Experiment 1

Code:

*import* sqlite3

*# Connect to SQLite (creates a new database file named 'sample\_database.db')*

connection = sqlite3.connect("sample\_database.db")

*# Create a cursor object to interact with the database*

cursor = connection.cursor()

*# Step 3: Create a table*

*# This table will have columns for ID, name, and age*

create\_table\_query = """

CREATE TABLE IF NOT EXISTS users (

    id INTEGER PRIMARY KEY AUTOINCREMENT,

    name TEXT NOT NULL,

    age INTEGER

);

"""

cursor.execute(create\_table\_query)

*# Step 4: Insert sample data*

insert\_data\_query = """

INSERT INTO users (name, age)

VALUES

    ('Alice', 25),

    ('Bob', 30),

    ('Charlie', 22);

"""

cursor.execute(insert\_data\_query)

*# Commit the changes to the database*

connection.commit()

*# Step 5: Fetch and display the data to verify*

cursor.execute("SELECT \* FROM users")

rows = cursor.fetchall()

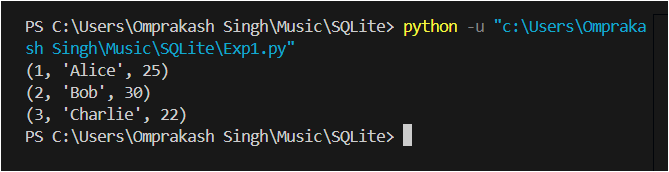
*for* row *in* rows:

    print(row)

*# Close the connection*

connection.close()

Output:



Experiment 2

Code:

*import sqlite3*

*# Connect to the SQLite database*

*connection = sqlite3.connect("sample\_database.db")*

*cursor = connection.cursor()*

*# Step 1: Create the 'users' table if it doesn't exist*

*create\_table\_query = """*

*CREATE TABLE IF NOT EXISTS users (*

*id INTEGER PRIMARY KEY AUTOINCREMENT,*

*name TEXT NOT NULL,*

*age INTEGER*

*);*

*"""*

*cursor.execute(create\_table\_query)*

*# Insert initial data*

*cursor.execute("INSERT INTO users (name, age) VALUES ('Alice', 25)")*

*cursor.execute("INSERT INTO users (name, age) VALUES ('Bob', 30)")*

*cursor.execute("INSERT INTO users (name, age) VALUES ('Charlie', 22)")*

*connection.commit()  # Commit initial data*

*# Step 2: SELECT Operation - Retrieve Specific Data*

*# Retrieve all records from the users table*

*cursor.execute("SELECT \* FROM users")*

*all\_users = cursor.fetchall()*

*print("All Users:")*

*for user in all\_users:*

*print(user)*

*# Retrieve specific records where age > 25*

*cursor.execute("SELECT \* FROM users WHERE age > 25")*

*older\_users = cursor.fetchall()*

*print("\nUsers older than 25:")*

*for user in older\_users:*

*print(user)*

*# Step 3: INSERT Operation - Add a New Record*

*cursor.execute("INSERT INTO users (name, age) VALUES (?, ?)", ('David', 28))*

*connection.commit()  # Commit the new record*

*# Confirm the insertion*

*cursor.execute("SELECT \* FROM users WHERE name = 'David'")*

*print("\nNew User:")*

*print(cursor.fetchone())*

*# Step 4: UPDATE Operation - Modify an Existing Record*

*cursor.execute("UPDATE users SET age = ? WHERE name = ?", (26, 'Alice'))*

*connection.commit()*

*# Confirm the update*

*cursor.execute("SELECT \* FROM users WHERE name = 'Alice'")*

*print("\nUpdated User Record:")*

*print(cursor.fetchone())*

*# Step 5: DELETE Operation - Remove a Record*

*cursor.execute("DELETE FROM users WHERE name = ?", ('Charlie',))*

*connection.commit()*

*# Confirm the deletion*

*cursor.execute("SELECT \* FROM users")*

*remaining\_users = cursor.fetchall()*

*print("\nRemaining Users after Deletion:")*

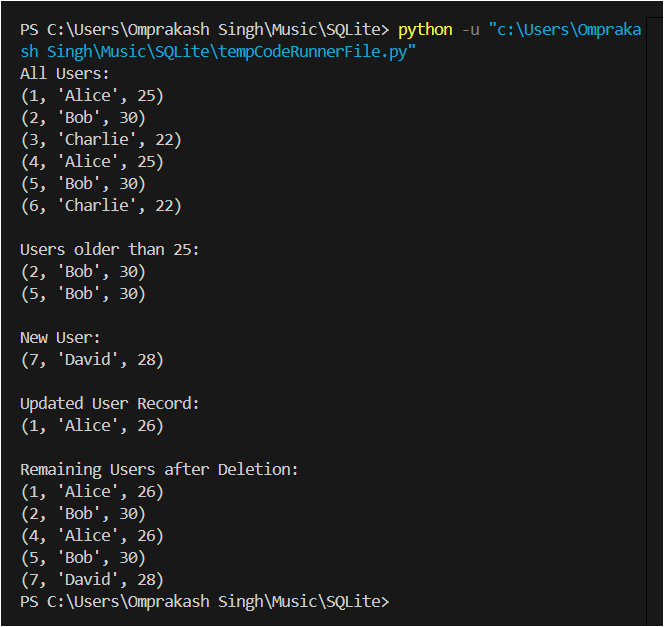
*for user in remaining\_users:*

*print(user)*

*# Step 6: Close the Database Connection*

*connection.close()*

Output:



Experiment 3

Code:

*import sqlite3*

*# Connect to the database (or create it if it doesn’t exist)*

*connection = sqlite3.connect("school.db")*

*cursor = connection.cursor()*

*# Create the Students table with PRIMARY KEY, UNIQUE, and NOT NULL constraints*

*cursor.execute("""*

*CREATE TABLE IF NOT EXISTS students (*

*student\_id INTEGER PRIMARY KEY AUTOINCREMENT,*

*name TEXT NOT NULL,*

*email TEXT UNIQUE NOT NULL*

*);*

*""")*

*# Create the Courses table with PRIMARY KEY and NOT NULL constraints*

*cursor.execute("""*

*CREATE TABLE IF NOT EXISTS courses (*

*course\_id INTEGER PRIMARY KEY AUTOINCREMENT,*

*course\_name TEXT NOT NULL,*

*course\_code TEXT UNIQUE NOT NULL*

*);*

*""")*

*# Insert sample data into the Students table*

*cursor.execute("INSERT INTO students (name, email) VALUES ('Alice', 'alice@example.com')")*

*cursor.execute("INSERT INTO students (name, email) VALUES ('Bob', 'bob@example.com')")*

*# Insert sample data into the Courses table*

*cursor.execute("INSERT INTO courses (course\_name, course\_code) VALUES ('Math', 'MATH101')")*

*cursor.execute("INSERT INTO courses (course\_name, course\_code) VALUES ('Physics', 'PHY101')")*

*# Commit changes*

*connection.commit()*

*# Query and print all students*

*print("Students:")*

*cursor.execute("SELECT \* FROM students")*

*for row in cursor.fetchall():*

*print(row)*

*# Query and print all courses*

*print("\nCourses:")*

*cursor.execute("SELECT \* FROM courses")*

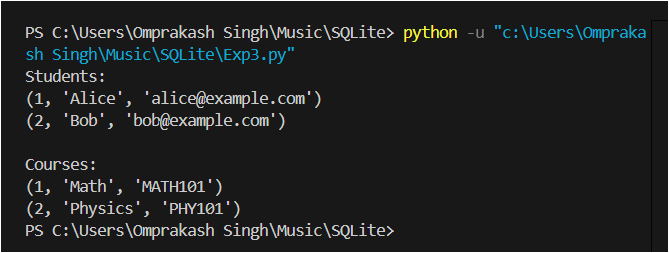
*for row in cursor.fetchall():*

*print(row)*

*# Close the connection*

*connection.close()*

Output:



Experiment 4

Code:

*import sqlite3*

*# Connect to the database*

*connection = sqlite3.connect("school\_management.db")*

*cursor = connection.cursor()*

*# Create the tables with constraints*

*cursor.executescript("""*

*CREATE TABLE IF NOT EXISTS students (*

*student\_id INTEGER PRIMARY KEY,*

*name TEXT NOT NULL,*

*email TEXT UNIQUE NOT NULL*

*);*

*CREATE TABLE IF NOT EXISTS courses (*

*course\_id INTEGER PRIMARY KEY,*

*course\_name TEXT NOT NULL,*

*course\_code TEXT UNIQUE NOT NULL*

*);*

*CREATE TABLE IF NOT EXISTS enrollments (*

*enrollment\_id INTEGER PRIMARY KEY,*

*student\_id INTEGER NOT NULL,*

*course\_id INTEGER NOT NULL,*

*enrollment\_date TEXT NOT NULL,*

*FOREIGN KEY (student\_id) REFERENCES students (student\_id),*

*FOREIGN KEY (course\_id) REFERENCES courses (course\_id)*

*);*

*""")*

*# Insert sample data into tables*

*cursor.executescript("""*

*INSERT OR IGNORE INTO students (student\_id, name, email) VALUES*

*(1, 'Alice', 'alice@example.com'),*

*(2, 'Bob', 'bob@example.com'),*

*(3, 'Charlie', 'charlie@example.com');*

*INSERT OR IGNORE INTO courses (course\_id, course\_name, course\_code) VALUES*

*(1, 'Math', 'MATH101'),*

*(2, 'Physics', 'PHY101');*

*INSERT OR IGNORE INTO enrollments (enrollment\_id, student\_id, course\_id, enrollment\_date) VALUES*

*(1, 1, 1, '2024-09-01'),*

*(2, 2, 2, '2024-09-01'),*

*(3, 3, 1, '2024-09-02');*

*""")*

*connection.commit()*

*# 1. INNER JOIN: Get students and the courses they are enrolled in*

*print("Students and Courses:")*

*cursor.execute("""*

*SELECT students.name AS student\_name, courses.course\_name AS course\_name*

*FROM enrollments*

*JOIN students ON enrollments.student\_id = students.student\_id*

*JOIN courses ON enrollments.course\_id = courses.course\_id;*

*""")*

*print(cursor.fetchall())*

*# 2. WHERE Clause: Students enrolled in Math*

*print("\nStudents Enrolled in Math:")*

*cursor.execute("""*

*SELECT students.name*

*FROM enrollments*

*JOIN students ON enrollments.student\_id = students.student\_id*

*JOIN courses ON enrollments.course\_id = courses.course\_id*

*WHERE courses.course\_name = 'Math';*

*""")*

*print(cursor.fetchall())*

*# 3. ORDER BY Clause: Enrollments ordered by date*

*print("\nEnrollments Ordered by Date:")*

*cursor.execute("""*

*SELECT students.name, courses.course\_name, enrollments.enrollment\_date*

*FROM enrollments*

*JOIN students ON enrollments.student\_id = students.student\_id*

*JOIN courses ON enrollments.course\_id = courses.course\_id*

*ORDER BY enrollments.enrollment\_date;*

*""")*

*print(cursor.fetchall())*

*# 4. GROUP BY Clause: Count students per course*

*print("\nStudent Count per Course:")*

*cursor.execute("""*

*SELECT courses.course\_name, COUNT(enrollments.student\_id)*

*FROM enrollments*

*JOIN courses ON enrollments.course\_id = courses.course\_id*

*GROUP BY courses.course\_name;*

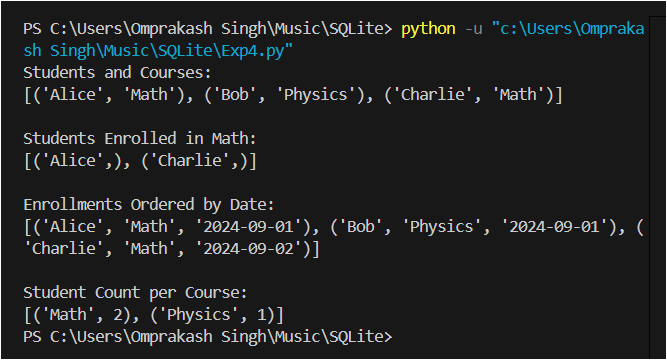
*""")*

*print(cursor.fetchall())*

*# Close the connection*

*connection.close()*

Output:



Experiment 5

Code:

*import sqlite3*

*# Connect to the SQLite database*

*connection = sqlite3.connect("bank.db")*

*cursor = connection.cursor()*

*# Create a bank\_accounts table*

*cursor.execute("""*

*CREATE TABLE IF NOT EXISTS bank\_accounts (*

*account\_id INTEGER PRIMARY KEY,*

*name TEXT NOT NULL,*

*balance REAL NOT NULL CHECK(balance >= 0)*

*);*

*""")*

*# Insert initial data*

*cursor.executescript("""*

*INSERT OR IGNORE INTO bank\_accounts (account\_id, name, balance) VALUES*

*(1, 'Alice', 500.0),*

*(2, 'Bob', 300.0);*

*""")*

*connection.commit()*

*# Function to perform a transaction: transfer money from one account to another*

