Lab 5
Advanced Data Manipulation 3

CSE 4308

Database Management System Lab

October 7, 2024

Contents

1 Creating Tables in MySQL

The `CREATE TABLE` statement is used to create a new table in the database.

1.1 Data Types in MySQL

MySQL supports various data types. Some common data types include:

- **CHAR(n)**: Fixed-length character string of length `n`.
- **VARCHAR(n)**: Variable-length character string with a maximum length of `n`.
- **INT**: Integer numerical value.
- **DECIMAL(p, s)**: Exact numeric value with precision `p` and scale `s`.
- **DATE**: Date value in `YYYY-MM-DD` format.

1.2 Defining Keys

1.2.1 Primary Key

A *Primary Key* uniquely identifies each row in a table. In MySQL, it is defined using the `PRIMARY KEY` constraint. It ensures that no two rows have the same key, facilitating efficient data retrieval and indexing.

1.2.2 Foreign Key

A *Foreign Key* references the primary key of another table, maintaining referential integrity. It links data between tables, enforcing relationships and preventing orphaned records.

```sql
CREATE TABLE orders (
    order_id INT,
    customer_id INT,
    PRIMARY KEY (order_id),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

2 Altering Tables

The `ALTER TABLE` statement is used to modify the structure of an existing table.

Adding Columns

You can add a new column to a table using the `ADD COLUMN` clause.

```sql
ALTER TABLE CUSTOMER
ADD COLUMN DATE_OF_BIRTH DATE;
```

Column Name	Data Type
CUSTOMER_NO	CHAR(5)
CUSTOMER_NAME	VARCHAR(20)
CUSTOMER_CITY	VARCHAR(10)

Table 1: Before

Column Name	Data Type
CUSTOMER_NO	CHAR(5)
CUSTOMER_NAME	VARCHAR(20)
CUSTOMER_CITY	VARCHAR(10)
DATE_OF_BIRTH	DATE

Table 2: After

2.1 Modifying Columns

Use the MODIFY COLUMN clause to change the data type or attributes of an existing column.

```
ALTER TABLE ACCOUNT
MODIFY COLUMN BALANCE DECIMAL(12, 2);
```

2.2 Renaming Columns and Tables

To rename a column or a table, use the CHANGE COLUMN clause or RENAME TABLE statement.

```
ALTER TABLE DEPOSITOR
CHANGE COLUMN ACCOUNT_NO A_NO CHAR(5),
CHANGE COLUMN CUSTOMER_NO C_NO CHAR(5);
```

```
RENAME TABLE DEPOSITOR TO DEPOSITOR_INFO;
```

2.3 Adding Constraints

Constraints enforce rules at the table level. To add a foreign key constraint after table creation:

```
ALTER TABLE DEPOSITOR_INFO
ADD CONSTRAINT FK_DEPOSITOR_ACCOUNT FOREIGN KEY (A_NO)
REFERENCES ACCOUNT(ACCOUNT_NO),
ADD CONSTRAINT FK_DEPOSITOR_CUSTOMER FOREIGN KEY (C_NO)
REFERENCES CUSTOMER(CUSTOMER_NO);
```

3 Inserting Data

The INSERT INTO statement is used to add new rows to a table.

Example: Inserting Data into ACCOUNT Table

```
INSERT INTO ACCOUNT VALUES ('A-101', 5000.00);
```

3.1 Inserting Multiple Records

You can insert multiple records in a single statement:

```
INSERT INTO ACCOUNT VALUES ('A-102', 15000.00);
INSERT INTO ACCOUNT VALUES ('A-103', 25000.00);
```

4 Querying Data

The SELECT statement is used to retrieve data from one or more tables.

4.1 Basic SELECT Statement

```
SELECT * FROM ACCOUNT;
```

4.2 Using WHERE Clause

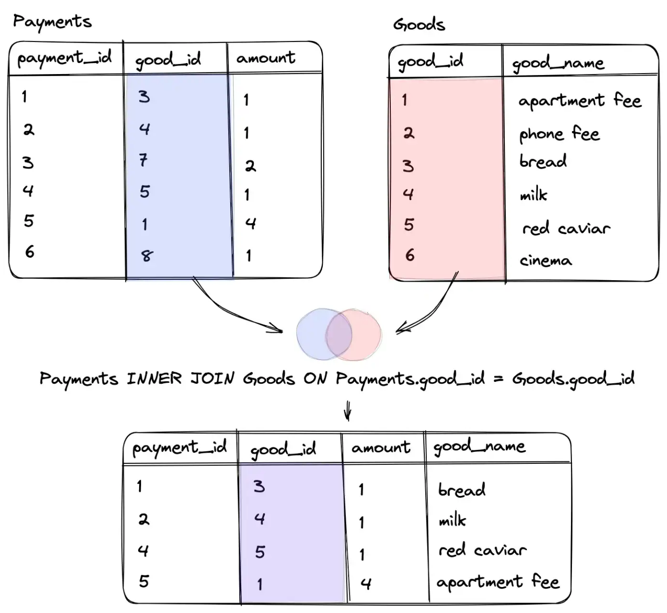
The WHERE clause filters records based on specified conditions.

