TUNKU ABDUL RAHMAN UNIVERSITY OF MANAGEMENT AND TECHNOLOGY

# FACULTY OF COMPUTING AND INFORMATION TECHNOLOGY

## BACS3183 ADVANCED DATABASE MANAGEMENT

**Assignment**

**Semester 202501**

| | | |
|---|---|---|
| **Programme (Year, Sem & Group)** | : | e.g. **RSD2S3G5, RSW2S3G13** |
| **Date Submitted** | : | e.g. **22/4/2025** |

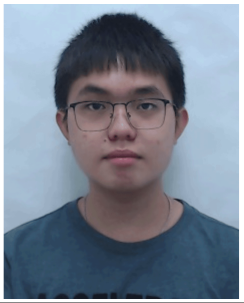
Team members:

| No | Name (Block Letters - Alphabetical Order) | Student ID | Marks |
|---|---|---|---|
| 1 | Chia Ming Yi | 24WMR09040 | |
| 2 | Daniel Tay Yi Fung | 24MWR09174 | |
| 3 | Kevin Lo Yung Khang | 24WMR09192 | |
| 4 | Lim Jun Wei | 24WMR09078 | |
| 5 | Ong Yi Xin | 24WMR09097 | |

## Declaration

We confirm that we have read and shall comply with all the terms and conditions of TAR UMT's plagiarism policy.

We declare that this assignment is free from all forms of plagiarism and for all intents and purposes is my own properly derived work.

| | | | | |
|---|---|---|---|---|
|  |  |  |  |  |
| Name:<br>Chia Ming Yi | Name:<br>Daniel Tay Yi Fung | Name:<br>Kevin Lo Yung Khang | Name:<br>Lim Jun Wei | Name:<br>Ong Yi Xin |
| Signature: | Signature: | Signature: | Signature: | Signature: |
| Date:23/2/2025 | Date: 23/2/2025 | Date: 23/2/2025 | Date: 23/2/2025 | Date: 23/2/2025 |

**Assignment Assessment Form**

| No | **Member Name** (Alphabetical order) | **Programme** (Year-Semester-Group) |
|---|---|---|
| 1. | Chia Ming Yi | RSD(2-3-5) |
| 2. | Daniel Tay Yi Fung | RSW(2-3-13) |
| 3. | Kevin Lo Yung Khang | RSW(2-3-13) |
| 4. | Lim Jun Wei | RSD(2-3-5) |
| 5. | Ong Yi Xin | RSD(2-3-5) |

| **Task No.** | **Task Descriptions** | **Weightage** | **Criteria** | **1** | **2** | **3** | **4** | **5** | **Comment** |
|---|---|---|---|---|---|---|---|---|---|
| 1 (CLO 1) | Entity Relationship Diagram | 10% | ● A complete ER data model in 3rd Normal Form.<br>● All primary keys, foreign keys, relationships and attributes must be clearly shown.<br>● Standalone system design will be given lower marks compare to online system design. | | | | | | |
| 2 (CLO 1) | Data Definition (DDL) | 10% | ● Relevant integrity constraints to ensure database integrity must be included.<br>● Necessary check constraints and default values to enforce business rules should also be included. | | | | | | |
| 3 (CLO 1) | Data records | 5% | ● Sufficient quality data records must be created for each table. | | | | | | |
| 4 (CLO 3) | Queries | 5% | ● Each team member is to design and produce two quality and useful queries for decision making at any two different management levels: strategic, tactical or operational.<br>● Single table queries are not allowed.<br>● Multiple table queries with aggregate functions (where appropriate) must be used.<br>● View(s) is/are ought to be incorporated, where necessary.<br>● These 2 queries cannot be used directly for report body. | | | | | | |
| | | 5% | | | | | | | |
| 5 (CLO 2) | Stored Procedures | 5% | ● Each team member is to design and create two stored procedures that cater for the use case scenarios for the system.<br>● Quality, usefulness and importance of the stored procedures must be considered. | | | | | | |
| | | 5% | | | | | | | |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 6<br>(CLO 2) | Triggers | 5% | • Each team member is to design and create two triggers that enforces system-wide business rules and policies.<br>• Quality, usefulness and functionality of the triggers must be considered. | | | | | | |
| | | 5% | | | | | | | |
| 7<br>(CLO 3) | Reports | 10% | • Each team member is to create two procedures to generate two reports (summary, detail and on demand basis reports) for the company.<br>• Parameter value(s) should be passed to the procedure, where necessary.<br>• Cursor must be used in report generation (prefer nested cursors or multiple cursors).<br>• Usefulness, complexity and presentation of the reports must be considered. | | | | | | |
| | | 10% | | | | | | | |
| 8<br>(CLO 2) | Extra Effort | 10% | • Sequences, indexes, views, output formatting, exceptions and/or functions, must be incorporated where necessary.<br>• Usefulness and application of each of the above to enhance the efficiency and effectiveness of the information system must be considered.<br>• Linking of all the tasks of every team member in creating a quality information system must be considered. | | | | | | |
| 9<br>(CLO 3) | Presentation & Participation | 15% | • Run the single script file from Task 2 – 3 to create the new system database on the lab server.<br>• Individual presentation on Task 4 – 8.<br>• Q & A<br>• Actively participate in class discussion. | | | | | | |
| **Assignment Marks / 100** | | | | | | | | | |

*CLO 1: Develop the relational database system with the appropriate integrity constraints and security control. (P3, PLO3)
*CLO 2: Design the solutions to issues pertaining to database efficiency and effectiveness using appropriate techniques. (C4, PLO2)
*CLO 3: Extract information from the database using efficient SQL query construct. (C4, PLO6)

# Table of Content

**Chapter 1 Background of the System** **5**
1.1 Type of System 5
1.2 List of Modules : 5
**Chapter 2** **6**
2.1 Business Rules And Assumption 6
Business Rules 6
Assumption 7
2.2 ERD 8
**Chapter 3 Data Definition** **10**
3.1 CREATE TABLE Statements 10
3.2 Sample Records (10 sample Records for each table) 21
**Chapter 4 Queries, Procedures, Triggers and Reports** **29**
4.1 Chia Ming Yi 29
4.2 Lim Jun Wei 42
4.3 Ong Yi Xin 56
4.4 Kevin Lo Yung Khang 72
4.5 Daniel Tay Yi Fung 87
**Chapter 5 Extra Effort Highlights** **95**
5.1 Chia Ming Yi 95
5.1.1 Views 95
5.1.2 Indexes 95
5.1.3 User Defined Functions 96
5.1.4 Sequence & Trigger 97
5.1.5 User Defined Exception 97
5.2 Lim Jun Wei 98
5.2.1 Views 98
5.2.2 Indexes 98
5.2.3 User Defined Functions 99
5.2.4 Job Scheduler 107
5.3.5 User Defined Exception 108
5.3 Ong Yi Xin 109
5.3.1 Views 109
5.3.2 Indexes 110
5.3.3 User Defined Functions 110
5.3.4 Sequence & Trigger 111
5.3.5 User Defined Exception 111
5.4 Kevin Lo Yung Khang 113
5.4.1 Views 113
5.4.2 Indexes 116
5.4.3 User Defined Functions 117
5.4.4 Sequence & Trigger 118
5.4.5 User Defined Exception 119

5.5 Daniel Tay Yi Fung 120
5.5.1 Views 120
5.5.2 Indexes 120
5.5.3 User Defined Functions 121
5.5.4 Sequence & Trigger 123
5.5.5 User Defined Exceptions 124

# Chapter 1 Background of the System

## 1.1 Type of System

Our library management system was developed as a **stand-alone system** to automate and streamline the day-to-day operations of a library. This means that it can **run independently** on a local computer or network **without** the need for a constant **Internet connection** or **server-based deployment**. All data is stored and processed locally, allowing the system to operate highly reliably in environments with limited or no Internet connectivity.

The system is a **Transaction Processing System (TPS)** with an added decision support element. The system is designed to meet the operational and tactical needs of library administrators, staff, and management. Traditionally, library operations such as membership tracking, book checkout, fine calculation and staff task management have been performed manually. Our system solves these problems by providing an integrated, user-friendly and easy-to-use system.

Traditionally, library operations such as membership tracking, book checkout, fine calculation and staff task management have been performed manually, which can lead to inefficiencies, errors and difficult record maintenance. Our system's can solve these problems by providing a user-friendly, integrated platform that streamlines all essential library tasks and improves business visibility.

## 1.2 List of Modules :

1. Membership Registration
2. Track Membership Validity
3. Borrowing and Returning Process
4. Book Reservation
5. Due Date and Overdue Tracking of Books
6. Fine Processing
7. Fine Invoicing
8. Loan Extension of Book Loan Period
9. Advanced Book Search
10. **Automate Member Credit Updates**
11. **Schedule Staff Tasks**
12. **Record Staff Attendance**
13. **Track Staff Performance and Shifts Schedules**

# Chapter 2

## 2.1 Business Rules And Assumption

### Business Rules

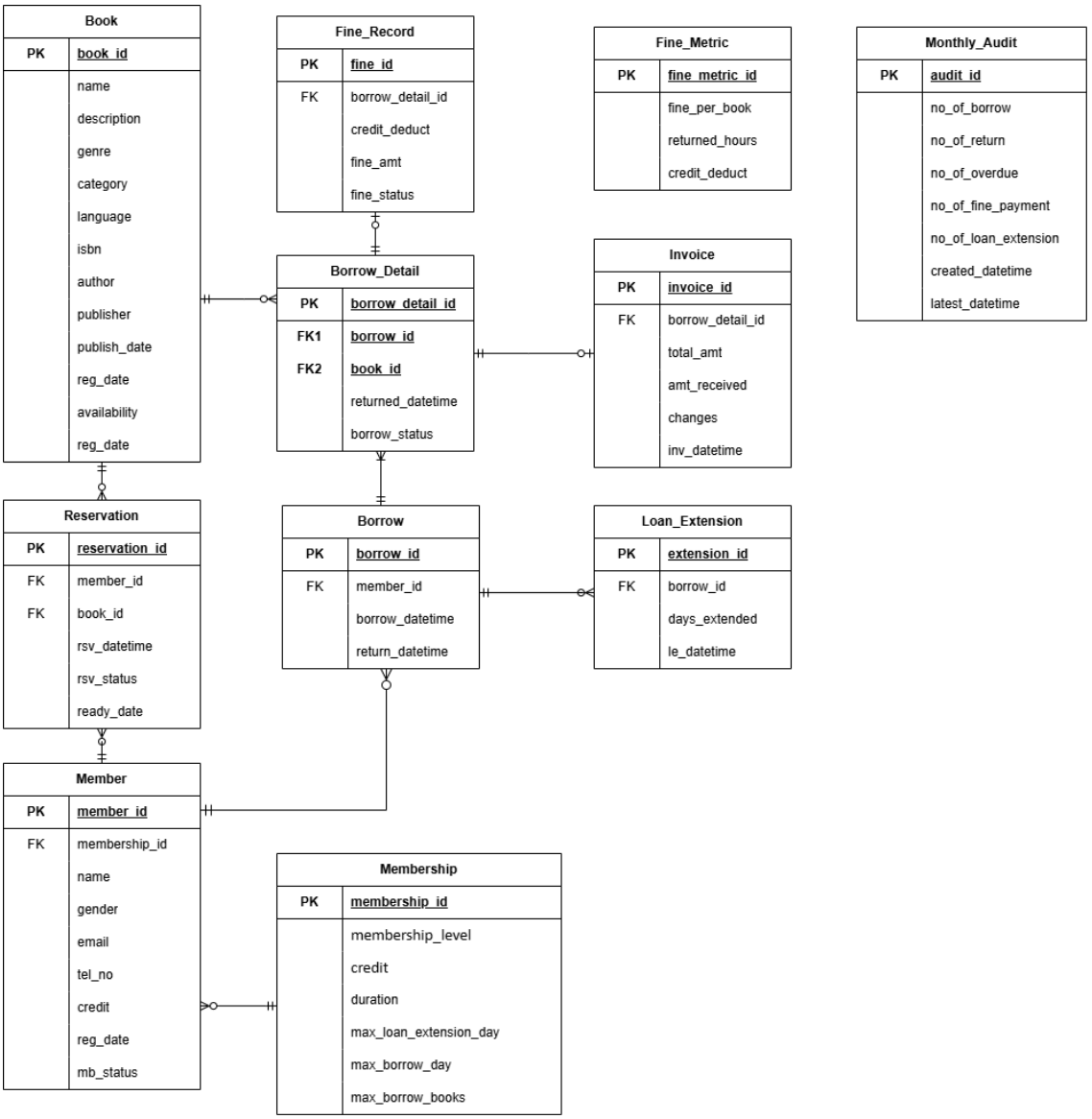
1. System operation hours start from 8.30 a.m. to 6.30 p.m. except Sunday and public holidays.
2. All the books which have been borrowed by members must be returned within 7 days when there is no loan extension applied.
3. The member who did not return the borrowed books within 7 days without applying for a loan extension will have to pay the fine.
4. The available loan extension for books will be provided based on the member's membership level.
5. Each staff can have zero or many schedules and each schedule is assigned to one and only one staff. (Zero to Many)
6. Each task can be assigned to zero or many schedules and each schedule consists of one and only one task. (One to Many)
7. Each task may contain zero or many auto assignments and each auto assign consists of one and only one task. (One to Many)
8. Each staff is assigned to zero or many auto assignments and each auto assignment belongs to one and only one staff. (One to Many)
9. Each schedule has zero or many schedule shifting and each schedule shifting is only applied on one and only one schedule. (Zero to Many)
10. Each schedule has one or many task progress records and each task progress is recorded for one and only one schedule. (One to Many)
11. Each book has zero or many borrowed details and each borrowed detail is created for one and only one book. (Zero to Many)
12. Each book has zero or many reservations and each reservation specifies one and only one book. (One to Many)
13. Each member has zero or many reservations and each reservation is made by one and only one member. (Zero to Many)
14. Each member must own one and only one membership while each membership can be assigned to zero or many members. (One to Many)
15. Each member has zero to many borrowed records and each borrowed record belongs to one and only one member. (One to Many)
16. Each borrow record has one or many borrow details and each borrow detail belongs to one and only one borrow record. (One to Many)
17. Each borrow record has zero or many loan extensions and each loan extension is applied on one and only one borrow record. (Zero to Many)
18. Each borrow detail has zero or one fine record and each fine record can be made for one and only one borrow detail. (Zero to One)
19. Each invoice belongs to one and only one borrow record and each borrow record exists in one and only one invoice. (One to One)
20. System operation hours start from 8.30 a.m. to 6.30 p.m. except Sunday and public holidays.
21. All the books which have been borrowed by members must be returned within 7 days when there is no loan extension applied.
22. The member who did not return the borrowed books within 7 days without applying for a loan extension will have to pay the fine.
23. The available loan extension for books will be provided based on the member's membership level.
24. Each staff can have zero or many schedules and each schedule is assigned to one and only one staff. (Zero to Many)

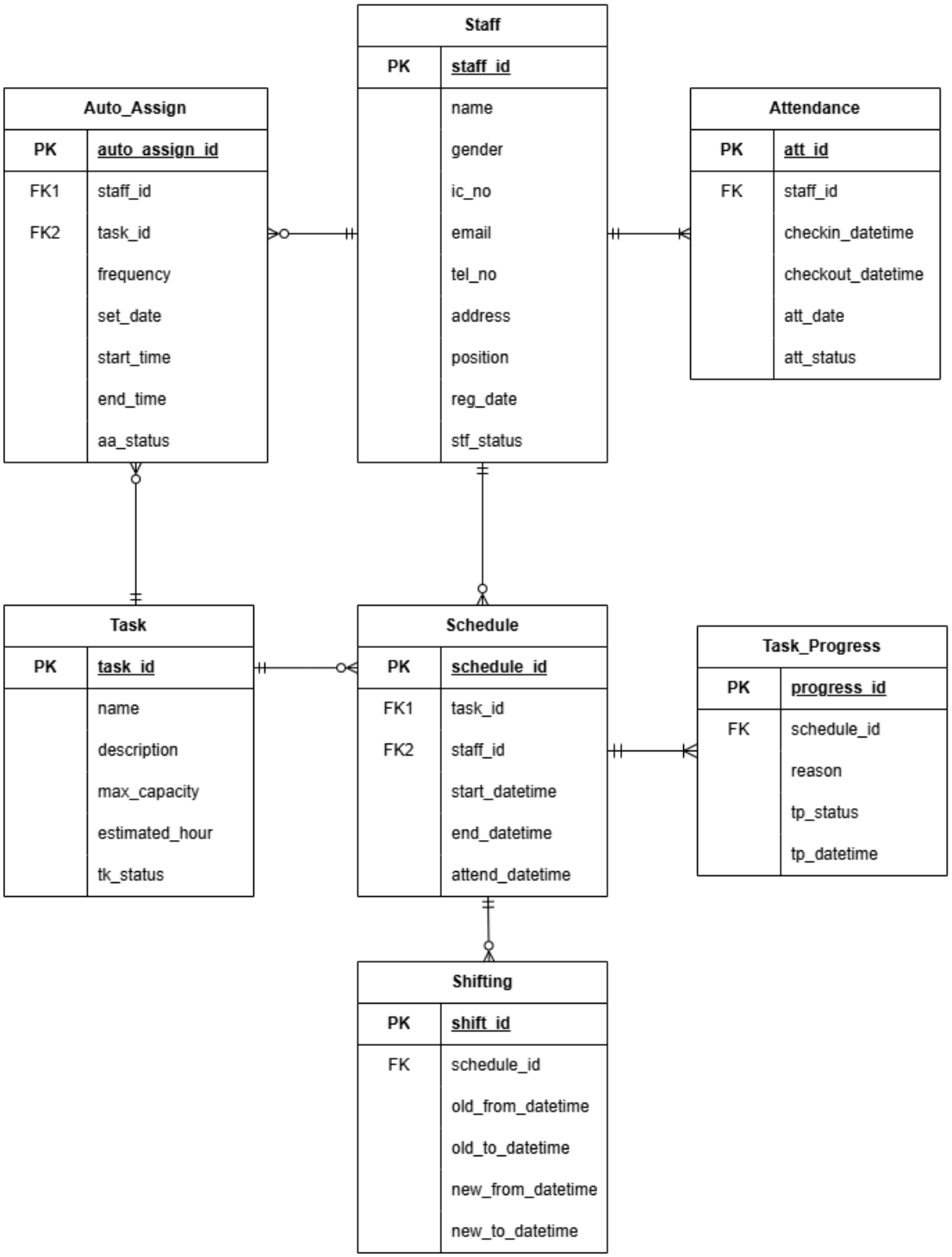
25. Each task can be assigned to zero or many schedules and each schedule consists of one and only one task. (One to Many)
26. Each task may contain zero or many auto assignments and each auto assign consists of one and only one task. (One to Many)
27. Each staff is assigned to zero or many auto assignments and each auto assignment belongs to one and only one staff. (One to Many)
28. Each schedule has zero or many schedule shifting and each schedule shifting is only applied on one and only one schedule. (Zero to Many)
29. Each schedule has one or many task progress records and each task progress is recorded for one and only one schedule. (One to Many)
30. Each book has zero or many borrowed details and each borrowed detail is created for one and only one book. (Zero to Many)
31. Each book has zero or many reservations and each reservation specifies one and only one book. (One to Many)
32. Each member has zero or many reservations and each reservation is made by one and only one member. (Zero to Many)
33. Each member must own one and only one membership while each membership can be assigned to zero or many members. (One to Many)
34. Each member has zero to many borrowed records and each borrowed record belongs to one and only one member. (One to Many)
35. Each borrow record has one or many borrow details and each borrow detail belongs to one and only one borrow record. (One to Many)
36. Each borrow record has zero or many loan extensions and each loan extension is applied on one and only one borrow record. (Zero to Many)
37. Each borrow detail has zero or one fine record and each fine record can be made for one and only one borrow detail. (Zero to One)
38. Each invoice belongs to one and only one borrow record and each borrow record exists in one and only one invoice. (One to One)

## Assumption

1. Customers must register as a member before they can borrow or reserve the books.
2. Members must borrow and return the books at the library counter.
3. Each staff must have his or her respective position such as manager, librarian, IT staff, facility maintainer.
4. If the books borrowed by a member are returned late, the member must pay the fine.
5. Each type of book only has one copy and the book will no longer be available for borrowing when it is borrowed by another member.
6. The fine payment can only be made via cash, credit card or TnG eWallet at the library counter.
7. The reservation status will be switched from ‘READY’ into ‘CANCELLED’ if the member has not collected the book within 2 days.
8. All the membership records will no longer be modified once it has been created.

## 2.2 ERD


Book
PK book_id
name
description
genre
category
language
isbn
author
publisher
publish_date
reg_date
availability
reg_date
Fine_Record
PK fine_id
FK borrow_detail_id
credit_deduct
fine_amt
fine_status
Fine_Metric
PK fine_metric_id
fine_per_book
returned_hours
credit_deduct
Monthly_Audit
PK audit_id
no_of_borrow
no_of_return
no_of_overdue
no_of_fine_payment
no_of_loan_extension
created_datetime
latest_datetime
Borrow_Detail
PK borrow_detail_id
FK1 borrow_id
FK2 book_id
returned_datetime
borrow_status
Invoice
PK invoice_id
FK borrow_detail_id
total_amt
amt_received
changes
inv_datetime
Reservation
PK reservation_id
FK member_id
FK book_id
rsv_datetime
rsv_status
ready_date
Borrow
PK borrow_id
FK member_id
borrow_datetime
return_datetime
Loan_Extension
PK extension_id
FK borrow_id
days_extended
le_datetime
Member
PK member_id
FK membership_id
name
gender
email
tel_no
credit
reg_date
mb_status
Membership
PK membership_id
membership_level
credit
duration
max_loan_extension_day
max_borrow_day
max_borrow_books


Staff
PK staff_id
name
gender
ic_no
email
tel_no
address
position
reg_date
stf_status
Auto_Assign
PK auto_assign_id
FK1 staff_id
FK2 task_id
frequency
set_date
start_time
end_time
aa_status
Attendance
PK att_id
FK staff_id
checkin_datetime
checkout_datetime
att_date
att_status
Task
PK task_id
name
description
max_capacity
estimated_hour
tk_status
Schedule
PK schedule_id
FK1 task_id
FK2 staff_id
start_datetime
end_datetime
attend_datetime
Task_Progress
PK progress_id
FK schedule_id
reason
tp_status
tp_datetime
Shifting
PK shift_id
FK schedule_id
old_from_datetime
old_to_datetime
new_from_datetime
new_to_datetime

# Chapter 3 Data Definition

## 3.1 CREATE TABLE Statements

```
-- DROP TABLE Task_Progress;
-- DROP TABLE Shifting;
-- DROP TABLE Schedule;
-- DROP TABLE Auto_Assign;
-- DROP TABLE Attendance;
-- DROP TABLE Task;
-- DROP TABLE Staff;
-- DROP TABLE Fine_Record;
-- DROP TABLE Invoice;
-- DROP TABLE Borrow_Detail;
-- DROP TABLE Loan_Extension;
-- DROP TABLE Borrow;
-- DROP TABLE Reservation;
-- DROP TABLE Member;
-- DROP TABLE Book;
-- DROP TABLE Fine_Metrics;
-- DROP TABLE Membership IF EXISTS;

-- Drop All Tables
BEGIN
	FOR t IN (
		SELECT table_name FROM user_tables
		WHERE table_name IN (
		'TASK_PROGRESS',
		'SHIFTING',
		'SCHEDULE',
		'AUTO_ASSIGN',
		'ATTENDANCE',
		'TASK',
		'STAFF',
		'FINE_RECORD',
		'INVOICE',
		'BORROW_DETAIL',
		'LOAN_EXTENSION',
		'BORROW',
		'RESERVATION',
		'MEMBER',
		'BOOK',
		'FINE_METRIC',
		'MEMBERSHIP'
		)
	) LOOP
	EXECUTE IMMEDIATE 'DROP TABLE ' || t.table_name || ' CASCADE CONSTRAINTS';
	END LOOP;
END;
/

-- Drop All Sequence
BEGIN
	FOR s IN (
		SELECT sequence_name FROM user_sequences
		WHERE sequence_name IN (
		'MEMBERSHIP_SEQ',
		'FINE_METRIC_SEQ',
		'BOOK_SEQ',
		'MEMBER_SEQ',
		'RESERVATION_SEQ',
		'BORROW_SEQ',
		'BORROW_DETAIL_SEQ',
		'EXTENSION_SEQ',
		'INVOICE_SEQ',
```

```
                    'FINE_SEQ',
                    'STAFF_SEQ',
                    'TASK_SEQ',
                    'ATTENDANCE_SEQ',
                    'AA_SEQ',
                    'SCHEDULE_SEQ',
                    'SHIFT_SEQ',
                    'TP_SEQ'
                    )
        ) LOOP
        EXECUTE IMMEDIATE 'DROP SEQUENCE ' || s.sequence_name;
        END LOOP;
END;
/

-- Drop All Trigger
BEGIN
        FOR trigger_rec IN (
                SELECT trigger_name
                FROM user_triggers
                WHERE trigger_name IN (
                'MEMBERSHIP_BEFORE_INSERT',
                'FINE_METRIC_BEFORE_INSERT',
                'BOOK_BEFORE_INSERT',
                'MEMBER_BEFORE_INSERT',
                'RESERVATION_BEFORE_INSERT',
                'BORROW_BEFORE_INSERT',
                'BORROW_DETAIL_BEFORE_INSERT',
                'EXTENSION_BEFORE_INSERT',
                'INVOICE_BEFORE_INSERT',
                'FINE_BEFORE_INSERT',
                'STAFF_BEFORE_INSERT',
                'TASK_BEFORE_INSERT',
                'ATT_BEFORE_INSERT',
                'AA_BEFORE_INSERT',
                'SCHEDULE_BEFORE_INSERT',
                'SHIFT_BEFORE_INSERT',
                'TP_BEFORE_INSERT'
                )
        ) LOOP
    EXECUTE IMMEDIATE 'DROP TRIGGER ' || trigger_rec.trigger_name;
        END LOOP;
END;
/


--Membership
CREATE TABLE Membership (
        membership_id VARCHAR2(6) NOT NULL PRIMARY KEY,
        membership_level NUMBER(3) NOT NULL,
        credit NUMBER(5) NOT NULL,
        duration_day NUMBER(5) NOT NULL,
        max_loan_extension_day NUMBER(5) NOT NULL,
        max_borrow_day NUMBER(5) NOT NULL,
        max_borrow_book NUMBER (5) NOT NULL,
        CHECK (membership_level >=0),
        CHECK (duration_day >= 0),
        CHECK (max_loan_extension_day  >= 0),
        CHECK (max_borrow_day >= 0),
        CHECK (max_borrow_book >= 0)
);

--Fine_Metrics
CREATE TABLE Fine_Metric (
        fine_metric_id VARCHAR2(6) NOT NULL PRIMARY KEY,
        fine_per_book NUMBER(7,2) NOT NULL,
```

```
	returned_hours NUMBER(5) NOT NULL,
	credit_deduct NUMBER(3) NOT NULL,
	CHECK(fine_per_book >= 0.1),
	CHECK(returned_hours > 0),
	CHECK(credit_deduct > 0 AND credit_deduct <= 100)
);

--Book
CREATE TABLE Book (
	book_id VARCHAR2(6) PRIMARY KEY,
	name VARCHAR2(50) NOT NULL,
	description VARCHAR2(150) NOT NULL,
	genre VARCHAR2(150) NOT NULL CHECK (genre IN ('fiction', 'non-fiction', 'academic')),
	category VARCHAR2(150) NOT NULL,
	language VARCHAR2(50) NOT NULL CHECK (language IN ('EN', 'ZH', 'MS', 'JA', 'KO')),
	isbn VARCHAR2(20) UNIQUE NOT NULL,
	author VARCHAR2(50) NOT NULL,
	publisher VARCHAR2(50) NOT NULL,
	publish_date DATE NOT NULL,
	reg_date DATE NOT NULL,
	availability CHAR NOT NULL CHECK (availability IN ('Y', 'N')),
	bk_status VARCHAR2(10) CHECK(Bk_status IN ('Active', 'Inactive')) NOT NULL,
	CHECK (
		(genre = 'fiction' AND category IN ('Adventure', 'Fantasy', 'Science Fiction', 'Mystery and
Thriller', 'Horror', 'Historical Fiction'))
		OR
		(genre = 'non-fiction' AND category IN ('Biography and Autobiography', 'Business and
Economics', 'Psychology', 'Health and Wellness', 'Philosophy', 'Religion and Spirituality', 'Politics and Current
Affairs'))
		OR
		(genre = 'academic' AND category IN ('Textbooks', 'Science and Technology', 'Engineering',
'Medicine', 'Law', 'History', 'Arts and Music', 'Computer Science', 'Travel and Tourism'))
	)
);

-- Member
CREATE TABLE Member (
	member_id VARCHAR2(6) PRIMARY KEY,
	membership_id VARCHAR2(6) NOT NULL,
	name VARCHAR2(50) NOT NULL,
	gender VARCHAR2(1) CHECK (gender IN ('F', 'M')) NOT NULL,
	email VARCHAR2(100) UNIQUE NOT NULL,
	tel_no VARCHAR2(12) NOT NULL,
	credit NUMBER(5) NOT NULL CHECK (credit BETWEEN 0 AND 100),
	reg_date DATE NOT NULL,
	mb_status NUMBER(1) CHECK (mb_status IN (0, 1)) NOT NULL,
	CONSTRAINT chk_email_format CHECK (REGEXP_LIKE(email,
'^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$')),
	CONSTRAINT chk_tel_format CHECK (REGEXP_LIKE(tel_no, '^01[0-9]-\d{7,8}$')),
	FOREIGN KEY (membership_id) REFERENCES Membership(membership_id)
);


-- Reservation
CREATE TABLE Reservation (
	reservation_id VARCHAR2(6) PRIMARY KEY,
	member_id VARCHAR2(6) NOT NULL,
	book_id VARCHAR2(6) NOT NULL,
	rsv_date DATE NOT NULL,
	rsv_status VARCHAR2(20) NOT NULL,
	CHECK (rsv_status IN ('RESERVED', 'READY', 'COMPLETED', 'CANCELLED')),
	FOREIGN KEY(member_id) REFERENCES Member(Member_id),
	FOREIGN KEY(book_id) REFERENCES Book(Book_id)
);

-- Borrow
```

```
CREATE TABLE Borrow (
	borrow_id VARCHAR2(6) PRIMARY KEY,
	member_id VARCHAR2(6) NOT NULL,
	borrow_datetime DATE NOT NULL,
	return_datetime DATE,
	FOREIGN KEY(member_id) REFERENCES Member(member_id)
);

-- Loan_Extension
CREATE TABLE Loan_Extension (
	extension_id VARCHAR2(6) PRIMARY KEY,
	borrow_id VARCHAR2(6) NOT NULL,
	days_extended NUMBER(2) NOT NULL,
	le_datetime DATE NOT NULL,
	CHECK (days_extended >= 1),
	FOREIGN KEY(borrow_id) REFERENCES Borrow(borrow_id)
);

-- Borrow_Detail
CREATE TABLE Borrow_Detail (
	borrow_detail_id VARCHAR2(6) NOT NULL PRIMARY KEY,
	borrow_id VARCHAR2(6) NOT NULL,
	book_id VARCHAR2(6) NOT NULL,
	returned_datetime DATE,
	borrow_status VARCHAR2(20),
	CHECK (borrow_status IN ('Borrowed', 'Returned', 'Missing', 'Damaged')),
	FOREIGN KEY(borrow_id) REFERENCES Borrow(borrow_id),
	FOREIGN KEY(book_id) REFERENCEs Book(book_id)
);

-- Invoice
CREATE TABLE Invoice (
	invoice_id VARCHAR2(6) NOT NULL PRIMARY KEY,
	borrow_detail_id VARCHAR2(6) NOT NULL,
	total_amt NUMBER(7,2) NOT NULL,
	inv_datetime DATE NOT NULL,
	CHECK (total_amt >= 0.01),
	FOREIGN KEY (borrow_detail_id) REFERENCEs Borrow_Detail(borrow_detail_id)
);

-- Fine_Record
	CREATE TABLE Fine_Record (
	fine_id VARCHAR2(6) NOT NULL PRIMARY KEY,
	borrow_detail_id VARCHAR2(6) NOT NULL,
	credit_deduct NUMBER(5) NOT NULL,
	fine_amt NUMBER(7,2) NOT NULL,
	CHECK(fine_amt >-0.01),
	FOREIGN KEY(borrow_detail_id ) REFERENCES Borrow_Detail(borrow_detail_id )
);


-- Sequence for Membership
CREATE SEQUENCE membership_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

-- Auto-Increment for membership_id
CREATE OR REPLACE TRIGGER membership_before_insert
	BEFORE INSERT ON Membership
	FOR EACH ROW
	BEGIN
		IF :NEW.membership_id IS NULL THEN
		SELECT 'MS' || LPAD(membership_seq.NEXTVAL, 4, '0') INTO :NEW.membership_id
FROM dual;
```

```
        END IF;
END;
/

-- Sequence for Fine_Metric
CREATE SEQUENCE fine_metric_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

-- Auto-Increment for fine_metric_id
CREATE OR REPLACE TRIGGER fine_metric_before_insert
        BEFORE INSERT ON Fine_Metric
        FOR EACH ROW
        BEGIN
                IF :NEW.fine_metric_id IS NULL THEN
                SELECT 'FM' || LPAD(fine_metric_seq.NEXTVAL, 4, '0') INTO :NEW.fine_metric_id
FROM dual;
        END IF;
END;
/

-- Sequence for Book
CREATE SEQUENCE book_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

-- Auto-Increment for book_id
CREATE OR REPLACE TRIGGER book_before_insert
        BEFORE INSERT ON Book
        FOR EACH ROW
        BEGIN
                IF :NEW.book_id IS NULL THEN
                SELECT 'BK' || LPAD(book_seq.NEXTVAL, 4, '0') INTO :NEW.book_id FROM dual;
        END IF;
END;
/

-- Sequence for Member
CREATE SEQUENCE member_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

-- Auto-Increment for member_id
CREATE OR REPLACE TRIGGER member_before_insert
        BEFORE INSERT ON Member
        FOR EACH ROW
                BEGIN
                        IF :NEW.member_id IS NULL THEN
                        SELECT 'MB' || LPAD(member_seq.NEXTVAL, 4, '0') INTO :NEW.member_id
FROM dual;
                END IF;
END;
/

-- Sequence for Reservation
CREATE SEQUENCE reservation_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;
```

```
-- Auto-Increment for reservation_id
CREATE OR REPLACE TRIGGER reservation_before_insert
	BEFORE INSERT ON Reservation
	FOR EACH ROW
	BEGIN
		IF :NEW.reservation_id IS NULL THEN
		SELECT 'RS' || LPAD(reservation_seq.NEXTVAL, 4, '0') INTO :NEW.reservation_id
FROM dual;
	END IF;
END;
/

-- Sequence for Borrow
CREATE SEQUENCE Borrow_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

-- Auto-Increment for borrow_id
CREATE OR REPLACE TRIGGER borrow_before_insert
	BEFORE INSERT ON Borrow
	FOR EACH ROW
	BEGIN
		IF :NEW.borrow_id IS NULL THEN
		SELECT 'BR' || LPAD(borrow_seq.NEXTVAL, 4, '0') INTO :NEW.borrow_id FROM dual;
	END IF;
END;
/

-- Sequence for Borrow_Detail
CREATE SEQUENCE Borrow_Detail_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

-- Auto-Increment for borrow_detail_id
CREATE OR REPLACE TRIGGER borrow_detail_before_insert
	BEFORE INSERT ON Borrow_Detail
	FOR EACH ROW
	BEGIN
		IF :NEW.borrow_detail_id IS NULL THEN
		SELECT 'BD' || LPAD(borrow_detail_seq.NEXTVAL, 4, '0') INTO :NEW.borrow_detail_id
FROM dual;
	END IF;
END;
/

-- Sequence for Loan_Extension
CREATE SEQUENCE extension_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

-- Auto-Increment for extension_id
CREATE OR REPLACE TRIGGER extension_before_insert
	BEFORE INSERT ON Loan_Extension
	FOR EACH ROW
	BEGIN
		IF :NEW.extension_id IS NULL THEN
		SELECT 'LE' || LPAD(extension_seq.NEXTVAL, 4, '0') INTO :NEW.extension_id FROM
dual;
END IF;
```

```
END;
/

-- Sequence for Invoice
CREATE SEQUENCE invoice_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

-- Auto-Increment for invoice_id
CREATE OR REPLACE TRIGGER invoice_before_insert
	BEFORE INSERT ON Invoice
	FOR EACH ROW
	BEGIN
		IF :NEW.invoice_id IS NULL THEN
		SELECT 'IV' || LPAD(invoice_seq.NEXTVAL, 4, '0') INTO :NEW.invoice_id FROM dual;
	END IF;
END;
/

-- Sequence for Fine_Record
CREATE SEQUENCE fine_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

-- Auto-Increment for fine_id
CREATE OR REPLACE TRIGGER fine_before_insert
	BEFORE INSERT ON Fine_Record
	FOR EACH ROW
	BEGIN
		IF :NEW.fine_id IS NULL THEN
		SELECT 'FN' || LPAD(fine_seq.NEXTVAL, 4, '0') INTO :NEW.fine_id FROM dual;
END IF;
	END;
/

BEGIN
	FOR t IN (
		SELECT table_name FROM user_tables
		WHERE table_name IN (
		'TASK_PROGRESS',
		'SHIFTING',
		'SCHEDULE',
		'AUTO_ASSIGN',
		'ATTENDANCE',
		'TASK',
		'STAFF',
		'FINE_RECORD',
		'INVOICE',
		'BORROW_DETAIL',
		'LOAN_EXTENSION',
		'BORROW',
		'RESERVATION',
		'MEMBER',
		'BOOK',
		'FINE_METRIC',
		'MEMBERSHIP'
		)
	) LOOP
	EXECUTE IMMEDIATE 'DROP TABLE ' || t.table_name || ' CASCADE CONSTRAINTS';
	END LOOP;
END;
/
```

```
-- Drop All Sequence
BEGIN
	FOR s IN (
		SELECT sequence_name FROM user_sequences
		WHERE sequence_name IN (
		'MEMBERSHIP_SEQ',
		'FINE_METRIC_SEQ',
		'BOOK_SEQ',
		'MEMBER_SEQ',
		'RESERVATION_SEQ',
		'BORROW_SEQ',
		'BORROW_DETAIL_SEQ',
		'EXTENSION_SEQ',
		'INVOICE_SEQ',
		'FINE_SEQ',
		'STAFF_SEQ',
		'TASK_SEQ',
		'ATTENDANCE_SEQ',
		'AA_SEQ',
		'SCHEDULE_SEQ',
		'SHIFT_SEQ',
		'TP_SEQ'
		)
	) LOOP
	EXECUTE IMMEDIATE 'DROP SEQUENCE ' || s.sequence_name;
	END LOOP;
END;
/

-- Drop All Trigger
BEGIN
	FOR trigger_rec IN (
		SELECT trigger_name
		FROM user_triggers
		WHERE trigger_name IN (
		'MEMBERSHIP_BEFORE_INSERT',
		'FINE_METRIC_BEFORE_INSERT',
		'BOOK_BEFORE_INSERT',
		'MEMBER_BEFORE_INSERT',
		'RESERVATION_BEFORE_INSERT',
		'BORROW_BEFORE_INSERT',
		'BORROW_DETAIL_BEFORE_INSERT',
		'EXTENSION_BEFORE_INSERT',
		'INVOICE_BEFORE_INSERT',
		'FINE_BEFORE_INSERT',
		'STAFF_BEFORE_INSERT',
		'TASK_BEFORE_INSERT',
		'ATT_BEFORE_INSERT',
		'AA_BEFORE_INSERT',
		'SCHEDULE_BEFORE_INSERT',
		'SHIFT_BEFORE_INSERT',
		'TP_BEFORE_INSERT'
		)
	) LOOP
    EXECUTE IMMEDIATE 'DROP TRIGGER ' || trigger_rec.trigger_name;
	END LOOP;
END;
/

-- Track Staff Performance and Shift Schedules
-- Staff
CREATE TABLE Staff (
	staff_id VARCHAR(6) NOT NULL PRIMARY KEY,
	name VARCHAR(50) NOT NULL,
	gender VARCHAR(1) NOT NULL,
```

```
	ic_no VARCHAR(14) NOT NULL,
	email VARCHAR(50) NOT NULL,
	tel_no VARCHAR(11) NOT NULL,
	address VARCHAR(150) NOT NULL,
	position VARCHAR(30) NOT NULL,
	reg_date DATE NOT NULL,
	stf_status VARCHAR(20) NOT NULL,
	CHECK(gender IN ('F', 'M')),
	CHECK (REGEXP_LIKE(email, '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$')),
	CHECK (REGEXP_LIKE(tel_no, '^01[0-9]-\d{7,8}$')),
	CHECK(stf_status IN ('Active', 'Inactive'))
);

-- Task
CREATE TABLE Task (
	task_id VARCHAR(6) NOT NULL PRIMARY KEY,
	description VARCHAR(150) NOT NULL,
	max_capacity NUMBER(2) NOT NULL,
	estimated_hour NUMBER(2) NOT NULL,
	tk_status VARCHAR(20) NOT NULL,
	CHECK(max_capacity > 0),
	CHECK(estimated_hour > 0 AND estimated_hour <= 12)
);

-- Attendance
CREATE TABLE Attendance (
	att_id VARCHAR(6) NOT NULL PRIMARY KEY,
	staff_id VARCHAR(6) NOT NULL,
	checkin_datetime DATE NOT NULL,
	checkout_datetime DATE NOT NULL,
	att_date DATE NOT NULL,
	att_status VARCHAR(20) NOT NULL,
	CHECK (checkin_datetime < checkout_datetime),
	CHECK(att_status IN ('Present', 'Absent', 'Leave')),
	FOREIGN KEY (staff_id) REFERENCES Staff(staff_id)
);

-- Auto_Assign
CREATE TABLE Auto_Assign (
	auto_assign_id VARCHAR(6) NOT NULL PRIMARY KEY,
	staff_id VARCHAR(6) NOT NULL,
	task_id VARCHAR(6) NOT NULL,
	frequency VARCHAR(10) NOT NULL,
	set_date DATE NOT NULL,
	start_time DATE NOT NULL,
	end_time DATE NOT NULL,
	aa_status VARCHAR(20) NOT NULL,
	CHECK(frequency IN ('Daily', 'Weekly', 'Monthly')),
	CHECK(start_time < end_time),
	CHECK(aa_status IN ('Active', 'Inactive')),
	FOREIGN KEY (staff_id) REFERENCES Staff(staff_id),
	FOREIGN KEY (task_id) REFERENCES Task(task_id)
);

-- Schedule
CREATE TABLE Schedule (
	schedule_id VARCHAR(6) NOT NULL PRIMARY KEY,
	task_id VARCHAR(6) NOT NULL,
	staff_id VARCHAR(6) NOT NULL,
	start_datetime DATE NOT NULL,
	end_datetime DATE NOT NULL,
	attend_datetime DATE NOT NULL,
	CHECK(start_datetime < end_datetime),
	FOREIGN KEY (task_id) REFERENCES Task(task_id),
	FOREIGN KEY (staff_id) REFERENCES Staff(staff_id)
);
```

```
-- Shifting
CREATE TABLE Shifting (
	shift_id VARCHAR(6) NOT NULL PRIMARY KEY,
	schedule_id VARCHAR(6) NOT NULL,
	old_from_datetime DATE NOT NULL,
	old_to_datetime DATE NOT NULL,
	new_from_datetime DATE NOT NULL,
	new_to_datetime DATE NOT NULL,
	CHECK(old_from_datetime < old_to_datetime),
	CHECK(new_from_datetime < new_to_datetime),
	FOREIGN KEY (schedule_id) REFERENCES Schedule(schedule_id)
);

-- Task_Progress
CREATE TABLE Task_Progress (
	progress_id VARCHAR(6) NOT NULL PRIMARY KEY,
	schedule_id VARCHAR(6) NOT NULL,
	reason VARCHAR(100),
	tp_status VARCHAR(20) NOT NULL,
	tp_datetime DATE NOT NULL,
	CHECK(tp_status IN ('Scheduled', 'Ongoing', 'Completed')),
	FOREIGN KEY (schedule_id) REFERENCES Schedule(schedule_id)
);

-- Sequence for Staff
CREATE SEQUENCE staff_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

-- Auto-Increment for staff_id
CREATE OR REPLACE TRIGGER staff_before_insert
	BEFORE INSERT ON Staff
	FOR EACH ROW
	BEGIN
		IF :NEW.staff_id IS NULL THEN
		SELECT 'SF' || LPAD(staff_seq.NEXTVAL, 4, '0') INTO :NEW.staff_id FROM dual;
	END IF;
END;
/

-- Sequence for Task
CREATE SEQUENCE task_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

-- Auto-Increment for task_id
CREATE OR REPLACE TRIGGER task_before_insert
	BEFORE INSERT ON Task
	FOR EACH ROW
	BEGIN
		IF :NEW.task_id IS NULL THEN
		SELECT 'TK' || LPAD(task_seq.NEXTVAL, 4, '0') INTO :NEW.task_id FROM dual;
	END IF;
END;
/

-- Sequence for Attendance
CREATE SEQUENCE attendance_seq
START WITH 1
INCREMENT BY 1
NOCACHE
```

```
NOCYCLE;

-- Auto-Increment for att_id
CREATE OR REPLACE TRIGGER att_before_insert
	BEFORE INSERT ON Attendance
	FOR EACH ROW
	BEGIN
		IF :NEW.att_id IS NULL THEN
		SELECT 'AT' || LPAD(attendance_seq.NEXTVAL, 4, '0') INTO :NEW.att_id FROM dual;
	END IF;
END;
/

-- Sequence for Auto_Assign
CREATE SEQUENCE aa_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

-- Auto-Increment for auto_assign_id
CREATE OR REPLACE TRIGGER aa_before_insert
	BEFORE INSERT ON Auto_Assign
	FOR EACH ROW
	BEGIN
		IF :NEW.auto_assign_id IS NULL THEN
		SELECT 'AA' || LPAD(aa_seq.NEXTVAL, 4, '0') INTO :NEW.auto_assign_id FROM dual;
	END IF;
END;
/

-- Sequence for Schedule
CREATE SEQUENCE schedule_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

-- Auto-Increment for schedule_id
CREATE OR REPLACE TRIGGER schedule_before_insert
	BEFORE INSERT ON Schedule
	FOR EACH ROW
	BEGIN
		IF :NEW.schedule_id IS NULL THEN
		SELECT 'SC' || LPAD(schedule_seq.NEXTVAL, 4, '0') INTO :NEW.schedule_id FROM
dual;
	END IF;
END;
/

-- Sequence for Shifting
CREATE SEQUENCE shift_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

-- Auto-Increment for shift_id
CREATE OR REPLACE TRIGGER shift_before_insert
	BEFORE INSERT ON Shifting
	FOR EACH ROW
	BEGIN
		IF :NEW.shift_id IS NULL THEN
		SELECT 'SH' || LPAD(shift_seq.NEXTVAL, 4, '0') INTO :NEW.shift_id FROM dual;
	END IF;
END;
```

```
/

-- Sequence for Task_Progress
CREATE SEQUENCE tp_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

-- Auto-Increment for progress_id
CREATE OR REPLACE TRIGGER tp_before_insert
        BEFORE INSERT ON Task_Progress
        FOR EACH ROW
        BEGIN
                IF :NEW.progress_id IS NULL THEN
                SELECT 'TP' || LPAD(tp_seq.NEXTVAL, 4, '0') INTO :NEW.progress_id FROM dual;
        END IF;
END;
/
```

