Data Nexus

Group 18

ZAHRA KARA – 41330595

ANGELINA RAMSUNAR – 41081269

DARIAN SCHREUDER – 43552595

ANDREW NARE – 31506534

RIYA PATEL – 41914228

ROBERT FERREIRA – 33343136

MUHAMMED CAJEE – 43496385

Table of Contents

Project Phase 1: Database Initial Study

1. Members of the group:

ZAHRA BASHIR KARA

Group Leader

Student Number: 41330595

Contact Info: 076 874 9355

ANGELINA JAEDENE RAMSUNAR

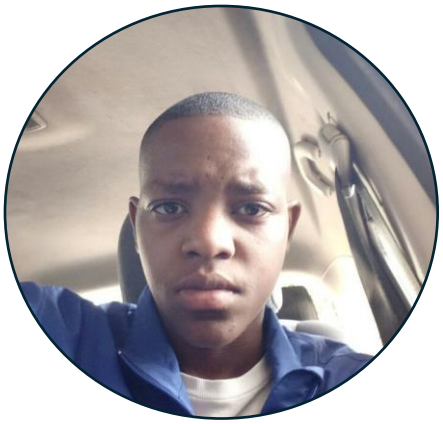
Student Number: 41081269

Contact Info: 071 162 4919

DARIAN SCHREUDER

Student number: 43552595

Contact Info: 076 548 4123



ANDREW NARE

Student number: 31506534

Contact Info: 065 910 6522

RIYA RAMESH PATEL

Student number: 41914228

Contact Info: 084 811 2525

ROBERT FERREIRA

Student number: 33343136

Contact Info: 076 020 0509

MUHAMMED CAJEE

Student number: 43496385

Contact Info: 081 516 1077

2. Analyse Company Situation:

Company Situation:

Lexicon is a tutoring business offered at the Potchefstroom North West University. It caters tuition to students across all academic years, including both undergraduate and postgraduate students. Tutors are appointed based on the students' modules and year of study. Over the past few years lexicon as seen a significant growth in its demand across students, which is why they have decided to from paper-based operations to a database system.

Objectives:

1. Implement a digital system to make administrative tasks quicker and more efficient.
2. Make it easier for students to schedule appointments.
3. Make it easier to track student appointments, transactions, and available tutoring venues.
4. Create a platform to accommodate the rapid growth of new clientele and business operations.

Operations:

- **Transaction management:** tracking financial transactions, recording payments made, managing invoices.
- **Appointment Scheduling:** matching students to tutors based on their module and year of study.
- **Venue Management:** Booking appointments at appropriate venues inside the university based on availability.
- **Administrative Tasks:** Such as keeping records, simplifying paperwork, and managing data.

Structure:

- **CEO:** Oversees business operations.
- **Operations Manager:** Oversees the day-to-day operations, including the database systems implementation as well as administrative tasks.
- **Tutors:** provide the students with personalised tutoring sessions to help them excel academically.
- **Receptionist**: manages the scheduling of appointments, payments, and record-keeping.

3. Define Problems and Constraints.

Problems and Constraints:

1. The manual process of keeping track of the business on paper causes both errors and delays.
2. Difficulty in managing the paperwork and administrative workload that staff members must deal with as the business expands.
3. Due to the lack of a system, it is difficult to track student appointments, venue details, and transactions.
4. Business is limited to how much they can grow due to their reliance on paper-based operations.

4. Database System Specification.

Objectives to Solve Identified Tasks.

1. Simplify the administrative tasks to reduce errors and delays.
2. Improve the efficiency of appointment scheduling for both students and tutors.
3. Improve the way that student appointments, transactions, and venues are tracked.
4. Allow for scalability to meet the expansion of the business.

Information Required from Database:

1. **Tutor Information**: Name, contact information, expertise, availability.
2. **Student Information:** name, contact information, year of study, modules.
3. **Appointment Details:** venue, date, time, duration, student- tutor pairing.
4. **Transaction Records:** payments made, invoices.
5. **Venue Details:** location, availability, session schedule.

Scope:

The system includes the following operational requirements:

1. A centralised place of storage for all student and tutor information.
2. Appointment scheduling that is automated based on the student's academic year and modules.
3. Efficient tracking of transactions, including payments recorded and invoices generated.
4. Functionalities for venue coordination to make planning a session easier.

Boundaries:

1. **Budget:** the cost of developing and implementing the system will be within the predefined budget constraints.
2. **Personnel:** limited to existing staff for maintenance and system development.
3. **Hardware:** use existing hardware infrastructure with upgrades if it deemed necessary.
4. **Software:** should be compatible with existing software systems and should take licensing agreements into consideration.