```
SELECT ACCOUNT_NO
FROM ACCOUNT
WHERE BALANCE < 100000;
```

Joins

Joins are used to combine rows from two or more tables based on related columns.

INNER JOIN


Payments
Goods
Payments INNER JOIN Goods ON Payments.good_id = Goods.good_id

payment_id	good_id	amount
1	3	1
2	4	1
3	7	2
4	5	1
5	1	4
6	8	1

good_id	good_name
1	apartment fee
2	phone fee
3	bread
4	milk
5	red caviar
6	cinema

payment_id	good_id	amount	good_name
1	3	1	bread
2	4	1	milk
4	5	1	red caviar
5	1	4	apartment fee

Figure 1: Sample of How Inner Join works

Another example:

```sql
SELECT S.STUDENT_NAME, C.COURSE_NAME
FROM STUDENT S
INNER JOIN ENROLLMENT E ON S.STUDENT_ID = E.STUDENT_ID
INNER JOIN COURSE C ON E.COURSE_ID = C.COURSE_ID;
```

This query aims to retrieve the names of students along with the courses they are enrolled in by joining three tables: **STUDENT**, **ENROLLMENT**, and **COURSE**.

S_ID	S_NAME	DEPT
1	Alice	CSE
2	Bob	EEE
3	Carol	MCE

Table 3: Student Table

S_ID	C_ID
1	101
2	102
1	103
3	101

Table 4: ENROLLMENT Table

C_ID	C_NAME	DEPT
101	Database Systems	CSE
102	Circuit Analysis	EEE
103	Operating Systems	CSE

Table 5: COURSE Table

Table 6: Before Tables

S_ID	S_NAME	DEPT	C_ID
1	Alice	CSE	101
1	Alice	CSE	103
2	Bob	EEE	102
3	Carol	MCE	101

Table 7: Intermediate Result (Table A)

S_ID	S_NAME	DEPT	C_ID	C_NAME
1	Alice	CSE	101	Database Systems
1	Alice	CSE	103	Operating Systems
2	Bob	EEE	102	Circuit Analysis
3	Carol	MCE	101	Database Systems

Table 8: Intermediate Result (Table B)

S_NAME	C_NAME
Alice	Database Systems
Alice	Operating Systems
Bob	Circuit Analysis
Carol	Database Systems

Table 9: Final Result: After INNER JOIN with COURSE

NATURAL JOIN

Natural join is an SQL join operation that creates a join on the base of the common columns in the tables. To perform natural join there must be one common attribute(Column) between two tables. Natural join will retrieve from multiple relations. It works in three steps.

```sql
SELECT *
FROM TABLE1
NATURAL JOIN TABLE2;
```

- A natural join matches columns with the same names in both tables and only includes rows where the values in those columns are equal (consistent tuples).
- It excludes rows where the values don't match (inconsistent tuples).
- After joining, it removes one of the duplicate columns from the result set.

```sql
SELECT *
FROM STUDENT
NATURAL JOIN ENROLLMENT;
```

S_ID	S_NAME	DEPT
1	Alice	CSE
2	Bob	EEE
3	Carol	MCE

Table 10: Student Table

S_ID	C_ID
1	101
2	102
1	103
3	101

Table 11: ENROLLMENT Table

C_ID	C_NAME	DEPT
101	Database Systems	CSE
102	Circuit Analysis	EEE
103	Operating Systems	CSE

Table 12: COURSE Table

Table 13: Before Tables

S_ID	S_NAME	DEPT	C_ID
1	Alice	CSE	101
1	Alice	CSE	103
2	Bob	EEE	102
3	Carol	MCE	101

Table 14: After Natural Join

4.3 Set Operators

Set operators combine results from two or more SELECT statements.

4.3.1 UNION

Combines results from two queries and removes duplicates.

```
SELECT CUSTOMER_NO FROM DEPOSITOR_INFO
UNION
SELECT CUSTOMER_NO FROM BORROWER;
```

4.3.2 INTERSECT

Returns common records from two queries. Note that MySQL does not support INTERSECT directly; you can simulate it using INNER JOIN or subqueries.

