[1]

Data Mining and Discovery

Master of Science in Data Science

SHUBHAM VERMA

22099668

Sv23abk@herts.ac.uk

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

www.herts.ac.uk

Contents

1. 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.

1.1 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.

1.1.1 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()
```

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

Open SQLite DB Browser.

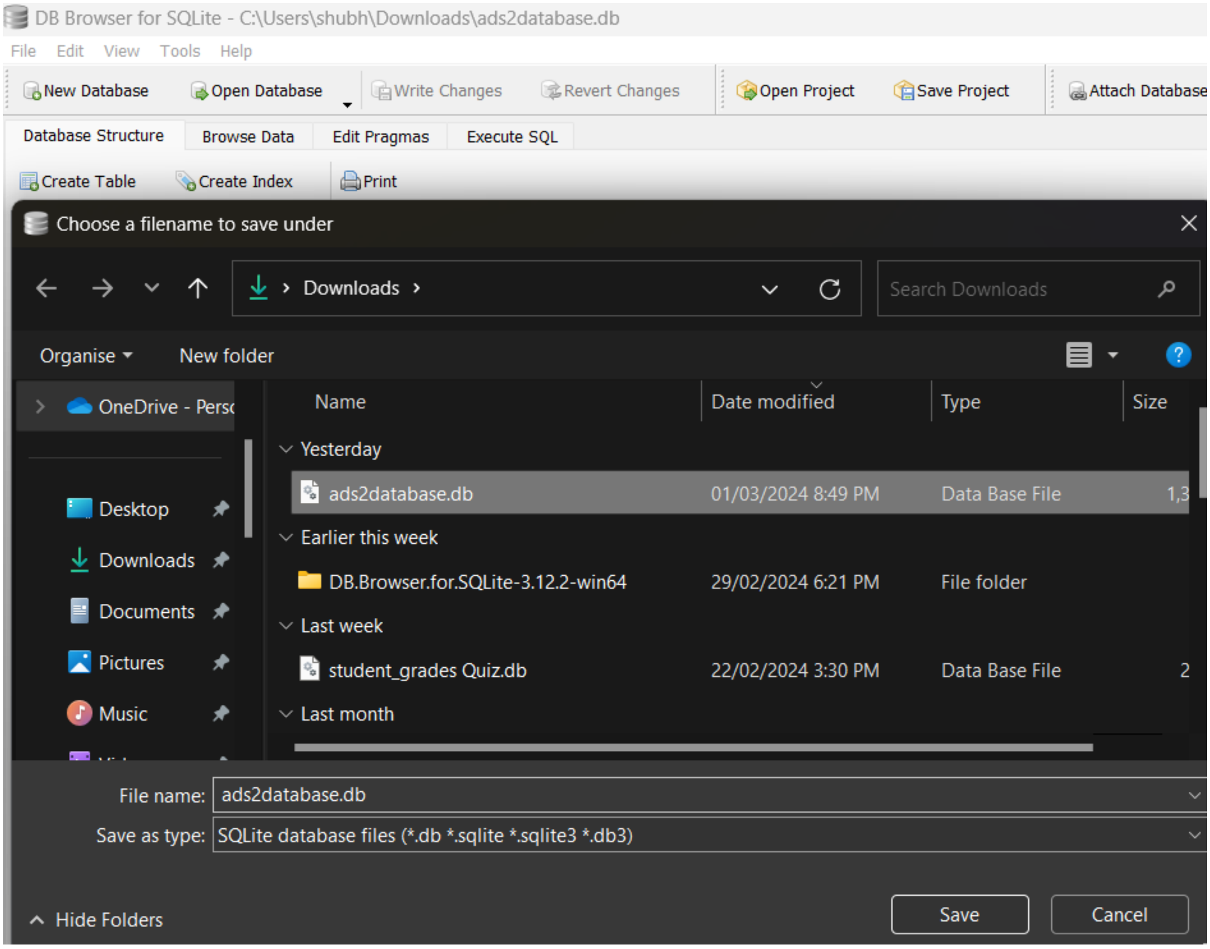
Go to "File" in the menu.

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.



DB Browser for SQLite - C:\Users\shubh\Downloads\ads2database.db

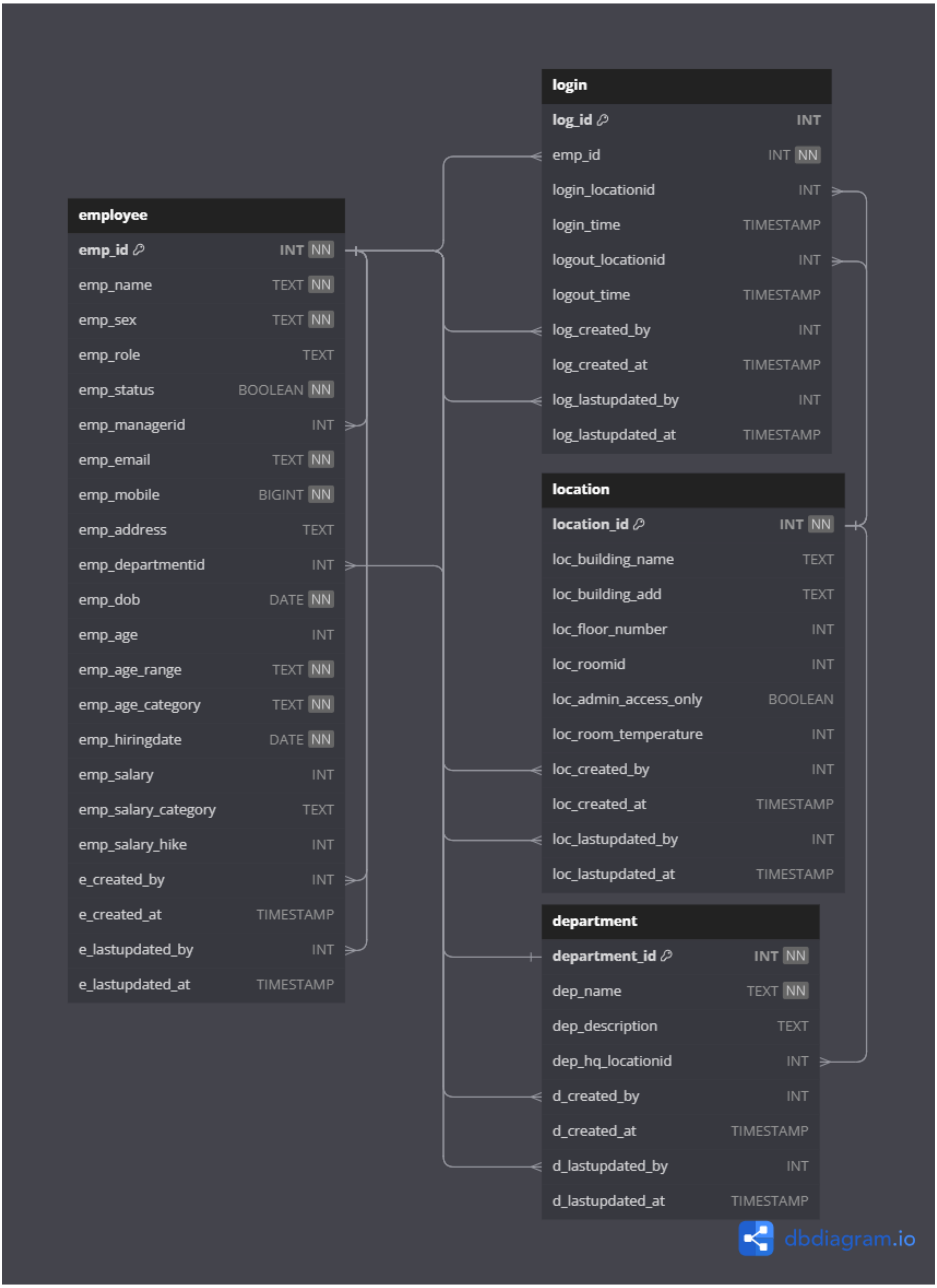
Choose a filename to save under

Name	Date modified	Type	Size
Yesterday			
ads2database.db	01/03/2024 8:49 PM	Data Base File	1,3
Earlier this week			
DB.Browser.for.SQLite-3.12.2-win64	29/02/2024 6:21 PM	File folder	
Last week			
student_grades Quiz.db	22/02/2024 3:30 PM	Data Base File	2
Last month			

File name: ads2database.db

Save as type: SQLite database files (*.db *.sqlite *.sqlite3 *.db3)

1.2 Database diagram



employee		
emp_id 🔑	INT	NN
emp_name	TEXT	NN
emp_sex	TEXT	NN
emp_role	TEXT	
emp_status	BOOLEAN	NN
emp_managerid	INT	
emp_email	TEXT	NN
emp_mobile	BIGINT	NN
emp_address	TEXT	
emp_departmentid	INT	
emp_dob	DATE	NN
emp_age	INT	
