UNIVERSITI TEKNOLOGI MALAYSIA

SCST1143-15 KEJURUTERAAN PENGKALAN DATA

(DATABASE ENGINEERING)

SESSION 2025/2026 – SEMESTER 1

Group 5

(Laksa - Micro:bit IoT Device Loan Management System)

Group Members

MD MAHEYAN ISLAM (A25MJ4015)

MD FOYSAL (A25MJ4014)

MAHMUDUL HOQUE SHARIF (A25MJ4012)

FAIAZ NAZEEF (A25MJ4009)

MD MRIDUL HASAN EMON (A25MJ4016)

PROJECT REPORT: MICRO:BIT LOAN MANAGEMENT SYSTEM

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our lecturer for the guidance and support provided throughout the course **SCST1143 Database Engineering**. This project provided us with a valuable opportunity to apply theoretical database concepts to a real-world scenario. We also thank the lab assistants and our group members for their cooperation and hard work in completing this project.

ABSTRACT

The **Micro:bit Loan Management System** is a relational database solution designed to automate the borrowing and inventory tracking of IoT devices within the faculty. Currently, the manual process of recording loans leads to data redundancy, inconsistency, and difficulty in tracking overdue items. This project aims to resolve these issues by implementing a centralized database using **MySQL**. The system handles student registration, device inventory, loan transactions, reservations, and fine calculations. The project follows the complete Database Systems Development Life Cycle (DSDLC), including conceptual modeling (ERD), logical design, normalization (up to 3NF/BCNF), and SQL implementation. The final system ensures data integrity, reduces administrative workload, and provides accurate reporting for the faculty staff.

TABLE OF CONTENTS

1.0 PART A: DATABASE PLANNING AND SYSTEM DEFINITION

1.1 System Overview (Current System & Problem Identification)

Currently, the Micro:bit loan management process in the faculty is primarily manual or semi-automated, relying on physical logbooks or disconnected spreadsheet files. When a student needs a device, they approach the staff, who manually checks the inventory. **Key Problems Identified:**

- **Data Redundancy:** Student information is repeated for every transaction.
- **Inconsistency:** Device statuses (e.g., 'GOOD' vs 'DAMAGED') are recorded inconsistently.
- **Slow Retrieval:** Searching for a specific device's history is time-consuming.
- **Missing Audit Trail:** It is difficult to track which staff member approved a specific loan.

1.2 Mission Statement

The mission of the **Micro:bit Loan Management System** is to design and implement a robust, centralized relational database that automates the borrowing lifecycle of IoT devices. It aims to ensure accountability, streamline inventory tracking, and provide a transparent reservation experience.

1.3 Mission Objectives

The database aims to achieve the following measurable objectives:

1. **Inventory Management:** To maintain real-time records of all devices, models, and their physical condition.
2. **Loan Automation:** To efficiently process loans with automatic tracking of due dates and fines.

3. **Reservation System:** To allow students to place priority-based reservations for specific models.

4. **Maintenance Tracking:** To schedule and track maintenance windows, ensuring broken items are not loaned out.

1.4 System Definition

- **Inside the System:** Student profiles, Staff records, Device inventory, Loan transactions, Reservations, Fines, and Maintenance logs.

- **Outside the System:** External financial payment gateways and the central university student database (integration not included).

- **Main Functions:** Registration, Device Search, Loan Approval, Reservation, Returns, and Reporting.

1.5 Major User Views and Access Matrix

The system identifies four key user roles:

User Role	**User View Description**	**Access Rights**
Administrator	Full system control	Maintain (CRUD) All Data
Librarian	Inventory & Reporting	Maintain Inventory, View Reports
Assistant	Counter operations	Process Loans, Returns, Reservations
Student	End-user	Read Profile, Request Reservation

2.0 PART B: DATABASE DESIGN REQUIREMENTS

2.1 Data Requirements (Entities & Attributes)

The following key entities have been identified:

Entity	Primary Key	Key Attributes	Validation Rules
Student	student_id	matric_no, email, status	Unique Email/Matric; Status: Active/Suspended
Staff	staff_id	staff_no, role	Role: Admin/Librarian/Assistant
Device	device_id	asset_tag, serial_no, condition	Condition: New/Good/Damaged
Loan	loan_id	loaned_at, due_at, status	Due date > Loan date
Reservation	reservation_id	priority, status	Priority: 1-10

2.2 Transaction Requirements (CRUD Analysis)

- **Create:** Register Students, Add Devices, Open Loans, Make Reservations.
- **Read:** Check Device Availability, View Loan History, Generate Reports.
- **Update:** Close Loans (Return), Update Device Condition, Cancel Reservations.
- **Delete:** Restricted (Soft delete preferred) to maintain audit trails.