## 3.2 Sample Records (10 sample Records for each table)

```
-- Insert into Membership Table
INSERT INTO Membership (membership_level, credit, duration_day, max_loan_extension_day,
max_borrow_day, max_borrow_book) VALUES (1, 60, 0, 3, 2, 2);
INSERT INTO Membership (membership_level, credit, duration_day, max_loan_extension_day,
max_borrow_day, max_borrow_book) VALUES (2, 80, 30, 5, 3, 3);
INSERT INTO Membership (membership_level, credit, duration_day, max_loan_extension_day,
max_borrow_day, max_borrow_book) VALUES (3, 100, 60, 7, 3, 3);
INSERT INTO Membership (membership_level, credit, duration_day, max_loan_extension_day,
max_borrow_day, max_borrow_book) VALUES (4, 100, 90, 10, 4, 4);
INSERT INTO Membership (membership_level, credit, duration_day, max_loan_extension_day,
max_borrow_day, max_borrow_book) VALUES (5, 100, 120, 12, 4, 4);
INSERT INTO Membership (membership_level, credit, duration_day, max_loan_extension_day,
max_borrow_day, max_borrow_book) VALUES (6, 100, 180, 14, 5, 5);
INSERT INTO Membership (membership_level, credit, duration_day, max_loan_extension_day,
max_borrow_day, max_borrow_book) VALUES (7, 100, 250, 16, 5, 5);
INSERT INTO Membership (membership_level, credit, duration_day, max_loan_extension_day,
max_borrow_day, max_borrow_book) VALUES (8, 100, 365, 18, 6, 6);
INSERT INTO Membership (membership_level, credit, duration_day, max_loan_extension_day,
max_borrow_day, max_borrow_book) VALUES (9, 100, 550, 20, 6, 6);
INSERT INTO Membership (membership_level, credit, duration_day, max_loan_extension_day,
max_borrow_day, max_borrow_book) VALUES (10, 100, 730, 25, 6, 6);

-- Insert into Fine_Metric Table (10 records)
INSERT INTO Fine_Metric (fine_per_book, returned_hours, credit_deduct) VALUES (1.5, 24, 2);
INSERT INTO Fine_Metric (fine_per_book, returned_hours, credit_deduct) VALUES (3.0, 48, 4);
INSERT INTO Fine_Metric (fine_per_book, returned_hours, credit_deduct) VALUES (4.5, 72, 6);
INSERT INTO Fine_Metric (fine_per_book, returned_hours, credit_deduct) VALUES (6.0, 96, 8);
INSERT INTO Fine_Metric (fine_per_book, returned_hours, credit_deduct) VALUES (7.5, 120, 10);
INSERT INTO Fine_Metric (fine_per_book, returned_hours, credit_deduct) VALUES (9.0, 144, 12);
INSERT INTO Fine_Metric (fine_per_book, returned_hours, credit_deduct) VALUES (10.5, 168, 14);
INSERT INTO Fine_Metric (fine_per_book, returned_hours, credit_deduct) VALUES (12.0, 192, 16);
INSERT INTO Fine_Metric (fine_per_book, returned_hours, credit_deduct) VALUES (13.5, 216, 18);
INSERT INTO Fine_Metric (fine_per_book, returned_hours, credit_deduct) VALUES (15.0, 240, 20);


-- Insert into Member Table (100 records)
INSERT INTO Member (membership_id, name, gender, email, tel_no, credit, reg_date, mb_status) VALUES
('MS0002', 'Adam Tan', 'M', 'adam.tan@example.com', '011-1234567', 100, TO_DATE('2025-01-03',
'YYYY-MM-DD'), 1);
INSERT INTO Member (membership_id, name, gender, email, tel_no, credit, reg_date, mb_status) VALUES
('MS0002', 'Emily Wong', 'F', 'emily.wong@example.com', '011-2234568', 100, TO_DATE('2025-01-02',
```

```
'YYYY-MM-DD'), 1);
INSERT INTO Member (membership_id, name, gender, email, tel_no, credit, reg_date, mb_status) VALUES
('MS0002', 'Joshua Lim', 'M', 'joshua.lim@example.com', '011-3234569', 100, TO_DATE('2025-01-01',
'YYYY-MM-DD'), 1);
INSERT INTO Member (membership_id, name, gender, email, tel_no, credit, reg_date, mb_status) VALUES
('MS0002', 'Sophia Lee', 'F', 'sophia.lee@example.com', '011-4234570', 100, TO_DATE('2024-12-31',
'YYYY-MM-DD'), 1);
INSERT INTO Member (membership_id, name, gender, email, tel_no, credit, reg_date, mb_status) VALUES
('MS0002', 'Daniel Chan', 'M', 'daniel.chan@example.com', '011-5234571', 100, TO_DATE('2024-12-30',
'YYYY-MM-DD'), 1);
INSERT INTO Member (membership_id, name, gender, email, tel_no, credit, reg_date, mb_status) VALUES
('MS0002', 'Grace Teoh', 'F', 'grace.teoh@example.com', '011-6234572', 100, TO_DATE('2024-12-29',
'YYYY-MM-DD'), 1);
INSERT INTO Member (membership_id, name, gender, email, tel_no, credit, reg_date, mb_status) VALUES
('MS0002', 'Marcus Ng', 'M', 'marcus.ng@example.com', '011-7234573', 100, TO_DATE('2024-12-28',
'YYYY-MM-DD'), 1);
INSERT INTO Member (membership_id, name, gender, email, tel_no, credit, reg_date, mb_status) VALUES
('MS0002', 'Chloe Lau', 'F', 'chloe.lau@example.com', '011-8234574', 100, TO_DATE('2024-12-27',
'YYYY-MM-DD'), 1);
INSERT INTO Member (membership_id, name, gender, email, tel_no, credit, reg_date, mb_status) VALUES
('MS0002', 'Ethan Ong', 'M', 'ethan.ong@example.com', '011-9234575', 100, TO_DATE('2024-12-26',
'YYYY-MM-DD'), 1);
INSERT INTO Member (membership_id, name, gender, email, tel_no, credit, reg_date, mb_status) VALUES
('MS0002', 'Mia Goh', 'F', 'mia.goh@example.com', '011-0234576', 100, TO_DATE('2024-12-25',
'YYYY-MM-DD'), 1);

-- Insert into Reservation Table (120 records)
INSERT INTO Reservation (member_id, book_id, rsv_date, rsv_status) VALUES ('MB0087', 'BK0053',
TO_DATE('2025-03-15', 'YYYY-MM-DD'), 'RESERVED');
INSERT INTO Reservation (member_id, book_id, rsv_date, rsv_status) VALUES ('MB0033', 'BK0047',
TO_DATE('2025-03-22', 'YYYY-MM-DD'), 'READY');
INSERT INTO Reservation (member_id, book_id, rsv_date, rsv_status) VALUES ('MB0003', 'BK0081',
TO_DATE('2025-01-10', 'YYYY-MM-DD'), 'COMPLETED');
INSERT INTO Reservation (member_id, book_id, rsv_date, rsv_status) VALUES ('MB0072', 'BK0056',
TO_DATE('2025-02-20', 'YYYY-MM-DD'), 'CANCELLED');
INSERT INTO Reservation (member_id, book_id, rsv_date, rsv_status) VALUES ('MB0091', 'BK0079',
TO_DATE('2025-03-25', 'YYYY-MM-DD'), 'READY');
INSERT INTO Reservation (member_id, book_id, rsv_date, rsv_status) VALUES ('MB0002', 'BK0042',
TO_DATE('2025-03-19', 'YYYY-MM-DD'), 'RESERVED');
INSERT INTO Reservation (member_id, book_id, rsv_date, rsv_status) VALUES ('MB0090', 'BK0033',
TO_DATE('2025-02-12', 'YYYY-MM-DD'), 'COMPLETED');
INSERT INTO Reservation (member_id, book_id, rsv_date, rsv_status) VALUES ('MB0079', 'BK0022',
TO_DATE('2025-03-14', 'YYYY-MM-DD'), 'CANCELLED');
INSERT INTO Reservation (member_id, book_id, rsv_date, rsv_status) VALUES ('MB0057', 'BK0010',
TO_DATE('2025-03-17', 'YYYY-MM-DD'), 'READY');
INSERT INTO Reservation (member_id, book_id, rsv_date, rsv_status) VALUES ('MB0092', 'BK0076',
TO_DATE('2025-02-18', 'YYYY-MM-DD'), 'COMPLETED');

-- Insert into Borrow Table (150 records)
INSERT INTO Borrow(member_id, borrow_datetime, return_datetime) VALUES ('MB0094',
TO_DATE('7-2-2025 17:23:40', 'DD-MM-YYYY HH24:MI:SS'), TO_DATE('9-2-2025 17:23:40',
'DD-MM-YYYY HH24:MI:SS'));
INSERT INTO Borrow(member_id, borrow_datetime, return_datetime) VALUES ('MB0060',
TO_DATE('20-3-2025 18:32:49', 'DD-MM-YYYY HH24:MI:SS'), TO_DATE('22-3-2025 18:32:49',
'DD-MM-YYYY HH24:MI:SS'));
INSERT INTO Borrow(member_id, borrow_datetime, return_datetime) VALUES ('MB0063',
TO_DATE('2-1-2025 13:32:47', 'DD-MM-YYYY HH24:MI:SS'), TO_DATE('7-1-2025 13:32:47',
'DD-MM-YYYY HH24:MI:SS'));
INSERT INTO Borrow(member_id, borrow_datetime, return_datetime) VALUES ('MB0082',
TO_DATE('18-2-2025 22:22:17', 'DD-MM-YYYY HH24:MI:SS'), TO_DATE('22-2-2025 22:22:17',
'DD-MM-YYYY HH24:MI:SS'));
INSERT INTO Borrow(member_id, borrow_datetime, return_datetime) VALUES ('MB0096',
TO_DATE('11-2-2025 9:52:57', 'DD-MM-YYYY HH24:MI:SS'), TO_DATE('15-2-2025 9:52:57',
'DD-MM-YYYY HH24:MI:SS'));
INSERT INTO Borrow(member_id, borrow_datetime, return_datetime) VALUES ('MB0064',
TO_DATE('22-3-2025 3:58:23', 'DD-MM-YYYY HH24:MI:SS'), TO_DATE('26-3-2025 3:58:23',
```

```
'DD-MM-YYYY HH24:MI:SS'));
INSERT INTO Borrow(member_id, borrow_datetime, return_datetime) VALUES ('MB0074',
TO_DATE('13-1-2025 2:14:22', 'DD-MM-YYYY HH24:MI:SS'), TO_DATE('16-1-2025 2:14:22',
'DD-MM-YYYY HH24:MI:SS'));
INSERT INTO Borrow(member_id, borrow_datetime, return_datetime) VALUES ('MB0034',
TO_DATE('9-3-2025 16:53:36', 'DD-MM-YYYY HH24:MI:SS'), TO_DATE('13-3-2025 16:53:36',
'DD-MM-YYYY HH24:MI:SS'));
INSERT INTO Borrow(member_id, borrow_datetime, return_datetime) VALUES ('MB0021',
TO_DATE('14-3-2025 17:59:49', 'DD-MM-YYYY HH24:MI:SS'), TO_DATE('18-3-2025 17:59:49',
'DD-MM-YYYY HH24:MI:SS'));
INSERT INTO Borrow(member_id, borrow_datetime, return_datetime) VALUES ('MB0048',
TO_DATE('24-1-2025 0:29:51', 'DD-MM-YYYY HH24:MI:SS'), TO_DATE('30-1-2025 0:29:51',
'DD-MM-YYYY HH24:MI:SS'));

-- Insert into Loan_Extension Table (150 records)
INSERT INTO Loan_Extension(borrow_id, days_extended, le_datetime) VALUES ('BR0051', 16,
TO_DATE('2024-12-01 06:03:07', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Loan_Extension(borrow_id, days_extended, le_datetime) VALUES ('BR0128', 5,
TO_DATE('2025-03-03 07:44:35', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Loan_Extension(borrow_id, days_extended, le_datetime) VALUES ('BR0029', 14,
TO_DATE('2024-12-02 04:50:02', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Loan_Extension(borrow_id, days_extended, le_datetime) VALUES ('BR0054', 20,
TO_DATE('2025-02-27 22:15:22', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Loan_Extension(borrow_id, days_extended, le_datetime) VALUES ('BR0032', 16,
TO_DATE('2025-01-01 08:28:13', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Loan_Extension(borrow_id, days_extended, le_datetime) VALUES ('BR0057', 18,
TO_DATE('2025-02-28 09:50:40', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Loan_Extension(borrow_id, days_extended, le_datetime) VALUES ('BR0026', 25,
TO_DATE('2025-03-19 07:22:42', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Loan_Extension(borrow_id, days_extended, le_datetime) VALUES ('BR0126', 7,
TO_DATE('2025-03-15 22:44:00', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Loan_Extension(borrow_id, days_extended, le_datetime) VALUES ('BR0068', 20,
TO_DATE('2025-01-12 05:42:58', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Loan_Extension(borrow_id, days_extended, le_datetime) VALUES ('BR0127', 14,
TO_DATE('2024-12-29 13:44:39', 'YYYY-MM-DD HH24:MI:SS'));

-- Insert into Borrow_Detail Table (150 records)
INSERT INTO Borrow_Detail(borrow_id, book_id, returned_datetime, borrow_status) VALUES ('BR0056',
'BK0056', TO_DATE('2025-01-03 11:59:51', 'YYYY-MM-DD HH24:MI:SS'), 'Returned');
INSERT INTO Borrow_Detail(borrow_id, book_id, returned_datetime, borrow_status) VALUES ('BR0032',
'BK0050', TO_DATE('2025-01-22 10:48:17', 'YYYY-MM-DD HH24:MI:SS'), 'Returned');
INSERT INTO Borrow_Detail(borrow_id, book_id, returned_datetime, borrow_status) VALUES ('BR0099',
'BK0022', TO_DATE('2025-01-14 17:58:08', 'YYYY-MM-DD HH24:MI:SS'), 'Damaged');
INSERT INTO Borrow_Detail(borrow_id, book_id, returned_datetime, borrow_status) VALUES ('BR0062',
'BK0006', TO_DATE('2025-02-28 01:30:43', 'YYYY-MM-DD HH24:MI:SS'), 'Damaged');
INSERT INTO Borrow_Detail(borrow_id, book_id, returned_datetime, borrow_status) VALUES ('BR0080',
'BK0056', TO_DATE('2025-01-20 23:04:36', 'YYYY-MM-DD HH24:MI:SS'), 'Missing');
INSERT INTO Borrow_Detail(borrow_id, book_id, returned_datetime, borrow_status) VALUES ('BR0143',
'BK0036', TO_DATE('2025-03-23 02:49:50', 'YYYY-MM-DD HH24:MI:SS'), 'Missing');
INSERT INTO Borrow_Detail(borrow_id, book_id, returned_datetime, borrow_status) VALUES ('BR0050',
'BK0025', TO_DATE('2025-02-02 02:27:00', 'YYYY-MM-DD HH24:MI:SS'), 'Damaged');
INSERT INTO Borrow_Detail(borrow_id, book_id, returned_datetime, borrow_status) VALUES ('BR0147',
'BK0029', TO_DATE('2025-02-24 11:57:48', 'YYYY-MM-DD HH24:MI:SS'), 'Borrowed');
INSERT INTO Borrow_Detail(borrow_id, book_id, returned_datetime, borrow_status) VALUES ('BR0117',
'BK0061', TO_DATE('2025-03-09 17:42:43', 'YYYY-MM-DD HH24:MI:SS'), 'Missing');
INSERT INTO Borrow_Detail(borrow_id, book_id, returned_datetime, borrow_status) VALUES ('BR0006',
'BK0023', TO_DATE('2025-01-01 20:50:34', 'YYYY-MM-DD HH24:MI:SS'), 'Borrowed');

-- Insert into Invoice Table (120 records)
INSERT INTO Invoice(borrow_detail_id, total_amt, inv_datetime) VALUES ('BD0032', 13.5,
TO_DATE('2024-12-14 09:14:00', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Invoice(borrow_detail_id, total_amt, inv_datetime) VALUES ('BD0007', 6,
TO_DATE('2025-04-03 14:02:39', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Invoice(borrow_detail_id, total_amt, inv_datetime) VALUES ('BD0029', 10.5,
TO_DATE('2025-01-21 01:43:11', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Invoice(borrow_detail_id, total_amt, inv_datetime) VALUES ('BD0046', 4.5,
```

```
TO_DATE('2025-01-31 10:46:58', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Invoice(borrow_detail_id, total_amt, inv_datetime) VALUES ('BD0149', 10.5,
TO_DATE('2025-01-16 22:40:26', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Invoice(borrow_detail_id, total_amt, inv_datetime) VALUES ('BD0053', 1.5,
TO_DATE('2025-02-17 17:29:22', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Invoice(borrow_detail_id, total_amt, inv_datetime) VALUES ('BD0116', 13.5,
TO_DATE('2025-01-24 02:17:26', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Invoice(borrow_detail_id, total_amt, inv_datetime) VALUES ('BD0094', 13.5,
TO_DATE('2025-03-07 19:32:47', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Invoice(borrow_detail_id, total_amt, inv_datetime) VALUES ('BD0101', 4.5,
TO_DATE('2024-12-04 15:27:03', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Invoice(borrow_detail_id, total_amt, inv_datetime) VALUES ('BD0112', 1.5,
TO_DATE('2024-12-19 06:11:05', 'YYYY-MM-DD HH24:MI:SS'));

-- Insert into Fine_Record Table (150 records)
INSERT INTO Fine_Record(borrow_detail_id,credit_deduct,fine_amt) VALUES ('BD0084',16,12);
INSERT INTO Fine_Record(borrow_detail_id,credit_deduct,fine_amt) VALUES ('BD0016',16,12);
INSERT INTO Fine_Record(borrow_detail_id,credit_deduct,fine_amt) VALUES ('BD0077',14,10.5);
INSERT INTO Fine_Record(borrow_detail_id,credit_deduct,fine_amt) VALUES ('BD0133',18,13.5);
INSERT INTO Fine_Record(borrow_detail_id,credit_deduct,fine_amt) VALUES ('BD0127',4,3);
INSERT INTO Fine_Record(borrow_detail_id,credit_deduct,fine_amt) VALUES ('BD0142',4,3);
INSERT INTO Fine_Record(borrow_detail_id,credit_deduct,fine_amt) VALUES ('BD0081',12,9);
INSERT INTO Fine_Record(borrow_detail_id,credit_deduct,fine_amt) VALUES ('BD0064',10,7.5);
INSERT INTO Fine_Record(borrow_detail_id,credit_deduct,fine_amt) VALUES ('BD0098',8,6);
INSERT INTO Fine_Record(borrow_detail_id,credit_deduct,fine_amt) VALUES ('BD0124',16,12);

-- Data for Staff (15 records)
INSERT INTO Staff (name, gender, ic_no, email, tel_no, address, position, reg_date, stf_status) VALUES
('John Doe', 'M', '900101-14-1234', 'johndoe@library.com', '012-3456789', '123 Library Lane, KL', 'Manager',
TO_DATE('2023-01-10', 'YYYY-MM-DD'), 'Active');
INSERT INTO Staff (name, gender, ic_no, email, tel_no, address, position, reg_date, stf_status) VALUES
('Jane Smith', 'F', '910202-10-5678', 'janesmith@library.com', '012-9876543', '456 Bookworm Blvd, PJ',
'Librarian', TO_DATE('2022-12-15', 'YYYY-MM-DD'), 'Active');
INSERT INTO Staff (name, gender, ic_no, email, tel_no, address, position, reg_date, stf_status) VALUES
('Ahmad Bakar', 'M', '880515-08-3321', 'ahmad.b@library.com', '019-1122334', '789 Reading Road, Shah
Alam', 'IT Support', TO_DATE('2023-03-20', 'YYYY-MM-DD'), 'Active');
INSERT INTO Staff (name, gender, ic_no, email, tel_no, address, position, reg_date, stf_status) VALUES ('Siti
Aminah', 'F', '951120-01-4455', 'siti.a@library.com', '011-5566778', '101 Chapter Close, Subang Jaya',
'Librarian', TO_DATE('2023-04-01', 'YYYY-MM-DD'), 'Active');
INSERT INTO Staff (name, gender, ic_no, email, tel_no, address, position, reg_date, stf_status) VALUES
('Alice Brown', 'F', '920303-08-9101', 'alice.b@library.com', '011-2233445', '789 Oak St, Village', 'Assistant
Librarian', TO_DATE('2022-11-20', 'YYYY-MM-DD'), 'Inactive');
INSERT INTO Staff (name, gender, ic_no, email, tel_no, address, position, reg_date, stf_status) VALUES ('Bob
White', 'M', '930404-07-1123', 'bobwhite@library.com', '010-5566778', '101 Maple St, City', 'Technician',
TO_DATE('2023-02-05', 'YYYY-MM-DD'), 'Active');
INSERT INTO Staff (name, gender, ic_no, email, tel_no, address, position, reg_date, stf_status) VALUES
('Charlie Green', 'M', '940505-05-3344', 'charlie.g@library.com', '019-6677889', '202 Birch St, Town', 'Security',
TO_DATE('2023-03-12', 'YYYY-MM-DD'), 'Inactive');
INSERT INTO Staff (name, gender, ic_no, email, tel_no, address, position, reg_date, stf_status) VALUES
('Daisy Black', 'F', '950606-04-5566', 'daisy.b@library.com', '012-7788990', '303 Cedar St, Village', 'Librarian',
TO_DATE('2023-04-01', 'YYYY-MM-DD'), 'Active');
INSERT INTO Staff (name, gender, ic_no, email, tel_no, address, position, reg_date, stf_status) VALUES
('Edward Blue', 'M', '960707-03-7788', 'edward.b@library.com', '013-8899001', '404 Pine St, City', 'Assistant
Librarian', TO_DATE('2023-05-18', 'YYYY-MM-DD'), 'Active');
INSERT INTO Staff (name, gender, ic_no, email, tel_no, address, position, reg_date, stf_status) VALUES
('Fiona Red', 'F', '970808-02-9900', 'fiona.r@library.com', '014-9900112', '505 Willow St, Town', 'Technician',
TO_DATE('2023-06-25', 'YYYY-MM-DD'), 'Inactive');

-- Data for Task (12 records)
INSERT INTO Task (description, max_capacity, estimated_hour, tk_status) VALUES ('Organize Fiction
Bookshelves A-G', 2, 3, 'Active');
INSERT INTO Task (description, max_capacity, estimated_hour, tk_status) VALUES ('Catalog New Arrivals -
Batch 1', 1, 4, 'Active');
INSERT INTO Task (description, max_capacity, estimated_hour, tk_status) VALUES ('Front Desk Assistance
(Morning Shift)', 1, 4, 'Active'),;
INSERT INTO Task (description, max_capacity, estimated_hour, tk_status) VALUES ('Database Backup and
```

```
Maintenance', 1, 2, 'Active');
INSERT INTO Task (description, max_capacity, estimated_hour, tk_status) VALUES ('Security Patrol - Level 1 and 2', 1, 2, 'Active');
INSERT INTO Task (description, max_capacity, estimated_hour, tk_status) VALUES ('Check Overdue List And Send Reminders', 2, 3, 'Active');
INSERT INTO Task (description, max_capacity, estimated_hour, tk_status) VALUES ('Prepare for Children Story Time Event', 2, 4, 'Active');
INSERT INTO Task (description, max_capacity, estimated_hour, tk_status) VALUES ('Minor Book Repair (Taping, Cleaning)', 1, 3, 'Inactive');
INSERT INTO Task (description, max_capacity, estimated_hour, tk_status) VALUES ('Clean And Sanitize Study Area Zone B', 2, 2, 'Active');
INSERT INTO Task (description, max_capacity, estimated_hour, tk_status) VALUES ('Assist Users at Digital Kiosks', 1, 4, 'Active');

-- Insert into Attendance (100 records)
INSERT INTO Attendance (staff_id, checkin_datetime, checkout_datetime, att_date, att_status) VALUES ('SF0006', TO_DATE('2025-02-06 08:45:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-02-06 14:06:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-02-06 08:33:08', 'YYYY-MM-DD HH24:MI:SS'), 'Leave');
INSERT INTO Attendance (staff_id, checkin_datetime, checkout_datetime, att_date, att_status) VALUES ('SF0014', TO_DATE('2025-02-08 10:28:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-02-08 19:06:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-02-08 04:38:54', 'YYYY-MM-DD HH24:MI:SS'), 'Present');
INSERT INTO Attendance (staff_id, checkin_datetime, checkout_datetime, att_date, att_status) VALUES ('SF0008', TO_DATE('2025-03-20 10:42:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-20 18:31:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-20 16:54:11', 'YYYY-MM-DD HH24:MI:SS'), 'Leave');
INSERT INTO Attendance (staff_id, checkin_datetime, checkout_datetime, att_date, att_status) VALUES ('SF0008', TO_DATE('2025-02-19 09:29:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-02-19 15:38:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-02-19 21:45:13', 'YYYY-MM-DD HH24:MI:SS'), 'Leave');
INSERT INTO Attendance (staff_id, checkin_datetime, checkout_datetime, att_date, att_status) VALUES ('SF0015', TO_DATE('2025-02-26 10:13:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-02-26 18:53:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-02-26 09:40:15', 'YYYY-MM-DD HH24:MI:SS'), 'Absent');
INSERT INTO Attendance (staff_id, checkin_datetime, checkout_datetime, att_date, att_status) VALUES ('SF0012', TO_DATE('2025-03-29 09:41:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-29 16:05:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-29 16:29:10', 'YYYY-MM-DD HH24:MI:SS'), 'Leave');
INSERT INTO Attendance (staff_id, checkin_datetime, checkout_datetime, att_date, att_status) VALUES ('SF0010', TO_DATE('2025-02-18 08:41:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-02-18 15:01:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-02-18 12:37:15', 'YYYY-MM-DD HH24:MI:SS'), 'Present');
INSERT INTO Attendance (staff_id, checkin_datetime, checkout_datetime, att_date, att_status) VALUES ('SF0013', TO_DATE('2025-04-06 08:51:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-04-06 14:50:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-04-06 01:04:45', 'YYYY-MM-DD HH24:MI:SS'), 'Leave');
INSERT INTO Attendance (staff_id, checkin_datetime, checkout_datetime, att_date, att_status) VALUES ('SF0013', TO_DATE('2025-02-01 10:43:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-02-01 17:44:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-02-01 01:36:36', 'YYYY-MM-DD HH24:MI:SS'), 'Leave');
INSERT INTO Attendance (staff_id, checkin_datetime, checkout_datetime, att_date, att_status) VALUES ('SF0002', TO_DATE('2025-02-14 10:12:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-02-14 17:26:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-02-14 16:15:50', 'YYYY-MM-DD HH24:MI:SS'), 'Present');

-- Data for AUTO_ASSIGN (110 records)
INSERT INTO Auto_Assign (staff_id, task_id, frequency, set_date, start_time, end_time, aa_status) VALUES ('SF0002', 'TK0009', 'Weekly', TO_DATE('2025-04-20 06:47:14', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-04-20 09:10:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-04-20 10:27:00', 'YYYY-MM-DD HH24:MI:SS'), 'Inactive');
INSERT INTO Auto_Assign (staff_id, task_id, frequency, set_date, start_time, end_time, aa_status) VALUES ('SF0006', 'TK0005', 'Weekly', TO_DATE('2025-03-11 19:06:22', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-11 09:07:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-11 12:12:00', 'YYYY-MM-DD HH24:MI:SS'), 'Inactive');
INSERT INTO Auto_Assign (staff_id, task_id, frequency, set_date, start_time, end_time, aa_status) VALUES
```

```
('SF0004', 'TK0004', 'Monthly', TO_DATE('2025-03-13 04:40:32', 'YYYY-MM-DD HH24:MI:SS'),
TO_DATE('2025-03-13 08:10:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-13 09:27:00',
'YYYY-MM-DD HH24:MI:SS'), 'Inactive');
INSERT INTO Auto_Assign (staff_id, task_id, frequency, set_date, start_time, end_time, aa_status) VALUES
('SF0011', 'TK0011', 'Daily', TO_DATE('2025-02-08 08:56:03', 'YYYY-MM-DD HH24:MI:SS'),
TO_DATE('2025-02-08 08:48:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-02-08 10:50:00',
'YYYY-MM-DD HH24:MI:SS'), 'Active');
INSERT INTO Auto_Assign (staff_id, task_id, frequency, set_date, start_time, end_time, aa_status) VALUES
('SF0014', 'TK0010', 'Daily', TO_DATE('2025-02-05 21:46:09', 'YYYY-MM-DD HH24:MI:SS'),
TO_DATE('2025-02-05 08:11:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-02-05 09:19:00',
'YYYY-MM-DD HH24:MI:SS'), 'Inactive');
INSERT INTO Auto_Assign (staff_id, task_id, frequency, set_date, start_time, end_time, aa_status) VALUES
('SF0015', 'TK0011', 'Daily', TO_DATE('2025-01-19 13:56:50', 'YYYY-MM-DD HH24:MI:SS'),
TO_DATE('2025-01-19 09:59:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-01-19 13:41:00',
'YYYY-MM-DD HH24:MI:SS'), 'Inactive');
INSERT INTO Auto_Assign (staff_id, task_id, frequency, set_date, start_time, end_time, aa_status) VALUES
('SF0011', 'TK0001', 'Daily', TO_DATE('2025-04-09 17:58:44', 'YYYY-MM-DD HH24:MI:SS'),
TO_DATE('2025-04-09 10:53:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-04-09 15:38:00',
'YYYY-MM-DD HH24:MI:SS'), 'Inactive');
INSERT INTO Auto_Assign (staff_id, task_id, frequency, set_date, start_time, end_time, aa_status) VALUES
('SF0010', 'TK0006', 'Monthly', TO_DATE('2025-03-09 05:10:11', 'YYYY-MM-DD HH24:MI:SS'),
TO_DATE('2025-03-09 08:35:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-09 12:31:00',
'YYYY-MM-DD HH24:MI:SS'), 'Inactive');
INSERT INTO Auto_Assign (staff_id, task_id, frequency, set_date, start_time, end_time, aa_status) VALUES
('SF0007', 'TK0003', 'Weekly', TO_DATE('2025-04-01 17:07:05', 'YYYY-MM-DD HH24:MI:SS'),
TO_DATE('2025-04-01 09:24:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-04-01 11:51:00',
'YYYY-MM-DD HH24:MI:SS'), 'Active');
INSERT INTO Auto_Assign (staff_id, task_id, frequency, set_date, start_time, end_time, aa_status) VALUES
('SF0011', 'TK0004', 'Monthly', TO_DATE('2025-01-27 14:43:34', 'YYYY-MM-DD HH24:MI:SS'),
TO_DATE('2025-01-27 10:33:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-01-27 13:14:00',
'YYYY-MM-DD HH24:MI:SS'), 'Active');

-- Insert data into Schedule (110 records)
INSERT INTO Schedule (task_id, staff_id, start_datetime, end_datetime, attend_datetime) VALUES ('TK0006',
'SF0015', TO_DATE('2025-03-24 10:57:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-24
16:42:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-24 11:08:00', 'YYYY-MM-DD
HH24:MI:SS'));
INSERT INTO Schedule (task_id, staff_id, start_datetime, end_datetime, attend_datetime) VALUES ('TK0003',
'SF0013', TO_DATE('2025-02-25 09:46:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-02-25
10:58:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-02-25 09:49:00', 'YYYY-MM-DD
HH24:MI:SS'));
INSERT INTO Schedule (task_id, staff_id, start_datetime, end_datetime, attend_datetime) VALUES ('TK0011',
'SF0010', TO_DATE('2025-04-13 08:21:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-04-13
10:11:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-04-13 08:37:00', 'YYYY-MM-DD
HH24:MI:SS'));
INSERT INTO Schedule (task_id, staff_id, start_datetime, end_datetime, attend_datetime) VALUES ('TK0009',
'SF0006', TO_DATE('2025-04-08 11:48:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-04-08
14:46:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-04-08 12:02:00', 'YYYY-MM-DD
HH24:MI:SS'));
INSERT INTO Schedule (task_id, staff_id, start_datetime, end_datetime, attend_datetime) VALUES ('TK0010',
'SF0010', TO_DATE('2025-04-06 09:26:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-04-06
14:02:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-04-06 09:46:00', 'YYYY-MM-DD
HH24:MI:SS'));
INSERT INTO Schedule (task_id, staff_id, start_datetime, end_datetime, attend_datetime) VALUES ('TK0003',
'SF0010', TO_DATE('2025-03-15 09:49:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-15
11:39:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-15 09:52:00', 'YYYY-MM-DD
HH24:MI:SS'));
INSERT INTO Schedule (task_id, staff_id, start_datetime, end_datetime, attend_datetime) VALUES ('TK0001',
'SF0005', TO_DATE('2025-03-18 09:32:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-18
13:23:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-18 09:48:00', 'YYYY-MM-DD
HH24:MI:SS'));
INSERT INTO Schedule (task_id, staff_id, start_datetime, end_datetime, attend_datetime) VALUES ('TK0001',
'SF0010', TO_DATE('2025-02-28 11:56:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-02-28
13:37:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-02-28 12:01:00', 'YYYY-MM-DD
HH24:MI:SS'));
INSERT INTO Schedule (task_id, staff_id, start_datetime, end_datetime, attend_datetime) VALUES ('TK0006',
```

```
'SF0011', TO_DATE('2025-03-28 09:36:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-28
15:17:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-28 09:55:00', 'YYYY-MM-DD
HH24:MI:SS'));
INSERT INTO Schedule (task_id, staff_id, start_datetime, end_datetime, attend_datetime) VALUES ('TK0008',
'SF0020', TO_DATE('2025-03-24 10:20:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-24
15:32:00', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-24 10:48:00', 'YYYY-MM-DD
HH24:MI:SS'));

-- Insert data into Shifting (120 records)
INSERT INTO Shifting (schedule_id, old_from_datetime, old_to_datetime, new_from_datetime,
new_to_datetime) VALUES ('SC0089', TO_DATE('2025-03-26 00:30:27', 'YYYY-MM-DD HH24:MI:SS'),
TO_DATE('2025-03-26 02:30:27', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-28 00:30:27',
'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-28 05:30:27', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Shifting (schedule_id, old_from_datetime, old_to_datetime, new_from_datetime,
new_to_datetime) VALUES ('SC0023', TO_DATE('2025-03-25 02:33:31', 'YYYY-MM-DD HH24:MI:SS'),
TO_DATE('2025-03-25 03:33:31', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-26 02:33:31',
'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-26 04:33:31', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Shifting (schedule_id, old_from_datetime, old_to_datetime, new_from_datetime,
new_to_datetime) VALUES ('SC0075', TO_DATE('2025-03-14 01:24:53', 'YYYY-MM-DD HH24:MI:SS'),
TO_DATE('2025-03-14 05:24:53', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-17 01:24:53',
'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-17 06:24:53', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Shifting (schedule_id, old_from_datetime, old_to_datetime, new_from_datetime,
new_to_datetime) VALUES ('SC0049', TO_DATE('2025-04-20 11:55:43', 'YYYY-MM-DD HH24:MI:SS'),
TO_DATE('2025-04-20 14:55:43', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-04-22 11:55:43',
'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-04-22 15:55:43', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Shifting (schedule_id, old_from_datetime, old_to_datetime, new_from_datetime,
new_to_datetime) VALUES ('SC0088', TO_DATE('2025-03-04 00:10:12', 'YYYY-MM-DD HH24:MI:SS'),
TO_DATE('2025-03-04 02:10:12', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-06 00:10:12',
'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-06 03:10:12', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Shifting (schedule_id, old_from_datetime, old_to_datetime, new_from_datetime,
new_to_datetime) VALUES ('SC0065', TO_DATE('2025-02-28 19:28:43', 'YYYY-MM-DD HH24:MI:SS'),
TO_DATE('2025-02-28 21:28:43', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-03 19:28:43',
'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-03 20:28:43', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Shifting (schedule_id, old_from_datetime, old_to_datetime, new_from_datetime,
new_to_datetime) VALUES ('SC0068', TO_DATE('2025-03-23 02:34:11', 'YYYY-MM-DD HH24:MI:SS'),
TO_DATE('2025-03-23 07:34:11', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-25 02:34:11',
'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-25 06:34:11', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Shifting (schedule_id, old_from_datetime, old_to_datetime, new_from_datetime,
new_to_datetime) VALUES ('SC0104', TO_DATE('2025-03-16 20:18:05', 'YYYY-MM-DD HH24:MI:SS'),
TO_DATE('2025-03-17 00:18:05', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-19 20:18:05',
'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-20 01:18:05', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Shifting (schedule_id, old_from_datetime, old_to_datetime, new_from_datetime,
new_to_datetime) VALUES ('SC0046', TO_DATE('2025-03-24 20:37:21', 'YYYY-MM-DD HH24:MI:SS'),
TO_DATE('2025-03-24 22:37:21', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-27 20:37:21',
'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-03-28 01:37:21', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Shifting (schedule_id, old_from_datetime, old_to_datetime, new_from_datetime,
new_to_datetime) VALUES ('SC0025', TO_DATE('2025-03-30 21:23:47', 'YYYY-MM-DD HH24:MI:SS'),
TO_DATE('2025-03-31 01:23:47', 'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-04-02 21:23:47',
'YYYY-MM-DD HH24:MI:SS'), TO_DATE('2025-04-03 01:23:47', 'YYYY-MM-DD HH24:MI:SS'));

-- Insert data into Task_Progress (105 records)
INSERT INTO Task_Progress (schedule_id, reason, tp_status, tp_datetime) VALUES ('SC0037', 'Between hope
view put cost.', 'Scheduled', TO_DATE('2025-03-31 07:15:55', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Task_Progress (schedule_id, reason, tp_status, tp_datetime) VALUES ('SC0065', '', 'Completed',
TO_DATE('2025-04-03 19:12:58', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Task_Progress (schedule_id, reason, tp_status, tp_datetime) VALUES ('SC0004', '', 'Completed',
TO_DATE('2025-04-16 14:24:32', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Task_Progress (schedule_id, reason, tp_status, tp_datetime) VALUES ('SC0107', '', 'Completed',
TO_DATE('2025-04-15 19:58:32', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Task_Progress (schedule_id, reason, tp_status, tp_datetime) VALUES ('SC0011', '', 'Completed',
TO_DATE('2025-04-09 09:42:28', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Task_Progress (schedule_id, reason, tp_status, tp_datetime) VALUES ('SC0075', '', 'Scheduled',
TO_DATE('2025-03-27 04:37:29', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Task_Progress (schedule_id, reason, tp_status, tp_datetime) VALUES ('SC0003', 'Local
structure what page current.', 'Ongoing', TO_DATE('2025-04-19 10:28:09', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Task_Progress (schedule_id, reason, tp_status, tp_datetime) VALUES ('SC0022', 'Game
```

```
difficult enjoy task.', 'Ongoing', TO_DATE('2025-03-25 02:59:42', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Task_Progress (schedule_id, reason, tp_status, tp_datetime) VALUES ('SC0047', '', 'Ongoing',
TO_DATE('2025-04-14 10:00:48', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Task_Progress (schedule_id, reason, tp_status, tp_datetime) VALUES ('SC0046', '', 'Ongoing',
TO_DATE('2025-04-15 01:31:07', 'YYYY-MM-DD HH24:MI:SS'));
```

# Chapter 4 Queries, Procedures, Triggers and Reports

## 4.1 Chia Ming Yi

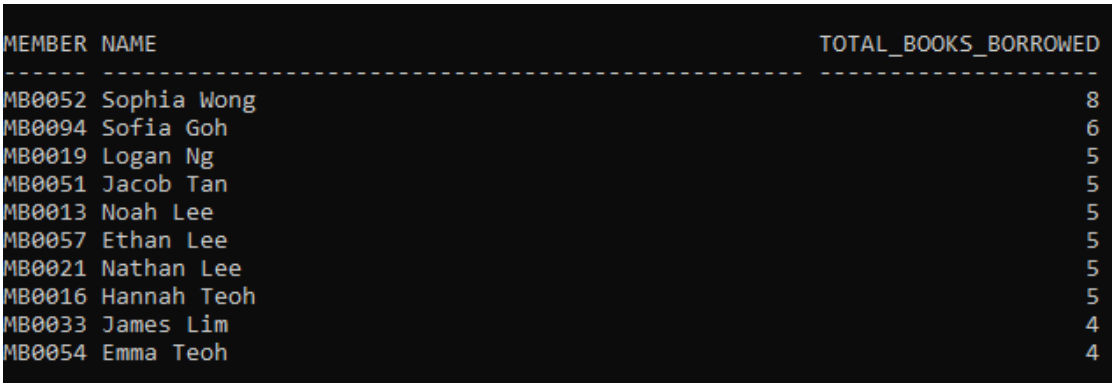
### 4.1.1 Query 1: Most Active Members by Borrow Count

Purpose: The purpose of this query is to help management understand user engagement and identify top readers or potential candidates for rewards/loyalty programs.

**SQL statement:**

```
SELECT
    m.member_id,
    m.name,
    COUNT(bd.borrow_detail_id) AS total_books_borrowed
FROM
    member m
JOIN
    borrow b ON m.member_id = b.member_id
JOIN
    borrow_detail bd ON b.borrow_id = bd.borrow_id
GROUP BY
    m.member_id, m.name
ORDER BY
    total_books_borrowed DESC
FETCH FIRST 10 ROWS ONLY;
```

**Sample Output:**



| MEMBER | NAME | TOTAL_BOOKS_BORROWED |
|---|---|---|
| MB0052 | Sophia Wong | 8 |
| MB0094 | Sofia Goh | 6 |
| MB0019 | Logan Ng | 5 |
| MB0051 | Jacob Tan | 5 |
| MB0013 | Noah Lee | 5 |
| MB0057 | Ethan Lee | 5 |
| MB0021 | Nathan Lee | 5 |
| MB0016 | Hannah Teoh | 5 |
| MB0033 | James Lim | 4 |
| MB0054 | Emma Teoh | 4 |

### 4.1.2 Query 2: Books with Most Overdue Returns

Purpose: The purpose of this query is to identify books that are often returned late so management can decide whether to increase copies, shorten loan duration, or increase penalties.

**SQL statement:**

```
SELECT
    b.name AS book_title,
    COUNT(fr.fine_id) AS overdue_count
FROM
    Fine_Record fr
JOIN
    Borrow_Detail bd ON fr.borrow_detail_id = bd.borrow_detail_id
JOIN
    Book b ON bd.book_id = b.book_id
GROUP BY
    b.name
ORDER BY
    overdue_count DESC;
```

Sample Output:

```
BOOK_TITLE                                          OVERDUE_COUNT
--------------------------------------------------- -------------
Heavy size.                                                     1
Quickly yard.                                                   1
Manage sense if.                                                1
Though anyone.                                                  1
Threat.                                                         1
Rate safe account.                                              1
Large enter push.                                               1
Enough.                                                         1
Pattern central.                                                1
Left fight must safe.                                           1
Cut toward through itself.                                      1

BOOK_TITLE                                          OVERDUE_COUNT
--------------------------------------------------- -------------
Door.                                                           1
Much.                                                           1
Into help.                                                      1

58 rows selected.
```

### 4.1.3 Procedure 1: Book Reservation Procedure for Members

**Purpose: The purpose of this procedure is to allows members to reserve a book, ensuring that the book is available and not already borrowed or reserved. It calculates the waiting time based on existing reservations and borrowing status, providing an estimated borrow date and reservation details to the member.**

**Procedure code:**

```
CREATE OR REPLACE PROCEDURE prc_reserve_book (
    p_member_id IN  Member.member_id%TYPE,
    p_book_id   IN  Book.book_id%TYPE
) AS
    v_formatted_member_id           Member.member_id%TYPE;
    v_formatted_book_id             Book.book_id%TYPE;
    v_formatted_reservation_id      Reservation.reservation_id%TYPE;
    v_book_exist                    NUMBER;
    v_reservation_count             NUMBER;
    v_borrow_count                  NUMBER;
    v_book_status                   Book.bk_status%TYPE;
    v_book_availability             Book.availability%TYPE;
    v_current_borrower_return_date  DATE;
    v_waiting_days                  NUMBER := 0;
    v_estimated_borrow_date         DATE;
    v_existing_reservation_days     NUMBER := 0;
BEGIN
    -- format member id
    v_formatted_member_id := 'MB' || LPAD(p_member_id, 4, '0');

    -- format book id
    v_formatted_book_id := 'BK' || LPAD(p_book_id, 4, '0');

    SAVEPOINT before_insert;

    -- check whether the book is exist
    SELECT COUNT(*)
    INTO v_book_exist
    FROM Book
```

```
    WHERE book_id = v_formatted_book_id;

    IF v_book_exist = 0 THEN
        DBMS_OUTPUT.PUT_LINE('Book ' || v_formatted_book_id || ' does
not exist.');
        RETURN;
    END IF;

    -- check whether the member has reserved the book before
    SELECT COUNT(*)
    INTO v_reservation_count
    FROM Reservation
    WHERE member_id = v_formatted_member_id
        AND book_id = v_formatted_book_id
        AND rsv_status IN ('Reserved', 'Ready');

    IF v_reservation_count > 0 THEN
        DBMS_OUTPUT.PUT_LINE('You have already reserved this book.');
        RETURN;
    END IF;

    -- check whether the member is borrowing the book currently
    SELECT COUNT(br.borrow_id)
    INTO v_borrow_count
    FROM Borrow br
    JOIN Borrow_Detail bd
        ON br.borrow_id = bd.borrow_id
    WHERE br.member_id = v_formatted_member_id
        AND bd.borrow_status = 'Borrowed'
        AND bd.book_id = v_formatted_book_id;

    IF v_borrow_count > 0 THEN
        DBMS_OUTPUT.PUT_LINE('You are currently borrowing this
book.');
        RETURN;
    END IF;

    -- check book status and availability
    SELECT bk_status, availability
    INTO v_book_status, v_book_availability
    FROM Book
    WHERE book_id = v_formatted_book_id;

    -- check whether book status is active
    IF v_book_status = 'Inactive' THEN
        DBMS_OUTPUT.PUT_LINE('Book ' || v_formatted_book_id || ' does
not exist.');
        RETURN;
    END IF;

    -- check whether book availability
    IF v_book_availability = 'Y' THEN
        DBMS_OUTPUT.PUT_LINE('Book ' || v_formatted_book_id || ' is
available right now. You may directly borrow the book now!');
        RETURN;
    END IF;

    -- obtain current borrower's return date if the book is borrowed
    BEGIN
        SELECT br.return_datetime
        INTO v_current_borrower_return_date
        FROM Borrow br
        JOIN Borrow_Detail bd
            ON br.borrow_id = bd.borrow_id
        WHERE bd.book_id = v_formatted_book_id
            AND bd.borrow_status = 'Borrowed'
        ORDER BY br.borrow_datetime DESC
        FETCH FIRST 1 ROWS ONLY;
```

```
        v_waiting_days := GREATEST(v_current_borrower_return_date -
SYSDATE, 0);
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            v_waiting_days := 0;
    END;

    -- sum the max_borrow_day for all existing reservation
    BEGIN
        SELECT NVL(SUM(ms.max_borrow_day), 0)
        INTO v_existing_reservation_days
        FROM Reservation rsv
        JOIN Member m
            ON rsv.member_id = m.member_id
        JOIN Membership ms
            ON m.membership_id = ms.membership_id
        WHERE rsv.book_id = v_formatted_book_id
            AND rsv.rsv_status IN ('Reserved', 'Ready');
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('No existing reservation');
    END;

    -- calculate estimated borrow date
    v_waiting_days := v_waiting_days + v_existing_reservation_days;
    v_estimated_borrow_date := SYSDATE + v_waiting_days;

    -- insert reservation record
    INSERT INTO Reservation (member_id, book_id, rsv_datetime,
rsv_status)
    VALUES (v_formatted_member_id, v_formatted_book_id, SYSDATE,
'Reserved')
    RETURNING reservation_id INTO v_formatted_reservation_id;
    COMMIT;

    -- successful message
    DECLARE
        msg_member_name                 Member.name%TYPE;
        msg_current_borrowing_person    NUMBER;
        msg_waiting_person_count        NUMBER;
        msg_book_name                   Book.name%TYPE;
        msg_book_isbn                   Book.isbn%TYPE;
    BEGIN
        -- obtain member name and book name
        SELECT m.name, b.name, b.isbn
        INTO msg_member_name, msg_book_name, msg_book_isbn
        FROM Reservation rsv
        JOIN Member m
            ON rsv.member_id = m.member_id
        JOIN Book b
            ON rsv.book_id = b.book_id
        WHERE rsv.reservation_id = v_formatted_reservation_id;

        -- obtain current borrowing person count
        SELECT COUNT(br.borrow_id)
        INTO msg_current_borrowing_person
        FROM Borrow br
        JOIN Borrow_Detail bd
            ON br.borrow_id = bd.borrow_id
        WHERE bd.borrow_status = 'Borrowed'
            AND bd.book_id = v_formatted_book_id;

        -- obtain waiting person count
        SELECT COUNT(*)
        INTO msg_waiting_person_count
        FROM Reservation rsv
        WHERE rsv.book_id = v_formatted_book_id
```

```
            AND rsv.rsv_status IN ('Reserved', 'Ready')
            AND rsv.reservation_id != v_formatted_reservation_id;

        msg_waiting_person_count := msg_waiting_person_count +
msg_current_borrowing_person;

        DBMS_OUTPUT.PUT_LINE(CHR(13));
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 15, '-') || RPAD('-', 40, '-')
|| RPAD('-', 12, '-'));
        DBMS_OUTPUT.PUT_LINE('BOOK RESERVATION');
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 15, '-') || RPAD('-', 40, '-')
|| RPAD('-', 12, '-'));
        DBMS_OUTPUT.PUT_LINE('Reservation ID: ' ||
v_formatted_reservation_id);
        DBMS_OUTPUT.PUT_LINE('Status        : ' || 'Reserved');
        DBMS_OUTPUT.PUT_LINE('Member ID     : ' ||
v_formatted_member_id);
        DBMS_OUTPUT.PUT_LINE('Member Name   : ' || msg_member_name);
        DBMS_OUTPUT.PUT_LINE(CHR(13));

        DBMS_OUTPUT.PUT_LINE(RPAD('Book ID', 15) || RPAD('Book Name',
40) || 'ISBN');
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 15, '-') || RPAD('-', 40, '-')
|| RPAD('-', 12, '-'));

        DBMS_OUTPUT.PUT_LINE(RPAD(v_formatted_book_id, 15) ||
RPAD(msg_book_name, 40) || msg_book_isbn);
        DBMS_OUTPUT.PUT_LINE(CHR(13));

        DBMS_OUTPUT.PUT_LINE('Waiting Person Count: ' ||
msg_waiting_person_count);
        DBMS_OUTPUT.PUT_LINE('Estimated Borrow Date: ' ||
TO_CHAR(v_estimated_borrow_date, 'DD-MON-YYYY'));

        DBMS_OUTPUT.PUT_LINE(CHR(13));
        DBMS_OUTPUT.PUT_LINE('You have reserved the book ' ||
v_formatted_book_id || ' successfully!');
    END;
END;
/
```

**Sample Output:**

The reservation of Member “MB0003” before reserve the book “BK0001” and the run the procedure for reserved for book “BK0001”:

```
SQL> select * from reservation where member_id = 'MB0003';

RESERV MEMBER BOOK_I RSV_DATET RSV_STATUS           READY_DAT
------ ------ ------ --------- -------------------- ---------
RS0079 MB0003 BK0084 12-AUG-24 Completed            13-AUG-24

SQL> -- disable line wrapping
SQL> SET WRAP OFF
SQL>
SQL> -- enable line wrapping for output
SQL> SET SERVEROUTPUT ON FORMAT WRAPPED
SQL>
SQL> BEGIN
  2      prc_reserve_book(3, 1);
  3  END;
  4  /

------------------------------------------------------------------
BOOK RESERVATION
------------------------------------------------------------------
Reservation ID: RS0124
Status        : Reserved
Member ID     : MB0003
Member Name   : Ava Ong

Book ID        Book Name                              ISBN
------------------------------------------------------------------
BK0001         Blood natural pretty worker.           978000000000

Waiting Person Count: 2
Estimated Borrow Date: 07-MAY-2025

You have reserved the book BK0001 successfully!

PL/SQL procedure successfully completed.
```

After reserve success:

```
SQL> select * from reservation where member_id = 'MB0003';

RESERV MEMBER BOOK_I RSV_DATET RSV_STATUS           READY_DAT
------ ------ ------ --------- -------------------- ---------
RS0079 MB0003 BK0084 12-AUG-24 Completed            13-AUG-24
RS0124 MB0003 BK0001 03-MAY-25 Reserved
```

### 4.1.4 Procedure 2: Process Member Book Loan Duration

**Purpose: The purpose of this procedure is to process a member's request to extend the loan duration of borrowed books by validating eligibility, recording the extension, and updating the return date. It then generates a detailed confirmation showing the updated loan information for all books under the specified borrow record.**

**Procedure code:**

```
CREATE OR REPLACE PROCEDURE prc_extend_loan (
    p_member_id     IN  Member.member_id%TYPE,
    P_borrow_id     IN  Borrow.borrow_id%TYPE,
```

```
    p_day_extend    IN  NUMBER
) AS
    v_formatted_member_id       Member.member_id%TYPE;
    v_formatted_borrow_id       Borrow.borrow_id%TYPE;
    v_formatted_extension_id    Loan_Extension.extension_id%TYPE;
    v_no_of_borrow              NUMBER;
    v_old_return_datetime       Borrow.return_datetime%TYPE;
    v_eligible                  NUMBER;
BEGIN
    -- format member_id and borrow_id
    v_formatted_member_id := 'MB' || LPAD(p_member_id, 4, '0');
    v_formatted_borrow_id := 'BR' || LPAD(p_borrow_id, 4, '0');

    -- check whether member and borrow exist
    SELECT COUNT(br.borrow_id), MAX(br.return_datetime)
    INTO v_no_of_borrow, v_old_return_datetime
    FROM Borrow br
    JOIN Borrow_Detail bd
        ON br.borrow_id = bd.borrow_id
    WHERE br.member_id = v_formatted_member_id
        AND br.borrow_id = v_formatted_borrow_id
        AND bd.borrow_status = 'Borrowed';

    IF v_no_of_borrow < 1 THEN
        DBMS_OUTPUT.PUT_LINE('Borrow ' || v_formatted_borrow_id || '
not found.');
        RETURN;
    END IF;

    -- check whether the member is eligible to apply loan extension
    prc_check_loan_extension_eligibility (
        p_member_id     =>  v_formatted_member_id,
        p_borrow_id     =>  v_formatted_borrow_id,
        p_day_extend    =>  p_day_extend,
        p_eligible      =>  v_eligible
    );

    IF v_eligible = 0 THEN
        DBMS_OUTPUT.PUT_LINE('Sorry, you are not eligible to apply
loan extension.');
        RETURN;
    END IF;

    BEGIN
        SAVEPOINT before_insert_update;
        -- insert loan extension
        INSERT INTO Loan_Extension (borrow_id, days_extended,
le_datetime)
        VALUES (v_formatted_borrow_id, p_day_extend, SYSDATE)
        RETURNING extension_id INTO v_formatted_extension_id;

        -- update borrow record
        UPDATE Borrow
        SET return_datetime = return_datetime + p_day_extend
        WHERE borrow_id = v_formatted_borrow_id;
    EXCEPTION
        WHEN OTHERS THEN
            ROLLBACK TO before_insert_update;
            DBMS_OUTPUT.PUT_LINE('Error occured: ' || SQLERRM);
            RETURN;
    END;

    -- successful message
    DECLARE
        msg_member_name Member.name%TYPE;
        msg_no_of_books  NUMBER;
    BEGIN
        -- obtain member name and number of books borrowed
```

```
        SELECT m.name, COUNT(bd.book_id)
        INTO msg_member_name, msg_no_of_books
        FROM Borrow br
        JOIN Borrow_Detail bd
            ON br.borrow_id = bd.borrow_id
        JOIN Member m
            ON br.member_id = m.member_id
        WHERE br.borrow_id = v_formatted_borrow_id
        GROUP BY m.name;

        DBMS_OUTPUT.PUT_LINE(CHR(13));
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 20, '-') || RPAD('-', 15, '-')
|| RPAD('-', 40, '-') || RPAD('-', 12, '-') || RPAD('-', 25, '-') ||
RPAD('-', 32, '-'));
        DBMS_OUTPUT.PUT_LINE('BOOK LOAN EXTENSION');
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 20, '-') || RPAD('-', 15, '-')
|| RPAD('-', 40, '-') || RPAD('-', 12, '-') || RPAD('-', 25, '-')
||RPAD('-', 32, '-'));
        DBMS_OUTPUT.PUT_LINE('Loan Extension ID: ' ||
v_formatted_extension_id);
        DBMS_OUTPUT.PUT_LINE('Borrow ID        : ' ||
v_formatted_borrow_id);
        DBMS_OUTPUT.PUT_LINE('Member ID        : ' ||
v_formatted_member_id);
        DBMS_OUTPUT.PUT_LINE('Member Name      : ' ||
msg_member_name);
        DBMS_OUTPUT.PUT_LINE('Days Extended    : ' || p_day_extend);
        DBMS_OUTPUT.PUT_LINE(CHR(13));

        DBMS_OUTPUT.PUT_LINE(RPAD('-', 20, '-') || RPAD('-', 15, '-')
|| RPAD('-', 40, '-') || RPAD('-', 12, '-') || RPAD('-', 25, '-')
||RPAD('-', 32, '-'));
        DBMS_OUTPUT.PUT_LINE(RPAD('Borrow Detail ID', 20) ||
RPAD('Book ID', 15) || RPAD('Book Name', 40) || RPAD('Borrow
Datetime', 25) || 'Return Datetime');
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 20, '-') || RPAD('-', 15, '-')
|| RPAD('-', 40, '-') || RPAD('-', 12, '-') || RPAD('-', 25, '-')
||RPAD('-', 32, '-'));

        FOR rec IN (
            SELECT bd.borrow_detail_id, b.book_id, b.name,
br.borrow_datetime, br.return_datetime
            FROM Borrow_Detail bd
            JOIN Book b ON bd.book_id = b.book_id
            JOIN Borrow br ON bd.borrow_id = br.borrow_id
            WHERE br.borrow_id = v_formatted_borrow_id
            AND bd.borrow_status = 'Borrowed'
            ORDER BY bd.borrow_detail_id ASC
        ) LOOP
            DBMS_OUTPUT.PUT_LINE(RPAD(rec.borrow_detail_id, 20) ||
RPAD(rec.book_id, 15) ||
                                 RPAD(rec.name, 40) ||
RPAD(TO_CHAR(rec.borrow_datetime, 'DD-MON-YYYY HH24:MI:SS'), 25) ||
TO_CHAR(v_old_return_datetime, 'DD-MON-YYYY HH24:MI:SS') ||
                                 ' -> ' || TO_CHAR(rec.return_datetime,
'DD-MON-YYYY HH24:MI:SS'));
        END LOOP;
    END;

    DBMS_OUTPUT.PUT_LINE(CHR(13));
    DBMS_OUTPUT.PUT_LINE('Loan extension is applied successfully!');
END;
/
```

**Sample Output:**

```
SQL> @"C:\Users\limju\OneDrive - student.tarc.edu.my\RSDY2S3\BACS3183 ADVANCED DATABASE MANAGEMENT\Assignment\BACS3183-ADVANCED-DATABASE-MANAGEME
NT\ChiaMingYi\user_script\loan_extension_user_script.txt"

--------------------------------------------------------------------------------------------------------------------------------------------------------------
BOOK LOAN EXTENSION
--------------------------------------------------------------------------------------------------------------------------------------------------------------
Loan Extension ID: LE0151
Borrow ID        : BR0151
Member ID        : MB0001
Member Name      : Adam Tan
Days Extended    : 2

--------------------------------------------------------------------------------------------------------------------------------------------------------------
Borrow Detail ID    Book ID         Book Name                                          Borrow Datetime           Return Datetime
--------------------------------------------------------------------------------------------------------------------------------------------------------------
BD0151              BK0001          Blood natural pretty worker.                       04-MAY-2025 11:48:58      06-MAY-2025 11:48:58 -> 08-MAY-2025 11:48:58
BD0152              BK0002          The.                                               04-MAY-2025 11:48:58      06-MAY-2025 11:48:58 -> 08-MAY-2025 11:48:58

Loan extension is applied successfully!

PL/SQL procedure successfully completed.

SQL> SELECT * FROM MONTHLY_AUDIT;

AUDIT_ NO_OF_BORROW NO_OF_RETURN NO_OF_OVERDUE NO_OF_FINE_PAYMENT NO_OF_LOAN_EXTENSION NO_OF_RESERVATION CREATED_D LATEST_DA
------ ------------ ------------ ------------- ------------------ --------------------- ------------------ -------- --------
MA0001            2            0             0                  0                     1                  0 04-MAY-25 04-MAY-25
```

### 4.1.5 Trigger 1: Trigger to Update Monthly Loan Extension Count

**Purpose: The purpose of this trigger is to update the Monthly_Audit table every time a loan extension is recorded, incrementing the count of loan extensions for the current month. It also updates the latest_datetime to reflect the current date and time of the extension.**

**Trigger code:**

```
CREATE OR REPLACE TRIGGER trg_record_loan_extension
AFTER INSERT ON Loan_Extension
FOR EACH ROW
BEGIN
  UPDATE Monthly_Audit
  SET no_of_loan_extension = no_of_loan_extension + 1,
      latest_datetime = SYSDATE
  WHERE created_datetime = TRUNC(SYSDATE, 'MM');
END;
/
```

