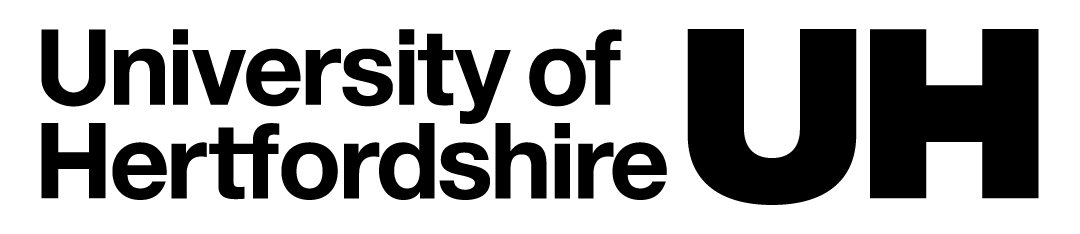
[1]

Data Mining and Discovery

Master of Science in Data Science

SHUBHAM VERMA

22099668

Sv23abk@herts.ac.uk

**SHUBHAM VERMA**

UH_report_cover_set blue.pdf

**SQL Assignment Report**

Creating Database, tables, generating random data

02-Mar-2024

**Department of Physics, Astronomy and Mathematics**

School of Physics, Engineering, and Computer Science

University of Hertfordshire

Contents

[1. Database Schema 3](#_Toc160275759)

[1.1 Creation of database 3](#_Toc160275760)

[*1.1.1* *Using python notebook* 3](#_Toc160275761)

[*1.1.2* *To create a new database in SQLite DB Browser (Another way):* 4](#_Toc160275762)

[1.2 Database diagram 5](#_Toc160275763)

[1.3 Need of separate tables and ethical discussion 6](#_Toc160275764)

[*1.3.1* *Separate tables* 6](#_Toc160275765)

[*1.3.2* *Ethical Discussion* 7](#_Toc160275766)

[2. Tables 8](#_Toc160275767)

[2.1 Location table 8](#_Toc160275768)

[2.2 Department table 9](#_Toc160275769)

[2.3 Employee table 10](#_Toc160275770)

[2.4 Login table 11](#_Toc160275771)

[3. Generation of data 13](#_Toc160275772)

[3.1 Data generation for location table 13](#_Toc160275773)

[3.1.1 Sample data of location table 14](#_Toc160275774)

[3.2 Data generation for department table 14](#_Toc160275775)

[3.2.1 Sample data of department table 15](#_Toc160275776)

[3.3 Data generation for employee table 15](#_Toc160275777)

[3.3.1 Sample data of employee table 16](#_Toc160275778)

[3.4 Data generation for login table 17](#_Toc160275779)

[3.4.1 Sample data of login table 18](#_Toc160275780)

[4. Demonstration 19](#_Toc160275781)

[4.1 joins and selections 19](#_Toc160275782)

[4.2 Different datatypes 20](#_Toc160275783)

# Database Schema

A database schema defines the organization and structure of a database, defining tables, columns, relationships, and constraints. It serves as a model for storing and managing data, ensuring consistency and integrity. Each table represents a specific entity, with fields that define properties. Relationships create connections between entities. Constraints enforce rules and maintain data accuracy. This pattern provides a system framework for storing, retrieving, and manipulating data, facilitating efficient database management, and improving data reliability in a variety of applications, including software development, analysis, and information systems.

## Creation of database

The database schema comprises four tables: location, employee, department, and login. It organizes information systematically, utilizing primary and foreign keys to establish relationships. This design enhances data integrity, minimizes redundancy, and ensures efficient management of diverse organizational data, promoting scalability and adherence to ethical data practices.

### *Using python notebook*

Code imports the library to create dummy data, process dates, and interact with SQLite databases. It uses the English Faker library to create an SQLite database called "ads2database.db" and establish a connection to the cursor to perform database operations.

# Import necessary libraries

!pip install Faker

from faker import Faker

from datetime import datetime, timedelta

import sqlite3

import random

# Initialize Faker with UK English locale

fake = Faker('en\_GB')

# Create a SQLite database

db\_path = '/content/ads2database.db'

connection = sqlite3.connect(db\_path)

cursor = connection.cursor()

### *To create a new database in SQLite DB Browser (Another way):*

Open SQLite DB Browser.

Go to "File" in the menu.

Select "New Database."

Choose a location and provide a name for your new database file.