2.3 Cross-Reference Analysis

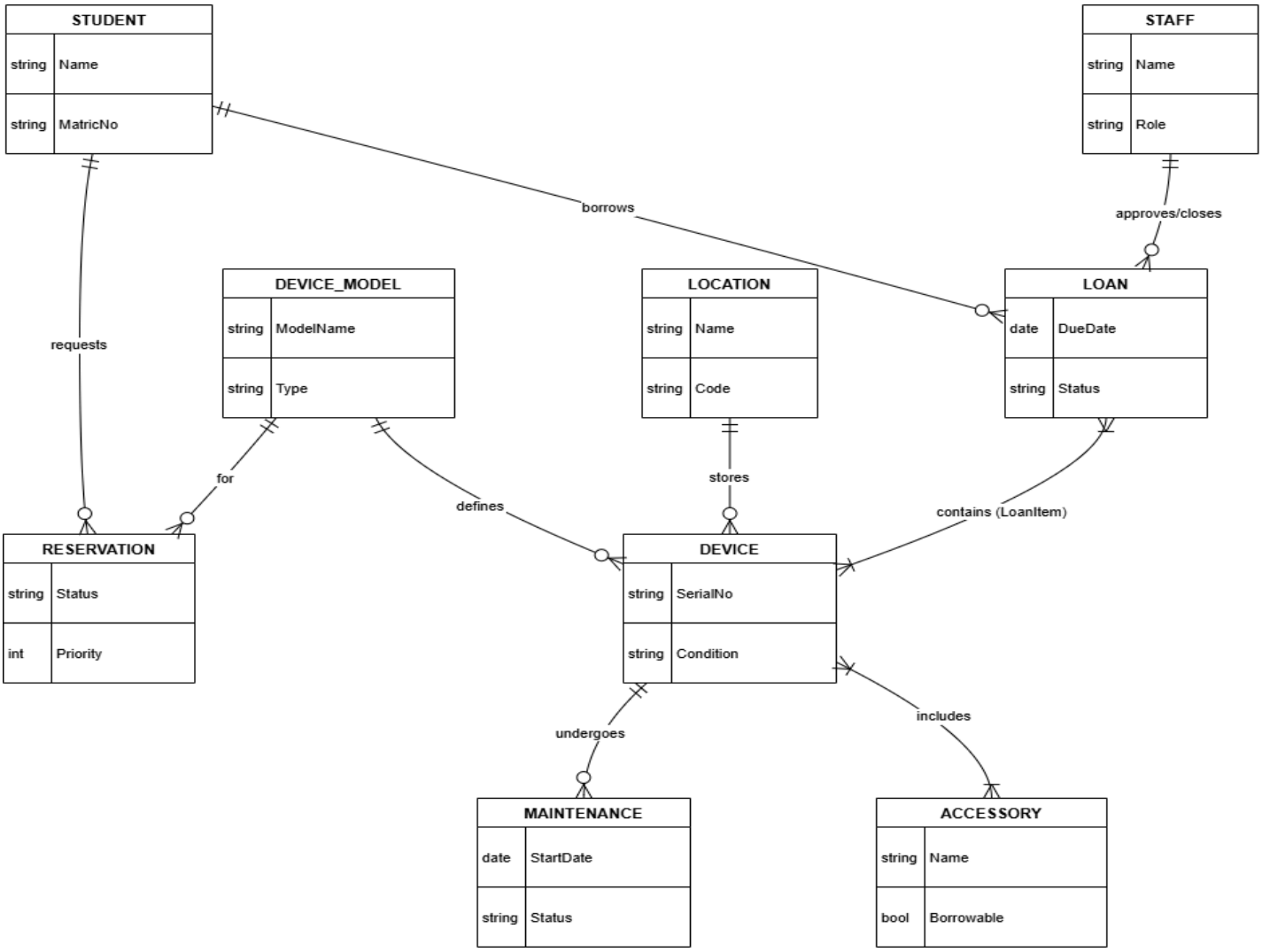
(Refer to the Access Matrix in Section 1.5 for the mapping between User Views and Data Entities).

User Role ↓ / Data →	**Student Data**	**Device Inventory**	**Loan Records**	**Reservations**	**Staff/Admin Data**
Administrator	Maintain (CRUD)	Maintain (CRUD)	Maintain (CRUD)	Maintain (CRUD)	Maintain (CRUD)
Librarian	Query / Report	Maintain (CRUD)	Query / Report	Query / Update	No Access
Assistant	Query / Update	Query / Update	Create / Update	Process / Update	No Access
Student	Read Own Profile	Query Only	Read Own History	Create / Read Own	No Access

3.0 PART C: CONCEPTUAL DATABASE DESIGN

3.1 Conceptual Entity Relationship Diagram (ERD)

The Conceptual ERD illustrates the relationships between Student, Staff, Device, Loan, and Reservation entities.



3.2 Data Dictionary (Conceptual)

The following data dictionary describes the attributes for the **9 entities** identified in the Conceptual ERD.

Table 1: STUDENT

Attribute Name	Type	Description
student_id	Integer	Unique internal identifier for the student.
matric_no	Text	Official university matriculation number (Unique).
full_name	Text	Full legal name of the student.
email	Text	University email address (Unique).
phone	Text	Contact phone number.
status	Text	Current standing (Active/Suspended).

Table 2: STAFF

Attribute Name	Type	Description
staff_id	Integer	Unique internal identifier for the staff.
staff_no	Text	Official employee ID number.
full_name	Text	Name of the staff member.
email	Text	Official staff email address.
role	Text	Job role (Admin, Librarian, Assistant).

Table 3: DEVICE MODEL

Attribute Name	Type	Description
model_id	Integer	Unique ID for the model type.
model_name	Text	Name of the maker (e.g., BBC).
manufacturer	Text	Name of the maker (e.g., BBC).
kit_type	Text	Category of the kit (Starter, IoT, etc.).
max_loan_days	Integer	Standard loan duration allowed for this model.