emp_age_range	TEXT	NN
emp_age_category	TEXT	NN
emp_hiringdate	DATE	NN
emp_salary	INT	
emp_salary_category	TEXT	
emp_salary_hike	INT	
e_created_by	INT	
e_created_at	TIMESTAMP	
e_lastupdated_by	INT	
e_lastupdated_at	TIMESTAMP	

login		
log_id 🔑	INT	
emp_id	INT	NN
login_locationid	INT	
login_time	TIMESTAMP	
logout_locationid	INT	
logout_time	TIMESTAMP	
log_created_by	INT	
log_created_at	TIMESTAMP	
log_lastupdated_by	INT	
log_lastupdated_at	TIMESTAMP	

location		
location_id 🔑	INT	NN
loc_building_name	TEXT	
loc_building_add	TEXT	
loc_floor_number	INT	
loc_roomid	INT	
loc_admin_access_only	BOOLEAN	
loc_room_temperature	INT	
loc_created_by	INT	
loc_created_at	TIMESTAMP	
loc_lastupdated_by	INT	
loc_lastupdated_at	TIMESTAMP	

department		
department_id 🔑	INT	NN
dep_name	TEXT	NN
dep_description	TEXT	
dep_hq_locationid	INT	
d_created_by	INT	
d_created_at	TIMESTAMP	
d_lastupdated_by	INT	
d_lastupdated_at	TIMESTAMP	

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.

1.3 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.

1.3.1 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.

1.3.2 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.

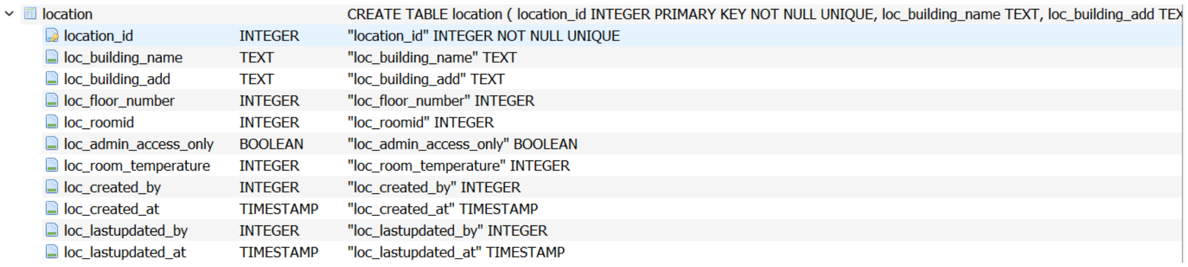
2. Tables

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.

2.1 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
)
''')
```