*def transfer\_funds(sender\_id, receiver\_id, amount):*

*try:*

*# Begin a transaction*

*connection.execute("BEGIN")*

*# Check sender's balance*

*cursor.execute("SELECT balance FROM bank\_accounts WHERE account\_id = ?", (sender\_id,))*

*sender\_balance = cursor.fetchone()[0]*

*if sender\_balance < amount:*

*raise ValueError("Insufficient funds in sender's account.")*

*# Deduct amount from sender*

*cursor.execute("UPDATE bank\_accounts SET balance = balance - ? WHERE account\_id = ?", (amount, sender\_id))*

*# Add amount to receiver*

*cursor.execute("UPDATE bank\_accounts SET balance = balance + ? WHERE account\_id = ?", (amount, receiver\_id))*

*# Commit transaction*

*connection.commit()*

*print("Transaction completed successfully.")*

*except Exception as e:*

*# Rollback transaction on error*

*connection.rollback()*

*print("Transaction failed:", e)*

*# Perform a transfer operation*

*transfer\_funds(1, 2, 150.0)*

*# Check updated balances*

*cursor.execute("SELECT \* FROM bank\_accounts")*

*print("\nUpdated Account Balances:")*

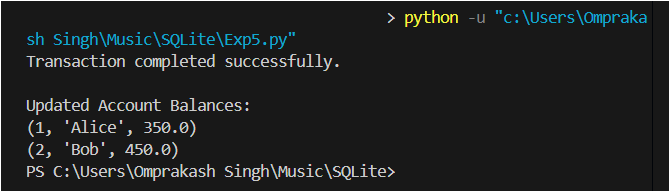
*for row in cursor.fetchall():*

*print(row)*

*# Close the connection*

*connection.close()*

Output:



Experiment 6

Code:

*import sqlite3*

*import time*

*# Connect to the SQLite database*

*connection = sqlite3.connect("bank.db")*

*cursor = connection.cursor()*

*# Create a bank\_accounts table if it doesn't exist*

*cursor.execute("""*

*CREATE TABLE IF NOT EXISTS bank\_accounts (*

*account\_id INTEGER PRIMARY KEY,*

*name TEXT NOT NULL,*

*balance REAL NOT NULL CHECK(balance >= 0)*

*);*

*""")*

*# Insert sample data*

*cursor.executescript("""*

*INSERT OR IGNORE INTO bank\_accounts (account\_id, name, balance) VALUES*

*(1, 'Alice', 500.0),*

*(2, 'Bob', 300.0),*

*(3, 'Charlie', 400.0),*

*(4, 'David', 200.0),*

*(5, 'Eve', 600.0);*

*""")*

*connection.commit()*

*# Step 1: Create Indexes*

*cursor.execute("CREATE INDEX IF NOT EXISTS idx\_name ON bank\_accounts(name);")*

*cursor.execute("CREATE INDEX IF NOT EXISTS idx\_balance ON bank\_accounts(balance);")*

*connection.commit()*

*# Function to measure query execution time*

*def measure\_query\_time(query):*

*start\_time = time.time()*

*cursor.execute(query)*

*results = cursor.fetchall()*

*end\_time = time.time()*

*print(f"Query results: {results}")*

*print(f"Execution time: {end\_time - start\_time:.6f} seconds\n")*

*# Step 2: Query without index*

*print("Querying without index (for balance > 300):")*

*measure\_query\_time("SELECT \* FROM bank\_accounts WHERE balance > 300;")*

*# Step 3: Query with index on name*

*print("Querying with index on name (search for 'Alice'):")*

*measure\_query\_time("SELECT \* FROM bank\_accounts WHERE name = 'Alice';")*

*# Step 4: Query with index on balance*

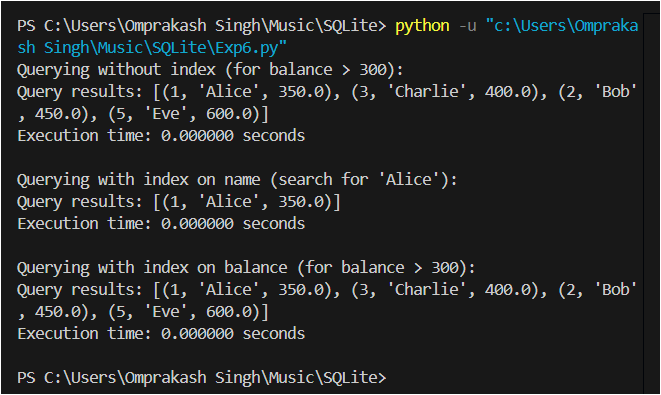
*print("Querying with index on balance (for balance > 300):")*

