KENDRIYA VIDYALAYA RAILWAY GANDHIDHAM

Project Work In
Computer Science (Code No -083)
2021 – 2022

Submitted By:
Name:- Vishal Meena
Roll No:- 11611181
Class:- 12th

CERTIFICATE

This is to certify that the **COMPUTER SCIENCE** Project work has been successfully completed by Master **Vishal Meena** Roll No. **11611181** of class **12th** during the academic session **2021 – 2022**. And this is the fulfillment of Term-1 CBSE AISSCE Examination 2022.

Examiner No:

Date:

Teacher In-Charge External Examiner

Principal

ACKNOWLEDGEMENT

It is a profound privilege to express my deep sense of gratitude towards our school **Kendriya Vidyalaya Railway Gandhidham**.

I would also like to thank our respected Principal **Mr. Chandan Singh Pilkhwal** for their encouragement and appreciation.

I wish to extend my gratitude in all sincerity to our teacher **Mr.Vinay Kumar Chauhan** who gave me an opportunity to work under his guidance. He has been a constant source of inspiration and without his valuable guidance the project would not have been successful. I would also like to thank our other teachers and staff members who provided me help and support regarding this project. At last, I am also thankful to all my classmates who helped me with wholeheartedness during the development of the project.

Table of contents

Introduction to Python

Python is an interpreted high-level general-purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

Guido van Rossum began working on Python in the late 1980s, as a successor to the ABC programming language, and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features, such as list comprehensions and a cycle-detecting garbage collection system (in addition to reference counting). Python 3.0 was released in 2008 and was a major revision of the language that is not completely backward-compatible. Python 2 was discontinued with version 2.7.18 in 2020.

Python consistently ranks as one of the most popular programming languages.

Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to ABC programming language, which was inspired by SETL, capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989. Van Rossum shouldered sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his "permanent vacation" from his responsibilities as Python's "Benevolent Dictator For Life", a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker. In January 2019, active Python core developers elected a five-member "Steering Council" to lead the project.

Python 2.0 was released on 16 October 2000, with many major new features, including a cycle-detecting garbage collector (in addition to reference counting) for memory management and support for Unicode.

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible. Many of its major features were backported to Python 2.6.x and 2.7.x version series. Releases of Python 3 include the 2 to 3 utility, which automates the translation of Python 2 code to Python 3.

Python 2.7's end-of-life date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3. No more security patches or other improvements will be released for it. With Python 2's end-of-life, only Python 3.6.x and later are supported.

Python 3.9.2 and 3.8.8 were expedited as all versions of Python (including 2.7) had security issues, leading to possible remote code execution and web cache poisoning.

Advantages of the Python Programming Language

Not for nothing, the biggest companies in the world use Python. It is taken advantage of by Pixar to produce films, by Google to crawl pages, by Netflix to deliver content, and by Spotify to recommend songs. The language is full of benefits and there are some good reasons to love it,

- **Simplicity.** Python's straightforward and simple syntax is something that makes beginners want to learn this scripting language. From some perspective, it may seem natural and pre-determined that Python can turn into the lingua franca of coding, manifesting all the rest of its opponents obsolete. Its code is easy to comprehend, share, and maintain.There is no verbosity and the language is easy to learn.
- **A powerful toolbox.** Inherently, Python programs are text files containing instructions for the interpreter and are written in a text editor or IDE. IDEs are full-featured and offer in-built tools like syntax checkers, debuggers and code browsers, text editors do not normally include IDE features but they can be customized. Python also has a huge array of third-party packages, libraries, and frameworks that facilitate the development process. These optimization capabilities thus make Python great for large-scale projects.
- **Development speed.** We mean business speed here and the time-to-market metric. Python is a dynamic scripting language, so it isn't intended for writing applications from scratch but it's primarily intended for plugging together components. Components are designed to be reusable while the interfaces between components and scripts are well-defined. It all accelerates the speed of software development with Python making the language highly concise and productive.
- **Flexibility.** Although Python puts emphasis on code simplicity and readability rather than flexibility, the language still has it. Python is usable across different projects. It allows developers to choose between object-oriented and procedural programming modes. Python is flexible in data type, too. There are 5 of them: Number, String, List, Tuple, and Dictionary and every sub-data type corresponds to one of these root types. As a result, the exploratory data analysis becomes easier to conduct due to Python's flexibility.
- **Portability.** Python is designed to be portable. Its programs are supported on any modern computer OS. Owing to the high-level nature of the language, Python script is interpreted, so it can be written for further interpretation equally well on Linux, Windows, Mac OS, and UNIX without demanding for adjustments. Python programs also allow implementing portable GUIs.
- **A strong community.** Python has a rapidly growing user base and actually is representative of what a strong community is. There are thousands of contributors to Python's powerful toolbox — Pythonists. There are already almost 200,000 custom-built software packages user-uploaded to an online repository. All it implies that the great supportive community is both the reason for and the consequence of the language's being in demand.

