TRIBHUWAN UNIVERSITY INSTITUTE OF ENGINEERING

KATHMANDU ENGINEERING COLLEGE

KALIMATI, KATHMANDU

LAB REPORT ON

Database Management System

PRE DATE: 2025/02/24 EXPERIMENT NO:6

POST DATE: 2025/02/24

SUBMITTED BY:

SUBMITTED TO:

NAME: Deepak Thapa Department of

ROLL: KAT078BCT029 Computer Engineering

GROUP: A2

YEAR: III/II

OPERATIONS OF DBMS

THEORY

SQL UNION, INTERSECTION, AND DIFFERENCE

SQL provides several set operations to combine or compare the results of multiple queries:

- UNION merges the results of two queries and eliminates duplicate values.
- UNION ALL combines results while keeping duplicates.
- INTERSECT returns only the records present in both queries.
- EXCEPT (or MINUS) shows records from the first query that are not found in the second query.

Syntax:

```
SELECT column_name FROM table1
UNION/INTERSECT/EXCEPT
SELECT column_name FROM table2;
```

COUNTING AND SUMMING DATA

SQL has built-in functions to count the rows and sum the values in a column:

- **COUNT()** returns the total number of rows in a table.
- SUM() calculates the sum of the values in a specified column.

Syntax:

```
SELECT COUNT(*) FROM table_name;
SELECT SUM(column_name) FROM table_name;
```

CREATING INDEX

Indexes enhance the speed of data retrieval in large datasets by providing faster access to specific records.

Syntax:

```
CREATE INDEX index_name ON table_name (column_name);
```

CREATING TRIGGER FOR UPDATE AND INSERT

A trigger is an automatic SQL operation that runs before or after an INSERT or UPDATE statement.

Syntax:

```
CREATE TRIGGER trigger_name
AFTER INSERT ON table_name
FOR EACH ROW
BEGIN
-- trigger action
END;
```

DROPPING TABLE IF EXISTS

To avoid errors when creating a table, it is a good practice to remove it first if it already exists.

Syntax:

```
DROP TABLE IF EXISTS table_name;
```

TRANSACTION AND ROLLBACK

A transaction ensures that multiple SQL operations execute as a single unit. If a rollback is issued before committing, all changes are reversed.

Syntax:

```
START TRANSACTION;

INSERT INTO table_name VALUES (...);

ROLLBACK; -- Revert changes
COMMIT; -- Permanently save changes
```

CREATING AND UPDATING PROCEDURES

A stored procedure is a reusable SQL command that can be executed whenever required.

Syntax for creating a procedure:

```
CREATE PROCEDURE procedure_name()
BEGIN
    SELECT * FROM table_name;
END;
```

Syntax for updating a procedure:

```
ALTER PROCEDURE procedure_name
BEGIN
SELECT column_name FROM table_name;
END;
```

CREATING VIEWS

A view is a virtual table that stores a SQL query, making it easier to reuse and abstract data.

Syntax:

```
CREATE VIEW view_name AS SELECT column_name FROM table_name WHERE condition;
```

Creating a view in database

Problem 1: Creating required tables and inserting value in those tables

```
--create database lastlab;
--use lastlab;
drop table Issues;
drop table book;
drop table Booklist;
drop table student;
drop table Teacher;
--select *from issues,book,booklist,student,teacher;
create table Booklist(isbn int NOT NULL,name varchar(50),publication varchar(25)
primary key(isbn));
create table Book(bid int not null,bname varchar(50),author varchar(25),price
bigint,primary key(bid));
create table student(sid int not null,sfirstname varchar(20),slastname
varchar(20), faculty varchar(20), primary key (sid));
create table Teacher(Tid int not null,tname varchar(20),tfaculty varchar(10),tsalary
int,hod int,primary key(Tid));
create table Issues(IID int not null, bid int foreign key references book(bid), tid int foreign
key references teacher(tid), sid int foreign key references student(sid),
year int, month int, day int primary key(IID));
```

```
insert into book
```

values(1,'AI','sujan',400.31),(2,'dbms','sujan',350.12),(3,'os','ritu',375.12),(4,'ai','dhawa',300),(5,'d bms','bison',400),(6,'NM','gaurav',400);

insert into booklist values('10001','Artificial

Intelligence', 'janata'), ('10002', 'dbms', 'kec'), ('10003', 'os', 'insights'), ('10004', 'dbms', 'heritage'), ('10005', 'ai', 'heritage');

insert into student values

(1,'ashish','shrestha','BEI'),(2,'bipan','raut','BEI'),(3,'sushant','thapa','BCT'),(4,'ram','kc','BEI'),(5,'s hyam','shah','BCT'),(6,'nitesh','panta','BCA'),(7,'sakshi','thapa','BCA');

insert into Teacher values

(1,'sujan','BEI',110000,3),(2,'nabin','BCT',125000,5),(3,'rajan','BEI',160000,3),(4,'ritu','BCT',100000,5),(5,'sudeep','BCT',160000,5),(6,'gaurav','BEI',110000,3),(7,'niko','BCA',120000,7),(8,'prajwal','BCA',100000,7);

insert into Issues

values(1,1,1,null,2022,01,20),(2,1,null,1,2022,01,20),(3,2,1,null,2022,06,25),(4,6,null,1,2022,06,25),(5,4,null,3,2024,3,14);

select *from book;

select *from Booklist;

select *from student;

select *from Teacher;

select *from Issues;

Output:

| | bid | bna | me | aut | thor | price | | | | |
|---|------|---------|------------------------|---------|---------------------|----------|------------|--------|------|-----|
| 1 | 1 | Al | | suj | jan | 400 | | | | |
| 2 | 2 | dbr | ms | suj | jan | 350 | | | | |
| 3 | 3 | os | | ritu | 1 | 375 | | | | |
| 4 | 4 | ai | | dh | awa | 300 | | | | |
| 5 | 5 | dbr | ns | bis | son | 400 | | | | |
| 6 | 6 | NM | 1 | ga | urav | 400 | | | | |
| | isbn | | name | е | | | pu | blicat | ion | |
| 1 | 100 | 01 | Artific | cial Ir | ntellig | ence | jar | nata | | |
| 2 | 100 | 02 | | | S | | ke | С | | |
| 3 | 100 | 03 | os | | | ins | sights | | | |
| 4 | 100 | 04 dbms | | s | | heritage | | | | |
| 5 | 100 | 05 ai | | | | | heritage | | | |
| | sid | sfirs | tnan | ne | slast | name | fac | culty | | |
| 1 | 1 | ash | ish | | shre | stha | В | ΕI | | |
| 2 | 2 | bip | an | | raut | | В | ΕI | | |
| 3 | 3 | sus | hant | | thap | a | В | СТ | | |
| 4 | 4 | ran | ram shyam nitesh | | kc shah panta | | BEI BCT | | | |
| 5 | 5 | shy | | | | | | | | |
| 6 | 6 | nite | | | | | В | BCA | | |
| 7 | 7 | sakshi | | | thap | a | В | CA | | |
| | Tid | tna | me | tfac | ulty | tsalar | у | hod | | |
| 1 | 1 | suj | sujan BE | | | 1100 | 00 | 3 | | |
| 2 | 2 | nal | nabin | | T | 1250 | 00 | 5 | | |
| 3 | 3 | rajan | | BEI | | 1600 | 00 | 3 | | |
| 4 | 4 | ritu | | BCT | | 1000 | 00 | 5 | | |
| 5 | 5 | sude | | BC | Т | 1600 | 00 | 5 | | |
| 6 | 6 | gai | Jr | BEI | | 1100 | 00 | 3 | | |
| 7 | 7 | nik | 0 | BC | A | 1200 | 00 | 7 | | |
| 8 | 8 | pra | i | BC | Α | 1000 | 00 | 7 | | |
| | IID | bid | tid | | sid | ye | ear | me | onth | day |
| 1 | 1 | 1 | 1 | | NU | LL 2 | 022 | 1 | | 20 |
| 2 | 2 | 1 | N | JLL | 1 | 2 | 022 | 1 | | 20 |
| 3 | 3 | 2 | 1 | | NU | LL 2 | 022 | 6 | | 25 |
| 4 | 4 | 6 | N | JLL | 1 | 2 | 022 | 6 | | 25 |
| 5 | 5 | 4 | 4 NUL | | 3 | 2 | 024 | 4 3 | | 14 |

Problem 2: SQL query to display the teacher name along with head name. (SELF-

JOIN) select *from teacher t1, teacher t2;

select t2.tname, t1.tname as head_name from Teacher t1 join Teacher t2 on t1.Tid = t2.hod;

| | tname | head_name |
|---|---------|-----------|
| 1 | sujan | rajan |
| 2 | nabin | sudeep |
| 3 | rajan | rajan |
| 4 | ritu | sudeep |
| 5 | sudeep | sudeep |
| 6 | gaurav | rajan |
| 7 | niko | niko |
| 8 | prajwal | niko |

Problem 3: Creating and executing the view.