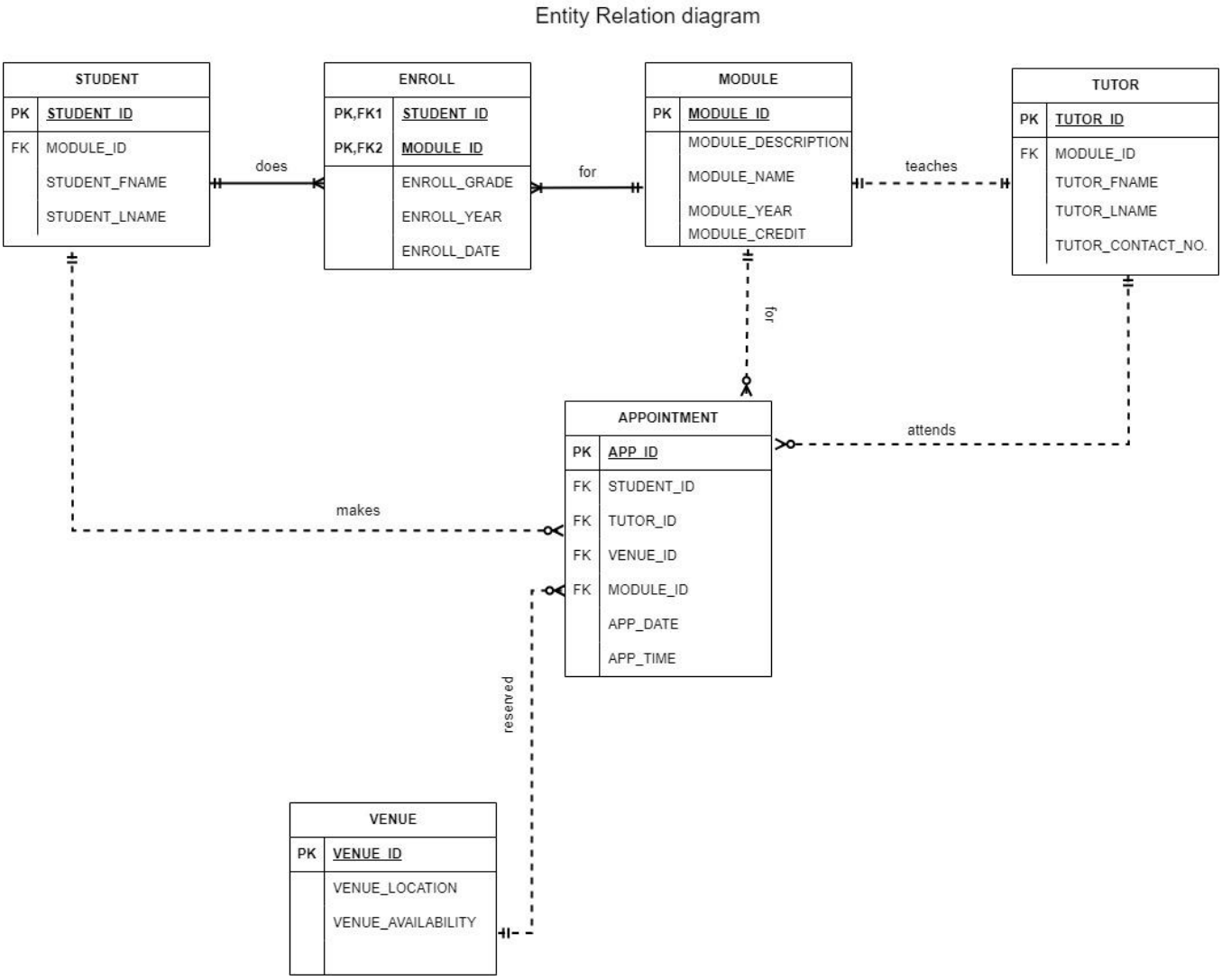
Project Phase 2: Database Design

1. Conceptual Design

Business Rules

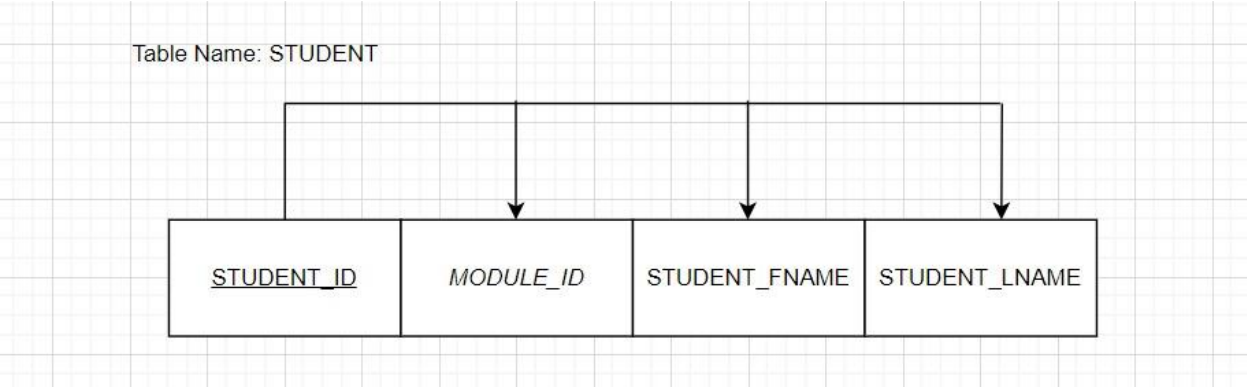
1. An appointment has only one tutor. Each tutor can have none or many appointments. (0:M)
2. Each appointment takes place in one venue. Each venue can have none or many appointments. (0:M)
3. Each appointment is made for only one module. One module can have none or many appointments. (0:M)
4. A tutor can only tutor one module. Each module has only one tutor tutoring it. (1:1)
5. A student can book many appointments. Each appointment is booked for only one student. (1:M)
6. Students do enrolment one or many times. Each enrolment is for one student. (1:M)
7. Each module can be enrolled one to many times. Each enrolment is for one module. (1:M)

ER Diagram

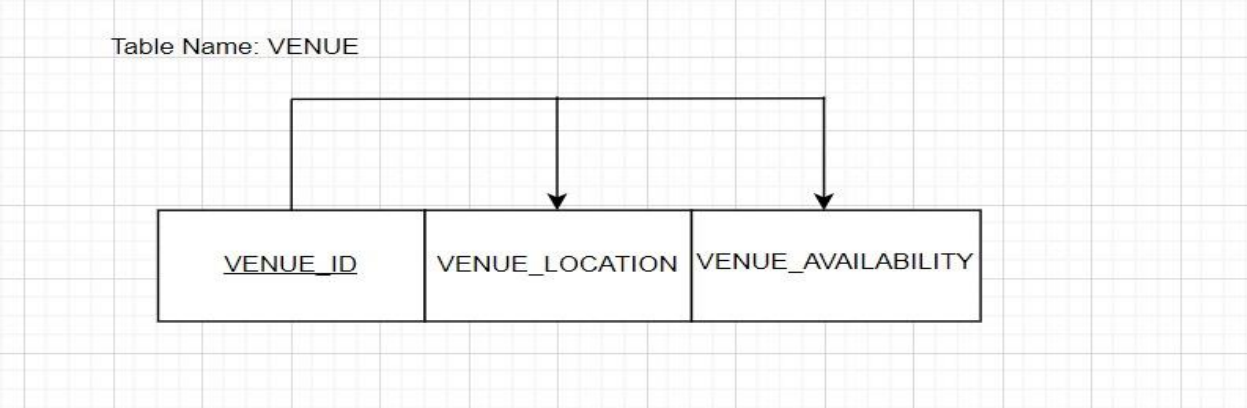

Entity Relation diagram
STUDENT
PK STUDENT_ID
FK MODULE_ID
STUDENT_FNAME
STUDENT_LNAME
does
ENROLL
PK,FK1 STUDENT_ID
PK,FK2 MODULE_ID
ENROLL_GRADE
ENROLL_YEAR
ENROLL_DATE
for
MODULE
PK MODULE_ID
MODULE_DESCRIPTION
MODULE_NAME
MODULE_YEAR
MODULE_CREDIT
teaches
TUTOR
PK TUTOR_ID
FK MODULE_ID
TUTOR_FNAME
TUTOR_LNAME
TUTOR_CONTACT_NO.
for
APPOINTMENT
PK APP_ID
FK STUDENT_ID
FK TUTOR_ID
FK VENUE_ID
FK MODULE_ID
APP_DATE
APP_TIME
attends
makes
reserved
VENUE
PK VENUE_ID
VENUE_LOCATION
VENUE_AVAILABILITY

