



MAPÚA UNIVERSITY

SCHOOL OF ELECTRICAL, ELECTRONICS, AND COMPUTER ENGINEERING

Experiment 5:

Data Modelling and Database Systems

CPE106L (Software Design Laboratory)

Brian Matthew E. Clemente

Maria Angelica Acantilado

Chalzea Fransen C. Dytianquin

Vance David G. Samia

Group No.: **4**

Section: **FOPI01**



Readings, Insights, and Reflection

METIS #1:

A Guide to SQL. Philip J. Pratt; et al.9780357419830 . VBIID: 9780357419830

Chapter 1.1. What is a Database . 1.2. Database Requirements of TAL Distributors, Colonial Adventure Tours, and Solmaris Condominium Group

The confines of the first chapter of *A Guide to SQL* by Philip J. Pratt provides a foundational understanding of databases, emphasizing their essential role in data organization, management, and retrieval. Chapter 1.1 defines a database as a structured system for storing and managing data efficiently, distinguishing it from traditional file-based storage methods. The chapter highlights key advantages of databases, including enhanced data integrity, security, and accessibility. It introduces database components, such as tables, relationships, and queries, which collectively enable efficient data storage and retrieval. As a group, we recognize the significance of databases in various professional and organizational contexts. The discussion has reinforced our understanding of how structured data management is integral to ensuring accuracy, consistency, and reliability in decision-making.

Chapter 1.2 builds upon this introduction by examining the database requirements of three organizations that utilizes databases for distinct operational functions, including inventory management, customer tracking, and reservation systems. Through such case studies, we have gained a deeper understanding of how databases are designed to meet specific business needs, thereby improving efficiency and facilitating data-driven decision-making. This chapter has underscored the practical applications of database systems beyond theoretical concepts, demonstrating their role in optimizing business operations. Hence, as a group, we acknowledge the importance of approaching database learning with a problem-solving perspective, recognizing that proficiency in SQL extends beyond writing queries to understanding how databases can be strategically implemented to address real-world challenges. This insight reinforces the necessity of mastering database principles to develop effective and scalable data management solutions.

Chapter 2.1. Database Concepts . 2.2. Database Design Fundamentals . 2.3. Normalization

Chapter 2 of *A Guide to SQL* by Philip J. Pratt and others provides a comprehensive discussion of essential database concepts, focusing on foundational principles that ensure efficient data management. Chapter 2.1 introduces key database concepts, including data models, relationships, and constraints that uphold data integrity. It emphasizes the significance of relational databases, which structure data into tables with well-defined relationships to minimize redundancy and enhance consistency. This section also discusses the importance of

understanding data types, primary keys, and foreign keys in maintaining a well-organized database system. As a group, we recognize that these fundamental concepts form the basis of effective database development. The discussions have strengthened our understanding of how a well-designed database serves as a critical component of modern data-driven applications, ensuring seamless data access, storage, and security.

Likewise, Chapter 2.2 expands on these ideas by exploring database design fundamentals, underscoring the importance of planning before implementation. It highlights key aspects such as defining tables, determining relationships, and applying constraints to maintain data accuracy. The chapter also introduces entity-relationship diagrams (ERDs) as a visual representation of database structures, assisting in logical design before physical implementation. Additionally, Chapter 2.3 focuses on normalization, a structured approach to refining database design by eliminating redundancy and ensuring data integrity. The chapter details normalization forms, each progressively reducing data anomalies and improving efficiency. Through such underscored notions, the group's takeaways have reinforced the idea that database design is not just about organizing data but also about ensuring long-term efficiency, maintainability, and adaptability.

METIS #2:

Core Python Programming. R. Nageswara Rao. VBIID: 9789351198918

Chapter 24.1. Types of Databases Used with Python . 24.2. Using MySQL from Python

Chapter 24 of *Core Python Programming* by R. Nageswara Rao explores the integration of databases with Python, focusing on the different types of databases used and how Python interacts with MySQL. Chapter 24.1 provides an overview of various database types compatible with Python, including relational databases like MySQL, PostgreSQL, and SQLite, as well as NoSQL databases such as MongoDB and Redis. The discussion highlights the advantages and use cases of each database type, emphasizing how relational databases ensure structured data management through tables and relationships while NoSQL databases offer flexibility for handling unstructured or semi-structured data. As a group, we recognize the importance of selecting the appropriate database type based on application requirements. Understanding the strengths and limitations of each option enables us to make informed decisions when developing Python-based applications that involve data storage and retrieval. This chapter has reinforced our appreciation for databases as essential components of software development, ensuring efficient and scalable data handling.

Chapter 24.2 delves into the practical implementation of MySQL with Python, demonstrating how Python interacts with relational databases using the MySQL Connector module. It covers essential operations such as establishing a database connection, executing SQL queries, retrieving data, and handling transactions. The chapter also emphasizes best practices for secure database interaction, including the use of parameterized queries to prevent SQL injection. Through this discussion, we have gained valuable insights into how Python serves as a powerful tool for database management, enabling seamless communication between

applications and data storage systems. As a group, we acknowledge the significance of learning database connectivity in Python, as it is a fundamental skill for developing data-driven applications. This knowledge equips us with the ability to design efficient, secure, and well-structured systems that effectively manage and process data, reinforcing the importance of mastering database integration in Python programming.

METIS #3:

Python Projects. Laura Cassell. VBID: 9781118908891

Chapter 3.1. Relational Database Concepts . 3.2. Structured Query Language. DML and DDL SQL commands

Chapter 3 of *Python Projects* by Laura Cassell provides an in-depth exploration of relational database concepts and the structured query language, focusing on commands used to manipulate and define data. Chapter 3.1 introduces relational databases, emphasizing how data is organized into structured tables with clear relationships between them. It discusses essential principles such as unique identifiers for records, relationships between tables, organization of data to minimize redundancy, and rules that ensure accuracy and consistency. The chapter also highlights the importance of relational databases in managing large-scale applications, making data retrieval and organization more efficient. As a group, we recognize that understanding these foundational principles is crucial for designing well-structured databases that optimize performance. This discussion has reinforced our appreciation of relational databases as essential tools for handling complex datasets, particularly in projects that require efficient data storage and retrieval.

Chapter 3.2 builds upon these concepts by introducing structured query language and its two main components: commands used to modify data and those used to define the structure of the database. Commands for modification allow users to retrieve, add, update, and remove information within a database, while commands for definition help create and manage the structure of tables and other database elements. The chapter provides practical examples of how these commands are used to interact with databases effectively, demonstrating their role in maintaining and organizing large volumes of information. As a group, we have reflected on the necessity of mastering structured query language as a fundamental skill in managing and developing applications that rely on databases. Understanding the distinction between commands for modifying data and those for defining structure has given us a clearer perspective on how databases are both designed and manipulated. This knowledge equips us with the ability to implement robust data management solutions, ensuring that we can efficiently design, query, and maintain relational databases in real-world applications.