Table 4: DEVICE

Attribute Name	Type	Description
device_id	Integer	Unique system ID for the physical item.
asset_tag	Text	University inventory tag number.
serial_no	Text	Manufacturer's serial number.
condition	Text	Physical state (New, Good, Damaged).
is_active	Boolean	Availability status (Yes/No).

Table 5: LOCATION

Attribute Name	Type	Description
location_id	Integer	Unique ID for the storage location.
name	Text	Name of the lab or room.
description	Text	Details about the storage shelf/area.

Table 6: RESERVATION

Attribute Name	Type	Description
reservation_id	Integer	Unique ID for the booking request.
requested_at	DateTime	When the reservation was placed.
priority	Integer	Urgency level (1-10).

status	Text	Request status (Pending, Fulfilled).

Table 7: LOAN

Attribute Name	Type	Description
loan_id	Integer	Unique transaction ID for the loan.
loaned_at	DateTime	Date and time items were borrowed.
due_at	DateTime	Expected return date.
status	DateTime	Expected return date.

Table 8: MAINTENANCE

Attribute Name	Type	Description
maintenance_id	Integer	Unique ID for maintenance record.
maint_start	DateTime	Start date of maintenance.
maint_end	DateTime	Expected completion date.
status	Text	Progress (Scheduled, In Progress, Completed).

Table 9: ACCESSORY

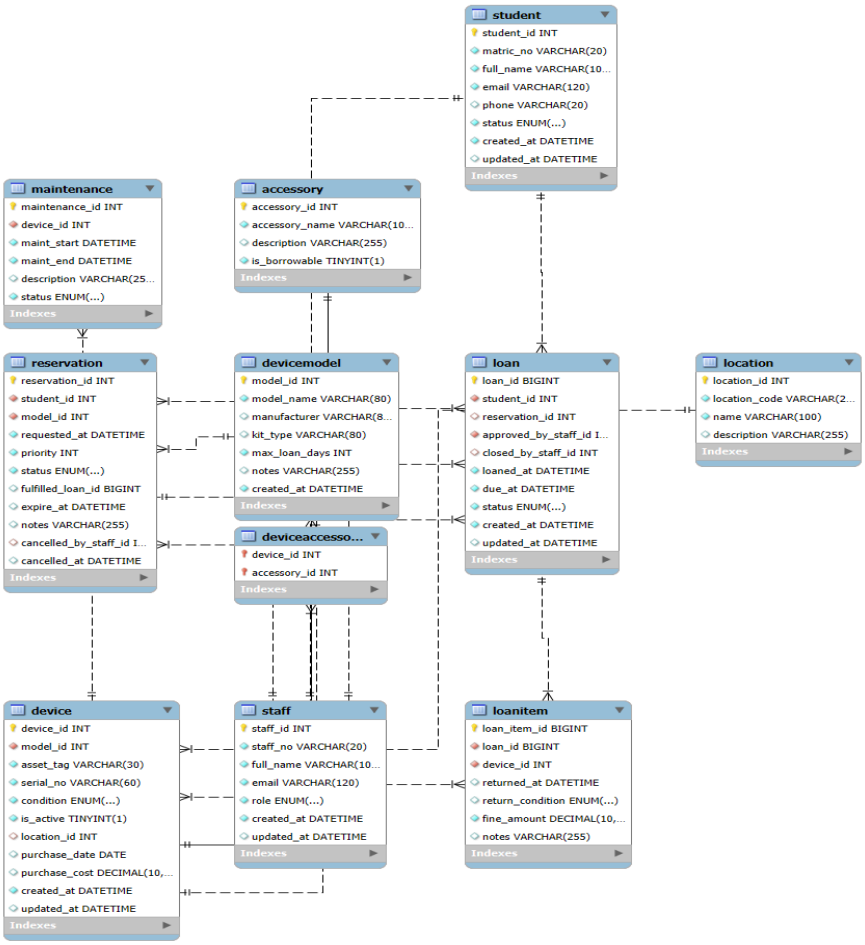
Attribute Name	Type	Description
accessory_id	Integer	Unique ID for the accessory.
accessory_name	Text	Name of the accessory (e.g., USB Cable).
is_borrowable	Boolean	Can this item be borrowed individually?

4.0 PART D: LOGICAL DATABASE DESIGN

4.1 Logical ERD (Relational Schema)

The Conceptual ERD has been refined into a Logical Relational Schema. In this stage, all Many-to-Many (M:N) relationships were resolved (specifically between Device and Accessory), and Foreign Keys (FK) were assigned to establish referential integrity.

Relational Schema Representation:



4.2 Updated Data Dictionary

The data dictionary is updated to reflect the final schema, specifically adding the **Associative Table** (DeviceAccessory) and the **Accessory** entity which were finalized during the logical design to resolve the M:N relationship.

Table: Accessory

Attribute Name	Data Type	Key	Constraints	Description
accessory_id	INT	PK	Auto Inc	Unique ID for the accessory type.
accessory_name	VARCHAR(100)		Unique	Name (e.g., USB Cable, Battery Pack).
is_borrowable	BOOLEAN		Default TRUE	Flag indicating if it can be loaned separately.

Table: DeviceAccessory (Associative Table)

Attribute Name	Data Type	Key	Constraints	Description
device_id	INT	PK, FK	Ref (Device)	Maps to the specific parent device.
accessory_id	INT	PK, FK	Ref (Accessory)	Maps to the included accessory type.
accessory_id	INT	PK, FK	Ref (Accessory)	Maps to the included accessory type.