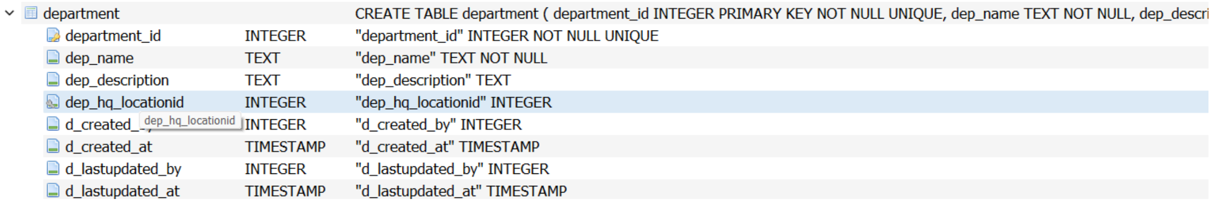


location		CREATE TABLE location (location_id INTEGER PRIMARY KEY NOT NULL UNIQUE, loc_building_name TEXT, loc_building_add TEX
location_id	INTEGER	"location_id" INTEGER NOT NULL UNIQUE
loc_building_name	TEXT	"loc_building_name" TEXT
loc_building_add	TEXT	"loc_building_add" TEXT
loc_floor_number	INTEGER	"loc_floor_number" INTEGER
loc_roomid	INTEGER	"loc_roomid" INTEGER
loc_admin_access_only	BOOLEAN	"loc_admin_access_only" BOOLEAN
loc_room_temperature	INTEGER	"loc_room_temperature" INTEGER
loc_created_by	INTEGER	"loc_created_by" INTEGER
loc_created_at	TIMESTAMP	"loc_created_at" TIMESTAMP
loc_lastupdated_by	INTEGER	"loc_lastupdated_by" INTEGER
loc_lastupdated_at	TIMESTAMP	"loc_lastupdated_at" TIMESTAMP

2.2 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)
)
''')
```

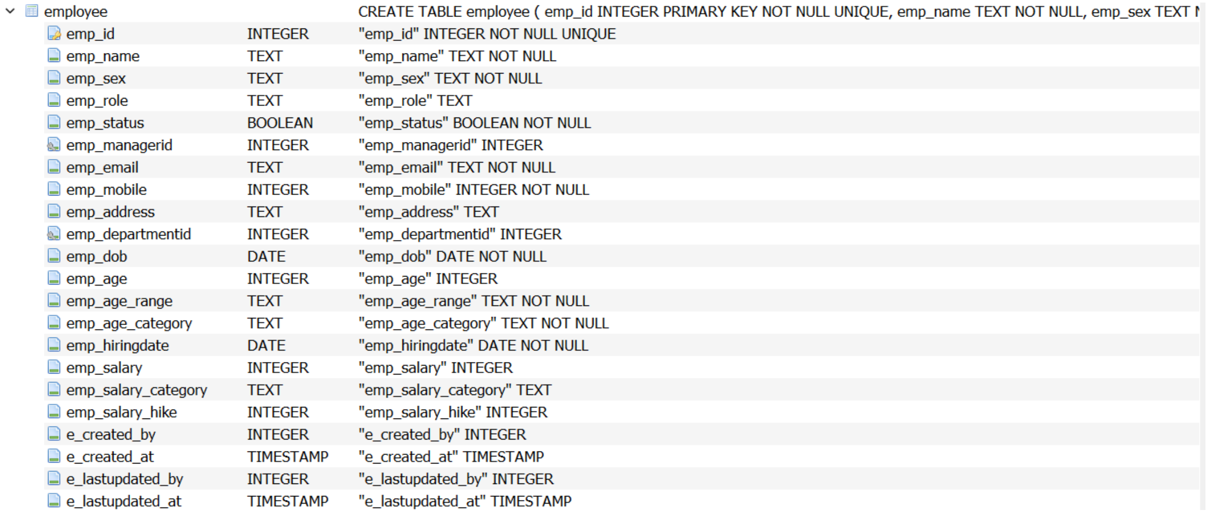


department		CREATE TABLE department (department_id INTEGER PRIMARY KEY NOT NULL UNIQUE, dep_name TEXT NOT NULL, dep_descri
department_id	INTEGER	"department_id" INTEGER NOT NULL UNIQUE
dep_name	TEXT	"dep_name" TEXT NOT NULL
dep_description	TEXT	"dep_description" TEXT
dep_hq_locationid	INTEGER	"dep_hq_locationid" INTEGER
d_created_ dep_hq_locationid	INTEGER	"d_created_by" INTEGER
d_created_at	TIMESTAMP	"d_created_at" TIMESTAMP
d_lastupdated_by	INTEGER	"d_lastupdated_by" INTEGER
d_lastupdated_at	TIMESTAMP	"d_lastupdated_at" TIMESTAMP

2.3 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)
)
''')
```

