Week2 - Day 1

Installing WSL

Understanding file permissions in Linux:

In Linux and Unix-like operating systems, file permissions are typically represented in either symbolic form (e.g., rwxr-xr--) or numeric form (also called **octal** or **decimal form**).

Understanding File Permissions (Octal Form)

File permissions in Linux are represented by three categories:

1. **Owner Permissions (User)**: Who owns the file (read, write, execute).
2. **Group Permissions**: Permissions for the group associated with the file.
3. **Other (World) Permissions**: Permissions for all other users.

Each of these categories has three types of permissions:

- **Read (r)**: Value = 4
- **Write (w)**: Value = 2
- **Execute (x)**: Value = 1

The permission values are added together for each category (Owner, Group, Other), creating a 3-digit **octal** value (where each digit can be from 0 to 7). Here's how the octal values work:

Permission	Read (r)	Write (w)	Execute (x)	Sum (Octal)
None	No (---)	No (---)	No (---)	0
Execute	No (---)	No (---)	Yes (--x)	1
Write	No (---)	Yes (w--)	No (---)	2
Write + Execute	No (---)	Yes (w--)	Yes (--x)	3

Permission	**Read (r)**	**Write (w)**	**Execute (x)**	**Sum (Octal)**
Read	Yes (r--)	No (---)	No (---)	4
Read + Execute	Yes (r--)	No (---)	Yes (--x)	5
Read + Write	Yes (rw-)	Yes (w--)	No (---)	6
Full (Read + Write + Execute)	Yes (rw-)	Yes (w--)	Yes (--x)	7

So, for example:

- **rwx** (read, write, execute) → 4 + 2 + 1 = 7
- **r-x** (read, no write, execute) → 4 + 0 + 1 = 5
- **rw-** (read, write, no execute) → 4 + 2 + 0 = 6

Example: File Permissions in Octal Form

Let's say a file has the following permissions:

-rwxr-xr--

- **Owner**: rwx → 7
- **Group**: r-x → 5
- **Other**: r-- → 4

So, the octal representation of the above file permissions is **754**.

Creating nested directories:

```
mkdir -p dir1/dir2/dir3
```

Creating files using touch:
touch hello.txt -> create a file named touch
touch {1..5}.txt -> will create 5 files
touch file1.txt file2.txt file3.txt -> create 3 files named file1.txt ,file2.txt, file3.txt

Copy files:

cp [source] [destination]
cp -rf /source/directory /path/to/destination/

r – recursively copy all files and directories
f – force copy

<u>Search for a specific word in a file</u>:

grep "hello" file.txt

- -i: Case-insensitive search
- -r or -R: Recursive search in directories
- -n: Show line numbers
- -v: Invert the match (show non-matching lines)
- -w: Match whole words
- -c: Count occurrences
- -E: Extended regular expressions (allows more complex patterns)

man – stands for manual, used to look for documentation of a partiular command.

<u>Chmod -</u>
The chmod command in Linux is used to change the permissions of a file or directory. The permissions define who can read, write, or execute a file.

- chmod 755 file.txt: Sets rwx (7) for the owner, rx (5) for the group, and rx (5) for others.
- chmod 644 file.txt: Sets rw (6) for the owner and r (4) for the group and others.
- chmod u+x file.txt : Adds execute permission to the owner of the file.
- chmod g-w file.txt: Removes write permission from the group.

<u>Assignment-</u> Created a school's database and tables based on an ER diagram inorder to understand working of stored procedures and indexes.

```
CREATE DATABASE SchoolDB;
USE SchoolDB;

CREATE TABLE Students (
   student_id INT PRIMARY KEY AUTO_INCREMENT,
   first_name VARCHAR(50),
   last_name VARCHAR(50),
   dob DATE,
   gender ENUM('Male', 'Female', 'Other'),
   class_id INT,
   admission_date DATE,
   FOREIGN KEY (class_id) REFERENCES Classes(class_id),
   INDEX idx_class_id (class_id),
   INDEX idx_dob (dob)
);

CREATE TABLE Classes (
   class_id INT PRIMARY KEY AUTO_INCREMENT,
   class_name VARCHAR(50) NOT NULL,
   section VARCHAR(10)
);

CREATE TABLE Teachers (
   teacher_id INT PRIMARY KEY AUTO_INCREMENT,
   first_name VARCHAR(50),
   last_name VARCHAR(50),
   subject VARCHAR(50)
);

CREATE TABLE Subjects (
   subject_id INT PRIMARY KEY AUTO_INCREMENT,
   subject_name VARCHAR(50) NOT NULL,
   teacher_id INT,
   FOREIGN KEY (teacher_id) REFERENCES Teachers(teacher_id),
   INDEX idx_teacher_id (teacher_id)
);

-- Creating the Enrollments Table
```

```
CREATE TABLE Enrollments (
    enrollment_id INT PRIMARY KEY AUTO_INCREMENT,
    student_id INT,
    subject_id INT,
    enroll_date DATE,
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (subject_id) REFERENCES Subjects(subject_id),
    INDEX idx_student_id (student_id),
    INDEX idx_subject_id (subject_id)
);

-- Automatic Data Insertion (Generating Random Data for 10,000 rows
using loops)
INSERT INTO Classes (class_name, section)
VALUES ('Grade 1', 'A'), ('Grade 1', 'B'), ('Grade 2', 'A'), ('Grade 2', 'B'),
('Grade 3', 'A');

INSERT INTO Teachers (first_name, last_name, subject)
VALUES ('John', 'Doe', 'Math'), ('Jane', 'Smith', 'Science'), ('Mike',
'Brown', 'English'), ('Emily', 'Johnson', 'History');

INSERT INTO Subjects (subject_name, teacher_id)
VALUES ('Mathematics', 1), ('Science', 2), ('English', 3), ('History', 4);

DELIMITER $$
CREATE PROCEDURE InsertStudents()
BEGIN
    DECLARE i INT DEFAULT 1;
    WHILE i <= 10000 DO
        INSERT INTO Students (first_name, last_name, dob, gender,
class_id, admission_date)
        VALUES (CONCAT('Student', i), 'Lastname', DATE_ADD('2000-01-
01', INTERVAL FLOOR(RAND() * 5000) DAY),
                IF(FLOOR(RAND() * 2) = 0, 'Male', 'Female'),
                FLOOR(1 + (RAND() * 5)), CURDATE());
        SET i = i + 1;
    END WHILE;
END $$
DELIMITER ;
```

```
CALL InsertStudents();


DELIMITER $$
CREATE PROCEDURE InsertEnrollments()
BEGIN
   DECLARE i INT DEFAULT 1;
   WHILE i <= 10000 DO
      INSERT INTO Enrollments (student_id, subject_id, enroll_date)
      VALUES (i, FLOOR(1 + (RAND() * 4)), CURDATE());
      SET i = i + 1;
   END WHILE;
END $$
DELIMITER ;
CALL InsertEnrollments();

-- Trigger
DELIMITER $$
CREATE TRIGGER before_student_insert
BEFORE INSERT ON Students
FOR EACH ROW
BEGIN
   IF NEW.dob > CURDATE() THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Date of birth
cannot be in the future';
   END IF;
END $$
DELIMITER ;

-- 1. Retrieve all students enrolled in Mathematics
SELECT s.first_name, s.last_name FROM Students s
JOIN Enrollments e ON s.student_id = e.student_id
JOIN Subjects sub ON e.subject_id = sub.subject_id
WHERE sub.subject_name = 'Mathematics';

-- 2. Update teacher assignment for a subject
UPDATE Subjects SET teacher_id = 3 WHERE subject_name = 'History';
```