```
-- Using INNER JOIN to simulate INTERSECT
SELECT A.*
FROM (SELECT column1, column2 FROM table1) A
INNER JOIN (SELECT column1, column2 FROM table2) B
ON A.column1 = B.column1 AND A.column2 = B.column2;


-- Using IN with subqueries to simulate INTERSECT
SELECT column1, column2
FROM table1
WHERE (column1, column2) IN  (SELECT column1, column2 FROM table2);
```

4.4 Subqueries

A subquery is a query nested inside another query.

```
SELECT CUSTOMER_NAME
FROM CUSTOMER
WHERE CUSTOMER_NO IN (
    SELECT C_NO FROM DEPOSITOR_INFO
);
```

4.5 Aggregate Functions

Aggregate functions perform calculations on multiple rows.

4.5.1 SUM

Calculates the total sum of a numeric column.

```
SELECT SUM(BALANCE) AS TOTAL_BALANCE
FROM ACCOUNT;
```

4.5.2 AVG

Calculates the average value.

```
SELECT AVG(BALANCE) AS AVERAGE_BALANCE
FROM ACCOUNT;
```

4.6 GROUP BY and HAVING Clauses

4.6.1 GROUP BY

Groups rows that have the same values in specified columns.

```
SELECT CUSTOMER_CITY, COUNT(*) AS NUM_CUSTOMERS
FROM CUSTOMER
GROUP BY CUSTOMER_CITY;
```

4.6.2 HAVING

Filters groups based on aggregate functions.

```
SELECT CUSTOMER_CITY, AVG(BALANCE) AS AVG_BALANCE
FROM CUSTOMER C
INNER JOIN DEPOSITOR_INFO D ON C.CUSTOMER_NO = D.C_NO
INNER JOIN ACCOUNT A ON D.A_NO = A.ACCOUNT_NO
GROUP BY CUSTOMER_CITY
HAVING AVG(BALANCE) > 10000;
```

5 Updating and Deleting Data

5.1 UPDATE Statement

The UPDATE statement modifies existing records.

```
UPDATE CUSTOMER
SET C_CITY = 'KLN'
WHERE C_CITY = 'KHL';
```

5.2 DELETE Statement

The DELETE statement removes existing records.

```
DELETE FROM CUSTOMER
WHERE CUSTOMER_NO = 'C-301';
```

5.3 TRUNCATE TABLE

The TRUNCATE TABLE statement removes all records from a table, resetting identity values.

```
TRUNCATE TABLE CUSTOMER;
```

ID	Name
1	John Doe
2	Jane Smith
3	Alice Brown
4	Bob Johnson

Table 15: Before TRUNCATE

ID	Name
No Data	

Table 16: After TRUNCATE

Table 17: Customer Table Before and After TRUNCATE

6 Tasks

Creating Tables

```
CREATE TABLE CUSTOMER (
    CUSTOMER_NO CHAR(5) PRIMARY KEY,
    CUSTOMER_NAME VARCHAR(20) NOT NULL,
    CUSTOMER_CITY VARCHAR(10)
);

CREATE TABLE ACCOUNT (
    ACCOUNT_NO CHAR(5) PRIMARY KEY,
    BALANCE DECIMAL(12,2) NOT NULL
);

CREATE TABLE DEPOSITOR (
    CUSTOMER_NO CHAR(5),
    ACCOUNT_NO CHAR(5),
    PRIMARY KEY (CUSTOMER_NO, ACCOUNT_NO)
);
```

Inserting Sample Data

Insert data into CUSTOMER table:

```
INSERT INTO CUSTOMER (CUSTOMER_NO, CUSTOMER_NAME, CUSTOMER_CITY) VALUES
('C-101', 'John Doe', 'DHK');
INSERT INTO CUSTOMER (CUSTOMER_NO, CUSTOMER_NAME, CUSTOMER_CITY) VALUES('
   C-102', 'Jane Smith', 'KHL');
INSERT INTO CUSTOMER (CUSTOMER_NO, CUSTOMER_NAME, CUSTOMER_CITY) VALUES('
   C-103', 'Alice Brown', 'CTG');
INSERT INTO CUSTOMER (CUSTOMER_NO, CUSTOMER_NAME, CUSTOMER_CITY) VALUES('
   C-104', 'Bob Johnson', 'DHK');
```

Insert data into ACCOUNT table:

```
INSERT INTO ACCOUNT (ACCOUNT_NO, BALANCE) VALUES
('A-101', 5000.00);
INSERT INTO ACCOUNT (ACCOUNT_NO, BALANCE) VALUES('A-102', 15000.00),
INSERT INTO ACCOUNT (ACCOUNT_NO, BALANCE) VALUES('A-103', 25000.00);
INSERT INTO ACCOUNT (ACCOUNT_NO, BALANCE) VALUES('A-104', 8000.00);
```

Insert data into DEPOSITOR table:

```
INSERT INTO DEPOSITOR (CUSTOMER_NO, ACCOUNT_NO) VALUES('C-101', 'A-101');
INSERT INTO DEPOSITOR (CUSTOMER_NO, ACCOUNT_NO) VALUES('C-102', 'A-102');
INSERT INTO DEPOSITOR (CUSTOMER_NO, ACCOUNT_NO) VALUES('C-103', 'A-103');
INSERT INTO DEPOSITOR (CUSTOMER_NO, ACCOUNT_NO) VALUES('C-104', 'A-104');
```

6.1 Task 1

1. **Alter the** CUSTOMER **table** to add a new column DATE_OF_BIRTH of type DATE.
2. **Rename the** DEPOSITOR **table** to DEPOSITOR_INFO.
3. **Change the column names** in DEPOSITOR_INFO table:
   - Change CUSTOMER_NO to C_NO.
   - Change ACCOUNT_NO to A_NO.
4. **Add foreign key constraints** to DEPOSITOR_INFO table:
   - C_NO references CUSTOMER(CUSTOMER_NO).
   - A_NO references ACCOUNT(ACCOUNT_NO).
5. **Retrieve all data** from the CUSTOMER table.
6. **Retrieve** CUSTOMER_NAME **and** BALANCE for all customers using an **INNER JOIN** between CUSTOMER, DEPOSITOR_INFO, and ACCOUNT tables.
7. **Find the average balance** of all accounts.
8. **Retrieve the** CUSTOMER_NAME **and** ACCOUNT_NO of customers using a **NATURAL JOIN** between CUSTOMER and DEPOSITOR_INFO tables.
9. **Update the city** of customer 'C-102' from 'KHL' to 'KLN'.
10. **Delete the account** 'A-104' from the ACCOUNT table.
11. **Delete all customers** who have no accounts.

6.2 Task 2

1. **Create the** LOAN **table** and insert sample data.
   - Create table LOAN with columns:
     - LOAN_NO (CHAR(5), Primary Key)
     - AMOUNT (DECIMAL(12,2), NOT NULL)
   - **SQL to create** LOAN **table:**

     ```
     CREATE TABLE LOAN (
         LOAN_NO CHAR(5) PRIMARY KEY,
         AMOUNT DECIMAL(12,2) NOT NULL
     );
     ```

   - **Insert sample data into** LOAN **table:**

     ```
     INSERT INTO LOAN VALUES ('L-201', 10000.00);
     INSERT INTO LOAN VALUES('L-202', 20000.00);
     INSERT INTO LOAN VALUES('L-203', 15000.00);
     ```

2. **Create the** BORROWER **table** with constraints and insert sample data.
   - Create table BORROWER with columns:
     - CUSTOMER_NO (CHAR(5), Foreign Key referencing CUSTOMER(CUSTOMER_NO))
     - LOAN_NO (CHAR(5), Foreign Key referencing LOAN(LOAN_NO))

  - Primary Key on (CUSTOMER_NO, LOAN_NO)
- **SQL to create** BORROWER **table:**

```
        CREATE TABLE BORROWER (
            CUSTOMER_NO CHAR(5),
            LOAN_NO CHAR(5),
            PRIMARY KEY (CUSTOMER_NO, LOAN_NO),
            FOREIGN KEY (CUSTOMER_NO) REFERENCES CUSTOMER(
               CUSTOMER_NO),
            FOREIGN KEY (LOAN_NO) REFERENCES LOAN(LOAN_NO)
        );
```

- **Insert sample data into** BORROWER **table:**

```
        INSERT INTO BORROWER VALUES ('C-101', 'L-201');
        INSERT INTO BORROWER VALUES('C-102', 'L-202');
        INSERT INTO BORROWER VALUES('C-103', 'L-203');
```

3. **Find customers** who have both accounts and loans using an **INNER JOIN**.
4. **Find customers** who have accounts but not loans.
5. **Update the balances** by adding 5% interest to accounts with balance less than 10,000.

Submission Guidelines

You are required to submit a report that includes your code, a detailed explanation, and the corresponding output. Rename your report as <StudentID_Lab_1.pdf>.

- Your lab report must be as a PDF and containing the code. Recommended tool: Overleaf. You can use some of the available templates in Overleaf. You can use Google docs or any other word processor
- Include all code/pseudo code relevant to the lab tasks in the lab report. Dont add image. Use lstlisting for latex or Codeblocks plugin for google docs for syntax highlighting.
- Please provide citations for any claims that are not self-explanatory or commonly understood.
- It is recommended to use vector graphics (e.g., .pdf, .svg) for all visualizations.
- In cases of high similarities between two reports, both reports will be discarded.