*measure\_query\_time("SELECT \* FROM bank\_accounts WHERE balance > 300;")*

*# Close the connection*

*connection.close()*

Output:



Experiment 7

Code:

*import sqlite3*

*# Connect to the SQLite database*

*connection = sqlite3.connect("bank.db")*

*cursor = connection.cursor()*

*# Create the bank\_accounts table*

*cursor.execute("""*

*CREATE TABLE IF NOT EXISTS bank\_accounts (*

*account\_id INTEGER PRIMARY KEY,*

*name TEXT NOT NULL,*

*balance REAL NOT NULL CHECK(balance >= 0)*

*);*

*""")*

*# Create a transactions table to log changes*

*cursor.execute("""*

*CREATE TABLE IF NOT EXISTS transactions (*

*transaction\_id INTEGER PRIMARY KEY AUTOINCREMENT,*

*account\_id INTEGER,*

*change REAL,*

*transaction\_date TEXT DEFAULT CURRENT\_TIMESTAMP,*

*FOREIGN KEY (account\_id) REFERENCES bank\_accounts (account\_id)*

*);*

*""")*

*# Step 1: Create a Trigger to log balance changes*

*cursor.execute("""*

*CREATE TRIGGER IF NOT EXISTS log\_balance\_change*

*AFTER UPDATE ON bank\_accounts*

*FOR EACH ROW*

*BEGIN*

*INSERT INTO transactions (account\_id, change)*

*VALUES (NEW.account\_id, NEW.balance - OLD.balance);*

*END;*

*""")*

*# Step 2: Insert sample data*

*cursor.executescript("""*

*INSERT OR IGNORE INTO bank\_accounts (account\_id, name, balance) VALUES*

*(1, 'Alice', 500.0),*

*(2, 'Bob', 300.0);*

*""")*

*connection.commit()*

*# Step 3: Update a balance (this will trigger the logging)*

*cursor.execute("UPDATE bank\_accounts SET balance = balance + 200 WHERE account\_id = 1;")*

*connection.commit()*

*# Step 4: Create a view to summarize account balances*

*cursor.execute("""*

*CREATE VIEW IF NOT EXISTS account\_summary AS*

*SELECT name, balance FROM bank\_accounts;*

*""")*

*# Query the view to show account summary*

*print("Account Summary:")*

*cursor.execute("SELECT \* FROM account\_summary;")*

*for row in cursor.fetchall():*

*print(row)*

*# Step 5: Query the transactions table to see logged changes*

*print("\nTransaction Log:")*

*cursor.execute("SELECT \* FROM transactions;")*

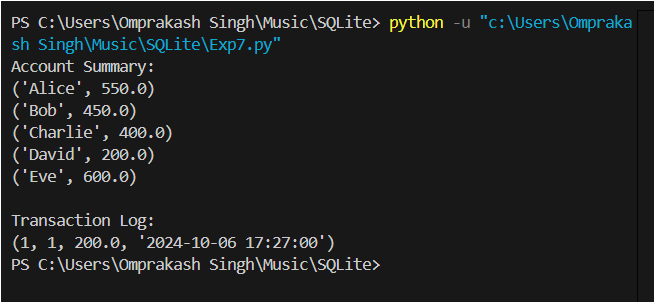
*for row in cursor.fetchall():*

*print(row)*

*# Close the connection*

*connection.close()*

Output:



Experiment 8

Code:

*import sqlite3*

*# Connect to the SQLite database*

*connection = sqlite3.connect("bank.db")*

*cursor = connection.cursor()*

*# Create the bank\_accounts table*

*def create\_table():*

*cursor.execute("""*

*CREATE TABLE IF NOT EXISTS bank\_accounts (*

*account\_id INTEGER PRIMARY KEY,*

*name TEXT NOT NULL,*

*balance REAL NOT NULL CHECK(balance >= 0)*

*);*

*""")*

*connection.commit()*

*# Function to create a new account*

*def create\_account(name, balance):*

*cursor.execute("INSERT INTO bank\_accounts (name, balance) VALUES (?, ?);", (name, balance))*