Click "Save."

This will create an empty SQLite database file at the specified location with the given name. You can then use this database file to create tables, insert data, and perform other database operations.

A screenshot of a computer

Description automatically generated

## Database diagram

A screenshot of a computer

Description automatically generated

The database schema comprises four main tables: location, employee, department, and login. The location table holds information about various places in a building, identified by a unique location\_id. The employee table contains details of individual employees, with a primary key emp\_id. It establishes relationships through foreign keys like emp\_departmentid and emp\_managerid, referencing the same employee table for hierarchy. The department table represents organizational departments, using dep\_hq\_locationid as a foreign key linked to the location table. The login table records employee login and logout events, utilizing foreign keys like emp\_id, login\_locationid, and logout\_locationid to connect with employee and location details. This schema effectively captures relationships such as organizational hierarchy and employee locations. Numeric IDs serve as primary keys for unique identification, while fields like created\_by and lastupdated\_by facilitate tracking system users responsible for data modifications.

## Need of separate tables and ethical discussion

The decision to use separate tables in the database schema is justified by the need to organize and manage different aspects of the information systematically. Each table serves a distinct purpose, facilitating data integrity, reducing redundancy, and improving overall database structure.

### *Separate tables*

The decision for separate tables is justified by the need for efficient organization and management of distinct data elements.

***Location Table***

Separating location information into its table allows efficient management of details specific to each location, such as building name, floor number, and room temperature. This avoids duplicating location details for each employee or department, promoting data normalization.

***Employee Table***

The employee table centralizes information about individuals in the organization, including personal details, role, and status. Using separate tables for employees, departments, and locations allows for easy expansion and modification of employee-related data without affecting other entities.

***Department Table***

The department table focuses on organizational units, storing information like department name and description. By having a dedicated table, changes or updates to department-related data won't impact individual employee details, providing flexibility in managing organizational structure.

***Login Table***

Tracking login events separately in the login table ensures a clear record of employee access to different locations. It simplifies the retrieval of login/logout information without cluttering the employee table with time-sensitive details.

### *Ethical Discussion*

The ethical considerations prioritize data privacy, accuracy, security, and transparency, aligning with responsible and ethical data handling practices.

***Data Privacy:*** The design respects employee privacy by organizing data logically. Sensitive information is stored securely, and access controls can be implemented to restrict unauthorized viewing.

***Data Accuracy:*** Separate tables promote data accuracy and consistency. Updating information in one table doesn't require changes across multiple tables, reducing the risk of errors.

***Data Security:*** By using foreign keys and proper indexing, the schema enhances data security. Unauthorized modifications can be minimized, ensuring the integrity of the information.

***Transparency:*** The design supports transparency in tracking changes with fields like created\_by and lastupdated\_by. This promotes accountability and ethical data management practices.

# Tables

SQL tables are structured data containers in relational databases. Each table is defined with columns specifying data types and constraints. Tables store related information, and their structure is governed by a schema, including primary keys for unique identification and relationships between tables for efficient data retrieval.

## Location table

The "location" table is defined with an integer primary key "location\_id" ensuring uniqueness. It captures details about a building location, including "loc\_building\_name," "loc\_building\_add," and attributes like "loc\_floor\_number" and "loc\_roomid." The "loc\_admin\_access\_only" field, a boolean, signifies restricted access. Temperature data is stored in "loc\_room\_temperature." Timestamps "loc\_created\_at" and "loc\_lastupdated\_at" record creation and last update times. "loc\_created\_by" and "loc\_lastupdated\_by" link to user IDs. This table structure enables efficient location tracking with key information and temporal data. The SQL "CREATE TABLE IF NOT EXISTS" statement ensures table creation if it doesn't exist already.

# Create table location

cursor.execute('''

CREATE TABLE IF NOT EXISTS location (

    location\_id INTEGER PRIMARY KEY NOT NULL UNIQUE,

    loc\_building\_name TEXT,

    loc\_building\_add TEXT,

    loc\_floor\_number INTEGER,

    loc\_roomid INTEGER,

    loc\_admin\_access\_only BOOLEAN,

    loc\_room\_temperature INTEGER,

    loc\_created\_by INTEGER,

    loc\_created\_at TIMESTAMP,

    loc\_lastupdated\_by INTEGER,

    loc\_lastupdated\_at TIMESTAMP

)

''')

A screenshot of a computer

Description automatically generated

## Department table

The "department" table is structured with an integer primary key "department\_id" ensuring uniqueness. It contains essential department details, including "dep\_name" for the name, "dep\_description" for additional information, and "dep\_hq\_locationid" as a foreign key referencing the "location" table. Timestamps "d\_created\_at" and "d\_lastupdated\_at" record creation and last update times. User IDs "d\_created\_by" and "d\_lastupdated\_by" establish responsible users. This table enables the association of departments with specific locations through foreign key relationships. The SQL "CREATE TABLE IF NOT EXISTS" statement ensures the table is created if it doesn't exist, emphasizing data integrity and consistency.