2. Logical Design

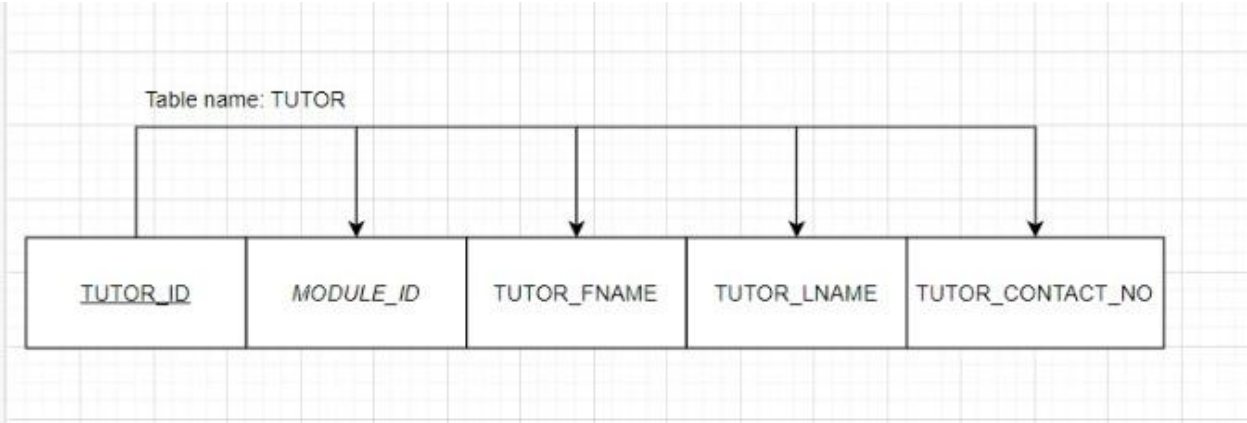
STUDENT (**STUDENT_ID(PK),** MODULE_ID(FK), STUDENT_FNAME, STUDENT_ LNAME)



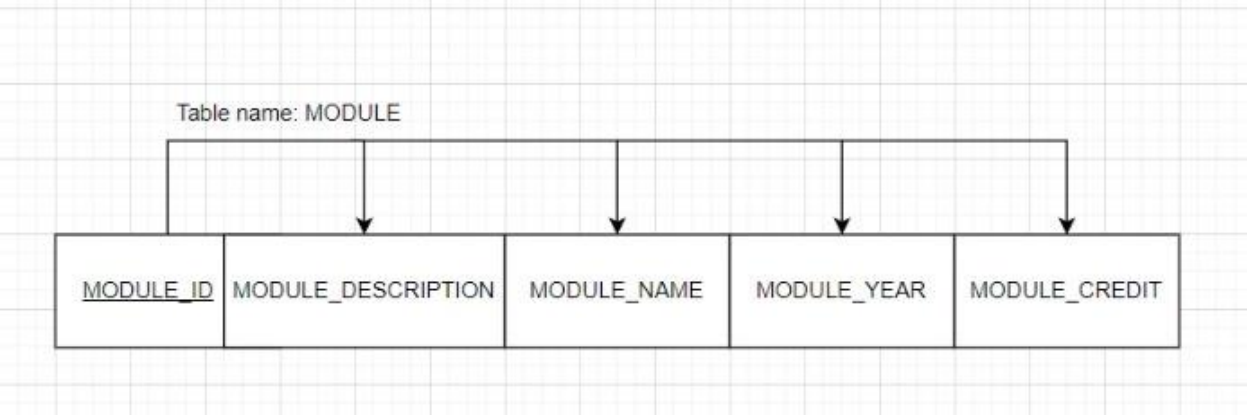
VENUE (**VENUE_ID(PK),** VENUE_LOCATION, VENUE_AVAILABILITY)



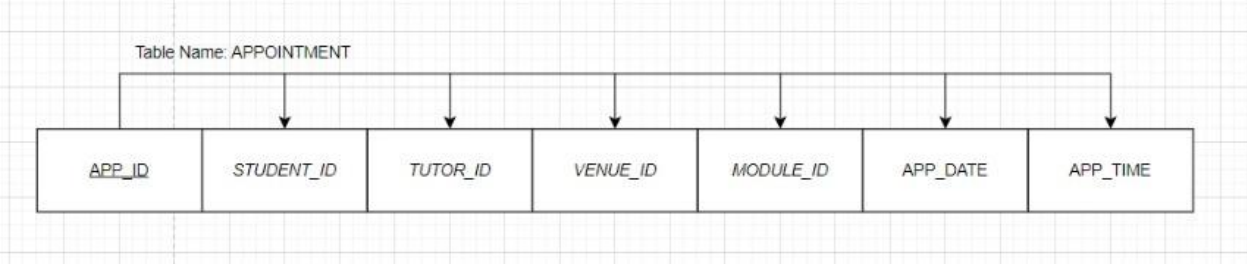
TUTOR (**TUTOR_ID (PK),** *MODULE_CODE (FK),* TOTUR_LNAME, TUTOR_FNAME, TUTOR_CONTACT_NO)