*connection.commit()*

*# Function to read all accounts*

*def read\_accounts():*

*cursor.execute("SELECT \* FROM bank\_accounts;")*

*return cursor.fetchall()*

*# Function to update an account's balance*

*def update\_account(account\_id, new\_balance):*

*cursor.execute("UPDATE bank\_accounts SET balance = ? WHERE account\_id = ?;", (new\_balance, account\_id))*

*connection.commit()*

*# Function to delete an account*

*def delete\_account(account\_id):*

*cursor.execute("DELETE FROM bank\_accounts WHERE account\_id = ?;", (account\_id,))*

*connection.commit()*

*# Demonstration of CRUD operations*

*create\_table() # Ensure the table exists*

*# Create accounts*

*create\_account('Alice', 500.0)*

*create\_account('Bob', 300.0)*

*print("Accounts after creation:")*

*print(read\_accounts())*

*# Update Alice's balance*

*update\_account(1, 600.0)*

*print("\nAccounts after updating Alice's balance:")*

*print(read\_accounts())*

*# Delete Bob's account*

*delete\_account(2)*

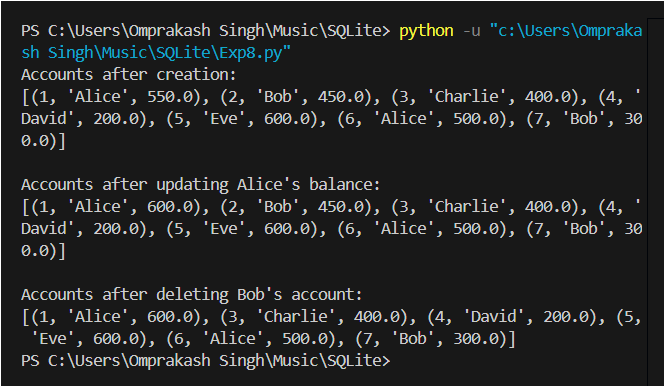
*print("\nAccounts after deleting Bob's account:")*

*print(read\_accounts())*

*# Close the connection*

*connection.close()*

Output:



Experiment 9

Create and configure a Firebase project

**1. Set Up Firebase Account**

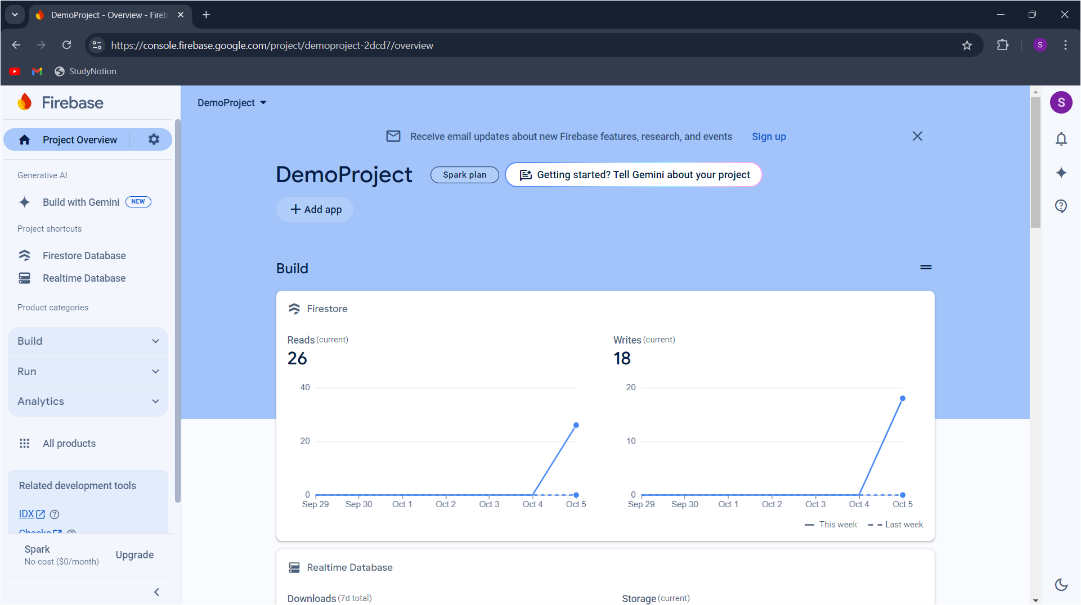
* Sign in at Firebase Console.