# Create table department

cursor.execute('''

CREATE TABLE IF NOT EXISTS department (

    department\_id INTEGER PRIMARY KEY NOT NULL UNIQUE,

    dep\_name TEXT NOT NULL,

    dep\_description TEXT,

    dep\_hq\_locationid INTEGER,

    d\_created\_by INTEGER,

    d\_created\_at TIMESTAMP,

    d\_lastupdated\_by INTEGER,

    d\_lastupdated\_at TIMESTAMP,

    FOREIGN KEY (dep\_hq\_locationid) REFERENCES location (location\_id)

)

''')

A screenshot of a computer

Description automatically generated

## Employee table

The "employee" table is designed with an integer primary key "emp\_id" to ensure uniqueness. It captures vital employee information, such as "emp\_name" for the name, "emp\_sex" for gender, "emp\_role" for job title, and "emp\_status" indicating active or inactive status. Foreign keys "emp\_departmentid" and "emp\_managerid" reference the "department" and "employee" tables, establishing departmental and managerial relationships. Timestamps "e\_created\_at" and "e\_lastupdated\_at" record creation and last update times, while user IDs "e\_created\_by" and "e\_lastupdated\_by" identify responsible users. The SQL "CREATE TABLE IF NOT EXISTS" statement ensures table creation if absent, emphasizing data consistency and integrity.

# Create table employee

cursor.execute('''

CREATE TABLE IF NOT EXISTS employee (

    emp\_id INTEGER PRIMARY KEY NOT NULL UNIQUE,

    emp\_name TEXT NOT NULL,

    emp\_sex TEXT NOT NULL,

    emp\_role TEXT,

    emp\_status BOOLEAN NOT NULL,

    emp\_managerid INTEGER,

    emp\_email TEXT NOT NULL,

    emp\_mobile INTEGER NOT NULL,

    emp\_address TEXT,

    emp\_departmentid INTEGER,

    emp\_dob DATE NOT NULL,

    emp\_age INTEGER,

    emp\_age\_range TEXT NOT NULL,

    emp\_age\_category TEXT NOT NULL,

    emp\_hiringdate DATE NOT NULL,

    emp\_salary INTEGER,

    emp\_salary\_category TEXT,

    emp\_salary\_hike INTEGER,

    e\_created\_by INTEGER,

    e\_created\_at TIMESTAMP,

    e\_lastupdated\_by INTEGER,

    e\_lastupdated\_at TIMESTAMP,

    FOREIGN KEY (emp\_departmentid) REFERENCES department (department\_id),

    FOREIGN KEY (emp\_managerid) REFERENCES employee (emp\_id)

)

''')

A screenshot of a computer

Description automatically generated

## Login table

The "login" table is structured with an auto-incrementing primary key "log\_id" for unique identification. It records login and logout events with "emp\_id" referencing the "employee" table. "login\_locationid" and "logout\_locationid" are foreign keys tied to the "location" table, linking events to specific locations. Timestamps "login\_time" and "logout\_time" denote entry and exit times. User IDs "log\_created\_by" and "log\_lastupdated\_by" attribute data changes, while "log\_created\_at" and "log\_lastupdated\_at" capture timestamps. The "CREATE TABLE IF NOT EXISTS" SQL statement ensures table creation if not present, promoting data consistency and integrity in the database.