METIS Book #1

Clemente (Chapter 1 and Chapter 2)

Databases play a crucial role in organizing and managing information, making data storage efficient and retrieval seamless. Chapter 1 of *A Guide to SQL* by Philip J. Pratt introduces the

fundamentals of databases, emphasizing their significance in real-world applications. The case studies of TAL Distributors, Colonial Adventure Tours, and Solmaris Condominium Group illustrate how businesses rely on databases to keep operations running smoothly. Each of these organizations has unique data management needs, but the core principles of structuring and maintaining a database remain the same. Building an effective database requires careful planning, which Chapter 2 explores through database design fundamentals and normalization. Without proper structure, data can become redundant and difficult to manage, leading to inefficiencies. Normalization helps eliminate these issues by organizing data into related tables, improving both accuracy and performance.

While normalization might seem complex at first, it ultimately simplifies database management and ensures long-term scalability. A well-designed database is more than just a collection of tables – it is the foundation for reliable data storage and retrieval. Learning these core concepts not only enhances technical skills but also provides a deeper understanding of how businesses and systems operate behind the scenes.

Acantilado (Chapter 1 and 2)

Databases play a crucial role in organizing and managing information, ensuring efficient data storage and retrieval. *A Guide to SQL* by Philip J. Pratt introduces key database concepts and their significance in real-world applications. The case studies of TAL Distributors, Colonial Adventure Tours, and Solmaris Condominium Group illustrate how businesses rely on structured databases to streamline operations and maintain accurate records. Despite their unique data management needs, all three organizations follow the same fundamental principles of database design to ensure efficiency and reliability. A well-structured database prevents data redundancy and inconsistencies, making it easier to store, retrieve, and manage large amounts of information. Without proper organization, data can become difficult to maintain, leading to inefficiencies that impact business performance.

Chapter 2 highlights the importance of careful database planning and the role of normalization in improving data structure. Normalization ensures that information is logically organized into related tables, eliminating duplication and enhancing data accuracy. Although this process may seem complex at first, it ultimately simplifies database management and improves scalability, allowing businesses to handle growing data needs effectively. A well-designed database is not just a collection of tables but a critical foundation for secure and efficient data handling. Learning these core concepts strengthens technical skills and deepens understanding of how businesses and systems operate behind the scenes. Mastering database principles enables better decision-making, allowing for the development of robust and reliable data management solutions that support business growth and efficiency.

Dytianquin (Chapter 1 and 2)

Lists are one of Python's most versatile data structures, offering a wide range of functions that enable efficient data processing. Unlike stacks and queues, which follow strict order constraints such as last-in, first-out and first-in, first-out, lists provide greater flexibility by supporting index-

based, content-based, and position-based operations. A key takeaway from this reading is the distinction between lists and arrays. While both use indices for element access, a list is an abstract data type that can be implemented using either an array or a linked structure, whereas an array is a low-level data structure requiring contiguous memory allocation. This flexibility allows lists to be used in a variety of applications, from implementing complex algorithms to managing dynamic data storage.

Index-based operations enable direct access and modification of elements based on their positions, making lists highly efficient for managing ordered data. Methods such as inserting and removing elements at specific positions allow for dynamic adjustments, enhancing their utility. However, inserting elements in the middle of a list can be computationally expensive, particularly in array-based implementations, as shifting elements to accommodate changes increases processing time. Despite this limitation, lists remain a fundamental and powerful tool in Python programming, supporting a wide range of data manipulation tasks.

Samia (Chapter 1 and 2)

With a large range of functions that enable effective data processing, lists are among Python's most popular and adaptable data structures. Lists offer more flexibility by enabling index-based, content-based, and position-based operations, in contrast to stacks and queues, which adhere to particular order limitations (LIFO and FIFO, respectively). The distinction between lists and arrays is among the most important lessons to be learned from this reading. A list is an abstract data type that can be implemented using either an array or a linked structure, while an array is a low-level data structure that needs contiguous memory allocation. Both employ indices to access elements. Because of their adaptability, lists can be used for a variety of tasks, including sophisticated algorithms and dynamic data storage.

Elements can be directly accessed and modified via their positions thanks to index-based operations. Lists are an effective tool for managing ordered data because of methods like `insert(i, item)` and `pop(i)` that make it possible to insert and remove elements quickly. However, because it necessitates shifting components to make room for the change, adding entries in the middle of a list can be expensive in terms of temporal complexity, particularly in array-based implementations.

METIS Book #2

Clemente (Chapter 24)

Working with databases in Python opens endless possibilities for managing and organizing data efficiently. Chapter 24 of *Core Python Programming* by R. Nageswara Rao explores different types of databases compatible with Python and how they integrate seamlessly with applications. Thorough understanding of these database types is essential for selecting the right one based on performance, scalability, and project requirements. Whether it's relational databases like MySQL or NoSQL options, Python's flexibility makes it a powerful tool for database interaction.

The chapter also introduces working with MySQL in Python, a skill that proves invaluable for real-world applications.

The ability to connect Python to MySQL, execute queries, and manage data programmatically enhances efficiency and automation. Writing SQL queries within Python scripts simplifies database operations, making data retrieval and manipulation more dynamic. Mastering this integration allows developers to build robust, data-driven applications while maintaining flexibility in how information is stored and accessed. Learning how Python interacts with databases is not just about storing data – it's about unlocking the potential for smarter, more efficient applications. As technology continues to evolve, having a solid grasp of database connectivity in Python becomes a crucial skill for any developer working with data.

Acantilado (Chapter 24)

Chapter 24 of *Core Python Programming* by Nageswara Rao explores the integration of MySQL with Python while examining the various database formats compatible with Python. This chapter highlights how Python, as a versatile programming language, enables seamless database connectivity, allowing for efficient data processing, retrieval, and storage. One of the key insights from this reading is Python's ability to support a wide range of databases, including relational databases. Understanding the differences between these database types is essential for making informed decisions when choosing a database for a specific project.

Relational databases ensure structured data organization using tables and defined relationships, making them ideal for applications that require consistency and integrity. In contrast, NoSQL databases provide greater flexibility in handling unstructured or semi-structured data, making them suitable for dynamic and scalable applications. This knowledge is particularly valuable when designing data-driven systems, as selecting the appropriate database format can significantly impact the efficiency and functionality of an application.

Dytianquin (Chapter 24)

Working with databases in Python provides endless opportunities for efficient data management and organization. Chapter 24 of *Core Python Programming* by R. Nageswara Rao delves into the various types of databases compatible with Python and their seamless integration with applications. A thorough understanding of these database types is essential for selecting the most suitable one based on factors such as performance, scalability, and specific project requirements.

The ability to connect Python with MySQL, execute queries, and manage data programmatically enhances both automation and efficiency. Writing database queries within Python scripts simplifies operations, allowing for more dynamic data retrieval and manipulation. Mastering this integration enables developers to create robust, data-driven applications while maintaining flexibility in data storage and access. Understanding how Python interacts with databases is not

just about storing information—it is about leveraging data to build smarter, more efficient systems.