Student Database Management System

Using CSV File Format And File Handling

Synopsis

Abstract:

Student Information Management System can be used by education institutes to maintain the records of students easily. Achieving this objective is difficult using a manual system as the information is scattered, can be redundant and collecting relevant information may be very time consuming. All these problems are solved using this project.

Name of the Project: Student Database Management System

Objectives:

- Maintenance of student records
- Searching student records
- Updating of student records

Software:

Operating Systems : Microsoft Windows
IDE : IDE Python
Front End : Python 3.10
Back End : CSV File Handling

Purpose

Manages Student Information

A student database management system manages all information pertaining to students' attendance, assignments, academic reports, curriculum details, project details, exam details, grades, achievements, medical history, address, accounts, and much more. It also allows teachers to access all student-related information with ease.

Streamlines Communication

A student database management system facilitates smooth communication between students and teachers, parents and teachers, and between students. Gone are the days when teachers used to send notes to parents in the school diary. With student database software, teachers can notify parents about important school events such as parent-teacher meetings, sports day/annual day celebrations, etc instantaneously. The software also allows teachers to send instant updates to parents about their ward's attendance, performance reports, disciplinary problems, students' bus boarding, school bus arrival, and much more over SMS or email.

Student database management software also facilitates smooth interaction between teachers and students by allowing them to collaborate beyond the four walls of the classroom. The software allows the interaction to take place over the online application where teachers can respond to students' queries in real-time.

Admission Management

An efficient student database management system takes care of the admission process as well. The software takes care of the registration process, admission approval, documents uploading, test and interview schedules, and much more, thus, reducing workloads to a great extent.

Student database software also allows students and parents to fill application forms as per their convenience. Thus, students and their parents don't need standing in long queues to get an application form or to get their admission-related queries answered as everything is done online.

Minimizes Paperwork

Student database management system helps you manage students' personal records effortlessly. It keeps the digital track of student data, thus reducing paperwork. Other than personal records, a lot of documents such as admission forms, student records, financial aid paperwork, etc. are created and filed by schools on a regular basis. It is important to maintain the security of these records not only for future reference but also because those documents may contain sensitive information about students. Managing and securing large volumes of paper files is not an easy task as you may end up misplacing the files, or the files may fall into wrong hands.

With a student database management system you neither have to worry about security risks nor have to worry about losing any physical file or confidential document. The software stores each document securely and also helps you track the required document in the blink of an eye.

Extremely Secure and Reliable

Cloud-based student database management systems are way more secure and reliable than manually stored information. Student database software stores all student and school-related data on a cloud-based server to secure your files. Moreover, role-based access prevents unauthorized access to your highly confidential files. The software offers multiple automatic backups of data to make sure you do not end up losing your valuable files. What's more, automated security updates also prevent the software from getting hacked by cybercriminals.

Simplifies Attendance Management

A student database management system leverages biometric tools to help teachers maintain quick records of students' attendance. The software offers student attendance management features that require students to scan their fingerprints while entering or leaving the classroom or school premises. The software automatically takes care of the attendance of students, including half days, late coming, etc., without any human intervention.

Thus, a student database management software relieves teachers from the monotonous task of calling out each student's name for attendance. It also keeps proxy attendance at bay and provides teachers with summaries of absentees and regular latecomers. What's more, this feature also allows school administrators to keep tabs on teachers' and other staff members' attendance besides helping them calculate leaves quickly and accurately.