# Create table login

cursor.execute('''

CREATE TABLE IF NOT EXISTS login (

    log\_id INTEGER PRIMARY KEY AUTOINCREMENT,

    emp\_id INTEGER NOT NULL,

    login\_locationid INTEGER,

    login\_time TIMESTAMP,

    logout\_locationid INTEGER,

    logout\_time TIMESTAMP,

    log\_created\_by INTEGER,

    log\_created\_at TIMESTAMP,

    log\_lastupdated\_by INTEGER,

    log\_lastupdated\_at TIMESTAMP,

    FOREIGN KEY (emp\_id) REFERENCES employee (emp\_id),

    FOREIGN KEY (login\_locationid) REFERENCES location (location\_id),

    FOREIGN KEY (logout\_locationid) REFERENCES location (location\_id)

)

''')

A screenshot of a computer

Description automatically generated

# Generation of data

Using the python and its Faker library, synthetic data is generated for an SQLite database. This includes details like employee names, roles, email addresses, and timestamps. The generated data encompasses various tables such as location, department, employee, and login, ensuring diverse and realistic information for testing and development purposes.

## Data generation for location table

The code inserts 1000 synthetic records into the location table of an SQLite database using Faker. Randomized data, such as building names, addresses, and timestamps. The script utilizes a loop to generate unique location details, including building-related specifics and user creation details, promoting realistic and varied test data. The database is then committed to persist the changes.

# Add 1000 records to location table

for i in range(1000):

    location\_id = i + 10000

    loc\_building\_name = fake.company()

    loc\_building\_add = fake.address()

    loc\_floor\_number = random.randint(0, 20)

    loc\_roomid = random.randint(1, 99)

    loc\_admin\_access\_only = random.choice([True, False])

    loc\_room\_temperature = random.randint(15, 28)

    loc\_created\_by = random.randint(30000, 32000)

    loc\_lastupdated\_by = loc\_created\_by + 1

    loc\_created\_at = random\_timestamp()

    loc\_lastupdated\_at = loc\_created\_at + timedelta(days=random.randint(1, 30))

    cursor.execute('''

    INSERT INTO location (location\_id, loc\_building\_name, loc\_building\_add, loc\_floor\_number, loc\_roomid,

                          loc\_admin\_access\_only, loc\_room\_temperature, loc\_created\_by, loc\_created\_at,

                          loc\_lastupdated\_by, loc\_lastupdated\_at)

    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

    ''', (location\_id, loc\_building\_name, loc\_building\_add, loc\_floor\_number, loc\_roomid, loc\_admin\_access\_only,

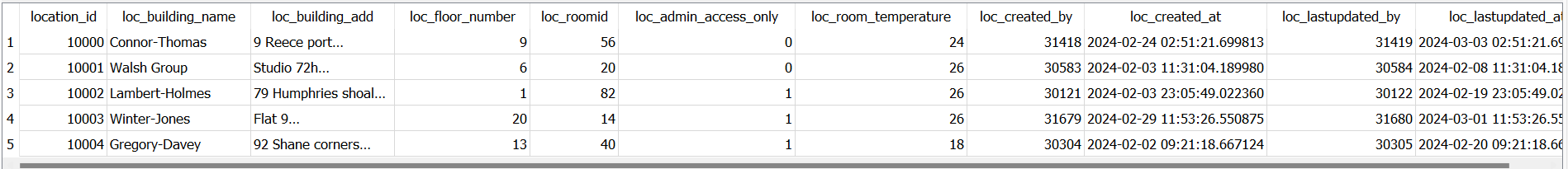
          loc\_room\_temperature, loc\_created\_by, loc\_created\_at, loc\_lastupdated\_by, loc\_lastupdated\_at))

# Commit the location data insertion

connection.commit()

### Sample data of location table

SELECT \* FROM location LIMIT 5



## Data generation for department table

The code defines a list of relevant IT department names. It then adds 25 records to the department table in an SQLite database, incorporating department-specific details, such as name, description, and location. Faker generates additional information for each record, and the data is committed to persist the changes in the database.

# Define relevant IT department names

it\_departments = [

    "IT Services", "Software Development", "Network Infrastructure", "Data Science", "Cybersecurity",

    "Database Management", "Quality Assurance", "Project Management", "Technical Support", "Cloud Computing",

    "Web Development", "Mobile App Development", "Systems Architecture", "Business Intelligence", "IT Operations",

    "Information Security", "User Experience (UX)", "IT Consulting", "Enterprise Solutions", "DevOps",

    "Artificial Intelligence", "Machine Learning", "Digital Transformation", "IT Governance", "IT Strategy"

]

# Add 25 records to the department table

for i in range(25):

    department\_id = i + 20000

    dep\_name = it\_departments[i]

    dep\_description = fake.sentence()

    dep\_hq\_locationid = random.randint(10000, 11000)

    d\_created\_by = random.randint(30000, 31000)

    d\_lastupdated\_by = d\_created\_by + 1

    d\_created\_at = random\_timestamp()

    d\_lastupdated\_at = d\_created\_at + timedelta(days=random.randint(1, 30))

    # Insert data into the department table

    cursor.execute('''

    INSERT INTO department (department\_id, dep\_name, dep\_description, dep\_hq\_locationid,

                           d\_created\_by, d\_created\_at, d\_lastupdated\_by, d\_lastupdated\_at)

    VALUES (?, ?, ?, ?, ?, ?, ?, ?)