4.3 Normalization Process

To ensure data consistency and eliminate redundancy, the database schema was designed by following the normalization process from Unnormalized Form (UNF) to Third Normal Form (3NF) and BCNF.

Step 1: Unnormalized Form (UNF)

In a manual logbook system, data is often recorded in a flat structure with repeating groups:

Loan_Log(student_name, matric_no, staff_name, loan_date, {device_model, serial_no, fine}, due_date)

- **Problem:** The list of devices {device_model, serial_no...} repeats within a single loan transaction.

Step 2: First Normal Form (1NF)

- **Rule:** Eliminate repeating groups and ensure atomicity.
- **Action:** Separate the repeating device details from the loan header information into a child table.
- **Resulting Schema:**
    1. Loan_Header (loan_id, student_info, staff_info, loan_date, due_date)
    2. Loan_Detail (loan_id, device_info, serial_no, model_info)

Step 3: Second Normal Form (2NF)

- **Rule:** Eliminate Partial Dependencies (Non-key attributes must depend on the *whole* Primary Key).
- **Analysis:** In the Loan_Detail table, the Primary Key is composite: (loan_id, device_id).
    - Attributes like Model_Name or Manufacturer depend only on device_id (the specific item), not on the loan_id.
- **Action:** Move device-specific static data to a separate Device table.
- **Resulting Schema:**
    1. Loan (loan_id, student_info...)

2. Device (device_id, serial_no, model_info...)
3. LoanItem (loan_item_id, loan_id, device_id, return_info) $\leftarrow$ *Links Loan and Device*

Step 4: Third Normal Form (3NF)

- **Rule:** Eliminate Transitive Dependencies (Non-key attributes depending on other non-key attributes).
- **Analysis:**
  - In the Loan table: student_name depends on matric_no (or student_id), not on the loan transaction itself.
  - In the Device table: Manufacturer and Kit_Type depend on Model_Name, not on the specific physical serial_no.
- **Action:** Create separate look-up tables for entities referenced by ID.
- **Final 3NF Schema:**
  - **Student Table:** Stores student details, referenced by student_id.
  - **DeviceModel Table:** Stores model specifications, referenced by model_id.
  - **Device Table:** Stores physical instance data, linked to model_id.
  - **Loan Table:** Stores transaction headers, linked to student_id.

Boyce-Codd Normal Form (BCNF)

The final schema adheres to BCNF because every determinant (e.g., student_id, model_id, loan_id) is a candidate key for its respective relation. There are no functional dependencies where a part of a composite key depends on a non-key attribute.

5.0 PART E: IMPLEMENTATION

5.1 Database Implementation (DDL)

The database schema was implemented using MySQL. Below are the DDL commands used to create the core entities with Primary Keys (PK), Foreign Keys (FK), and constraints.

-- 1. Table: Student

```
CREATE TABLE Student (
  student_id INT AUTO_INCREMENT PRIMARY KEY,
  matric_no  VARCHAR(20)  NOT NULL,
  full_name  VARCHAR(100) NOT NULL,
  email  VARCHAR(120) NOT NULL,
  phone              VARCHAR(20)  NULL,
  status  ENUM('ACTIVE','SUSPENDED') NOT NULL DEFAULT 'ACTIVE',
  CONSTRAINT uq_student_matric_no UNIQUE (matric_no),
  CONSTRAINT uq_student_email        UNIQUE (email)
) ENGINE=InnoDB;
```

-- 2. Table: Device

```
CREATE TABLE Device (
  device_id   INT AUTO_INCREMENT PRIMARY KEY,
  model_id        INT NOT NULL,
  asset_tag   VARCHAR(30) NOT NULL,
  serial_no   VARCHAR(60) NOT NULL,
  `condition` ENUM('NEW','GOOD','FAIR','DAMAGED','LOST') NOT NULL DEFAULT 'GOOD',
  is_active   BOOLEAN NOT NULL DEFAULT TRUE,
  location_id INT NULL,
  CONSTRAINT fk_device_model FOREIGN KEY (model_id) REFERENCES DeviceModel(model_id)
) ENGINE=InnoDB;
```

