East West University

Department of Computer Science and Engineering

CSE302: LAB03(Handout)

Course Instructor: Mahmuda Rawnak Jahan

Writing SQL Statements for Multi-table, Set and Aggregate

`banking.sql` Script: Download Link

Execute the SQL Script:

- Open your SQL client (e.g., SQL*Plus for Oracle or any compatible database environment).
- Run the script using the command: `@f:\SQL\banking.sql`

```
1) Branch (branch_name, branch_city, assets)

2) Customer (customer_name, customer_street, customer_city)

3) Account (account_number, branch_name, balance)

4) Loan (loan_number, branch_name, amount)

5) Depositor (customer_name, account_number)

6) Borrower (customer_name, loan_number)
```

– 1.1 String Matching with `LIKE` Operator

– Find customers from cities starting with "New":

```
SELECT * FROM Customer WHERE customer_city LIKE 'New%';
```

– 1.2 Using `DISTINCT` Keyword

– List unique cities where branches are located:

```
SELECT DISTINCT branch_city FROM Branch;
```

– 1.3 Arithmetic Operations in `SELECT` Clause

– Calculate half of each account's balance:

```
SELECT account_number, balance / 2 AS half_balance FROM Account;
```

– 2. Joins and Cartesian Product

– 2.1 `CROSS JOIN` (Cartesian Product)

– Get all combinations of accounts and customers (useful only if needed for a specific purpose):

```
SELECT * FROM Account CROSS JOIN Customer;
```

– 2.2 `JOIN ... ON` (Conditional Join)

– Join `Account` and `Branch` based on branch name:

```
SELECT Account.account_number, Account.balance, Branch.branch_city
FROM Account
JOIN Branch ON Account.branch_name = Branch.branch_name;
```

– 2.3 `JOIN ... USING` (Join with Common Column)

– Join `Loan` and `Branch` on branch name using `USING`:

```
SELECT loan_number, amount, branch_city
FROM Loan
JOIN Branch USING(branch_name);
```

– 2.4 `NATURAL JOIN`

– Natural join between `Depositor` and `Customer` tables based on `customer_name`:

```
SELECT * FROM Depositor NATURAL JOIN Customer;
```

– 3. Sorting and Set Operations

– 3.1 `ORDER BY` Clause

– List accounts ordered by balance in descending order:

```
SELECT * FROM Account ORDER BY balance DESC;
```

– 3.2 Set Operations (`UNION`, `INTERSECT`, `MINUS`)

– Find all unique customer names who are either depositors or borrowers (use `UNION`):

```
SELECT customer_name FROM Depositor
UNION
```

```
SELECT customer_name FROM Borrower;
```

– Find customers who are both depositors and borrowers (use `INTERSECT`):

```
SELECT customer_name FROM Depositor
INTERSECT
SELECT customer_name FROM Borrower;
```

– Find customers who are depositors but not borrowers (use `MINUS`):

```
SELECT customer_name FROM Depositor
MINUS
SELECT customer_name FROM Borrower;
```

– 4. Aggregate Functions and Grouping

– 4.1 `AVG`, `SUM`, `MIN`, `MAX`, and `COUNT`

– Calculate the total assets in each branch:

```
SELECT branch_name, SUM(assets) AS total_assets FROM Branch GROUP BY
branch_name;
```

– Find the average balance of all accounts:

```
SELECT AVG(balance) AS average_balance FROM Account;
```

– Get the highest loan amount in each branch:

```
SELECT branch_name, MAX(amount) AS max_loan_amount FROM Loan GROUP BY
branch_name;
```

– Count the total number of customers in each city:

```
SELECT customer_city, COUNT(*) AS customer_count FROM Customer GROUP
BY customer_city;
```