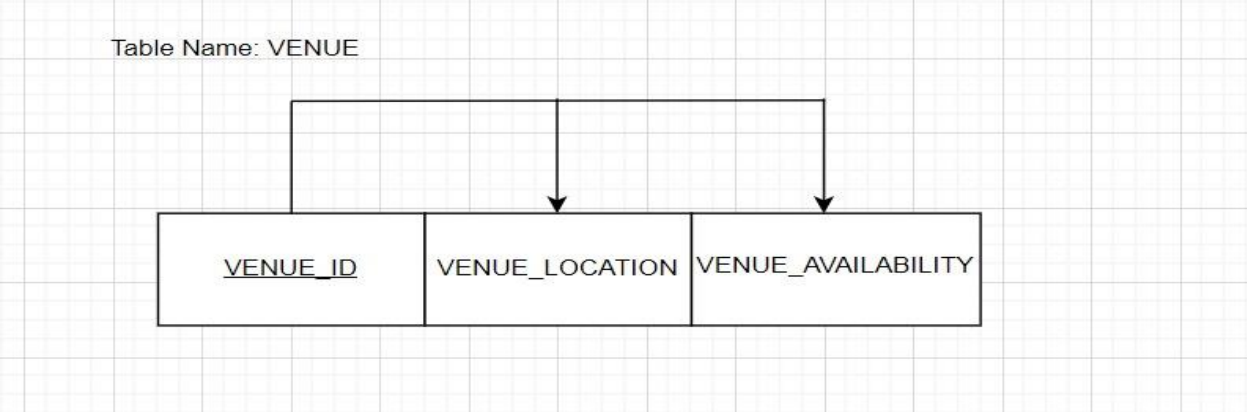
MODULE (**<u>MODULE_CODE (PK),</u>** MODULE_DESCRIPTION, MODULE_NAME, MODULE_YEAR, MODULE_CREDIT)



APPOINTMENT (**<u>APP_ID (PK),</u>** *STUDENT_ID (FK), TUTOR_ID(FK), VENUE_NR(FK),* DATE, TIME, *MODULE_CODE (FK)*)



VENUE (**<u>VENUE_ID (PK</u>**), VENUE_LOCATION, VENUE_AVAILABILITY)



Project Phase 3: Physical Design

1. Database objects

Removing all tables to allow the creation of new tables:

```
-----------------------------
---------DROP TABLES---------
-----------------------------
DROP TABLE STUDENT CASCADE CONSTRAINTS;
DROP TABLE TUTOR CASCADE CONSTRAINTS;
DROP TABLE MODULES CASCADE CONSTRAINTS;
DROP TABLE APPOINTMENT CASCADE CONSTRAINTS;
DROP TABLE PAYMENT CASCADE CONSTRAINTS;
DROP TABLE SALARY CASCADE CONSTRAINTS;
DROP TABLE PAYMENT_APP CASCADE CONSTRAINTS;
DROP TABLE ENROLL CASCADE CONSTRAINTS;
DROP TABLE VENUE CASCADE CONSTRAINTS;
```

The "Drop table" part of the command specifies that you want to delete a table from the database. When you drop a table, all the data in the table, as well as the table structure, are removed. The "Cascade constraints" clause specifies that all referential integrity constraints that refer to the table should also be dropped.

1.1. Tables

1.1.1. Create STUDENT table.

```
---Student---
CREATE TABLE STUDENT (
STUDENT_ID NUMBER(11) PRIMARY KEY,
STUDENT_FNAME VARCHAR(50)NOT NULL,
STUDENT_LNAME VARCHAR(50),
START_DATE DATE
);
```

The table "STUDENT" is created to keep track of all the students who have signed up for tutoring. It keeps general information such as their names and the date they started.

1.1.2. Create MODULE table.

```
---Module---
CREATE TABLE MODULES (
MODULE_ID INT PRIMARY KEY,
MODULE_NAME VARCHAR(10)NOT NULL,
MODULE_DESCRIPTION VARCHAR(100),
MODULE_YEAR NUMBER(4),
MODULE_CREDIT NUMBER(2)
);
```

The table "MODULE" keeps record of all the available modules that the tutoring business has to offer. General information about module is stores, such as its credit and year.

1.1.3. Create TUTOR table.

```
---Tutor---
CREATE TABLE TUTOR (
TUTOR_ID NUMBER(8) PRIMARY KEY,
TUTOR_FNAME VARCHAR(50)NOT NULL,
TUTOR_LNAME VARCHAR(50),
TUTOR_CONTACT_NO VARCHAR(15)UNIQUE,
HIRE_DATE DATE,
MODULE_ID NUMBER(8),
FOREIGN KEY (MODULE_ID) REFERENCES MODULES(MODULE_ID)
);
```

The table "TUTOR" stores all the employed tutors information. Businesses can find tutors full names, contact information as well as the module they tutoring in this table.

1.1.4. Create ENROLL table.

```
---Enroll---
CREATE TABLE ENROLL (
STUDENT_ID NUMBER(8),
MODULE_ID NUMBER(8),
ENROLL_GRADE INT,
ENROLL_YEAR NUMBER(4),
ENROLL_DATE DATE,
PRIMARY KEY (STUDENT_ID, MODULE_ID),
FOREIGN KEY (STUDENT_ID) REFERENCES STUDENT(STUDENT_ID),
FOREIGN KEY (MODULE_ID) REFERENCES MODULES(MODULE_ID)
);
```

The table "ENROLL" is created to keep track of which modules, students signed up for. It store the students id as well as the module id for which the student needs tutoring.

1.1.5. Create APPOINTMENT table.

```
---Appointment---
CREATE TABLE APPOINTMENT (
APP_ID NUMBER(8) PRIMARY KEY,
STUDENT_ID NUMBER(11),
TUTOR_ID NUMBER(8),
VENUE_ID NUMBER(8),
MODULE_ID NUMBER(8),
APP_DATETIME DATE,
FOREIGN KEY (STUDENT_ID) REFERENCES STUDENT(STUDENT_ID),
FOREIGN KEY (TUTOR_ID) REFERENCES TUTOR(TUTOR_ID),
FOREIGN KEY (VENUE_ID) REFERENCES VENUE(VENUE_ID),
FOREIGN KEY (MODULE_ID) REFERENCES MODULES(MODULE_ID)
);
```

The table "APPOINTMENT" contains information about every appointment the business makes. Stores the student, tutor, module and venue ids as well as the date and time of the appointment.

1.1.6. Create SALARY table.

```
---Salary---
CREATE TABLE SALARY (
SALARY_ID NUMBER(8)PRIMARY KEY,
TUTOR_ID NUMBER(8),
AMOUNT NUMBER(5),
PAID_DATE DATE,
PAYSALARY_METHOD VARCHAR(50),
FOREIGN KEY (TUTOR_ID) REFERENCES TUTOR(TUTOR_ID)
);
```

The table "SALARY" shows all the payments made to the tutors at the ned of every month. A tutor makes R1000 per month the first year they employed, thereafter each year it increases with R500 a month.

1.1.7. Create PAYMENT table.

```
---Payment---
CREATE TABLE PAYMENT (
PAYMENT_ID NUMBER(4) PRIMARY KEY,
AMOUNT NUMBER(5),
PAYMENT_DATE DATE,
PAYMENT_METHOD VARCHAR(50)
);
```

The table "PAYMENT" stores information for each payment the student makes. It is R100 per appointment and a student can make multiple payments to pay of one appointment.

1.1.8. Create PAYMENT_APP table.

```
---Payment_app---
CREATE TABLE PAYMENT_APP (
PAYMENT_ID NUMBER(8),
APP_ID NUMBER(8),
STATUS VARCHAR(20),
PRIMARY KEY (PAYMENT_ID, APP_ID),
FOREIGN KEY (PAYMENT_ID) REFERENCES PAYMENT(PAYMENT_ID),
FOREIGN KEY (APP_ID) REFERENCES APPOINTMENT(APP_ID)
);
```

The table "PAYMENT_APP" is a bridge entity. This entity is between "APPPOINTMENT" and "PAYMENT". Multiple payments can be made for one appointment and each appointment must have at least one payment.

1.1.9. Create VENUE table.

```
---Venue---
CREATE TABLE VENUE (
VENUE_ID NUMBER(8) PRIMARY KEY,
VENUE_LOCATION VARCHAR(25),
VENUE_AVAILABILITY VARCHAR(3)
);
```

The table "VENUE" is created to store the location of all available tutoring venues. Venues are found on the NWU campus. Venues become unavailable when there is construction been done on that building.

1.2. Indexes

```
---Filters Students---
CREATE INDEX idx_student_start_date
ON STUDENT(START_DATE);
```

An index is created based on the date students started at the NWU.

```
---Manages appointments---
CREATE INDEX idx_appointment_app_datetime
ON APPOINTMENT(APP_DATETIME);
```

An index is created based on the date and time of every appointment.

1.3. Views

This view shows only the students who have enrolled into tutoring lessons. This view narrows the students down to only those who are enrolled in tutoring lessons.

```
-------------------------------
---SQL statement to show current students taking tutoring
CREATE OR REPLACE VIEW Current_Enrolled_Students AS
SELECT s.STUDENT_FNAME AS Student_First_Name, s.STUDENT_LNAME AS Student_Last_Name,
       m.MODULE_NAME, e.ENROLL_GRADE, e.ENROLL_YEAR, e.ENROLL_DATE
FROM STUDENT s
JOIN ENROLL e ON s.STUDENT_ID = e.STUDENT_ID
JOIN MODULES m ON e.MODULE_ID = m.MODULE_ID;
```

This view displays the full module name next to the tutors information instead of just the id. The business can now see exactly what module each tutor is responsible for.

```
---Showing all current tutors with module name---
CREATE OR REPLACE VIEW Current_Tutors AS
SELECT t.TUTOR_FNAME AS Tutor_First_Name, t.TUTOR_LNAME AS Tutor_Last_Name,
       t.TUTOR_CONTACT_NO AS Contact_Number, t.HIRE_DATE, m.MODULE_NAME
FROM TUTOR t
JOIN MODULES m ON t.MODULE_ID = m.MODULE_ID;
```

In this view all the current modules that business offers tutoring for can be seen.

```
---View all current modules---
CREATE OR REPLACE VIEW Current_Modules AS
SELECT module_name, module_description, module_year, module_credit
FROM MODULES;
```

In this view only the upcoming appointments are displayed. Helps tutors and business stay up to date with appointments easy to see which appointments are still coming up.

```
---Sql statement to create a table which shows upcoming appointments---
CREATE OR REPLACE VIEW Upcoming_Appointments AS
SELECT s.STUDENT_FNAME AS Student_First_Name, s.STUDENT_LNAME AS Student_Last_Name,
       t.TUTOR_FNAME AS Tutor_First_Name, t.TUTOR_LNAME AS Tutor_Last_Name,
       m.MODULE_NAME, v.VENUE_LOCATION, a.APP_DATETIME
FROM APPOINTMENT a
JOIN STUDENT s ON a.STUDENT_ID = s.STUDENT_ID
JOIN TUTOR t ON a.TUTOR_ID = t.TUTOR_ID
JOIN MODULES m ON a.MODULE_ID = m.MODULE_ID
JOIN VENUE v ON a.VENUE_ID = v.VENUE_ID
WHERE a.APP_DATETIME > CURRENT_TIMESTAMP;
```

This view displays how many appointments each venue has. This can help businesses see and assess which venue is most popular.

```
---View amount of appointments in a specific venue---
CREATE OR REPLACE VIEW Appointments_By_Venue AS
SELECT v.VENUE_LOCATION, COUNT(*) AS Number_of_Appointments
FROM APPOINTMENT a
JOIN VENUE v ON a.VENUE_ID = v.VENUE_ID
GROUP BY v.VENUE_LOCATION;
```

This view shows how many appointments each tutor has. Can see which modules require the most assistance based of the tutor with the most appointments.

```
---Counts number of appointments for each tutor---
CREATE OR REPLACE VIEW Tutor_Appointment_Count AS
SELECT t.TUTOR_ID, t.TUTOR_FNAME AS Tutor_First_Name, t.TUTOR_LNAME AS Tutor_Last_Name,
COUNT(a.APP_ID) AS Number_of_Appointments
FROM TUTOR t
LEFT JOIN APPOINTMENT a ON t.TUTOR_ID = a.TUTOR_ID
GROUP BY t.TUTOR_ID, t.TUTOR_FNAME, t.TUTOR_LNAME;
```

1.4. Data Loading

1.4.1. STUDENT

```
---Insert data into the STUDENT table---
INSERT INTO STUDENT (STUDENT_ID, STUDENT_FNAME, STUDENT_LNAME, START_DATE)
VALUES (1, 'John', 'Doe', TO_DATE('2024-02-10', 'YYYY-MM-DD'));
INSERT INTO STUDENT (STUDENT_ID, STUDENT_FNAME, STUDENT_LNAME, START_DATE)
VALUES (2, 'Jane', 'Smith', TO_DATE('2024-02-10', 'YYYY-MM-DD'));
INSERT INTO STUDENT (STUDENT_ID, STUDENT_FNAME, STUDENT_LNAME, START_DATE)
VALUES (3, 'Alice', 'Johnson', TO_DATE('2024-02-10', 'YYYY-MM-DD'));
```