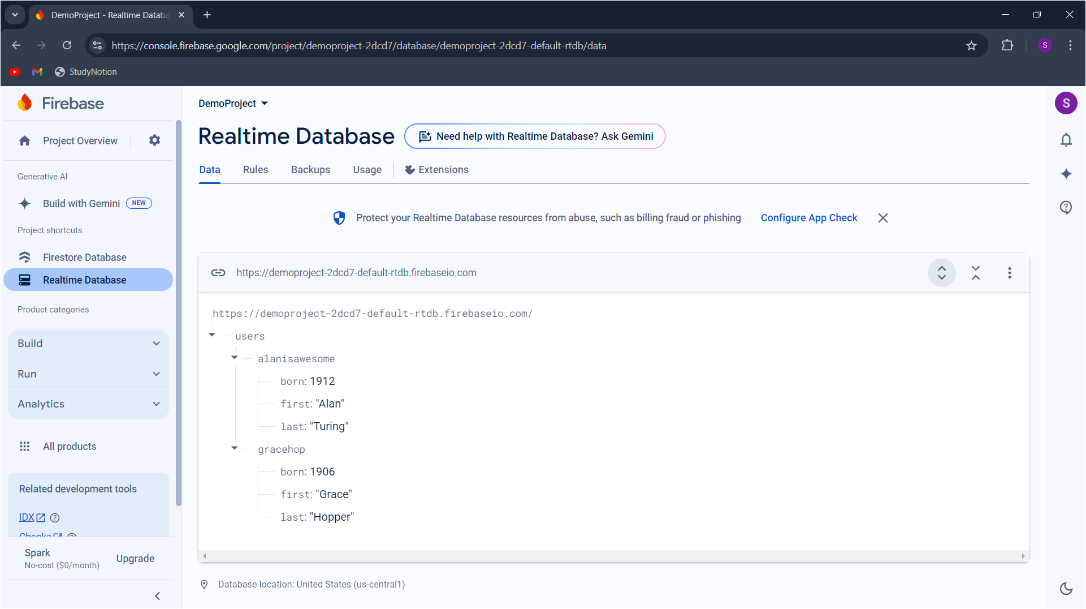
**2. Create New Project**

* Click on "Add project."
* Enter project name and enable Google Analytics (optional).
* Click "Create Project" and wait for completion.

**3. Configure Firebase for Your App**

* **Select Platform**: Choose Web, iOS, or Android.
* **Register App**:
  + Web: Click </> icon, enter nickname, and follow SDK setup instructions.
  + iOS: Click Apple icon, download GoogleService-Info.plist.
  + Android: Click Android icon, download google-services.json.





Experiment 10

Code:

*import firebase\_admin*

*from firebase\_admin import credentials, auth*

*# Initialize Firebase Admin SDK*

*def initialize\_firebase():*

*cred = credentials.Certificate("./serviceAccountKey.json")  # Path to your service account key*

*firebase\_admin.initialize\_app(cred)*

*# Function to create a new user*

*def create\_user(email, password):*

*try:*

*user = auth.create\_user(email=email, password=password)*

*print(f'Successfully created user: {user.uid}')*

*return user.uid  # Return the user UID for further operations*

*except Exception as e:*

*print(f'Error creating user: {str(e)}')*

*# Function to get user information*

*def get\_user\_info(uid):*

*try:*

*user = auth.get\_user(uid)*

*print(f'User ID: {user.uid}')*

*print(f'Email: {user.email}')*

*except Exception as e:*

*print(f'Error fetching user data: {str(e)}')*

*# Function to delete a user*

*def delete\_user(uid):*

*try:*

*auth.delete\_user(uid)*

*print(f'Successfully deleted user: {uid}')*

*except Exception as e:*

*print(f'Error deleting user: {str(e)}')*

*# Example Usage*

*if \_\_name\_\_ == "\_\_main\_\_":*

*initialize\_firebase()*

*# Create a new user*

*user\_uid = create\_user('test@example.com', 'securePassword123')*

*# Fetch user info (use the UID returned from create\_user)*

*if user\_uid:*

*get\_user\_info(user\_uid)*

*# Delete the user (using the same UID)*

*delete\_user(user\_uid)*

Output:

