

Tribhuvan University

GREENFIELD NATIONAL COLLEGE

**Bafal- Sitapaila, Kathmandu**

**Lab Report**

**of**

**“Database Management System”**

**Submitted By: Submitted To:**

Chet Raj Ojha

Lecturer/ Faculty Member

BCA Department

Name: Manish karki

Semester: IV

Program: Bachelor of Computer Application

**Table of Contents**

[Creating User in SQL Plus 3](#_Toc192263629)

[Creating Table 3](#_Toc192263630)

[Inserting Data 4](#_Toc192263631)

[Displaying data from table 4](#_Toc192263632)

[Alter Table – Adding Column 5](#_Toc192263633)

[Updating data 5](#_Toc192263634)

[Displaying data in UpperCase and LowerCase 6](#_Toc192263635)

[Displaying the sum of data 6](#_Toc192263636)

[Order By Clause 7](#_Toc192263637)

[Group By Clause 8](#_Toc192263638)

[Add Primary key on existing table 8](#_Toc192263639)

[Creating table with Foreign Key 9](#_Toc192263640)

[SQL Join 10](#_Toc192263641)

[Inner Join 10](#_Toc192263642)

[Left Join 10](#_Toc192263643)

[PL/SQL Block 11](#_Toc192263644)

[First Program 11](#_Toc192263645)

[Sum of two numbers 11](#_Toc192263646)

[PL/SQL Functions 12](#_Toc192263647)

[Adder Function 12](#_Toc192263648)

[Full name Function 13](#_Toc192263649)

[Creating Procedure 14](#_Toc192263650)

[To insert into table 14](#_Toc192263651)

[To Update data 15](#_Toc192263652)

[To Delete data 15](#_Toc192263653)

[Creating Triggers 16](#_Toc192263654)

[Creating Index 18](#_Toc192263655)

[Renaming Index 18](#_Toc192263656)

[Dropping Index 19](#_Toc192263657)

[Access Control 20](#_Toc192263658)

[Grant Privileges on Table 20](#_Toc192263659)

[Revoke Privileges on Table 20](#_Toc192263660)

# Creating User in SQL Plus

User creation is a fundamental administrative task in database management.

First of all, open SQL Plus (run as administrator).

**Enter user-name**: sys as sysdba  
**Enter password**: sss

**Connected to**:  
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production  
With the Partitioning, OLAP, Data Mining and Real Application Testing options

**SQL>** Create User Manish identified by B01;

User created.

**SQL>** grant connect, resource to Manish;

Grant succeeded.

**SQL>** connect Manish;  
Enter password: M01  
Connected.

# Creating Table

**Theory**: Tables are the fundamental structure for storing data in relational databases. A table consists of rows and columns, where each column has a specific data type that determines what kind of data it can store.

*Query*:

CREATE TABLE table\_name (

column1 datatype,

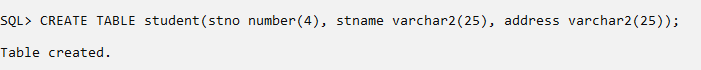
column2 datatype,

column3 datatype,

....

);

*For example*:

CREATE TABLE student (stno number(4), stname varchar2(25), address varchar2(25));

# Inserting Data

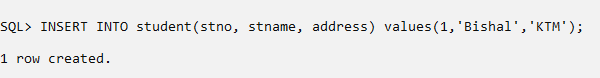
**Theory**: The INSERT statement is used to add new records to tables.

*Query*:

INSERT INTO table\_name (column1, column2, column3, ...)

VALUES (value1, value2, value3, ...);

*For example*:

INSERT INTO student(stno, stname, address) values(1,'Bishal','KTM');

# Displaying data from table

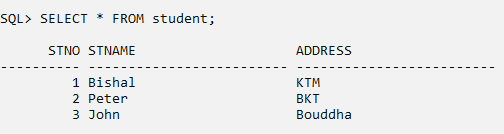
**Theory**: The SELECT statement allows you to retrieve data from one or more tables based on specific criteria.