Samia (Chapter 24)

Nageswara Rao focuses on integrating MySQL with Python while investigating the many database formats that can be utilized with Python. This chapter demonstrates how Python, a flexible programming language, offers smooth database connectivity, facilitating effective data processing, retrieval, and storage. The range of databases that Python supports—from relational databases like MySQL, PostgreSQL, and SQLite to NoSQL databases like MongoDB—is one important lesson to be learned. NoSQL databases offer flexibility for managing unstructured or semi-structured data, whereas relational databases guarantee structured data with tables and relationships. This information is crucial for selecting the appropriate database in accordance with project specifications.

METIS Book #3

Clemente (Chapter 3)

Databases serve as the backbone of many applications, making it essential to understand how they function. Chapter 3 of *Python Projects* by Laura Cassell provides a solid introduction to relational database concepts, emphasizing how structured data is stored, managed, and retrieved efficiently. Understanding relationships between tables, keys, and constraints helps in designing databases that maintain integrity and avoid redundancy. The discussion on SQL commands, particularly DML (Data Manipulation Language) and DDL (Data Definition Language), highlights the importance of executing queries to manage data effectively. DML commands like SELECT, INSERT, UPDATE, and DELETE allow interaction with data, while DDL commands define the structure of the database through operations like CREATE and ALTER. Knowing how to use these commands within Python opens opportunities for automating data processes and building dynamic applications.

Working with databases goes beyond simple storage – it’s about ensuring efficiency, reliability, and scalability. Gaining a strong foundation in SQL and relational databases enhances the ability to develop applications that handle data effectively, making this knowledge invaluable for any aspiring programmer.

Acantilado (Chapter 3)

Databases are fundamental to many applications, making it essential to understand their structure. Chapter 3 of *Python Projects* by Laura Cassell introduces key relational database concepts, focusing on how structured data is stored, managed, and retrieved efficiently. A deep understanding of table relationships, keys, and constraints ensures that databases maintain integrity and minimize redundancy, which is crucial for handling large-scale data. The chapter also explores how structured query language enables interaction with databases, defining and

manipulating data to support various application needs. Learning how to integrate database management within Python allows for greater efficiency in automating data processes, enabling developers to build scalable and dynamic applications. Mastering these concepts not only strengthens technical skills but also provides a solid foundation for designing reliable database-driven systems that can adapt to real-world challenges. By leveraging Python's capabilities in database management, developers can create more efficient, automated, and intelligent solutions for handling complex data operations.

Dytianquin (Chapter 3)

Chapter 3 of *Python Projects* by Laura Cassell explores relational database concepts, structured query language, and Python's role in data management. It highlights the limitations of simple file formats like CSV and XML, which lack efficient retrieval and update capabilities, making them less suitable for dynamic applications. In contrast, databases provide a structured and scalable solution, ensuring data integrity and optimized performance. By leveraging relational database systems, developers can efficiently manage interconnected data, enforce relationships, and support concurrent access. Understanding how Python integrates with databases enables automation, improves efficiency, and allows for the creation of scalable, data-driven applications. Mastering these concepts equips developers with the skills to design reliable and adaptable database solutions for modern software development.

Samia (Chapter 3)

An extensive examination of relational database ideas, SQL commands, and Python data management and storage is given in Chapter 3 of Laura Cassell's book *Python Projects*. Because it emphasizes the value of data durability and the function of databases in contemporary applications, this chapter is very illuminating. This chapter's examination of various data storage techniques is one of its most important revelations. For storing structured data, simple files like CSV and XML are helpful, but they don't have effective finding or updating features. In contrast, databases provide a standardized method of storing, retrieving, and manipulating data through SQL instructions, making them a more reliable solution. Applications that need relational data processing, frequent data access, and revisions will find this especially helpful.

Answers to Questions

1. What are DML and DDL statements in Structured Query Language? Give examples of each.

- DML (Data Manipulation Language) is used to manipulate data within tables.
Examples:
 - Retrieve Data: `SELECT * FROM employee;`
 - Insert Data: `INSERT INTO employee (id, name) VALUES (1, 'Kirsten');`
 - Update Data: `UPDATE employee SET name = 'Matthew' WHERE id = 1;`
 - Delete Data: `DELETE FROM employee WHERE id = 1;`

- DDL (Data Definition Language) is used to define and modify database structures.

Examples:

- Create a Table: `CREATE TABLE students (id INTEGER, name TEXT);`
- Modify a Table: `ALTER TABLE students ADD COLUMN age INTEGER;`
- Delete a Table: `DROP TABLE students`

2. What are the categories of SQLite Functions? Give 3 examples of each category.

- Aggregate Functions perform calculations on multiple rows.

Examples:

- `SUM(column_name)`: Return the summation of a column.
- `AVG(column_name)`: Return the average value of a column.
- `COUNT(column_name)`: Return the number of rows in a column.

- String Functions manipulate text values.

Examples:

- `LENGTH(string)`: Return the length of the specified string.
- `UPPER(string)`: Convert text to uppercase.
- `LOWER(string)`: Convert text to lowercase

- Date & Time Functions work with date and time values.

Examples:

- `DATE('now')`: Return the current date.
- `DATETIME('now')`: Return the current date and time.
- `STRFTIME('%Y', 'now')`: Return the current year.

3. How do you check if you have SQLite installed in system using the Linux terminal?

- Enter "`sqlite3 --version`" into the system's command line interface.

Programming Problems

This post-laboratory report explores three programming exercises that reinforce fundamental concepts in database management, SQL operations, and Python's interaction with relational databases. The tasks focus on designing and optimizing databases for business operations, particularly in managing company resources, which reflect real-world database applications, such as structuring multi-entity relationships, tracking transactions, and ensuring efficient data retrieval for organizations that rely on dynamic information systems. The exercises emphasize critical principles such as database normalization, referential integrity, and structured query formulation to maintain consistency and scalability. Each task was assigned to different group members, with individual contributions documented to ensure clarity and accountability. The development process prioritized structured modelling techniques and best practices in database design to create well-structured schemas that support business operations effectively. By addressing these challenges, the exercises demonstrate how database systems serve as the backbone of modern organizations, enabling efficient data management and decision-making.

A. Machine Problems

Programming Exercise #1

The confines of this section present the structured database representation essential for managing the operational framework of an outdoor adventure class system. A well-defined database design is necessary to establish efficient data organization, ensuring seamless scheduling, participant enrollment, and instructor assignment. The relationships between entities are represented through shorthand notation, providing a clear and concise depiction of data attributes and constraints. Furthermore, the use of crow's foot notation enhances the visualization of these relationships, facilitating a comprehensive understanding of data interactions. By implementing a structured relational model, the database can optimize data retrieval, maintain consistency, and support the efficient management of class operations within the system.