Easy Library Management

Student database management system comes with library management tools to streamline the process of managing in-house operations of school libraries. The library management tool offers a lot of benefits to school librarians, students, teachers, as well as parents. It assigns a barcode to each library book to facilitate seamless tracking, cataloging, and transaction of books. The tool also allows library users to search, check status, and read books and periodicals online.

Functionality

The GUI will provide the options to access the databases of different classes and of specific students. The database will also save the dates of admission .
There will be a menu with options to,

- Add new student
- Remove existing student
- Update the information of students
- Print desired information of students (through CSV files)

Primary Keys

- Roll No.
- Name
- Blood group
- Sex
- Phone number
- Attendance
- Fee Status
- Marks

Sample Data (from database of a random class)
(Maybe different in practice)

Roll No.	Name	Blood group	Sex	Phone Number	Attendance (in %)	Fee Status	Marks
1	A	B+	M		90	PAID	
2	B	A+	F		75	PAID	
3	C	O-	F		45	PAID	
4	D	AB-	M		85	UNPAID	
5	E	B+	F		65	UNPAID	
6	F	A-	F		25	PAID	
7	G	AB+	M		95	PAID	

Modules and Methods used

Modules

- CSV
- Pandas
- SQLite

What Is a CSV File?

A CSV file (Comma Separated Values file) is a type of plain text file that uses specific structuring to arrange tabular data. Because it's a plain text file, it can contain only actual text data—in other words, printable ASCII or Unicode characters.

The structure of a CSV file is given away by its name. Normally, CSV files use a comma to separate each specific data value. Here's what that structure looks like:

```
column 1 name,column 2 name, column 3 name
first row data 1,first row data 2,first row data 3
second row data 1,second row data 2,second row data 3
...
```

Notice how each piece of data is separated by a comma. Normally, the first line identifies each piece of data—in other words, the name of a data column. Every subsequent line after that is actual data and is limited only by file size constraints.

In general, the separator character is called a delimiter, and the comma is not the only one used. Other popular delimiters include the tab (\t), colon (:) and semicolon (;) characters. Properly parsing a CSV file requires us to know which delimiter is being used.

Where Do CSV Files Come From?

CSV files are normally created by programs that handle large amounts of data. They are a convenient way to export data from spreadsheets and databases as well as import or use it in other programs. For example, you might export the results of a data mining program to a CSV file and then import that into a spreadsheet to analyze the data, generate graphs for a presentation, or prepare a report for publication.

CSV files are very easy to work with programmatically. Any language that supports text file input and string manipulation (like Python) can work with CSV files directly.

Reading CSV Files With `csv`

Reading from a CSV file is done using the reader object. The CSV file is opened as a text file with Python's built-in `open()` function, which returns a file object. This is then passed to the reader, which does the heavy lifting.

Here's the employee_birthday.txt file:

```
name,department,birthday month

John Smith,Accounting,November

Erica Meyers,IT,March
```

Pandas Library

pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals. Its name is a play on the phrase "Python data analysis" itself. Wes McKinney started building what would become pandas at AQR Capital while he was a researcher there from 2007 to 2010.

Library Features

- DataFrame object for data manipulation with integrated indexing.
- Tools for reading and writing data between in-memory data structures and different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of data sets.
- Label-based slicing, fancy indexing, and subsetting of large data sets.
- Data structure column insertion and deletion.
- Group by engine allowing split-apply-combine operations on data sets.
- Data set merging and joining.
- Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure.
- Time series-functionality: Date range generation and frequency conversions, moving window statistics, moving window linear regressions, date shifting and lagging.
- Provides data filtration.

The library is highly optimized for performance.

Dataframes

Pandas is mainly used for data analysis. Pandas allows importing data from various file formats such as comma-separated values, JSON, SQL, and Microsoft Excel. Pandas allows various data manipulation operations such as merging, reshaping, selecting, as well as data cleaning, and data wrangling features.