*Query*:

SELECT column1, column2, ... (\*)FROM table\_name;

*For example*:

SELECT \* FROM student;



# Alter Table – Adding Column

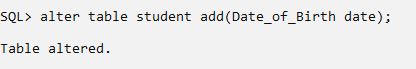
**Theory**: The ALTER TABLE statement allows you to modify the structure of existing tables without losing data.

*Query*:

ALTER TABLE table\_name ADD column\_name datatype;

*For example*:

ALTER table student ADD(Date\_of\_Birth date);



# Updating data

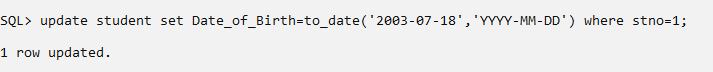
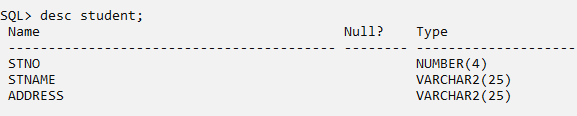
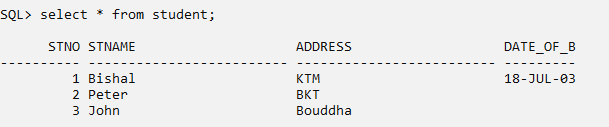
**Theory**: The UPDATE statement is used to modify existing records in a table.

*Query*:

UPDATE table\_name

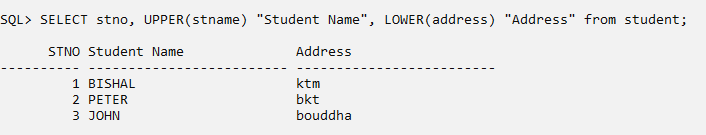
SET column1 = value1, column2 = value2, ...WHERE condition;

*For example*:

UPDATE student SET Date\_of\_Birth=to\_date('2003-07-18','YYYY-MM-DD') WHERE stno=1;

# Displaying data in UpperCase and LowerCase

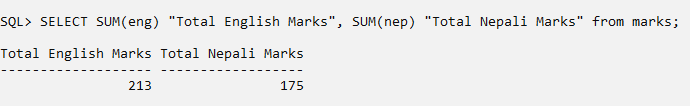
**Theory**: SQL functions can transform data during retrieval. String functions like UPPER and LOWER modify the case of character data, which is useful for standardizing output or performing case-insensitive comparisons. Other common string functions include SUBSTR, CONCAT, LENGTH, TRIM, REPLACE .

**Query:**

# Displaying the sum of data

**Theory**: Aggregate functions perform calculations on sets of rows, returning a single value. Common aggregate functions include SUM, AVG, COUNT, MAX and MIN. These functions are often used with GROUP BY to calculate statistics for different categories of data.

**Query:**



# Order By Clause

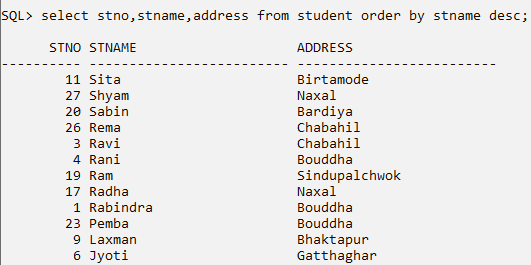
**Theory**: The ORDER BY clause sorts query results based on one or more columns. Data can be sorted in ascending (ASC, the default) or descending (DESC) order. When sorting by multiple columns, the first column determines the primary sort order, with subsequent columns used to break ties. Sorting can be performed on columns that appear in the SELECT list or on expressions. The ORDER BY processing occurs after most other query operations, including WHERE filtering and GROUP BY aggregation.

*Query*:

SELECT column1, column2, ...FROM table\_name

ORDER BY column1, column2, ... ASC|DESC;

*For example*:



# Group By Clause

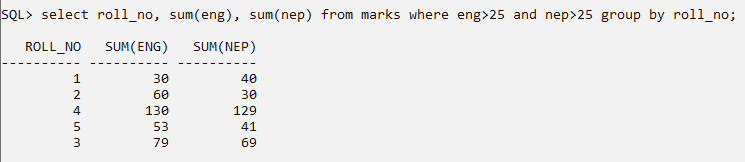
**Theory:** The GROUP BY clause organizes rows with the same values in specified columns into groups, which can then be processed by aggregate functions.