**Sample Output:**

```
SQL> SELECT * FROM MONTHLY_AUDIT;

AUDIT_ NO_OF_BORROW NO_OF_RETURN NO_OF_OVERDUE NO_OF_FINE_PAYMENT NO_OF_LOAN_EXTENSION NO_OF_RESERVATION CREATED_D LATEST_DA
------ ------------ ------------ ------------- ------------------ --------------------- ------------------ -------- --------
MA0001            2            0             0                  0                     0                  0 04-MAY-25 04-MAY-25

SQL> @"C:\Users\limju\OneDrive - student.tarc.edu.my\RSDY2S3\BACS3183 ADVANCED DATABASE MANAGEMENT\Assignment\BACS3183-ADVANCED-DATABASE-MANAGEME
NT\ChiaMingYi\user_script\loan_extension_user_script.txt"

--------------------------------------------------------------------------------------------------------------------------------------------------------------
BOOK LOAN EXTENSION
--------------------------------------------------------------------------------------------------------------------------------------------------------------
Loan Extension ID: LE0151
Borrow ID        : BR0151
Member ID        : MB0001
Member Name      : Adam Tan
Days Extended    : 3

--------------------------------------------------------------------------------------------------------------------------------------------------------------
Borrow Detail ID    Book ID         Book Name                                          Borrow Datetime           Return Datetime
--------------------------------------------------------------------------------------------------------------------------------------------------------------
BD0151              BK0003          Role network even.                                 04-MAY-2025 12:03:19      06-MAY-2025 12:03:19 -> 09-MAY-2025 12:03:19
BD0152              BK0004          Recently material social.                          04-MAY-2025 12:03:19      06-MAY-2025 12:03:19 -> 09-MAY-2025 12:03:19

Loan extension is applied successfully!

PL/SQL procedure successfully completed.

SQL> SELECT * FROM MONTHLY_AUDIT;

AUDIT_ NO_OF_BORROW NO_OF_RETURN NO_OF_OVERDUE NO_OF_FINE_PAYMENT NO_OF_LOAN_EXTENSION NO_OF_RESERVATION CREATED_D LATEST_DA
------ ------------ ------------ ------------- ------------------ --------------------- ------------------ -------- --------
MA0001            2            0             0                  0                     1                  0 04-MAY-25 04-MAY-25
```

### 4.1.6 Trigger 2: Monthly Reservation Audit Tracker

**Purpose: The purpose of this trigger is to automatically updates the Monthly_Audit table by incrementing the reservation count and updating the latest activity timestamp whenever a new reservation is made. It ensures accurate tracking of reservation activity per month.**

**Trigger code:**

```
CREATE OR REPLACE TRIGGER trg_record_reservation
AFTER INSERT ON Reservation
FOR EACH ROW
BEGIN
  UPDATE Monthly_Audit
  SET no_of_reservation = no_of_reservation + 1,
      latest_datetime = SYSDATE
  WHERE TO_CHAR(created_datetime, 'YYYY-MM') = TO_CHAR(SYSDATE,
'YYYY-MM');
END;
/
```

**Sample Output:**

```
SQL> SELECT * FROM MONTHLY_AUDIT;

AUDIT_ NO_OF_BORROW NO_OF_RETURN NO_OF_OVERDUE NO_OF_FINE_PAYMENT NO_OF_LOAN_EXTENSION NO_OF_RESERVATION CREATED_D LATEST_DA
------ ------------ ------------ ------------ ------------------ -------------------- ----------------- --------- ---------
MA0001            2            0            0                  0                    0                 0 04-MAY-25 04-MAY-25

SQL> @"C:\Users\limju\OneDrive - student.tarc.edu.my\RSDY2S3\BACS3183 ADVANCED DATABASE MANAGEMENT\Assignment\BACS3183-ADVANCE
NT\ChiaMingYi\user_script\reserve_book_user_script.txt"

----------------------------------------------------------------
BOOK RESERVATION
----------------------------------------------------------------
Reservation ID: RS0121
Status        : Reserved
Member ID     : MB0002
Member Name   : Zoe Wong

Book ID        Book Name                                ISBN
---------------------------------------------------------------
BK0001         Blood natural pretty worker.             978000000000

Waiting Person Count: 1
Estimated Borrow Date: 06-MAY-2025

You have reserved the book BK0001 successfully!

PL/SQL procedure successfully completed.

SQL> SELECT * FROM MONTHLY_AUDIT;

AUDIT_ NO_OF_BORROW NO_OF_RETURN NO_OF_OVERDUE NO_OF_FINE_PAYMENT NO_OF_LOAN_EXTENSION NO_OF_RESERVATION CREATED_D LATEST_DA
------ ------------ ------------ ------------ ------------------ -------------------- ----------------- --------- ---------
MA0001            2            0            0                  0                    0                 1 04-MAY-25 04-MAY-25
```

### 4.1.7 Report 1: Overdue Borrowed Books Report with Member Details and Due Dates

**Purpose: The purpose of this report is to identify all members who have at least one overdue book (marked as 'Missing') and list the overdue books along with member contact details and due dates. It helps the library track overdue returns and follow up with responsible members efficiently.**

**Report code:**

```
CREATE OR REPLACE PROCEDURE report_borrowed_books_with_due_dates AS
    TYPE report_record IS RECORD (
        member_id      VARCHAR2(15),
        member_name    VARCHAR2(25),
        tel_no         VARCHAR2(15),
        book_name      VARCHAR2(40),
        due_date       DATE,
        overdue_days   NUMBER
    );

    TYPE report_table IS TABLE OF report_record;
    report_data report_table := report_table();

    total_members  NUMBER := 0;
    total_books    NUMBER := 0;
    total_records  NUMBER := 0;

    TYPE member_table IS TABLE OF BOOLEAN INDEX BY VARCHAR2(15);
    member_set member_table;

    CURSOR overdue_cursor IS
```

```
        SELECT *
        FROM vw_overdue_borrowed_books
        ORDER BY overdue_days DESC;

BEGIN
    DBMS_OUTPUT.NEW_LINE;
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 125, '='));
    DBMS_OUTPUT.PUT_LINE(RPAD('= Overdue Borrowed Books Report with
Member Details and Due Dates', 124) || '=');
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 125, '='));
    DBMS_OUTPUT.NEW_LINE;

    FOR rec IN overdue_cursor LOOP
        IF NOT member_set.EXISTS(rec.member_id) THEN
            member_set(rec.member_id) := TRUE;
            total_members := total_members + 1;
        END IF;

        total_books := total_books + 1;
        total_records := total_records + 1;

        report_data.EXTEND;
        report_data(report_data.LAST) := report_record(
            rec.member_id,
            rec.member_name,
            rec.tel_no,
            NVL(rec.book_name, 'Unknown Book'),
            rec.return_datetime,
            rec.overdue_days
        );
    END LOOP;

    DBMS_OUTPUT.PUT_LINE(
        RPAD('Member ID', 15) ||
        RPAD('Member Name', 25) ||
        RPAD('Tel No', 15) ||
        RPAD('Book Name', 40) ||
        RPAD('Due Date', 15) ||
        RPAD('Overdue Days', 15)
    );
    DBMS_OUTPUT.PUT_LINE(
        RPAD('-', 15, '-') ||
        RPAD('-', 25, '-') ||
        RPAD('-', 15, '-') ||
        RPAD('-', 40, '-') ||
        RPAD('-', 15, '-') ||
        RPAD('-', 15, '-')
    );

    FOR i IN 1 .. report_data.LAST LOOP
        DBMS_OUTPUT.PUT_LINE(
            RPAD(report_data(i).member_id, 15) ||
            RPAD(report_data(i).member_name, 25) ||
            RPAD(report_data(i).tel_no, 15) ||
            RPAD(report_data(i).book_name, 40) ||
            RPAD(TO_CHAR(report_data(i).due_date, 'YYYY-MM-DD'), 15)
||
            RPAD(TO_CHAR(report_data(i).overdue_days), 15)
        );
    END LOOP;

    DBMS_OUTPUT.NEW_LINE;
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 125, '='));
    DBMS_OUTPUT.PUT_LINE(RPAD('= Report Summary', 124) || '=');
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 125, '='));
    DBMS_OUTPUT.PUT_LINE(RPAD('Total Overdue Members: ' ||
total_members, 125));
```

```
    DBMS_OUTPUT.PUT_LINE(RPAD('Total Overdue Books:    ' ||
total_books, 125));
    DBMS_OUTPUT.PUT_LINE(RPAD('Total Records:          ' ||
total_records, 125));
END report_borrowed_books_with_due_dates;
/
```

**Sample Output:**

```
================================================================================================================================
= Overdue Borrowed Books Report with Member Details and Due Dates                                                             =
================================================================================================================================
Member ID      Member Name          Tel No         Book Name                          Due Date       Overdue Days
--------------------------------------------------------------------------------------------------------------------------------
MB0054         Emma Teoh            011-4234620    Rate safe account.                 2024-12-03     138
MB0082         Sophia Wong          011-2234648    Car forget.                        2024-12-05     136
MB0060         Lily Wong            011-0234626    Citizen no your.                   2024-12-07     134
MB0034         Ella Chan            011-4234600    Drive hotel land.                  2024-12-19     122
MB0071         Nathan Teoh          011-1234637    Thousand.                          2024-12-20     121
MB0053         Logan Lim            011-3234619    Soon woman draw.                   2024-12-20     121
MB0067         Liam Goh             011-7234633    Likely.                            2024-12-21     120
MB0050         Charlotte Lim        011-0234616    Model indicate.                    2024-12-24     117
MB0048         Olivia Chua          011-8234614    Result.                            2024-12-29     112
MB0052         Sophia Wong          011-2234618    Expect task report cell.           2024-12-30     111
MB0082         Sophia Wong          011-2234648    Cut toward through itself.         2024-12-31     110
MB0087         Benjamin Tan         011-7234653    Window challenge political.        2025-01-02     108
MB0063         Ryan Lim             011-3234629    Sister whole.                      2025-01-03     107
MB0099         Luke Wong            011-9234665    Energy traditional.                2025-01-03     107
MB0060         Lily Wong            011-0234626    Enough.                            2025-01-06     104
MB0094         Sofia Goh            011-4234660    Instead compare control protect.   2025-01-19     91
MB0024         Sofia Lim            011-4234590    Instead compare control protect.   2025-01-20     90
MB0094         Sofia Goh            011-4234660    Return big course top.             2025-02-04     75
MB0052         Sophia Wong          011-2234618    Soon woman draw.                   2025-02-04     75
MB0089         Henry Ng             011-9234655    Every international.               2025-02-09     70
MB0052         Sophia Wong          011-2234618    Drop goal plan.                    2025-02-11     68
MB0026         Amelia Chan          011-6234592    When interesting inside.           2025-02-21     58
MB0033         James Lim            011-3234599    Country relationship.              2025-02-21     58
MB0074         Sofia Lim            011-4234640    Energy traditional.                2025-02-27     52
MB0038         Olivia Goh           011-8234604    Expect task report cell.           2025-03-03     48
MB0011         Ryan Tan             011-1234577    Give return.                       2025-03-03     48
MB0016         Hannah Teoh          011-6234582    Role network even.                 2025-03-03     48
MB0082         Sophia Wong          011-2234648    Arrive step always.                2025-03-07     44
MB0029         Luke Ng              011-9234595    Stock traditional approach.        2025-03-07     44
MB0052         Sophia Wong          011-2234618    Arrive room.                       2025-03-09     42
MB0054         Emma Teoh            011-4234620    Work you.                          2025-03-11     40
MB0057         Ethan Lee            011-7234623    Within staff world.                2025-03-15     36
MB0051         Jacob Tan            011-1234617    Fact gas suddenly.                 2025-03-22     29
MB0013         Noah Lee             011-3234579    Quickly yard.                      2025-03-23     28
MB0025         Henry Lau            011-5234591    Benefit fast.                      2025-03-23     28
MB0010         Mia Goh              011-0234576    Answer much.                       2025-03-26     25
================================================================================================================================
= Report Summary                                                                                                              =
================================================================================================================================
Total Overdue Members: 28
Total Overdue Books:   36
Total Records:         36
```

### 4.1.8 Report 2: Top 10 Most Borrowed Books in [Year] with Top 5 Borrowers for Each Book

**Purpose: The purpose of this report is to identifies the 10 most borrowed books in a specified year and highlights the top 5 members who frequently borrowed each one. It helps library management make data-driven decisions for future book acquisitions, recognize reading trends, and engage high-participation members through targeted programs or rewards.**

**Report code:**

```
CREATE OR REPLACE PROCEDURE report_top_10_books (
    p_year IN NUMBER
) AS
    -- Outer cursor: Top 10 most borrowed books
    CURSOR book_cursor IS
        SELECT
            b.book_id,
            b.name,
            b.author,
            COUNT(*) AS borrow_count
        FROM borrow_detail bd
        JOIN borrow br ON bd.borrow_id = br.borrow_id
```

```
        JOIN book b ON bd.book_id = b.book_id
        WHERE TO_CHAR(br.borrow_datetime, 'YYYY') = TO_CHAR(p_year)
        GROUP BY b.book_id, b.name, b.author
        ORDER BY COUNT(*) DESC
        FETCH FIRST 10 ROWS ONLY;

    -- Nested cursor: Top 5 borrowers per book
    CURSOR borrower_cursor(p_book_id book.book_id%TYPE) IS
        SELECT * FROM (
            SELECT
                m.member_id,
                m.name AS member_name,
                COUNT(*) AS times_borrowed
            FROM borrow_detail bd
            JOIN borrow br ON bd.borrow_id = br.borrow_id
            JOIN member m ON br.member_id = m.member_id
            WHERE bd.book_id = p_book_id
              AND TO_CHAR(br.borrow_datetime, 'YYYY') =
TO_CHAR(p_year)
            GROUP BY m.member_id, m.name
            ORDER BY times_borrowed DESC
        )
        WHERE ROWNUM <= 5;  -- Limit to top 5

    v_rank NUMBER := 1;
BEGIN
    DBMS_OUTPUT.PUT_LINE(CHR(10));
    DBMS_OUTPUT.PUT_LINE('Most 10 Popular Books in ' || p_year);
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 90, '='));
    DBMS_OUTPUT.PUT_LINE('| ' || RPAD('Rank', 5) || ' | ' ||
RPAD('Title', 30) || ' | ' ||
                         RPAD('Author', 27) || ' | ' || RPAD('Borrow
Count', 15) || ' |');
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 90, '='));

    FOR book_rec IN book_cursor LOOP
        DBMS_OUTPUT.PUT_LINE('| ' || RPAD(v_rank, 5) || ' | ' ||
                             RPAD(SUBSTR(book_rec.name, 1, 30), 30) ||
' | ' ||
                             RPAD(SUBSTR(book_rec.author, 1, 25), 27)
|| ' | ' ||
                             RPAD(book_rec.borrow_count, 15) || ' |');
        DBMS_OUTPUT.PUT_LINE(RPAD('=', 90, '='));
        DBMS_OUTPUT.PUT_LINE('| ' || LPAD('Member ID', 10) ||
LPAD('Member Name', 30) ||
                               LPAD('Times Borrowed', 20) || ' |');
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 64, '-'));

        FOR borrower_rec IN borrower_cursor(book_rec.book_id) LOOP
            DBMS_OUTPUT.PUT_LINE('| ' ||  LPAD(borrower_rec.member_id,
10) ||
                                    LPAD(borrower_rec.member_name, 30)
||
                                    LPAD(borrower_rec.times_borrowed,
20) || ' |');
        END LOOP;

        DBMS_OUTPUT.PUT_LINE(RPAD('=', 90, '='));
        v_rank := v_rank + 1;
    END LOOP;
END;
/
```

**Sample Output:**

Top 10 Most Borrowed Books in 2025 with Top 5 Borrowers for Each Book

| Rank | Title | Author | Borrow Count |
|---|---|---|---|
| 1 | Within staff world. | Jennifer Harper | 4 |

| Member ID | Member Name | Times Borrowed |
|---|---|---|
| MB0020 | Mia Chua | 1 |
| MB0029 | Luke Ng | 1 |
| MB0044 | Isabella Wong | 1 |
| MB0057 | Ethan Lee | 1 |

| Rank | Title | Author | Borrow Count |
|---|---|---|---|
| 2 | Fish staff top. | Kathy Davis | 4 |

| Member ID | Member Name | Times Borrowed |
|---|---|---|
| MB0013 | Noah Lee | 1 |
| MB0021 | Nathan Lee | 1 |
| MB0027 | Jack Teoh | 1 |
| MB0052 | Sophia Wong | 1 |

| Rank | Title | Author | Borrow Count |
|---|---|---|---|
| 3 | People deal or mean. | Tammy Russell | 4 |

| Member ID | Member Name | Times Borrowed |
|---|---|---|
| MB0057 | Ethan Lee | 1 |
| MB0060 | Lily Wong | 1 |
| MB0064 | Emily Teoh | 1 |
| MB0089 | Henry Ng | 1 |

| Rank | Title | Author | Borrow Count |
|---|---|---|---|
| 4 | Role camera. | Deborah Wood | 3 |

| Member ID | Member Name | Times Borrowed |
|---|---|---|
| MB0019 | Logan Ng | 1 |
| MB0038 | Olivia Goh | 1 |
| MB0051 | Jacob Tan | 1 |

| Rank | Title | Author | Borrow Count |
|---|---|---|---|
| 5 | Wrong factor. | Michael Nelson | 3 |

| Member ID | Member Name | Times Borrowed |
|---|---|---|
| MB0021 | Nathan Lee | 1 |
| MB0051 | Jacob Tan | 1 |
| MB0099 | Luke Wong | 1 |

| Rank | Title | Author | Borrow Count |
|---|---|---|---|
| 6 | Energy traditional. | Mary Warren | 3 |

| Member ID | Member Name | Times Borrowed |
|---|---|---|
| MB0032 | Emma Wong | 1 |
| MB0052 | Sophia Wong | 1 |
| MB0074 | Sofia Lim | 1 |

| Rank | Title | Author | Borrow Count |
|---|---|---|---|
| 7 | Cold network hand. | Roy Cole | 3 |

| Member ID | Member Name | Times Borrowed |
|---|---|---|
| MB0057 | Ethan Lee | 2 |
| MB0013 | Noah Lee | 1 |

| 8 | Sister whole. | Angela Moore DDS | 3 |
|---|---|---|---|

| Member ID | Member Name | Times Borrowed |
|---|---|---|
| MB0021 | Nathan Lee | 2 |
| MB0063 | Ryan Lim | 1 |

| 9 | Return big course top. | Kenneth Stewart | 3 |
|---|---|---|---|

| Member ID | Member Name | Times Borrowed |
|---|---|---|
| MB0002 | Emily Wong | 1 |
| MB0056 | Sophia Goh | 1 |
| MB0094 | Sofia Goh | 1 |

| 10 | Rate safe account. | William Hooper | 3 |
|---|---|---|---|

| Member ID | Member Name | Times Borrowed |
|---|---|---|
| MB0032 | Emma Wong | 1 |
| MB0035 | Michael Lau | 1 |
| MB0054 | Emma Teoh | 1 |

## 4.2 Lim Jun Wei

### 4.2.1 Query 1: Members with Most Overdue Books

**Purpose: The purpose of this query is to identify members who frequently return books late, it is useful for enforcing borrowing policies.**

**SQL statement:**

```
SELECT
    m.member_id,
    m.name,
    COUNT(*) AS overdue_books
FROM
    member m
JOIN
    borrow b ON m.member_id = b.member_id
JOIN
    borrow_detail bd ON b.borrow_id = bd.borrow_id
WHERE
    (b.return_datetime < SYSDATE AND bd.borrow_status = 'Borrowing')
OR
    bd.borrow_status = 'Missing'
GROUP BY
    m.member_id, m.name
ORDER BY
    overdue_books DESC;
```

**Sample Output:**

```
MEMBER NAME                                        OVERDUE_BOOKS
------ ------------------------------------------------- -------------
MB0048 Olivia Chua                                                1
MB0050 Charlotte Lim                                              1
MB0051 Jacob Tan                                                  1
MB0053 Logan Lim                                                  1
MB0057 Ethan Lee                                                  1
MB0063 Ryan Lim                                                   1
MB0067 Liam Goh                                                   1
MB0071 Nathan Teoh                                                1
MB0074 Sofia Lim                                                  1
MB0087 Benjamin Tan                                               1
MB0089 Henry Ng                                                   1

MEMBER NAME                                        OVERDUE_BOOKS
------ ------------------------------------------------- -------------
MB0024 Sofia Lim                                                  1
MB0016 Hannah Teoh                                                1
MB0013 Noah Lee                                                   1
MB0099 Luke Wong                                                  1
MB0010 Mia Goh                                                    1
MB0011 Ryan Tan                                                   1

28 rows selected.
```

### 4.2.2 Query 2: Most Borrowed Books by Genre

**Purpose: The purpose of this query is to help management understand which genres are most popular, useful for acquisitions and stocking.**

**SQL statement:**

```
COLUMN genre HEADING 'Book Genre' FORMAT A20
```

```
COLUMN total_borrowed HEADING 'Total Borrowed' FORMAT 9999

SELECT
    bk.genre,
    COUNT(bd.borrow_detail_id) AS total_borrowed
FROM
    book bk
JOIN
    borrow_detail bd ON bk.book_id = bd.book_id
GROUP BY
    bk.genre
ORDER BY
    total_borrowed DESC;
```

**Sample Output:**

| Book Genre | Total Borrowed |
|---|---|
| fiction | 72 |
| non-fiction | 45 |
| academic | 33 |

### 4.2.3 Procedure 1: Book Borrowing Process for a Member

**Purpose: The purpose of this procedure is to handle the complete book borrowing process for a library member, including availability checks, reservation status, eligibility verification, and transaction logging. It ensures that borrowing rules are enforced and provides the member with a detailed confirmation of their borrow request, supporting efficient circulation management.**

**Procedure statement:**

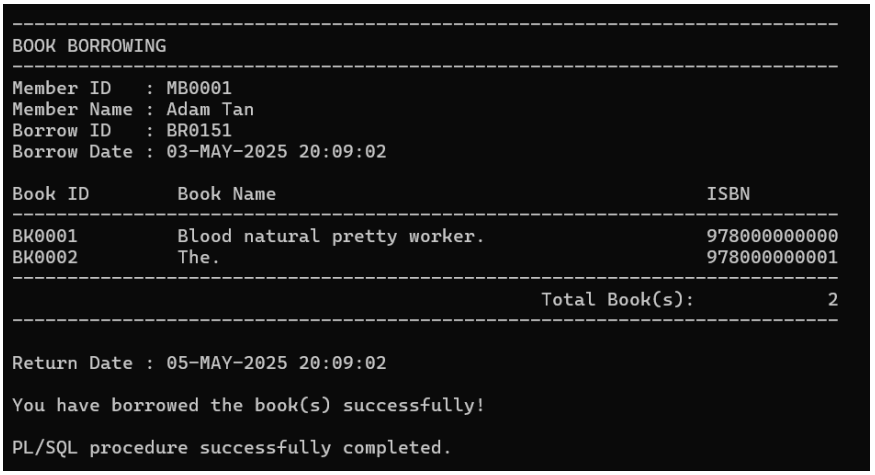
```
CREATE OR REPLACE PROCEDURE prc_borrow_books (
    p_member_id IN borrow.member_id%TYPE,
    p_book_ids  IN book_id_list_t
) AS
    v_formatted_member_id   VARCHAR2(6);
    v_eligible              NUMBER := 0;
    v_borrow_id             borrow.borrow_id%TYPE;
    v_result                NUMBER;
BEGIN
    -- format member_id
    v_formatted_member_id := 'MB' || LPAD(p_member_id, 4, '0');

    -- check book availability
    FOR i IN 1 .. p_book_ids.COUNT LOOP
        DECLARE
            v_formatted_book_id     VARCHAR2(10);
            v_available             VARCHAR2(10);
            v_member_id_borrowing   VARCHAR2(6);
            v_is_reserved           NUMBER;
        BEGIN
            -- format book_id
            v_formatted_book_id := 'BK' || LPAD(p_book_ids(i), 4,
'0');

            prc_check_book_availability(
                p_book_id => v_formatted_book_id,
                p_book_available => v_available,
                p_member_id_borrowing => v_member_id_borrowing
            );
```

```
            IF v_available = 0 THEN
                DBMS_OUTPUT.PUT_LINE('Error: The book ' ||
v_formatted_book_id || ' is not available currently.');

                -- check if the current user has borrowed the book and
not yet return
                IF v_member_id_borrowing = v_formatted_member_id THEN
                    DBMS_OUTPUT.PUT_LINE('Error: You have borrowed the
book ' || v_formatted_book_id || ' and not yet return it.');
                END IF;
                RETURN;
            END IF;

            -- check if the book has been reserved
            prc_check_book_reservation(
                p_book_id => v_formatted_book_id,
                p_is_reserved => v_is_reserved
            );

            IF v_is_reserved = 1 THEN
                DBMS_OUTPUT.PUT_LINE('Error: Sorry, the book ' ||
v_formatted_book_id || ' has been reserved by other member currently.
You may apply a book reservation right now to get your interested book
early!');
                RETURN;
            END IF;
        EXCEPTION
            WHEN NO_DATA_FOUND THEN
                RETURN;
        END;
    END LOOP;

    -- check membership level
    prc_check_user_borrow(
        p_member_id => v_formatted_member_id,
        p_borrow_count => p_book_ids.COUNT,
        p_eligible => v_eligible
    );

    -- check if the member is eligible to borrow books
    IF v_eligible = 0 THEN
        DBMS_OUTPUT.PUT_LINE('Error: Sorry. You are not eligible to
borrow the books.');
        RETURN;
    END IF;

    -- insert borrow record
    prc_insert_borrow(
        p_member_id => v_formatted_member_id,
        p_borrow_id => v_borrow_id,
        p_result => v_result
    );

    -- check if borrow record is inserted successfully
    IF v_result = 0 THEN
        DBMS_OUTPUT.PUT_LINE('Error: Failed inserting borrow
record.');
        RETURN;
    END IF;

    -- insert borrow_detail record and update book availability
    prc_insert_borrow_detail(
        p_borrow_id => v_borrow_id,
        p_book_ids => p_book_ids,
        p_result => v_result
    );

    -- check if borrow detail record is inserted successfully
```

```
    IF v_result = 0 THEN
        DBMS_OUTPUT.PUT_LINE('Error: Failed inserting borrow
record.');
        RETURN;
    END IF;

    DECLARE
        msg_borrow_datetime Borrow.borrow_datetime%TYPE;
        msg_return_datetime Borrow.return_datetime%TYPE;
        msg_member_name Member.name%TYPE;
    BEGIN
        SELECT m.name, br.borrow_datetime, br.return_datetime
        INTO msg_member_name, msg_borrow_datetime, msg_return_datetime
        FROM Borrow br
        JOIN Member m
            ON br.member_id = m.member_id
        WHERE br.borrow_id = v_borrow_id;

        -- successful message
        DBMS_OUTPUT.PUT_LINE(CHR(13));
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 15, '-') || RPAD('-', 48, '-')
|| RPAD('-', 12, '-'));
        DBMS_OUTPUT.PUT_LINE('BOOK BORROWING');
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 15, '-') || RPAD('-', 48, '-')
|| RPAD('-', 12, '-'));
        DBMS_OUTPUT.PUT_LINE('Member ID   : ' ||
v_formatted_member_id);
        DBMS_OUTPUT.PUT_LINE('Member Name : ' || msg_member_name);
        DBMS_OUTPUT.PUT_LINE('Borrow ID   : ' || v_borrow_id);
        DBMS_OUTPUT.PUT_LINE('Borrow Date : ' ||
TO_CHAR(msg_borrow_datetime, 'DD-MON-YYYY HH24:MI:SS'));
        DBMS_OUTPUT.PUT_LINE(CHR(13));

        DBMS_OUTPUT.PUT_LINE(RPAD('Book ID', 15) || RPAD('Book Name',
48) || 'ISBN');
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 15, '-') || RPAD('-', 48, '-')
|| RPAD('-', 12, '-'));

        FOR i IN 1 .. p_book_ids.COUNT LOOP
            DECLARE
                v_formatted_book_id VARCHAR2(10);
                v_book_name Book.name%TYPE;
                v_book_isbn Book.isbn%TYPE;
                v_return_date DATE;
            BEGIN
                v_formatted_book_id := 'BK' || LPAD(p_book_ids(i), 4,
'0');

                SELECT name, isbn
                INTO v_book_name, v_book_isbn
                FROM Book
                WHERE book_id = v_formatted_book_id;

                DBMS_OUTPUT.PUT_LINE(RPAD(v_formatted_book_id, 15) ||
RPAD(v_book_name, 48) || LPAD(v_book_isbn, 12));
            EXCEPTION
                WHEN NO_DATA_FOUND THEN
                    DBMS_OUTPUT.PUT_LINE(RPAD(v_formatted_book_id, 15)
|| RPAD('Name Not Found', 48) || LPAD('N/A', 12));
            END;
        END LOOP;

        DBMS_OUTPUT.PUT_LINE(RPAD('-', 15, '-') || RPAD('-', 48, '-')
|| RPAD('-', 12, '-'));
        DBMS_OUTPUT.PUT_LINE(LPAD('Total Book(s): ', 63) ||
LPAD(p_book_ids.COUNT, 12));
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 15, '-') || RPAD('-', 48, '-')
|| RPAD('-', 12, '-'));
```

```
        DBMS_OUTPUT.PUT_LINE(CHR(13));
        DBMS_OUTPUT.PUT_LINE('Return Date : ' ||
TO_CHAR(msg_return_datetime, 'DD-MON-YYYY HH24:MI:SS'));

        DBMS_OUTPUT.PUT_LINE(CHR(13));
        DBMS_OUTPUT.PUT_LINE('You have borrowed the book(s)
successfully!');
    END;
END;
/
```

Scenario 1: Member MB0001 borrows books BK0001 and BK0002 which are currently available for borrowing.

| | |
|---|---|
| Input | DECLARE<br>v_book_ids book_id_list_t := book_id_list_t();<br>BEGIN<br>v_book_ids.EXTEND(2);<br>v_book_ids(1) := 1;<br>v_book_ids(2) := 2;<br><br>prc_borrow_books(2, v_book_ids);<br>END;<br>/ |
| Output |  |

Scenario 2: Member MB0005 borrows books BK0001 and BK0002 which are currently being borrowed by Member MB0001.

| | |
|---|---|
| Input | DECLARE<br>v_book_ids book_id_list_t := book_id_list_t();<br>BEGIN<br>-- Add book IDs to the collection<br>v_book_ids.EXTEND(2); -- Adjust the number based on how many books you're adding<br>v_book_ids(1) := 1; -- Replace with actual book ID<br>v_book_ids(2) := 2; -- Replace with actual book ID<br><br>-- Call the procedure with member_id and book_id list |

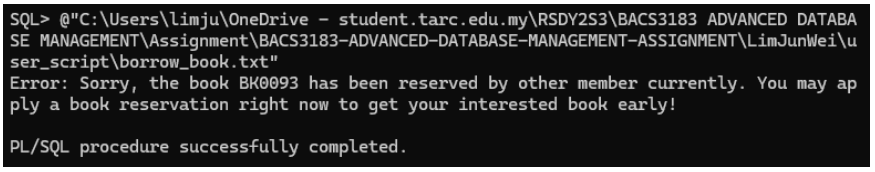
| | |
|---|---|
| | prc_borrow_books(5, v_book_ids); -- Replace 123 with actual member_id<br>END;<br>/ |
| Output | SQL> @"C:\Users\limju\OneDrive - student.tarc.edu.m<br>NT\LimJunWei\user_script\borrow_book.txt"<br>Error: The book BK0001 is not available currently.<br>Error: The book BK0001 is not available currently.<br><br>PL/SQL procedure successfully completed. |

Scenario 3: Member MB0009 borrows books BK0156 and BK0160 which do not exist in the Book table.

| | |
|---|---|
| Input | DECLARE<br>v_book_ids book_id_list_t := book_id_list_t();<br>BEGIN<br>-- Add book IDs to the collection<br>v_book_ids.EXTEND(2); -- Adjust the number based on how many books you're adding<br>v_book_ids(1) := 156; -- Replace with actual book ID<br>v_book_ids(2) := 160; -- Replace with actual book ID<br><br>-- Call the procedure with member_id and book_id list<br>prc_borrow_books(9, v_book_ids); -- Replace 123 with actual member_id<br>END;<br>/ |
| Output | SQL> @"C:\Users\limju\OneDrive - student.tarc.edu.my<br>Error: No data found for book_id: BK0156<br>Error: The book BK0156 is not available currently.<br><br>PL/SQL procedure successfully completed. |

Scenario 4: Member MB0013 borrows book BK0093 which is currently reserved by another member.

| | |
|---|---|
| Input | DECLARE<br>v_book_ids book_id_list_t := book_id_list_t();<br>BEGIN<br>-- Add book IDs to the collection<br>v_book_ids.EXTEND(1);<br>v_book_ids(1) := 93;<br><br>-- Call the procedure with member_id and book_id list<br>prc_borrow_books(13, v_book_ids); -- Replace 123 with actual member_id<br>END;<br>/ |

| Output |  |
|---|---|



### 4.2.4 Procedure 2: Insert Borrow Record Based on Membership Policy

**Purpose: The purpose of this procedure is to insert a new borrow record for a member by calculating the return date based on their membership's maximum borrow duration. It ensures borrowing policies are followed and returns the borrow ID for further processing, enabling traceable and rule-compliant lending transactions. It was also used to support the main 'prc_borrow_books' procedure.**

**Procedure statement:**

```
CREATE OR REPLACE PROCEDURE prc_insert_borrow (
    p_member_id IN  borrow.member_id%TYPE,
    p_borrow_id OUT borrow.borrow_id%TYPE,
    p_result    OUT NUMBER
) AS
    v_max_borrow_day    membership.max_borrow_day%TYPE;
    v_borrow_datetime   borrow.borrow_datetime%TYPE;
    v_return_datetime   borrow.return_datetime%TYPE;
BEGIN
    BEGIN
        SAVEPOINT before_insert;
        -- obtain the membership max borrow day
        SELECT max_borrow_day
        INTO v_max_borrow_day
        FROM Membership ms
        JOIN Member m
        ON ms.membership_id = m.membership_id
        WHERE m.member_id = p_member_id;

        v_borrow_datetime := SYSDATE;
        v_return_datetime := v_borrow_datetime + v_max_borrow_day;

        -- insert new borrow record
        INSERT INTO Borrow (member_id, borrow_datetime,
return_datetime)
        VALUES (p_member_id, v_borrow_datetime, v_return_datetime)
        RETURNING borrow_id INTO p_borrow_id;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Error: No membership found for
member ID ' || p_member_id);
            ROLLBACK TO before_insert;
            p_borrow_id := NULL;
            p_result := 0;

        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
            ROLLBACK TO before_insert;
            p_borrow_id := NULL;
            p_result := 0;
    END;
    COMMIT;
    p_result := 1;
END;
/
```

Scenario 1: Member MB0001 borrows books BK0001 and BK0002 successfully.

| | |
|---|---|
| Output | After member MB001 has successfully borrowed the books, the borrow record will be inserted. |

```
--------------------------------------------------------------------------------
BOOK BORROWING
--------------------------------------------------------------------------------
Member ID   : MB0001
Member Name : Adam Tan
Borrow ID   : BR0151
Borrow Date : 03-MAY-2025 20:09:02

Book ID        Book Name                                          ISBN
--------------------------------------------------------------------------------
BK0001         Blood natural pretty worker.                       978000000000
BK0002         The.                                               978000000001
--------------------------------------------------------------------------------
                                                  Total Book(s):            2
--------------------------------------------------------------------------------

Return Date : 05-MAY-2025 20:09:02

You have borrowed the book(s) successfully!

PL/SQL procedure successfully completed.
```

```
SQL> SELECT * FROM BORROW WHERE BORROW_ID = 'BR0151';

BORROW MEMBER BORROW_DA RETURN_DA
------ ------ --------- ---------
BR0151 MB0001 03-MAY-25 05-MAY-25
```

### 4.2.5 Trigger 1: Update Book Availability and Monthly Borrow Audit

**Purpose: The purpose of this trigger is to automatically updates the book's availability status to 'Not Available' when a new borrow detail is recorded with a 'Borrowed' status. It also increments the monthly borrow count in the Monthly_Audit table, ensuring real-time tracking of borrowing activities.**

**Trigger code:**

```
CREATE OR REPLACE TRIGGER trg_update_book_availability
AFTER INSERT ON Borrow_Detail
FOR EACH ROW
WHEN (NEW.borrow_status = 'Borrowed')
BEGIN
    UPDATE Book
    SET availability = 'N'
    WHERE book_id = :NEW.book_id;

    UPDATE Monthly_Audit
    SET no_of_borrow = no_of_borrow + 1,
        latest_datetime = SYSDATE
    WHERE created_datetime = TRUNC(SYSDATE, 'MM');
END;
/
```

Scenario 1: Member MB0021 borrows books BK0004, BK0061 and BK0020.

| | |
|---|---|
| Input | DECLARE<br>v_book_ids book_id_list_t := book_id_list_t();<br>BEGIN<br>-- Add book IDs to the collection<br>v_book_ids.EXTEND(3);<br>v_book_ids(1) := 4;<br>v_book_ids(2) := 61;<br>v_book_ids(3) := 20; |

```
    -- Call the procedure with member_id and book_id list
    prc_borrow_books(21, v_book_ids); -- Replace 123 with actual
member_id
END;
/
```

Output

```
SQL> @"C:\Users\limju\OneDrive - student.tarc.edu.my\RSDY2S3\BACS3183 ADVANCED

--------------------------------------------------------------------------------
BOOK BORROWING
--------------------------------------------------------------------------------
Member ID   : MB0021
Member Name : Ethan Lim
Borrow ID   : BR0152
Borrow Date : 03-MAY-2025 22:55:44

Book ID        Book Name                                          ISBN
--------------------------------------------------------------------------------
BK0004         Recently material social.                          978000000003
BK0061         Arrive room.                                       978000000060
BK0020         Quite fish.                                        978000000019
--------------------------------------------------------------------------------
                                                 Total Book(s):            3
--------------------------------------------------------------------------------

Return Date : 06-MAY-2025 22:55:44

You have borrowed the book(s) successfully!

PL/SQL procedure successfully completed.
```

After the new borrow record is inserted into the Borrow table, the borrowed book availability in the Book table will be updated into 'N' automatically.

```
SQL> SELECT BOOK_ID, NAME, AVAILABILITY FROM BOOK WHERE BOOK_ID IN ('BK0004', 'BK0061', 'BK0020');

BOOK_I NAME                                               A
------ -------------------------------------------------- -
BK0004 Recently material social.                          N
BK0020 Quite fish.                                        N
BK0061 Arrive room.                                       N
```

Meanwhile, the no_of_borrow in Monthly_Audit table will also be updated automatically.

Before borrowing books:

```
SQL> SELECT * FROM MONTHLY_AUDIT;

AUDIT_ NO_OF_BORROW NO_OF_RETURN NO_OF_OVERDUE NO_OF_FINE_PAYMENT NO_OF_LOAN_EXTENSION NO_OF_RESERVATION CREATED_D LATEST_DA
------ ------------ ------------ ------------ ------------------ -------------------- ---------------- --------- ---------
MA0001            0            0            0                  0                    0                0 03-MAY-25 03-MAY-25
```

After borrowed 3 books:

```
SQL> SELECT * FROM MONTHLY_AUDIT;

AUDIT_ NO_OF_BORROW NO_OF_RETURN NO_OF_OVERDUE NO_OF_FINE_PAYMENT NO_OF_LOAN_EXTENSION NO_OF_RESERVATION CREATED_D LATEST_DA
------ ------------ ------------ ------------ ------------------ -------------------- ---------------- --------- ---------
MA0001            3            0            0                  0                    0                0 03-MAY-25 03-MAY-25
```

### 4.2.6 Trigger 2: Auto Update Membership Level Based on Credit and Duration

**Purpose: The purpose of this trigger is to ensure that when a member's credit is updated, it is capped at a maximum of 100 and evaluated against membership criteria. After the update, it automatically adjusts the member's membership level based on their current credit and registration duration, keeping membership tiers accurate and up to date.**

**Trigger code:**

```
CREATE INDEX idx_membership_credit_duration
ON Membership(credit DESC, duration_day DESC);

CREATE OR REPLACE TRIGGER trg_check_membership_lvl_based_credit
FOR UPDATE OF credit ON Member
COMPOUND TRIGGER

  TYPE MemberIDList IS TABLE OF Member.member_id%TYPE INDEX BY
PLS_INTEGER;
  member_ids MemberIDList;
  idx PLS_INTEGER := 0;

  BEFORE EACH ROW IS
  BEGIN
    -- Cap the credit at 100
    IF :NEW.credit > 100 THEN
      :NEW.credit := 100;
    END IF;

    -- Collect member IDs for post-statement processing
    idx := idx + 1;
    member_ids(idx) := :NEW.member_id;
  END BEFORE EACH ROW;

  AFTER STATEMENT IS
    v_updated_membership_id Membership.membership_id%TYPE;
  BEGIN
    FOR i IN 1 .. member_ids.COUNT LOOP
      BEGIN
        -- Use the view to get the best matching membership level
        SELECT membership_id
        INTO v_updated_membership_id
        FROM vw_eligible_membership
        WHERE member_id = member_ids(i)
        ORDER BY required_credit DESC, required_duration DESC
        FETCH FIRST 1 ROWS ONLY;

        -- Update the member's membership if it has changed
        UPDATE Member
        SET membership_id = v_updated_membership_id
        WHERE member_id = member_ids(i)
          AND membership_id != v_updated_membership_id;

      EXCEPTION
        WHEN NO_DATA_FOUND THEN
          DBMS_OUTPUT.PUT_LINE('Error: No valid membership found for
member_id: ' || member_ids(i));
        WHEN OTHERS THEN
          DBMS_OUTPUT.PUT_LINE('Error: Failed to update membership for
member_id: ' || member_ids(i) || '. ' || SQLERRM);
      END;
    END LOOP;
  END AFTER STATEMENT;

END trg_check_membership_lvl_based_credit;
/
```

Scenario 1: Decrease the member credit for member MB0015 beyond the threshold credit of membership level 1

| | |
|---|---|
| Input | `SQL> UPDATE MEMBER SET CREDIT = 40 WHERE MEMBER_ID = 'MB0015';`<br><br>`1 row updated.` |

| | |
|---|---|
| Output | Before updating the credit of member MB0015:<br>SQL> SELECT MS.MEMBERSHIP_ID, MS.MEMBERSHIP_LEVEL, MB.CREDIT, MB.NAME FROM MEMBER MB JC<br><br>MEMBER MEMBERSHIP_LEVEL CREDIT NAME<br>------ ---------- ---------- --------------------------------------------------<br>MS0002 1 100 Jack Wong<br><br>After updating the credit of member MB0015 from 100 to 40, the membership level will be automatically downgraded based on the credit threshold:<br>SQL> UPDATE MEMBER SET CREDIT = 40 WHERE MEMBER_ID = 'MB0015';<br><br>1 row updated.<br><br>SQL> SELECT MS.MEMBERSHIP_ID, MS.MEMBERSHIP_LEVEL, MB.CREDIT, MB.NAME FROM MEMBER MB JC<br><br>MEMBER MEMBERSHIP_LEVEL CREDIT NAME<br>------ ---------- ---------- --------------------------------------------------<br>MS0001 0 40 Jack Wong |

Scenario 2: Increase the member credit for member MB0005 from 40 to 90.

| | |
|---|---|
| Input | SQL> UPDATE MEMBER SET CREDIT = 90 WHERE MEMBER_ID = 'MB0015';<br><br>1 row updated. |
| Output | Before updating the credit of member MB0015:<br>SQL> SELECT MS.MEMBERSHIP_ID, MS.MEMBERSHIP_LEVEL, MB.CREDIT, MB.NAME FROM MEMBER MB J<br>S ON MB.MEMBERSHIP_ID = MS.MEMBERSHIP_ID WHERE MEMBER_ID = 'MB0015';<br><br>MEMBER MEMBERSHIP_LEVEL CREDIT NAME<br>------ ---------- ---------- --------------------------------------------------<br>MS0001 0 40 Jack Wong<br><br>After updating the credit of member MB0015 from 40 to 90, the membership level will be automatically upgraded based on the latest member credit and credit threshold for membership level:<br>SQL> UPDATE MEMBER SET CREDIT = 90 WHERE MEMBER_ID = 'MB0015';<br><br>1 row updated.<br><br>SQL> SELECT MS.MEMBERSHIP_ID, MS.MEMBERSHIP_LEVEL, MB.CREDIT, MB.NAME FROM MEMBER MB J<br>S ON MB.MEMBERSHIP_ID = MS.MEMBERSHIP_ID WHERE MEMBER_ID = 'MB0015';<br><br>MEMBER MEMBERSHIP_LEVEL CREDIT NAME<br>------ ---------- ---------- --------------------------------------------------<br>MS0002 1 90 Jack Wong |

### 4.2.7 Report 1: Monthly Top 10 Member Borrowing Summary Report

**Purpose: The purpose of this report is to identify the top 10 members who borrowed the most books in a specified month. It also provides a detailed breakdown of their top 5 most borrowed book titles, helping the library understand borrowing trends and active member engagement.**

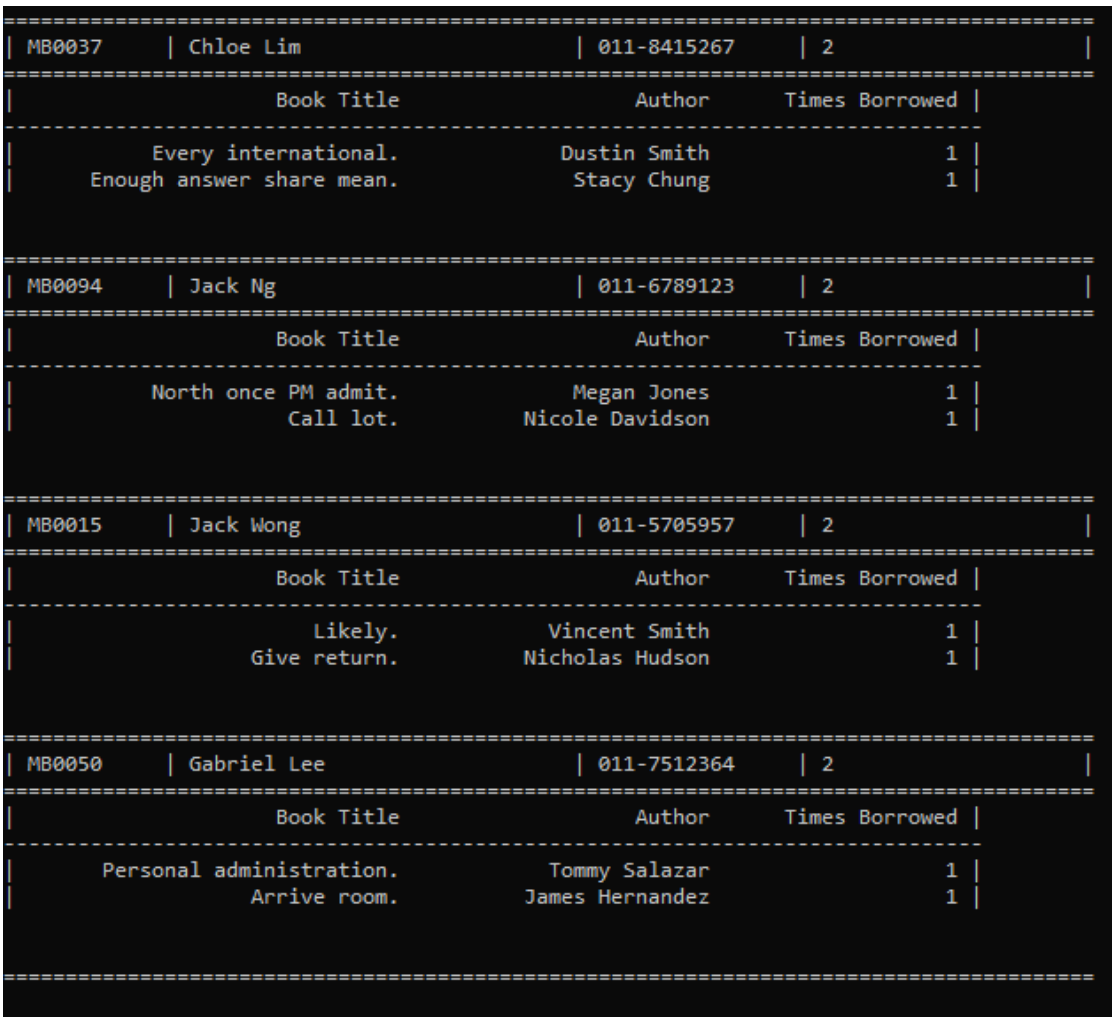
**Report code:**

```
CREATE OR REPLACE PROCEDURE report_member_borrow_summary (
    p_month IN VARCHAR2 -- Format: 'YYYY-MM'
) AS
    -- Outer cursor: Top 10 members who borrowed the most books in the
given month
```

```
    CURSOR member_cursor IS
        SELECT
            m.member_id,
            m.name AS member_name,
            m.tel_no,
            COUNT(*) AS total_borrowed
        FROM borrow_detail bd
        JOIN borrow br ON bd.borrow_id = br.borrow_id
        JOIN member m ON br.member_id = m.member_id
        WHERE TO_CHAR(br.borrow_datetime, 'YYYY-MM') = p_month
        GROUP BY m.member_id, m.name, m.tel_no
        ORDER BY total_borrowed DESC
        FETCH FIRST 10 ROWS ONLY;

    -- Nested cursor: Top 5 books borrowed by a member in that month
    CURSOR book_cursor(p_member_id member.member_id%TYPE) IS
        SELECT
            b.name AS book_title,
            b.author,
            COUNT(*) AS times_borrowed
        FROM borrow_detail bd
        JOIN borrow br ON bd.borrow_id = br.borrow_id
        JOIN book b ON bd.book_id = b.book_id
        WHERE br.member_id = p_member_id
          AND TO_CHAR(br.borrow_datetime, 'YYYY-MM') = p_month
        GROUP BY b.name, b.author
        ORDER BY times_borrowed DESC
        FETCH FIRST 5 ROWS ONLY;

BEGIN
    DBMS_OUTPUT.PUT_LINE(CHR(10));
    DBMS_OUTPUT.PUT_LINE('Top 10 Member Borrowing Summary for ' ||
p_month);
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 88, '='));
    DBMS_OUTPUT.PUT_LINE('| ' || RPAD('Member ID', 10) || ' | ' ||
RPAD('Member Name', 30) || ' | ' || RPAD('Tel No', 15) || ' | ' ||
RPAD('Total Borrowed', 20) || ' |');
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 88, '='));

    FOR mem_rec IN member_cursor LOOP
        DBMS_OUTPUT.PUT_LINE('| ' || RPAD(mem_rec.member_id, 10) || '
| ' || RPAD(mem_rec.member_name, 30) || ' | ' || RPAD(mem_rec.tel_no,
15) || ' | ' || RPAD(mem_rec.total_borrowed, 20) || ' |');
        DBMS_OUTPUT.PUT_LINE(RPAD('=', 88, '='));
        DBMS_OUTPUT.PUT_LINE('| ' || LPAD('Book Title', 30) ||
LPAD('Author', 25) || LPAD('Times Borrowed', 20) || ' |');
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 79, '-'));

        FOR book_rec IN book_cursor(mem_rec.member_id) LOOP
            DBMS_OUTPUT.PUT_LINE('| ' ||
LPAD(SUBSTR(book_rec.book_title, 1, 30), 30) ||
                                 LPAD(SUBSTR(book_rec.author, 1,
25), 25) ||
                                 LPAD(book_rec.times_borrowed, 20)
|| ' |');
        END LOOP;
        DBMS_OUTPUT.PUT_LINE(CHR(10));
        DBMS_OUTPUT.PUT_LINE(RPAD('=', 88, '='));
    END LOOP;
END;
/
```

**Sample Output:**

```
Top 10 Member Borrowing Summary for 2025-03
================================================================================
| Member ID  | Member Name                | Tel No          | Total Borrowed       |
================================================================================
| MB0021     | Ethan Lim                  | 011-7845123     | 3                    |
================================================================================
|                    Book Title                   Author      Times Borrowed |
--------------------------------------------------------------------------------
|               Citizen no your.          Mary Thompson                   1 |
|                  Quickly yard.         Adrian Barnett                   1 |
|                   Role camera.           Deborah Wood                   1 |


================================================================================
| MB0013     | Isabelle Tan               | 011-2845766     | 3                    |
================================================================================
|                    Book Title                   Author      Times Borrowed |
--------------------------------------------------------------------------------
|                  Scene anyone.         Ryan Dominguez                   1 |
|                Represent right.           Travis Craig                   1 |
|                   Role camera.           Deborah Wood                   1 |


================================================================================
| MB0057     | Noah Chua                  | 011-9632517     | 3                    |
================================================================================
|                    Book Title                   Author      Times Borrowed |
--------------------------------------------------------------------------------
|         Return big course top.        Kenneth Stewart                   1 |
|            Within staff world.        Jennifer Harper                   1 |
|                     Into help.           Anna Sanders                   1 |


================================================================================
| MB0041     | Nathan Lee                 | 011-7458963     | 2                    |
================================================================================
|                    Book Title                   Author      Times Borrowed |
--------------------------------------------------------------------------------
|                Model indicate.          Tracey Frazier                   1 |
|     Cut toward through itself.            Jordan Hogan                   1 |


================================================================================
| MB0064     | Max Wong                   | 011-9654783     | 2                    |
================================================================================
|                    Book Title                   Author      Times Borrowed |
--------------------------------------------------------------------------------
|                  Answer much.         Sarah Anderson                   1 |
|              Fact gas suddenly.         Patricia Smith                   1 |


================================================================================
| MB0061     | Ryan Goh                   | 011-8654392     | 2                    |
================================================================================
|                    Book Title                   Author      Times Borrowed |
--------------------------------------------------------------------------------
|               Mind term happen.         April Sheppard                   1 |
|                       Rich box.           Robert Davis                   1 |
```



| MB0037 | Chloe Lim | 011-8415267 | 2 |
|---|---|---|---|

| Book Title | Author | Times Borrowed |
|---|---|---|
| Every international. | Dustin Smith | 1 |
| Enough answer share mean. | Stacy Chung | 1 |

| MB0094 | Jack Ng | 011-6789123 | 2 |
|---|---|---|---|

| Book Title | Author | Times Borrowed |
|---|---|---|
| North once PM admit. | Megan Jones | 1 |
| Call lot. | Nicole Davidson | 1 |

| MB0015 | Jack Wong | 011-5705957 | 2 |
|---|---|---|---|

| Book Title | Author | Times Borrowed |
|---|---|---|
| Likely. | Vincent Smith | 1 |
| Give return. | Nicholas Hudson | 1 |

| MB0050 | Gabriel Lee | 011-7512364 | 2 |
|---|---|---|---|

| Book Title | Author | Times Borrowed |
|---|---|---|
| Personal administration. | Tommy Salazar | 1 |
| Arrive room. | James Hernandez | 1 |

### 4.2.8 Report 2: Yearly Membership-Level Borrowing Summary Report

**Purpose: The purpose of this report is to analyze yearly borrowing trends categorized by membership levels. It highlights the most borrowed genre and the top 5 most borrowed books for each level, helping to understand member preferences and genre popularity across different tiers.**