    ''', (department\_id, dep\_name, dep\_description, dep\_hq\_locationid, d\_created\_by, d\_created\_at,

          d\_lastupdated\_by, d\_lastupdated\_at))

# Commit the department data insertion

connection.commit()

### Sample data of department table

SELECT \* FROM department LIMIT 5

A white background with black text

Description automatically generated with medium confidence

## Data generation for employee table

The code adds 2000 records to the employee table in an SQLite database. It generates employee details using Faker for attributes like name, gender, job role, and salary. Random timestamps are generated for fields like date of birth, hiring date, and creation/update timestamps. The data is inserted into the employee table, including foreign keys like emp\_departmentid and emp\_managerid, establishing relationships. Finally, the changes are committed to persist the employee data in the database.

# Add 2000 records to the employee table

for i in range(2000):

    # Generate employee details

    emp\_id = i + 30000

    emp\_name = fake.name()

    emp\_sex = random.choice(['Male', 'Female'])  # Add random gender (male/female)

    emp\_role = fake.job()

    emp\_status = random.choice([True, False])

    emp\_email = fake.email()

    emp\_mobile = fake.random\_int(100000000, 999999999)

    emp\_address = fake.address()

    emp\_departmentid = random.randint(20000, 20025)

    emp\_dob\_date = fake.date\_of\_birth(minimum\_age=18, maximum\_age=60)

    emp\_dob = datetime.combine(emp\_dob\_date, datetime.min.time())  # Convert to datetime object

    emp\_age = (datetime.now() - emp\_dob).days // 365

    emp\_age\_category = 'Young Adults' if 18 <= emp\_age <= 30 else ('Mid Adults' if 31 <= emp\_age <= 60 else 'Senior Adults')

    emp\_age\_range = '18 - 30' if 18 <= emp\_age <= 30 else ('31 - 60' if 31 <= emp\_age <= 60 else '60 above')

    emp\_hiringdate = random\_timestamp()

    emp\_salary = random.randint(30000, 150000)

    emp\_salary\_category = 'very\_low' if emp\_salary < 50000 else (

        'low' if emp\_salary < 80000 else ('mid' if emp\_salary < 120000 else ('high' if emp\_salary < 150000 else 'very\_high')))

    emp\_salary\_hike = random.randint(1, 20)

    e\_created\_by = random.randint(30000, 31000)

    e\_lastupdated\_by = e\_created\_by + 1

    e\_created\_at = random\_timestamp()

    e\_lastupdated\_at = e\_created\_at + timedelta(days=random.randint(1, 30))

    emp\_managerid = random.randint(30000, 31000)

    # Insert employee record into the database

    cursor.execute('''

    INSERT INTO employee (emp\_id, emp\_name, emp\_sex, emp\_role, emp\_status, emp\_managerid, emp\_email, emp\_mobile,

                          emp\_address, emp\_departmentid, emp\_dob, emp\_age, emp\_age\_range, emp\_age\_category,

                          emp\_hiringdate, emp\_salary, emp\_salary\_category, emp\_salary\_hike,

                          e\_created\_by, e\_created\_at, e\_lastupdated\_by, e\_lastupdated\_at)

    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

    ''', (emp\_id, emp\_name, emp\_sex, emp\_role, emp\_status, emp\_managerid, emp\_email, emp\_mobile, emp\_address,

          emp\_departmentid, emp\_dob\_date, emp\_age, emp\_age\_range, emp\_age\_category, emp\_hiringdate, emp\_salary,

          emp\_salary\_category, emp\_salary\_hike, e\_created\_by, e\_created\_at, e\_lastupdated\_by,