*Query*:

SELECT column\_name(s) FROM table\_name WHERE condition

GROUP BY column\_name(s);

*For example*:

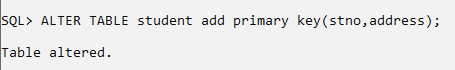


# Add Primary key on existing table

**Theory:** Primary keys are constraints that uniquely identify each record in a table. They enforce entity integrity by ensuring no duplicate or NULL values exist in the key column(s).

Query:

ALTER TABLE student add primary key(stno, address);

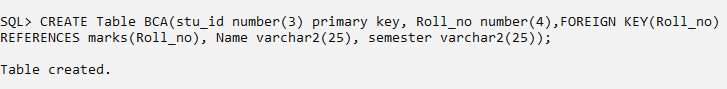


# Creating table with Foreign Key

**Theory:** Foreign keys establish and enforce relationships between tables, forming the foundation of relational database design. A foreign key in one table references a primary key in another table, creating a parent-child relationship. Foreign keys enforce referential integrity, preventing orphaned records by ensuring that values in the child table match values in the parent table or are NULL.

*Query*:

CREATE TABLE TABLE\_NAME(Column 1 datatype, Column 2 datatype, Column 3 datatype FOREIGN KEY REFERENCES Table\_name(Column name) );

*For example*:

# SQL Join

**Theory:** Joins are operations that combine rows from two or more tables based on related columns. Different types of joins retrieve different sets of data: INNER JOIN returns only matching rows from both tables, LEFT JOIN returns all rows from the left table and matching rows from the right, RIGHT JOIN returns all rows from the right table and matching rows from the left, and FULL JOIN returns all rows when there is a match in either table.

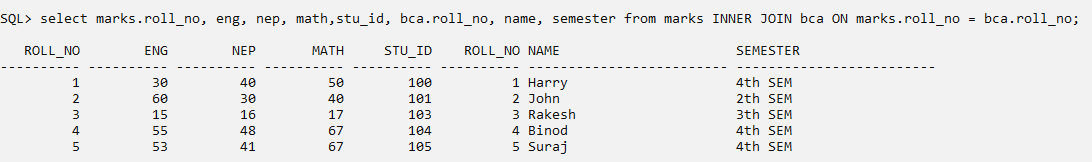
## Inner Join

*Query*:

SELECT column\_name(s) FROM table1

INNER JOIN table2 ON table1.column\_name = table2.column\_name;

*For example*:

SELECT marks.roll\_no, eng, nep, math,stu\_id, bca.roll\_no, name, semester from marks LEFT JOIN bca ON marks.roll\_no = bca.roll\_no;

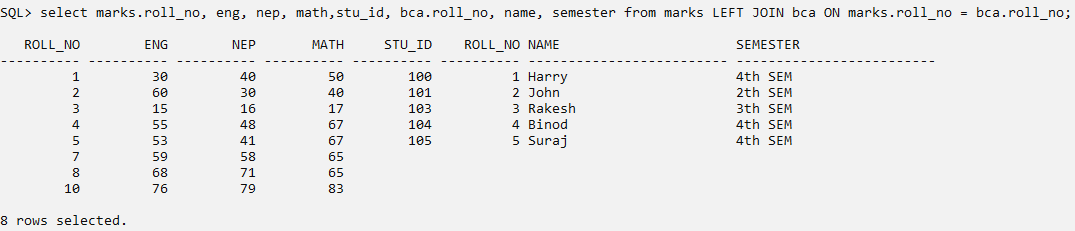
## Left Join

*Query*:

SELECT column\_name(s) FROM table1

LEFT JOIN table2 ON table1.column\_name = table2.column\_name;

*For example*:

SELECT marks.roll\_no, eng, nep, math,stu\_id, bca.roll\_no, name, semester from marks LEFT JOIN bca ON marks.roll\_no = bca.roll\_no;

# PL/SQL Block

**Theory:** PL/SQL (Procedural Language for SQL) extends SQL with procedural programming capabilities. A PL/SQL block is the basic unit of a PL/SQL program, consisting of three sections: declaration (optional, for defining variables, constants, and other program elements), execution (required, containing the procedural code), and exception handling (optional, for managing errors).