- a. For each participant, list his or her number, last name, first name, address, city, state, postal code, telephone number, and date of birth.*

Shorthand Representation:

Participant (*ParticipantID* (PK), LastName, FirstName, Address, City, State, PostalCode, PhoneNumber, DateOfBirth)

Relationships:

- A participant is able to enroll in multiple classes.
- Each enrollment links a participant to a specific adventure class.
- ParticipantID (Primary Key) uniquely identifies each participant.
- The other attributes store the participant's personal information.

- b. For each adventure class, list the class number, class description, maximum number of people in the class, and class fee.*

Shorthand Representation:

AdventureClass (*ClassID* (PK), ClassDescription, MaxParticipants, ClassFee)

Relationships:

- A single adventure class can have multiple participants enrolled.
- An adventure class is taught by one guide (instructor unknown until the day of the class).
- ClassID (Primary Key) uniquely identifies each adventure class.
- ClassDescription provides details about the class.
- MaxParticipants indicates the maximum number of people allowed in the class.
- ClassFee represents the cost of enrolling in the class.

- c. For each participant, list his or her number, last name, first name, and the class number, class description, and date of the class for each class in which the participant is enrolled.*

Shorthand Representation:

Participant (*ParticipantID* (PK), LastName, FirstName)

AdventureClass (*ClassID* (PK), ClassDescription)

Enrollment (*ParticipantID* (PK, FK), *ClassID* (PK, FK), ClassDate)

Relationships:

- Each participant can enroll in multiple adventure classes.
- Each adventure class can have multiple participants.
- ParticipantID and ClassID form a composite primary key, ensuring each participant can only enroll in the same class once per date.
- ClassDate represents the specific date when the participant takes the class.

d. For each class, list the class date, class number, and class description; and the number, last name, and first name of each participant in the class.

Shorthand Representation:

AdventureClass (*ClassID* (PK), ClassDescription)

Participant (*ParticipantID* (PK), LastName, FirstName)

Enrollment (*ClassID* (PK, FK), *ParticipantID* (PK, FK), ClassDate)

Relationships:

- Each class session (ClassID + ClassDate) has multiple participants.
- Each participant is linked to multiple class sessions through enrollment.
- ClassID links to the AdventureClass table.
- ParticipantID links to the Participant table.
- ClassDate records when the participant attended the class.
- Enrollment keeps track of which participant attended which class on a specific date.

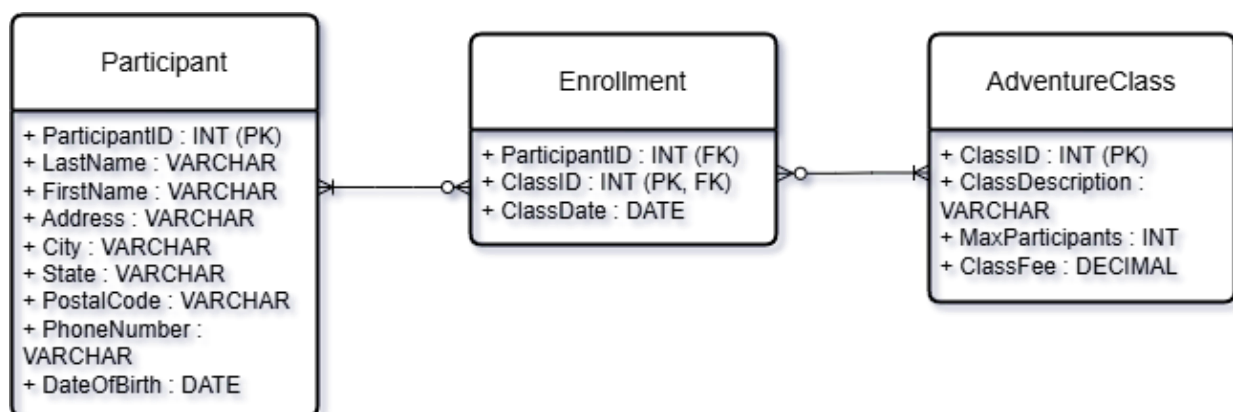


Figure 1 UML Class Diagram Representing the Many-to-Many Relationship Between Participant and AdventureClass

Figure 1 illustrates the many-to-many (M:N) relationship between Participant and AdventureClass, resolved through the Enrollment associative entity. The multiplicity notation

indicates that a Participant can enroll in one or more AdventureClasses (1..*), an AdventureClass can have one or more enrolled participants (1..*), and the Enrollment class serves as the bridge, linking Participants and AdventureClasses with foreign keys while tracking the ClassDate.

Programming Exercise #2

The implementation of a robust database system is crucial for optimizing the operational efficiency of Solmaris Condominium Group's vacation rental management. Given the complexity of managing numerous rental properties, renters, and lease agreements, a well-structured relational database ensures data integrity, minimizes redundancy, and enhances performance. By establishing well-defined entity relationships, this system will facilitate seamless data retrieval, record-keeping, and efficient transaction processing. Furthermore, the database will enable real-time tracking of rental reservations, property availability, and tenant information while supporting business intelligence functions such as financial reporting and occupancy analytics.

```
1  CREATE TABLE Renter (
2      RenterID INT PRIMARY KEY,
3      FirstName VARCHAR(50),
4      MiddleInitial CHAR(1),
5      LastName VARCHAR(50),
6      Address VARCHAR(100),
7      City VARCHAR(50),
8      State CHAR(2),
9      PostalCode VARCHAR(10),
10     TelephoneNumber VARCHAR(15),
11     EmailAddress VARCHAR(100)
12 );
```

Figure 2.1 Database Holding the List of Information for Each Renter

The confines of *Figure 2.1* show the structured database that stores comprehensive information for each tenant who reserves a condominium unit. This database includes essential personal details such as the renter's first name, middle initial, last name, and a unique RenterID, which serves as a primary identifier within the system. Additionally, it maintains complete contact information, including the renter's address, city, state, postal code, phone number, and email address. By systematically organizing this data, the Solmaris Condominium Group can efficiently manage tenant records, facilitate seamless communication, and ensure accurate tracking of rental transactions. Such a structured approach not only enhances the company's ability to verify renter identities and manage rental histories but also streamlines the reservation process, improving overall operational efficiency.

```

14 CREATE TABLE Property (
15     CondoLocationNumber INT,
16     CondoLocationName VARCHAR(100),
17     Address VARCHAR(100),
18     City VARCHAR(50),
19     State CHAR(2),
20     PostalCode VARCHAR(10),
21     CondoUnitNumber INT,
22     SquareFootage INT,
23     NumberOfBedrooms INT,
24     NumberOfBathrooms INT,
25     MaximumOccupancy INT,
26     BaseWeeklyRate DECIMAL(10, 2),
27     PRIMARY KEY (CondoLocationNumber, CondoUnitNumber)
28 );

```

Figure 2.2 Database Holding the List of Information for Each Property

On the other hand, *Figure 2.2* shows the database structure that maintains a comprehensive and up-to-date list of all condominium units available for rent. This database stores key property details, including the `CondoLocationNumber` and `CondoLocationName`, which help identify and categorize rental properties. Additionally, it records complete address information, such as the street address, city, state, and postal code, ensuring precise location tracking. Unit-specific attributes, including condo unit number, square footage, number of bedrooms, and number of bathrooms, provide detailed property specifications for prospective tenants. Furthermore, rental details such as the maximum occupancy limit and base weekly rate are included to support pricing and availability management.

```

30 CREATE TABLE RentalAgreement (
31     RentalAgreementID INT PRIMARY KEY,
32     RenterID INT,
33     CondoLocationNumber INT,
34     CondoUnitNumber INT,
35     StartDate DATE,
36     EndDate DATE,
37     WeeklyRentalAmount DECIMAL(10, 2),
38     FOREIGN KEY (RenterID) REFERENCES Renter(RenterID),
39     FOREIGN KEY (CondoLocationNumber, CondoUnitNumber) REFERENCES Property(CondoLocationNumber, CondoUnitNumber)
40 );

```

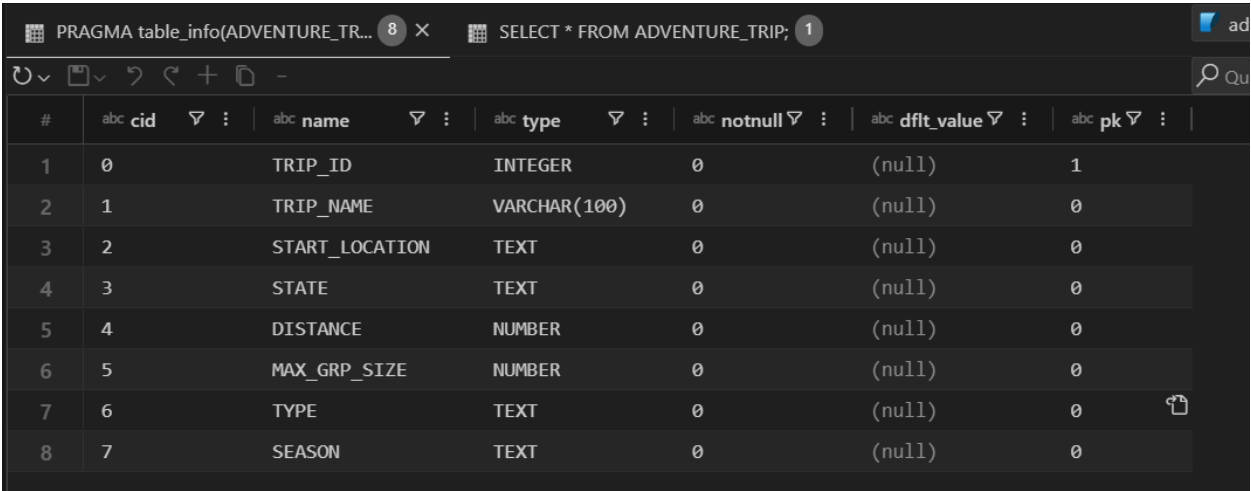
Figure 2.3 Database Holding the List of Information for each Rental Agreement of the User

Lastly, *Figure 2.3* shows the database structure that systematically records all rental agreements between tenants and properties. Each entry includes a `RenterID`, which serves as a unique identifier for the tenant, along with personal and contact details such as first name, middle initial, last name, address, city, state, postal code, and phone number. Additionally, the database tracks rental duration, including the start and end dates, allowing for efficient lease monitoring and occupancy management. The weekly rental amount is also recorded to ensure accurate billing, financial tracking, and compliance with rental agreements. By maintaining a structured and

detailed record of lease information, the system enables the Solmaris Condominium Group to streamline reservation management, monitor rental terms effectively, and ensure precise payment processing, ultimately enhancing overall operational efficiency.

Programming Exercise #3

SQLite serves as a widely utilized relational database management system known for its efficiency, lightweight structure, and ease of integration. Through the use of SQLite commands, databases can be systematically designed, modified, and queried to support structured data management. The following stipulations emphasize the practical application of SQLite commands to perform fundamental database operations, specifically within the confines of an Adventure Trip domain. Hence, by engaging with these tasks, users gain a deeper understanding of database architecture, relational data handling, and structured query language execution, are particularly valuable for developers, analysts, and database administrators seeking to optimize data storage and retrieval in various application scenarios.



#	abc cid	abc name	abc type	abc notnull	abc dfmt_value	abc pk
1	0	TRIP_ID	INTEGER	0	(null)	1
2	1	TRIP_NAME	VARCHAR(100)	0	(null)	0
3	2	START_LOCATION	TEXT	0	(null)	0
4	3	STATE	TEXT	0	(null)	0
5	4	DISTANCE	NUMBER	0	(null)	0
6	5	MAX_GRP_SIZE	NUMBER	0	(null)	0
7	6	TYPE	TEXT	0	(null)	0
8	7	SEASON	TEXT	0	(null)	0

Figure 3.1 Structure of the ADVENTURE_TRIP Table

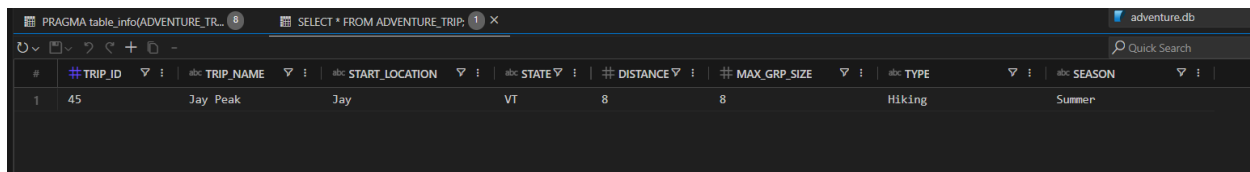
The confines of *Figure 3.1* illustrate the structured database that organizes and manages essential details for each adventure trip. This table follows a well-defined schema that ensures data consistency, accuracy, and efficiency within the system. SQLite, as a relational database management system, plays a crucial role in structuring and maintaining this database by enabling systematic data storage, retrieval, and manipulation. Through the execution of structured commands, the database is designed to enforce data integrity by defining appropriate data types and constraints, preventing errors and inconsistencies in stored records. Additionally, by leveraging SQLite’s lightweight yet powerful architecture, the database optimizes performance, ensuring that queries are executed efficiently, and data is managed effectively. The structured

approach depicted here not only enhances database reliability but also streamlines operations by supporting seamless record management.

```
ProgrammingProblem3a-c.sql X
LR5 > ProgrammingProblem3a-c.sql
1  -- ACANTILADO, MARIA ANGELICA
2  -- DYTIANQUIN, CHALZEA FRANSEN C.
3
4  -- a)
5  CREATE TABLE ADVENTURE_TRIP(
6      TRIP_ID INTEGER PRIMARY KEY,
7      TRIP_NAME VARCHAR(75),
8      START_LOCATION CHAR(50),
9      STATE CHAR(2),
10     DISTANCE NUMBER,
11     MAX_GRP_SIZE NUMBER,
12     TYPE CHAR(20),
13     SEASON CHAR(20)
14 );
15
16 PRAGMA table_info(ADVENTURE_TRIP);
17
18 -- b)
19
20 INSERT INTO ADVENTURE_TRIP (TRIP_ID, TRIP_NAME, START_LOCATION, STATE, DISTANCE, MAX_GRP_SIZE, TYPE, SEASON)
21 VALUES (45, 'Jay Peak', 'Jay', 'VT', 8, 8, 'Hiking', 'Summer');
22
23 SELECT * FROM ADVENTURE_TRIP;
```