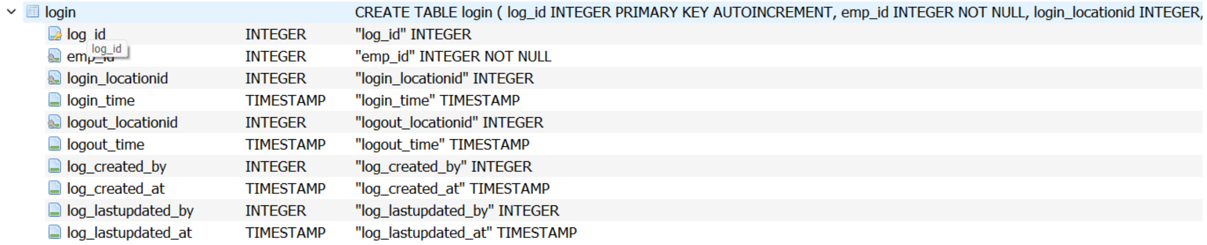


employee		CREATE TABLE employee (emp_id INTEGER PRIMARY KEY NOT NULL UNIQUE, emp_name TEXT NOT NULL, emp_sex TEXT N
emp_id	INTEGER	"emp_id" INTEGER NOT NULL UNIQUE
emp_name	TEXT	"emp_name" TEXT NOT NULL
emp_sex	TEXT	"emp_sex" TEXT NOT NULL
emp_role	TEXT	"emp_role" TEXT
emp_status	BOOLEAN	"emp_status" BOOLEAN NOT NULL
emp_managerid	INTEGER	"emp_managerid" INTEGER
emp_email	TEXT	"emp_email" TEXT NOT NULL
emp_mobile	INTEGER	"emp_mobile" INTEGER NOT NULL
emp_address	TEXT	"emp_address" TEXT
emp_departmentid	INTEGER	"emp_departmentid" INTEGER
emp_dob	DATE	"emp_dob" DATE NOT NULL
emp_age	INTEGER	"emp_age" INTEGER
emp_age_range	TEXT	"emp_age_range" TEXT NOT NULL
emp_age_category	TEXT	"emp_age_category" TEXT NOT NULL
emp_hiringdate	DATE	"emp_hiringdate" DATE NOT NULL
emp_salary	INTEGER	"emp_salary" INTEGER
emp_salary_category	TEXT	"emp_salary_category" TEXT
emp_salary_hike	INTEGER	"emp_salary_hike" INTEGER
e_created_by	INTEGER	"e_created_by" INTEGER
e_created_at	TIMESTAMP	"e_created_at" TIMESTAMP
e_lastupdated_by	INTEGER	"e_lastupdated_by" INTEGER
e_lastupdated_at	TIMESTAMP	"e_lastupdated_at" TIMESTAMP

2.4 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)
)
''')
```



login		CREATE TABLE login (log_id INTEGER PRIMARY KEY AUTOINCREMENT, emp_id INTEGER NOT NULL, login_locationid INTEGER,
log_id	INTEGER	"log_id" INTEGER
emp_id	INTEGER	"emp_id" INTEGER NOT NULL
login_locationid	INTEGER	"login_locationid" INTEGER
login_time	TIMESTAMP	"login_time" TIMESTAMP
logout_locationid	INTEGER	"logout_locationid" INTEGER
logout_time	TIMESTAMP	"logout_time" TIMESTAMP
log_created_by	INTEGER	"log_created_by" INTEGER
log_created_at	TIMESTAMP	"log_created_at" TIMESTAMP
log_lastupdated_by	INTEGER	"log_lastupdated_by" INTEGER
log_lastupdated_at	TIMESTAMP	"log_lastupdated_at" TIMESTAMP

3. 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.

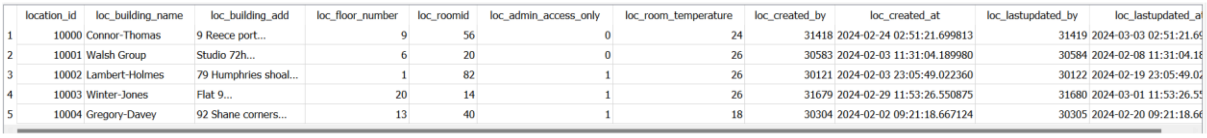
3.1 Data generation for location table

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()
```

3.1.1 Sample data of location table

SELECT * FROM location LIMIT 5



	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_a
1	10000	Connor-Thomas	9 Reece port...	9	56	0	24	31418	2024-02-24 02:51:21.699813	31419	2024-03-03 02:51:21.69
2	10001	Walsh Group	Studio 72h...	6	20	0	26	30583	2024-02-03 11:31:04.189980	30584	2024-02-08 11:31:04.18
3	10002	Lambert-Holmes	79 Humphries shoal...	1	82	1	26	30121	2024-02-03 23:05:49.022360	30122	2024-02-19 23:05:49.02
4	10003	Winter-Jones	Flat 9...	20	14	1	26	31679	2024-02-29 11:53:26.550875	31680	2024-03-01 11:53:26.55
5	10004	Gregory-Davey	92 Shane corners...	13	40	1	18	30304	2024-02-02 09:21:18.667124	30305	2024-02-20 09:21:18.66

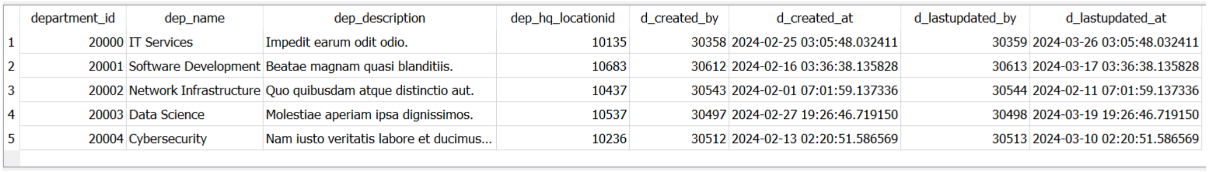
3.2 Data generation for department table