*Query*:

DECLARE

declaration statements;

BEGIN

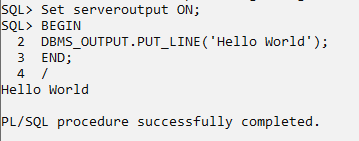
executable statements

EXCEPTIONS

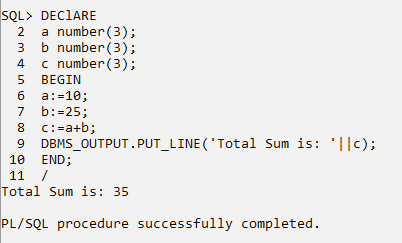
exception handling statements

END;

## First Program



## Sum of two numbers



# PL/SQL Functions

**Theory:** PL/SQL functions are named blocks that return a value, making them ideal for calculations and transformations. They can accept parameters, process data, and return a result of a specified data type. Functions can be called from SQL statements, PL/SQL blocks, or other functions, providing reusable logic throughout the database.

Query:

CREATE [OR REPLACE] FUNCTION function\_name

(parameter\_name type [, …])

RETURN return\_datatype

{IS | AS}

BEGIN

— program code

[EXCEPTION

exception\_section;

END [function\_name];

## Adder Function

CREATE OR REPLACE function adder(a in number, b in number)

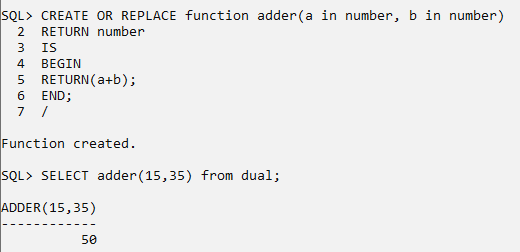
RETURN number

IS

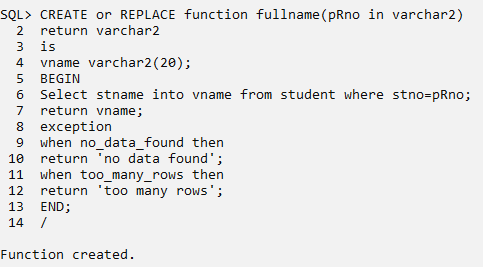
BEGIN

RETURN(a+b);

END;

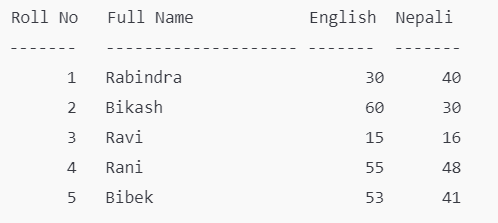
/

## Full name Function





**Output:**



# Creating Procedure

**Theory**: Procedures are PL/SQL blocks that perform specific tasks but don't return values directly (unlike functions). They are ideal for operations that modify database data or perform complex sequences of actions.

*Query*:

CREATE [or REPLACE] PROCEDURE proce­dure\_name(parameters)

IS [declaration\_section]

BEGIN executable\_section

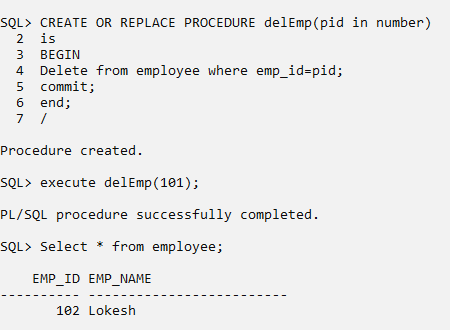
[ EXCEPTION exception\_section]

END;

## To insert into table

## To Update data

## To Delete data



# Creating Triggers

**Theory:** Triggers are special PL/SQL programs that automatically execute in response to specific database events, such as INSERT, UPDATE, DELETE operations on a table.