Figure 3.2 Code for ADVENTURE_TRIP table

Likewise, *Figure 3.2* illustrates the structured process of inserting a new record into the database and subsequently retrieving the updated table contents. This process begins with executing an insertion command that appends a new row to the table, ensuring that all required attributes are accurately recorded according to the predefined schema. SQLite facilitates this operation by maintaining data integrity, enforcing constraints, and efficiently managing relational data. Once the new entry is successfully added, a retrieval command is executed to display the contents of the table, allowing verification of the updated dataset. By systematically organizing and storing information, SQLite ensures that the database remains structured, accessible, and optimized for efficient query execution. This method supports seamless data entry, retrieval, and management, reinforcing the reliability and accuracy of the stored records within the database environment.



#	TRIP_ID	TRIP_NAME	START_LOCATION	STATE	DISTANCE	MAX_GRP_SIZE	TYPE	SEASON
1	45	Jay Peak	Jay	VT	8	8	Hiking	Summer

Figure 3.3 Added Data in the ADVENTURE_TRIP Table

```
25  -- c)
26
27  DROP TABLE ADVENTURE_TRIP;
28
29
```

Figure 3.4. Deletion of the ADVENTURE_TRIP table

On the other hand, *Figure 3.4* substantiates the structured process of deleting a table from the database, demonstrating SQLite's capability to manage and modify relational data efficiently. The deletion of a table involves executing a command that permanently removes its structure and all stored records, ensuring that no residual data remains within the system. SQLite enforces this operation by eliminating the table from the schema while maintaining database integrity and optimizing storage space. This process is particularly useful when restructuring the database or removing outdated information that is no longer required. As depicted in the underscored figure, properly executing this operation ensures that the database remains organized, efficient, and adaptable to evolving data management needs.

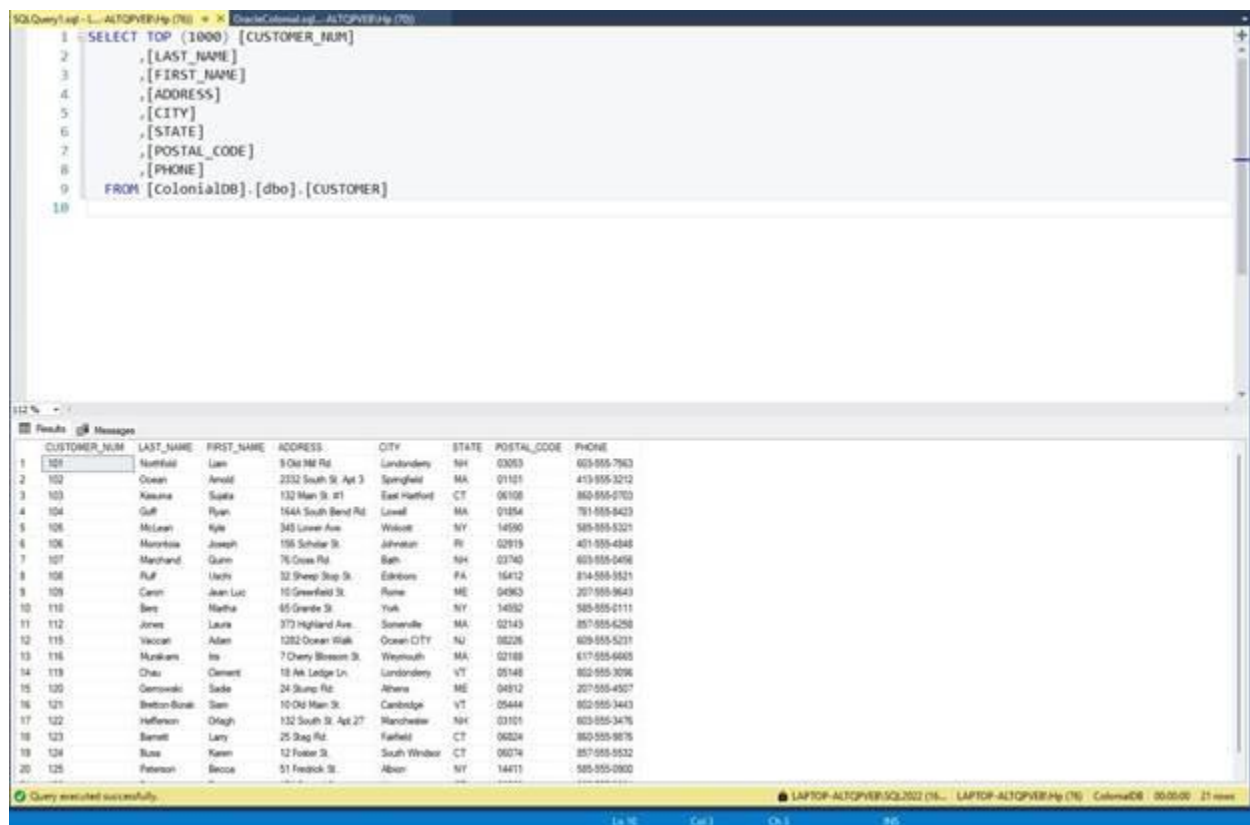


Figure 3.5. Modifying and Executing the SQL Script in SQLite

Figure 3.5 illustrates the process of adapting an existing SQL script to ensure compatibility with SQLite while maintaining database integrity and functionality. This process begins with opening the script file, which contains commands for creating multiple tables and inserting records. Given that SQL syntax can vary across database management systems, necessary modifications must be made to align with SQLite's specific requirements, such as adjusting data types, constraint definitions, or command structures. Once the script is revised, it is executed within the database environment to create the required tables and populate them with relevant data. SQLite plays a crucial role in ensuring that the revised script runs efficiently, enforcing schema constraints, and

optimizing data storage. As shown in the figure, this structured approach enables seamless integration of database objects while ensuring that the system remains functional, organized, and capable of handling queries effectively.

B. Debugging and Sample Run

Programming Exercise #2

```
def load_data():
    with open('Database.json', 'r') as file:
        data = json.load(file)
    return data

def fetch_renters(data):
    print("Fetching all renters...")
    for renter in data["Renters"]:
        print(renter)

def fetch_properties(data):
    print("Fetching all properties...")
    for property in data["Properties"]:
        print(property)

def fetch_rental_agreements(data):
    print("Fetching all rental agreements...")
    for agreement in data["RentalAgreements"]:
        print(agreement)

# Main program
if __name__ == "__main__":
    data = load_data()
    fetch_renters(data)
    fetch_properties(data)
    fetch_rental_agreements(data)
```

Figure 4.1. Python Script for Programming Exercise #2 Database

The confines of *Figure 4.1* illustrate the structured implementation of a Python script to interact with the database, facilitating efficient data retrieval and processing. By integrating database queries within a scripted environment, the system enables seamless extraction, organization, and presentation of stored records. The script establishes a connection to the database, executes queries, and processes the retrieved information in a structured format, ensuring consistency and accessibility. This approach enhances automation, reducing manual intervention while improving efficiency in managing rental records. The structured execution of commands ensures that relevant data is efficiently retrieved and displayed, supporting streamlined database operations and improving overall system functionality.

```
26
27 # Main program

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
310\python.exe "c:\Users\User\.vscode\extensions\ms-python.debugpy-2025.0.1-win32-x64\bundle\libs\debugpy\launcher" '58774' '--' 'D:\Cardinal Life\Anaconda\Troll\Eventlink\backend\rou
tes\Lab5_1.py'
PS D:\Cardinal Life\Anaconda\Troll\Eventlink\backend\routes> ^C
PS D:\Cardinal Life\Anaconda\Troll\Eventlink\backend\routes>
PS D:\Cardinal Life\Anaconda\Troll\Eventlink\backend\routes> d:; cd 'd:\Cardinal Life\Anaconda\Troll\Eventlink\backend\routes'; & 'c:\Users\User\AppData\Local\Programs\Python\Python310
\python.exe' 'c:\Users\User\.vscode\extensions\ms-python.debugpy-2025.0.1-win32-x64\bundle\libs\debugpy\launcher' '58791' '--' 'D:\Cardinal Life\Anaconda\Troll\Eventlink\backend\routes
\Lab5_1.py'
Fetching all renters...
{'RenterID': 1, 'FirstName': 'John', 'MiddleInitial': 'A', 'LastName': 'Doe', 'Address': '123 Main St', 'City': 'New York', 'State': 'NY', 'PostalCode': '10001', 'TelephoneNumber': '555
-1234', 'EmailAddress': 'john.doe@example.com'}
Fetching all properties...
{'CondoLocationNumber': 101, 'CondoLocationName': 'Beachfront Condo', 'Address': '456 Ocean Dr', 'City': 'Miami', 'State': 'FL', 'PostalCode': '33101', 'CondoUnitNumber': 201, 'SquareFo
otage': 1200, 'NumberOfBedrooms': 2, 'NumberOfBathrooms': 2, 'MaximumOccupancy': 4, 'BaseWeeklyRate': 1500.0}
Fetching all rental agreements...
{'RentalAgreementID': 1, 'RenterID': 1, 'CondoLocationNumber': 101, 'CondoUnitNumber': 201, 'StartDate': '2023-11-01', 'EndDate': '2023-11-08', 'WeeklyRentalAmount': 1500.0}
PS D:\Cardinal Life\Anaconda\Troll\Eventlink\backend\routes> []
```

Figure 4.2. Python Script Execution of Programming Exercise #2 Database

On the other hand, *Figure 4.2* illustrates the execution process of the Python script, demonstrating its ability to interact with the database and retrieve structured information. The script reads and processes each line of data, extracting relevant records and presenting them in a clear and organized format. This method ensures efficient data handling while maintaining accuracy and accessibility. By leveraging a lightweight scripting approach, this execution process provides a practical and effective alternative to traditional database management systems, making it particularly suitable for small-scale applications and rapid prototyping. The structured retrieval and display of information enhance database interaction, optimizing functionality while maintaining ease of use.

Programming Exercise #3

Ultimately, the following outputs provide a comprehensive comparison, demonstrating the correctness of the database setup in accordance with the established figures under the discussion for Programming Exercise #3. By carefully reviewing each table's structure and verifying that all records have been accurately inserted, this step ensures that the database aligns with the intended design specifications. Any inconsistencies in table attributes, data types, or values can be identified and addressed through this comparison, confirming that it is fully operational.

#	abc GUIDE_NUM ▾	abc LAST_NAME ▾	abc FIRST_NAME ▾	abc ADDRESS ▾	abc CITY ▾	abc STATE ▾	abc POSTAL_CODE ▾	abc PHONE_NUM ▾	HIRE DA... ▾
1	BR01	Boyers	Rita	140 Oakton Rd.	Jaffrey	NH	03452	603-555-2134	2012-03-03
2	AM01	Abrams	Miles	54 Quest Ave.	Williamsburg	MA	01096	617-555-6032	2012-06-02
3	UG01	Unser	Glory	342 Pineview St.	Danbury	CT	06810	203-555-8534	2015-02-01
4	SL01	Stevens	Lori	15 Riverton Rd.	Coventry	VT	05825	802-555-3339	2014-09-04
5	DH01	Devon	Harley	25 Old Ranch Rd.	Sunderland	MA	01375	781-555-7767	2012-01-07
6	GZ01	Gregory	Zach	7 Moose Head Rd.	Dummer	NH	03588	603-555-8765	2012-11-03
7	KS01	Kiley	Susan	943 Oakton Rd.	Jaffrey	NH	03452	603-555-1230	2013-04-07
8	KS02	Kelly	Sam	9 Congaree Ave.	Fraconia	NH	03580	603-555-0003	2013-06-09
9	MR01	Marston	Ray	24 Shenandoah Rd.	Springfield	MA	01101	781-555-2323	2015-09-13
10	RH01	Rowan	Hal	12 Heather Rd.	Mount Desert	ME	04660	207-555-9009	2014-06-01

Figure 5.1. Customer Database Output for ADVENTURE_TRIP Table

1	1600001	40	2016-03-25	2	55	0	101
2	1600002	21	2016-06-07	2	95	0	101
3	1600003	28	2016-09-11	1	35	0	103
4	1600004	26	2016-10-15	4	45	15	104
5	1600005	39	2016-06-24	5	55	0	105
6	1600006	32	2016-06-17	1	80	20	106
7	1600007	22	2016-07-08	8	75	10	107
8	1600008	28	2016-09-11	2	35	0	108
9	1600009	38	2016-09-10	2	90	40	109
10	1600010	2	2016-05-13	3	25	0	102
11	1600011	3	2016-09-14	3	25	0	102
12	1600012	1	2016-06-11	4	15	0	115
13	1600013	8	2016-07-08	1	20	5	116
14	1600014	12	2016-09-30	2	40	5	119
15	1600015	10	2016-07-22	1	20	0	120
16	1600016	11	2016-07-22	6	75	15	121
17	1600017	39	2016-06-17	3	20	5	122
18	1600018	38	2016-09-17	4	85	15	126
19	1600019	25	2016-08-28	2	110	25	124
20	1600020	28	2016-08-26	2	35	10	124
21	1600021	32	2016-06-10	3	90	20	112
22	1600022	21	2016-06-07	1	95	25	119
23	1600024	38	2016-09-10	1	70	30	121
24	1600025	38	2016-09-10	2	70	45	125
25	1600026	12	2016-09-30	2	40	0	126
26	1600029	4	2016-09-18	4	105	25	120
27	1600030	15	2016-07-24	6	60	15	104