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()
```

3.2.1 Sample data of department table

SELECT * FROM department LIMIT 5



	department_id	dep_name	dep_description	dep_hq_locationid	d_created_by	d_created_at	d_lastupdated_by	d_lastupdated_at
1	20000	IT Services	Impedit earum odit odio.	10135	30358	2024-02-25 03:05:48.032411	30359	2024-03-26 03:05:48.032411
2	20001	Software Development	Beatae magnam quasi blanditiis.	10683	30612	2024-02-16 03:36:38.135828	30613	2024-03-17 03:36:38.135828
3	20002	Network Infrastructure	Quo quibusdam atque distinctio aut.	10437	30543	2024-02-01 07:01:59.137336	30544	2024-02-11 07:01:59.137336
4	20003	Data Science	Molestiae aperiam ipsa dignissimos.	10537	30497	2024-02-27 19:26:46.719150	30498	2024-03-19 19:26:46.719150
5	20004	Cybersecurity	Nam iusto veritatis labore et ducimus...	10236	30512	2024-02-13 02:20:51.586569	30513	2024-03-10 02:20:51.586569

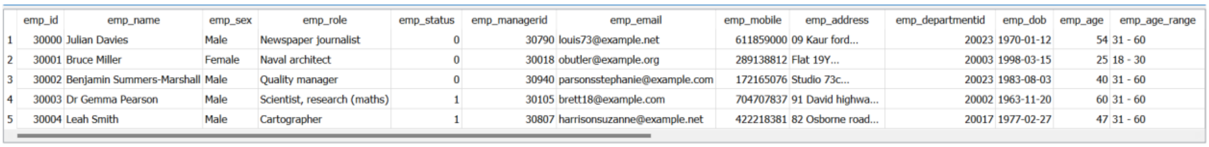
3.3 *Data generation for employee table*

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.

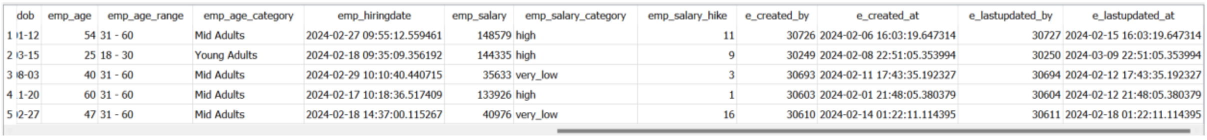
```
# Add 2000 records to the employee table
for i in range(2000):
    # Generate employee details
    emp_id = i + 30000
    emp_name = fake.name()
    emp_sex = random.choice(['Male', 'Female'])  # Add random gender
(male/female)
    emp_role = fake.job()
    emp_status = random.choice([True, False])
    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()
```

3.3.1 Sample data of employee table

SELECT * FROM employee LIMIT 5



	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
1	30000	Julian Davies	Male	Newspaper journalist	0	30790	louis73@example.net	611859000	09 Kaur ford...	20023	1970-01-12	54	31 - 60
2	30001	Bruce Miller	Female	Naval architect	0	30018	obutler@example.org	289138812	Flat 19Y...	20003	1998-03-15	25	18 - 30
3	30002	Benjamin Summers-Marshall	Male	Quality manager	0	30940	parsonsstephanie@example.com	172165076	Studio 73c...	20023	1983-08-03	40	31 - 60
4	30003	Dr Gemma Pearson	Male	Scientist, research (maths)	1	30105	brett18@example.com	704707837	91 David highwa...	20002	1963-11-20	60	31 - 60
5	30004	Leah Smith	Male	Cartographer	1	30807	harrisonsuzanne@example.net	422218381	82 Osborne road...	20017	1977-02-27	47	31 - 60



	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
1	01-12	54	31 - 60	Mid Adults	2024-02-27 09:55:12.559461	148579	high	11	30726	2024-02-06 16:03:19.647314	30727	2024-02-15 16:03:19.647314
2	03-15	25	18 - 30	Young Adults	2024-02-18 09:35:09.356192	144335	high	9	30249	2024-02-08 22:51:05.353994	30250	2024-03-09 22:51:05.353994
3	08-03	40	31 - 60	Mid Adults	2024-02-29 10:10:40.440715	35633	very_low	3	30693	2024-02-11 17:43:35.192327	30694	2024-02-12 17:43:35.192327
4	1-20	60	31 - 60	Mid Adults	2024-02-17 10:18:36.517409	133926	high	1	30603	2024-02-01 21:48:05.380379	30604	2024-02-12 21:48:05.380379
5	02-27	47	31 - 60	Mid Adults	2024-02-18 14:37:00.115267	40976	very_low	16	30610	2024-02-14 01:22:11.114395	30611	2024-02-18 01:22:11.114395

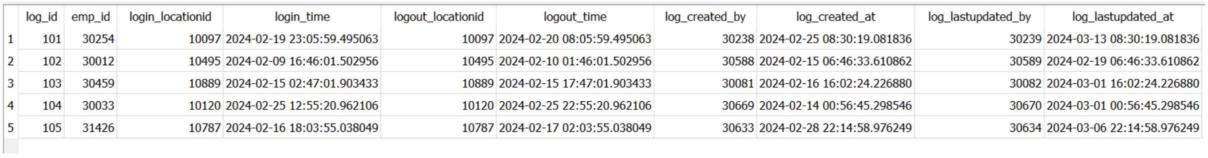
3.4 Data generation for login table

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()
```

3.4.1 Sample data of login table

SELECT * FROM login LIMIT 5



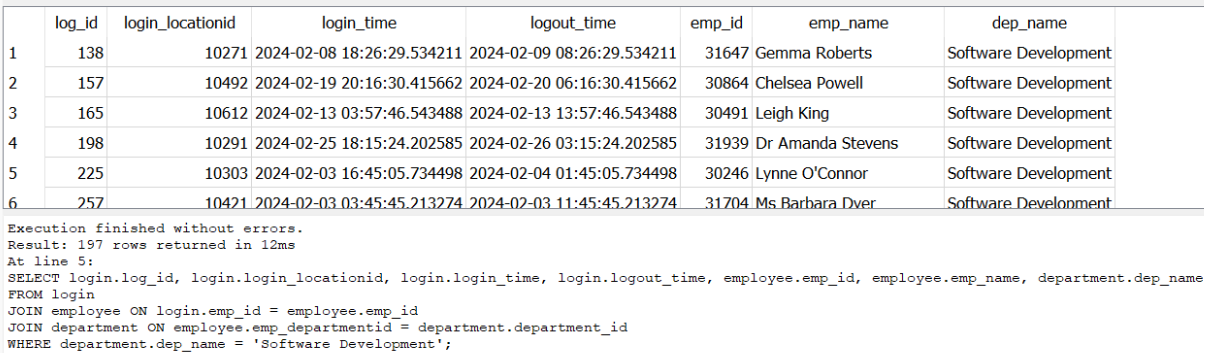
	log_id	emp_id	login_locationid	login_time	logout_locationid	logout_time	log_created_by	log_created_at	log_lastupdated_by	log_lastupdated_at
1	101	30254	10097	2024-02-19 23:05:59.495063	10097	2024-02-20 08:05:59.495063	30238	2024-02-25 08:30:19.081836	30239	2024-03-13 08:30:19.081836
2	102	30012	10495	2024-02-09 16:46:01.502956	10495	2024-02-10 01:46:01.502956	30588	2024-02-15 06:46:33.610862	30589	2024-02-19 06:46:33.610862
3	103	30459	10889	2024-02-15 02:47:01.903433	10889	2024-02-15 17:47:01.903433	30081	2024-02-16 16:02:24.226880	30082	2024-03-01 16:02:24.226880
4	104	30033	10120	2024-02-25 12:55:20.962106	10120	2024-02-25 22:55:20.962106	30669	2024-02-14 00:56:45.298546	30670	2024-03-01 00:56:45.298546
5	105	31426	10787	2024-02-16 18:03:55.038049	10787	2024-02-17 02:03:55.038049	30633	2024-02-28 22:14:58.976249	30634	2024-03-06 22:14:58.976249

4. Demonstration

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

4.1 joins and selections

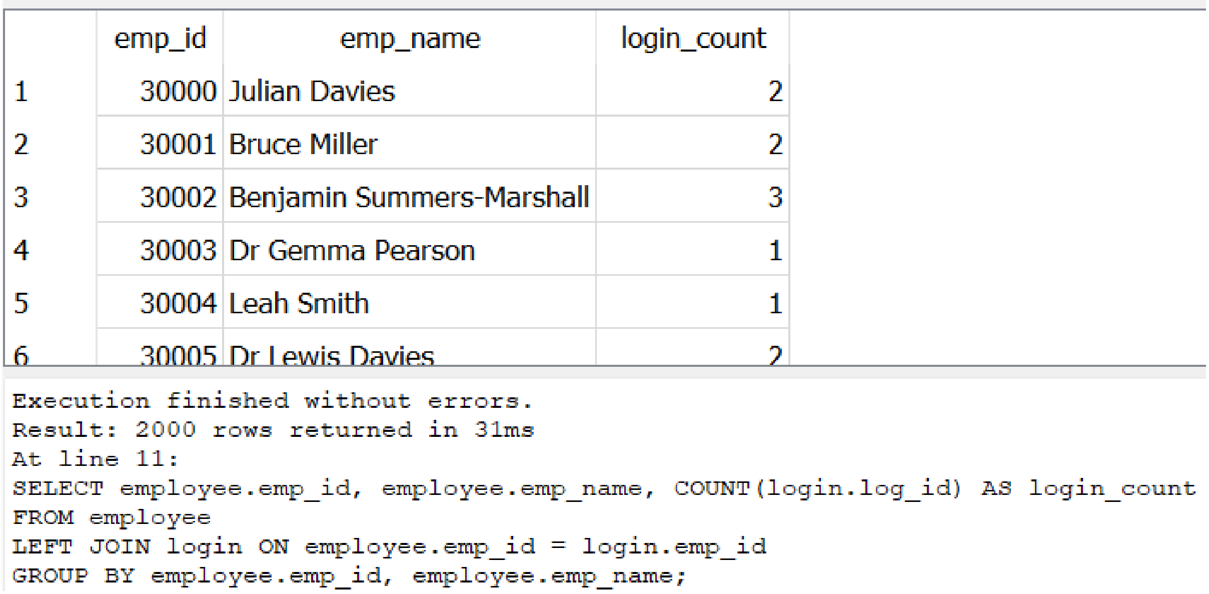
Login details with employee and department information for those in the 'Software Development' department, joining “login”, “employee” and “department” tables.



	log_id	login_locationid	login_time	logout_time	emp_id	emp_name	dep_name
1	138	10271	2024-02-08 18:26:29.534211	2024-02-09 08:26:29.534211	31647	Gemma Roberts	Software Development
2	157	10492	2024-02-19 20:16:30.415662	2024-02-20 06:16:30.415662	30864	Chelsea Powell	Software Development
3	165	10612	2024-02-13 03:57:46.543488	2024-02-13 13:57:46.543488	30491	Leigh King	Software Development
4	198	10291	2024-02-25 18:15:24.202585	2024-02-26 03:15:24.202585	31939	Dr Amanda Stevens	Software Development
5	225	10303	2024-02-03 16:45:05.734498	2024-02-04 01:45:05.734498	30246	Lynne O'Connor	Software Development
6	257	10421	2024-02-03 03:45:45.213274	2024-02-03 11:45:45.213274	31704	Ms Barbara Dyer	Software Development

```
Execution finished without errors.
Result: 197 rows returned in 12ms
At line 5:
SELECT login.log_id, login.login_locationid, login.login_time, login.logout_time, employee.emp_id, employee.emp_name, department.dep_name
FROM login
JOIN employee ON login.emp_id = employee.emp_id
JOIN department ON employee.emp_departmentid = department.department_id
WHERE department.dep_name = 'Software Development';
```