**Report code:**

```
CREATE OR REPLACE PROCEDURE
report_membership_borrow_summary_yearly(p_year IN VARCHAR2) IS
    -- Cursor for each membership level
    CURSOR level_cursor IS
        SELECT DISTINCT membership_level, membership_id
        FROM Membership;

    -- Cursor for most borrowed genre by membership level and year
    CURSOR genre_cursor(p_level Membership.membership_level%TYPE,
p_year VARCHAR2) IS
        SELECT b.genre, COUNT(*) AS total_borrowed
        FROM Borrow_Detail bd
        JOIN Borrow br ON br.borrow_id = bd.borrow_id
        JOIN Member m ON m.member_id = br.member_id
        JOIN Membership ms ON ms.membership_id = m.membership_id
        JOIN Book b ON b.book_id = bd.book_id
        WHERE ms.membership_level = p_level
          AND TO_CHAR(br.borrow_datetime, 'YYYY') = p_year
```

```
        GROUP BY b.genre
        ORDER BY total_borrowed DESC
        FETCH FIRST 1 ROWS ONLY;

    -- Cursor for top 5 most borrowed books by membership level and
year
    CURSOR book_cursor(p_level Membership.membership_level%TYPE,
p_year VARCHAR2) IS
        SELECT b.name, COUNT(*) AS times_borrowed
        FROM Borrow_Detail bd
        JOIN Borrow br ON br.borrow_id = bd.borrow_id
        JOIN Member m ON m.member_id = br.member_id
        JOIN Membership ms ON ms.membership_id = m.membership_id
        JOIN Book b ON b.book_id = bd.book_id
        WHERE ms.membership_level = p_level
          AND TO_CHAR(br.borrow_datetime, 'YYYY') = p_year
        GROUP BY b.name
        ORDER BY times_borrowed DESC
        FETCH FIRST 5 ROWS ONLY;

    -- Variable to hold count of borrow records for each level
    v_count NUMBER;

BEGIN
    DBMS_OUTPUT.PUT_LINE(CHR(10));
    DBMS_OUTPUT.PUT_LINE('Yearly Borrowing Summary by Membership Level
(' || p_year || ')');
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 87, '='));
    DBMS_OUTPUT.PUT_LINE('| ' || RPAD('Membership ID', 15) || ' | ' ||
                         RPAD('Membership Level', 20) || ' | ' ||
                         RPAD('Most Borrowed Genre', 30) || '| ' ||
                         RPAD('Times', 10) || ' |');
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 87, '='));

    FOR level_rec IN level_cursor LOOP
        -- Check if this level has any borrow record for the year
        SELECT COUNT(*)
        INTO v_count
        FROM Borrow_Detail bd
        JOIN Borrow br ON br.borrow_id = bd.borrow_id
        JOIN Member m ON m.member_id = br.member_id
        JOIN Membership ms ON ms.membership_id = m.membership_id
        WHERE ms.membership_level = level_rec.membership_level
          AND TO_CHAR(br.borrow_datetime, 'YYYY') = p_year;

        IF v_count > 0 THEN
            DBMS_OUTPUT.PUT('| ' || RPAD(level_rec.membership_id, 15)
|| ' | ' ||
                                RPAD(level_rec.membership_level, 20)
|| ' | ');

            -- Most borrowed genre
            FOR genre_rec IN genre_cursor(level_rec.membership_level,
p_year) LOOP
                DBMS_OUTPUT.PUT(RPAD(genre_rec.genre, 29) || ' | ' ||
                                RPAD(genre_rec.total_borrowed, 10) ||
' |');
                DBMS_OUTPUT.PUT_LINE('');
            END LOOP;

            DBMS_OUTPUT.PUT_LINE(RPAD('=', 87, '='));

            -- Top 5 borrowed books
            DBMS_OUTPUT.PUT_LINE(RPAD('| Top 5 Most Borrowed Books:',
52) || ' |');
            DBMS_OUTPUT.PUT_LINE('| ' || LPAD('Title', 30) ||
LPAD('Times Borrowed', 20) || ' |');
            DBMS_OUTPUT.PUT_LINE(RPAD('-', 54, '-'));
```

```
            FOR book_rec IN book_cursor(level_rec.membership_level,
p_year) LOOP
                DBMS_OUTPUT.PUT_LINE('| ' || LPAD(book_rec.name, 30)
|| LPAD(book_rec.times_borrowed, 20) || ' |');
            END LOOP;
            DBMS_OUTPUT.PUT_LINE(CHR(13));
            DBMS_OUTPUT.PUT_LINE(RPAD('=', 87, '='));
        END IF;

    END LOOP;
END;
/
```

**Sample Output:**

Yearly Borrowing Summary by Membership Level (2025)

| Membership ID | Membership Level | Most Borrowed Genre | Times |
|---|---|---|---|
| MS0002 | 1 | non-fiction | 8 |

Top 5 Most Borrowed Books:

| Title | Times Borrowed |
|---|---|
| Organization election store. | 2 |
| Likely. | 2 |
| Agreement finish scene. | 2 |
| Recently material social. | 1 |
| When interesting inside. | 1 |

| Membership ID | Membership Level | Most Borrowed Genre | Times |
|---|---|---|---|
| MS0003 | 2 | non-fiction | 11 |

Top 5 Most Borrowed Books:

| Title | Times Borrowed |
|---|---|
| Quickly yard. | 2 |
| Work you. | 2 |
| Rate safe account. | 2 |
| Return of. | 1 |
| Benefit fast. | 1 |

| Membership ID | Membership Level | Most Borrowed Genre | Times |
|---|---|---|---|
| MS0004 | 3 | fiction | 10 |

Top 5 Most Borrowed Books:

| Title | Times Borrowed |
|---|---|
| Car forget. | 2 |
| Soon woman draw. | 2 |
| Answer much. | 1 |
| People deal or mean. | 1 |
| Role network even. | 1 |

| Membership ID | Membership Level | Most Borrowed Genre | Times |
|---|---|---|---|
| MS0005 | 4 | academic | 5 |

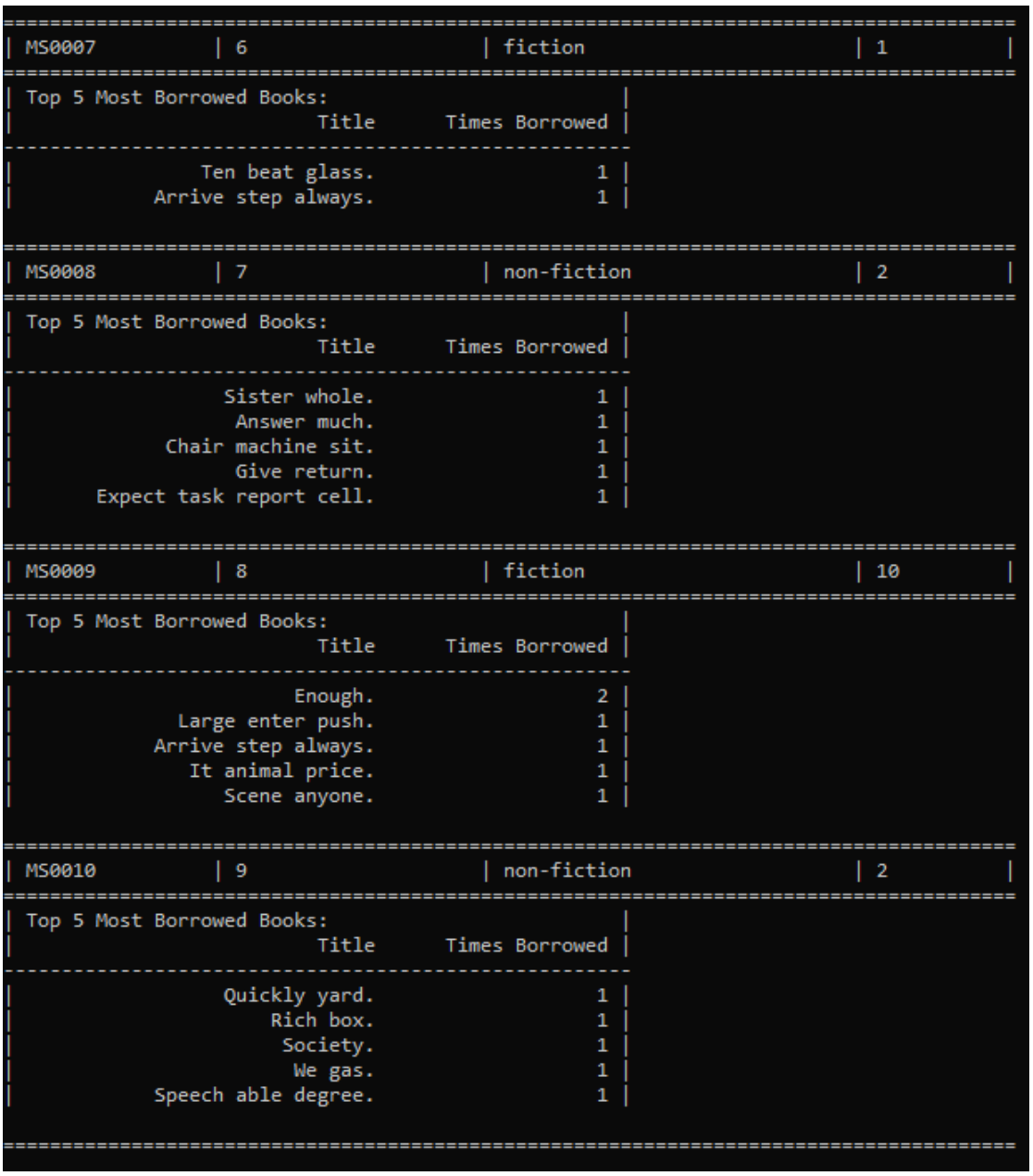
Top 5 Most Borrowed Books:

| Title | Times Borrowed |
|---|---|
| Return big course top. | 1 |
| Fact gas suddenly. | 1 |
| Though anyone. | 1 |
| Into help. | 1 |
| Organization election store. | 1 |

| Membership ID | Membership Level | Most Borrowed Genre | Times |
|---|---|---|---|
| MS0006 | 5 | non-fiction | 6 |

Top 5 Most Borrowed Books:

| Title | Times Borrowed |
|---|---|
| Minute reach. | 2 |
| Door. | 1 |
| Rich box. | 1 |
| The. | 1 |
| Answer much. | 1 |



```
| MS0007          | 6                    | fiction                    | 1          |

| Top 5 Most Borrowed Books:                                |
|                          Title      Times Borrowed |
|-----------------------------------------------------|
|                Ten beat glass.                  1 |
|             Arrive step always.                 1 |

| MS0008          | 7                    | non-fiction                | 2          |

| Top 5 Most Borrowed Books:                                |
|                          Title      Times Borrowed |
|-----------------------------------------------------|
|                  Sister whole.                  1 |
|                   Answer much.                  1 |
|              Chair machine sit.                 1 |
|                   Give return.                  1 |
|        Expect task report cell.                 1 |

| MS0009          | 8                    | fiction                    | 10         |

| Top 5 Most Borrowed Books:                                |
|                          Title      Times Borrowed |
|-----------------------------------------------------|
|                         Enough.                 2 |
|               Large enter push.                 1 |
|             Arrive step always.                 1 |
|                It animal price.                 1 |
|                   Scene anyone.                 1 |

| MS0010          | 9                    | non-fiction                | 2          |

| Top 5 Most Borrowed Books:                                |
|                          Title      Times Borrowed |
|-----------------------------------------------------|
|                 Quickly yard.                   1 |
|                     Rich box.                   1 |
|                      Society.                   1 |
|                       We gas.                   1 |
|             Speech able degree.                 1 |
```

## 4.3 Ong Yi Xin

### 4.3.1 Query 1: Daily Borrowing Trends

**Purpose: The purpose of this query is to monitor how many books are borrowed per day, helping with staff planning and library activity analysis.**

**SQL statement:**

```
SELECT
    TO_CHAR(b.borrow_datetime, 'YYYY-MM-DD') AS borrow_date,
    COUNT(bd.borrow_detail_id) AS total_books_borrowed
FROM
    borrow b
JOIN
    borrow_detail bd ON b.borrow_id = bd.borrow_id
GROUP BY
    TO_CHAR(b.borrow_datetime, 'YYYY-MM-DD')
ORDER BY
    borrow_date DESC;
```

**Sample Output:**

```
BORROW_DAT TOTAL_BOOKS_BORROWED
---------- --------------------
2025-01-13                    3
2025-01-12                    3
2025-01-10                    5
2025-01-08                    1
2025-01-07                    1
2025-01-06                    3
2025-01-03                    1
2025-01-02                    2
2025-01-01                    4
2024-12-23                    3
2024-12-21                    3

BORROW_DAT TOTAL_BOOKS_BORROWED
---------- --------------------
2024-12-19                    1
2024-12-17                    1
2024-12-16                    2
2024-12-15                    1
2024-12-10                    4
2024-12-09                    1
2024-12-08                    2
2024-12-04                    1
2024-12-03                    8
2024-12-01                    4
```

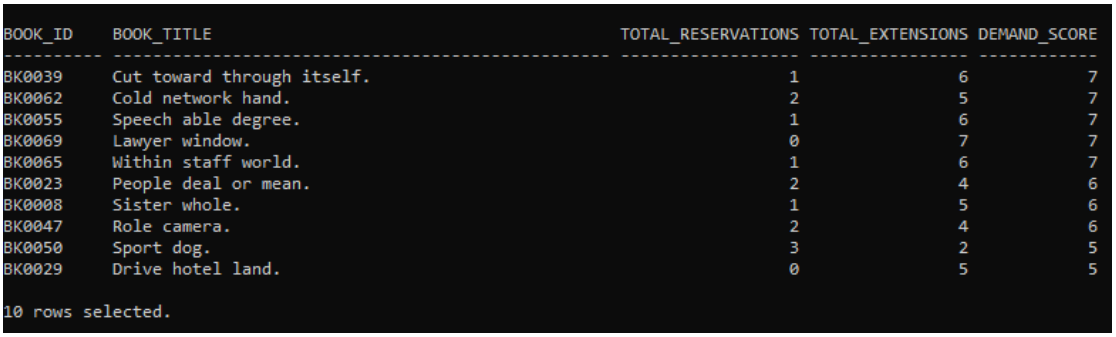
### 4.3.2 Query 2: High-Demand Books Based on Reservations and Loan Extensions

**Purpose: The purpose of this query is to identify books that are consistently in high demand, either because they are frequently reserved or have many loan extension requests. This can help the management decide whether to increase the number of copies of certain titles or review loan extension policies for high-demand books..**

**SQL statement:**

```
COLUMN book_id HEADING 'BOOK_ID' FORMAT A10
SELECT
    b.book_id,
    b.name AS book_title,
    COUNT(DISTINCT r.reservation_id) AS total_reservations,
    COUNT(DISTINCT le.extension_id) AS total_extensions,
    COUNT(DISTINCT r.reservation_id) + COUNT(DISTINCT le.extension_id)
AS demand_score
FROM
    Book b
LEFT JOIN
    Reservation r ON b.book_id = r.book_id
LEFT JOIN
    Borrow_Detail bd ON b.book_id = bd.book_id
LEFT JOIN
    Borrow br ON bd.borrow_id = br.borrow_id
LEFT JOIN
    Loan_Extension le ON br.borrow_id = le.borrow_id
GROUP BY
    b.book_id, b.name
HAVING
    COUNT(DISTINCT r.reservation_id) + COUNT(DISTINCT le.extension_id)
> 0
ORDER BY
    demand_score DESC
FETCH FIRST 10 ROWS ONLY;
```

**Sample Output:**



| BOOK_ID | BOOK_TITLE | TOTAL_RESERVATIONS | TOTAL_EXTENSIONS | DEMAND_SCORE |
|---|---|---|---|---|
| BK0039 | Cut toward through itself. | 1 | 6 | 7 |
| BK0062 | Cold network hand. | 2 | 5 | 7 |
| BK0055 | Speech able degree. | 1 | 6 | 7 |
| BK0069 | Lawyer window. | 0 | 7 | 7 |
| BK0065 | Within staff world. | 1 | 6 | 7 |
| BK0023 | People deal or mean. | 2 | 4 | 6 |
| BK0008 | Sister whole. | 1 | 5 | 6 |
| BK0047 | Role camera. | 2 | 4 | 6 |
| BK0050 | Sport dog. | 3 | 2 | 5 |
| BK0029 | Drive hotel land. | 0 | 5 | 5 |

10 rows selected.

### 4.2.3 Procedure 1: Process Fine Payment and Generate Invoice

**Purpose: The purpose of this procedure is to handle the payment process for a member's fine by verifying eligibility, ensuring sufficient payment, updating the fine status, and generating an official invoice record. It also prints a detailed invoice receipt including member and book details for confirmation.**

**Procedure code:**

```
CREATE OR REPLACE PROCEDURE prc_pay_fine_record (
    p_member_id IN Member.member_id%TYPE,
    p_fine_id   IN Fine_Record.fine_id%TYPE,
    p_amount_paid  IN  NUMBER
) AS
    v_formatted_member_id   Member.member_id%TYPE;
```

```
    v_formatted_fine_id     Fine_Record.fine_id%TYPE;
    v_borrow_detail_id      Fine_Record.borrow_detail_id%TYPE;
    v_formatted_invoice_id  Invoice.invoice_id%TYPE;
    v_fine_amount           Fine_Record.fine_amt%TYPE;
    v_inv_datetime          Invoice.inv_datetime%TYPE;
    v_change                NUMBER(7, 2) := 0;
    v_eligible              NUMBER;
BEGIN
    -- Format member_id and fine_id
    v_formatted_member_id := 'MB' || LPAD(p_member_id, 4, '0');
    v_formatted_fine_id := 'FN' || LPAD(p_fine_id, 4, '0');

    -- Check eligibility using UDF
    v_eligible := fn_is_fine_eligible(v_formatted_member_id,
v_formatted_fine_id);

    IF v_eligible = 0 THEN
        DBMS_OUTPUT.PUT_LINE('Error: The fine failed to pay.');
        RETURN;
    END IF;

    -- Check whether the amount given is sufficient to pay the fine
    SELECT borrow_detail_id, fine_amt
    INTO v_borrow_detail_id, v_fine_amount
    FROM Fine_Record
    WHERE fine_id = v_formatted_fine_id;

    IF p_amount_paid < v_fine_amount THEN
        DBMS_OUTPUT.PUT_LINE('Error: The amount given is insufficient
to pay the fine.');
        RETURN;
    ELSIF p_amount_paid > v_fine_amount THEN
        v_change := p_amount_paid - v_fine_amount;
    END IF;

    -- Create invoice and update fine record
    BEGIN
        SAVEPOINT before_insert;
        INSERT INTO Invoice (borrow_detail_id, total_amt,
amt_received, changes, inv_datetime)
        VALUES (v_borrow_detail_id, v_fine_amount, p_amount_paid,
v_change, SYSDATE)
        RETURNING invoice_id, inv_datetime INTO
v_formatted_invoice_id, v_inv_datetime;

        UPDATE Fine_Record
        SET fine_status = 'Paid'
        WHERE fine_id = v_formatted_fine_id;
    EXCEPTION
        WHEN OTHERS THEN
            ROLLBACK TO before_insert;
            DBMS_OUTPUT.PUT_LINE('Fine is failed to pay: ' ||
SQLERRM);
            RETURN;
    END;

    -- Successful message
    DECLARE
        msg_book_id     Book.book_id%TYPE;
        msg_book_name   Book.name%TYPE;
        msg_member_name Member.name%TYPE;
    BEGIN
        SELECT b.book_id, b.name, m.name
        INTO msg_book_id, msg_book_name, msg_member_name
        FROM Book b
        JOIN Borrow_Detail bd ON b.book_id = bd.book_id
        JOIN Fine_Record fr ON bd.borrow_detail_id =
fr.borrow_detail_id
```

```
        JOIN Borrow br ON bd.borrow_id = br.borrow_id
        JOIN Member m ON br.member_id = m.member_id
        WHERE fr.fine_id = v_formatted_fine_id;

        DBMS_OUTPUT.PUT_LINE(CHR(13));
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 10, '-') || RPAD('-', 70, '-')
|| RPAD('-', 20, '-'));
        DBMS_OUTPUT.PUT_LINE('FINE INVOICE');
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 10, '-') || RPAD('-', 70, '-')
|| RPAD('-', 20, '-'));
        DBMS_OUTPUT.PUT_LINE('Invoice ID   : ' ||
v_formatted_invoice_id);
        DBMS_OUTPUT.PUT_LINE('Date and Time  : ' || v_inv_datetime);
        DBMS_OUTPUT.PUT_LINE('Member ID    : ' ||
v_formatted_member_id);
        DBMS_OUTPUT.PUT_LINE('Member Name  : ' || msg_member_name);
        DBMS_OUTPUT.PUT_LINE(CHR(13));

        DBMS_OUTPUT.PUT_LINE(RPAD('No.', 10) || RPAD('Item', 70) ||
LPAD('Price (RM)', 20));
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 10, '-') || RPAD('-', 70, '-')
|| RPAD('-', 20, '-'));
        DBMS_OUTPUT.PUT_LINE(RPAD('1.', 10) || RPAD('Late returning
book : ' || msg_book_id || ' - ' || msg_book_name, 70) ||
LPAD(TO_CHAR(ABS(v_fine_amount), 'FM9999990.00'), 20));
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 10, '-') || RPAD('-', 70, '-')
|| RPAD('-', 20, '-'));

        DBMS_OUTPUT.PUT_LINE(LPAD('Total Amount    : RM ', 80) ||
LPAD(TO_CHAR(v_fine_amount, 'FM9999990.00'), 20));
        DBMS_OUTPUT.PUT_LINE(LPAD('Amount Received : RM ', 80) ||
LPAD(TO_CHAR(p_amount_paid, 'FM9999990.00'), 20));
        DBMS_OUTPUT.PUT_LINE(LPAD('Changes         : RM ', 80) ||
LPAD(TO_CHAR(v_change, 'FM9999990.00'), 20));
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 10, '-') || RPAD('-', 70, '-')
|| RPAD('-', 20, '-'));
        DBMS_OUTPUT.PUT_LINE(CHR(13));

        DBMS_OUTPUT.PUT_LINE('Fine is paid successfully!');
    END;
END;
/
```

**Sample Output:**

Find the fine record of member MB0070, who want to pay their fine:

```
SEARCHING FINE RECORDS FOR MEMBER MB0070...

Fine ID   Book ID   Book Name                 Fine Amount (RM)   Credit Deducted    Fine Status
--------------------------------------------------------------------------------------------------
FN0001    BK0075    Personal administration.  12.00              -16               Unpaid
FN0014    BK0075    Personal administration.  9.00               -12               Unpaid
FN0121    BK0075    Personal administration.  12.00              -16               Paid
--------------------------------------------------------------------------------------------------
                                                                 Total Fine(s):                 3

PL/SQL procedure successfully completed.
```

Run procedure to pay fine and when amount pay not enough:

```
SQL> @"D:\Study\RSD\Y2S3\Advanced Database\Assignment\OngYiXin\user_script\pay_fine_user_script.txt"
Error: The amount given is insufficient to pay the fine.

PL/SQL procedure successfully completed.
```

When amount have more and need to return change:

```
SQL> @"D:\Study\RSD\Y2S3\Advanced Database\Assignment\OngYiXin\user_script\pay_fine_user_script.txt"
Error: The amount given is insufficient to pay the fine.

PL/SQL procedure successfully completed.

SQL> @"D:\Study\RSD\Y2S3\Advanced Database\Assignment\OngYiXin\user_script\pay_fine_user_script.txt"

-----------------------------------------------------------------------------------------------------
FINE INVOICE
-----------------------------------------------------------------------------------------------------
Invoice ID   : IV0122
Date & Time  : 04-MAY-25
Member ID    : MB0070
Member Name  : Henry Lim

No.       Item                                                                          Price (RM)
-----------------------------------------------------------------------------------------------------
1.        Late returning book : BK0075 - Personal administration.                            9.00
-----------------------------------------------------------------------------------------------------
                                                    Total Amount    : RM                   9.00
                                                    Amount Received : RM                  10.00
                                                    Changes         : RM                   1.00
-----------------------------------------------------------------------------------------------------

Fine is paid successfully!

PL/SQL procedure successfully completed.
```

### 4.2.4 Procedure 2: Process Member Book Returns

**Purpose: The purpose of this procedure is to handle the return of books by a library member. It validates the return request, updates the borrowing records based on book conditions, checks for late returns, and displays a summary message for the member.**

**Procedure code:**

```
CREATE OR REPLACE PROCEDURE prc_return_books (
    p_member_id IN Member.member_id%TYPE,
    p_return_books IN return_book_list_t
) AS
    v_formatted_member_id   VARCHAR2(6);
    v_eligible_returning    NUMBER;
    v_book_ids              book_id_list_t;
    v_borrow_detail_ids     borrow_detail_id_list_t;
BEGIN
    -- format member_id
    v_formatted_member_id := 'MB' || LPAD(p_member_id, 4, '0');

    -- initialize v_book_ids type
    v_book_ids := book_id_list_t();
    v_book_ids.EXTEND(p_return_books.COUNT);

    -- initialize v_borrow_detail_ids type
    v_borrow_detail_ids := borrow_detail_id_list_t();
    v_borrow_detail_ids.EXTEND(p_return_books.COUNT);

    FOR i IN 1 .. p_return_books.COUNT LOOP
        BEGIN
            IF p_return_books(i).book_status != 'Good' AND
p_return_books(i).book_status != 'Damaged' THEN
                DBMS_OUTPUT.PUT_LINE('Invalid return book list
passed');
                RETURN;
            END IF;

            v_book_ids(i) := p_return_books(i).book_id;
        END;
    END LOOP;

    -- check eligibility of member and book for returning book
    prc_check_user_return (
        p_member_id =>  v_formatted_member_id,
```

```
        p_book_ids  =>  v_book_ids,
        p_result    =>  v_eligible_returning
    );

    IF v_eligible_returning = 0 THEN
        DBMS_OUTPUT.PUT_LINE('Error: Sorry. You are not eligible to
return the books.');
        RETURN;
    END IF;

    -- update borrow detail
    SAVEPOINT before_insert;
    FOR i IN 1 .. p_return_books.COUNT LOOP
        DECLARE
            v_formatted_book_id Book.book_id%TYPE;
            v_borrow_detail_id  Borrow_Detail.borrow_detail_id%TYPE;
        BEGIN
            BEGIN
                -- format book_id
                v_formatted_book_id := 'BK' ||
LPAD(p_return_books(i).book_id, 4, '0');

                -- obtain borrow_detail_id
                SELECT bd.borrow_detail_id
                INTO v_borrow_detail_id
                FROM Borrow_Detail bd
                JOIN Borrow b
                    ON bd.borrow_id = b.borrow_id
                WHERE b.member_id = v_formatted_member_id
                    AND bd.book_id = v_formatted_book_id
                    AND (bd.borrow_status = 'Borrowed' OR
bd.borrow_status = 'Missing');

                -- update v_borrow_detail_id list
                v_borrow_detail_ids(i) := v_borrow_detail_id;

                -- update Borrow_Detail status
                IF p_return_books(i).book_status = 'Damaged' THEN
                    UPDATE Borrow_Detail
                    SET borrow_status = 'Damaged'
                    , returned_datetime = SYSDATE
                    WHERE borrow_detail_id = v_borrow_detail_id;
                ELSE
                    UPDATE Borrow_Detail
                    SET borrow_status = 'Returned'
                    , returned_datetime = SYSDATE
                    WHERE borrow_detail_id = v_borrow_detail_id;
                END IF;
            EXCEPTION
                WHEN NO_DATA_FOUND THEN
                    DBMS_OUTPUT.PUT_LINE('Error: No borrow detail
found for borrow_detail_id: ' || v_borrow_detail_id);
                    ROLLBACK TO before_insert;
                    RETURN;
            END;
        END;
    END LOOP;
    COMMIT;

    -- successful message
    DECLARE
        v_borrow_id Borrow.borrow_id%TYPE;
        msg_member_name VARCHAR2(100);
        msg_returned_datetime Borrow_Detail.returned_datetime%TYPE;
    BEGIN
        -- obtain returned datetime
        SELECT m.name, bd.returned_datetime
        INTO msg_member_name, msg_returned_datetime
```

```
        FROM Borrow_Detail bd
        JOIN Borrow b
            ON bd.borrow_id = b.borrow_id
        JOIN Member m
            ON b.member_id = m.member_id
        WHERE bd.borrow_detail_id = v_borrow_detail_ids(1);

        DBMS_OUTPUT.PUT_LINE(CHR(13));
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 20, '-') || RPAD('-', 15, '-')
|| RPAD('-', 40, '-') || RPAD('-', 20, '-') || RPAD('-', 15, '-'));
        DBMS_OUTPUT.PUT_LINE('BOOK RETURNING');
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 20, '-') || RPAD('-', 15, '-')
|| RPAD('-', 40, '-') || RPAD('-', 20, '-') || RPAD('-', 15, '-'));
        DBMS_OUTPUT.PUT_LINE('Member ID     : ' ||
v_formatted_member_id);
        DBMS_OUTPUT.PUT_LINE('Member Name   : ' || msg_member_name);
        DBMS_OUTPUT.PUT_LINE('Returned Date : ' ||
TO_CHAR(msg_returned_datetime, 'DD-MON-YYYY HH24:MI:SS'));
        DBMS_OUTPUT.PUT_LINE(CHR(13));

        DBMS_OUTPUT.PUT_LINE(RPAD('Borrow Detail ID', 20) ||
RPAD('Book ID', 15) || RPAD('Book Name', 40) || RPAD('Borrow
Datetime', 25) || 'Status');
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 20, '-') || RPAD('-', 15, '-')
|| RPAD('-', 40, '-') || RPAD('-', 20, '-') || RPAD('-', 15, '-'));

        FOR i IN 1 .. v_borrow_detail_ids.COUNT LOOP
            DECLARE
                -- v_borrow_detail_id
Borrow_Detail.borrow_detail_id%TYPE;
                v_book_id           Book.book_id%TYPE;
                v_book_name         Book.name%TYPE;
                v_borrow_datetime   Borrow.borrow_datetime%TYPE;
                v_return_datetime   Borrow.return_datetime%TYPE;
            BEGIN
                -- obtain borrow_detail_id and borrow_datetime
                SELECT bd.book_id, b.name, br.borrow_datetime,
br.return_datetime
                INTO v_book_id, v_book_name, v_borrow_datetime,
v_return_datetime
                FROM Borrow_Detail bd
                JOIN Borrow br
                    ON bd.borrow_id = br.borrow_id
                JOIN Book b
                    ON bd.book_id = b.book_id
                WHERE bd.borrow_detail_id = v_borrow_detail_ids(i);

                IF SYSDATE > v_return_datetime THEN
                    DBMS_OUTPUT.PUT_LINE(RPAD(v_borrow_detail_ids(i),
20) || RPAD(v_book_id, 15) || RPAD(v_book_name, 40) ||
RPAD(TO_CHAR(v_borrow_datetime, 'DD-MON-YYYY HH24:MI:SS'), 25) ||
'Late');
                ELSE
                    DBMS_OUTPUT.PUT_LINE(RPAD(v_borrow_detail_ids(i),
20) || RPAD(v_book_id, 15) || RPAD(v_book_name, 40) ||
RPAD(TO_CHAR(v_borrow_datetime, 'DD-MON-YYYY HH24:MI:SS'), 25) ||
'Returned');
                END IF;
            EXCEPTION
                WHEN NO_DATA_FOUND THEN
                    DBMS_OUTPUT.PUT_LINE('Error: No borrow detail
found for book ID: ' || v_book_id);
                    ROLLBACK TO before_insert;
                    RETURN;
            END;
        END LOOP;
```

```
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 20, '-') || RPAD('-', 15, '-')
|| RPAD('-', 40, '-') || RPAD('-', 20, '-') || RPAD('-', 15, '-'));
        DBMS_OUTPUT.PUT_LINE(LPAD('Total Book(s): ', 95) ||
LPAD(v_borrow_detail_ids.COUNT, 15));
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 20, '-') || RPAD('-', 15, '-')
|| RPAD('-', 40, '-') || RPAD('-', 20, '-') || RPAD('-', 15, '-'));

        DBMS_OUTPUT.PUT_LINE(CHR(13));
        DBMS_OUTPUT.PUT_LINE('Book(s) are returned successfully!');
        DBMS_OUTPUT.PUT_LINE(CHR(13));
        DBMS_OUTPUT.PUT_LINE('ALERT: Please check the book(s) status
in the system.');
        DBMS_OUTPUT.PUT_LINE('ALERT: ''Late'' status indicates that
the book(s) are returned late. Fine will be charged and credit will be
deducted.');
    END;
END;
/
```

**Sample Output:**

After member done borrow book:

```
SQL> @"D:\Study\RSD\Y2S3\Advanced Database\Assignment\LimJunWei\user_script\borrow_book.txt"

------------------------------------------------------------------------
BOOK BORROWING
------------------------------------------------------------------------
Member ID   : MB0001
Member Name : Adam Tan
Borrow ID   : BR0151
Borrow Date : 03-MAY-2025 21:35:10

Book ID       Book Name                                ISBN
------------------------------------------------------------------------
BK0001        Blood natural pretty worker.             978000000000
BK0002        The.                                     978000000001
------------------------------------------------------------------------
                                          Total Book(s):          2
------------------------------------------------------------------------
```

Return the book with the procedure by using user_script:

```
DECLARE
    v_return_books return_book_list_t := return_book_list_t();
BEGIN
    -- Add book IDs to the collection
    v_return_books.EXTEND(2); -- Adjust the number based on how many
books you're adding
    v_return_books(1) := return_book_t(1, 'Good');
    v_return_books(2) := return_book_t(2, 'Good');
    -- v_book_ids(3) := 3; -- Replace with actual book ID

    -- Call the procedure with member_id and book_id list
    prc_return_books(1, v_return_books);
END;
/
```

```
------------------------------------------------------------------------------------------------------------------------
BOOK RETURNING
------------------------------------------------------------------------------------------------------------------------
Member ID     : MB0001
Member Name   : Adam Tan
Returned Date : 03-MAY-2025 21:35:41

Borrow Detail ID    Book ID       Book Name                          Borrow Datetime         Status
------------------------------------------------------------------------------------------------------------------------
BD0151              BK0001        Blood natural pretty worker.       03-MAY-2025 21:35:10    Returned
BD0152              BK0002        The.                               03-MAY-2025 21:35:10    Returned
------------------------------------------------------------------------------------------------------------------------
                                                                                 Total Book(s):              2
------------------------------------------------------------------------------------------------------------------------

Book(s) are returned successfully!

ALERT: Please check the book(s) status in the system.
ALERT: 'Late' status indicates that the book(s) are returned late. Fine will be charged and credit will be deducted.

PL/SQL procedure successfully completed.
```

### 4.2.6 Trigger 2: Handle Late Book Returns and Apply Fines

**Purpose: The purpose of this trigger is to automatically execute after a book's borrow status is updated to 'Returned'. It checks whether the return is late, calculates applicable fines and credit deductions based on configured metrics, logs the fine, updates the member's credit, and tracks overdue activity in the audit trail.**

**Trigger code:**

```
CREATE OR REPLACE TRIGGER trg_check_late_return
AFTER UPDATE OF borrow_status ON Borrow_Detail
FOR EACH ROW
WHEN (NEW.borrow_status = 'Returned')
DECLARE
    v_return_datetime   Borrow.return_datetime%TYPE;
    v_hours_late            NUMBER;
    v_fine_per_book         NUMBER(7, 2) := 0;
    v_fine_amount           NUMBER := 0;
    v_credit_deduct         NUMBER := 0;
    v_total_fine            NUMBER(7, 2);
    v_total_credit_deduct   NUMBER;
    v_member_id             Member.member_id%TYPE;
    v_current_member_credit Member.credit%TYPE;
    v_remaining_credit      Member.credit%TYPE;
BEGIN
    -- obtain the expected return date for borrowed book
    SELECT return_datetime
    INTO v_return_datetime
    FROM Borrow
    WHERE borrow_id = :NEW.borrow_id;

    -- calculate the number of hours the book is returned late
    v_hours_late := CEIL((:NEW.returned_datetime - v_return_datetime)
* 24);

    -- check if the book is returned late
    IF v_hours_late > 0 THEN
        -- fetch applicable fine metrics
        SELECT fine_per_book, credit_deduct
        INTO v_fine_per_book, v_credit_deduct
        FROM Fine_Metric
        WHERE returned_hours >= v_hours_late
        ORDER BY returned_hours ASC
        FETCH FIRST 1 ROWS ONLY;

        -- fetch current member's credit
        SELECT m.member_id, m.credit
        INTO v_member_id, v_current_member_credit
        FROM Member m
        JOIN Borrow br
            ON m.member_id = br.member_id
        WHERE br.borrow_id = :NEW.borrow_id;

        -- calculate total credit deduct and total fine amount
        v_total_credit_deduct := CEIL(v_hours_late * v_credit_deduct);
        v_total_fine := CEIL(v_hours_late * v_fine_per_book);
        v_remaining_credit := v_current_member_credit -
v_total_credit_deduct;

        -- check whether the credit deduct exceed the threshold 0
        IF v_remaining_credit < 0 THEN
            v_total_credit_deduct := v_total_credit_deduct +
v_remaining_credit;
            v_remaining_credit := 0;
        END IF;
```

```
        -- insert fine_record
        INSERT INTO Fine_Record (borrow_detail_id, credit_deduct,
fine_amt, fine_status)
        VALUES (:NEW.borrow_detail_id, v_total_credit_deduct,
v_total_fine, 'Unpaid');

        -- update overdue book in audit trail
        UPDATE Monthly_Audit
        SET no_of_overdue = no_of_overdue + 1,
            latest_datetime = SYSDATE
        WHERE created_datetime = TRUNC(SYSDATE, 'MM');

        -- update member credit
        UPDATE Member
        SET credit = v_remaining_credit
        WHERE member_id = v_member_id;
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Error: Fine metric record not found.');
        RETURN;
END;
/
```

**Sample Output:**

```
SQL> @"C:\Users\limju\OneDrive - student.tarc.edu.my\RSDY2S3\BACS3183 ADVANCED DATABASE MANAGEMENT\Assignment\BA

--------------------------------------------------------------------------------------------------------------
BOOK RETURNING
--------------------------------------------------------------------------------------------------------------
Member ID     : MB0001
Member Name   : Adam Tan
Returned Date : 04-MAY-2025 01:48:16

Borrow Detail ID    Book ID        Book Name                                  Borrow Datetime          Status
--------------------------------------------------------------------------------------------------------------
BD0151              BK0001         Blood natural pretty worker.               04-MAY-2025 01:46:32     Late
BD0152              BK0002         The.                                       04-MAY-2025 01:46:32     Late
--------------------------------------------------------------------------------------------------------------
                                                                                    Total Book(s):       2
--------------------------------------------------------------------------------------------------------------

Book(s) are returned successfully!
```

```
SQL> SELECT * FROM FINE_RECORD WHERE BORROW_DETAIL_ID IN ('BD0151', 'BD0152');

FINE_I BORROW CREDIT_DEDUCT   FINE_AMT FINE_S
------ ------ ------------- ---------- ------
FN0151 BD0151           100        444 Unpaid
FN0152 BD0152             0        444 Unpaid
```

### 4.2.6 Trigger 2: Update Book Availability and Return Audit

**Purpose: The purpose of this trigger is to run after a book's borrow status is updated to 'Returned'. It marks the book as available for borrowing again and updates the monthly audit to reflect the return activity.**

**Trigger code:**

```
CREATE OR REPLACE TRIGGER trg_update_book_return
AFTER UPDATE OF borrow_status ON Borrow_Detail
FOR EACH ROW
WHEN (NEW.borrow_status = 'Returned')
BEGIN
    UPDATE Book
    SET availability = 'Y'
    WHERE book_id = :NEW.book_id;

    UPDATE Monthly_Audit
    SET no_of_return = no_of_return + 1,
```

```
        latest_datetime = SYSDATE
    WHERE created_datetime = TRUNC(SYSDATE, 'MM');
END;
/
```

Sample Output:

When return book in time:

```
SQL> SELECT BOOK_ID, AVAILABILITY FROM BOOK WHERE BOOK_ID IN ('BK0001', 'BK0002');

BOOK_I A
------ -
BK0001 N
BK0002 N

SQL> @"C:\Users\limju\OneDrive - student.tarc.edu.my\RSDY2S3\BACS3183 ADVANCED DATABASE MANAGEMENT\Assignment\BACS318
NT\OngYiXin\user_script\return_book_user_script.txt"

--------------------------------------------------------------------------------------------------------------------
BOOK RETURNING
--------------------------------------------------------------------------------------------------------------------
Member ID     : MB0001
Member Name   : Adam Tan
Returned Date : 04-MAY-2025 11:39:27

Borrow Detail ID    Book ID        Book Name                                Borrow Datetime          Status
--------------------------------------------------------------------------------------------------------------------
BD0151              BK0001         Blood natural pretty worker.             04-MAY-2025 11:38:19     Returned
BD0152              BK0002         The.                                     04-MAY-2025 11:38:19     Returned
--------------------------------------------------------------------------------------------------------------------
                                                                                    Total Book(s):              2
--------------------------------------------------------------------------------------------------------------------

Book(s) are returned successfully!

ALERT: Please check the book(s) status in the system.
ALERT: 'Late' status indicates that the book(s) are returned late. Fine will be charged and credit will be deducted.

PL/SQL procedure successfully completed.

SQL> SELECT BOOK_ID, AVAILABILITY FROM BOOK WHERE BOOK_ID IN ('BK0001', 'BK0002');

BOOK_I A
------ -
BK0001 Y
BK0002 Y
```

```
SQL> SELECT * FROM MONTHLY_AUDIT;

AUDIT_ NO_OF_BORROW NO_OF_RETURN NO_OF_OVERDUE NO_OF_FINE_PAYMENT NO_OF_LOAN_EXTENSION NO_OF_RESERVATION CREATED_D LATEST_DA
------ ------------ ------------ ------------ ------------------ -------------------- ---------------- --------- ---------
MA0001            2            2            0                  0                    0                0 04-MAY-25 04-MAY-25
```

### 4.3.7 Report 1: Annual Fine Collection and Overdue Book Summary

**Purpose: The purpose of this query is to provide a monthly breakdown of fines collected, unpaid fines, and overdue books for a specified year. It also summarizes total fines and overdue books across the entire year, including detailed borrower-level fine information per month.**

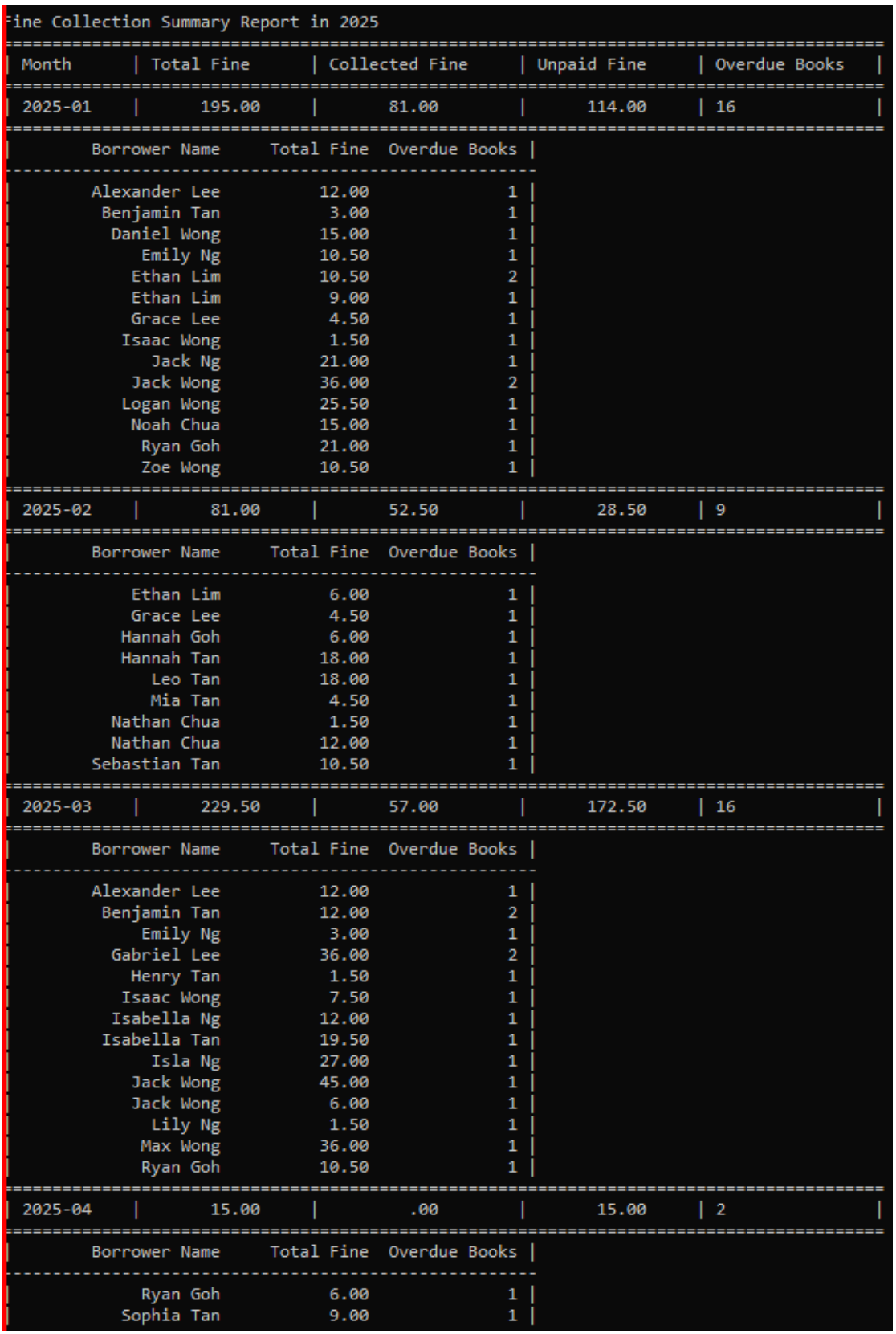
**Report code:**

```
CREATE OR REPLACE PROCEDURE report_fine_collect_status (
    p_year IN NUMBER
) AS
    -- Outer cursor for collecting fine details by month
    CURSOR fine_collect_cursor IS
        SELECT
            TO_CHAR(i.inv_datetime, 'YYYY-MM') AS month,
            SUM(fr.fine_amt) AS total_fine,
            SUM(CASE WHEN fr.fine_status = 'Paid' THEN fr.fine_amt
ELSE 0 END) AS total_collected_fine,
            SUM(CASE WHEN fr.fine_status = 'Unpaid' THEN fr.fine_amt
ELSE 0 END) AS total_unpaid_fine,
            COUNT(DISTINCT CASE
                           WHEN bd.borrow_status = 'Missing' OR
fr.borrow_detail_id IS NOT NULL THEN bd.borrow_detail_id
                           ELSE NULL
                           END) AS overdue_books
        FROM fine_record fr
        JOIN invoice i ON fr.borrow_detail_id = i.borrow_detail_id
```

```
        JOIN borrow_detail bd ON i.borrow_detail_id =
bd.borrow_detail_id
        JOIN borrow b ON bd.borrow_id = b.borrow_id
        WHERE TO_CHAR(i.inv_datetime, 'YYYY') = TO_CHAR(p_year)
        GROUP BY TO_CHAR(i.inv_datetime, 'YYYY-MM')
        ORDER BY TO_CHAR(i.inv_datetime, 'YYYY-MM');

    -- Nested cursor for fine details by borrower within each month
    CURSOR borrower_fine_details_cursor(p_month IN VARCHAR2) IS
        SELECT
            b.member_id,
            m.name AS borrower_name,  -- Corrected to get member name
            SUM(fr.fine_amt) AS total_fine,
            COUNT(DISTINCT CASE
                           WHEN bd.borrow_status = 'Missing' OR
fr.borrow_detail_id IS NOT NULL THEN bd.borrow_detail_id
                           ELSE NULL
                           END) AS overdue_books
        FROM fine_record fr
        JOIN invoice i ON fr.borrow_detail_id = i.borrow_detail_id
        JOIN borrow_detail bd ON i.borrow_detail_id =
bd.borrow_detail_id
        JOIN borrow b ON bd.borrow_id = b.borrow_id
        JOIN member m ON b.member_id = m.member_id -- Join with member
table for borrower name
        WHERE TO_CHAR(i.inv_datetime, 'YYYY-MM') = p_month
        GROUP BY b.member_id, m.name  -- Group by member_id and name
        ORDER BY m.name;

    -- Variables to store the results for the summary
    total_fine_collected NUMBER := 0;
    total_unpaid_fine NUMBER := 0;
    total_fine_for_year NUMBER := 0;
    total_overdue_books NUMBER := 0;

BEGIN
    -- Header of the report
    DBMS_OUTPUT.PUT_LINE(CHR(13));
    DBMS_OUTPUT.PUT_LINE('Fine Collection Summary Report in ' ||
p_year);
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 89, '='));
    DBMS_OUTPUT.PUT_LINE('| ' || RPAD('Month', 10) || ' | ' ||
RPAD('Total Fine', 15) || ' | ' || RPAD('Collected Fine', 18) || ' | '
|| RPAD('Unpaid Fine', 15) || ' | ' || RPAD('Overdue Books', 15) || '
|');

    -- Process each month
    FOR fine_record IN fine_collect_cursor LOOP
        -- Output data for each month with proper alignment
        DBMS_OUTPUT.PUT_LINE(RPAD('=', 89, '='));
        DBMS_OUTPUT.PUT_LINE('| ' || RPAD(fine_record.month, 10) || '
| ' ||
                             RPAD(TO_CHAR(fine_record.total_fine,
'999,999.99'), 15) || ' | ' ||

RPAD(TO_CHAR(fine_record.total_collected_fine, '999,999.99'), 18) || '
| ' ||

RPAD(TO_CHAR(fine_record.total_unpaid_fine, '999,999.99'), 15) || ' |
' ||
                             RPAD(TO_CHAR(fine_record.overdue_books),
15) || ' |');
        DBMS_OUTPUT.PUT_LINE(RPAD('=', 89, '='));
        -- Nested cursor for fine details per borrower in the current
month
        DBMS_OUTPUT.PUT_LINE('| ' || LPAD('Borrower Name', 20) ||
                               LPAD('Total Fine', 15) ||
                               LPAD('Overdue Books', 15) || ' |');
```

```
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 54, '-'));

        -- Display each borrower
        FOR borrower_record IN
borrower_fine_details_cursor(fine_record.month) LOOP
            DBMS_OUTPUT.PUT_LINE('| ' ||
LPAD(borrower_record.borrower_name, 20) ||

LPAD(TO_CHAR(borrower_record.total_fine, '999,990.00'), 15) ||

LPAD(TO_CHAR(borrower_record.overdue_books), 15) || ' |');
        END LOOP;

        -- Summing the total fines, collected fines, unpaid fines, and
overdue books for the year
        total_fine_for_year := total_fine_for_year +
fine_record.total_fine;
        total_fine_collected := total_fine_collected +
fine_record.total_collected_fine;
        total_unpaid_fine := total_unpaid_fine +
fine_record.total_unpaid_fine;
        total_overdue_books := total_overdue_books +
fine_record.overdue_books;
    END LOOP;

    -- Summary of the report
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 89, '='));
    DBMS_OUTPUT.PUT_LINE(RPAD('= Report Summary', 88) || '=');
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 89, '='));
    DBMS_OUTPUT.PUT_LINE('| ' || RPAD('Total Fine Collected', 30) || '
: ' || RPAD(TO_CHAR(total_fine_collected, '999,999.99'), 52) || ' |');
    DBMS_OUTPUT.PUT_LINE('| ' || RPAD('Total Fine Unpaid', 30) || ' :
' || RPAD(TO_CHAR(total_unpaid_fine, '999,999.99'), 52) || ' |');
    DBMS_OUTPUT.PUT_LINE('| ' || RPAD('Total Fine for Year', 30) || '
: ' || RPAD(TO_CHAR(total_fine_for_year, '999,999.99'), 52) || ' |');
    DBMS_OUTPUT.PUT_LINE('| ' || RPAD('Total Overdue Books', 30) || '
: ' || RPAD(TO_CHAR(total_overdue_books, '99,999,999'), 52) || ' |');
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 89, '='));

END;
/
```

**Sample Output:**



Fine Collection Summary Report in 2025

| Month | Total Fine | Collected Fine | Unpaid Fine | Overdue Books |
|---|---|---|---|---|
| 2025-01 | 195.00 | 81.00 | 114.00 | 16 |

| Borrower Name | Total Fine | Overdue Books |
|---|---|---|
| Alexander Lee | 12.00 | 1 |
| Benjamin Tan | 3.00 | 1 |
| Daniel Wong | 15.00 | 1 |
| Emily Ng | 10.50 | 1 |
| Ethan Lim | 10.50 | 2 |
| Ethan Lim | 9.00 | 1 |
| Grace Lee | 4.50 | 1 |
| Isaac Wong | 1.50 | 1 |
| Jack Ng | 21.00 | 1 |
| Jack Wong | 36.00 | 2 |
| Logan Wong | 25.50 | 1 |
| Noah Chua | 15.00 | 1 |
| Ryan Goh | 21.00 | 1 |
| Zoe Wong | 10.50 | 1 |

| Month | Total Fine | Collected Fine | Unpaid Fine | Overdue Books |
|---|---|---|---|---|
| 2025-02 | 81.00 | 52.50 | 28.50 | 9 |

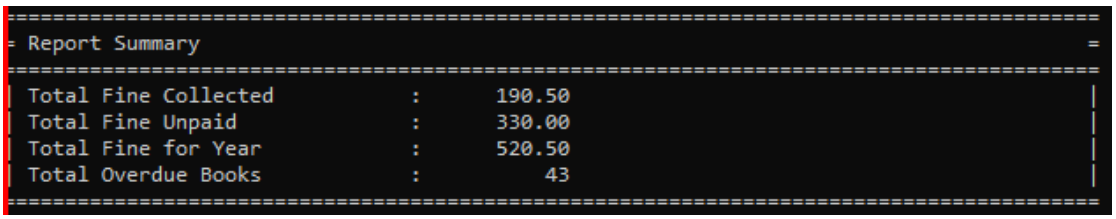
| Borrower Name | Total Fine | Overdue Books |
|---|---|---|
| Ethan Lim | 6.00 | 1 |
| Grace Lee | 4.50 | 1 |
| Hannah Goh | 6.00 | 1 |
| Hannah Tan | 18.00 | 1 |
| Leo Tan | 18.00 | 1 |
| Mia Tan | 4.50 | 1 |
| Nathan Chua | 1.50 | 1 |
| Nathan Chua | 12.00 | 1 |
| Sebastian Tan | 10.50 | 1 |

| Month | Total Fine | Collected Fine | Unpaid Fine | Overdue Books |
|---|---|---|---|---|
| 2025-03 | 229.50 | 57.00 | 172.50 | 16 |

| Borrower Name | Total Fine | Overdue Books |
|---|---|---|
| Alexander Lee | 12.00 | 1 |
| Benjamin Tan | 12.00 | 2 |
| Emily Ng | 3.00 | 1 |
| Gabriel Lee | 36.00 | 2 |
| Henry Tan | 1.50 | 1 |
| Isaac Wong | 7.50 | 1 |
| Isabella Ng | 12.00 | 1 |
| Isabella Tan | 19.50 | 1 |
| Isla Ng | 27.00 | 1 |
| Jack Wong | 45.00 | 1 |
| Jack Wong | 6.00 | 1 |
| Lily Ng | 1.50 | 1 |
| Max Wong | 36.00 | 1 |
| Ryan Goh | 10.50 | 1 |

| Month | Total Fine | Collected Fine | Unpaid Fine | Overdue Books |
|---|---|---|---|---|
| 2025-04 | 15.00 | .00 | 15.00 | 2 |

| Borrower Name | Total Fine | Overdue Books |
|---|---|---|
| Ryan Goh | 6.00 | 1 |
| Sophia Tan | 9.00 | 1 |




### 4.3.8 Report 2: Top Reserved Books and Reservation Status Analysis

**Purpose: The purpose of this report is to identify the top 10 most reserved books in a given year and provides a detailed breakdown of their reservation statuses (e.g., Approved, Rejected, Cancelled). It helps assess book demand versus availability trends to support inventory and policy decisions.**

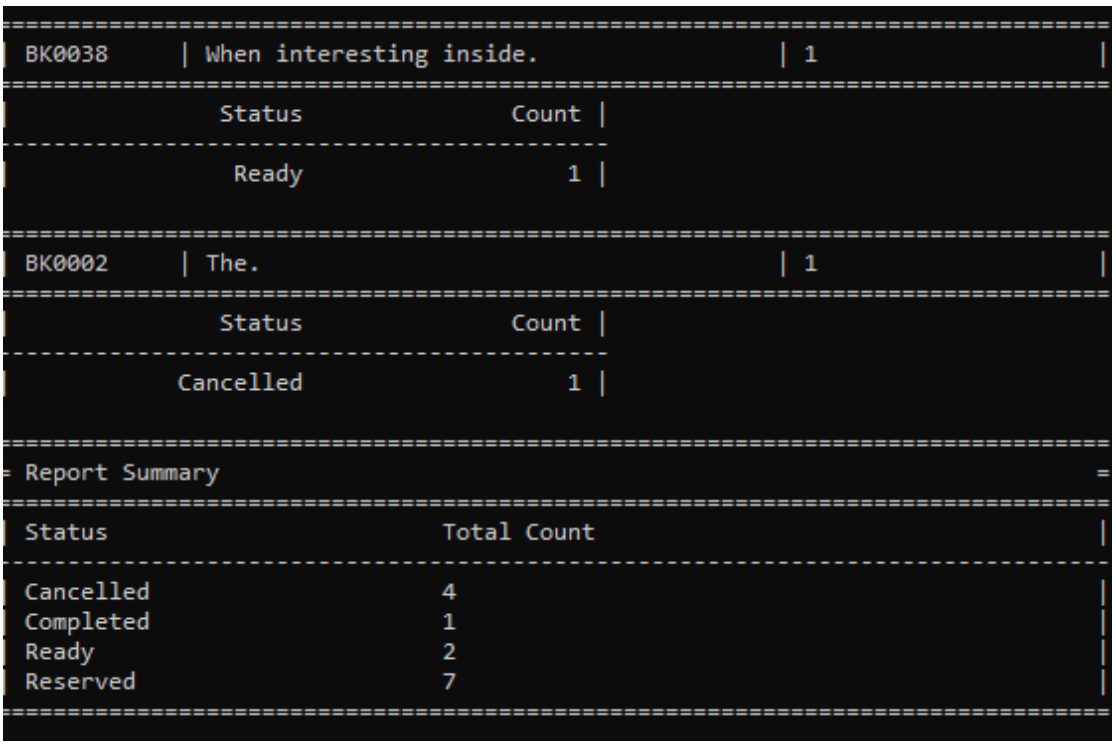
**Report code:**

```
CREATE OR REPLACE PROCEDURE report_reservation_vs_availability(
    p_year IN NUMBER
) AS
    -- Cursor: Top 10 most reserved books
    CURSOR book_cursor IS
        SELECT
            b.book_id,
            b.name AS book_title,
            COUNT(*) AS total_reservations
        FROM reservation r
        JOIN book b ON r.book_id = b.book_id
        WHERE TO_CHAR(r.rsv_datetime, 'YYYY') = TO_CHAR(p_year)
        GROUP BY b.book_id, b.name
        ORDER BY total_reservations DESC
        FETCH FIRST 10 ROWS ONLY;

    -- Cursor: Status breakdown per book
    CURSOR status_cursor(p_book_id book.book_id%TYPE) IS
        SELECT
            r.rsv_status,
            COUNT(*) AS count_status
        FROM reservation r
        WHERE r.book_id = p_book_id
          AND TO_CHAR(r.rsv_datetime, 'YYYY') = TO_CHAR(p_year)
        GROUP BY r.rsv_status;

    -- Associative array for status totals
    TYPE status_map_type IS TABLE OF NUMBER INDEX BY VARCHAR2(50);
    status_totals status_map_type;

    status_key VARCHAR2(50);

BEGIN
    DBMS_OUTPUT.PUT_LINE(CHR(10));
    DBMS_OUTPUT.PUT_LINE('Reservation Demand vs Availability Report
for ' || p_year);
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 80, '='));
    DBMS_OUTPUT.PUT_LINE('| ' || RPAD('Book ID', 10) || ' | ' ||
                         RPAD('Book Title', 40) || ' | ' ||
                         RPAD('Total Reservations', 20) || ' |');
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 80, '='));

    FOR book_rec IN book_cursor LOOP
        DBMS_OUTPUT.PUT_LINE('| ' || RPAD(book_rec.book_id, 10) || ' |
' ||
                             RPAD(book_rec.book_title, 40) || ' | ' ||
```

```
                            RPAD(book_rec.total_reservations, 20) ||
' |');
        DBMS_OUTPUT.PUT_LINE(RPAD('=', 80, '='));
        DBMS_OUTPUT.PUT_LINE('| ' || LPAD('Status', 20) ||
                             LPAD('Count', 20) || ' |');
        DBMS_OUTPUT.PUT_LINE(RPAD('-', 44, '-'));

        FOR stat_rec IN status_cursor(book_rec.book_id) LOOP
            DBMS_OUTPUT.PUT_LINE('| ' || LPAD(stat_rec.rsv_status, 20)
||
                                    LPAD(stat_rec.count_status, 20) ||
' |');

            -- Accumulate into associative array safely
            IF status_totals.EXISTS(stat_rec.rsv_status) THEN
                status_totals(stat_rec.rsv_status) :=
status_totals(stat_rec.rsv_status) + stat_rec.count_status;
            ELSE
                status_totals(stat_rec.rsv_status) :=
stat_rec.count_status;
            END IF;
        END LOOP;
        DBMS_OUTPUT.PUT_LINE(CHR(13));
        DBMS_OUTPUT.PUT_LINE(RPAD('=', 80, '='));
    END LOOP;

    -- Print status summary
    DBMS_OUTPUT.PUT_LINE(RPAD('= Report Summary', 79) || '=');
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 80, '='));
    DBMS_OUTPUT.PUT_LINE('| ' || RPAD('Status', 30) || RPAD('Total
Count', 46) || ' |');
    DBMS_OUTPUT.PUT_LINE(RPAD('-', 80, '-'));

    status_key := status_totals.FIRST;
    WHILE status_key IS NOT NULL LOOP
        DBMS_OUTPUT.PUT_LINE('| ' || RPAD(status_key, 30) ||
RPAD(status_totals(status_key), 46) || ' |');
        status_key := status_totals.NEXT(status_key);
    END LOOP;

    DBMS_OUTPUT.PUT_LINE(RPAD('=', 80, '='));
END;
/
```

**Sample Output:**

```
Reservation Demand vs Availability Report for 2025
================================================================================
 Book ID    | Book Title                                | Total Reservations   |
================================================================================
 BK0035     | Away single record.                       | 2                    |
================================================================================
              Status              Count |
-----------------------------------------
           Cancelled                  1 |
            Reserved                  1 |

================================================================================
 BK0046     | Experience moment.                        | 2                    |
================================================================================
              Status              Count |
-----------------------------------------
            Reserved                  1 |
           Cancelled                  1 |

================================================================================
 BK0059     | Left fight must safe.                     | 2                    |
================================================================================
              Status              Count |
-----------------------------------------
           Completed                  1 |
           Cancelled                  1 |

================================================================================
 BK0100     | Car forget.                               | 2                    |
================================================================================
              Status              Count |
-----------------------------------------
            Reserved                  2 |

================================================================================
 BK0031     | Threat.                                   | 1                    |
================================================================================
              Status              Count |
-----------------------------------------
            Reserved                  1 |

================================================================================
 BK0027     | Manage sense if.                          | 1                    |
================================================================================
              Status              Count |
-----------------------------------------
             Ready                    1 |

================================================================================
 BK0022     | Cost western perform evening.             | 1                    |
================================================================================
              Status              Count |
-----------------------------------------
            Reserved                  1 |

================================================================================
 BK0018     | Consider.                                 | 1                    |
================================================================================
              Status              Count |
-----------------------------------------
            Reserved                  1 |
```



```
BK0038     | When interesting inside.                | 1
              Status              Count |
               Ready                  1 |

BK0002     | The.                                    | 1
              Status              Count |
           Cancelled                  1 |

Report Summary
Status                        Total Count
Cancelled                     4
Completed                     1
Ready                         2
Reserved                      7
```

## 4.4 Kevin Lo Yung Khang

### 4.4.1 Query 1: Task Assignment Overview by Staff

**Purpose: The purpose of this query is to identify the most frequently assigned tasks and their corresponding staff to aid in planning staff development and resource allocation.**

**SQL statement:**

```
COLUMN task_name FORMAT A40
COLUMN staff_name FORMAT A20
COLUMN total_assignments FORMAT 9999

SELECT
    t.description AS task_name,
    s.name AS staff_name,
    COUNT(aa.task_id) AS total_assignments
FROM
    Auto_Assign aa
JOIN
    Task t ON aa.task_id = t.task_id
JOIN
    Staff s ON aa.staff_id = s.staff_id
GROUP BY
    t.description, s.name
ORDER BY
    total_assignments DESC;
```

**Sample Output:**

```
TASK_NAME                                STAFF_NAME           TOTAL_ASSIGNMENTS
---------------------------------------- -------------------- -----------------
Catalog New Arrivals - Batch 1           Daisy Black                          4
Check Overdue List And Send Reminders    Fiona Red                            3
Inventory Audit - Nonfiction Section     Bob White                            3
Inventory Audit - Nonfiction Section     George Yellow                        3
Inventory Audit - Nonfiction Section     Daisy Black                          3
Restock Stationery Supplies              Julia Azure                          3
Database Backup and Maintenance          Charlie Green                        3
Organize Fiction Bookshelves A-G         Charlie Green                        3
Restock Stationery Supplies              Siti Aminah                          3
Check Overdue List And Send Reminders    Bob White                            3
Security Patrol - Level 1 and 2          Edward Blue                          3

TASK_NAME                                STAFF_NAME           TOTAL_ASSIGNMENTS
---------------------------------------- -------------------- -----------------
Prepare for Children Story Time Event    Fiona Red                            3
Database Backup and Maintenance          Edward Blue                          3
Database Backup and Maintenance          Ahmad Bakar                          2
Catalog New Arrivals - Batch 1           Kevin Teal                           2
Inventory Audit - Nonfiction Section     Charlie Green                        2
Database Backup and Maintenance          Jane Smith                           2
Database Backup and Maintenance          Fiona Red                            2
Database Backup and Maintenance          Siti Aminah                          2
Prepare for Children Story Time Event    Kevin Teal                           2
Clean And Sanitize Study Area Zone B     Charlie Green                        2
Minor Book Repair (Taping, Cleaning)     Julia Azure                          2
```

```
TASK_NAME                                STAFF_NAME           TOTAL_ASSIGNMENTS
---------------------------------------- -------------------- -----------------
Restock Stationery Supplies              Edward Blue                          2
Prepare for Children Story Time Event    Daisy Black                          2
Restock Stationery Supplies              Bob White                            2
Prepare for Children Story Time Event    Bob White                            1
Minor Book Repair (Taping, Cleaning)     Jane Smith                           1
Security Patrol - Level 1 and 2          John Doe                             1
Inventory Audit - Nonfiction Section     Edward Blue                          1
Database Backup and Maintenance          John Doe                             1
Assist Users at Digital Kiosks           Siti Aminah                          1
Database Backup and Maintenance          George Yellow                        1
Inventory Audit - Nonfiction Section     Fiona Red                            1

TASK_NAME                                STAFF_NAME           TOTAL_ASSIGNMENTS
---------------------------------------- -------------------- -----------------
Security Patrol - Level 1 and 2          Siti Aminah                          1
Restock Stationery Supplies              Kevin Teal                           1
Inventory Audit - Nonfiction Section     Kevin Teal                           1
Organize Fiction Bookshelves A-G         Fiona Red                            1
Organize Fiction Bookshelves A-G         Ian Grey                             1
Inventory Audit - Nonfiction Section     Jane Smith                           1
Organize Fiction Bookshelves A-G         George Yellow                        1
Catalog New Arrivals - Batch 1           Edward Blue                          1
Check Overdue List And Send Reminders    Alice Brown                          1
Catalog New Arrivals - Batch 1           Alice Brown                          1
Security Patrol - Level 1 and 2          Hannah Purple                        1

TASK_NAME                                STAFF_NAME           TOTAL_ASSIGNMENTS
---------------------------------------- -------------------- -----------------
Security Patrol - Level 1 and 2          Bob White                            1
Organize Fiction Bookshelves A-G         Julia Azure                          1
Prepare for Children Story Time Event    Ahmad Bakar                          1
Check Overdue List And Send Reminders    Edward Blue                          1
Security Patrol - Level 1 and 2          Charlie Green                        1
Clean And Sanitize Study Area Zone B     Jane Smith                           1
Inventory Audit - Nonfiction Section     Siti Aminah                          1
Prepare for Children Story Time Event    John Doe                             1
Restock Stationery Supplies              Hannah Purple                        1
Restock Stationery Supplies              Charlie Green                        1
```

```
TASK_NAME                                STAFF_NAME           TOTAL_ASSIGNMENTS
---------------------------------------- -------------------- -----------------
Clean And Sanitize Study Area Zone B     Ian Grey                             1
Assist Users at Digital Kiosks           Jane Smith                           1
Inventory Audit - Nonfiction Section     John Doe                             1
Assist Users at Digital Kiosks           Fiona Red                            1
Restock Stationery Supplies              Alice Brown                          1
Restock Stationery Supplies              Daisy Black                          1
Security Patrol - Level 1 and 2          George Yellow                        1
Security Patrol - Level 1 and 2          Fiona Red                            1
Minor Book Repair (Taping, Cleaning)     Kevin Teal                           1
Catalog New Arrivals - Batch 1           Hannah Purple                        1
Minor Book Repair (Taping, Cleaning)     Hannah Purple                        1

TASK_NAME                                STAFF_NAME           TOTAL_ASSIGNMENTS
---------------------------------------- -------------------- -----------------
Prepare for Children Story Time Event    Edward Blue                          1
Database Backup and Maintenance          Bob White                            1

68 rows selected.
```

### 4.4.2 Query 2: Staff Attendance Performance (Last 3 Month)

**Purpose: To review the attendance patterns of active staff members for the previous calendar month, showing counts for 'Present', 'Absent', and 'Leave' statuses. This helps identify potential attendance issues or trends.**

**SQL statement:**

```
COLUMN staff_name FORMAT A25 HEADING 'Staff Name'
COLUMN present_count FORMAT 9999 HEADING 'Present'
COLUMN absent_count FORMAT 9999 HEADING 'Absent'
COLUMN leave_count FORMAT 9999 HEADING 'Leave'

SELECT
    s.name AS staff_name,
    COUNT(CASE WHEN a.att_status = 'Present' THEN 1 END) AS
present_count,
    COUNT(CASE WHEN a.att_status = 'Absent' THEN 1 END) AS
absent_count,
    COUNT(CASE WHEN a.att_status = 'Leave' THEN 1 END) AS leave_count
FROM
    Staff s
JOIN
    Attendance a ON s.staff_id = a.staff_id
WHERE
    s.stf_status = 'Active'
AND
    a.att_date >= TRUNC(ADD_MONTHS(SYSDATE, -3), 'MM') -- Start of
last month (edit here for choosing how many months)
AND
    a.att_date < TRUNC(SYSDATE, 'MM') -- Start of this month
(exclusive)
GROUP BY
    s.staff_id, s.name
ORDER BY
    absent_count DESC, leave_count DESC, staff_name;
```

**Sample Output:**

| Staff Name | Present | Absent | Leave |
|---|---|---|---|
| Edward Blue | 1 | 3 | 6 |
| Bob White | 0 | 3 | 2 |
| John Doe | 1 | 2 | 4 |
| Hannah Purple | 1 | 2 | 2 |
| Siti Aminah | 1 | 2 | 1 |
| Ahmad Bakar | 0 | 2 | 0 |
| Daisy Black | 3 | 1 | 3 |
| George Yellow | 1 | 1 | 3 |
| Kevin Teal | 2 | 1 | 1 |
| Jane Smith | 3 | 0 | 3 |
| Julia Azure | 1 | 0 | 3 |

### 4.4.3 Procedure 1: Assign Manual Task Schedule

**Purpose: Allows a manager or system administrator to manually assign a specific task to an active staff member for a defined time slot. It creates a schedule record and an initial 'Scheduled' progress entry.**

**SQL statement:**

```
CREATE OR REPLACE PROCEDURE prc_Assign_Manual_Schedule (
    p_staff_id_num  IN NUMBER, -- Expecting raw number e.g., 1
    p_task_id_num   IN NUMBER, -- Expecting raw number e.g., 1
    p_start_datetime IN DATE,
    p_end_datetime  IN DATE
) AS
    v_formatted_staff_id Staff.staff_id%TYPE;
    v_formatted_task_id  Task.task_id%TYPE;
    v_staff_status       Staff.stf_status%TYPE;
    v_task_status        Task.tk_status%TYPE;
    v_schedule_id        Schedule.schedule_id%TYPE;
    -- Custom Named Exceptions
    e_invalid_input      EXCEPTION;
    e_inactive_entity    EXCEPTION;
    e_invalid_datetime   EXCEPTION;
    e_entity_not_found   EXCEPTION;
    PRAGMA EXCEPTION_INIT(e_invalid_input, -20101);      -- For
Staff/Task not found
    PRAGMA EXCEPTION_INIT(e_inactive_entity, -20102);    -- For
Staff/Task inactive
    PRAGMA EXCEPTION_INIT(e_invalid_datetime, -20103);  -- For invalid
start/end dates
    PRAGMA EXCEPTION_INIT(e_entity_not_found, -20105);   -- Generic
not found if needed

BEGIN
    -- Format IDs
    v_formatted_staff_id := 'SF' || LPAD(p_staff_id_num, 4, '0');
    v_formatted_task_id  := 'TK' || LPAD(p_task_id_num, 4, '0');

    -- Validate Input Datetimes
    IF p_start_datetime IS NULL OR p_end_datetime IS NULL OR
p_end_datetime <= p_start_datetime THEN
        RAISE_APPLICATION_ERROR(-20103, 'Invalid schedule dates: End
datetime must be after start datetime.');
    END IF;

    -- Check if Staff exists and is active
    BEGIN
        SELECT stf_status INTO v_staff_status FROM Staff WHERE
staff_id = v_formatted_staff_id;
        IF v_staff_status != 'Active' THEN
            RAISE_APPLICATION_ERROR(-20102, 'Staff ' ||
v_formatted_staff_id || ' is not active.');
        END IF;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20101, 'Staff ' ||
v_formatted_staff_id || ' not found.');
    END;

    -- Check if Task exists and is active
    BEGIN
        SELECT tk_status INTO v_task_status FROM Task WHERE task_id =
v_formatted_task_id;
         IF v_task_status != 'Active' THEN
            RAISE_APPLICATION_ERROR(-20102, 'Task ' ||
v_formatted_task_id || ' is not active.');
        END IF;
    EXCEPTION
```

```
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20101, 'Task ' ||
v_formatted_task_id || ' not found.');
    END;

    -- Insert into Schedule (Trigger handles auto-increment ID)
    SAVEPOINT before_schedule_insert;
    INSERT INTO Schedule (task_id, staff_id, start_datetime,
end_datetime)
    VALUES (v_formatted_task_id, v_formatted_staff_id,
p_start_datetime, p_end_datetime)
    RETURNING schedule_id INTO v_schedule_id;

    -- Insert initial Task Progress (Trigger handles auto-increment
ID)
    INSERT INTO Task_Progress (schedule_id, tp_status, tp_datetime)
    VALUES (v_schedule_id, 'Scheduled', SYSDATE);

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Successfully created Schedule ID: ' ||
v_schedule_id ||
                         ' for Staff: ' || v_formatted_staff_id ||
                         ', Task: ' || v_formatted_task_id ||
                         ' from ' || TO_CHAR(p_start_datetime,
'DD-MON-YYYY HH24:MI') ||
                         ' to ' || TO_CHAR(p_end_datetime,
'DD-MON-YYYY HH24:MI'));

EXCEPTION
    -- Catch specific custom errors
    WHEN e_invalid_input OR e_inactive_entity OR e_invalid_datetime OR
e_entity_not_found THEN
        ROLLBACK TO before_schedule_insert;
        DBMS_OUTPUT.PUT_LINE(SQLERRM); -- Display the custom error
message

    -- Catch overlap error raised by trigger (using assumed error
code)
    WHEN OTHERS THEN
        ROLLBACK TO before_schedule_insert;
        IF SQLCODE = -20111 THEN -- Check if it's the overlap error
code
            DBMS_OUTPUT.PUT_LINE('Error: Schedule overlaps with an
existing assignment for Staff ' || v_formatted_staff_id);
        -- Can add checks for other trigger errors (-20112, -20113) if
needed
        ELSE
            DBMS_OUTPUT.PUT_LINE('An unexpected error occurred during
schedule assignment: ' || SQLERRM);
        END IF;
END;
/
```

**Sample Input :**
**- - For StaffId = SF002, Assign task = TK0003 on April 25th, 2025 from 9am to 11am**

```
BEGIN
    prc_Assign_Manual_Schedule(
        p_staff_id_num   => 2, -- Corresponds to SF0002
        p_task_id_num    => 4, -- Corresponds to TK0003
        p_start_datetime => TO_DATE('2025-04-25
09:00:00', 'YYYY-MM-DD HH24:MI:SS'),
        p_end_datetime   => TO_DATE('2025-04-25
11:00:00', 'YYYY-MM-DD HH24:MI:SS')
    );
```

```
END;
/
```

**Sample Output:**

```
SQL> -- Expected Output: Success message with a new Schedule ID (e.g., SC0111).
SQL> -- Expected DB Changes: New row in Schedule, new row in Task_Progress.
SQL> SELECT staff_id, name, stf_status
  2  FROM Staff
  3  WHERE staff_id = 'SF0002';

STAFF_ NAME                                               STF_STATUS
------ -------------------------------------------------- --------------------
SF0002 Jane Smith                                         Active

SQL> SELECT task_id, description, tk_status
  2  FROM Task
  3  WHERE task_id = 'TK0003';

TASK_I DESCRIPTION                                                                                                                                TK_STATUS
------ ---------------------------------------------------------------------------------------------------------------------------------------------
------------------------------ --------------------
TK0003 Database Backup and Maintenance                                                                                                            Active

SQL> SELECT schedule_id, task_id, staff_id,
  2         TO_CHAR(start_datetime, 'YYYY-MM-DD HH24:MI:SS') AS start_time,
  3         TO_CHAR(end_datetime, 'YYYY-MM-DD HH24:MI:SS') AS end_time,
  4         attend_datetime
  5  FROM Schedule
  6  WHERE staff_id = 'SF0002'
  7    AND TRUNC(start_datetime) = TO_DATE('2025-04-25', 'YYYY-MM-DD')
  8  ORDER BY start_datetime;

no rows selected
```

1. here shows that staff_id = 2 is SF0002 is 'Active'
2. here shows that task_id = 3 is TK0003 is 'Active' to assign
3. shows that staff on this specific day have no pre-existing schedules

```
SQL> SET SERVEROUTPUT ON;
SQL> BEGIN
  2      prc_Assign_Manual_Schedule(
  3          p_staff_id_num   => 2, -- Corresponds to SF0002
  4          p_task_id_num    => 3, -- Corresponds to TK0003
  5          p_start_datetime => TO_DATE('2025-04-25 09:00:00', 'YYYY-MM-DD HH24:MI:SS'),
  6          p_end_datetime   => TO_DATE('2025-04-25 11:00:00', 'YYYY-MM-DD HH24:MI:SS')
  7      );
  8  END;
  9  /
Successfully created Schedule ID: SC0117 for Staff: SF0002, Task: TK0003 from 25-APR-2025 09:00 to 25-APR-2025 11:00

PL/SQL procedure successfully completed.

SQL> SELECT schedule_id, task_id, staff_id,
  2         TO_CHAR(start_datetime, 'YYYY-MM-DD HH24:MI:SS') AS start_time,
  3         TO_CHAR(end_datetime, 'YYYY-MM-DD HH24:MI:SS') AS end_time,
  4         attend_datetime -- Verify this is NULL
  5  FROM Schedule
  6  WHERE staff_id = 'SF0002'
  7    AND TRUNC(start_datetime) = TO_DATE('2025-04-25', 'YYYY-MM-DD')
  8  ORDER BY start_datetime;

SCHEDU TASK_I STAFF_ START_TIME          END_TIME            ATTEND_DA
------ ------ ------ ------------------- ------------------- ---------
SC0117 TK0003 SF0002 2025-04-25 09:00:00 2025-04-25 11:00:00
```

1. assign task to the staff
2. shows that staff successfully been successfully assign task

### 4.4.4 Procedure 2: Update Task Progress

**Purpose: Allows staff or managers to update the status of a scheduled task (e.g., mark as 'Ongoing', 'Completed', 'Cancelled') and optionally provide a reason. This inserts a new record into the Task_Progress history.**

**SQL statement:**

```
CREATE OR REPLACE PROCEDURE prc_Update_Task_Progress (
    p_schedule_id_num IN NUMBER, -- Expecting raw number e.g., 1
    p_new_status      IN VARCHAR2, -- Use VARCHAR2 for flexibility
    p_reason          IN Task_Progress.reason%TYPE DEFAULT NULL
) AS
```

```
    -- Variables
    v_formatted_schedule_id Schedule.schedule_id%TYPE;
    v_schedule_exists       NUMBER;
    v_current_status        Task_Progress.tp_status%TYPE;
    v_input_status_upper    VARCHAR2(20) := UPPER(TRIM(p_new_status));
-- Store uppercase, trimmed version

    -- Custom Named Exceptions for better error handling
    e_invalid_status        EXCEPTION;
    e_schedule_not_found    EXCEPTION;
    e_invalid_transition    EXCEPTION;
    e_no_progress_found     EXCEPTION; -- For data inconsistency
    PRAGMA EXCEPTION_INIT(e_invalid_status, -20104);
    PRAGMA EXCEPTION_INIT(e_schedule_not_found, -20105);
    PRAGMA EXCEPTION_INIT(e_invalid_transition, -20106);
    PRAGMA EXCEPTION_INIT(e_no_progress_found, -20107);

BEGIN
    -- 1. Format Input ID
    v_formatted_schedule_id := 'SC' || LPAD(p_schedule_id_num, 4,
'0');

    -- 2. Validate Input Status (Case-Insensitive)
    IF v_input_status_upper NOT IN ('SCHEDULED', 'ONGOING',
'COMPLETED', 'CANCELLED') THEN
        RAISE_APPLICATION_ERROR(-20104, 'Invalid status provided: '''
|| p_new_status || '''. Must be Scheduled, Ongoing, Completed, or
Cancelled.');
    END IF;

    -- 3. Check if Schedule ID Exists in the Schedule table
    SELECT COUNT(*) INTO v_schedule_exists FROM Schedule s WHERE
s.schedule_id = v_formatted_schedule_id;
    IF v_schedule_exists = 0 THEN
         RAISE_APPLICATION_ERROR(-20105, 'Schedule ID ' ||
v_formatted_schedule_id || ' not found.');
    END IF;

    -- 4. Get Current Latest Status for the Schedule from
Task_Progress (*** FIXED PART ***)
    BEGIN
        SELECT tp_status
        INTO v_current_status
        FROM (
            SELECT tp.tp_status
            FROM Task_Progress tp
            WHERE tp.schedule_id = v_formatted_schedule_id
            ORDER BY tp.tp_datetime DESC
        )
        WHERE ROWNUM = 1; -- Get only the first row (latest) after
ordering
    EXCEPTION
         WHEN NO_DATA_FOUND THEN
             RAISE_APPLICATION_ERROR(-20107, 'Data inconsistency: No
progress records found for existing Schedule ' ||
v_formatted_schedule_id || '. Initial record missing.');
    END;

    -- 5. Validate Status Transition Logic (same as before)
    IF UPPER(v_current_status) IN ('COMPLETED', 'CANCELLED') AND
v_input_status_upper != UPPER(v_current_status) THEN
         RAISE_APPLICATION_ERROR(-20106, 'Invalid transition: Task is
already ' || v_current_status || '. Cannot change status to ' ||
INITCAP(p_new_status) || '.');
    END IF;
    IF UPPER(v_current_status) = 'ONGOING' AND v_input_status_upper =
'SCHEDULED' THEN
```

```
            RAISE_APPLICATION_ERROR(-20106, 'Invalid transition: Cannot
change status from Ongoing back to Scheduled.');
    END IF;
    IF UPPER(v_current_status) = v_input_status_upper THEN
         DBMS_OUTPUT.PUT_LINE('Info: Task for Schedule '||
v_formatted_schedule_id ||' is already in "'|| INITCAP(p_new_status)
||'" status. No update needed.');
         RETURN;
    END IF;

    -- 6. Insert New Progress Record (same as before)
    INSERT INTO Task_Progress (schedule_id, reason, tp_status,
tp_datetime)
    VALUES (v_formatted_schedule_id, p_reason, INITCAP(p_new_status),
SYSDATE);

    -- 7. Commit the Transaction
    COMMIT;

    -- 8. Success Message (same as before)
    DBMS_OUTPUT.PUT_LINE('Task Progress updated successfully for
Schedule ID: ' || v_formatted_schedule_id || ' to Status: ' ||
INITCAP(p_new_status));

EXCEPTION
    -- Exception handling block remains the same
    WHEN e_invalid_status OR e_schedule_not_found OR
e_invalid_transition OR e_no_progress_found THEN
         DBMS_OUTPUT.PUT_LINE(SQLERRM);
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('An unexpected error occurred while
updating task progress for Schedule '|| v_formatted_schedule_id ||': '
|| SQLCODE || ' - ' || SQLERRM);
END;
/
```

**Sample Input :**
**- - For schedule_id_num = 37, p_new status = Ongoing, Reason = Task is now being worked on**

```
BEGIN
    prc_Update_Task_Progress(37, 'Ongoing', 'Task is now being worked
on');
END;
/
```

**Sample Output:**

```
SQL> SET SERVEROUTPUT ON;
SQL> SELECT progress_id, schedule_id, reason, tp_status,
  2    TO_CHAR(tp_datetime, 'YYYY-MM-DD HH24:MI:SS') AS progress_time
  3    FROM Task_Progress tp
  4    WHERE schedule_id = 'SC0037' ORDER BY tp_datetime DESC;

PROGRE SCHEDU REASON                                                                                          TP_STATUS           PROGRESS_TIME
------ ------ ------------------------------------------------------------------------------------------------ ------------------- -------------
----
TP0039 SC0037                                                                                                 Scheduled           2025-04-16 18:1
7:24
TP0001 SC0037 Between hope view put cost.                                                                     Scheduled           2025-03-31 07:1
5:55
```

1. see the actual task progress

```
Procedure created.

SQL> BEGIN
  2      prc_Update_Task_Progress(
  3          p_schedule_id_num => 37,
  4  p_new_status      => 'Ongoing',
  5          p_reason          => 'Work has started'
  6      );
  7  END;
  8  /
Task Progress updated successfully for Schedule ID: SC0037 to Status: Ongoing

PL/SQL procedure successfully completed.

SQL>
```

1. update the schedule

```
SQL> SELECT progress_id, schedule_id, reason, tp_status,
  2    TO_CHAR(tp_datetime, 'YYYY-MM-DD HH24:MI:SS') AS progress_time
  3    FROM Task_Progress tp
  4    WHERE schedule_id = 'SC0037' ORDER BY tp_datetime DESC;

PROGRE SCHEDU REASON                                                                                          TP_STATUS           PROGRESS_TIME
------ ------ ------------------------------------------------------------------------------------------------ ------------------- ----------------
----
TP0108 SC0037 Work has started                                                                                Ongoing             2025-05-04 20:5
5:27
TP0039 SC0037                                                                                                 Scheduled           2025-04-16 18:1
7:24
```

1. shows that it is updated

```
SQL> DBMS_OUTPUT.PUT_LINE('---> Test Case 3: BEFORE <---');
SP2-0734: unknown command beginning "DBMS_OUTPU..." - rest of line ignored.
SQL> SELECT progress_id, schedule_id, reason, tp_status,
  2         TO_CHAR(tp_datetime, 'YYYY-MM-DD HH24:MI:SS') AS progress_time
  3  FROM Task_Progress
  4  WHERE schedule_id = 'SC0038' ORDER BY tp_datetime DESC;

PROGRE SCHEDU REASON                                                                                          TP_STATUS           PROGRESS_TIME
------ ------ ------------------------------------------------------------------------------------------------ ------------------- ----------------
----
TP0014 SC0038 Along carry country strategy let garden.                                                        Scheduled           2025-03-29 09:3
7:42
```

1. shows the data details for schedule_id = ‘SC0038’

```
SQL> BEGIN
  2      prc_Update_Task_Progress(
  3          p_schedule_id_num => 38,
  4  p_new_status      => 'Scheduled' );
  5  END;
  6  /
Info: Task for Schedule SC0038 is already in "Scheduled" status. No update needed.

PL/SQL procedure successfully completed.
```

1. data successfully cancel the update if it is the same status

```
SQL> DBMS_OUTPUT.PUT_LINE('---> Test Case 4: RUNNING PROCEDURE <---');
SP2-0734: unknown command beginning "DBMS_OUTPU..." - rest of line ignored.
SQL> BEGIN
  2      prc_Update_Task_Progress(
  3          p_schedule_id_num => 1,
  4          p_new_status      => 'Pending' -- Invalid status
  5      );
  6  END;
  7  /
ORA-20104: Invalid status provided: 'Pending'. Must be Scheduled, Ongoing, Completed, or Cancelled.

PL/SQL procedure successfully completed.
```

1. successfully cancel the update if the status is invalid (must be scheduled, ongoing, completed or cancelled)

```
SQL> DBMS_OUTPUT.PUT_LINE('---> Test Case 6: BEFORE <---');
SP2-0734: unknown command beginning "DBMS_OUTPU..." - rest of line ignored.
SQL> SELECT progress_id, schedule_id, reason, tp_status,
  2         TO_CHAR(tp_datetime, 'YYYY-MM-DD HH24:MI:SS') AS progress_time
  3  FROM Task_Progress
  4  WHERE schedule_id = 'SC0003' -- Use the actual ID
  5  ORDER BY tp_datetime DESC;

PROGRE SCHEDU REASON                                                                                                TP_STATUS           PROGRESS_TIME
------ ------ ---------------------------------------------------------------------------------------------------- ------------------- ---------------
----
TP0007 SC0003 Local structure what page current.                                                                    Ongoing             2025-04-19 10:2
8:09

SQL> DBMS_OUTPUT.PUT_LINE('---> Test Case 6: RUNNING PROCEDURE <---');
SP2-0734: unknown command beginning "DBMS_OUTPU..." - rest of line ignored.
SQL> BEGIN
  2      prc_Update_Task_Progress(
  3          p_schedule_id_num => 3, -- Use the actual ID
  4          p_new_status      => 'Scheduled'
  5      );
  6  END;
  7  /
ORA-20106: Invalid transition: Cannot change status from Ongoing back to Scheduled.

PL/SQL procedure successfully completed.
```

1. prevent when the status is ongoing it cannot reverse back to scheduled status

### 4.4.5 Trigger 1: Prevent Schedule Overlap for Staff

**Purpose: Ensures that a staff member cannot be assigned a new schedule (Schedule table) if the proposed time slot overlaps with any of their existing schedules. This trigger fires before an insert or update.**

**SQL statement:**

```
CREATE OR REPLACE TRIGGER trg_Prevent_Schedule_Overlap
BEFORE INSERT OR UPDATE ON Schedule
FOR EACH ROW
DECLARE
    v_overlap_count NUMBER;
    PRAGMA AUTONOMOUS_TRANSACTION; -- Needed for querying the same
table in row trigger
BEGIN
    -- Check for overlaps with other schedules for the same staff
member
    -- Overlap condition: (ExistingStart < NewEnd) AND (ExistingEnd >
NewStart)
    SELECT COUNT(*)
    INTO v_overlap_count
    FROM Schedule
    WHERE staff_id = :NEW.staff_id
      AND schedule_id != NVL(:OLD.schedule_id, '-1') -- Exclude self
if updating
      AND start_datetime < :NEW.end_datetime      -- Existing starts
before new one ends
      AND end_datetime > :NEW.start_datetime;     -- Existing ends
after new one starts

    IF v_overlap_count > 0 THEN
        -- Use a specific error code for overlap
        RAISE_APPLICATION_ERROR(-20111, 'Schedule overlap detected for
Staff ' || :NEW.staff_id ||
                                        ' between ' ||
TO_CHAR(:NEW.start_datetime, 'DD-MON HH24:MI') ||
                                        ' and ' ||
TO_CHAR(:NEW.end_datetime, 'DD-MON HH24:MI'));
    END IF;
    -- COMMIT; -- Commit autonomous transaction
EXCEPTION
   WHEN OTHERS THEN
       -- ROLLBACK; -- Rollback autonomous transaction on error
       -- Re-raise the original error or a generic one if needed
       RAISE_APPLICATION_ERROR(-20199, 'Error during overlap check: '
|| SQLERRM);
END;
/
```

**SAMPLE INPUT :**
**- - to test when add the time that is overlap**

```
INSERT INTO Schedule (task_id, staff_id, start_datetime, end_datetime)
VALUES (
    'TK0003', -- Different task, doesn't matter for overlap check
    'SF0002', -- Same Staff ID
    TO_DATE('2025-04-10 11:00:00', 'YYYY-MM-DD HH24:MI:SS'), -- Starts
DURING SC0030
    TO_DATE('2025-04-10 13:00:00', 'YYYY-MM-DD HH24:MI:SS')  -- Ends
AFTER SC0030
);
```

**Sample Output:**

```
SQL> SELECT schedule_id, task_id, staff_id,
  2         TO_CHAR(start_datetime, 'YYYY-MM-DD HH24:MI:SS') AS start_time,
  3         TO_CHAR(end_datetime, 'YYYY-MM-DD HH24:MI:SS') AS end_time
  4  FROM Schedule
  5  WHERE schedule_id = 'SC0030';

SCHEDU TASK_I STAFF_ START_TIME          END_TIME
------ ------ ------ ------------------- -------------------
SC0030 TK0002 SF0002 2025-04-10 10:01:00 2025-04-10 12:40:00
```

1. CHECK THE STAFF CURRENT DATA

```
SQL> INSERT INTO Schedule (task_id, staff_id, start_datetime, end_datetime)
  2  VALUES (
  3      'TK0003', -- Different task, doesn't matter for overlap check
  4      'SF0002', -- Same Staff ID
  5      TO_DATE('2025-04-10 11:00:00', 'YYYY-MM-DD HH24:MI:SS'), -- Starts DURING S
C0030
  6      TO_DATE('2025-04-10 13:00:00', 'YYYY-MM-DD HH24:MI:SS')  -- Ends AFTER SC00
30
  7  );
INSERT INTO Schedule (task_id, staff_id, start_datetime, end_datetime)
            *
ERROR at line 1:
ORA-20199: Error during overlap check: ORA-20111: Schedule overlap detected for Staf
f SF0002 between 10-APR 11:00 and 10-APR 13:00
ORA-06512: at "ASSIGNMENT.TRG_PREVENT_SCHEDULE_OVERLAP", line 26
ORA-04088: error during execution of trigger 'ASSIGNMENT.TRG_PREVENT_SCHEDULE_OVERLA
P'
```

1. THIS IS WHEN WE WANT TO ADD SCHEDULE WITH THE TIME THAT CURRENTLY SCHEDULED, IT WILL BE NOTIFIED

### 4.4.6 Trigger 2: Check Whether Staff is active to assign Task on Schedule Assignment

**Purpose: Prevents the creation or update of a Schedule record if either the assigned staff member (Staff.stf_status) or the assigned task (Task.tk_status) is currently 'Inactive'.**

**SQL statement:**

```
CREATE OR REPLACE TRIGGER trg_Check_Active_Staff_Task
BEFORE INSERT OR UPDATE ON Schedule
FOR EACH ROW
DECLARE
    v_staff_status Staff.stf_status%TYPE;
    v_task_status Task.tk_status%TYPE;
BEGIN
    -- Check Staff Status
    SELECT stf_status INTO v_staff_status
    FROM Staff
    WHERE staff_id = :NEW.staff_id;

    IF v_staff_status != 'Active' THEN
```

```
            RAISE_APPLICATION_ERROR(-20112, 'Cannot assign schedule: Staff
' || :NEW.staff_id || ' is inactive.');
        END IF;

        -- Check Task Status
        SELECT tk_status INTO v_task_status
        FROM Task
        WHERE task_id = :NEW.task_id;

        IF v_task_status != 'Active' THEN
            RAISE_APPLICATION_ERROR(-20112, 'Cannot assign schedule: Task
' || :NEW.task_id || ' is inactive.');
        END IF;

    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            -- This handles if staff_id or task_id doesn't exist (FK
    constraint should also catch this)
            RAISE_APPLICATION_ERROR(-20113, 'Cannot assign schedule:
    Staff or Task ID not found.');
        WHEN OTHERS THEN
            -- Handle unexpected errors during the check
            RAISE_APPLICATION_ERROR(-20198, 'Error checking staff/task
    status: ' || SQLERRM);
    END;
    /
```

**SAMPLE INPUT :**

```
-- try to assign task for the inactive staff
INSERT INTO Schedule (task_id, staff_id, start_datetime, end_datetime)
VALUES (
    'TK0003', -- Active Task
    'SF0005', -- Inactive Staff
    TO_DATE('2025-06-02 09:00:00', 'YYYY-MM-DD HH24:MI:SS'),
    TO_DATE('2025-06-02 11:00:00', 'YYYY-MM-DD HH24:MI:SS')
);
```

**Sample Output:**

```
SQL> select staff_id,name, stf_status from staff where staff_id = 'SF0005';

STAFF_ NAME                                               STF_STATUS
------ -------------------------------------------------- --------------------
SF0005 Alice Brown                                        Inactive
```

1. check the staff status

```
SQL> INSERT INTO Schedule (task_id, staff_id, start_datetime, end_datetime)
  2  VALUES (
  3      'TK0003', -- Active Task
  4      'SF0005', -- Inactive Staff
  5      TO_DATE('2025-06-02 09:00:00', 'YYYY-MM-DD HH24:MI:SS'),
  6      TO_DATE('2025-06-02 11:00:00', 'YYYY-MM-DD HH24:MI:SS')
  7  );
INSERT INTO Schedule (task_id, staff_id, start_datetime, end_datetime)
            *
ERROR at line 1:
ORA-20198: Error checking staff/task status: ORA-20112: Cannot assign schedule: Staf
f SF0005 is inactive.
ORA-06512: at "ASSIGNMENT.TRG_CHECK_ACTIVE_STAFF_TASK", line 29
ORA-04088: error during execution of trigger 'ASSIGNMENT.TRG_CHECK_ACTIVE_STAFF_TASK
'
```

1. successfully prevent to add task to inactive staff

```
SQL> INSERT INTO Schedule (task_id, staff_id, start_datetime, end_datetime)
  2  VALUES (
  3      'TK0003', -- Active Task
  4      'SF9999', -- Non-existent Staff
  5      TO_DATE('2025-06-04 09:00:00', 'YYYY-MM-DD HH24:MI:SS'),
  6      TO_DATE('2025-06-04 11:00:00', 'YYYY-MM-DD HH24:MI:SS')
  7  );
INSERT INTO Schedule (task_id, staff_id, start_datetime, end_datetime)
            *
ERROR at line 1:
ORA-20113: Cannot assign schedule: Staff or Task ID not found.
ORA-06512: at "ASSIGNMENT.TRG_CHECK_ACTIVE_STAFF_TASK", line 26
ORA-04088: error during execution of trigger 'ASSIGNMENT.TRG_CHECK_ACTIVE_STAFF_TASK
'
```

1. successfully prevent add when staff is non-exist also

### 4.4.7 Report 1: Staff Performance Summary Report

**Purpose: Generates a performance summary for a specific staff member over a given date range. It includes counts of completed tasks, total scheduled hours, and an attendance breakdown (Present, Absent, Leave days).**

**SQL statement:**

```
CREATE OR REPLACE PROCEDURE rpt_Staff_Performance_Summary (
    p_staff_id_num  IN NUMBER, -- Renamed parameter for clarity
    p_start_date    IN DATE,
    p_end_date      IN DATE
) AS
    v_formatted_staff_id Staff.staff_id%TYPE;
    v_staff_name         Staff.name%TYPE;
    v_staff_status       Staff.stf_status%TYPE;
    v_tasks_completed    NUMBER := 0;
    v_total_hours        NUMBER := 0;
    v_present_days       NUMBER := 0;
    v_absent_days        NUMBER := 0;
    v_leave_days         NUMBER := 0;
    v_latest_progress_status Task_Progress.tp_status%TYPE; -- Ensure
declared
    v_sched_id           Schedule.schedule_id%TYPE;
    v_sched_start        DATE;
    v_sched_end          DATE;
    v_att_status         Attendance.att_status%TYPE;

    -- Cursor 1: Staff Details
    CURSOR staff_cursor (cp_staff_id VARCHAR2) IS
        SELECT name, stf_status FROM Staff WHERE staff_id =
cp_staff_id;

    -- Cursor 2: Scheduled Tasks within the period
    CURSOR schedule_cursor (cp_staff_id VARCHAR2, cp_start DATE,
cp_end DATE) IS
        SELECT schedule_id, start_datetime, end_datetime
        FROM Schedule
        WHERE staff_id = cp_staff_id
          AND start_datetime >= TRUNC(cp_start)
          AND start_datetime < TRUNC(cp_end) + 1;

    -- Cursor 3: Attendance within the period
    CURSOR attendance_cursor (cp_staff_id VARCHAR2, cp_start DATE,
cp_end DATE) IS
        SELECT att_status FROM Attendance
        WHERE staff_id = cp_staff_id
          AND att_date BETWEEN TRUNC(cp_start) AND TRUNC(cp_end);

BEGIN
    v_formatted_staff_id := 'SF' || LPAD(p_staff_id_num, 4, '0');
```

```
    -- Check End Date >= Start Date
    IF TRUNC(p_end_date) < TRUNC(p_start_date) THEN
        DBMS_OUTPUT.PUT_LINE('Error: End date must be on or after
start date.');
        RETURN;
    END IF;
    OPEN staff_cursor(v_formatted_staff_id);
    FETCH staff_cursor INTO v_staff_name, v_staff_status;
    IF staff_cursor%NOTFOUND THEN
        CLOSE staff_cursor;
        DBMS_OUTPUT.PUT_LINE('Staff ' || v_formatted_staff_id || ' not
found.');
        RETURN;
    END IF;
    CLOSE staff_cursor;

    OPEN schedule_cursor(v_formatted_staff_id, p_start_date,
p_end_date);
    LOOP
        FETCH schedule_cursor INTO v_sched_id, v_sched_start,
v_sched_end;
        EXIT WHEN schedule_cursor%NOTFOUND;
        v_total_hours := v_total_hours + (v_sched_end - v_sched_start)
* 24;

        BEGIN
            SELECT tp_status
            INTO v_latest_progress_status
            FROM (
                SELECT tp.tp_status
                FROM Task_Progress tp
                WHERE tp.schedule_id = v_sched_id
                ORDER BY tp.tp_datetime DESC
            )
            WHERE ROWNUM = 1; -- Use ROWNUM = 1 instead of FETCH FIRST

            IF v_latest_progress_status = 'Completed' THEN
                v_tasks_completed := v_tasks_completed + 1;
            END IF;
        EXCEPTION WHEN NO_DATA_FOUND THEN NULL;
        END;

    END LOOP;
    CLOSE schedule_cursor;

    OPEN attendance_cursor(v_formatted_staff_id, p_start_date,
p_end_date);
    LOOP
        FETCH attendance_cursor INTO v_att_status;
        EXIT WHEN attendance_cursor%NOTFOUND;
        IF v_att_status = 'Present' THEN v_present_days :=
v_present_days + 1;
        ELSIF v_att_status = 'Absent' THEN v_absent_days :=
v_absent_days + 1;
        ELSIF v_att_status = 'Leave' THEN v_leave_days := v_leave_days
+ 1;
        END IF;
    END LOOP;
    CLOSE attendance_cursor;

    DBMS_OUTPUT.PUT_LINE(CHR(10));
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 60, '='));
    DBMS_OUTPUT.PUT_LINE('| Staff Performance Summary Report' ||
LPAD('|', 27));
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 60, '='));
    DBMS_OUTPUT.PUT_LINE('Period: ' || TO_CHAR(p_start_date,
'DD-MON-YYYY') || ' to ' || TO_CHAR(p_end_date, 'DD-MON-YYYY'));
```

```
    DBMS_OUTPUT.PUT_LINE('Staff ID    : ' || v_formatted_staff_id);
    DBMS_OUTPUT.PUT_LINE('Staff Name  : ' || v_staff_name);
    DBMS_OUTPUT.PUT_LINE('Staff Status: ' || v_staff_status);
    DBMS_OUTPUT.PUT_LINE(RPAD('-', 60, '-'));
    DBMS_OUTPUT.PUT_LINE('Performance Metrics:');
    DBMS_OUTPUT.PUT_LINE(' - Tasks Completed       : ' ||
v_tasks_completed);
    DBMS_OUTPUT.PUT_LINE(' - Total Scheduled Hours : ' ||
TO_CHAR(NVL(v_total_hours, 0), '9990.00'));
    DBMS_OUTPUT.PUT_LINE(RPAD('-', 60, '-'));
    DBMS_OUTPUT.PUT_LINE('Attendance Summary:');
    DBMS_OUTPUT.PUT_LINE(' - Days Present          : ' ||
v_present_days);
    DBMS_OUTPUT.PUT_LINE(' - Days Absent           : ' ||
v_absent_days);
    DBMS_OUTPUT.PUT_LINE(' - Days On Leave         : ' ||
v_leave_days);
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 60, '='));
EXCEPTION
    WHEN OTHERS THEN
         IF staff_cursor%ISOPEN THEN CLOSE staff_cursor; END IF;
         IF schedule_cursor%ISOPEN THEN CLOSE schedule_cursor; END IF;
         IF attendance_cursor%ISOPEN THEN CLOSE attendance_cursor; END
IF;
         DBMS_OUTPUT.PUT_LINE('Error generating staff performance
report: ' || SQLERRM);
END;
/
```

**Sample Input : For StaffID = 11 from date 2025 Jan 01 to 31**

```
BEGIN
rpt_Staff_Performance_Summary(11, TO_DATE('01-JAN-2025',
'DD-MON-YYYY'), TO_DATE('31-JAN-2025', 'DD-MON-YYYY'));
END;
/
```

**Sample Output:**

```
============================================================
| Staff Performance Summary Report                         |
============================================================
Period: 01-JAN-2025 to 31-JAN-2025
Staff ID    : SF0011
Staff Name  : George Yellow
Staff Status: Active
------------------------------------------------------------
Performance Metrics:
- Tasks Completed       : 0
- Total Scheduled Hours :     0.00
------------------------------------------------------------
Attendance Summary:
- Days Present          : 0
- Days Absent           : 1
- Days On Leave         : 1
============================================================
```

### 4.4.8 Report 2: Staff Weekly Task Schedule Report

**Purpose: Displays the detailed weekly schedule for a specified staff member, starting from a given date. It lists all assigned tasks day by day for that week, including task description, scheduled times, and the latest progress status.**

**SQL statement:**

```
CREATE OR REPLACE PROCEDURE rpt_Staff_Weekly_Schedule (
    p_staff_id_num     IN NUMBER, -- Expecting raw number
```

```
    p_week_start_date  IN DATE
) AS
    v_formatted_staff_id Staff.staff_id%TYPE;
    v_staff_name         Staff.name%TYPE;
    v_week_end_date      DATE := TRUNC(p_week_start_date) + 6;
    v_current_date       DATE;
    v_tasks_found_on_day BOOLEAN;
    v_report_has_data    BOOLEAN := FALSE;
    v_latest_status      Task_Progress.tp_status%TYPE; -- Declare
variable for status

    -- Cursor for Staff Details
    CURSOR staff_cursor (cp_staff_id VARCHAR2) IS
        SELECT name FROM Staff WHERE staff_id = cp_staff_id AND
stf_status = 'Active';

    -- Cursor for Schedules on a specific day (Simpler definition)
    CURSOR schedule_cursor (cp_staff_id VARCHAR2, cp_date DATE) IS
        SELECT
            sc.schedule_id,
            t.description AS task_description,
            sc.start_datetime,
            sc.end_datetime
        FROM Schedule sc
        JOIN Task t ON sc.task_id = t.task_id
        WHERE sc.staff_id = cp_staff_id
          AND TRUNC(sc.start_datetime) = TRUNC(cp_date) -- Match the
date part only
        ORDER BY sc.start_datetime;

    -- Define a record type based on the simpler cursor
    TYPE schedule_rec_type IS RECORD (
         schedule_id      Schedule.schedule_id%TYPE,
         task_description Task.description%TYPE,
         start_datetime   Schedule.start_datetime%TYPE,
         end_datetime     Schedule.end_datetime%TYPE
    );
    sched_rec schedule_rec_type; -- Declare variable of the record
type

BEGIN
    v_formatted_staff_id := 'SF' || LPAD(p_staff_id_num, 4, '0');

    -- Get Staff Name using Cursor
    OPEN staff_cursor(v_formatted_staff_id);
    FETCH staff_cursor INTO v_staff_name;
    IF staff_cursor%NOTFOUND THEN
        CLOSE staff_cursor;
        DBMS_OUTPUT.PUT_LINE('Active Staff ' || v_formatted_staff_id
|| ' not found.');
        RETURN;
    END IF;
    CLOSE staff_cursor;

    -- Print Report Header (Code as before)
    DBMS_OUTPUT.PUT_LINE(CHR(10));
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 100, '='));
    DBMS_OUTPUT.PUT_LINE('| Weekly Schedule Report' || LPAD('|', 77));
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 100, '='));
    DBMS_OUTPUT.PUT_LINE('Staff ID    : ' || v_formatted_staff_id);
    DBMS_OUTPUT.PUT_LINE('Staff Name  : ' || v_staff_name);
    DBMS_OUTPUT.PUT_LINE('Week Starting: ' ||
TO_CHAR(TRUNC(p_week_start_date), 'DD-MON-YYYY') ||
                         ' (to ' || TO_CHAR(v_week_end_date,
'DD-MON-YYYY') || ')');
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 100, '='));

    -- Loop through each day of the week
```

```
    FOR i IN 0..6 LOOP
        v_current_date := TRUNC(p_week_start_date) + i;
        DBMS_OUTPUT.PUT_LINE('| ' || TO_CHAR(v_current_date,
'DD-MON-YYYY (DY)') || LPAD('|', 80));
        DBMS_OUTPUT.PUT_LINE('| ' || RPAD('Time Slot', 15) ||
RPAD('Task Description', 50) || RPAD('Status', 15) || LPAD('|', 17));
        DBMS_OUTPUT.PUT_LINE('| ' || RPAD('-', 15, '-') || RPAD('-',
50, '-') || RPAD('-', 15, '-') || LPAD('|', 17));

        v_tasks_found_on_day := FALSE;
        -- Open and loop through the schedule cursor for the current
day
        OPEN schedule_cursor(v_formatted_staff_id, v_current_date);
        LOOP
            FETCH schedule_cursor INTO sched_rec; -- Fetch into the
declared record variable
            EXIT WHEN schedule_cursor%NOTFOUND;

            v_tasks_found_on_day := TRUE;
            v_report_has_data := TRUE;

            -- Get the latest status for this specific schedule_id
separately (*** FIXED PART ***)
            BEGIN
                 SELECT tp_status
                 INTO v_latest_status
                 FROM ( -- Inline view to order first
                      SELECT tp.tp_status
                      FROM Task_Progress tp
                      WHERE tp.schedule_id = sched_rec.schedule_id --
Use ID from fetched record
                      ORDER BY tp.tp_datetime DESC
                 ) ordered_progress -- Give the inline view an alias
                 WHERE ROWNUM = 1; -- Filter for the top row (latest)
            EXCEPTION
                WHEN NO_DATA_FOUND THEN
                    v_latest_status := 'Scheduled'; -- Default if no
progress yet
            END;
            -- *** END OF FIXED PART ***

            -- Print the line using data from sched_rec and
v_latest_status
            DBMS_OUTPUT.PUT_LINE('| ' ||
RPAD(TO_CHAR(sched_rec.start_datetime, 'HH24:MI') || '-' ||
TO_CHAR(sched_rec.end_datetime, 'HH24:MI'), 15) ||

RPAD(SUBSTR(sched_rec.task_description, 1, 48), 50) ||
                                  RPAD(NVL(v_latest_status,
'Scheduled'), 15) || LPAD('|', 17));
        END LOOP;
        CLOSE schedule_cursor; -- Close cursor for the day

        IF NOT v_tasks_found_on_day THEN
             DBMS_OUTPUT.PUT_LINE('| ' || RPAD('No tasks scheduled for
this day.', 80) || LPAD('|', 17));
        END IF;
         DBMS_OUTPUT.PUT_LINE(RPAD('-', 100, '-'));
    END LOOP; -- End loop through days

    IF NOT v_report_has_data THEN
         DBMS_OUTPUT.PUT_LINE('| No schedules found for this staff
member during the specified week. |');
         DBMS_OUTPUT.PUT_LINE(RPAD('-', 100, '-'));
    END IF;
    DBMS_OUTPUT.PUT_LINE(RPAD('=', 100, '='));

EXCEPTION
```

```
    WHEN OTHERS THEN
         IF staff_cursor%ISOPEN THEN CLOSE staff_cursor; END IF;
         IF schedule_cursor%ISOPEN THEN CLOSE schedule_cursor; END IF;
         DBMS_OUTPUT.PUT_LINE('Error generating weekly schedule
report: ' || SQLERRM);
END;
/
```

**Sample Input : For StaffID = 1 on date 2025-04-12 onwards**

```
BEGIN
rpt_Staff_Weekly_Schedule(1, DATE '2025-04-12');
END;
/
```

**Sample Output:**

```
================================================================================
====================
| Weekly Schedule Report
|
================================================================================
====================
Staff ID    : SF0001
Staff Name  : John Doe
Week Starting: 12-APR-2025 (to 18-APR-2025)
================================================================================
====================
| 12-APR-2025 (SAT)
|
| Time Slot      Task Description                                Status
|
|
--------------------------------------------------------------------------------
|
| 08:45-13:20    Security Patrol - Level 1 and 2                 Completed
|
--------------------------------------------------------------------------------
--------------------
| 13-APR-2025 (SUN)
|
| Time Slot      Task Description                                Status
|
|
--------------------------------------------------------------------------------
|
| No tasks scheduled for this day.
|
--------------------------------------------------------------------------------
```

## 4.5 Daniel Tay Yi Fung

### 4.5.1 Query 1: Current Ongoing Tasks and Assigned Staff

**Purpose: The purpose of this query is to list all currently ongoing tasks along with the staff handling them, helping managers monitor daily execution.**

**SQL statement:**

```
-- Set column formats and custom headings
COLUMN tp_datetime   FORMAT A20 HEADING 'Progress DateTime'
COLUMN tp_status     FORMAT A15 HEADING 'Status'
COLUMN task_description  FORMAT A80 HEADING 'Task Description'
COLUMN staff_name    FORMAT A40 HEADING 'Staff Name'

-- Query
SELECT
 tp.tp_datetime,
 tp.tp_status,
 t.description AS task_description,
 s.name AS staff_name
FROM
 Task_Progress tp
JOIN
 Schedule sch ON tp.schedule_id = sch.schedule_id
JOIN
 Task t ON sch.task_id = t.task_id
JOIN
 Staff s ON sch.staff_id = s.staff_id
WHERE
 tp.tp_status = 'Ongoing'
ORDER BY
 tp.tp_datetime DESC;
```

**Sample Output:**

| Progress DateTime | Status | Task Description | Staff Name |
|---|---|---|---|
| 20-APR-25 | Ongoing | Prepare for Children Story Time Event | George Yellow |
| 20-APR-25 | Ongoing | Database Backup and Maintenance | Bob White |
| 19-APR-25 | Ongoing | Restock Stationery Supplies | Siti Aminah |
| 19-APR-25 | Ongoing | Inventory Audit - Nonfiction Section | Fiona Red |
| 19-APR-25 | Ongoing | Catalog New Arrivals - Batch 1 | Ahmad Bakar |
| 18-APR-25 | Ongoing | Inventory Audit - Nonfiction Section | Daisy Black |
| 17-APR-25 | Ongoing | Inventory Audit - Nonfiction Section | Charlie Green |
| 17-APR-25 | Ongoing | Check Overdue List And Send Reminders | Ahmad Bakar |
| 16-APR-25 | Ongoing | Inventory Audit - Nonfiction Section | Daisy Black |
| 15-APR-25 | Ongoing | Prepare for Children Story Time Event | Fiona Red |
| 15-APR-25 | Ongoing | Restock Stationery Supplies | Alice Brown |
| 14-APR-25 | Ongoing | Security Patrol - Level 1 and 2 | Edward Blue |
| 14-APR-25 | Ongoing | Restock Stationery Supplies | George Yellow |
| 14-APR-25 | Ongoing | Minor Book Repair (Taping, Cleaning) | Jane Smith |
| 14-APR-25 | Ongoing | Inventory Audit - Nonfiction Section | Julia Azure |
| 12-APR-25 | Ongoing | Security Patrol - Level 1 and 2 | Edward Blue |
| 12-APR-25 | Ongoing | Clean And Sanitize Study Area Zone B | Edward Blue |
| 12-APR-25 | Ongoing | Assist Users at Digital Kiosks | Charlie Green |
| 11-APR-25 | Ongoing | Restock Stationery Supplies | Siti Aminah |
| 11-APR-25 | Ongoing | Inventory Audit - Nonfiction Section | Charlie Green |
| 10-APR-25 | Ongoing | Minor Book Repair (Taping, Cleaning) | Bob White |
| 08-APR-25 | Ongoing | Prepare for Children Story Time Event | Ian Grey |
| 07-APR-25 | Ongoing | Security Patrol - Level 1 and 2 | Edward Blue |
| 07-APR-25 | Ongoing | Organize Fiction Bookshelves A-G | Charlie Green |
| 07-APR-25 | Ongoing | Organize Fiction Bookshelves A-G | Ian Grey |
| 06-APR-25 | Ongoing | Catalog New Arrivals - Batch 1 | Jane Smith |
| 05-APR-25 | Ongoing | Inventory Audit - Nonfiction Section | Daisy Black |
| 03-APR-25 | Ongoing | Restock Stationery Supplies | Fiona Red |
| 02-APR-25 | Ongoing | Organize Fiction Bookshelves A-G | Hannah Purple |
| 31-MAR-25 | Ongoing | Catalog New Arrivals - Batch 1 | Kevin Teal |
| 31-MAR-25 | Ongoing | Organize Fiction Bookshelves A-G | Charlie Green |
| 30-MAR-25 | Ongoing | Database Backup and Maintenance | Alice Brown |
| 30-MAR-25 | Ongoing | Security Patrol - Level 1 and 2 | Edward Blue |
| 28-MAR-25 | Ongoing | Minor Book Repair (Taping, Cleaning) | Alice Brown |
| 27-MAR-25 | Ongoing | Clean And Sanitize Study Area Zone B | Kevin Teal |
| 26-MAR-25 | Ongoing | Organize Fiction Bookshelves A-G | Alice Brown |
| 25-MAR-25 | Ongoing | Assist Users at Digital Kiosks | Charlie Green |
| 24-MAR-25 | Ongoing | Inventory Audit - Nonfiction Section | Julia Azure |
| 23-MAR-25 | Ongoing | Catalog New Arrivals - Batch 1 | Fiona Red |
| 23-MAR-25 | Ongoing | Clean And Sanitize Study Area Zone B | John Doe |

### 4.5.2 Query 2: Recent Task Progress Tracker

**Purpose: The purpose of this query is to track the most recent task progress for each staff member over the past 90 days. This query will help in monitoring the status and completion times of tasks assigned to staff, providing insight into their recent performance and progress.**

**SQL Statement:**

```
COLUMN staff_name FORMAT A20
COLUMN task_description FORMAT A80
COLUMN tp_status FORMAT A15
COLUMN tp_datetime FORMAT A20
SELECT
 s.name AS staff_name,
 t.description AS task_description,
 tp.tp_status,
 tp.tp_datetime
FROM
 Task_Progress tp
JOIN
 Schedule sc ON tp.schedule_id = sc.schedule_id
JOIN
 Staff s ON sc.staff_id = s.staff_id
JOIN
 Task t ON sc.task_id = t.task_id
WHERE
 tp.tp_datetime >= SYSDATE - 90
ORDER BY
 tp.tp_datetime DESC;
```

**Sample Output:**

| Staff Name | Task Description | Status | Progress DateTime |
|---|---|---|---|
| Bob White | Assist Users at Digital Kiosks | Completed | 23-APR-25 |
| George Yellow | Prepare for Children Story Time Event | Ongoing | 20-APR-25 |
| Bob White | Database Backup and Maintenance | Ongoing | 20-APR-25 |
| Fiona Red | Database Backup and Maintenance | Scheduled | 19-APR-25 |
| Siti Aminah | Restock Stationery Supplies | Ongoing | 19-APR-25 |
| Fiona Red | Inventory Audit - Nonfiction Section | Ongoing | 19-APR-25 |
| Daisy Black | Assist Users at Digital Kiosks | Scheduled | 19-APR-25 |
| Ahmad Bakar | Catalog New Arrivals - Batch 1 | Ongoing | 19-APR-25 |
| Daisy Black | Inventory Audit - Nonfiction Section | Ongoing | 18-APR-25 |
| Ahmad Bakar | Security Patrol - Level 1 and 2 | Completed | 18-APR-25 |
| Ian Grey | Database Backup and Maintenance | Completed | 17-APR-25 |
| Ian Grey | Database Backup and Maintenance | Scheduled | 17-APR-25 |
| Charlie Green | Inventory Audit - Nonfiction Section | Ongoing | 17-APR-25 |
| Ahmad Bakar | Check Overdue List And Send Reminders | Ongoing | 17-APR-25 |
| Bob White | Assist Users at Digital Kiosks | Scheduled | 16-APR-25 |
| Bob White | Assist Users at Digital Kiosks | Completed | 16-APR-25 |
| Ian Grey | Database Backup and Maintenance | Scheduled | 16-APR-25 |
| Siti Aminah | Security Patrol - Level 1 and 2 | Completed | 16-APR-25 |
| Hannah Purple | Catalog New Arrivals - Batch 1 | Scheduled | 16-APR-25 |
| Daisy Black | Inventory Audit - Nonfiction Section | Ongoing | 16-APR-25 |
| George Yellow | Prepare for Children Story Time Event | Scheduled | 16-APR-25 |
| Fiona Red | Restock Stationery Supplies | Completed | 15-APR-25 |
| Fiona Red | Prepare for Children Story Time Event | Ongoing | 15-APR-25 |
| Fiona Red | Catalog New Arrivals - Batch 1 | Completed | 15-APR-25 |
| Kevin Teal | Clean And Sanitize Study Area Zone B | Scheduled | 15-APR-25 |
| John Doe | Security Patrol - Level 1 and 2 | Completed | 15-APR-25 |
| Alice Brown | Restock Stationery Supplies | Ongoing | 15-APR-25 |
| Edward Blue | Security Patrol - Level 1 and 2 | Ongoing | 14-APR-25 |
| George Yellow | Restock Stationery Supplies | Ongoing | 14-APR-25 |
| Bob White | Assist Users at Digital Kiosks | Scheduled | 14-APR-25 |
| Jane Smith | Minor Book Repair (Taping, Cleaning) | Ongoing | 14-APR-25 |
| Julia Azure | Inventory Audit - Nonfiction Section | Ongoing | 14-APR-25 |
| Ahmad Bakar | Check Overdue List And Send Reminders | Completed | 13-APR-25 |
| George Yellow | Prepare for Children Story Time Event | Scheduled | 13-APR-25 |
| Charlie Green | Catalog New Arrivals - Batch 1 | Completed | 13-APR-25 |
| Daisy Black | Organize Fiction Bookshelves A-G | Completed | 12-APR-25 |
| Edward Blue | Security Patrol - Level 1 and 2 | Ongoing | 12-APR-25 |
| Edward Blue | Clean And Sanitize Study Area Zone B | Ongoing | 12-APR-25 |
| Fiona Red | Inventory Audit - Nonfiction Section | Scheduled | 12-APR-25 |
| Charlie Green | Assist Users at Digital Kiosks | Ongoing | 12-APR-25 |
| Daisy Black | Organize Fiction Bookshelves A-G | Scheduled | 12-APR-25 |
| Julia Azure | Inventory Audit - Nonfiction Section | Scheduled | 11-APR-25 |
| Siti Aminah | Restock Stationery Supplies | Ongoing | 11-APR-25 |
| Charlie Green | Inventory Audit - Nonfiction Section | Ongoing | 11-APR-25 |
| Edward Blue | Organize Fiction Bookshelves A-G | Completed | 10-APR-25 |
| Jane Smith | Security Patrol - Level 1 and 2 | Scheduled | 10-APR-25 |
| Bob White | Minor Book Repair (Taping, Cleaning) | Ongoing | 10-APR-25 |

| Staff Name | Task Description | Status | Progress DateTime |
|---|---|---|---|
| Daisy Black | Check Overdue List And Send Reminders | Scheduled | 09-APR-25 |
| Fiona Red | Catalog New Arrivals - Batch 1 | Completed | 09-APR-25 |
| Edward Blue | Organize Fiction Bookshelves A-G | Scheduled | 08-APR-25 |
| Ian Grey | Prepare for Children Story Time Event | Ongoing | 08-APR-25 |
| Fiona Red | Restock Stationery Supplies | Completed | 07-APR-25 |
| Edward Blue | Security Patrol - Level 1 and 2 | Ongoing | 07-APR-25 |
| Charlie Green | Organize Fiction Bookshelves A-G | Ongoing | 07-APR-25 |
| Ian Grey | Organize Fiction Bookshelves A-G | Ongoing | 07-APR-25 |
| Ahmad Bakar | Security Patrol - Level 1 and 2 | Completed | 07-APR-25 |
| Charlie Green | Prepare for Children Story Time Event | Completed | 06-APR-25 |
| Jane Smith | Catalog New Arrivals - Batch 1 | Ongoing | 06-APR-25 |
| Jane Smith | Minor Book Repair (Taping, Cleaning) | Scheduled | 06-APR-25 |
| Daisy Black | Inventory Audit - Nonfiction Section | Ongoing | 05-APR-25 |
| Kevin Teal | Catalog New Arrivals - Batch 1 | Completed | 04-APR-25 |
| Ian Grey | Organize Fiction Bookshelves A-G | Completed | 04-APR-25 |
| Julia Azure | Inventory Audit - Nonfiction Section | Scheduled | 04-APR-25 |
| Daisy Black | Organize Fiction Bookshelves A-G | Completed | 03-APR-25 |
| Fiona Red | Restock Stationery Supplies | Ongoing | 03-APR-25 |
| Fiona Red | Catalog New Arrivals - Batch 1 | Scheduled | 03-APR-25 |
| Fiona Red | Database Backup and Maintenance | Scheduled | 02-APR-25 |
| Hannah Purple | Prepare for Children Story Time Event | Scheduled | 02-APR-25 |
| Hannah Purple | Organize Fiction Bookshelves A-G | Ongoing | 02-APR-25 |
| Hannah Purple | Organize Fiction Bookshelves A-G | Scheduled | 02-APR-25 |
| Fiona Red | Clean And Sanitize Study Area Zone B | Scheduled | 01-APR-25 |
| Kevin Teal | Catalog New Arrivals - Batch 1 | Ongoing | 31-MAR-25 |
| Ahmad Bakar | Catalog New Arrivals - Batch 1 | Completed | 31-MAR-25 |
| Jane Smith | Catalog New Arrivals - Batch 1 | Completed | 31-MAR-25 |
| Charlie Green | Organize Fiction Bookshelves A-G | Ongoing | 31-MAR-25 |
| Alice Brown | Database Backup and Maintenance | Ongoing | 30-MAR-25 |
| Edward Blue | Security Patrol - Level 1 and 2 | Ongoing | 30-MAR-25 |
| John Doe | Clean And Sanitize Study Area Zone B | Scheduled | 30-MAR-25 |
| Kevin Teal | Clean And Sanitize Study Area Zone B | Completed | 30-MAR-25 |
| George Yellow | Minor Book Repair (Taping, Cleaning) | Scheduled | 29-MAR-25 |
| Alice Brown | Minor Book Repair (Taping, Cleaning) | Ongoing | 28-MAR-25 |
| Hannah Purple | Assist Users at Digital Kiosks | Scheduled | 28-MAR-25 |
| Hannah Purple | Assist Users at Digital Kiosks | Scheduled | 27-MAR-25 |
| Kevin Teal | Clean And Sanitize Study Area Zone B | Ongoing | 27-MAR-25 |
| Fiona Red | Catalog New Arrivals - Batch 1 | Scheduled | 27-MAR-25 |
| Charlie Green | Assist Users at Digital Kiosks | Scheduled | 27-MAR-25 |
| Siti Aminah | Security Patrol - Level 1 and 2 | Scheduled | 27-MAR-25 |
| Edward Blue | Security Patrol - Level 1 and 2 | Completed | 27-MAR-25 |
| Fiona Red | Database Backup and Maintenance | Scheduled | 27-MAR-25 |
| Charlie Green | Catalog New Arrivals - Batch 1 | Completed | 26-MAR-25 |
| Alice Brown | Organize Fiction Bookshelves A-G | Ongoing | 26-MAR-25 |
| Daisy Black | Inventory Audit - Nonfiction Section | Scheduled | 25-MAR-25 |
| Bob White | Database Backup and Maintenance | Scheduled | 25-MAR-25 |
| Charlie Green | Assist Users at Digital Kiosks | Ongoing | 25-MAR-25 |

| Staff Name | Task Description | Status | Progress DateTime |
|---|---|---|---|
| Ahmad Bakar | Security Patrol - Level 1 and 2 | Completed | 24-MAR-25 |
| Bob White | Database Backup and Maintenance | Scheduled | 24-MAR-25 |
| Julia Azure | Inventory Audit - Nonfiction Section | Ongoing | 24-MAR-25 |
| Charlie Green | Security Patrol - Level 1 and 2 | Completed | 24-MAR-25 |
| Fiona Red | Catalog New Arrivals - Batch 1 | Ongoing | 23-MAR-25 |
| Daisy Black | Inventory Audit - Nonfiction Section | Completed | 23-MAR-25 |
| John Doe | Clean And Sanitize Study Area Zone B | Ongoing | 23-MAR-25 |
| Fiona Red | Clean And Sanitize Study Area Zone B | Scheduled | 23-MAR-25 |
| Bob White | Security Patrol - Level 1 and 2 | Scheduled | 21-MAR-25 |
| Edward Blue | Organize Fiction Bookshelves A-G | Completed | 21-MAR-25 |
| Kevin Teal | Restock Stationery Supplies | Completed | 21-MAR-25 |

### 4.5.3 Procedure 1: Reschedule Task

**Purpose:**
**The prc_Reschedule_Task procedure is designed to reschedule an existing task assigned to a staff member in the Schedule table. It ensures that the new time slot does not conflict (overlap) with any of the staff member's other scheduled tasks before proceeding with the update. The procedure prc_Reschedule_Task helps manage task rescheduling for staff members while ensuring that no scheduling conflicts occur. It also logs all changes in the Shifting table for historical tracking and audit purposes.**

**SQL Statement:**

```
-- Format for Schedule table columns
COLUMN schedule_id    FORMAT A6  HEADING 'Schedule ID'
COLUMN task_id        FORMAT A6  HEADING 'Task ID'
COLUMN staff_id       FORMAT A6  HEADING 'Staff ID'
COLUMN start_datetime FORMAT A30 HEADING 'Start DateTime'
COLUMN end_datetime   FORMAT A30 HEADING 'End DateTime'
COLUMN attend_datetime FORMAT A30 HEADING 'Attend DateTime'

-- Format for Shifting table columns
COLUMN shift_id       FORMAT A6  HEADING 'Shift ID'
COLUMN schedule_id    FORMAT A6  HEADING 'Schedule ID'
COLUMN old_from_datetime  FORMAT A30 HEADING 'Old From DateTime'
COLUMN old_to_datetime FORMAT A30 HEADING 'Old To DateTime'
COLUMN new_from_datetime  FORMAT A30 HEADING 'New From DateTime'
COLUMN new_to_datetime FORMAT A30 HEADING 'New To DateTime'

-- select SC0004 in Schedule table
SELECT
 schedule_id,
 task_id,
 staff_id,
 TO_CHAR(start_datetime, 'YYYY-MM-DD HH24:MI:SS') AS start_datetime,
 TO_CHAR(end_datetime, 'YYYY-MM-DD HH24:MI:SS') AS end_datetime,
 TO_CHAR(attend_datetime, 'YYYY-MM-DD HH24:MI:SS') AS attend_datetime
FROM
 Schedule
WHERE
 schedule_id = 'SC0004';

-- select SC0004 in Shifting table
SELECT
 shift_id,
 schedule_id,
 TO_CHAR(old_from_datetime, 'YYYY-MM-DD HH24:MI:SS') AS
old_from_datetime,
 TO_CHAR(old_to_datetime,   'YYYY-MM-DD HH24:MI:SS') AS
old_to_datetime,
 TO_CHAR(new_from_datetime, 'YYYY-MM-DD HH24:MI:SS') AS
new_from_datetime,
 TO_CHAR(new_to_datetime,   'YYYY-MM-DD HH24:MI:SS') AS
new_to_datetime
FROM
 Shifting
WHERE
 schedule_id = 'SC0004';

CREATE OR REPLACE PROCEDURE prc_Reschedule_Task (
p_schedule_id_num   IN VARCHAR2,
p_new_start_datetime IN DATE,
p_new_end_datetime  IN DATE
) AS
v_staff_id Schedule.staff_id%TYPE;
v_overlap_count NUMBER;
BEGIN
```

```
-- Get the staff_id for the given schedule
SELECT staff_id INTO v_staff_id
FROM Schedule
WHERE schedule_id = p_schedule_id_num;

-- Check for overlapping schedules for this staff member
SELECT COUNT(*) INTO v_overlap_count
FROM Schedule
WHERE staff_id = v_staff_id
  AND schedule_id != p_schedule_id_num
  AND start_datetime < p_new_end_datetime
  AND end_datetime > p_new_start_datetime;

-- If overlap exists, exit early
IF v_overlap_count > 0 THEN
    DBMS_OUTPUT.PUT_LINE('Overlap detected. Cannot reschedule task for
Staff ID ' || v_staff_id);
    RETURN;
END IF;

-- No overlap found, proceed to update
UPDATE Schedule
SET start_datetime = p_new_start_datetime,
    end_datetime = p_new_end_datetime
WHERE schedule_id = p_schedule_id_num;

-- Insert a record into Shifting table to track the shift
INSERT INTO Shifting (
    shift_id,
    schedule_id,
    old_from_datetime,
    old_to_datetime,
    new_from_datetime,
    new_to_datetime
)
VALUES (
    'SH' || LPAD(shift_seq.NEXTVAL, 4, '0'),
    p_schedule_id_num,
    (SELECT start_datetime FROM Schedule WHERE schedule_id =
p_schedule_id_num),
    (SELECT end_datetime FROM Schedule WHERE schedule_id =
p_schedule_id_num),
    p_new_start_datetime,
    p_new_end_datetime
);

COMMIT;

DBMS_OUTPUT.PUT_LINE('Task with Schedule ' ||
p_schedule_id_num || ' has been rescheduled to ' ||
TO_CHAR(p_new_start_datetime, 'DD-MON-YYYY HH24:MI') ||
' to ' ||
TO_CHAR(p_new_end_datetime, 'DD-MON-YYYY HH24:MI'));

EXCEPTION
WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    ROLLBACK;
END;
/

BEGIN
prc_Reschedule_Task(
    p_schedule_id_num   => 'SC0004',
    p_new_start_datetime => TO_DATE('2025-04-29 14:00:00','YYYY-MM-DD
HH24:MI:SS'),
    p_new_end_datetime   => TO_DATE('2025-04-29 17:00:00', 'YYYY-MM-DD
HH24:MI:SS')
```

```
);
END;
/
```

```
SQL> SELECT
  2      shift_id,
  3      schedule_id,
  4      TO_CHAR(old_from_datetime, 'YYYY-MM-DD HH24:MI:SS') AS old_from_datetime,
  5      TO_CHAR(old_to_datetime,   'YYYY-MM-DD HH24:MI:SS') AS old_to_datetime,
  6      TO_CHAR(new_from_datetime, 'YYYY-MM-DD HH24:MI:SS') AS new_from_datetime,
  7      TO_CHAR(new_to_datetime,   'YYYY-MM-DD HH24:MI:SS') AS new_to_datetime
  8  FROM
  9      Shifting
 10  WHERE
 11      schedule_id = 'SC0004';

Shift  Schedu Old From DateTime           Old To DateTime             New From DateTime           New To DateTime
------ ------ ---------------------------- --------------------------- --------------------------- ---------------------------
SH0069 SC0004 2025-04-08 06:18:50          2025-04-08 07:18:50         2025-04-09 06:18:50         2025-04-09 07:18:50
```

Initially, this is the previous data for SC0004 from the shifting table.

```
SQL> BEGIN
  2  prc_Reschedule_Task(
  3      p_schedule_id_num   => 'SC0004',
  4      p_new_start_datetime => TO_DATE('2025-04-29 14:00:00','YYYY-MM-DD HH24:MI:SS'),
  5      p_new_end_datetime   => TO_DATE('2025-04-29 17:00:00', 'YYYY-MM-DD HH24:MI:SS')
  6  );
  7  END;
  8  /
Task with Schedule SC0004 has been rescheduled to 29-APR-2025 14:00 to 29-APR-2025 17:00

PL/SQL procedure successfully completed.
```

Reschedule SC0004 to the duration above.

```
SQL> SELECT
  2      shift_id,
  3      schedule_id,
  4      TO_CHAR(old_from_datetime, 'YYYY-MM-DD HH24:MI:SS') AS old_from_datetime,
  5      TO_CHAR(old_to_datetime,   'YYYY-MM-DD HH24:MI:SS') AS old_to_datetime,
  6      TO_CHAR(new_from_datetime, 'YYYY-MM-DD HH24:MI:SS') AS new_from_datetime,
  7      TO_CHAR(new_to_datetime,   'YYYY-MM-DD HH24:MI:SS') AS new_to_datetime
  8  FROM
  9      Shifting
 10  WHERE
 11      schedule_id = 'SC0004';

Shift  Schedu Old From DateTime           Old To DateTime             New From DateTime           New To DateTime
------ ------ ---------------------------- --------------------------- --------------------------- ---------------------------
SH0069 SC0004 2025-04-08 06:18:50          2025-04-08 07:18:50         2025-04-09 06:18:50         2025-04-09 07:18:50
SH0121 SC0004 2025-04-29 14:00:00          2025-04-29 17:00:00         2025-04-29 14:00:00         2025-04-29 17:00:00
```

After rescheduling, it will auto increment in the shifting table to add the reschedule data.

```
SQL> SELECT
  2      schedule_id,
  3      task_id,
  4      staff_id,
  5      TO_CHAR(start_datetime, 'YYYY-MM-DD HH24:MI:SS') AS start_datetime,
  6      TO_CHAR(end_datetime, 'YYYY-MM-DD HH24:MI:SS') AS end_datetime,
  7      TO_CHAR(attend_datetime, 'YYYY-MM-DD HH24:MI:SS') AS attend_datetime
  8  FROM
  9      Schedule
 10  WHERE
 11      schedule_id = 'SC0004';

Schedu Task I Staff  Start DateTime               End DateTime                Attend DateTime
------ ------ ------ ---------------------------- --------------------------- ---------------------------
SC0004 TK0009 SF0006 2025-04-29 14:00:00          2025-04-29 17:00:00         2025-04-08 12:02:00
```

In the Schedule table, it will also be updated.

### 4.5.4 Procedure 2: Update Staff Attendance

**Purpose:**
**The prc_Update_Staff_Attendance procedure is designed to manage staff attendance records by either updating existing records or inserting new ones, depending on whether an attendance record already exists for a specific staff member on a specific date. This procedure ensures that attendance for a staff member is accurately recorded or updated, without creating duplicate entries, and that attendance logs remain consistent and traceable.**

**SQL Statement:**

```
SELECT
staff_id,
 TO_CHAR(checkin_datetime, 'YYYY-MM-DD HH24:MI:SS') AS checkin_time,
 TO_CHAR(checkout_datetime, 'YYYY-MM-DD HH24:MI:SS') AS checkout_time,
 TO_CHAR(att_date, 'YYYY-MM-DD HH24:MI:SS') AS att_date,
att_status
FROM
 Attendance
WHERE
 staff_id = 'SF0002';

CREATE OR REPLACE PROCEDURE prc_Update_Staff_Attendance (
p_staff_id_num IN VARCHAR2,
p_att_date     IN DATE,
p_checkin_time IN DATE,
p_checkout_time   IN DATE,
p_att_status  IN VARCHAR2
) AS
BEGIN
-- Insert or update attendance record for the staff member
MERGE INTO Attendance a
USING dual
ON (a.staff_id = p_staff_id_num AND a.att_date = p_att_date)
WHEN MATCHED THEN
    UPDATE SET
        a.checkin_datetime = p_checkin_time,
        a.checkout_datetime = p_checkout_time,
        a.att_status = p_att_status
WHEN NOT MATCHED THEN
    INSERT (att_id, staff_id, att_date, checkin_datetime,
checkout_datetime, att_status)
    VALUES (
        'AT' || LPAD(attendance_seq.NEXTVAL, 4, '0'),  -- Generate
att_id
        p_staff_id_num,
        p_att_date,
        p_checkin_time,
        p_checkout_time,
        p_att_status
    );

COMMIT;
DBMS_OUTPUT.PUT_LINE('Attendance for staff ' || p_staff_id_num || ' on
' || TO_CHAR(p_att_date, 'DD-MON-YYYY') || ' updated to status: ' ||
p_att_status);
EXCEPTION
WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    ROLLBACK;
END;
/

BEGIN
prc_Update_Staff_Attendance(
    p_staff_id_num  => 'SF0002',
```

```
    p_att_date  => TO_DATE('2025-04-25', 'YYYY-MM-DD'),
    p_checkin_time  => TO_DATE('2025-04-25 08:00:00', 'YYYY-MM-DD
HH24:MI:SS'),
    p_checkout_time => TO_DATE('2025-04-25 17:00:00', 'YYYY-MM-DD
HH24:MI:SS'),
    p_att_status => 'Present'
);
END;
/
```

```
SQL> SELECT
  2      staff_id,
  3      TO_CHAR(checkin_datetime, 'YYYY-MM-DD HH24:MI:SS') AS checkin_time,
  4      TO_CHAR(checkout_datetime, 'YYYY-MM-DD HH24:MI:SS') AS checkout_time,
  5      TO_CHAR(att_date, 'YYYY-MM-DD HH24:MI:SS') AS att_date,
  6      att_status
  7  FROM
  8      Attendance
  9  WHERE
 10      staff_id = 'SF0002';

Staff  CHECKIN_TIME        CHECKOUT_TIME       ATT_DATE            ATT_STATUS
------ ------------------- ------------------- ------------------- --------------------
SF0002 2025-02-14 10:12:00 2025-02-14 17:26:00 2025-02-14 16:15:50 Present
SF0002 2025-02-15 09:24:00 2025-02-15 16:01:00 2025-02-15 11:08:16 Leave
SF0002 2025-02-01 08:03:00 2025-02-01 15:47:00 2025-02-01 21:14:03 Leave
SF0002 2025-01-22 10:49:00 2025-01-22 18:24:00 2025-01-22 03:44:51 Present
SF0002 2025-04-06 08:42:00 2025-04-06 15:22:00 2025-04-06 17:59:58 Present
SF0002 2025-02-23 10:05:00 2025-02-23 17:59:00 2025-02-23 23:35:29 Leave
SF0002 2025-03-16 09:19:00 2025-03-16 15:08:00 2025-03-16 05:55:34 Present
SF0002 2025-04-25 08:00:00 2025-04-25 17:00:00 2025-04-25 00:00:00 Absent
```

At the last row, the att_status is marked as absent.

```
SQL> BEGIN
  2  prc_Update_Staff_Attendance(
  3      p_staff_id_num  => 'SF0002',
  4      p_att_date  => TO_DATE('2025-04-25', 'YYYY-MM-DD'),
  5      p_checkin_time  => TO_DATE('2025-04-25 08:00:00', 'YYYY-MM-DD HH24:MI:SS'),
  6      p_checkout_time => TO_DATE('2025-04-25 17:00:00', 'YYYY-MM-DD HH24:MI:SS'),
  7      p_att_status => 'Present'
  8  );
  9  END;
 10  /
Attendance for staff SF0002 on 25-APR-2025 updated to status: Present

PL/SQL procedure successfully completed.
```

So, we will update SF0002's attendance to present.

```
SQL> SELECT
  2      staff_id,
  3      TO_CHAR(checkin_datetime, 'YYYY-MM-DD HH24:MI:SS') AS checkin_time,
  4      TO_CHAR(checkout_datetime, 'YYYY-MM-DD HH24:MI:SS') AS checkout_time,
  5      TO_CHAR(att_date, 'YYYY-MM-DD HH24:MI:SS') AS att_date,
  6      att_status
  7  FROM
  8      Attendance
  9  WHERE
 10      staff_id = 'SF0002';

Staff  CHECKIN_TIME        CHECKOUT_TIME       ATT_DATE            ATT_STATUS
------ ------------------- ------------------- ------------------- --------------------
SF0002 2025-02-14 10:12:00 2025-02-14 17:26:00 2025-02-14 16:15:50 Present
SF0002 2025-02-15 09:24:00 2025-02-15 16:01:00 2025-02-15 11:08:16 Leave
SF0002 2025-02-01 08:03:00 2025-02-01 15:47:00 2025-02-01 21:14:03 Leave
SF0002 2025-01-22 10:49:00 2025-01-22 18:24:00 2025-01-22 03:44:51 Present
SF0002 2025-04-06 08:42:00 2025-04-06 15:22:00 2025-04-06 17:59:58 Present
SF0002 2025-02-23 10:05:00 2025-02-23 17:59:00 2025-02-23 23:35:29 Leave
SF0002 2025-03-16 09:19:00 2025-03-16 15:08:00 2025-03-16 05:55:34 Present
SF0002 2025-04-25 08:00:00 2025-04-25 17:00:00 2025-04-25 00:00:00 Present

8 rows selected.
```

The last row is now marked as present.

### 4.5.5 Trigger 1: Auto Update Task_Progress to ‘In-Progress’ on New Record

**Purpose:**
**The purpose of this trigger is to automatically set the status of a task in the Task_Progress table to "In Progress" and record the current timestamp when a new task progress record is inserted. This ensures that when a task is being worked on, its status is immediately updated to reflect that it is in progress, improving tracking and visibility of task statuses.**

**SQL Statement:**

```
CREATE OR REPLACE TRIGGER trg_task_progress_status
BEFORE INSERT ON Task_Progress
FOR EACH ROW
BEGIN
  :NEW.tp_status := 'In Progress';
  :NEW.tp_datetime := SYSDATE;
END;
/
```

### 4.5.6 Trigger 2: Update attendance status when checkout is null (still at work)

**Purpose:**
**The purpose of this trigger is to automatically update the attendance status based on the checkout time. If the checkout time is null (indicating that the staff member has not yet completed their workday), the attendance status is set to "In Progress." If the checkout time is provided (indicating the staff member has finished for the day), the status is updated to "Present." This trigger helps maintain accurate attendance records and provides a real-time status of employees’ attendance.**

**SQL Statement:**

```
SELECT
 att_id,
 staff_id,
 TO_CHAR(checkin_datetime, 'YYYY-MM-DD HH24:MI:SS') AS checkin_time,
 TO_CHAR(checkout_datetime, 'YYYY-MM-DD HH24:MI:SS') AS checkout_time,
 TO_CHAR(att_date, 'YYYY-MM-DD HH24:MI:SS') AS att_date,
 att_status
FROM
```

```
 Attendance
WHERE
 staff_id = 'SF0003';

CREATE OR REPLACE TRIGGER trg_attendance_status
BEFORE INSERT OR UPDATE ON Attendance
FOR EACH ROW
BEGIN
  IF :NEW.checkout_datetime IS NULL THEN
:NEW.att_status := 'In Progress';
  ELSE
:NEW.att_status := 'Present';
  END IF;
END;
/

UPDATE Attendance
SET checkout_datetime = TO_DATE('2025-05-04 17:00:00', 'YYYY-MM-DD
HH24:MI:SS')
WHERE staff_id = 'SF0003' AND att_date = TO_DATE('2025-05-04',
'YYYY-MM-DD');
```

```
SQL> SELECT
  2      att_id,
  3      staff_id,
  4      TO_CHAR(checkin_datetime, 'YYYY-MM-DD HH24:MI:SS') AS checkin_time,
  5      TO_CHAR(checkout_datetime, 'YYYY-MM-DD HH24:MI:SS') AS checkout_time,
  6      TO_CHAR(att_date, 'YYYY-MM-DD HH24:MI:SS') AS att_date,
  7      att_status
  8  FROM
  9      Attendance
 10  WHERE
 11      staff_id = 'SF0003';

ATT_ID Staff  CHECKIN_TIME        CHECKOUT_TIME       ATT_DATE            ATT_STATUS
------ ------ ------------------- ------------------- ------------------- --------------------
AT0037 SF0003 2025-02-07 09:30:00 2025-02-07 17:15:00 2025-02-07 10:44:47 Absent
AT0059 SF0003 2025-04-08 10:10:00 2025-04-08 19:01:00 2025-04-08 20:24:17 Absent
AT0075 SF0003 2025-03-20 10:13:00 2025-03-20 17:18:00 2025-03-20 16:08:33 Absent
```

```
SQL> INSERT INTO Attendance (
  2      staff_id,
  3      checkin_datetime,
  4      checkout_datetime,
  5      att_date
  6  ) VALUES (
  7      'SF0003',
  8      TO_DATE('2025-05-04 09:00:00', 'YYYY-MM-DD HH24:MI:SS'),
  9      NULL,
 10      TO_DATE('2025-05-04', 'YYYY-MM-DD')
 11  );

1 row created.

SQL> SELECT
  2      att_id,
  3      staff_id,
  4      TO_CHAR(checkin_datetime, 'YYYY-MM-DD HH24:MI:SS') AS checkin_time,
  5      TO_CHAR(checkout_datetime, 'YYYY-MM-DD HH24:MI:SS') AS checkout_time,
  6      TO_CHAR(att_date, 'YYYY-MM-DD HH24:MI:SS') AS att_date,
  7      att_status
  8  FROM
  9      Attendance
 10  WHERE
 11      staff_id = 'SF0003';

ATT_ID Staff  CHECKIN_TIME        CHECKOUT_TIME       ATT_DATE            ATT_STATUS
------ ------ ------------------- ------------------- ------------------- --------------------
AT0037 SF0003 2025-02-07 09:30:00 2025-02-07 17:15:00 2025-02-07 10:44:47 Absent
AT0059 SF0003 2025-04-08 10:10:00 2025-04-08 19:01:00 2025-04-08 20:24:17 Absent
AT0075 SF0003 2025-03-20 10:13:00 2025-03-20 17:18:00 2025-03-20 16:08:33 Absent
AT0102 SF0003 2025-05-04 09:00:00                     2025-05-04 00:00:00 In Progress
```

SF0003 takes attendance so the att_status will be marked as ‘In Progress’ since the checkout time is not known yet (null).

```
SQL> UPDATE Attendance
  2  SET checkout_datetime = TO_DATE('2025-05-04 17:00:00', 'YYYY-MM-DD HH24:MI:SS')
  3  WHERE staff_id = 'SF0003' AND att_date = TO_DATE('2025-05-04', 'YYYY-MM-DD');

1 row updated.

SQL> SELECT
  2   att_id,
  3   staff_id,
  4   TO_CHAR(checkin_datetime, 'YYYY-MM-DD HH24:MI:SS') AS checkin_time,
  5   TO_CHAR(checkout_datetime, 'YYYY-MM-DD HH24:MI:SS') AS checkout_time,
  6   TO_CHAR(att_date, 'YYYY-MM-DD HH24:MI:SS') AS att_date,
  7   att_status
  8  FROM
  9   Attendance
 10  WHERE
 11   staff_id = 'SF0003';

ATT_ID Staff  CHECKIN_TIME       CHECKOUT_TIME      ATT_DATE           ATT_STATUS
------ ------ ------------------ ------------------ ------------------ --------------------
AT0037 SF0003 2025-02-07 09:30:00 2025-02-07 17:15:00 2025-02-07 10:44:47 Absent
AT0059 SF0003 2025-04-08 10:10:00 2025-04-08 19:01:00 2025-04-08 20:24:17 Absent
AT0075 SF0003 2025-03-20 10:13:00 2025-03-20 17:18:00 2025-03-20 16:08:33 Absent
AT0102 SF0003 2025-05-04 09:00:00 2025-05-04 17:00:00 2025-05-04 00:00:00 Present
```

Then, when SF0003 has completed his working hours (check out), the att_status will be updated to “Present” since SF0003 has checked out.

### 4.5.7 Report 1: Staff Daily Attendance Summary Report

**Purpose:**
**This report provides a day-by-day summary of staff attendance statuses (Present, Absent, Leave) within a specified date range. It helps management quickly assess daily workforce availability, detect absenteeism trends, and plan staffing accordingly. This is especially useful for operations, HR teams, and supervisors who need to ensure adequate staffing levels for daily operations.**

**SQL Statement:**

```
CREATE OR REPLACE PROCEDURE rpt_Daily_Attendance_Summary (
p_start_date IN DATE,
p_end_date   IN DATE
) AS
CURSOR date_cursor IS
    SELECT TRUNC(p_start_date + LEVEL - 1) AS report_date
    FROM dual
    CONNECT BY LEVEL <= (p_end_date - p_start_date + 1);

v_date DATE;
v_present NUMBER;
v_absent NUMBER;
v_leave NUMBER;
BEGIN
IF p_end_date < p_start_date THEN
    DBMS_OUTPUT.PUT_LINE('Error: End date must be on or after start
date.');
    RETURN;
END IF;

DBMS_OUTPUT.PUT_LINE(CHR(10));
DBMS_OUTPUT.PUT_LINE(RPAD('=', 50, '='));
DBMS_OUTPUT.PUT_LINE('| Staff Daily Attendance Summary Report |');
DBMS_OUTPUT.PUT_LINE(RPAD('=', 50, '='));

FOR d IN date_cursor LOOP
    v_date := d.report_date;

    SELECT COUNT(*) INTO v_present FROM Attendance
    WHERE att_date = v_date AND att_status = 'Present';
```

```
    SELECT COUNT(*) INTO v_absent FROM Attendance
    WHERE att_date = v_date AND att_status = 'Absent';

    SELECT COUNT(*) INTO v_leave FROM Attendance
    WHERE att_date = v_date AND att_status = 'Leave';

    DBMS_OUTPUT.PUT_LINE('Date: ' || TO_CHAR(v_date, 'DD-MON-YYYY'));
    DBMS_OUTPUT.PUT_LINE(' - Present: ' || v_present);
    DBMS_OUTPUT.PUT_LINE(' - Absent : ' || v_absent);
    DBMS_OUTPUT.PUT_LINE(' - Leave  : ' || v_leave);
    DBMS_OUTPUT.PUT_LINE(RPAD('-', 50, '-'));
END LOOP;
DBMS_OUTPUT.PUT_LINE(RPAD('=', 50, '='));
EXCEPTION
WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error in attendance summary: ' || SQLERRM);
END;
/
```

**Sample Output:**

```
============================================================
| Staff Daily Attendance Summary Report |
============================================================
Date: 20-MAR-2025
 - Present: 0
 - Absent : 1
 - Leave  : 1
------------------------------------------------------------
Date: 21-MAR-2025
 - Present: 0
 - Absent : 1
 - Leave  : 0
------------------------------------------------------------
Date: 22-MAR-2025
 - Present: 1
 - Absent : 0
 - Leave  : 0
------------------------------------------------------------
Date: 23-MAR-2025
 - Present: 0
 - Absent : 0
 - Leave  : 0
------------------------------------------------------------
Date: 24-MAR-2025
 - Present: 0
 - Absent : 0
 - Leave  : 1
------------------------------------------------------------
Date: 25-MAR-2025
 - Present: 0
 - Absent : 0
 - Leave  : 0
------------------------------------------------------------
Date: 26-MAR-2025
 - Present: 0
 - Absent : 0
 - Leave  : 0
------------------------------------------------------------
Date: 27-MAR-2025
 - Present: 0
 - Absent : 0
 - Leave  : 0
------------------------------------------------------------
Date: 28-MAR-2025
 - Present: 0
 - Absent : 1
 - Leave  : 0
------------------------------------------------------------
Date: 29-MAR-2025
 - Present: 0
 - Absent : 0
 - Leave  : 1
------------------------------------------------------------
Date: 30-MAR-2025
 - Present: 0
 - Absent : 0
 - Leave  : 1
------------------------------------------------------------
Date: 31-MAR-2025
 - Present: 1
 - Absent : 0
 - Leave  : 1
------------------------------------------------------------
============================================================
```

### 4.5.8 Report 2: Staff Task Assignment Overview Report

**Purpose:**
**This report gives a summary of how many tasks were assigned to each staff member during a specified period. It is useful for analyzing workload distribution, identifying under- or over-utilized staff, and supporting decisions related to resource planning, task reallocation, or performance evaluation based on task load.**

**SQL Statement:**

```
CREATE OR REPLACE PROCEDURE rpt_Staff_Task_Overview (
p_start_date IN DATE,
p_end_date   IN DATE
) AS
CURSOR staff_cursor IS
    SELECT staff_id, name FROM Staff;

v_staff_id   Staff.staff_id%TYPE;
v_name   Staff.name%TYPE;
v_task_count NUMBER;
BEGIN
IF p_end_date < p_start_date THEN
    DBMS_OUTPUT.PUT_LINE('Error: End date must be on or after start
date.');
    RETURN;
END IF;

DBMS_OUTPUT.PUT_LINE(CHR(10));
DBMS_OUTPUT.PUT_LINE(RPAD('=', 60, '='));
DBMS_OUTPUT.PUT_LINE('| Staff Task Assignment Overview Report |');
DBMS_OUTPUT.PUT_LINE(RPAD('=', 60, '='));
DBMS_OUTPUT.PUT_LINE('Period: ' || TO_CHAR(p_start_date,
'DD-MON-YYYY') || ' to ' || TO_CHAR(p_end_date, 'DD-MON-YYYY'));
DBMS_OUTPUT.PUT_LINE(RPAD('-', 60, '-'));

FOR s IN staff_cursor LOOP
    SELECT COUNT(*) INTO v_task_count
    FROM Schedule
    WHERE staff_id = s.staff_id
      AND start_datetime BETWEEN TRUNC(p_start_date) AND
TRUNC(p_end_date) + 1;

    DBMS_OUTPUT.PUT_LINE('Staff ID   : ' || s.staff_id);
    DBMS_OUTPUT.PUT_LINE('Name   : ' || s.name);
    DBMS_OUTPUT.PUT_LINE('Tasks Assigned: ' || v_task_count);
    DBMS_OUTPUT.PUT_LINE(RPAD('-', 60, '-'));
END LOOP;

DBMS_OUTPUT.PUT_LINE(RPAD('=', 60, '='));
EXCEPTION
WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error generating task overview: ' ||
SQLERRM);
END;
/
```

**Sample Output:**

```
================================================================
| Staff Task Assignment Overview Report |
================================================================
Period: 01-FEB-2025 to 28-FEB-2025
----------------------------------------------------------------
Staff ID   : SF0001
Name       : John Doe
Tasks Assigned: 2
----------------------------------------------------------------
Staff ID   : SF0002
Name       : Jane Smith
Tasks Assigned: 4
----------------------------------------------------------------
Staff ID   : SF0003
Name       : Ahmad Bakar
Tasks Assigned: 2
----------------------------------------------------------------
Staff ID   : SF0004
Name       : Siti Aminah
Tasks Assigned: 0
----------------------------------------------------------------
Staff ID   : SF0005
Name       : Alice Brown
Tasks Assigned: 0
----------------------------------------------------------------
Staff ID   : SF0006
Name       : Bob White
Tasks Assigned: 0
----------------------------------------------------------------
Staff ID   : SF0007
Name       : Charlie Green
Tasks Assigned: 0
----------------------------------------------------------------
Staff ID   : SF0008
Name       : Daisy Black
Tasks Assigned: 1
----------------------------------------------------------------
Staff ID   : SF0009
Name       : Edward Blue
Tasks Assigned: 0
----------------------------------------------------------------
Staff ID   : SF0010
Name       : Fiona Red
Tasks Assigned: 1
----------------------------------------------------------------
Staff ID   : SF0011
Name       : George Yellow
Tasks Assigned: 1
----------------------------------------------------------------
Staff ID   : SF0012
Name       : Hannah Purple
Tasks Assigned: 2
----------------------------------------------------------------
Staff ID   : SF0013
Name       : Ian Grey
Tasks Assigned: 2
----------------------------------------------------------------
Staff ID   : SF0014
Name       : Julia Azure
Tasks Assigned: 0
----------------------------------------------------------------
Staff ID   : SF0015
Name       : Kevin Teal
Tasks Assigned: 2
----------------------------------------------------------------
================================================================
```

# Chapter 5 Extra Effort Highlights

## 5.1 Chia Ming Yi

### 5.1.1 Views

**View 1: Member Fine Details**

**Purpose: The purpose of this view is to provide the status of books borrowed by members, including the member's status, book status, availability, and borrow details. It filters for books that are either borrowed or missing and are unavailable, ensuring only active books are considered for loan extensions or other processes.**

**View code:**

```
CREATE OR REPLACE VIEW vw_member_fine_details AS
SELECT
    br.member_id,
    f.fine_id,
    b.book_id,
    b.name AS book_name,
    f.fine_amt,
    f.credit_deduct,
    f.fine_status
FROM
    Fine_Record f
    JOIN Borrow_Detail bd ON f.borrow_detail_id = bd.borrow_detail_id
    JOIN Borrow br ON bd.borrow_id = br.borrow_id
    JOIN Book b ON bd.book_id = b.book_id;
```

**Sample Output:**

```
SQL> select * from vw_member_fine_details;

MEMBER FINE_I BOOK_I BOOK_NAME                                          FINE_AMT CREDIT_DEDUCT FINE_S
------ ------ ------ -------------------------------------------------- ---------- ------------- ------
MB0070 FN0001 BK0075 Personal administration.                               12            16 Unpaid
MB0068 FN0002 BK0003 Role network even.                                     12            16 Paid
MB0094 FN0003 BK0087 North once PM admit.                                 10.5            14 Paid
MB0100 FN0004 BK0082 Society.                                             13.5            18 Unpaid
MB0028 FN0005 BK0072 Minute reach.                                           3             4 Unpaid
MB0056 FN0006 BK0058 Child bag such own.                                     3             4 Paid
MB0038 FN0007 BK0024 Quickly yard.                                           9            12 Paid
MB0024 FN0008 BK0042 Much.                                                 7.5            10 Unpaid
MB0044 FN0009 BK0079 Lose some yard.                                         6             8 Paid
MB0013 FN0010 BK0066 Agreement finish scene.                                12            16 Paid
MB0067 FN0011 BK0051 Give return.                                          1.5             2 Paid

MEMBER FINE_I BOOK_I BOOK_NAME                                          FINE_AMT CREDIT_DEDUCT FINE_S
------ ------ ------ -------------------------------------------------- ---------- ------------- ------
MB0054 FN0012 BK0015 Though anyone.                                          3             4 Unpaid
MB0065 FN0013 BK0072 Minute reach.                                           3             4 Paid
MB0070 FN0014 BK0075 Personal administration.                                9            12 Unpaid
MB0082 FN0015 BK0042 Much.                                                  12            16 Paid
MB0060 FN0016 BK0002 The.                                                  4.5             6 Unpaid
MB0013 FN0017 BK0042 Much.                                                 4.5             6 Paid
MB0013 FN0018 BK0042 Much.                                                10.5            14 Unpaid
MB0100 FN0019 BK0082 Society.                                              1.5             2 Paid
MB0064 FN0020 BK0019 Answer much.                                           12            16 Paid
MB0098 FN0021 BK0055 Speech able degree.                                    12            16 Unpaid
MB0046 FN0022 BK0090 Seven size.                                           7.5            10 Unpaid
```

**View 2: View of Overdue and Missing Borrowed Books**

**Purpose: The purpose of this view is to provide a list of members who have missing books along with contact details, book names, and the number of days overdue. It helps staff monitor and follow up on overdue borrowings efficiently.**

**View code:**

```
CREATE OR REPLACE VIEW vw_overdue_borrowed_books AS
SELECT
    m.member_id,
    m.name AS member_name,
    m.tel_no,
    bk.name AS book_name,
    b.return_datetime,
    TRUNC(SYSDATE) - TRUNC(b.return_datetime) AS overdue_days
FROM member m
JOIN borrow b ON m.member_id = b.member_id
JOIN borrow_detail bd ON b.borrow_id = bd.borrow_id
JOIN book bk ON bd.book_id = bk.book_id
WHERE bd.borrow_status = 'Missing';
```

**Sample Output:**

```
SQL> select * from vw_member_fine_details;

MEMBER FINE_I BOOK_I BOOK_NAME                                          FINE_AMT CREDIT_DEDUCT FINE_S
------ ------ ------ -------------------------------------------------- ---------- ------------- ------
MB0070 FN0001 BK0075 Personal administration.                               12            16 Unpaid
MB0068 FN0002 BK0003 Role network even.                                     12            16 Paid
MB0094 FN0003 BK0087 North once PM admit.                                 10.5            14 Paid
MB0100 FN0004 BK0082 Society.                                             13.5            18 Unpaid
MB0028 FN0005 BK0072 Minute reach.                                           3             4 Unpaid
MB0056 FN0006 BK0058 Child bag such own.                                     3             4 Paid
MB0038 FN0007 BK0024 Quickly yard.                                           9            12 Paid
MB0024 FN0008 BK0042 Much.                                                 7.5            10 Unpaid
MB0044 FN0009 BK0079 Lose some yard.                                         6             8 Paid
MB0013 FN0010 BK0066 Agreement finish scene.                                12            16 Paid
MB0067 FN0011 BK0051 Give return.                                          1.5             2 Paid

MEMBER FINE_I BOOK_I BOOK_NAME                                          FINE_AMT CREDIT_DEDUCT FINE_S
------ ------ ------ -------------------------------------------------- ---------- ------------- ------
MB0054 FN0012 BK0015 Though anyone.                                          3             4 Unpaid
MB0065 FN0013 BK0072 Minute reach.                                           3             4 Paid
MB0070 FN0014 BK0075 Personal administration.                                9            12 Unpaid
MB0082 FN0015 BK0042 Much.                                                  12            16 Paid
MB0060 FN0016 BK0002 The.                                                  4.5             6 Unpaid
MB0013 FN0017 BK0042 Much.                                                 4.5             6 Paid
MB0013 FN0018 BK0042 Much.                                                10.5            14 Unpaid
MB0100 FN0019 BK0082 Society.                                              1.5             2 Paid
MB0064 FN0020 BK0019 Answer much.                                           12            16 Paid
MB0098 FN0021 BK0055 Speech able degree.                                    12            16 Unpaid
MB0046 FN0022 BK0090 Seven size.                                           7.5            10 Unpaid

MEMBER FINE_I BOOK_I BOOK_NAME                                          FINE_AMT CREDIT_DEDUCT FINE_S
------ ------ ------ -------------------------------------------------- ---------- ------------- ------
MB0056 FN0023 BK0058 Child bag such own.                                  10.5            14 Unpaid
MB0064 FN0024 BK0019 Answer much.                                         13.5            18 Unpaid
MB0086 FN0025 BK0095 Likely.                                              13.5            18 Paid
MB0015 FN0026 BK0095 Likely.                                                 6             8 Paid
MB0010 FN0027 BK0046 Experience moment.                                    7.5            10 Unpaid
MB0064 FN0028 BK0009 Fact gas suddenly.                                      6             8 Paid
MB0021 FN0029 BK0080 Enough answer share mean.                               6             8 Paid
MB0048 FN0030 BK0032 Generation society.                                    12            16 Unpaid
MB0020 FN0031 BK0004 Recently material social.                             1.5             2 Unpaid
MB0078 FN0032 BK0051 Give return.                                         13.5            18 Paid
MB0053 FN0033 BK0021 Enough.                                                 6             8 Paid
```

## 5.1.2 Indexes

**Index 1: Index on Fine Status in Fine_Record**

**Purpose: The purpose of this index is to optimize queries that filter fine records based on their fine_status, such as retrieving only "Unpaid" or "Paid" fines. Especially useful in WHERE clauses with conditional filtering in reports or procedures.**

**Index code:**

```
CREATE INDEX idx_fine_record_status ON Fine_Record(fine_status);
```

**Index 2: Composite Index on Borrow and Detail IDs in Borrow_Detail**

**Purpose: The purpose of this index is to improve performance of joins between Borrow and Borrow_Detail, and to speed up lookups involving both borrow_id and borrow_detail_id. This is commonly used in cascaded relationships for tracking individual borrow transactions.**

**Index code:**

```
CREATE INDEX idx_borrow_detail_borrow_id ON Borrow_Detail(borrow_id,
borrow_detail_id);
```

**Index 3: Index on Member ID in Borrow**

**Purpose: The purpose of this index is to accelerate queries that search for all borrowed records related to a specific member. This is crucial when joining Borrow with Member, Borrow_Detail, or Fine_Record to generate member-specific histories or reports.**

**Index code:**

```
CREATE INDEX idx_borrow_member_id ON Borrow(member_id);
```

### 5.1.3 User Defined Functions

**Function 1: Calculate Time Remaining Until a Specific Date**

**Purpose: The purpose of this function is to compute the remaining time from the current system date to a given return date (p_return_datetime), returning the result as a formatted string in days, hours, and minutes. It helps track deadlines or time left for due events in a human-readable format.**

**Function code:**

```
CREATE OR REPLACE FUNCTION fn_remaining_time(
    p_return_datetime IN DATE
) RETURN VARCHAR2 IS
    v_days NUMBER;
    v_hours NUMBER;
    v_minutes NUMBER;
    v_remaining_time VARCHAR2(50);
BEGIN
    -- Calculate the difference between the return date and current
date
    v_days := FLOOR(p_return_datetime - SYSDATE);
    v_hours := FLOOR((p_return_datetime - SYSDATE - v_days) * 24);
    v_minutes := FLOOR(((p_return_datetime - SYSDATE - v_days) * 24 -
v_hours) * 60);

    -- Format the remaining time
    v_remaining_time := v_days || ' days ' || v_hours || ' hours ' ||
v_minutes || ' minutes';
    RETURN v_remaining_time;
END;
/
```

### 5.1.4 Sequence & Trigger

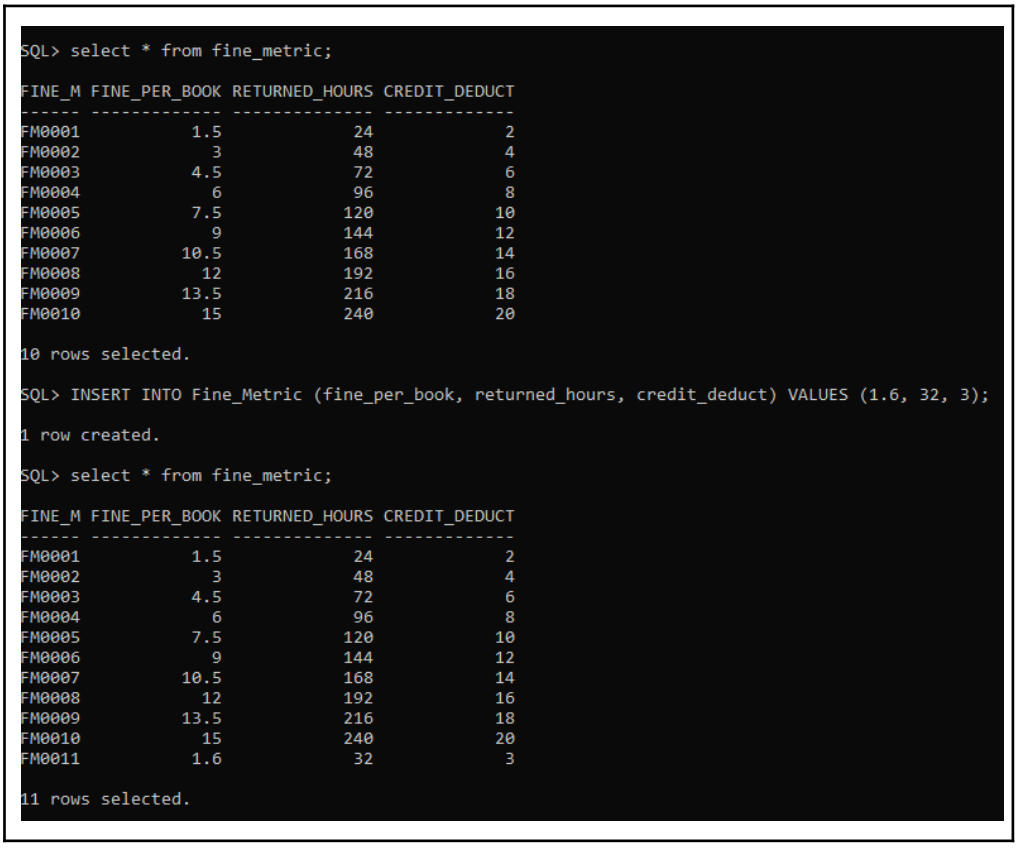
**Procedure 1: Auto-Increment Fine Metric ID**

**Purpose: The purpose of this sequence and trigger is to automatically generate a unique fine_metric_id for each record inserted into the Fine_Metric table. The trigger ensures that if the fine_metric_id is not provided, it will be automatically populated with a value based on the sequence.**

**Sequence & Trigger code:**

```
CREATE SEQUENCE fine_metric_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

-- Auto-Increment for fine_metric_id
CREATE OR REPLACE TRIGGER fine_metric_before_insert
       BEFORE INSERT ON Fine_Metric
       FOR EACH ROW
       BEGIN
              IF :NEW.fine_metric_id IS NULL THEN
                     SELECT 'FM' || LPAD(fine_metric_seq.NEXTVAL, 4,
                     '0') INTO :NEW.fine_metric_id FROM dual;
              END IF;
       END;
/
```

**Sample Output:**



```
SQL> select * from fine_metric;

FINE_M FINE_PER_BOOK RETURNED_HOURS CREDIT_DEDUCT
------ ------------- -------------- -------------
FM0001           1.5             24             2
FM0002             3             48             4
FM0003           4.5             72             6
FM0004             6             96             8
FM0005           7.5            120            10
FM0006             9            144            12
FM0007          10.5            168            14
FM0008            12            192            16
FM0009          13.5            216            18
FM0010            15            240            20

10 rows selected.

SQL> INSERT INTO Fine_Metric (fine_per_book, returned_hours, credit_deduct) VALUES (1.6, 32, 3);

1 row created.

SQL> select * from fine_metric;

FINE_M FINE_PER_BOOK RETURNED_HOURS CREDIT_DEDUCT
------ ------------- -------------- -------------
FM0001           1.5             24             2
FM0002             3             48             4
FM0003           4.5             72             6
FM0004             6             96             8
FM0005           7.5            120            10
FM0006             9            144            12
FM0007          10.5            168            14
FM0008            12            192            16
FM0009          13.5            216            18
FM0010            15            240            20
FM0011           1.6             32             3

11 rows selected.
```

## 5.1.5 User Defined Exception

**Exception 1: Generic Error Handling with Rollback**

**Purpose: The purpose of this exception is to ensures that if an error occurs during the insert or update operations, the changes are rolled back to the**

**before_insert_update savepoint to maintain data consistency. It also provides user-friendly error feedback through DBMS_OUTPUT.**

**Exception code:**

```
       BEGIN
        SAVEPOINT before_insert_update;
        -- insert loan extension
        INSERT INTO Loan_Extension (borrow_id, days_extended,
le_datetime)
        VALUES (v_formatted_borrow_id, p_day_extend, SYSDATE)
        RETURNING extension_id INTO v_formatted_extension_id;

        -- update borrow record
        UPDATE Borrow
        SET return_datetime = return_datetime + p_day_extend
        WHERE borrow_id = v_formatted_borrow_id;
    EXCEPTION
        WHEN OTHERS THEN
            ROLLBACK TO before_insert_update;
            DBMS_OUTPUT.PUT_LINE('Error occurred: ' || SQLERRM);
            RETURN;
    END;
```

## 5.2 Lim Jun Wei

### 5.2.1 Views

**Views 1: Eligible membership views**

**Purpose: This view serves to simplify and centralize the logic for determining eligible membership levels. Instead of repeatedly writing complex join and filter conditions in various parts of your application or database procedures, you can query this view to obtain the necessary information.**

**View code:**

```
CREATE OR REPLACE VIEW vw_eligible_membership AS
SELECT
    m.member_id,
    m.credit AS member_credit,
    TRUNC(SYSDATE - m.reg_date) AS member_duration,
    ms.membership_id,
    ms.membership_level,
    ms.credit AS required_credit,
    ms.duration_day AS required_duration
FROM
    Member m
JOIN
    Membership ms
ON
    m.credit >= ms.credit
AND
    TRUNC(SYSDATE - m.reg_date) >= ms.duration_day;
/
```

| Input | SELECT * FROM vw_eligible_membership; |
|---|---|
| Output | |

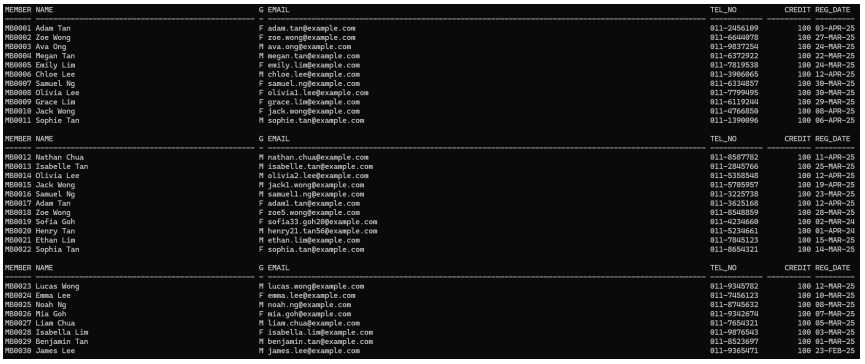
```
MEMBER MEMBER_CREDIT MEMBER_DURATION MEMBER MEMBERSHIP_LEVEL REQUIRED_CREDIT REQUIRED_DURATION
------ ------------- --------------- ------ ---------------- --------------- -----------------
MB0089           100             516 MS0001                0               0                 0
MB0090           100             439 MS0001                0               0                 0
MB0091           100             689 MS0001                0               0                 0
MB0092           100             911 MS0001                0               0                 0
MB0093           100             876 MS0001                0               0                 0
MB0094           100             747 MS0001                0               0                 0
MB0095           100             418 MS0001                0               0                 0
MB0096           100             819 MS0001                0               0                 0
MB0097           100             784 MS0001                0               0                 0
MB0098           100             747 MS0001                0               0                 0
MB0099           100             710 MS0001                0               0                 0

MEMBER MEMBER_CREDIT MEMBER_DURATION MEMBER MEMBERSHIP_LEVEL REQUIRED_CREDIT REQUIRED_DURATION
------ ------------- --------------- ------ ---------------- --------------- -----------------
MB0100           100             674 MS0001                0               0                 0
MB0101           100               0 MS0001                0               0                 0
MB0102           100               0 MS0001                0               0                 0
MB0001           100              31 MS0002                1              60                 0
MB0002           100              38 MS0002                1              60                 0
MB0003           100              41 MS0002                1              60                 0
MB0004           100              43 MS0002                1              60                 0
MB0005           100              41 MS0002                1              60                 0
MB0006           100              22 MS0002                1              60                 0
MB0007           100              35 MS0002                1              60                 0
MB0008           100              35 MS0002                1              60                 0

MEMBER MEMBER_CREDIT MEMBER_DURATION MEMBER MEMBERSHIP_LEVEL REQUIRED_CREDIT REQUIRED_DURATION
------ ------------- --------------- ------ ---------------- --------------- -----------------
MB0009           100              36 MS0002                1              60                 0
MB0010           100              26 MS0002                1              60                 0
MB0011           100              28 MS0002                1              60                 0
MB0012           100              23 MS0002                1              60                 0
MB0013           100              40 MS0002                1              60                 0
MB0014           100              22 MS0002                1              60                 0
MB0015           100              15 MS0002                1              60                 0
MB0016           100              42 MS0002                1              60                 0
MB0017           100              22 MS0002                1              60                 0
MB0018           100              37 MS0002                1              60                 0
MB0019           100             428 MS0002                1              60                 0
```

**Views 2: Member detail views**

**Purpose: This view serves to simplify data retrieval by combining relevant information from the Member and Membership tables. Instead of performing complex joins each time this combined data is needed, the view offers a streamlined and reusable way to access it.**

**View code:**

```
CREATE OR REPLACE VIEW vw_member_details AS
SELECT
    mb.member_id,
    mb.name,
    mb.gender,
    mb.email,
    mb.tel_no,
    mb.credit,
    mb.reg_date,
    mb.mb_status,
    ms.membership_level,
    ms.max_borrow_day,
    ms.max_borrow_book,
    ms.max_loan_extension_day
FROM Member mb
JOIN Membership ms ON mb.membership_id = ms.membership_id;
```

| | |
|---|---|
| Input | `SELECT * FROM vw_member_details;` |
| Output |  |



## 5.2.2 Indexes

**Indexes 1: Index on Member(credit, reg_date)**

**Purpose: This composite index is tailored for queries that filter or sort members based on their credit and registration date. By indexing both credit and reg_date, the database can efficiently navigate to the relevant records, enhancing query performance based on credit and reg_date criteria.**

**Index code:**

```
CREATE INDEX idx_member_credit_regdate ON Member(credit, reg_date);
```

**Indexes 2: Index on Member(member_id, membership_id)**

**Purpose: This composite index is beneficial for operations that involve both the member_id and membership_id columns. The index facilitates rapid identification of the specific member record and ensures that the update is only performed if the membership level has changed. By indexing both columns, the database can swiftly locate and verify the relevant record, optimizing the update operation.**

**Index code:**

```
CREATE INDEX idx_member_id_membership ON Member(member_id,
membership_id);
```

**Indexes 3: Index on Membership(credit DESC, duration_day DESC)**

**Purpose: This descending composite index is designed for queries that aim to find the most suitable membership level based on credit and duration requirements.**

**Index code:**

```
CREATE INDEX idx_membership_credit_duration
ON Membership(credit DESC, duration_day DESC);
```

**Indexes 4: Index on Member(tel_no)**

**Purpose: the idx_member_tel_no index would allow the database to quickly locate any existing records with the same telephone number, thereby improving the efficiency of the uniqueness check.**

**Index code:**

```
CREATE INDEX idx_member_tel_no ON Member(tel_no);
```

### 5.2.3 User Defined Functions

**Procedure 1: Check availability of book for borrowing**

**Purpose: The purpose of this procedure is to assist the book borrowing process via identifying the current availability and status of books. It is crucial for avoiding the books which have not been returned back or are currently inactive from being borrowed. Meanwhile, it also helps to identify which member is currently borrowing the book and check whether the person is the one who is currently applying for borrowing the books. This can effectively prevent the redundant borrowing from the same person.**

**Procedure code:**

```
CREATE OR REPLACE PROCEDURE prc_check_book_availability (
    p_book_id               IN  book.book_id%TYPE,
    p_book_available        OUT NUMBER,
    p_member_id_borrowing   OUT member.member_id%TYPE
) AS
    v_status    book.bk_status%TYPE;
    v_availability  book.availability%TYPE;
BEGIN
    BEGIN
        -- obtain status, availability from book table
        SELECT bk_status, availability
        INTO v_status, v_availability
        FROM Book
        WHERE book_id = p_book_id;

        -- check if the book is available
        IF v_status = 'Inactive' OR v_availability = 'N' THEN
```

```
            p_book_available := 0;
            DBMS_OUTPUT.PUT_LINE('Error: The book ' || p_book_id || '
is not available currently.');

            BEGIN
                -- obtain member_id who is borrowing the book
                SELECT b.member_id
                INTO p_member_id_borrowing
                FROM Borrow b
                JOIN Borrow_Detail bd
                    ON b.borrow_id = bd.borrow_id
                WHERE bd.book_id = p_book_id
                    AND bd.borrow_status = 'Borrowed'
                ORDER BY b.return_datetime DESC
                FETCH FIRST 1 ROWS ONLY;
            EXCEPTION
                WHEN NO_DATA_FOUND THEN
                    p_member_id_borrowing := NULL;
            END;
            RETURN;
        END IF;

        p_book_available := 1;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Error: No data found for book_id: '
|| p_book_id);
            p_book_available := 0;
            p_member_id_borrowing := NULL;
    END;
END;
/
```

Scenario 1: Check book BK0044 which is currently not available and inactive.

| | |
|---|---|
| Input | `DECLARE`<br>`  v_book_id  Book.book_id%TYPE;`<br>`  v_book_available  Book.availability%TYPE;`<br>`  v_member_id_borrowing  Member.member_id%TYPE;`<br>`BEGIN`<br>`  v_book_id := 'BK0044';`<br>`  prc_check_book_availability(`<br>`  p_book_id => v_book_id,`<br>`  p_book_available => v_book_available,`<br>`  p_member_id_borrowing => v_member_id_borrowing`<br>`  );`<br>`END;`<br>`/` |
| Output | `Error: The book BK0044 is not available currently.`<br><br>`PL/SQL procedure successfully completed.` |

Scenario 2: Check book BK0178 which currently does not exist.

| | |
|---|---|
| Input | `DECLARE`<br>`  v_book_id  Book.book_id%TYPE;`<br>`  v_book_available  Book.availability%TYPE;`<br>`  v_member_id_borrowing  Member.member_id%TYPE;`<br>`BEGIN`<br>`  v_book_id := 'BK0178';`<br>`  prc_check_book_availability(` |

| | |
|---|---|
| | p_book_id => v_book_id,<br>p_book_available => v_book_available,<br>p_member_id_borrowing => v_member_id_borrowing<br>);<br>END;<br>/ |
| Output | Error: No data found for book_id: BK0178<br><br>PL/SQL procedure successfully completed. |

**Procedure 2: Check book reservation status**

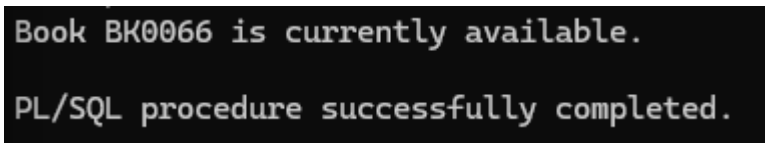
**Purpose: This procedure is mainly for identifying whether the book that member wants to borrow is currently reserved by other members.**

**Procedure code:**

```
CREATE OR REPLACE PROCEDURE prc_check_book_reservation (
    p_book_id       IN  Book.book_id%TYPE,
    p_is_reserved   OUT NUMBER
) AS
    v_no_of_reservation NUMBER;
BEGIN
    SELECT COUNT(*)
    INTO v_no_of_reservation
    FROM Reservation
    WHERE book_id = p_book_id
        AND rsv_status IN ('Reserved', 'Ready');

    IF v_no_of_reservation > 0 THEN
        p_is_reserved := 1;
    ELSE
        p_is_reserved := 0;
    END IF;
END;
/
```

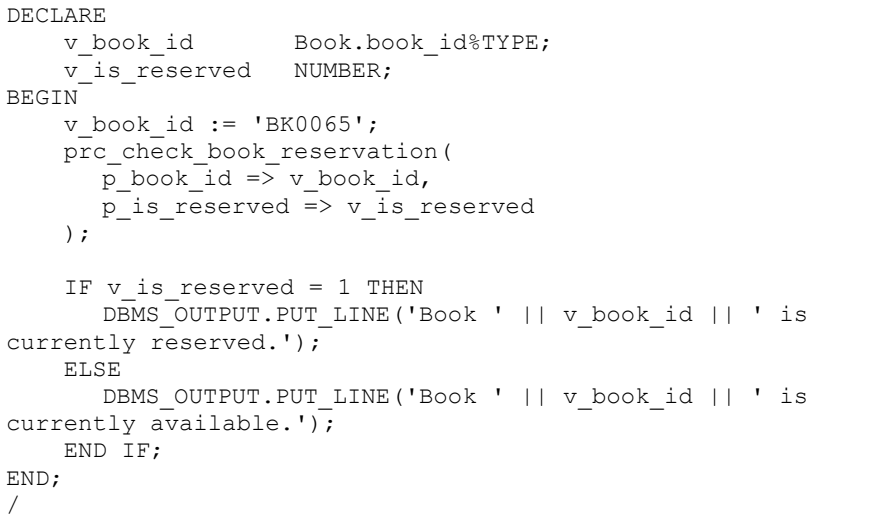
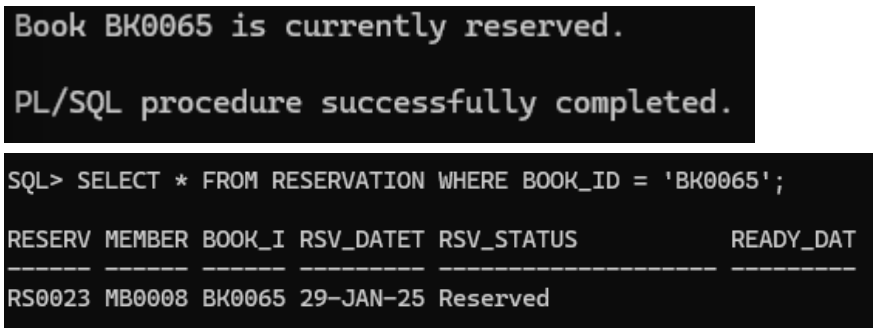
Scenario 1: Check reservation status of book BK0066 which is currently not reserved by other member.

| | |
|---|---|
| Input | DECLARE<br>v_book_id Book.book_id%TYPE;<br>v_is_reserved NUMBER;<br>BEGIN<br>v_book_id := 'BK0066';<br>prc_check_book_reservation(<br>p_book_id => v_book_id,<br>p_is_reserved => v_is_reserved<br>);<br><br>IF v_is_reserved = 1 THEN<br>DBMS_OUTPUT.PUT_LINE('Book ' \|\| v_book_id \|\| ' is currently reserved.');<br>ELSE<br>DBMS_OUTPUT.PUT_LINE('Book ' \|\| v_book_id \|\| ' is currently available.');<br>END IF;<br>END;<br>/ |

| | |
|---|---|
| Output |  |



Scenario 2: Check reservation status of book BK0065 which is currently reserved by another member.

| | |
|---|---|
| Input |  |
| Output |  |

```
DECLARE
    v_book_id       Book.book_id%TYPE;
    v_is_reserved   NUMBER;
BEGIN
    v_book_id := 'BK0065';
    prc_check_book_reservation(
       p_book_id => v_book_id,
       p_is_reserved => v_is_reserved
    );

    IF v_is_reserved = 1 THEN
       DBMS_OUTPUT.PUT_LINE('Book ' || v_book_id || ' is
currently reserved.');
    ELSE
       DBMS_OUTPUT.PUT_LINE('Book ' || v_book_id || ' is
currently available.');
    END IF;
END;
/
```

```
Book BK0065 is currently reserved.

PL/SQL procedure successfully completed.
```

```
SQL> SELECT * FROM RESERVATION WHERE BOOK_ID = 'BK0065';

RESERV MEMBER BOOK_I RSV_DATET RSV_STATUS           READY_DAT
------ ------ ------ --------- -------------------- ---------
RS0023 MB0008 BK0065 29-JAN-25 Reserved
```

**Procedure 3: Check user eligibility for borrowing book**

**Purpose: This procedure focuses on identifying member's membership level, maximum number of borrowing books and current no of books borrowed to decide whether the users' are eligible for borrowing the books.**

**Procedure code:**

```
CREATE OR REPLACE PROCEDURE prc_check_user_borrow (
    p_member_id     IN  borrow.member_id%TYPE,
    p_borrow_count  IN  NUMBER,
    p_eligible      OUT NUMBER
) AS
    v_membership_id     membership.membership_id%TYPE;
    v_membership_level  membership.membership_level%TYPE;
    v_max_borrow_book   membership.max_borrow_book%TYPE;
    v_current_borrow_no NUMBER;
BEGIN
    -- obtain membership details
    SELECT membership_id
```

```
    INTO v_membership_id
    FROM member
    WHERE member_id = p_member_id;

    -- obtain membership details
    SELECT membership_level, max_borrow_book
    INTO v_membership_level, v_max_borrow_book
    FROM membership
    WHERE membership_id = v_membership_id;

    -- obtain number of books are currently borrowed by the member
(not yet returned)
    SELECT COUNT(*)
    INTO v_current_borrow_no
    FROM borrow_detail bd
    JOIN borrow b
        ON bd.borrow_id = b.borrow_id
    WHERE b.member_id = p_member_id
        AND bd.borrow_status = 'Borrowed';

    -- check if user has exceeded the book borrow limit
    IF v_current_borrow_no + p_borrow_count > v_max_borrow_book THEN
        DBMS_OUTPUT.PUT_LINE('Error: You have exceedded the book
borrow limit');
        DBMS_OUTPUT.PUT_LINE('Maximum book borrow: ' ||
v_max_borrow_book);
        DBMS_OUTPUT.PUT_LINE('You have borrowed ' ||
v_current_borrow_no || ' books currently.');
        p_eligible := 0;
        RETURN;
    END IF;

    p_eligible := 1;
END;
/
```

Scenario 1: Member MB0001 borrows 2 books after borrowing 1 book when his maximum borrowed book is 2 only. (prc_check_user_borrow procedure is the sub procedure called in the prc_borrow_books procedure)

| Input | Borrow 1 book:<br><br>`DECLARE`<br>`  v_book_ids book_id_list_t := book_id_list_t();`<br>`BEGIN`<br>`  v_book_ids.EXTEND(1);`<br>`  v_book_ids(1) := 8;`<br><br>`  prc_borrow_books(1, v_book_ids);`<br>`END;`<br>`/`<br><br>Borrow 3 more books:<br><br>`DECLARE`<br>`  v_book_ids book_id_list_t := book_id_list_t();`<br>`BEGIN`<br>`  v_book_ids.EXTEND(3);`<br>`  v_book_ids(1) := 12;`<br>`  v_book_ids(2) := 11;`<br>`  v_book_ids(3) := 5;`<br><br>`  prc_borrow_books(1, v_book_ids);`<br>`END;` |
|---|---|

```
/
```

```
SQL> DECLARE
  2      v_book_ids book_id_list_t := book_id_list_t();
  3  BEGIN
  4      -- Add book IDs to the collection
  5      v_book_ids.EXTEND(1);
  6      v_book_ids(1) := 8;
  7
  8      -- Call the procedure with member_id and book_id list
  9      prc_borrow_books(1, v_book_ids);
 10  END;
 11  /

--------------------------------------------------------------------------------
BOOK BORROWING
--------------------------------------------------------------------------------
Member ID   : MB0001
Member Name : Adam Tan
Borrow ID   : BR0153
Borrow Date : 04-MAY-2025 00:39:59

Book ID        Book Name                                           ISBN
--------------------------------------------------------------------------------
BK0008         Sister whole.                                       978000000007
--------------------------------------------------------------------------------
                                                Total Book(s):            1
--------------------------------------------------------------------------------

Return Date : 06-MAY-2025 00:39:59

You have borrowed the book(s) successfully!

PL/SQL procedure successfully completed.
```

```
SQL> DECLARE
  2      v_book_ids book_id_list_t := book_id_list_t();
  3  BEGIN
  4      v_book_ids.EXTEND(3);
  5      v_book_ids(1) := 12;
  6      v_book_ids(2) := 11;
  7      v_book_ids(3) := 5;
  8
  9      prc_borrow_books(1, v_book_ids);
 10  END;
 11  /
Error: You have exceedded the book borrow limit
Maximum book borrow: 3
You have borrowed 1 books currently.
Error: Sorry. You are not eligible to borrow the books.

PL/SQL procedure successfully completed.
```

**Procedure 4: Borrow detail insertion for borrowing books**

**Purpose: This procedure is designed to handle the insertion of multiple book borrowing records into the Borrow_Detail table for a specific borrow transaction. It ensures that each book is properly recorded as borrowed, handles potential errors gracefully, and provides appropriate feedback if there are operational failures.**

**Procedure code:**

```
CREATE OR REPLACE PROCEDURE prc_insert_borrow_detail (
```

```
    p_borrow_id IN  borrow.borrow_id%TYPE,
    p_book_ids  IN  book_id_list_t,
    p_result    OUT NUMBER
) AS
BEGIN
    SAVEPOINT before_insert;
    FOR i IN 1 .. p_book_ids.COUNT LOOP
        DECLARE
            v_formatted_book_id book.book_id%TYPE;
            v_borrow_detail_id  borrow_detail.borrow_detail_id%TYPE;
        BEGIN
            -- format book_id
            v_formatted_book_id := 'BK' || LPAD(p_book_ids(i), 4,
'0');

            -- insert borrow_detail record
            BEGIN
                INSERT INTO Borrow_detail (borrow_id, book_id,
borrow_status)
                VALUES (p_borrow_id, v_formatted_book_id, 'Borrowed')
                RETURNING borrow_detail_id INTO v_borrow_detail_id;
            EXCEPTION
                WHEN DUP_VAL_ON_INDEX THEN
                    DBMS_OUTPUT.PUT_LINE('Error: Duplicate entry for
book_id: ' || v_formatted_book_id);
                    ROLLBACK TO before_insert;
                    p_result := 0;
                    RETURN;
                WHEN OTHERS THEN
                    DBMS_OUTPUT.PUT_LINE('Error: Failed inserting
borrow detail for book_id: ' || v_formatted_book_id || ' - ' ||
SQLERRM);
                    ROLLBACK TO before_insert;
                    p_result := 0;
                    RETURN;
            END;
        END;
    END LOOP;
    COMMIT;
    p_result := 1;
END;
/
```

Scenario 1: Member MB0001 borrows books BK0001 and BK0002.

| Input | |
|---|---|

```
DECLARE
    v_book_ids book_id_list_t := book_id_list_t();
BEGIN
    v_book_ids.EXTEND(2);
    v_book_ids(1) := 1;
    v_book_ids(2) := 2;

    prc_borrow_books(1, v_book_ids);
END;
/
```

Output

```
SQL> DECLARE
  2      v_book_ids book_id_list_t := book_id_list_t();
  3  BEGIN
  4      v_book_ids.EXTEND(2);
  5      v_book_ids(1) := 1;
  6      v_book_ids(2) := 2;
  7
  8      prc_borrow_books(1, v_book_ids);
  9  END;
 10  /

--------------------------------------------------------------------------------
BOOK BORROWING
--------------------------------------------------------------------------------
Member ID   : MB0001
Member Name : Adam Tan
Borrow ID   : BR0151
Borrow Date : 04-MAY-2025 00:46:29

Book ID        Book Name                                          ISBN
--------------------------------------------------------------------------------
BK0001         Blood natural pretty worker.                       978000000000
BK0002         The.                                               978000000001
--------------------------------------------------------------------------------
                                                 Total Book(s):            2
--------------------------------------------------------------------------------

Return Date : 06-MAY-2025 00:46:29

You have borrowed the book(s) successfully!

PL/SQL procedure successfully completed.
```

```
SQL> SELECT * FROM BORROW_DETAIL WHERE BORROW_ID = 'BR0151';

BORROW BORROW BOOK_I RETURNED_ BORROW_STATUS
------ ------ ------ --------- --------------------
BD0151 BR0151 BK0001           Borrowed
BD0152 BR0151 BK0002           Borrowed
```

**Procedure 5: Searching Book Borrowed by Member**

**Purpose: The prc_search_book_borrowed_by_member procedure is designed to retrieve and display a list of books currently borrowed by a specific library member. It validates the provided member ID, counts the number of borrowed books, and for each borrowed book, it displays details such as the borrow detail ID, book ID, book name, borrow and return dates, and the remaining days until the return date.**

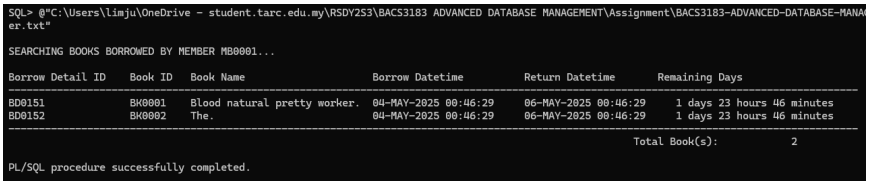
**Procedure code:**

```
CREATE OR REPLACE PROCEDURE prc_search_book_borrowed_by_member (
    p_member_id IN Member.member_id%TYPE
) AS
    v_formatted_member_id Member.member_id%TYPE;
    v_valid_member_id   NUMBER;
    v_book_count NUMBER;
    v_remaining_time VARCHAR2(50);
BEGIN
    -- check if member_id is valid
    v_formatted_member_id := 'MB' || LPAD(p_member_id, 4, '0');

    SELECT COUNT(*)
    INTO v_valid_member_id
    FROM Member
    WHERE member_id = v_formatted_member_id;
```

```
    IF v_valid_member_id = 0 THEN
        DBMS_OUTPUT.PUT_LINE('Error: Invalid member ID.');
        RETURN;
    END IF;

    -- Count borrowed books
    SELECT COUNT(*)
    INTO v_book_count
    FROM Borrow_Detail bd
    JOIN Borrow br
        ON bd.borrow_id = br.borrow_id
    WHERE br.member_id = v_formatted_member_id
      AND bd.borrow_status = 'Borrowed';

    -- search for books borrowed by the member
    DBMS_OUTPUT.PUT_LINE(CHR(13));
    DBMS_OUTPUT.PUT_LINE('SEARCHING BOOKS BORROWED BY MEMBER ' ||
v_formatted_member_id || '...');
    DBMS_OUTPUT.PUT_LINE(CHR(13));

    DBMS_OUTPUT.PUT_LINE(RPAD('Borrow Detail ID', 20) || RPAD('Book
ID', 10) || RPAD('Book Name', 30) || RPAD('Borrow Datetime', 25) ||
RPAD('Return Datetime', 25, ' ') || RPAD('Remaining Days', 30));
    DBMS_OUTPUT.PUT_LINE(RPAD('-', 20, '-') || RPAD('-', 10, '-') ||
RPAD('-', 30, '-') || RPAD('-', 25, '-') || RPAD('-', 25, '-') ||
RPAD('-', 30, '-'));

    FOR rec IN (
        SELECT bd.borrow_detail_id, b.book_id, b.name,
br.borrow_datetime, br.return_datetime
        FROM Borrow_Detail bd
        JOIN Book b
            ON bd.book_id = b.book_id
        JOIN Borrow br
            ON bd.borrow_id = br.borrow_id
        WHERE br.member_id = v_formatted_member_id
            AND bd.borrow_status = 'Borrowed'
    ) LOOP
        -- Calculate remaining time using the UDF
        v_remaining_time := fn_remaining_time(rec.return_datetime);

        DBMS_OUTPUT.PUT_LINE(RPAD(rec.borrow_detail_id, 20) ||
RPAD(rec.book_id, 10) || RPAD(SUBSTR(rec.name, 1, 28), 30) ||
RPAD(TO_CHAR(rec.borrow_datetime, 'DD-MON-YYYY HH24:MI:SS'), 25) ||
RPAD(TO_CHAR(rec.return_datetime,  'DD-MON-YYYY HH24:MI:SS'), 25) ||
RPAD(v_remaining_time, 30));
    END LOOP;
    DBMS_OUTPUT.PUT_LINE(RPAD('-', 20, '-') || RPAD('-', 10, '-') ||
RPAD('-', 30, '-') || RPAD('-', 25, '-') || RPAD('-', 25, '-') ||
RPAD('-', 30, '-'));
    DBMS_OUTPUT.PUT_LINE(LPAD('Total Book(s):', 120) ||
LPAD(v_book_count, 20));

    IF SQL%ROWCOUNT = 0 THEN
        DBMS_OUTPUT.PUT_LINE('No books found for this member.');
    END IF;
END;
/
```

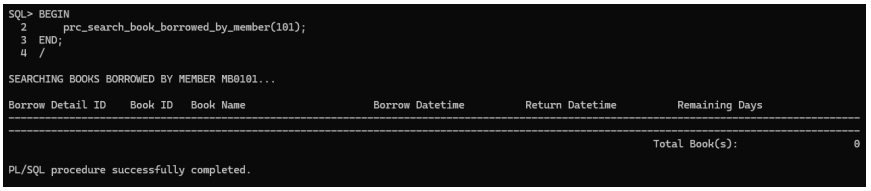
Scenario 1: Search books for member MB0001 who has borrowed books BK0001 and BK0002.

| Input | `BEGIN`<br>`    prc_search_book_borrowed_by_member(1);`<br>`END;`<br>`/` |
|---|---|

| Output |  |
|---|---|



Scenario 2: Search books for member MB0101 who never borrow any book.

| Input |  |
|---|---|



| Output |  |
|---|---|



**Procedure 6: Member Registration**

**Purpose: This procedure is mainly used for registering a new library member by validating inputs (name, gender, email, and Malaysian phone number), ensuring the email is unique, formatting the phone number, and inserting the member into the database with default membership and credit. Upon successful registration, it retrieves and displays the member's details, including membership level and borrowing privileges.**

**Procedure code:**

```
CREATE OR REPLACE PROCEDURE prc_member_registration (
    p_name      IN Member.name%TYPE,
    p_gender    IN Member.gender%TYPE,
    p_email     IN Member.email%TYPE,
    p_tel_no    IN Member.tel_no%TYPE
) AS
    v_name_valid NUMBER := 1;
    v_gender_valid NUMBER := 1;
    v_email_valid NUMBER := 1;
    v_email_count NUMBER;
    v_tel_no_valid NUMBER := 1;
    v_formatted_tel_no Member.tel_no%TYPE;
    v_member_id Member.member_id%TYPE;
BEGIN
    -- validate name
    IF p_name IS NULL OR NOT REGEXP_LIKE(p_name, '^[A-Za-z ]+$') THEN
        v_name_valid := 0;
        DBMS_OUTPUT.PUT_LINE('Error: Invalid name. Only letters and
spaces are allowed.');
        RETURN;
    END IF;

    -- validate gender
    IF UPPER(p_gender) NOT IN ('M', 'F') THEN
        v_gender_valid := 0;
        DBMS_OUTPUT.PUT_LINE('Error: Invalid gender. Must be ''M'' or
''F''.');
        RETURN;
```

```
    END IF;

    -- validate email
    IF NOT REGEXP_LIKE(p_email,
'^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$') THEN
        v_email_valid := 0;
        DBMS_OUTPUT.PUT_LINE('Error: Invalid email format.');
        RETURN;
    END IF;

    -- check if email already exists
    SELECT COUNT(*) INTO v_email_count FROM Member WHERE email =
p_email;

    IF v_email_count > 0 THEN
        v_email_valid := 0;
        DBMS_OUTPUT.PUT_LINE('Error: Email already exists.');
        RETURN;
    END IF;

    -- validate tel_no
    IF p_tel_no IS NULL OR NOT REGEXP_LIKE(p_tel_no,
'^01[0-46-9]-?\d{7,8}$') THEN
        v_tel_no_valid := 0;
        DBMS_OUTPUT.PUT_LINE('Error: Invalid Malaysian mobile phone
number format.');
        RETURN;
    END IF;

    -- format tel_no to include hyphen if missing
    IF INSTR(p_tel_no, '-') = 0 THEN
        v_formatted_tel_no := SUBSTR(p_tel_no, 1, 3) || '-' ||
SUBSTR(p_tel_no, 4);
    ELSE
        v_formatted_tel_no := p_tel_no;
    END IF;

    -- if all validations passed, insert into Member table
    SAVEPOINT before_insert;
    IF v_name_valid = 1 AND v_gender_valid = 1 AND v_email_valid = 1
AND v_tel_no_valid = 1 THEN
        INSERT INTO Member (membership_id, name, gender, email,
tel_no, credit, reg_date, mb_status)
        VALUES ('MS0002', p_name, UPPER(p_gender), p_email,
v_formatted_tel_no, 100, SYSDATE, 1)
        RETURNING member_id INTO v_member_id;

        -- successful message using view
        DECLARE
            msg_name Member.name%TYPE;
            msg_gender Member.gender%TYPE;
            msg_formatted_gender VARCHAR2(20);
            msg_email Member.email%TYPE;
            msg_tel_no Member.tel_no%TYPE;
            msg_credit Member.credit%TYPE;
            msg_reg_date Member.reg_date%TYPE;
            msg_mb_status Member.mb_status%TYPE;
            msg_formatted_mb_status VARCHAR2(20);
            msg_membership_level Membership.membership_level%TYPE;
            msg_max_borrow_day Membership.max_borrow_day%TYPE;
            msg_max_borrow_book Membership.max_borrow_book%TYPE;
            msg_max_loan_extension_day
Membership.max_loan_extension_day%TYPE;
        BEGIN
            SELECT name, gender, email, tel_no, credit, reg_date,
mb_status,
                   membership_level, max_borrow_day, max_borrow_book,
max_loan_extension_day
```

```
            INTO
                msg_name, msg_gender, msg_email, msg_tel_no,
msg_credit, msg_reg_date, msg_mb_status,
                msg_membership_level, msg_max_borrow_day,
msg_max_borrow_book, msg_max_loan_extension_day
            FROM vw_member_details
            WHERE member_id = v_member_id;

            -- format gender and status
            msg_formatted_gender := CASE msg_gender WHEN 'M' THEN
'Male' ELSE 'Female' END;
            msg_formatted_mb_status := CASE msg_mb_status WHEN 1 THEN
'Active' ELSE 'Inactive' END;

            -- Output
            DBMS_OUTPUT.PUT_LINE(CHR(13));
            DBMS_OUTPUT.PUT_LINE(RPAD('-', 70, '-'));
            DBMS_OUTPUT.PUT_LINE('Member Registration');
            DBMS_OUTPUT.PUT_LINE(RPAD('-', 70, '-'));
            DBMS_OUTPUT.PUT_LINE('Member ID                    : ' ||
v_member_id);
            DBMS_OUTPUT.PUT_LINE('Name                         : ' ||
msg_name);
            DBMS_OUTPUT.PUT_LINE('Gender                       : ' ||
msg_formatted_gender);
            DBMS_OUTPUT.PUT_LINE('Email                        : ' ||
msg_email);
            DBMS_OUTPUT.PUT_LINE('Tel No                       : ' ||
msg_tel_no);
            DBMS_OUTPUT.PUT_LINE('Membership Level             : ' ||
msg_membership_level);
            DBMS_OUTPUT.PUT_LINE('Credit                       : ' ||
msg_credit);
            DBMS_OUTPUT.PUT_LINE('Maximum Borrow Days          : ' ||
msg_max_borrow_day || ' days');
            DBMS_OUTPUT.PUT_LINE('Maximum Books Borrowed       : ' ||
msg_max_borrow_book || ' books');
            DBMS_OUTPUT.PUT_LINE('Maximum Loan Extension Days  : ' ||
msg_max_loan_extension_day || ' days');
            DBMS_OUTPUT.PUT_LINE('Registration Date            : ' ||
TO_CHAR(msg_reg_date, 'YYYY-MM-DD'));
            DBMS_OUTPUT.PUT_LINE('Member Status                : ' ||
msg_formatted_mb_status);
            DBMS_OUTPUT.PUT_LINE(RPAD('-', 70, '-'));
            DBMS_OUTPUT.PUT_LINE(CHR(13));
            DBMS_OUTPUT.PUT_LINE('Member registered successfully!');
        END;
    ELSE
        DBMS_OUTPUT.PUT_LINE('Error: Member registration failed due to
validation errors.');
        RETURN;
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK TO before_insert;
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
        RETURN;
END;
/
```

Scenario 1: Member registered with correct attributes.

| Input | DECLARE<br>v_name Member.name%TYPE := 'Chow Kim Meng'; -- Replace with actual member name<br>v_gender Member.gender%TYPE := 'M';<br>v_email Member.email%TYPE := 'cchow_meng386@gmail.com'; |
|---|---|

| | |
|---|---|
| | v_tel_no Member.tel_no%TYPE := '011-5684213';<br>BEGIN<br>prc_member_registration(v_name, v_gender, v_email, v_tel_no);<br>END;<br>/ |
| Output | SQL> DECLARE<br>2 v_name Member.name%TYPE := 'Chow Kim Meng'; -- Replace with actual member name<br>3 v_gender Member.gender%TYPE := 'M';<br>4 v_email Member.email%TYPE := 'cchow_meng386@gmail.com';<br>5 v_tel_no Member.tel_no%TYPE := '011-5684213';<br>6 BEGIN<br>7 prc_member_registration(v_name, v_gender, v_email, v_tel_no);<br>8 END;<br>9 /<br><br>---<br>Member Registration<br>---<br>Member ID : MB0102<br>Name : Chow Kim Meng<br>Gender : Male<br>Email : cchow_meng386@gmail.com<br>Tel No : 011-5684213<br>Membership Level : 1<br>Credit : 100<br>Maximum Borrow Days : 2 days<br>Maximum Books Borrowed : 2 books<br>Maximum Loan Extension Days : 3 days<br>Registration Date : 2025-05-04<br>Member Status : Active<br>---<br><br>Member registered successfully!<br><br>PL/SQL procedure successfully completed. |

Scenario 2: Registering member using incorrect name format.

| | |
|---|---|
| Input | DECLARE<br>v_name Member.name%TYPE := 'Alice Chan@/'; -- Replace with actual member name<br>v_gender Member.gender%TYPE := 'F';<br>v_email Member.email%TYPE := 'alice_cchan386@gmail.com';<br>v_tel_no Member.tel_no%TYPE := '011-6823541';<br>BEGIN<br>prc_member_registration(v_name, v_gender, v_email, v_tel_no);<br>END;<br>/ |
| Output | SQL> DECLARE<br>2 v_name Member.name%TYPE := 'Alice Chan@/'; -- Replace with actual member name<br>3 v_gender Member.gender%TYPE := 'F';<br>4 v_email Member.email%TYPE := 'alice_cchan386@gmail.com';<br>5 v_tel_no Member.tel_no%TYPE := '011-6823541';<br>6 BEGIN<br>7 prc_member_registration(v_name, v_gender, v_email, v_tel_no);<br>8 END;<br>9 /<br>Error: Invalid name. Only letters and spaces are allowed.<br><br>PL/SQL procedure successfully completed. |

Scenario 3: Registering member using incorrect gender.

| | |
|---|---|
| Input | DECLARE<br>v_name Member.name%TYPE := 'Ranold Lim'; -- Replace with |

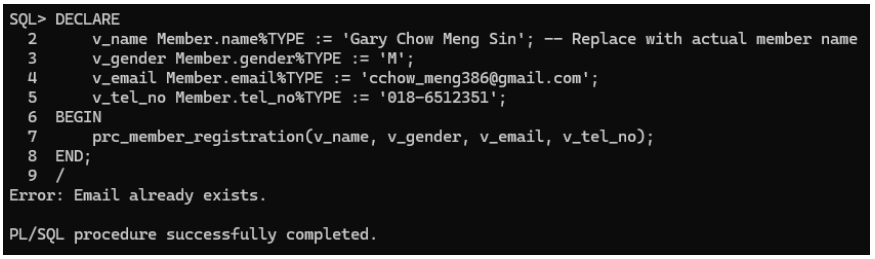
| | |
|---|---|
| | actual member name<br>v_gender Member.gender%TYPE := 'T';<br>v_email Member.email%TYPE := 'ran_old_236@gmail.com';<br>v_tel_no Member.tel_no%TYPE := '018-6321023';<br>BEGIN<br>prc_member_registration(v_name, v_gender, v_email, v_tel_no);<br>END;<br>/ |
| Output | SQL> DECLARE<br>2 v_name Member.name%TYPE := 'Ranold Lim'; -- Replace with actual member name<br>3 v_gender Member.gender%TYPE := 'T';<br>4 v_email Member.email%TYPE := 'ran_old_236@gmail.com';<br>5 v_tel_no Member.tel_no%TYPE := '018-6321023';<br>6 BEGIN<br>7 prc_member_registration(v_name, v_gender, v_email, v_tel_no);<br>8 END;<br>9 /<br>Error: Invalid gender. Must be 'M' or 'F'.<br><br>PL/SQL procedure successfully completed. |

Scenario 4: Registering member using invalid email address.

| | |
|---|---|
| Input | DECLARE<br>v_name Member.name%TYPE := 'Sow Heng Cham'; -- Replace with actual member name<br>v_gender Member.gender%TYPE := 'M';<br>v_email Member.email%TYPE := 'sh_cham-gmail.com';<br>v_tel_no Member.tel_no%TYPE := '018-6513021';<br>BEGIN<br>prc_member_registration(v_name, v_gender, v_email, v_tel_no);<br>END;<br>/ |
| Output | SQL> DECLARE<br>2 v_name Member.name%TYPE := 'Sow Heng Cham'; -- Replace with actual member name<br>3 v_gender Member.gender%TYPE := 'M';<br>4 v_email Member.email%TYPE := 'sh_cham-gmail.com';<br>5 v_tel_no Member.tel_no%TYPE := '018-6513021';<br>6 BEGIN<br>7 prc_member_registration(v_name, v_gender, v_email, v_tel_no);<br>8 END;<br>9 /<br>Error: Invalid email format.<br><br>PL/SQL procedure successfully completed. |

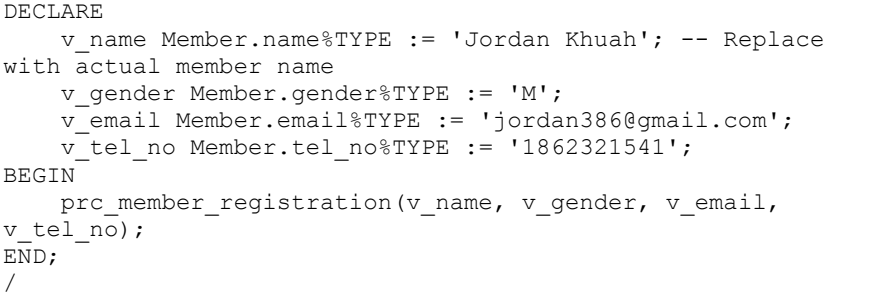
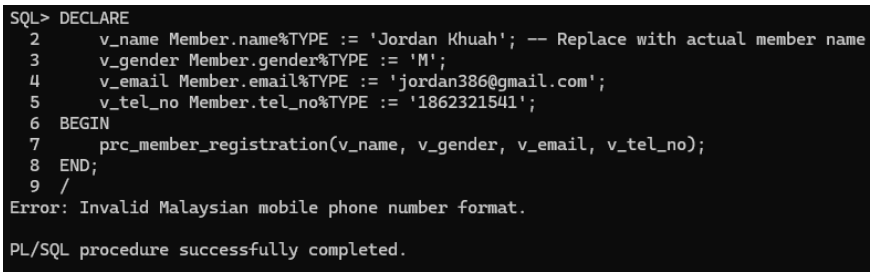
Scenario 5: Registering member using an existing email address.

| | |
|---|---|
| Input | DECLARE<br>v_name Member.name%TYPE := 'Gary Chow Meng Sin'; -- Replace with actual member name<br>v_gender Member.gender%TYPE := 'M';<br>v_email Member.email%TYPE := 'cchow_meng386@gmail.com';<br>v_tel_no Member.tel_no%TYPE := '018-6512351';<br>BEGIN<br>prc_member_registration(v_name, v_gender, v_email, v_tel_no);<br>END;<br>/ |

| | |
|---|---|
| Output |  |



Scenario 6: Registering member using invalid telephone numbers.

| | |
|---|---|
| Input |  |
| Output |  |





### 5.2.4 Job Scheduler

**Scheduler 1: Automation of daily updating member credit**

**Purpose: This job scheduler is created for constantly adding the member credit every day. It acts like a "recovering" process for increasing each member's credit since the credit may decide how many books can be borrowed, how long the books can be borrowed, how many days the borrowed book can be extended and so on. The job will run the stored procedure called "prc_add_member_credit_daily" every day at midnight (00:00:00). Once created, the job is immediately enabled and will automatically execute daily to perform its task for adding 2 credits for each library member.**

**Scheduler code:**

```
BEGIN
    DBMS_SCHEDULER.create_job (
        job_name        => 'add_member_credit_daily',
        job_type        => 'PLSQL_BLOCK',
        job_action      => 'BEGIN prc_add_member_credit_daily();
END;',
        start_date      => SYSTIMESTAMP,
        repeat_interval => 'FREQ=DAILY; BYHOUR=0; BYMINUTE=0;
BYSECOND=0',
        enabled         => TRUE
```

```
    );
END;
/
CREATE OR REPLACE PROCEDURE prc_add_member_credit_daily
AS
BEGIN
    UPDATE Member
    SET credit = credit + 2;
END;
/
```

### 5.2.5 Sequence & Trigger

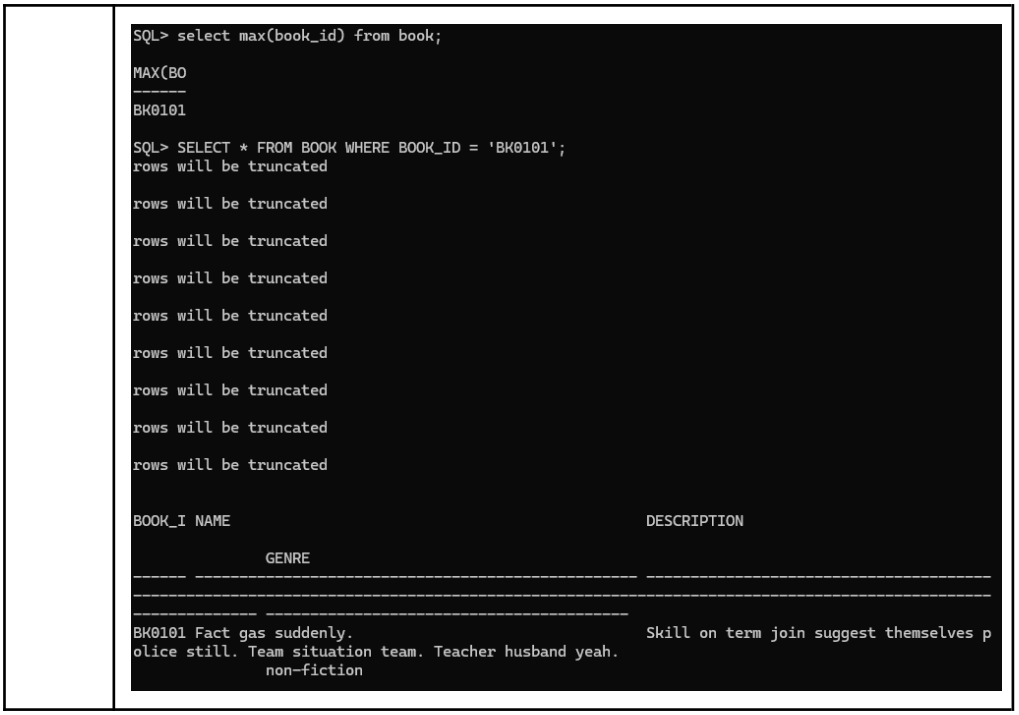
**Sequence & Trigger 1: Auto-Increment Book ID**

**Purpose: The book_seq sequence and book_before_insert trigger collaborate to automatically generate unique, formatted book_id values for new records in the Book table. When a new book is inserted without a specified book_id, the trigger activates before the insertion and assigns an ID in the format 'BK0001', 'BK0002'.**

**Exception code:**

```
CREATE SEQUENCE book_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

-- Auto-Increment for book_id
CREATE OR REPLACE TRIGGER book_before_insert
                                                    BEFORE INSERT ON Book
                                                            FOR EACH ROW
                                                                BEGIN

                                                IF :NEW.book_id IS NULL THEN

  SELECT 'BK' || LPAD(book_seq.NEXTVAL, 4, '0') INTO :NEW.book_id FROM
dual;
                                                                    END IF;
END;
/
```

| | |
|---|---|
| Input | `INSERT INTO Book (name, description, genre, category, language, isbn, author, publisher, publish_date, reg_date, availability, bk_status)`<br>`VALUES ('Fact gas suddenly.', 'Skill on term join suggest themselves police still. Team situation team. Teacher husband yeah.', 'non-fiction', 'Politics and Current Affairs', 'ZH', '978000000128', 'Patricia Smith', 'Adams Group', TO_DATE('2023-11-03', 'YYYY-MM-DD'), TO_DATE('2025-03-29', 'YYYY-MM-DD'), 'N', 'Inactive');` |
| Output | `SQL> INSERT INTO Book (name, description, genre, category, language, isbn, author, publisher, publish_date, reg_date, availability, bk_status)`<br>`  2  VALUES ('Fact gas suddenly.', 'Skill on term join suggest themselves police still. Team situation team. Teacher husband yeah.', 'non-fiction', 'Politics and Current Affairs', 'ZH', '978000000128', 'Patricia Smith', 'Adams Group', TO_DATE('2023-11-03', 'YYYY-MM-DD'), TO_DATE('2025-03-29', 'YYYY-MM-DD'), 'N', 'Inactive');`<br>`1 row created.` |



```
SQL> select max(book_id) from book;

MAX(BO
------
BK0101

SQL> SELECT * FROM BOOK WHERE BOOK_ID = 'BK0101';
rows will be truncated

rows will be truncated

rows will be truncated

rows will be truncated

rows will be truncated

rows will be truncated

rows will be truncated

rows will be truncated

rows will be truncated


BOOK_I NAME                                               DESCRIPTION

              GENRE
------ ---------------------------------------------- ------------------------------------
-------------------------------------------------------------------------------------------
------------- ----------------------------------------
BK0101 Fact gas suddenly.                                 Skill on term join suggest themselves p
olice still. Team situation team. Teacher husband yeah.
              non-fiction
```

### 5.2.6 User Defined Exception

**Exception 1: Exception Handling for Borrow Detail Insertion**

**Purpose: This exception purpose is to handle potential errors when inserting a new borrow detail, including duplicate book entries and other unforeseen issues, ensuring transaction consistency and user feedback.**

**Exception code:**

```
BEGIN
                INSERT INTO Borrow_detail (borrow_id, book_id,
borrow_status)
                VALUES (p_borrow_id, v_formatted_book_id, 'Borrowed')
                RETURNING borrow_detail_id INTO v_borrow_detail_id;
            EXCEPTION
                WHEN DUP_VAL_ON_INDEX THEN
                    DBMS_OUTPUT.PUT_LINE('Error: Duplicate entry for
book_id: ' || v_formatted_book_id);
                    ROLLBACK TO before_insert;
                    p_result := 0;
                    RETURN;
                WHEN OTHERS THEN
                    DBMS_OUTPUT.PUT_LINE('Error: Failed inserting
borrow detail for book_id: ' || v_formatted_book_id || ' - ' ||
SQLERRM);
                    ROLLBACK TO before_insert;
                    p_result := 0;
                    RETURN;
END;
```

## 5.3 Ong Yi Xin

### 5.3.1 Views

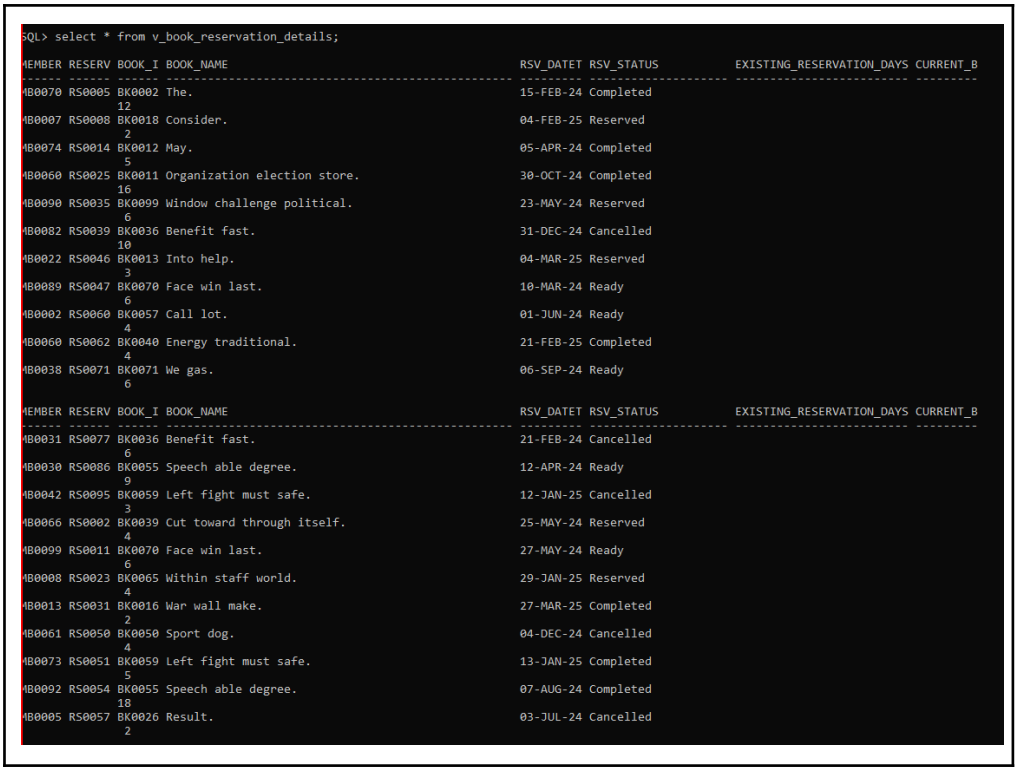
**View 1: Book Reservation Details**

**Purpose: The purpose of this view is to provide detailed information about book reservations, including the member ID, reservation ID, book details, reservation datetime, status, and the calculated existing reservation days. It also retrieves the current borrower's return date for each book, if applicable.**

**View code:**

```
CREATE OR REPLACE VIEW v_book_reservation_details AS
SELECT
    m.member_id,
    r.reservation_id,
    r.book_id,
    b.name AS book_name,
    r.rsv_datetime,
    r.rsv_status,
    NVL(SUM(ms.max_borrow_day), 0) AS existing_reservation_days,
    br.return_datetime AS current_borrower_return_date
FROM Reservation r
JOIN Book b ON r.book_id = b.book_id
JOIN Member m ON r.member_id = m.member_id
JOIN Membership ms ON m.membership_id = ms.membership_id
LEFT JOIN Borrow_Detail bd ON bd.book_id = r.book_id
LEFT JOIN Borrow br ON br.borrow_id = bd.borrow_id AND
bd.borrow_status = 'Borrowed'
GROUP BY
    m.member_id,
    r.reservation_id,
    r.book_id,
    b.name,
    r.rsv_datetime,
    r.rsv_status,
    br.return_datetime;
```

**Sample Output:**



```
SQL> select * from v_book_reservation_details;

MEMBER RESERV BOOK_I BOOK_NAME                                          RSV_DATET RSV_STATUS          EXISTING_RESERVATION_DAYS CURRENT_B
------ ------ ------ -------------------------------------------------- --------- ------------------- ------------------------- ---------
MB0070 RS0005 BK0002 The.                                               15-FEB-24 Completed
                  12
MB0007 RS0008 BK0018 Consider.                                          04-FEB-25 Reserved
                   2
MB0074 RS0014 BK0012 May.                                               05-APR-24 Completed
                   5
MB0060 RS0025 BK0011 Organization election store.                       30-OCT-24 Completed
                  16
MB0090 RS0035 BK0099 Window challenge political.                        23-MAY-24 Reserved
                   6
MB0082 RS0039 BK0036 Benefit fast.                                      31-DEC-24 Cancelled
                  10
MB0022 RS0046 BK0013 Into help.                                         04-MAR-25 Reserved
                   3
MB0089 RS0047 BK0070 Face win last.                                     10-MAR-24 Ready
                   6
MB0002 RS0060 BK0057 Call lot.                                          01-JUN-24 Ready
                   4
MB0060 RS0062 BK0040 Energy traditional.                                21-FEB-25 Completed
                   4
MB0038 RS0071 BK0071 We gas.                                            06-SEP-24 Ready
                   6

MEMBER RESERV BOOK_I BOOK_NAME                                          RSV_DATET RSV_STATUS          EXISTING_RESERVATION_DAYS CURRENT_B
------ ------ ------ -------------------------------------------------- --------- ------------------- ------------------------- ---------
MB0031 RS0077 BK0036 Benefit fast.                                      21-FEB-24 Cancelled
                   6
MB0030 RS0086 BK0055 Speech able degree.                                12-APR-24 Ready
                   9
MB0042 RS0095 BK0059 Left fight must safe.                              12-JAN-25 Cancelled
                   3
MB0066 RS0002 BK0039 Cut toward through itself.                         25-MAY-24 Reserved
                   4
MB0099 RS0011 BK0070 Face win last.                                     27-MAY-24 Ready
                   6
MB0008 RS0023 BK0065 Within staff world.                                29-JAN-25 Reserved
                   4
MB0013 RS0031 BK0016 War wall make.                                     27-MAR-25 Completed
                   2
MB0061 RS0050 BK0050 Sport dog.                                         04-DEC-24 Cancelled
                   4
MB0073 RS0051 BK0059 Left fight must safe.                              13-JAN-25 Completed
                   5
MB0092 RS0054 BK0055 Speech able degree.                                07-AUG-24 Completed
                  18
MB0005 RS0057 BK0026 Result.                                            03-JUL-24 Cancelled
                   2
```

**View 2: View for Member's Borrowed or Missing Book Return Status**

**Purpose: The purpose of this view is to provide a streamlined summary of active members and the books they currently hold with a status of either Borrowed or Missing. It helps verify whether books are eligible for return based on availability and condition, assisting in return validation and library audit processes.**

**SQL statement:**

```
CREATE OR REPLACE VIEW v_member_book_return_status AS
SELECT
    m.member_id,
    m.mb_status AS member_status,
    bd.book_id,
    bk.bk_status AS book_status,
    bk.availability,
    bd.borrow_id,
    bd.borrow_status AS book_borrow_status
FROM
    Member m
JOIN Borrow b ON m.member_id = b.member_id  -- member_id is now in the
Borrow table
JOIN Borrow_Detail bd ON b.borrow_id = bd.borrow_id
JOIN Book bk ON bd.book_id = bk.book_id
WHERE
    bd.borrow_status IN ('Borrowed', 'Missing')
    AND bk.availability = 'N'
    AND bk.bk_status = 'Active';
```

**Sample Output:**

```
SQL> select * from v_member_book_return_status
  2  ;

MEMBER MEMBER_STATUS BOOK_I BOOK_STATU A BORROW BOOK_BORROW_STATUS
------ ------------- ------ ---------- - ------ --------------------
MB0012              1 BK0066 Active     N BR0141 Missing
MB0013              1 BK0042 Active     N BR0052 Missing
MB0013              1 BK0066 Active     N BR0092 Missing
MB0014              1 BK0085 Active     N BR0137 Missing
MB0015              1 BK0095 Active     N BR0023 Missing
MB0019              1 BK0081 Active     N BR0126 Missing
MB0020              1 BK0004 Active     N BR0065 Missing
MB0021              1 BK0076 Active     N BR0104 Missing
MB0024              1 BK0042 Active     N BR0080 Missing
MB0026              1 BK0096 Active     N BR0131 Missing
MB0029              1 BK0096 Active     N BR0020 Missing

MEMBER MEMBER_STATUS BOOK_I BOOK_STATU A BORROW BOOK_BORROW_STATUS
------ ------------- ------ ---------- - ------ --------------------
MB0029              1 BK0081 Active     N BR0020 Missing
MB0031              1 BK0085 Active     N BR0047 Missing
MB0037              1 BK0043 Active     N BR0015 Missing
MB0044              1 BK0023 Active     N BR0058 Missing
MB0044              1 BK0085 Active     N BR0142 Missing
MB0050              1 BK0010 Active     N BR0149 Missing
MB0051              1 BK0003 Active     N BR0019 Missing
MB0051              1 BK0010 Active     N BR0034 Missing
MB0054              1 BK0085 Active     N BR0083 Missing
MB0057              1 BK0042 Active     N BR0043 Missing
MB0057              1 BK0013 Active     N BR0076 Missing

MEMBER MEMBER_STATUS BOOK_I BOOK_STATU A BORROW BOOK_BORROW_STATUS
------ ------------- ------ ---------- - ------ --------------------
MB0061              1 BK0030 Active     N BR0100 Missing
MB0065              1 BK0072 Active     N BR0121 Missing
MB0067              1 BK0072 Active     N BR0095 Missing
MB0070              1 BK0075 Active     N BR0150 Missing
MB0076              1 BK0019 Active     N BR0135 Missing
MB0077              1 BK0052 Active     N BR0146 Missing
MB0082              1 BK0042 Active     N BR0091 Missing
MB0087              1 BK0085 Active     N BR0067 Missing
MB0093              1 BK0010 Active     N BR0123 Missing
```

### 5.3.2 Indexes

**Index 1: Index Reservation Member Status**

**Purpose: The purpose of this query is to optimize searches and joins involving both member_id and rsv_status in the Reservation table, aiding in efficient retrieval of reservation data based on member and status filters.**

**SQL statement:**

```
CREATE INDEX idx_reservation_member_status ON Reservation(member_id,
rsv_status);
```

**Index 2: Index Book Borrow Book Status**

**Purpose: The purpose of this query is to improve query performance for filtering and joining on book_id and borrow_status in the Borrow table, particularly for tracking borrow statuses and related book details.**

**SQL statement:**

```
CREATE INDEX idx_borrow_book_status ON Borrow(book_id, borrow_status);
```

### 5.3.3 User Defined Functions

**Function 1: Fine Payment Eligibility Checker**

**Purpose: The purpose of this function is to check whether a specific fine belongs to a given member and is still unpaid. It returns 1 if eligible for payment, otherwise returns 0 with an appropriate error message.**

**SQL statement:**

```
CREATE OR REPLACE FUNCTION fn_is_fine_eligible (
    p_member_id IN Member.member_id%TYPE,
    p_fine_id   IN Fine_Record.fine_id%TYPE
) RETURN NUMBER IS
    v_fine_member_id    Member.member_id%TYPE;
    v_fine_status       Fine_Record.fine_status%TYPE;
BEGIN
    SELECT br.member_id, fr.fine_status
    INTO v_fine_member_id, v_fine_status
    FROM Fine_Record fr
    JOIN Borrow_Detail bd ON fr.borrow_detail_id = bd.borrow_detail_id
    JOIN Borrow br ON bd.borrow_id = br.borrow_id
    WHERE fr.fine_id = p_fine_id;

    IF v_fine_member_id != p_member_id THEN
        RAISE_APPLICATION_ERROR(-20001, 'Error: Fine record does not
belong to the member.');
        RETURN 0;
    ELSIF v_fine_status = 'Paid' THEN
        RAISE_APPLICATION_ERROR(-20001, 'Error: Fine record does not
belong to the member.');
        RETURN 0;
    END IF;

    RETURN 1;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Error: Fine record not found.');
        RETURN 0;
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
        RETURN 0;
END;
/
```

**Sample Output:**

| |
|---|
| Check is it the fine record exists and if not, show error message: |

```
SQL> select * from fine_record where fine_id = 'FN0300';

no rows selected

SQL> SET SERVEROUTPUT ON;
SQL>
SQL> -- increase line width
SQL> SET LINESIZE 200
SQL>
SQL> -- disable line wrapping
SQL> SET WRAP OFF
SQL>
SQL> -- enable line wrapping for output
SQL> SET SERVEROUTPUT ON FORMAT WRAPPED
SQL>
SQL> -- prc_pay_fine_record(member_id, fine_id, pay_amount)
SQL> BEGIN
  2      prc_pay_fine_record(70, 300, 10);
  3  END;
  4  /
Error: Fine record not found.
Error: The fine is failed to pay.

PL/SQL procedure successfully completed.
```

Check is the fine record belongs to member specific in parameter, if not show error message:

```
SQL> @SET SERVEROUTPUT ON;
SP2-0310: unable to open file "SET.sql"
SQL>
SQL> -- increase line width
SQL> SET LINESIZE 200
SQL>
SQL> -- disable line wrapping
SQL> SET WRAP OFF
SQL>
SQL> -- enable line wrapping for output
SQL> SET SERVEROUTPUT ON FORMAT WRAPPED
SQL>
SQL> -- prc_pay_fine_record(member_id, fine_id, pay_amount)
SQL> BEGIN
  2      prc_pay_fine_record(70, 30, 10);
  3  END;
  4  /
Error: Fine record is not available.
Error: The fine is failed to pay.

PL/SQL procedure successfully completed.
```

### 5.3.4 Sequence & Trigger

**Sequence & Trigger 1: Auto-Increment Borrow ID**

**Purpose: The purpose of this query is to automatically generate a unique borrow_id for each record inserted into the Borrow table. The trigger ensures that if the borrow_id is not provided, it will be automatically populated with a value based on the sequence.**

**SQL statement:**

```
CREATE SEQUENCE Borrow_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

-- Auto-Increment for borrow_id
CREATE OR REPLACE TRIGGER borrow_before_insert
                                                    BEFORE INSERT ON Borrow
                                                          FOR EACH ROW
                                                                BEGIN

                                                IF :NEW.borrow_id IS NULL THEN

   SELECT 'BR' || LPAD(borrow_seq.NEXTVAL, 4, '0') INTO :NEW.borrow_id
FROM dual;
                                                                      END IF;
END;
/
```

**Sample Output:**

```
BORROW MEMBER BORROW_DA RETURN_DA
------ ------ --------- ---------
BR0144 MB0091 27-JAN-25 31-JAN-25
BR0145 MB0098 04-MAR-25 08-MAR-25
BR0146 MB0077 14-DEC-24 19-DEC-24
BR0147 MB0095 01-MAR-25 07-MAR-25
BR0148 MB0060 05-JAN-25 11-JAN-25
BR0149 MB0050 18-DEC-24 23-DEC-24
BR0150 MB0070 28-JAN-25 31-JAN-25
BR0151 MB0001 07-FEB-25 09-FEB-25

151 rows selected.

SQL> INSERT INTO Borrow(member_id, borrow_datetime, return_datetime) VALUES ('MB0002', TO_DATE('7-2-2025 17:23:40',
DD-MM-YYYY HH24:MI:SS'), TO_DATE('9-2-2025 17:23:40', 'DD-MM-YYYY HH24:MI:SS'));

1 row created.

SQL> select * from borrow where member_id = 'MB0002';

BORROW MEMBER BORROW_DA RETURN_DA
------ ------ --------- ---------
BR0063 MB0002 15-FEB-25 20-FEB-25
BR0152 MB0002 07-FEB-25 09-FEB-25
```

### 5.3.5 User Defined Exception

**Exception 1: Default Exception Handling for Fine Record Lookup**

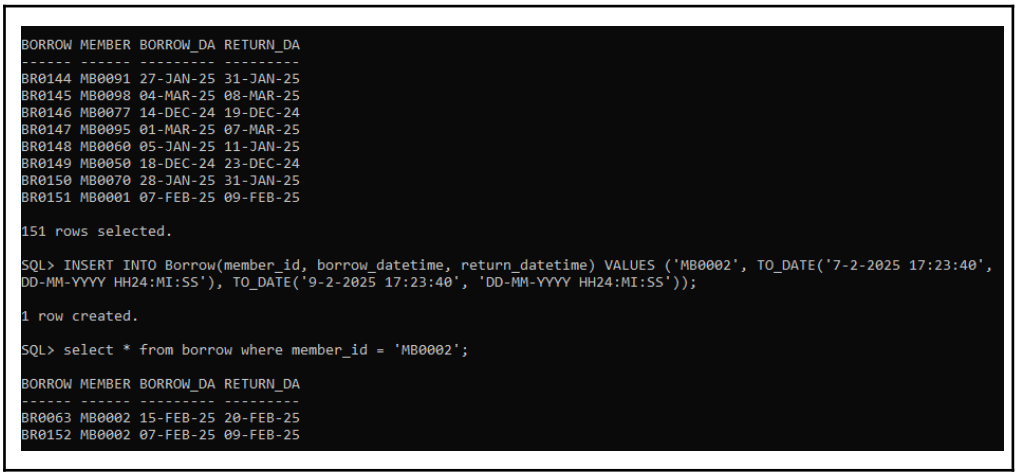
**Purpose: The purpose of this exception is to handles the errors during the fine record lookup process. It specifically addresses cases where no data is found or any unexpected errors occur, ensuring that the process doesn't fail silently.**

**SQL statement:**

```
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Error: Fine record not found.');
        RETURN 0;
```

```
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
        RETURN 0;
END;
/
```

**Sample Output:**



```
BORROW MEMBER BORROW_DA RETURN_DA
------ ------ --------- ---------
BR0144 MB0091 27-JAN-25 31-JAN-25
BR0145 MB0098 04-MAR-25 08-MAR-25
BR0146 MB0077 14-DEC-24 19-DEC-24
BR0147 MB0095 01-MAR-25 07-MAR-25
BR0148 MB0060 05-JAN-25 11-JAN-25
BR0149 MB0050 18-DEC-24 23-DEC-24
BR0150 MB0070 28-JAN-25 31-JAN-25
BR0151 MB0001 07-FEB-25 09-FEB-25

151 rows selected.

SQL> INSERT INTO Borrow(member_id, borrow_datetime, return_datetime) VALUES ('MB0002', TO_DATE('7-2-2025 17:23:40',
DD-MM-YYYY HH24:MI:SS'), TO_DATE('9-2-2025 17:23:40', 'DD-MM-YYYY HH24:MI:SS'));

1 row created.

SQL> select * from borrow where member_id = 'MB0002';

BORROW MEMBER BORROW_DA RETURN_DA
------ ------ --------- ---------
BR0063 MB0002 15-FEB-25 20-FEB-25
BR0152 MB0002 07-FEB-25 09-FEB-25
```

**Exception 2: Application Error for Fine Record Validation**

**Purpose: The purpose of this exception is to check if the fine record belongs to the specified member and if the fine has already been paid. If either condition is violated, an application error is raised with a descriptive message to indicate the issue.**

**SQL statement:**

```
   IF v_fine_member_id != p_member_id THEN
        RAISE_APPLICATION_ERROR(-20001, 'Error: Fine record does not
belong to the member.');
        RETURN 0;
    ELSIF v_fine_status = 'Paid' THEN
        RAISE_APPLICATION_ERROR(-20001, 'Error: Fine record does not
belong to the member.');
        RETURN 0;
```

**Sample Output:**

```
SQL> @SET SERVEROUTPUT ON;
SP2-0310: unable to open file "SET.sql"
SQL>
SQL> -- increase line width
SQL> SET LINESIZE 200
SQL>
SQL> -- disable line wrapping
SQL> SET WRAP OFF
SQL>
SQL> -- enable line wrapping for output
SQL> SET SERVEROUTPUT ON FORMAT WRAPPED
SQL>
SQL> -- prc_pay_fine_record(member_id, fine_id, pay_amount)
SQL> BEGIN
  2      prc_pay_fine_record(70, 30, 10);
  3  END;
  4  /
Error: ORA-20001: Error: Fine record does not belong to the member.
Error: The fine failed to pay.

PL/SQL procedure successfully completed.

SQL> select b.member_id from fine_record f JOIN borrow_detail bd ON f.borrow_detail_id = bd.borrow_detail_id JOIN bor
row b ON bd.borrow_id = b.borrow_id where fine_id = 'FN0030';

MEMBER
------
MB0048
```

## 5.4 Kevin Lo Yung Khang

### 5.4.1 Views

**View 1: Staff Task Assignments**

**Purpose: Combines Staff, Schedule, Task, and latest Task_Progress information to give a snapshot of who is assigned what task, when, and its current status. Useful for operational dashboards or quick lookups.**

**SQL Statement:**

```
CREATE OR REPLACE VIEW vw_Staff_Task_Assignments AS
SELECT
    s.staff_id,
    s.name AS staff_name,
    s.position,
    sc.schedule_id,
    t.task_id,
    t.description AS task_description,
    sc.start_datetime,
    sc.end_datetime,
    tp.tp_status AS latest_status,
    tp.tp_datetime AS last_update_time
FROM Staff s
JOIN Schedule sc ON s.staff_id = sc.staff_id
JOIN Task t ON sc.task_id = t.task_id
LEFT JOIN (
     SELECT
         tp_inner.schedule_id,
         tp_inner.tp_status,
         tp_inner.tp_datetime,
         ROW_NUMBER() OVER (PARTITION BY tp_inner.schedule_id ORDER BY
tp_inner.tp_datetime DESC) as rn
     FROM Task_Progress tp_inner
 ) tp ON sc.schedule_id = tp.schedule_id AND tp.rn = 1
WHERE s.stf_status = 'Active';
```

**Sample Output :**

```
STAFF_ Staff Name                SCHEDU TASK_I
------ ------------------------- ------ ------
TASK_DESCRIPTION
--------------------------------------------------------------------------------
START_DAT END_DATET LATEST_STATUS        LAST_UPDA
--------- --------- -------------------- ---------
SF0013 Ian Grey                  SC0002 TK0003
Database Backup and Maintenance
25-FEB-25 25-FEB-25 Completed            17-APR-25

SF0006 Bob White                 SC0004 TK0009
Assist Users at Digital Kiosks
08-APR-25 08-APR-25 Completed            16-APR-25

STAFF_ Staff Name                SCHEDU TASK_I
------ ------------------------- ------ ------
TASK_DESCRIPTION
--------------------------------------------------------------------------------
START_DAT END_DATET LATEST_STATUS        LAST_UPDA
--------- --------- -------------------- ---------

SF0013 Ian Grey                  SC0012 TK0006
Prepare for Children Story Time Event
03-MAR-25 03-MAR-25 Ongoing              08-APR-25
```

**View 2: Task Summary**

**Purpose: Provides summary statistics for each active task, including how many times it's scheduled, how many times it's completed, and the average duration of completed schedules (Tactical Level).**

**SQL Statement:**

```
CREATE OR REPLACE VIEW vw_Task_Summary AS
SELECT
    t.task_id,
    t.description,
    t.estimated_hour,
    COUNT(sc.schedule_id) AS times_scheduled,
    SUM(CASE WHEN tp.tp_status = 'Completed' THEN 1 ELSE 0 END) AS
times_completed,
    AVG(CASE WHEN tp.tp_status = 'Completed' THEN (sc.end_datetime -
sc.start_datetime) * 24 ELSE NULL END) AS avg_completed_duration_hours
FROM Task t
LEFT JOIN Schedule sc ON t.task_id = sc.task_id
LEFT JOIN (
     SELECT tp1.schedule_id, tp1.tp_status
     FROM Task_Progress tp1
     WHERE tp1.tp_datetime = (SELECT MAX(tp2.tp_datetime)
                              FROM Task_Progress tp2
                              WHERE tp2.schedule_id = tp1.schedule_id)
 ) tp ON sc.schedule_id = tp.schedule_id
WHERE t.tk_status = 'Active'
GROUP BY t.task_id, t.description, t.estimated_hour
ORDER BY t.task_id;
```

**Sample Output :**

```
TASK_I
------
DESCRIPTION
--------------------------------------------------------------------------------
ESTIMATED_HOUR TIMES_SCHEDULED TIMES_COMPLETED AVG_COMPLETED_DURATION_HOURS
-------------- --------------- --------------- ----------------------------
TK0001
Organize Fiction Bookshelves A-G
             3              13               2                         5.05

TK0002
Catalog New Arrivals - Batch 1
             4              16               3                   3.52777778

TASK_I
------
DESCRIPTION
--------------------------------------------------------------------------------
ESTIMATED_HOUR TIMES_SCHEDULED TIMES_COMPLETED AVG_COMPLETED_DURATION_HOURS
-------------- --------------- --------------- ----------------------------

TK0003
Database Backup and Maintenance
             2               8               1                          1.2
```

### 5.4.2 Indexes

**Index 1: Retrieving Latest Task Progress by Schedule**

**Purpose: Efficiently retrieves the latest task progress record for a given schedule_id by indexing schedule_id first and then tp_datetime in descending order. Used in procedures like prc_Update_Task_Progress, reports, and the vw_Staff_Task_Assignments view.**

**Index Code:**

```
CREATE INDEX idx_task_progress_sched_time ON
Task_Progress(schedule_id, tp_datetime DESC);
```

**Index 2: Filtering and Sorting Active Staff by Name**

**Purpose: Optimizes queries that filter by stf_status (e.g., 'Active') and potentially sort or select by name. Useful for various staff-related lookups and reports.**

**Index Code:**

```
CREATE INDEX idx_staff_status_name ON Staff(stf_status, name);
```

### 5.4.3 User Defined Functions

**Function 1: Get Latest Task Status for Schedule ID**

**Purpose: Encapsulates the logic to find the most recent status for a given schedule ID. Returns the status string or 'Not Started' if no progress record exists. Makes reports and queries cleaner.**

**Code:**

```
CREATE OR REPLACE FUNCTION fn_Get_Latest_Task_Status(
    p_schedule_id_num IN NUMBER
) RETURN VARCHAR2
AS
    v_formatted_schedule_id Schedule.schedule_id%TYPE;
    v_latest_status Task_Progress.tp_status%TYPE := 'Scheduled'; --
Changed default to 'Scheduled'
BEGIN
    -- Check if input is null
    IF p_schedule_id_num IS NULL THEN
         RETURN 'Invalid ID';
    END IF;

    v_formatted_schedule_id := 'SC' || LPAD(p_schedule_id_num, 4,
'0');

    -- Use ROWNUM=1 for compatibility
    BEGIN
        SELECT tp_status
        INTO v_latest_status
        FROM ( -- Inline view to order first
            SELECT tp.tp_status
            FROM Task_Progress tp
            WHERE tp.schedule_id = v_formatted_schedule_id
            ORDER BY tp.tp_datetime DESC
        ) ordered_progress -- Alias for inline view
        WHERE ROWNUM = 1; -- Filter for the top row (latest)
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            -- If no progress records found, assume it's still
'Scheduled'
            -- (as created by prc_Assign_Manual_Schedule)
            v_latest_status := 'Scheduled';
    END;

    RETURN NVL(v_latest_status, 'Scheduled'); -- Return status or
'Scheduled' if somehow NULL

EXCEPTION
     WHEN OTHERS THEN
          -- Log error if possible (e.g., using a logging procedure)
          RETURN 'Error'; -- Return generic error indicator
END;
/
```

**Sample Input: For schedule ID 38**

```
SELECT fn_Get_Latest_Task_Status(p_schedule_id_num =>
38) AS latest_status
  from dual;
```

**Sample Output :**

```
SQL> SELECT schedule_id, tp_status, tp_datetime
  2  FROM Task_Progress
  3  WHERE schedule_id = 'SC0038'
  4  ORDER BY tp_datetime DESC;

SCHEDU TP_STATUS           TP_DATETI
------ ------------------- ---------
SC0038 Scheduled           29-MAR-25

SQL> SELECT fn_Get_Latest_Task_Status(p_schedule_id_num => 111) AS latest_status
  2
SQL>
SQL> SELECT fn_Get_Latest_Task_Status(p_schedule_id_num => 38) AS latest_status
  2  from dual;

LATEST_STATUS
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
------------------------------
Scheduled
```

**Function 2: Calculate Planned Schedule Duration in Hours**

**Purpose: Calculates the planned duration of a schedule in hours.**

**Index Code:**

```
CREATE OR REPLACE FUNCTION fn_Calculate_Schedule_Duration(
    p_schedule_id_num IN NUMBER
) RETURN NUMBER
AS
    v_formatted_schedule_id Schedule.schedule_id%TYPE;
    v_duration NUMBER := 0;
BEGIN
    -- Check if input is null
    IF p_schedule_id_num IS NULL THEN
         RETURN -2; -- Indicate invalid input
    END IF;

    v_formatted_schedule_id := 'SC' || LPAD(p_schedule_id_num, 4,
'0');

    SELECT (end_datetime - start_datetime) * 24 -- Calculate hours
    INTO v_duration
    FROM Schedule
    WHERE schedule_id = v_formatted_schedule_id;

    -- NVL not really needed here because if NO_DATA_FOUND, exception
handles it.
    -- If SELECT returns NULL (shouldn't happen if dates are NOT
NULL), v_duration would be NULL.
    RETURN v_duration;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN 0; -- Return 0 hours if schedule not found
    WHEN OTHERS THEN
        -- Log error if possible
        RETURN -1; -- Return negative value to indicate an error
END;
/
```

**Sample Input: For schedule ID 1**

```
SELECT fn_Calculate_Schedule_Duration(11) FROM dual;
```

**Sample Output: For schedule ID 1**

```
SQL> SELECT schedule_id, start_datetime, end_datetime,
  2         (end_datetime - start_datetime) * 24 AS expected_duration
  3    FROM Schedule
  4   WHERE schedule_id = 'SC0001';

SCHEDU START_DAT END_DATET EXPECTED_DURATION
------ --------- --------- -----------------
SC0001 24-MAR-25 24-MAR-25              5.75
```

1. select the data and check

```
SQL> SELECT fn_Calculate_Schedule_Duration(p_schedule_id_num => 1) AS calculated_dur
ation
  2    FROM dual;

CALCULATED_DURATION
-------------------
               5.75
```

1. it is same and more efficient to calculate the duration

### 5.4.4 Sequence & Trigger

**Sequence 1: Auto-Assign ID Generator Using Sequence and Trigger**

**Purpose: This sequence and trigger work together to automatically generate a unique identifier (auto_assign_id) for new records inserted into the Auto_Assign table. The trigger uses the aa_seq sequence to format IDs with the prefix 'AA' followed by a 4-digit number (e.g., AA0001, AA0002, etc.), ensuring consistent and unique primary key values without manual input.**

**Sequence code:**

```
-- Sequence for Auto_Assign
CREATE SEQUENCE aa_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

-- Auto-Increment for auto_assign_id
CREATE OR REPLACE TRIGGER aa_before_insert
                                          BEFORE INSERT ON Auto_Assign
                                                       FOR EACH ROW
                                                              BEGIN

                                       IF :NEW.auto_assign_id IS NULL THEN

  SELECT 'AA' || LPAD(aa_seq.NEXTVAL, 4, '0') INTO :NEW.auto_assign_id
FROM dual;
                                                                    END IF;
END;
/
```

**Sample Input:**

```
-- Use valid Staff/Task IDs from your data (SF0001, TK0001)
INSERT INTO Auto_Assign
  (staff_id, task_id, frequency, set_date, start_time,
end_time, aa_status)
VALUES
  ('SF0001',                    -- staff_id
   'TK0001',                    -- task_id
   'Daily',                     -- frequency
```

```
        SYSDATE,                          -- set_date
        SYSDATE + interval '1' hour,    -- start_time (example)
        SYSDATE + interval '3' hour,    -- end_time (example)
        'Active'                          -- aa_status
       );
```

**Sample Output :**

```
SQL> SELECT sequence_name, last_number
  2    FROM user_sequences
  3   WHERE sequence_name = 'AA_SEQ';

SEQUENCE_NAME                 LAST_NUMBER
----------------------------- -----------
AA_SEQ                                111

SQL> INSERT INTO Auto_Assign
  2    (staff_id, task_id, frequency, set_date, start_time, end_time, aa_status)
  3  VALUES
  4    ('SF0001',                        -- staff_id
  5     'TK0001',                        -- task_id
  6     'Daily',                         -- frequency
  7     SYSDATE,                         -- set_date
  8     SYSDATE + interval '1' hour,   -- start_time (example)
  9     SYSDATE + interval '3' hour,   -- end_time (example)
 10     'Active'                         -- aa_status
 11    );

1 row created.
```

1. **the last number can be use for auto_assig is 111**

```
SQL>
SQL> SELECT auto_assign_id, staff_id, task_id, frequency, aa_status
  2    FROM Auto_Assign
  3   WHERE staff_id = 'SF0001'
  4     AND task_id = 'TK0001'
  5     AND frequency = 'Daily';

AUTO_A STAFF_ TASK_I FREQUENCY  AA_STATUS
------ ------ ------ ---------- --------------------
AA0111 SF0001 TK0001 Daily      Active
```

**1. it is added and auto_assign_id is successfully assign to 111**

```
SQL> INSERT INTO Auto_Assign
  2    (staff_id, task_id, frequency, set_date, start_time, end_ti
me, aa_status)
  3  VALUES
  4    ('SF0002',                        -- staff_id
  5     'TK0003',                        -- task_id
  6     'Weekly',                        -- frequency
  7     SYSDATE,                         -- set_date
  8     SYSDATE + interval '9' hour,    -- start_time (example)
  9     SYSDATE + interval '12' hour,   -- end_time (example)
 10     'Active'                         -- aa_status
 11    );

1 row created.

SQL>
SQL> COMMIT;

Commit complete.

SQL> SELECT auto_assign_id, staff_id, task_id, frequency, aa_statu
s
  2    FROM Auto_Assign
  3   WHERE staff_id = 'SF0002'
  4     AND task_id = 'TK0003'
  5     AND frequency = 'Weekly';

AUTO_A STAFF_ TASK_I FREQUENCY  AA_STATUS
------ ------ ------ ---------- --------------------
AA0112 SF0002 TK0003 Weekly     Active

SQL>
```

1. **check and try another one and it is successful**

**Trigger 1: Automatic Staff ID Generation Trigger**

**Purpose: This trigger ensures that every new staff record in the Staff table receives a unique and properly formatted staff_id. When no ID is provided during insertion, it generates an ID automatically using the staff_seq sequence, prefixing it with 'SF' and padding it to four digits (e.g., SF0001). This maintains consistency and avoids manual errors in assigning staff IDs.**

**Trigger code:**

```
-- Auto-Increment for staff_id
CREATE OR REPLACE TRIGGER staff_before_insert
                                                BEFORE INSERT ON Staff
                                                          FOR EACH ROW
                                                                  BEGIN

                                              IF :NEW.staff_id IS NULL THEN

SELECT 'SF' || LPAD(staff_seq.NEXTVAL, 4, '0') INTO :NEW.staff_id FROM
dual;
                                                                    END IF;

END;
/
```

**Sample Input:**

```
INSERT INTO Staff
  (name, gender, ic_no, email, tel_no, address, position,
reg_date, stf_status)
VALUES
  ('Jeanny', 'F', '040721120631', 'jeanny@email.com',
'011-1234567',
   '1 Test Address', 'Tester', SYSDATE, 'Active');
```

**Sample Output:**

```
SQL> -- check the last staff number
SQL> SELECT sequence_name, last_number
  2    FROM user_sequences
  3   WHERE sequence_name = 'STAFF_SEQ';

SEQUENCE_NAME                  LAST_NUMBER
------------------------------ -----------
STAFF_SEQ                               18

SQL>
SQL> INSERT INTO Staff
  2    (name, gender, ic_no, email, tel_no, address, position, reg
_date, stf_status)
  3  VALUES
  4    ('Jeanny', 'F', '040721120631', 'jeanny@email.com', '011-12
34567',
  5     '1 Test Address', 'Tester', SYSDATE, 'Active');

1 row created.
```

1. check the last number of staff id that can be use

```
SQL> SELECT staff_id, name, stf_status
  2    FROM Staff
  3   WHERE name = 'Jeanny';

STAFF_ NAME                                                STF_STAT
US
------ -------------------------------------------------- --------
------------
SF0018 Jeanny                                              Active
```

1. the last staff_id is assigned to the newest insertion of staff

### 5.4.5 User Defined Exception

**Exception 1: Generic Error Handling with Rollback in Schedule Assignment**

**Purpose: This exception handling mechanism in the prc_Assign_Manual_Schedule procedure serves as a safeguard. If any errors arise during the insertion of records into the Schedule or Task_Progress tables, the system will roll back all changes made since the defined before_schedule_insert savepoint. This ensures that incomplete or incorrect data is not committed, preserving the integrity of the transaction. A generic error message is also generated, including the specific Oracle error, to assist in troubleshooting.**

**Exception code:**

```
BEGIN
    SAVEPOINT before_schedule_insert; -- Define savepoint before DML
```

```
    COMMIT;
EXCEPTION
    WHEN OTHERS THEN -- Catch any other error during INSERTs or before
COMMIT
        ROLLBACK TO before_schedule_insert; -- Revert changes since
the savepoint
        DBMS_OUTPUT.PUT_LINE('Error assigning schedule: ' || SQLERRM);
END;
/
```

## 5.5 Daniel Tay Yi Fung

### 5.5.1 Views

**View 1: Staff Attendance Status**

**Purpose:**

**This view summarizes each staff's attendance status (e.g., Present, Absent, In Progress) and counts how many days they were in each status. The accompanying query specifically shows how many days each staff was marked "Present".**

**View Code:**

```
CREATE OR REPLACE VIEW vw_staff_attendance_summary AS

SELECT s.staff_id, s.name, a.att_status, COUNT(*) AS total_days

FROM Staff s

JOIN Attendance a ON s.staff_id = a.staff_id

GROUP BY s.staff_id, s.name, a.att_status;
```

**Output:**

```
SELECT staff_id, name, SUM(CASE WHEN att_status = 'Present' THEN total_days
ELSE 0 END) AS present_days

FROM vw_staff_attendance_summary

GROUP BY staff_id, name;
```

**View 2: Staff Attendance Compliance Rate**

**Purpose:**

**This view calculates the attendance compliance rate as a percentage for each staff member (e.g., 85.7% attendance). The query converts raw data into an easy-to-read performance metric.**

**View Code:**

```
CREATE OR REPLACE VIEW vw_attendance_compliance AS

SELECT staff_id, COUNT(CASE WHEN att_status = 'Present' THEN 1 END) AS
present_days,

COUNT(*) AS total_days

FROM Attendance

GROUP BY staff_id;
```

**Output:**

```
SELECT staff_id, ROUND((present_days / total_days) * 100, 2) AS
attendance_rate

FROM vw_attendance_compliance;
```

### 5.5.2 Indexes

**Index 1: Index Staff Attendance Status**

**Purpose:**

**This index improves the performance of queries that filter or group attendance records based on their status (e.g., Present, Absent, Leave). Since attendance reports and compliance checks frequently use the att_status field in WHERE conditions and aggregations, this index allows the database to quickly locate and retrieve relevant records without scanning the entire table.**

**SQL Statement:**

```
CREATE INDEX idx_staff_status ON Staff(stf_status);
```

**Index 2: Index Schedule Staff by ID**

**Purpose:**

**This index enhances the speed of queries that filter, join, or search schedules by staff_id. Since the Schedule table is often queried to list all assignments, shifts, or tasks for a particular staff member, this index ensures efficient access to the related schedule data and improves join performance with the Staff table.**

**SQL Statement:**

```
CREATE INDEX idx_schedule_staff ON Schedule(staff_id);
```

### 5.5.3 User Defined Functions

**Procedure 1: Mark Staff as Inactive**

**Purpose:**

**Performs a *soft delete* by marking staff as Inactive instead of physically deleting them — preserving historical data and maintaining referential integrity in related tables like Attendance, Schedule, and Task_Progress.**

**Procedure Code:**

```
CREATE OR REPLACE PROCEDURE sp_soft_delete_staff(
  p_staff_id IN VARCHAR2
)
IS
BEGIN
  UPDATE Staff
  SET stf_status = 'Inactive'
  WHERE staff_id = p_staff_id;

  IF SQL%ROWCOUNT = 0 THEN
                    RAISE_APPLICATION_ERROR(-20002, 'Staff ID not found.');
  END IF;
END;
/
```

**Procedure 2: Assign Tasks Automatically**

**Purpose:**

**Automates task assignment to staff by generating a new schedule_id and inserting a schedule entry with start, end, and attend times. Reduces human error and manual entry in task assignments.**

**Procedure Code:**

```
CREATE OR REPLACE PROCEDURE sp_assign_task(

  p_staff_id IN VARCHAR2,

  p_task_id IN VARCHAR2,

  p_schedule_id OUT VARCHAR2

)

IS

BEGIN

  SELECT 'SC' || LPAD(TO_CHAR(NVL(MAX(TO_NUMBER(SUBSTR(schedule_id, 3))),0)
+ 1), 4, '0')

  INTO p_schedule_id

  FROM Schedule;



  INSERT INTO Schedule(schedule_id, task_id, staff_id, start_datetime,
end_datetime, attend_datetime)

  VALUES (p_schedule_id, p_task_id, p_staff_id, SYSDATE, SYSDATE + 1,
SYSDATE);

END;

/
```

**Procedure 3: Calculation of the Total Working Hours for a Staff**

**Purpose:**

**Calculates the total working hours for a staff member based on their attendance records. Useful for generating payroll reports, performance tracking, or compliance checks.**

**Procedure Code:**

```
CREATE OR REPLACE PROCEDURE sp_total_hours_worked(

  p_staff_id IN VARCHAR2,

  p_total_hours OUT NUMBER

)

IS

BEGIN

  SELECT NVL(SUM(
```

```
                                    (checkout_datetime - checkin_datetime) * 24
  ),0)
  INTO p_total_hours
  FROM Attendance
  WHERE staff_id = p_staff_id;
END;
/
```

**Procedure 4: Total Completed Tasks Report**

**Purpose:**

**Generates a report listing each staff member and their total completed tasks, useful for supervisor review, KPI evaluations, and performance-based bonuses.**

**Procedure Code:**

```
CREATE OR REPLACE PROCEDURE sp_generate_staff_performance
IS
  CURSOR staff_cur IS
          SELECT s.staff_id, s.stf_name, COUNT(tp.task_id) AS completed_tasks
FROM Staff s
LEFT JOIN Task_Progress tp ON s.staff_id = tp.staff_id
AND tp.tp_status = 'Completed'
GROUP BY s.staff_id, s.stf_name;

  v_row staff_cur%ROWTYPE;
BEGIN
  OPEN staff_cur;
  LOOP
FETCH staff_cur INTO v_row;
EXIT WHEN staff_cur%NOTFOUND;
DBMS_OUTPUT.PUT_LINE('Staff: ' || v_row.stf_name || ', Completed Tasks: ' ||
v_row.completed_tasks);
  END LOOP;
  CLOSE staff_cur;
END;
/
```

## 5.5.4 Sequence & Trigger

**Sequence & Trigger 1: Auto-Increment Staff ID**

**Purpose:**

**For the sequence part, it generates a unique, incrementing number for each new Staff record. Ensures no duplicate IDs and automates primary key generation. For the trigger part, it automatically assigns a new staff_id to the inserted staff record before the row is saved — only if the staff_id is not provided manually. The trigger formats the ID as 'SF' + 4-digit padded number (e.g., SF0001, SF0002, etc.).**

**Sequence & Trigger Code:**

```
CREATE SEQUENCE staff_seq

START WITH 1

INCREMENT BY 1

NOCACHE

NOCYCLE;


-- Auto-Increment for staff_id

CREATE OR REPLACE TRIGGER staff_before_insert

                                                        BEFORE INSERT ON Staff

                                                                FOR EACH ROW

                                                                        BEGIN

                                                    IF :NEW.staff_id IS NULL THEN

       SELECT 'SF' || LPAD(staff_seq.NEXTVAL, 4, '0') INTO :NEW.staff_id FROM
dual;

                                                                            END IF;

END;
```

### 5.5.5 User Defined Exceptions

**Exception 1: Exception Handling when Inserting Staff Attendance**

**Purpose:**

**This procedure ensures safe and validated insertion of staff attendance records into the system.**

**It uses custom exceptions, PRAGMA exception binding, and validation logic to guarantee:**

1. **Staff Existence Validation:**
   **Prevents inserting attendance for staff members who do not exist in the Staff table.**

2. **Logical Time Validation:**
   **Prevents check-in times that are later than check-out times, ensuring accurate attendance data.**

3. **Unique Constraint Handling:**
   **Handles duplicate attendance records gracefully using PRAGMA**

**EXCEPTION_INIT for database constraint violations.**

4. **Error Transparency:**
   **Converts low-level database errors into clear, meaningful messages for users or developers via RAISE_APPLICATION_ERROR.**

**Exception Handling Code:**

```
CREATE OR REPLACE PROCEDURE sp_insert_attendance(
  p_staff_id IN VARCHAR2,
  p_checkin IN DATE,
  p_checkout IN DATE
)
IS
  -- Custom User-Defined Exception
  ex_staff_not_found EXCEPTION;
  ex_invalid_time EXCEPTION;

  -- PRAGMA EXCEPTION_INIT for a DB error (example: unique violation)
  ex_unique_violation EXCEPTION;
  PRAGMA EXCEPTION_INIT(ex_unique_violation, -00001);

  v_count NUMBER;
BEGIN
  -- Validate if staff exists
  SELECT COUNT(*) INTO v_count FROM Staff WHERE staff_id = p_staff_id;
  IF v_count = 0 THEN
                                                    RAISE ex_staff_not_found;
  END IF;

  -- Validate time logic
  IF p_checkin > p_checkout THEN
                                                        RAISE ex_invalid_time;
  END IF;

  -- Insert Attendance
  INSERT INTO Attendance(attendance_id, staff_id, checkin_datetime,
checkout_datetime, att_status)
```

```
  VALUES (SEQ_ATTENDANCE_ID.NEXTVAL, p_staff_id, p_checkin, p_checkout,
'Present');


EXCEPTION

  -- Custom User Exception Handling

  WHEN ex_staff_not_found THEN

          RAISE_APPLICATION_ERROR(-20010, 'Error: Staff ID does not exist.');


  WHEN ex_invalid_time THEN

        RAISE_APPLICATION_ERROR(-20011, 'Error: Check-in time cannot be after
check-out time.');


  -- PRAGMA EXCEPTION_INIT Handling

  WHEN ex_unique_violation THEN

      RAISE_APPLICATION_ERROR(-20012, 'Error: Duplicate Attendance Record.');


  -- Default Oracle Exceptions

  WHEN OTHERS THEN

            RAISE_APPLICATION_ERROR(-20999, 'Unexpected error: ' || SQLERRM);

END;

/
```