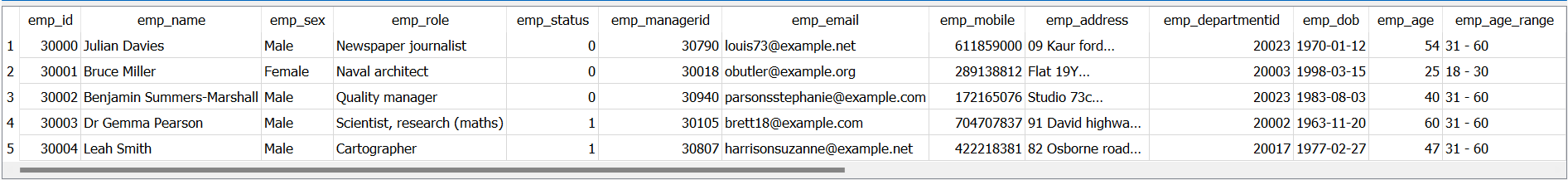
          e\_lastupdated\_at))

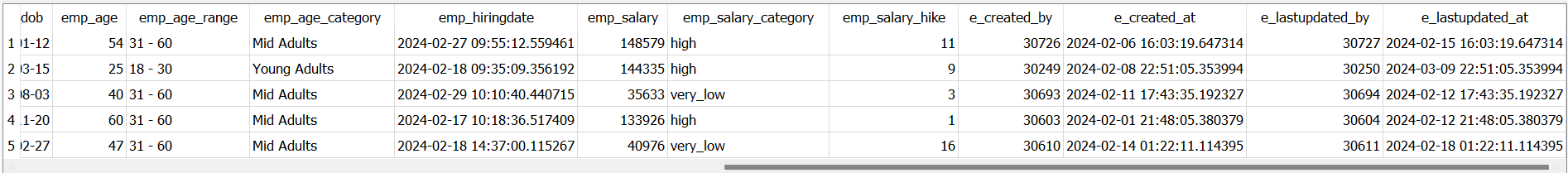
# Commit the employee data insertion

connection.commit()

### Sample data of employee table

SELECT \* FROM employee LIMIT 5





## Data generation for login table

The code inserts 5000 records into the login table of a SQLite database. It generates login details using random employee IDs, location IDs, and timestamps. The data includes fields like login\_time, logout\_time, and creation/update timestamps. The changes are committed to persist the login data in the database.

# Add 5000 records to login table

for log\_id in range(101, 5101):

    # Generate login details

    emp\_id = random.randint(30000, 32000)

    login\_locationid = random.randint(10000, 11000)

    login\_time = random\_timestamp()

    logout\_locationid = login\_locationid

    logout\_time = login\_time + timedelta(hours=random.randint(6, 15))

    log\_created\_by = random.randint(30000, 31000)

    log\_lastupdated\_by = log\_created\_by + 1

    log\_created\_at = random\_timestamp()

    log\_lastupdated\_at = log\_created\_at + timedelta(days=random.randint(1, 30))

    # Insert login record into the database

    cursor.execute('''

    INSERT INTO login (log\_id, emp\_id, login\_locationid, login\_time,

                       logout\_locationid, logout\_time, log\_created\_by, log\_created\_at,

                       log\_lastupdated\_by, log\_lastupdated\_at)

    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

    ''', (log\_id, emp\_id, login\_locationid, login\_time, logout\_locationid,

          logout\_time, log\_created\_by, log\_created\_at, log\_lastupdated\_by, log\_lastupdated\_at))

# Commit the login data insertion

connection.commit()

### Sample data of login table

SELECT \* FROM login LIMIT 5

A screenshot of a computer

Description automatically generated

# Demonstration

Example queries of your database including joins and selections, demonstrating different data types.

## joins and selections

Login details with employee and department information for those in the 'Software Development' department, joining “login”, “employee” and “department” tables.

A screenshot of a computer

Description automatically generated

The total number of logins for each employee

A screenshot of a computer

Description automatically generated

List employees hired in the last month along with their departments.

A screenshot of a computer

Description automatically generated

The average salary hike for employees in each age category and sex

A screenshot of a computer

Description automatically generated

## Different datatypes

SELECT emp\_id, emp\_status, emp\_name, emp\_role, emp\_mobile, emp\_dob, emp\_age, emp\_age\_range, emp\_age\_category, emp\_salary, emp\_salary\_category, e\_lastupdated\_at FROM employee LIMIT 10

A screenshot of a computer

Description automatically generated

SELECT \* FROM location LIMIT 10

A screenshot of a computer

Description automatically generated