DROP VIEW [bca faculty];

select * from student;

CREATE VIEW [bct faculty] AS

SELECT sfirstname, slastname

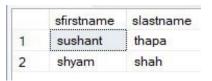
FROM student

WHERE faculty = 'bct';

--Execute view

SELECT *FROM [bct faculty];

Output:



Problem 4: Creating Procedure.

DROP PROCEDURE SelectAlltable;

CREATE PROCEDURE SelectAlltable

AS

SELECT * FROM

teacher SELECT *FROM

student select *from

Issues

select *from book

select *from Booklist

GO;

exec SelectAlltable;



Problem 5: Ordering book name by ascending and then author by descending order. select *from Book order by bname,author desc; **Output:**



Problem 6: Ordering book name and author by ascending order.

select *from Book order by author, bname;

Output:



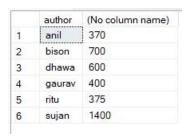
Problem 7: Inserting new values and get the sum of prices of book according to teacher.

insert into book

values(11,'AI','sujan',350),(7,'dbms','sujan',300),(8,'os','anil',370),(9,'ai','dhawa',300),(10,'dbms','b ison',300);

--write a sql query to get the sum of prices of book according to teacher select author, sum(price) from Book group by author;

Output:



Problem 8: Display count of books according to author.

select author,count(bid) as count_of_book from Book group by author;

Output:

| | author | count_of_book |
|---|--------|---------------|
| 1 | anil | 1 |
| 2 | bison | 2 |
| 3 | dhawa | 2 |
| 4 | gaurav | 1 |
| 5 | ritu | 1 |
| 6 | sujan | 4 |

Problem 9: Sum the prices of book according to author and book name. select author, bname, sum(price) from Book group by author, bname; select author, bname, count(*) from Book group by author, bname;

Output:

| | author | bname | (No column name) | | |
|---|--------|-------|------------------|--|--|
| 1 | dhawa | ai | 600 | | |
| 2 | sujan | Al | 750 | | |
| 3 | bison | dbms | 700 | | |
| 4 | sujan | dbms | 650 | | |
| 5 | gaurav | NM | 400 | | |
| 6 | anil | os | 370 | | |
| 7 | ritu | os | 375 | | |
| | author | bname | (No column name) | | |
| 1 | dhawa | ai | 2 | | |
| 2 | sujan | Al | 2 | | |
| 3 | bison | dbms | 2 | | |
| 4 | sujan | dbms | 2 | | |
| 5 | gaurav | NM | 1 | | |
| 6 | anil | os | 1 | | |
| 7 | ritu | os | 1 | | |

Problem 10: Create and execute procedure that selects certain student.

DROP PROCEDURE selectstudent;

CREATE PROCEDURE selectstudent @sfirstname varchar(50)

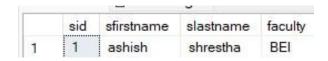
AS BEGIN

SELECT * FROM student WHERE sfirstname =

@sfirstname END;

EXEC selectstudent ashish;

Output:



Problem 11: Changing the selection criteria of existing procedure.

alter procedure selectstudent @slastname nvarchar(50) as

begin

select *from student where slastname =

@slastname end;

EXEC selectstudent shrestha;

Output:



Problem 12: Creating index.

drop index idx price on book;

CREATE INDEX idx price

ON book (price);



Problem 13: Checking how rollback works.

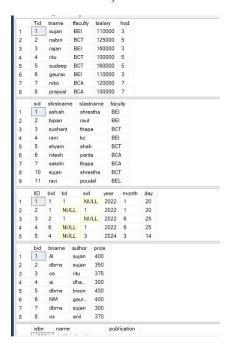
Begin transaction;

insert into student values (10, 'sujan', 'shrestha', 'BCT'), (11, 'ravi', 'poudel', 'BEL');

exec SelectAlltable;

rollback;

Case I: Before rollback (the value is indifferent).



Case II: After Rollback. (The inserted value of student is not saved as it is not committed)



Problem 14: Rollback after commit.

Begin transaction;

insert into student values (8,'sujan','tamang','BCT'),

(9,'ravi','poudel','BEL'); commit;

Begin transaction;

rollback;

exec SelectAlltable;

Output:

Case I: After commit.



Case II: After rollback. (the result is same as the value is updated permanently)



```
Problem 15: Creating new table employee and adding trigger on update and viewing the result.
drop table employee;
select *from employee;
create table employee(
e id int primary key,
e_name varchar(20),
e course varchar(20),
e Salary int);
create trigger triforupdateemployee
on employee
for update
as
begin
print(' trigger for update executed')
end
go
insert into employee values (1,'sujan','nm',50000);
update employee set e_name = 'rubi' where e_course =
'nm'; select *from employee;
Output
Result:
                                    e_id e_name e_course e_Salary
```

Message:

```
(1 row affected)
trigger for update executed

(1 row affected)
(1 row affected)

Completion time: 2025-02-20723:53:56.0878265+05:45
```

Problem 16: Dropping table if they exist.

DROP TABLE IF EXISTS employee_log;

DROP TABLE IF EXISTS employee;

Problem 17: Creating table employee and employee_log and making trigger for recording the date of data being insert into table.

```
CREATE TABLE employee(
e_id INT PRIMARY KEY,
e_name VARCHAR(20),
e_course VARCHAR(20),
e_Salary INT
);
- Create employee_log table CREATE TABLE employee_log(
eid INT,
action VARCHAR(50),
action_time DATETIME, -- Changed to DATETIME for both date and time
FOREIGN KEY (eid) REFERENCES employee(e_id) -- Correct placement of FOREIGN
KEY constraint
```

);

- Create the trigger for insert operations on the employee table CREATE TRIGGER triforinsertemployee

ON employee

FOR

INSERT AS

BEGIN

INSERT INTO employee log(eid, action, action time)

SELECT i.e_id, 'INSERT', CURRENT_TIMESTAMP -- i.e_id to match your table

definition FROM inserted i -- Use 'inserted', not the table name 'employee'

END;

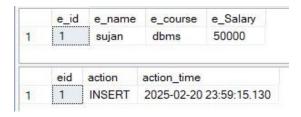
GO

insert into employee values (1,'sujan','dbms',50000);

SELECT * FROM employee;

SELECT * FROM employee log;

Output:



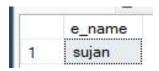
Problem 18: Display intersection of employee and teacher.

SELECT e_name FROM employee

INTERSECT

SELECT tname FROM teacher;

Output:



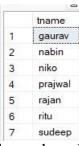
Problem 19: Display names which are teacher but not employee.

SELECT tname FROM teacher

EXCEPT

SELECT e_name FROM employee;

Output:

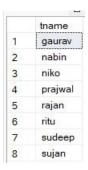


Problem 20: Displaying the union of teacher and employee.

SELECT tname FROM teacher

UNION

SELECT e_name FROM employee;



DISCUSSION

In this DBMS lab, we explored various SQL queries such as union, intersection, and difference to enhance our ability to merge and compare data from different tables. We also utilized functions like COUNT, SUM, and ORDER BY to summarize and organize data effectively.

We wrote queries to identify department heads, create indexes, and implement triggers that automate updates and inserts. Our work with transactions illustrated how rolling back changes before a commit reverses the alterations, while rolling back after a commit leaves the data unchanged.

Furthermore, we created and modified stored procedures to improve selection criteria for more efficient data retrieval. Finally, we implemented views to simplify data presentation and make it more structured.

CONCLUSION

This lab provided practical experience with essential SQL queries for effective database management. It deepened our understanding of data combination, modification, and organization. By working with transactions and rollbacks, we gained insight into how databases maintain data consistency.

Additionally, the implementation of triggers, indexes, and stored procedures highlighted the significance of automation and performance optimization. Overall, this lab enhanced our SQL skills and demonstrated their practical use in real-world database management.