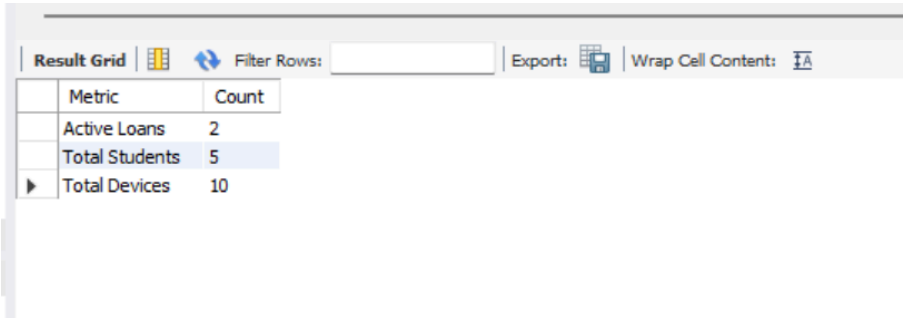
47	00:15:04	SELECT d.device_id, d.serial_no, d.asset_tag, dm.model_name, d.`condition`, d.is_active, loc.name AS locatio...	9 row(s) returned	0.000 sec / 0.000 sec
48	00:15:04	SELECT r.reservation_id, s.full_name AS student_name, dm.model_name AS device_model, r.requested_at, r.priority...	8 row(s) returned	0.000 sec / 0.000 sec
49	00:15:04	SELECT dm.model_name, COUNT(d.device_id) AS total_devices FROM DeviceModel dm LEFT JOIN Device d ON dm.mo...	7 row(s) returned	0.000 sec / 0.000 sec
50	00:15:04	SELECT l.loan_id, s.full_name AS student_name, l.loaned_at, l.due_at, l.status FROM Loan l JOIN Student s ON l.st...	2 row(s) returned	0.000 sec / 0.000 sec
51	00:15:04	SELECT s.full_name AS student_name, COUNT(DISTINCT li.loan_item_id) AS devices_on_loan FROM Loan l JOIN LoanIt...	1 row(s) returned	0.000 sec / 0.000 sec
52	00:15:04	SELECT s.full_name AS student_name, SUM(li.fine_amount) AS total_fines FROM Student s JOIN Loan l ON s.student_i...	3 row(s) returned	0.000 sec / 0.000 sec
53	00:15:04	SELECT dm.model_name, COUNT(li.loan_item_id) AS times_loaned FROM LoanItem li JOIN Device d ON li.device_id = ...	6 row(s) returned	0.000 sec / 0.000 sec
54	00:15:04	SELECT s.full_name AS student_name, d.serial_no AS device_serial, dm.model_name AS device_type, l.due_at, st....	3 row(s) returned	0.000 sec / 0.000 sec
55	00:15:04	SELECT s.full_name AS student_name, GROUP_CONCAT(DISTINCT d.serial_no ORDER BY d.serial_no SEPARATOR ', '...	1 row(s) returned	0.000 sec / 0.000 sec
56	00:15:04	SELECT d.serial_no AS item_name FROM Device d UNION SELECT a.accessory_name AS item_name FROM Accessory ...	14 row(s) returned	0.000 sec / 0.000 sec
57	00:15:04	SELECT d.serial_no FROM Device d LEFT JOIN LoanItem li ON d.device_id = li.device_id WHERE li.device_id IS NULL LIM...	2 row(s) returned	0.000 sec / 0.000 sec
58	00:15:04	SELECT s.full_name, SUM(li.fine_amount) AS total_fines FROM Student s JOIN Loan l ON s.student_id = l.student_id JO...	3 row(s) returned	0.000 sec / 0.000 sec
59	00:15:04	SELECT dm.model_name AS device_type, loc.name AS location_name FROM DeviceModel dm CROSS JOIN Location ...	13 row(s) returned	0.000 sec / 0.000 sec
60	00:15:04	SELECT d.serial_no AS serial, dm.model_name AS device_type, GROUP_CONCAT(a.accessory_name ORDER BY a.ac...	10 row(s) returned	0.000 sec / 0.000 sec
61	00:15:04	SELECT d.serial_no, l.loan_id, m.maintenance_id, l.loaned_at, l.due_at, m.maint_start, m.maint_end FROM Loa...	2 row(s) returned	0.000 sec / 0.000 sec
62	00:15:04	SELECT st.full_name AS staff_name, COUNT(*) AS loan_actions FROM Staff st LEFT JOIN (SELECT approved_by_staf...	5 row(s) returned	0.000 sec / 0.000 sec
63	00:15:04	SELECT dm.model_name AS device_type, li.return_condition, AVG(li.fine_amount) AS avg_fine, COUNT(*) AS total_ret...	1 row(s) returned	0.000 sec / 0.000 sec

5.2 Sample Data (DML) & Verification

Sample data was inserted into all tables to verify the relationships between Student, Staff, Device, and Loans.

Verification Query: To prove the database is populated, a statistical summary query is executed using UNION ALL.

```
SELECT 'Total Students' AS Metric, COUNT(*) AS Count FROM Student
UNION ALL
SELECT 'Total Devices', COUNT(*) FROM Device
UNION ALL
SELECT 'Active Loans', COUNT(*) FROM Loan WHERE status = 'OPEN';
```



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Metric	Count
Active Loans	2
Total Students	5
Total Devices	10