Reading CSV Files With pandas

To show some of the power of pandas CSV capabilities, I've created a slightly more complicated file to read, called hrdata.csv. It contains data on company employees:

```
Name,Hire Date,Salary,Sick Days remaining
Graham Chapman,03/15/14,50000.00,10
John Cleese,06/01/15,65000.00,8
Eric Idle,05/12/14,45000.00,10
Terry Jones,11/01/13,70000.00,3
Terry Gilliam,08/12/14,48000.00,7
Michael Palin,05/23/13,66000.00,8
```

Reading the CSV into a pandas DataFrame is quick and straightforward:

```
import pandas
df = pandas.read_csv('hrdata.csv')
print(df)
```

That's it: three lines of code, and only one of them is doing the actual work. pandas.read_csv() opens, analyzes, and reads the CSV file provided, and stores the data in a DataFrame. Printing the DataFrame results in the following output:

```
              Name  Hire Date   Salary  Sick Days  remaining
0  Graham Chapman   03/15/14  50000.0          10
1     John Cleese   06/01/15  65000.0           8
2       Eric Idle   05/12/14  45000.0          10
3     Terry Jones   11/01/13  70000.0           3
4   Terry Gilliam   08/12/14  48000.0           7
5   Michael Palin   05/23/13  66000.0           8
```

Here are a points worth noting:

- First, pandas recognized that the first line of the CSV contained column names, and used them automatically. I call this Goodness.
- However, pandas is also using zero-based integer indices in the DataFrame. That's because we didn't tell it what our index should be.

What is SQL (Structured Query Language)

SQL (Structured Query Language) is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system (RDSMS). It is particularly useful in handling structured data, i.e. data incorporating relations among entities and variables.

SQL was initially developed at IBM by Donald D. Chamberlin and Raymond F. Boyce after learning about the relational model from Edgar F. Codd in the early 1970s. This version, initially called SEQUEL (Structured English Query Language), was designed to manipulate and retrieve data stored in IBM's original quasi relational database management system, System R, which a group at IBM San Jose Research Laboratory had developed during the 1970s.

Chamberlin and Boyce's first attempt at a relational database language was SQUARE (Specifying QUeries in A Relational Environment), but it was difficult to use due to subscript/superscript notation. After moving to the San Jose Research Laboratory in 1973, they began work on a sequel to SQUARE. The name SEQUEL was later changed to SQL (dropping the vowels) because "SEQUEL" was a trademark of the UK-based Hawker Siddeley Dynamics Engineering Limited company. The label SQL later became the acronym for Structured Query Language.

After testing SQL at customer test sites to determine the usefulness and practicality of the system, IBM began developing commercial products based on their System R prototype, including System/38, SQL/DS, and DB2, which were commercially available in 1979, 1981, and 1983, respectively.

In the late 1970s, Relational Software, Inc. (now Oracle Corporation) saw the potential of the concepts described by Codd, Chamberlin, and Boyce, and developed their own SQL-based RDBMS with aspirations of selling it to the U.S. Navy, Central Intelligence Agency, and other U.S. government agencies. In June 1979, Relational Software introduced the first commercially available implementation of SQL, Oracle V2 (Version2) for VAX computers.

By 1986, ANSI and ISO standard groups officially adopted the standard "Database Language SQL" language definition. New versions of the standard were published in 1989, 1992, 1996, 1999, 2003, 2006, 2008, 2011 and most recently, 2016.

Language Elements

The SQL language is subdivided into several language elements, including:

Keywords are words that are defined in the SQL language. They are either reserved (e.g. SELECT, COUNT and YEAR), or non-reserved (e.g. ASC, DOMAIN and KEY). List of SQL reserved words.

Identifiers are names on database objects, like tables, columns and schemas. An identifier may not be equal to a reserved keyword, unless it is a delimited identifier. Delimited identifiers means identifiers enclosed in double quotation marks. They can

contain characters normally not supported in SQL identifiers, and they can be identical to a reserved word, e.g. a column named YEAR is specified as "YEAR".

Clauses, which are constituent components of statements and queries. (In some cases, these are optional.)

Expressions, which can produce either scalar values, or tables consisting of columns and rows of data

Predicates, which specify conditions that can be evaluated to SQL three-valued logic (3VL) (true/false/unknown) or Boolean truth values and are used to limit the effects of statements and queries, or to change program flow.

Queries, which retrieve the data based on specific criteria. This is an important element of *SQL*.

Statements, which may have a persistent effect on schemata and data, or may control transactions, program flow, connections, sessions, or diagnostics.

SQL statements also include the semicolon (";") statement terminator. Though not required on every platform, it is defined as a standard part of the SQL grammar.

Insignificant whitespace is generally ignored in SQL statements and queries, making it easier to format SQL code for readability.

Python - Databases and SQL

The Python programming language has powerful features for database programming. Python supports various databases like SQLite, MySQL, Oracle, Sybase, PostgreSQL, etc. Python also supports Data Definition Language (DDL), Data Manipulation Language (DML) and Data Query Statements. The Python standard for database interfaces is the Python DB-API. Most Python database interfaces adhere to this standard.

In this chapter we will see the use of SQLite databases in the python programming language. It is done by using python's inbuilt, sqlite3 module. You should first create a connection object that represents the database and then create some cursor objects to execute SQL statements.

SQLite

SQLite is a C library that provides a lightweight disk-based database that doesn't require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language. Some applications can use SQLite for internal data storage. It's also possible to prototype an application using SQLite and then port the code to a larger database such as PostgreSQL or Oracle.

The sqlite3 module was written by Gerhard Häring. It provides a SQL interface compliant with the DB-API 2.0 specification described by PEP 249, and requires SQLite 3.7.15 or newer.

To use the module, start by creating a `Connection` object that represents the database. Here the data will be stored in the `example.db` file:

```
import sqlite3
con = sqlite3.connect('example.db')
```

The special path name `:memory:` can be provided to create a temporary database in RAM.

Once a `Connection` has been established, create a `Cursor` object and call its `execute()` method to perform SQL commands:

```
cur = con.cursor()

# Create table
cur.execute('''CREATE TABLE stocks
               (date text, trans text, symbol text, qty real,
price real)''')

# Insert a row of data
cur.execute("INSERT INTO stocks VALUES
('2006-01-05','BUY','RHAT',100,35.14)")

# Save (commit) the changes
con.commit()

# We can also close the connection if we are done with it.
# Just be sure any changes have been committed or they will be
lost.
con.close()
```

The saved data is persistent: it can be reloaded in a subsequent session even after restarting the Python interpreter:

```
import sqlite3
con = sqlite3.connect('example.db')
cur = con.cursor()
```

To retrieve data after executing a SELECT statement, either treat the cursor as an iterator, call the cursor's fetchone() method to retrieve a single matching row, or call fetchall() to get a list of the matching rows.

This example uses the iterator form:

```
>>> for row in cur.execute('SELECT * FROM stocks ORDER BY
price'):

        print(row)
('2006-01-05', 'BUY', 'RHAT', 100, 35.14)
('2006-03-28', 'BUY', 'IBM', 1000, 45.0)
('2006-04-06', 'SELL', 'IBM', 500, 53.0)
('2006-04-05', 'BUY', 'MSFT', 1000, 72.0)
```

Connect To Database

Following Python code shows how to connect to an existing database. If the database does not exist, then it will be created and finally a database object will be returned.

```
import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";
```

Here, you can also supply the database name as the special name :memory: to create a database in RAM. Now, let's run the above program to create our database test.db in the current directory. You can change your path as per your requirement. Keep the above code in sqlite.py file and execute it as shown below. If the database is successfully created, then it will display the following message.

```
$chmod +x sqlite.py
$./sqlite.py
Open database successfully
```

Create a Table

Following Python program will be used to create a table in the previously created database.

```
import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";
conn.execute('''CREATE TABLE COMPANY
         (ID INT PRIMARY KEY     NOT NULL,
         NAME           TEXT    NOT NULL,
         AGE            INT     NOT NULL,
         ADDRESS        CHAR(50),
         SALARY         REAL);''')
print "Table created successfully";
conn.close()
```

When the above program is executed, it will create the COMPANY table in your test.db and it will display the following messages −

```
Opened database successfully
Table created successfully
```