*Query*:

CREATE OR REPLACE TRIGGER trigger\_name

{BEFORE or AFTER or INSTEAD OF}

INSERT or UPDATE or DELETE

[OF column\_name ON Table\_name]

[FOR EACH ROW]

DECLARE Declaration section

BEGIN

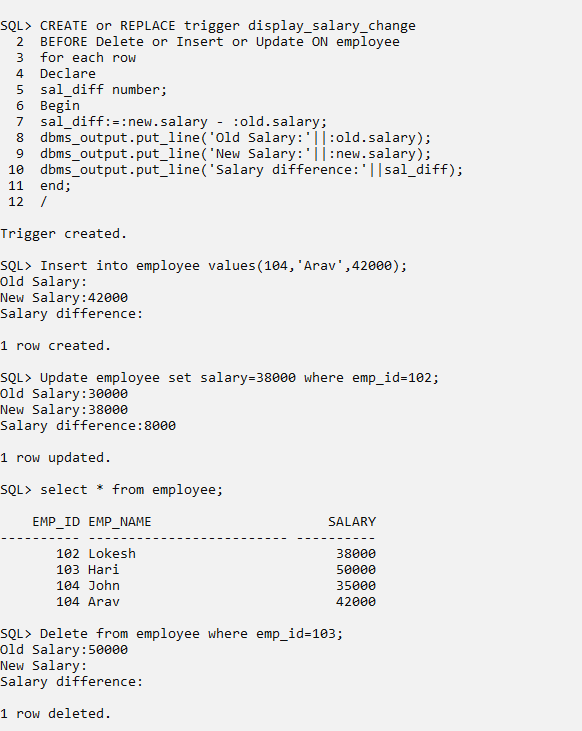
Execution section

EXCEPTION Exception section

END;

/

*For example*:



# Creating Index

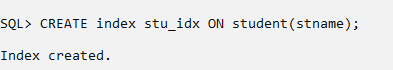
**Theory:** Indexes are database objects that improve query performance by providing faster access paths to table data. They work similarly to book indexes, allowing the database to find rows matching certain criteria without scanning the entire table.

*Query*:

CREATE INDEX index\_name ON table\_name (column1, column2, ...);

*For example*:

CREATE index stu\_idx ON student(stname);



# Renaming Index

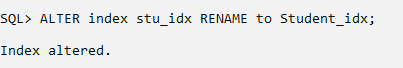
**Theory:** Database object names should be clear and meaningful to facilitate maintenance. The ALTER INDEX statement allows you to rename an existing index without rebuilding it, which is more efficient than dropping and recreating it.

*Query*:

ALTER INDEX old\_index\_name RENAME to new\_index\_name;

*For example*:

ALTER index stu\_idx RENAME to Student\_idx;



# Dropping Index

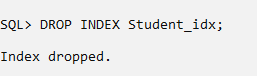
**Theory:** Dropping an index removes it from the database, which might be necessary when it's no longer needed, when table usage patterns have changed, or when restructuring the database. The DROP INDEX statement immediately removes the index, potentially improving the performance of data modification operations at the cost of slower queries.

*Query*:

DROP INDEX index\_name;

*For example*:

DROP INDEX Student\_idx;



# Access Control

**Theory:** Database security is implemented through various mechanisms, with privilege management being a fundamental component. Oracle uses a privilege-based security model where users are granted permissions to perform specific operations on specific database objects.

## Grant Privileges on Table

*Query*:

GRANT privileges ON Object (table\_name) TO user;

*For example*:

GRANT Select, Insert, Update ON employee TO John;

GRANT Select, Insert, Update ON employee TO public; (anyone can access)

## Revoke Privileges on Table

*Query*:

REVOKE privileges ON Object (table\_name) FROM user;

*For example*:

REVOKE Select, Insert, Update ON employee FROM John;

REVOKE Select, Insert, Update ON employee FROM public; (anyone can access)

REVOKE all ON marks FROM Rakesh;