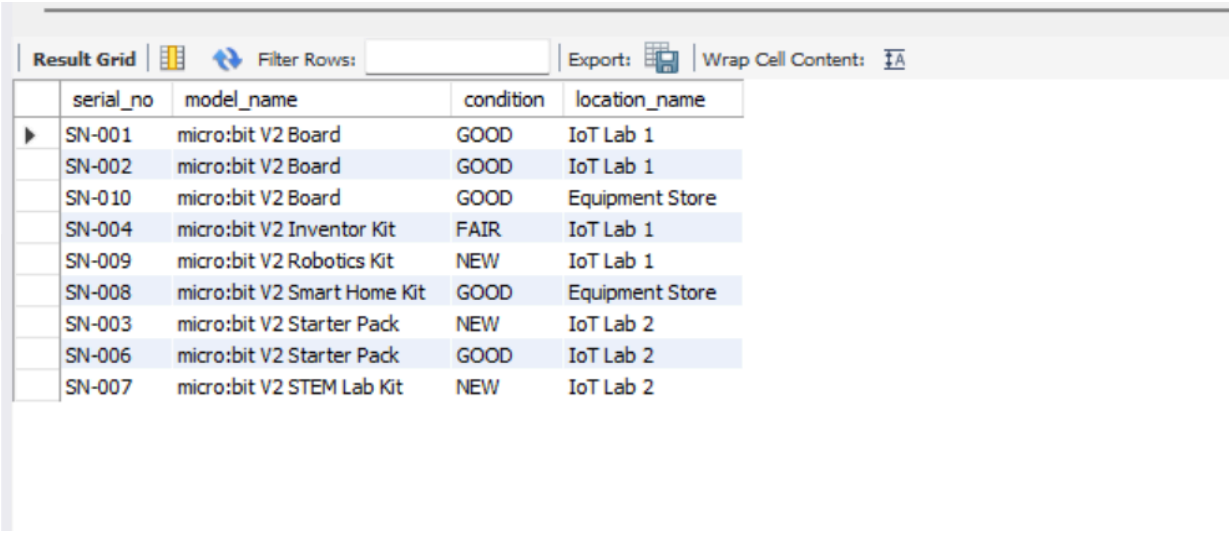
5.3 SQL Query Tasks + Outputs

The following SQL queries demonstrate the system's capability to answer complex business questions required by the stakeholders.

Query 1: List all active devices with their model and location

This query joins the Device, DeviceModel, and Location tables to show available inventory.

```
SELECT
    d.serial_no,
    dm.model_name,
    d.`condition`,
    loc.name AS location_name
FROM Device d
JOIN DeviceModel dm ON d.model_id = dm.model_id
LEFT JOIN Location loc ON d.location_id = loc.location_id
WHERE d.is_active = TRUE;
```

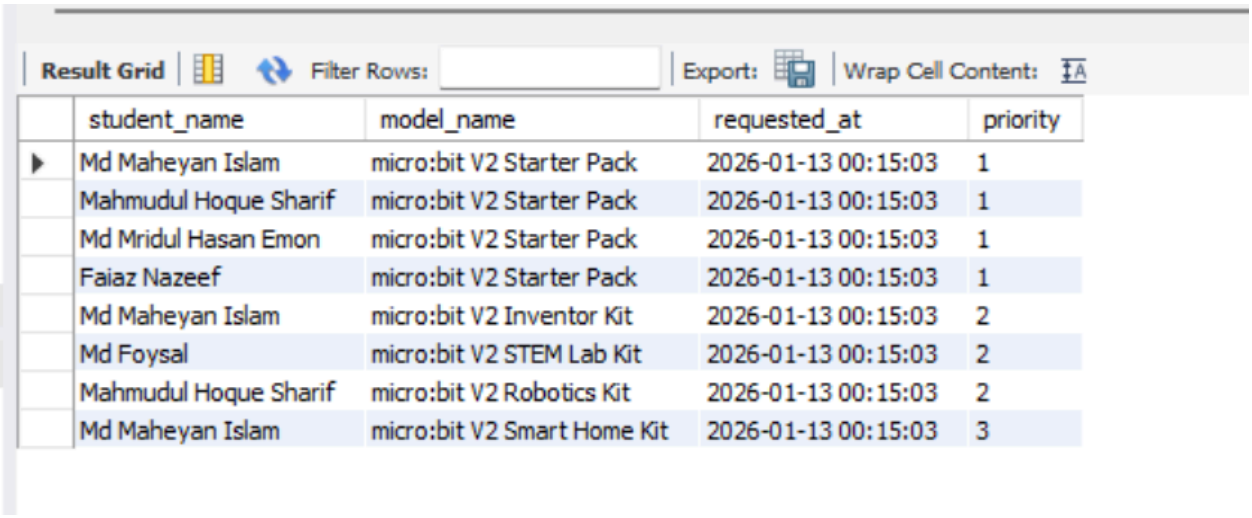


serial_no	model_name	condition	location_name
SN-001	micro:bit V2 Board	GOOD	IoT Lab 1
SN-002	micro:bit V2 Board	GOOD	IoT Lab 1
SN-010	micro:bit V2 Board	GOOD	Equipment Store
SN-004	micro:bit V2 Inventor Kit	FAIR	IoT Lab 1
SN-009	micro:bit V2 Robotics Kit	NEW	IoT Lab 1
SN-008	micro:bit V2 Smart Home Kit	GOOD	Equipment Store
SN-003	micro:bit V2 Starter Pack	NEW	IoT Lab 2
SN-006	micro:bit V2 Starter Pack	GOOD	IoT Lab 2
SN-007	micro:bit V2 STEM Lab Kit	NEW	IoT Lab 2

Query 2: Show pending reservations with priority

This query helps staff identify which students are waiting for devices based on priority.

```
SELECT
        s.full_name AS student_name,
        dm.model_name,
        r.requested_at,
        r.priority
FROM Reservation r
JOIN Student s ON r.student_id = s.student_id
JOIN DeviceModel dm ON r.model_id = dm.model_id
WHERE r.status = 'PENDING'
ORDER BY r.priority ASC;
```