Insert Operation

Following Python program shows how to create records in the COMPANY table created in the above example.

```
import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
```

```
      VALUES (1, 'Paul', 32, 'California', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (2, 'Allen', 25, 'Texas', 15000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (3, 'Teddy', 23, 'Norway', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (4, 'Mark', 25, 'Rich-Mond ', 65000.00 )");

conn.commit()
print "Records created successfully";
conn.close()
```

When the above program is executed, it will create the given records in the COMPANY table and it will display the following two lines −

```
Opened database successfully
Records created successfully
```

Select Operation

Following Python program shows how to fetch and display records from the COMPANY table created in the above example.

```
import sqlite3
conn = sqlite3.connect('test.db')
print "Opened database successfully";
cursor = conn.execute("SELECT id, name, address, salary from
COMPANY")
for row in cursor:
   print "ID = ", row[0]
   print "NAME = ", row[1]
   print "ADDRESS = ", row[2]
   print "SALARY = ", row[3], "\n"
print "Operation done successfully";
conn.close()
```

When the above program is executed, it will produce the following result.

```
Opened database successfully
ID = 1
NAME = Paul
ADDRESS = California
SALARY = 20000.0

ID = 2
NAME = Allen
ADDRESS = Texas
SALARY = 15000.0

ID = 3
NAME = Teddy
ADDRESS = Norway
```

```
SALARY = 20000.0

ID = 4
NAME = Mark
ADDRESS = Rich-Mond
SALARY = 65000.0

Operation done successfully
```

Update Operation

Following Python code shows how to use UPDATE statement to update any record and then fetch and display the updated records from the COMPANY table.

```
import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";

conn.execute("UPDATE COMPANY set SALARY = 25000.00 where ID = 1")
conn.commit
print "Total number of rows updated :", conn.total_changes
cursor = conn.execute("SELECT id, name, address, salary from
COMPANY")
for row in cursor:
   print "ID = ", row[0]
   print "NAME = ", row[1]
   print "ADDRESS = ", row[2]
   print "SALARY = ", row[3], "\n"
print "Operation done successfully";
conn.close()
```

When the above program is executed, it will produce the following result.

```
Opened database successfully
Total number of rows updated : 1
ID = 1
NAME = Paul
ADDRESS = California
SALARY = 25000.0

ID = 2
NAME = Allen
ADDRESS = Texas
SALARY = 15000.0

ID = 3
NAME = Teddy
ADDRESS = Norway
SALARY = 20000.0

ID = 4
NAME = Mark
ADDRESS = Rich-Mond
```

```
SALARY = 65000.0

Operation done successfully
```

Delete Operation

Following Python code shows how to use the DELETE statement to delete any record and then fetch and display the remaining records from the COMPANY table.

```
import sqlite3
conn = sqlite3.connect('test.db')
print "Opened database successfully";
conn.execute("DELETE from COMPANY where ID = 2;")
conn.commit()
print "Total number of rows deleted :", conn.total_changes
cursor = conn.execute("SELECT id, name, address, salary from
COMPANY")
for row in cursor:
   print "ID = ", row[0]
   print "NAME = ", row[1]
   print "ADDRESS = ", row[2]
   print "SALARY = ", row[3], "\n"
print "Operation done successfully";
conn.close()
```

When the above program is executed, it will produce the following result.

```
Opened database successfully
Total number of rows deleted : 1
ID = 1
NAME = Paul
ADDRESS = California
SALARY = 20000.0

ID = 3
NAME = Teddy
ADDRESS = Norway
SALARY = 20000.0

ID = 4
NAME = Mark
ADDRESS = Rich-Mond
SALARY = 65000.0

Operation done successfully
```

Bibliography

Made by :- Vishal Meena,12A, K.V. Rly GIM
Editor :- Amit Meena, 12A, K.V. Rly GIM

Wikipedia

- Info on different topics i.e. SQL, Python, etc. https://en.wikipedia.org

Python (Official website)

- How to https://docs.python.org/3/howto/index.html
- CSV Module https://docs.python.org/3/library/csv.html?highlight=csv#module-csv
- Pandas Reference https://docs.python.org/3/using/windows.html?highlight=pandas
- SQLite https://docs.python.org/3/library/sqlite3.html?highlight=sql#module-sqlite3