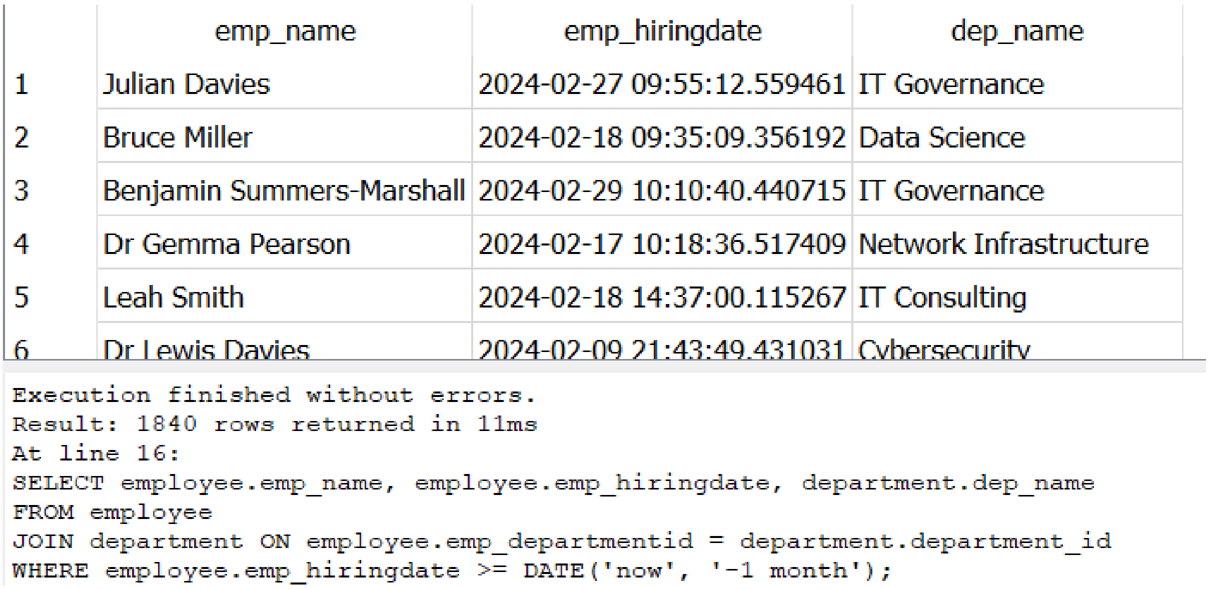
The total number of logins for each employee



	emp_id	emp_name	login_count
1	30000	Julian Davies	2
2	30001	Bruce Miller	2
3	30002	Benjamin Summers-Marshall	3
4	30003	Dr Gemma Pearson	1
5	30004	Leah Smith	1
6	30005	Dr Lewis Davies	2

```
Execution finished without errors.
Result: 2000 rows returned in 31ms
At line 11:
SELECT employee.emp_id, employee.emp_name, COUNT(login.log_id) AS login_count
FROM employee
LEFT JOIN login ON employee.emp_id = login.emp_id
GROUP BY employee.emp_id, employee.emp_name;
```

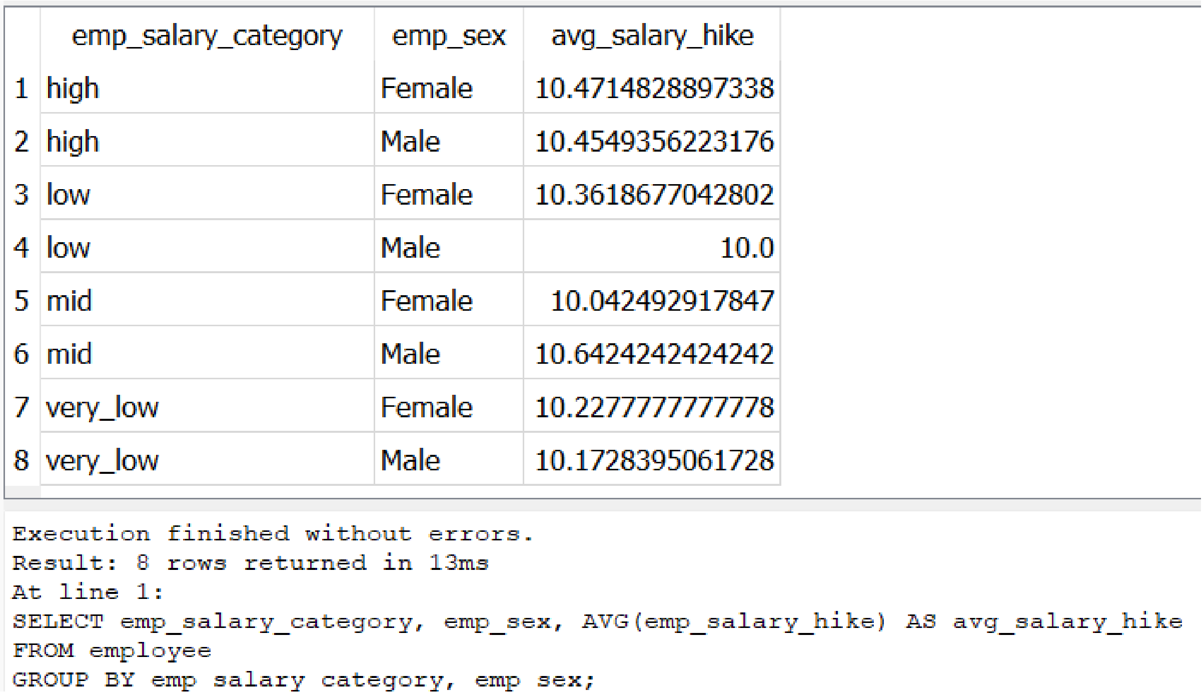
List employees hired in the last month along with their departments.



	emp_name	emp_hiringdate	dep_name
1	Julian Davies	2024-02-27 09:55:12.559461	IT Governance
2	Bruce Miller	2024-02-18 09:35:09.356192	Data Science
3	Benjamin Summers-Marshall	2024-02-29 10:10:40.440715	IT Governance
4	Dr Gemma Pearson	2024-02-17 10:18:36.517409	Network Infrastructure
5	Leah Smith	2024-02-18 14:37:00.115267	IT Consulting
6	Dr Lewis Davies	2024-02-09 21:43:49.431031	Cybersecurity

```
Execution finished without errors.
Result: 1840 rows returned in 11ms
At line 16:
SELECT employee.emp_name, employee.emp_hiringdate, department.dep_name
FROM employee
JOIN department ON employee.emp_departmentid = department.department_id
WHERE employee.emp_hiringdate >= DATE('now', '-1 month');
```