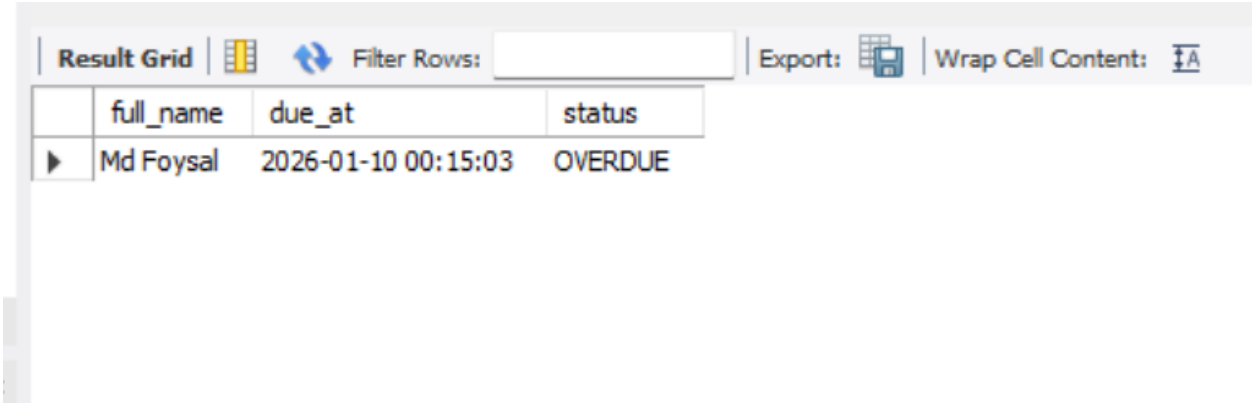
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

student_name	model_name	requested_at	priority
Md Maheyan Islam	micro:bit V2 Starter Pack	2026-01-13 00:15:03	1
Mahmudul Hoque Sharif	micro:bit V2 Starter Pack	2026-01-13 00:15:03	1
Md Mridul Hasan Emon	micro:bit V2 Starter Pack	2026-01-13 00:15:03	1
Faiaz Nazeef	micro:bit V2 Starter Pack	2026-01-13 00:15:03	1
Md Maheyan Islam	micro:bit V2 Inventor Kit	2026-01-13 00:15:03	2
Md Foysal	micro:bit V2 STEM Lab Kit	2026-01-13 00:15:03	2
Mahmudul Hoque Sharif	micro:bit V2 Robotics Kit	2026-01-13 00:15:03	2
Md Maheyan Islam	micro:bit V2 Smart Home Kit	2026-01-13 00:15:03	3

Query 3: List students with overdue loans

This query identifies students who have not returned devices by the due date.

```
SELECT
        s.full_name,
        l.due_at,
        l.status
FROM Loan l
JOIN Student s ON l.student_id = s.student_id
WHERE l.status = 'OVERDUE';
```

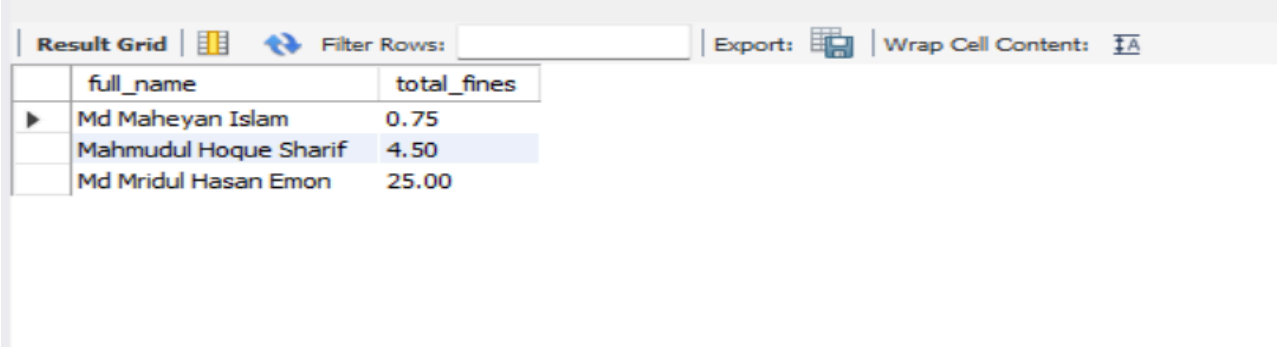


full_name	due_at	status
Md Foysal	2026-01-10 00:15:03	OVERDUE

Query 4: Calculate total fines for each student

This aggregation query calculates the outstanding fine amount for students.

```
SELECT
        s.full_name,
        SUM(li.fine_amount) AS total_fines
FROM Student s
JOIN Loan l ON s.student_id = l.student_id
JOIN LoanItem li ON l.loan_id = li.loan_id
GROUP BY s.student_id, s.full_name
HAVING total_fines > 0;
```

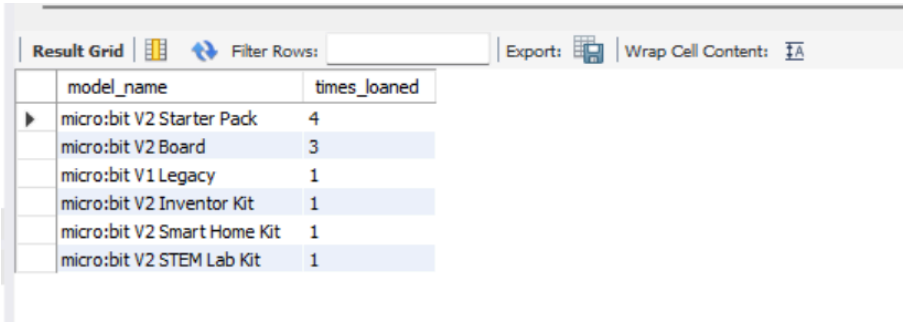


full_name	total_fines
Md Maheyan Islam	0.75
Mahmudul Hoque Sharif	4.50
Md Mridul Hasan Emon	25.00