1.4.2. MODULES

```
---Insert data into the MODULE table---
INSERT INTO MODULES (MODULE_ID, MODULE_NAME, MODULE_DESCRIPTION, MODULE_YEAR, MODULE_CREDIT)
VALUES (1, 'CMGPlll', 'Introduction to IT', 1, 12);
INSERT INTO MODULES (MODULE_ID, MODULE_NAME, MODULE_DESCRIPTION, MODULE_YEAR, MODULE_CREDIT)
VALUES (2, 'STTNlll', 'Statistics', 1, 16);
INSERT INTO MODULES (MODULE_ID, MODULE_NAME, MODULE_DESCRIPTION, MODULE_YEAR, MODULE_CREDIT)
VALUES (3, 'BMANlll', 'Business Management', 1, 12);
```

1.4.3. TUTOR

```
-- Insert data into the TUTOR table
INSERT INTO TUTOR (TUTOR_ID, TUTOR_FNAME, TUTOR_LNAME, TUTOR_CONTACT_NO, HIRE_DATE, MODULE_ID)
VALUES (1, 'Tom', 'Adams', '072-456-7890', TO_DATE('2023-02-10', 'YYYY-MM-DD'), 1);
INSERT INTO TUTOR (TUTOR_ID, TUTOR_FNAME, TUTOR_LNAME, TUTOR_CONTACT_NO, HIRE_DATE, MODULE_ID)
VALUES (2, 'Sarah', 'Baker', '036-567-8901', TO_DATE('2022-02-10', 'YYYY-MM-DD'), 2);
INSERT INTO TUTOR (TUTOR_ID, TUTOR_FNAME, TUTOR_LNAME, TUTOR_CONTACT_NO, HIRE_DATE, MODULE_ID)
VALUES (3, 'Jim', 'Clark', '082-678-9012', TO_DATE('2023-02-10', 'YYYY-MM-DD'), 3);
```

1.4.4. ENROLL

```
-- Insert data into the ENROLL table
INSERT INTO ENROLL (STUDENT_ID, MODULE_ID, ENROLL_GRADE, ENROLL_YEAR, ENROLL_DATE)
VALUES (1, 1, 45, 2024, TO_DATE('2024-03-15', 'YYYY-MM-DD'));
INSERT INTO ENROLL (STUDENT_ID, MODULE_ID, ENROLL_GRADE, ENROLL_YEAR, ENROLL_DATE)
VALUES (2, 2, 63, 2024, TO_DATE('2024-03-16', 'YYYY-MM-DD'));
INSERT INTO ENROLL (STUDENT_ID, MODULE_ID, ENROLL_GRADE, ENROLL_YEAR, ENROLL_DATE)
VALUES (3, 3, 38, 2024, TO_DATE('2024-03-17', 'YYYY-MM-DD'));
```

1.4.5. SALARY

```
-- Insert data into the SALARY table
INSERT INTO SALARY (SALARY_ID, TUTOR_ID, AMOUNT, PAID_DATE, PAYSALARY_METHOD)
VALUES (1, 1, 1500, TO_DATE('2024-04-01', 'YYYY-MM-DD'), 'Direct Deposit');
INSERT INTO SALARY (SALARY_ID, TUTOR_ID, AMOUNT, PAID_DATE, PAYSALARY_METHOD)
VALUES (2, 2, 2000, TO_DATE('2024-04-01', 'YYYY-MM-DD'), 'Bank Transfer');
INSERT INTO SALARY (SALARY_ID, TUTOR_ID, AMOUNT, PAID_DATE, PAYSALARY_METHOD)
VALUES (3, 3, 1000, TO_DATE('2024-04-01', 'YYYY-MM-DD'), 'Cash');
```

1.4.6. APPOINTMENT

```
-- Insert data into the APPOINTMENT table
INSERT INTO APPOINTMENT (APP_ID, STUDENT_ID, TUTOR_ID, VENUE_ID, MODULE_ID, APP_DATETIME)
VALUES (1, 1, 1, 1, 1, TO_DATE('2024-03-15 10:00', 'YYYY-MM-DD HH24:MI'));
INSERT INTO APPOINTMENT (APP_ID, STUDENT_ID, TUTOR_ID, VENUE_ID, MODULE_ID, APP_DATETIME)
VALUES (2, 2, 2, 3, 2, TO_DATE('2024-03-17 11:00', 'YYYY-MM-DD HH24:MI'));
INSERT INTO APPOINTMENT (APP_ID, STUDENT_ID, TUTOR_ID, VENUE_ID, MODULE_ID, APP_DATETIME)
VALUES (3, 3, 3, 2, 3, TO_DATE('2024-03-24 12:00', 'YYYY-MM-DD HH24:MI'));
```

1.4.7. PAYMENT

```
-- Insert data into the PAYMENT table
INSERT INTO PAYMENT (PAYMENT_ID, AMOUNT, PAYMENT_DATE, PAYMENT_METHOD)
VALUES (1, 100, TO_DATE('2024-03-16', 'YYYY-MM-DD'), 'Credit Card');
INSERT INTO PAYMENT (PAYMENT_ID, AMOUNT, PAYMENT_DATE, PAYMENT_METHOD)
VALUES (2, 100, TO_DATE('2024-03-24', 'YYYY-MM-DD'), 'Debit Card');
INSERT INTO PAYMENT (PAYMENT_ID, AMOUNT, PAYMENT_DATE, PAYMENT_METHOD)
VALUES (3, 100, TO_DATE('2024-03-25', 'YYYY-MM-DD'), 'Bank Transfer');
```

1.4.8. PAYMENT_APP

```
-- Load data into Payment_App table
INSERT INTO PAYMENT_APP (PAYMENT_ID, APP_ID, STATUS) VALUES (1, 1, 'Completed');
INSERT INTO PAYMENT_APP (PAYMENT_ID, APP_ID, STATUS) VALUES (2, 2, 'Completed');
INSERT INTO PAYMENT_APP (PAYMENT_ID, APP_ID, STATUS) VALUES (3, 3, 'Completed');
```

1.4.9. VENUE

```
-- Insert data into the VENUE table
INSERT INTO VENUE (VENUE_ID, VENUE_LOCATION, VENUE_AVAILABILITY) VALUES (1, 'Room 101', 'Yes');
INSERT INTO VENUE (VENUE_ID, VENUE_LOCATION, VENUE_AVAILABILITY) VALUES (2, 'Room 102', 'Yes');
INSERT INTO VENUE (VENUE_ID, VENUE_LOCATION, VENUE_AVAILABILITY) VALUES (3, 'Room 103', 'Yes');
INSERT INTO VENUE (VENUE_ID, VENUE_LOCATION, VENUE_AVAILABILITY) VALUES (4, 'Room 104', 'Yes');
INSERT INTO VENUE (VENUE_ID, VENUE_LOCATION, VENUE_AVAILABILITY) VALUES (5, 'Room 105', 'Yes');
```

