TIERS mt ON c.tier\_id = mt.tier\_id

GROUP BY

    CASE WHEN mi.item\_type = 'SET\_MEAL' THEN 'Setmeal' ELSE 'À La Carte' END,

    mt.tier\_name

ORDER BY

    order\_type,

    CASE mt.tier\_name

        WHEN 'Diamond' THEN 1

        WHEN 'Platinum' THEN 2

        WHEN 'Gold' THEN 3

        WHEN 'Silver' THEN 4

        WHEN 'Bronze' THEN 5

    END;

*---------------------------------------------------------*

*-- Test à la carte order*

*---------------------------------------------------------*

*-- Purpose: Validate the query works with real order data*

*---------------------------------------------------------*

BEGIN

    PROC\_CREATE\_ALA\_CARTE\_ORDER(1001, '1,3', 'CARD');

    COMMIT;

EXCEPTION

    WHEN OTHERS THEN

        DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

        ROLLBACK;

END;

/

*------------------------------------------------------*

*-- Test set meal order*

*------------------------------------------------------*

BEGIN

    PROC\_CREATE\_SETMEAL\_ORDER(1002, 5, 'ONLINE');

    COMMIT;

EXCEPTION

    WHEN OTHERS THEN

        DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

        ROLLBACK;

END;

/

*------------------------------------------------------------------*

*-- Verify results*

*------------------------------------------------------------------*

*-- Purpose: Validate order calculations match expected amounts*

*------------------------------------------------------------------*

SELECT o.order\_id, o.total\_amount, o.discount\_amount, o.final\_amount,

       (SELECT SUM(price \* quantity) FROM ORDER\_ITEMS WHERE order\_id = o.order\_id) AS calc\_total,

       (SELECT SUM(discount\_applied \* quantity) FROM ORDER\_ITEMS WHERE order\_id = o.order\_id) AS calc\_discount

FROM ORDERS o

WHERE o.order\_id IN (

    SELECT MAX(order\_id) FROM ORDERS WHERE customer\_id = 1001

    UNION

    SELECT MAX(order\_id) FROM ORDERS WHERE customer\_id = 1002

);