Query 5: Find the most frequently loaned device models

This helps the admin decide which models to purchase more of in the future.

```
SELECT
        dm.model_name,
        COUNT(li.loan_item_id) AS times_loaned
FROM LoanItem li
JOIN Device d ON li.device_id = d.device_id
JOIN DeviceModel dm ON d.model_id = dm.model_id
GROUP BY dm.model_name
ORDER BY times_loaned DESC;
```



model_name	times_loaned
micro:bit V2 Starter Pack	4
micro:bit V2 Board	3
micro:bit V1 Legacy	1
micro:bit V2 Inventor Kit	1
micro:bit V2 Smart Home Kit	1
micro:bit V2 STEM Lab Kit	1

6.0 F. Relational Algebra Expressions

Q1. Retrieve all tuples from the User/Student relation.

To view the complete list of registered students.

- Expression:

Student (Alternatively: _true(Student)

Q2. Retrieve specific attributes (ID, Name, Status) from the User relation.

To view only the matriculation number, name, and current status of students.

- Expression:

Π(matric_no,full_name,status(STUDENT)

Q3. Select items (Devices) that satisfy a particular condition (e.g., Active/Good).

Transaction relation = Loan, date attribute = loaned_at

Expression:

σ_{loaned_at ≥ d1 ∧ loaned_at ≤ d2}(Loan)

Q4. Retrieve transactions (Loans) within a specified date range.

To find loans created between date d_1 and d_2.

- Expression:

σloaned_at ≥d1∧loaned_at≤d2

Q5. Identify incomplete transactions (Missing return date).

In our schema, “completion” is per device via **LoanItem.returned_at**.

Expression:

σ_{returned_at IS NULL}(LoanItem)

Q6. Combine Users and Transactions to display user info with transaction details.

Students + Loans:

Expression:

Student ⋈_{Student.student_id = Loan.student_id} Loan

Q7. Retrieve combined information showing which user is associated with which item.

Student → Loan → LoanItem → Device (and optionally DeviceModel).

Expression:

(Student ⋈*{Student.student_id = Loan.student_id} Loan)* ⋈{Loan.loan_id = LoanItem.loan_id} LoanItem ⋈_{LoanItem.device_id = Device.device_id} Device

Q8. Determine the total number of transactions performed by each user.

Count loans per student:

Expression:

γ_{student_id; COUNT(loan_id) → total_loans}(Loan)

Q9. Identify users whose number of transactions exceeds a given threshold (e.g., > 5).

Let:

$T = \gamma_{student\_id;\ COUNT(loan\_id) \rightarrow total\_loans}(Loan)$

Expression:

$\sigma_{total\_loans > k}(T)$

Q10. Compute how many times each item (Device) has been involved in a transaction.

Count how often each device appears in **LoanItem**:

Expression:

$\gamma_{device\_id;\ COUNT(loan\_item\_id) \rightarrow times\_in\_transactions}(LoanItem)$

Q11. Identify items that have never appeared in any transaction.

All devices minus loaned devices:

Expression:

$\Pi_{device\_id}(Device) - \Pi_{device\_id}(LoanItem)$

Q12. Display inventory movement (Maintenance) records together with item details.

Our schema doesn't have a separate "movement/audit" table like Movement/Log.

Closest equivalent for "audit over time" is **Maintenance** (device events) with **Device** details.

Expression:

$Maintenance \bowtie_{Maintenance.device\_id = Device.device\_id} Device$

Q13. Retrieve identifiers of items that have experienced a specific movement (e.g., 'IN_PROGRESS').

We don't have a Movement table with OUT/IN. In your system, "OUT" naturally maps to **being loaned out**.

So: devices that appear in LoanItem (or appear in open/overdue loans).

Expression: (devices that have been loaned out at least once):

Π_{device_id}(LoanItem)

Expression: (currently OUT = in OPEN/OVERDUE and not returned):

Π_{device_id}(
σ_{(Loan.status='OPEN' ∨ Loan.status='OVERDUE') ∧ LoanItem.returned_at IS NULL}
(Loan ⋈_{Loan.loan_id=LoanItem.loan_id} LoanItem)
)

Q14. Produce a unified list of staff records from different relations.

We store all roles in one table (**Staff.role**). So the "unified list" is already:

Expression:

Staff

If your instructor expects a union form, you can express it as:

Π_{staff_id, staff_no, full_name, email, role}(σ_{role='ADMIN'}(Staff))
∪ Π_{staff_id, staff_no, full_name, email, role}(σ_{role='LIBRARIAN'}(Staff))
∪ Π_{staff_id, staff_no, full_name, email, role}(σ_{role='ASSISTANT'}(Staff))

Q15. Retrieve user-item associations from an associative (bridge) relation.

Our many-to-many is: **LoanItem** bridges **Loan** and **Device**, and Loan links to Student.

So user–item associations = Student ↔ Device through Loan + LoanItem.

- Expression:

(Student ⋈*{Student.student_id = Loan.student_id} Loan)* ⋈{Loan.loan_id = LoanItem.loan_id} LoanItem ⋈_{LoanItem.device_id = Device.device_id} Device