2. Queries

2.1. Limitation of rows and columns

Due to the business only focusing on tutoring IT related modules the business therefore only employs up to 30 tutors at a time. This is done to avoid over employment. A trigger is created so that the table “TUTOR” can only take 30 entries.

```
-- Trigger to limit rows in TUTOR table
CREATE OR REPLACE TRIGGER TUTOR_ROW_LIMIT
BEFORE INSERT ON TUTOR
FOR EACH ROW
DECLARE
    v_count NUMBER;
    v_limit CONSTANT NUMBER := 30; -- Set your row limit here
BEGIN
    -- Count the current number of rows in the TUTOR table
    SELECT COUNT(*) INTO v_count FROM TUTOR;

    -- Check if the count exceeds the limit
    IF v_count >= v_limit THEN
        RAISE_APPLICATION_ERROR(-20001, 'Row limit exceeded for TUTOR table.');
    END IF;
END;
```

Businesses often require employees contact information in order for them to be able to communicate with them, With this query we able to get the full name as well as the contact information of every tutor.

```
---PROVIDE ONLY SPECIFIC INFO---
SELECT
    (t.TUTOR_FNAME || ' ' || t.TUTOR_LNAME) AS "Name",
    t.TUTOR_CONTACT_NO AS "Contact Number"
FROM
    TUTOR t
ORDER BY
    "Name" DESC;
```

2.2. Sorting

Businesses often need to keep reports on their employees and usually want these reports to appear neatly in alphabetical order.

```
---Sort students by last name---
SELECT *
FROM STUDENT
ORDER BY STUDENT_LNAME;
```

Student and Enrol table are joined to show all the information in both the tables. This helps the tutors see which students require more help during tutoring.

```
---Sort by enroll grade---
SELECT s.STUDENT_FNAME || ' ' || s.STUDENT_LNAME AS STUDENT_NAME, e.*
FROM ENROLL e
JOIN STUDENT s ON e.STUDENT_ID = s.STUDENT_ID
ORDER BY e.ENROLL_GRADE ASC;
```

2.3. LIKE, AND and OR.

This statement shows the all the values from the ENROLL table for student with a grade of less then 40%. This helps the business see which students require extra help and can thus offer additional assistance.

```
---Find students with grade of less than 40 for additional help
SELECT * FROM ENROLL
WHERE ENROLL_YEAR = 2024 AND ENROLL_GRADE < 40;
```

This statement shows all the values in the payment table for people who used “apple” or “google pay”. It can help businesses see which payment method is the most popular.

```
---Seeing who utilizes apple and google pay---
SELECT * FROM PAYMENT
WHERE PAYMENT_METHOD = 'PayPal' OR PAYMENT_METHOD = 'Google Pay';
```

2.4. Variable and Character Functions

This statement shows appointment information for a certain student. User is asked to provide the student_id. Helps business see all the appointment details of a certain student. Can help the business in reporting as well as assist in student enquiries.

```
---SHOW DETAILS APPOINTMENT DETAILS FOR CERTAIN STUDENT_ID
SELECT A.APP_ID,
       S.STUDENT_FNAME || ' ' || S.STUDENT_LNAME AS STUDENT_NAME,
       T.TUTOR_FNAME || ' ' || T.TUTOR_LNAME AS TUTOR_NAME,
       M.MODULE_NAME,
       V.VENUE_LOCATION AS VENUE_NAME,
       A.APP_DATETIME
FROM APPOINTMENT A
JOIN STUDENT S ON A.STUDENT_ID = S.STUDENT_ID
JOIN TUTOR T ON A.TUTOR_ID = T.TUTOR_ID
JOIN MODULES M ON A.MODULE_ID = M.MODULE_ID
JOIN VENUE V ON A.VENUE_ID = V.VENUE_ID
WHERE S.STUDENT_ID = &STUDENT_ID;
```

This query displays the students initials and their last name. Having a students initials and last name make communication with them much easier and more personal,

```
---STUDENTS LAST NAME AND INITIALS
SELECT
    CONCAT(SUBSTR(STUDENT_FNAME, 1, 1), SUBSTR(STUDENT_LNAME, 1, 1)) AS STUDENT_INITIALS,
    STUDENT_LNAME AS LAST_NAME
FROM STUDENT;
```

2.5. Round or Trunc

Calculates the average amount of salary paid monthly to tutors. Helps businesses know on average how much they spend on salaries per tutor each month.

```
---Rounding salary amount in salary table---
SELECT ROUND(AVG(AMOUNT), 2) AS Average_Salary
FROM SALARY;
```

Calculates the total amount received from students rounded off to give a whole number. This can help business manage its incoming.

```
---ROUNDING PAYMENT AMOUNT IN PAYMENT TABLE---
SELECT
    ROUND(SUM(AMOUNT), 2) AS Total_Paid
FROM
    PAYMENT;
```

2.6. Date Functions

Shows the date a tutor was hired tow work for the business. This can be used for promotional purposes as well as to give increases to tutors based on how many years they've been tutoring.

```
---YEAR HIRED---
SELECT TUTOR_FNAME,
       TO_CHAR(HIRE_DATE, 'DD-MON-YYYY') AS HIRE_DATE
FROM TUTOR;
```

Displays in weeks how long students have been receiving tutoring for the current year they enrolled in. Business can see how long a student has been receiving tutoring and can enquire if the student sees an improvement in their marks.

```
---LENGTH IN TUTORING FOR MODULE---
SELECT
    S.STUDENT_ID,
    S.STUDENT_FNAME,
    S.STUDENT_LNAME,
    TRUNC((SYSDATE - E.ENROLL_DATE) / 7) AS WEEKS_RECEIVING_TUTORING
FROM
    STUDENT S
JOIN
    ENROLL E ON S.STUDENT_ID = E.STUDENT_ID;
```