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

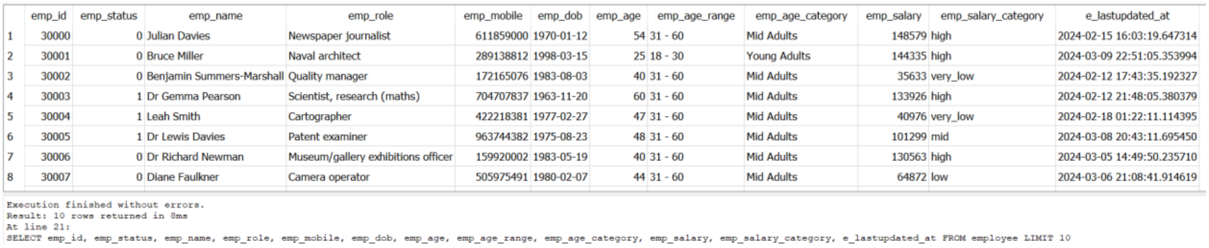


	emp_salary_category	emp_sex	avg_salary_hike
1	high	Female	10.4714828897338
2	high	Male	10.4549356223176
3	low	Female	10.3618677042802
4	low	Male	10.0
5	mid	Female	10.042492917847
6	mid	Male	10.6424242424242
7	very_low	Female	10.2277777777778
8	very_low	Male	10.1728395061728

```
Execution finished without errors.
Result: 8 rows returned in 13ms
At line 1:
SELECT emp_salary_category, emp_sex, AVG(emp_salary_hike) AS avg_salary_hike
FROM employee
GROUP BY emp salary category, emp sex;
```

4.2 Different datatypes

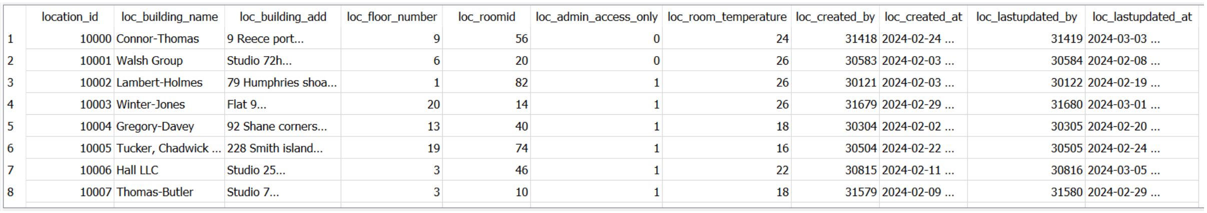
SELECT emp_id, emp_status, emp_name, emp_role, emp_mobile, emp_dob, emp_age, emp_age_range, emp_age_category, emp_salary, emp_salary_category, e_lastupdated_at FROM employee LIMIT 10



	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
1	30000	0	Julian Davies	Newspaper journalist	611859000	1970-01-12	54	31 - 60	Mid Adults	148579	high	2024-02-15 16:03:19.647314
2	30001	0	Bruce Miller	Naval architect	289138812	1998-03-15	25	18 - 30	Young Adults	144335	high	2024-03-09 22:51:05.353994
3	30002	0	Benjamin Summers-Marshall	Quality manager	172165076	1983-08-03	40	31 - 60	Mid Adults	35633	very_low	2024-02-12 17:43:35.192327
4	30003	1	Dr Gemma Pearson	Scientist, research (maths)	704707837	1963-11-20	60	31 - 60	Mid Adults	133926	high	2024-02-12 21:48:05.380379
5	30004	1	Leah Smith	Cartographer	422218381	1977-02-27	47	31 - 60	Mid Adults	40976	very_low	2024-02-18 01:22:11.114395
6	30005	1	Dr Lewis Davies	Patent examiner	963744382	1975-08-23	48	31 - 60	Mid Adults	101299	mid	2024-03-08 20:43:11.695450
7	30006	0	Dr Richard Newman	Museum/gallery exhibitions officer	159920002	1983-05-19	40	31 - 60	Mid Adults	130563	high	2024-03-05 14:49:50.235710
8	30007	0	Diane Faulkner	Camera operator	505975491	1980-02-07	44	31 - 60	Mid Adults	64872	low	2024-03-06 21:08:41.914619

```
Execution finished without errors.
Result: 10 rows returned in 8ms
At line 21:
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
```

SELECT * FROM location LIMIT 10



	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
1	10000	Connor-Thomas	9 Reece port...	9	56	0	24	31418	2024-02-24 ...	31419	2024-03-03 ...
2	10001	Walsh Group	Studio 72h...	6	20	0	26	30583	2024-02-03 ...	30584	2024-02-08 ...
3	10002	Lambert-Holmes	79 Humphries shoa...	1	82	1	26	30121	2024-02-03 ...	30122	2024-02-19 ...
4	10003	Winter-Jones	Flat 9...	20	14	1	26	31679	2024-02-29 ...	31680	2024-03-01 ...
5	10004	Gregory-Davey	92 Shane corners...	13	40	1	18	30304	2024-02-02 ...	30305	2024-02-20 ...
6	10005	Tucker, Chadwick ...	228 Smith island...	19	74	1	16	30504	2024-02-22 ...	30505	2024-02-24 ...
7	10006	Hall LLC	Studio 25...	3	46	1	22	30815	2024-02-11 ...	30816	2024-03-05 ...
8	10007	Thomas-Butler	Studio 7...	3	10	1	18	31579	2024-02-09 ...	31580	2024-02-29 ...

5. Github links

- Source code python notebook
- SQL assignment report
- Database file