Figure 5.2. Guide Database Output for ADVENTURE_TRIP Table

#	TRIP_ID	TRIP_NAME	START_LOCATION	STATE	DISTANCE	MAX_GRP_SIZE	TYPE	SEASON
1	1	Aethusa Falls	Harts Location	NH	5	10	Hiking	Summer
2	2	Mt. Ascutney - North Peak	Weathersfield	VT	5	6	Hiking	Late Spring
3	3	Mt. Ascutney - West Peak	Weathersfield	VT	6	10	Hiking	Early Fall
4	4	Bradbury Mountain Ride	Lewiston-Auburn	ME	25	8	Biking	Early Fall
5	5	Baldpate Mountain	North Newry	ME	6	10	Hiking	Late Spring
6	6	Blueberry Mountain	Batchelders Grant	ME	8	8	Hiking	Early Fall
7	7	Bloomfield - Maidstone	Bloomfield	CT	10	6	Paddling	Late Spring
8	8	Black Pond	Lincoln	NH	8	12	Hiking	Summer
9	9	Big Rock Cave	Tamworth	NH	6	10	Hiking	Summer
10	10	Mt. Cardigan - Firescrew	Orange	NH	7	8	Hiking	Summer
11	11	Chocorua Lake Tour	Tamworth	NH	12	15	Paddling	Summer
12	12	Cadillac Mountain Ride	Bar Harbor	ME	8	16	Biking	Early Fall
13	13	Cadillac Mountain	Bar Harbor	ME	7	8	Hiking	Late Spring
14	14	Cannon Mtn	Franconia	NH	6	6	Hiking	Early Fall
15	15	Crawford Path Presidentials Hike	Crawford Notch	NH	16	4	Hiking	Summer
16	16	Cherry Pond	Whitefield	NH	6	16	Hiking	Spring
17	17	Huguenot Head Hike	Bar Harbor	ME	5	10	Hiking	Early Fall
18	18	Low Bald Spot Hike	Pinkam Notch	NH	8	6	Hiking	Early Fall
19	19	Mason's Farm	North Stratford	CT	12	7	Paddling	Late Spring
20	20	Lake Mephemagog Tour	Newport	VT	8	15	Paddling	Late Spring
21	21	Long Pond	Rutland	MA	8	12	Hiking	Summer
22	22	Long Pond Tour	Greenville	ME	12	10	Paddling	Summer
23	23	Lower Pond Tour	Poland	ME	8	15	Paddling	Late Spring
24	24	Mt. Adams	Randolph	NH	9	6	Hiking	Summer
25	25	Mount Battie Ride	Camden	ME	20	8	Biking	Early Fall
26	26	Mount Cardigan Hike	Cardigan	NH	4	16	Hiking	Late Fall
27	27	Mt. Chocorua	Albany	NH	6	10	Hiking	Spring
28	28	Mount Garfield Hike	Woodstock	NH	5	10	Hiking	Early Fall
29	29	Metacomet-Monadnock Trail Hike	Pelham	MA	10	12	Hiking	Late Spring
30	30	McLennan Reservation Hike	Tyringham	MA	6	16	Hiking	Summer
31	31	Missisquoi River - VT	Lowell	VT	12	10	Paddling	Summer
32	32	Northern Forest Canoe Trail	Stark	NH	15	10	Paddling	Summer
33	33	Park Loop Ride	Mount Desert Island	ME	27	8	Biking	Late Spring
34	34	Pontook Reservoir Tour	Dummer	NH	15	14	Paddling	Late Spring
35	35	Pisgah STATE Park Ride	Northborough	NH	12	10	Biking	Summer
36	36	Pondicherry Trail Ride	White Mountains	NH	15	16	Biking	Late Spring
37	37	Seal Beach Harbor	Bar Harbor	ME	5	16	Hiking	Early Spring
38	38	Sawyer River Ride	Mount Carraigain	NH	10	18	Biking	Early Fall
39	39	Welch and Dickey Mountains Hike	Thorton	NH	5	10	Hiking	Summer
40	40	Wachusett Mountain	Princeton	MA	8	8	Hiking	Early Spring
41	41	Westfield River Loop	Fort Fairfield	ME	20	10	Biking	Late Spring

Figure 5.3. Reservation Database Output for ADVENTURE_TRIP Table

#	CUSTOMER_NUM	LAST_NAME	FIRST_NAME	ADDRESS	CITY	STATE	POSTAL_CODE	PHONE
1	101	Northfold	Liam	9 Old Mill Rd.	Londonderry	NH	03053	603-555-7563
2	102	Ocean	Arnold	2332 South St. Apt 3	Springfield	MA	01101	413-555-3212
3	103	Kasuma	Sujata	132 Main St. #1	East Hartford	CT	06108	860-555-0703
4	104	Goff	Ryan	164A South Bend Rd.	Lowell	MA	01854	781-555-8423
5	105	McLean	Kyle	345 Lower Ave.	Wolcott	NY	14590	585-555-5321
6	106	Morontoia	Joseph	156 Scholar St.	Johnston	RI	02919	401-555-4848
7	107	Marchand	Quinn	76 Cross Rd.	Bath	NH	03740	603-555-0456
8	108	Rulf	Uschi	32 Sheep Stop St.	Edinboro	PA	16412	814-555-5521
9	109	Caron	Jean Luc	10 Greenfield St.	Rome	ME	04963	207-555-9643
10	110	Bers	Martha	65 Granite St.	York	NY	14592	585-555-0111
11	112	Jones	Laura	373 Highland Ave.	Somerville	MA	02143	857-555-6258
12	115	Vaccari	Adam	1282 Ocean Walk	Ocean CITY	NJ	08226	609-555-5231
13	116	Murakami	Iris	7 Cherry Blossom St.	Weymouth	MA	02188	617-555-6665
14	119	Chau	Clement	18 Ark Ledge Ln.	Londonderry	VT	05148	802-555-3096
15	120	Gernowski	Sadie	24 Stump Rd.	Athens	ME	04912	207-555-4507
16	121	Bretton-Borak	Siam	10 Old Main St.	Cambridge	VT	05444	802-555-3443
17	122	Hefferson	Orlagh	132 South St. Apt 27	Manchester	NH	03101	603-555-3476
18	123	Barnett	Larry	25 Stag Rd.	Fairfield	CT	06824	860-555-9876
19	124	Busa	Karen	12 Foster St.	South Windsor	CT	06074	857-555-5532
20	125	Peterson	Becca	51 Fredrick St.	Albion	NY	14411	585-555-0900
21	126	Brown	Brianne	154 Central St.	Vernon	CT	06066	860-555-3234

Figure 5.4. Trip_Guide Database Output for ADVENTURE_TRIP Table