2.7. Aggregate Functions

Display the number of appointments each tutor has. The number of appointments are displayed in descending order. Business can see if they need to employ an extra tutor for a module if the demand is big.

```
---Total Number of Appointments Per Tutor---
SELECT
    t.TUTOR_ID,
    (t.TUTOR_FNAME || ' ' || t.TUTOR_LNAME) AS "Name",
    COUNT(a.APP_ID) AS "Total Appointments"
FROM
    TUTOR t
LEFT JOIN
    APPOINTMENT a ON t.TUTOR_ID = a.TUTOR_ID
GROUP BY
    t.TUTOR_ID, t.TUTOR_FNAME, t.TUTOR_LNAME
ORDER BY
    "Total Appointments" DESC;
```

Displays how much income each tutor brings into the company. Can help business see which tutor brings in the most income and they can decide to reward that tutor with a bonus,

```
---Total income from Appointments Per Tutor---
SELECT
    t.TUTOR_ID,
    (t.TUTOR_FNAME || ' ' || t.TUTOR_LNAME) AS "Name",
    COUNT(a.APP_ID) AS "Total Appointments",
    SUM(100) AS "Total Income from Appointments"
FROM
    TUTOR t
LEFT JOIN
    APPOINTMENT a ON t.TUTOR_ID = a.TUTOR_ID
GROUP BY
    t.TUTOR_ID, t.TUTOR_FNAME, t.TUTOR_LNAME
ORDER BY
    "Total Income from Appointments" DESC;
```

This query calculates the total amount for salaries each month. Can help business keep track of their monthly tutor expenses.

```
---Total Salary Paid to Tutors Per Month---
SELECT
    TO_CHAR(s.PAID_DATE, 'YYYY-MM') AS "Month",
    SUM(s.AMOUNT) AS "Total Salary Paid"
FROM
    SALARY s
GROUP BY
    TO_CHAR(s.PAID_DATE, 'YYYY-MM')
ORDER BY
    "Month" DESC;
```

2.8. Group By and Having

This displays modules in which more than 3 students are enrolled in. This can help the business see which modules students are struggling in and can offer extra help for those modules.

```
---AMOUNT OF ENROLLMENTS FOR A MODULE ONLY FOR MODULES WITH MORE THAN 3 ENROLLMENTS---
SELECT MODULE_ID, COUNT(*) AS TOTAL_ENROLLMENTS
FROM ENROLL
GROUP BY MODULE_ID
HAVING COUNT(*) > 3;
```

Displays the average amount that each student has paid for appointments however only for R100 or above. Business can see which students are paying mostly in full.

```
---STUDENTS MAKING BIG FINANCIAL CONTRIBUTION---
SELECT
    STUDENT_ID,
    ROUND(AVG(AMOUNT), 2) AS AVG_PAYMENT_AMOUNT
FROM
    PAYMENT
GROUP BY
    STUDENT_ID
HAVING
    ROUND(AVG(AMOUNT), 2) >= 100;
```

2.9. Joins

Displays appointment details for the most popular venues. Business can see which venues are popular and consider adding an additional venue near to the popular one.

```
---SEE POPULARITY OF VENUES---
SELECT A.APP_ID, A.APP_DATETIME, V.VENUE_LOCATION
FROM APPOINTMENT A
JOIN (
    SELECT VENUE_ID, VENUE_LOCATION
    FROM VENUE
    WHERE VENUE_ID IN (
        SELECT VENUE_ID
        FROM (
            SELECT VENUE_ID, COUNT(*) AS num_appointments
            FROM APPOINTMENT
            GROUP BY VENUE_ID
            ORDER BY COUNT(*) DESC
        )
        WHERE ROWNUM <= 1
    )
) V ON A.VENUE_ID = V.VENUE_ID;
```

Shows the tutors name and surname as well as the total salary that they have earned. Business can keep track of how much they employees make.

```
---KEEP RECORD OF TUTOR SALARY---
SELECT T.TUTOR_ID, T.TUTOR_FNAME, SUM(S.AMOUNT) AS TOTAL_SALARY
FROM TUTOR T
JOIN SALARY S ON T.TUTOR_ID = S.TUTOR_ID
GROUP BY T.TUTOR_ID, T.TUTOR_FNAME;
```

2.10. Sub- Queries

Displays the top three most popular tutors. Business can decide to increase these tutors salary or give them a bonus.

```
---TOP 3 TUTORS---
SELECT TUTOR_ID, TUTOR_FNAME, TUTOR_LNAME
FROM TUTOR
WHERE TUTOR_ID IN (
    SELECT TUTOR_ID
    FROM (
        SELECT TUTOR_ID, COUNT(*) AS num_appointments
        FROM APPOINTMENT
        GROUP BY TUTOR_ID
        ORDER BY COUNT(*) DESC
    )
    WHERE ROWNUM <= 3
);
```

Displays the module with the most student enrolled. Can help business see which module most students struggle in and can decide to hire an extra tutor to help out.

```
---Displays module with most enrolled students---
SELECT  MODULE_ID, MODULE_NAME, MODULE_DESCRIPTION, MODULE_YEAR, MODULE_CREDIT
FROM
    MODULES
WHERE
    MODULE_ID = (
        SELECT
            MODULE_ID
        FROM
            (
                SELECT
                    MODULE_ID,
                    COUNT(*) AS ENROLL_COUNT
                FROM
                    ENROLL
                GROUP BY
                    MODULE_ID
                ORDER BY
                    ENROLL_COUNT DESC
            )
        WHERE
            ROWNUM = 1
    );
```

Display which module has the highest income. Business can decide to offer this module at a discounted price or see which module is bringing in the most money.

```
SELECT M.MODULE_ID, M.MODULE_NAME, M.MODULE_DESCRIPTION, M.MODULE_YEAR, M.MODULE_CREDIT,
    (SELECT SUM(P.AMOUNT)
     FROM PAYMENT P
     JOIN PAYMENT_APP PA ON P.PAYMENT_ID = PA.PAYMENT_ID
     JOIN APPOINTMENT A ON PA.APP_ID = A.APP_ID
     WHERE A.MODULE_ID = M.MODULE_ID
    ) AS TOTAL_INCOME
FROM MODULES M
WHERE M.MODULE_ID = (
        SELECT MODULE_ID
        FROM (
                SELECT MODULE_ID, SUM(P.AMOUNT) AS TOTAL_INCOME
                FROM
                    PAYMENT P
                    JOIN PAYMENT_APP PA ON P.PAYMENT_ID = PA.PAYMENT_ID
                    JOIN APPOINTMENT A ON PA.APP_ID = A.APP_ID
                GROUP BY MODULE_ID
                ORDER BY TOTAL_INCOME DESC)
        WHERE ROWNUM = 1
    );
```