Province of the
EASTERN CAPE
EDUCATION

Iphondo leMpuma Kapa: Isebe leMfundo
Provinsie van die Oos Kaap: Department van Onderwys
Porafensie Ya Kapa Botjahabela: Lefapha la Thuto

NATIONAL SENIOR CERTIFICATE

GRADE 12

SEPTEMBER 2025

INFORMATION TECHNOLOGY P1

MARKS: **150**

TIME: **3 hours**

This question paper consists of 24 pages, 2 data sheets, and a page for planning.

INSTRUCTIONS AND INFORMATION

1. This question paper is divided into FOUR sections. Candidates must answer ALL the questions in ALL FOUR sections.

2. One blank page have been provided at the end of the question paper, which may be used for planning purposes.

3. The duration of this examination is three hours. Because of the nature of this examination, it is important to note that you will not be permitted to leave the examination room before the end of the examination session.

4. This question paper is set with programming terms that are specific to the Delphi programming language. The Delphi programming language must be used to answer the questions.

5. Make sure that you answer the questions according to the specifications that are given in each question. Marks will be awarded according to the set requirements.

6. Answer only what is asked in each question. For example, if the question does not ask for data validation, then no marks will be awarded for data validation.

7. Your programs must be coded in such a way that they will work with any data and not just the sample data supplied or any data extracts that appear in the question paper.

8. Routines, such as search, sort and selection, must be developed from first principles. You may NOT use the built-in features of the Delphi programming language for any of these routines.

9. All data structures must be defined by you, the programmer, unless the data structures are supplied.

10. You must save your work regularly on the disk/CD/DVD/flash disk you have been given, or on the disk space allocated to you for this examination session.

11. Make sure that your name and surname appears as a comment in every program that you code, as well as on every event indicated.

12. If required, print the programming code of all the programs/classes that you completed. Your name and surname must appear on all printouts. You will be given half an hour printing time after the examination session.

13. At the end of this examination session, you must hand in a disk/CD/DVD/ flash disk with all your work saved on it OR you must make sure that all your work has been saved on the disk space allocated to you for this examination session. Ensure that all files can be read.

14. Save your work regularly.

15. The files that you need to complete this question paper have been provided to you on the disk/CD/DVD/flash disk or on the disk space allocated to you. The files are provided in the form of password-protected executable file.

    Do the following:

    - Double click on the password-protected executable file: **DataENGSEP2025.exe**
    - Click on the 'Extract' button.
    - Enter the following password: **@Planets#SEP2025**

    Once extracted, the following list of files will be available in the folder **DataENGSEP2025**:

Question 1:

Question1_P.dpr
Question1_P.dproj
Question1_P_Icon.ico
Question1_U.dfm
Question1_U.pas

Question 2:

dbConnection_U.pas
Question2_P.dpr
Question2_P.dproj
Question2_P_Icon.ico
Question2_U.dfm
Question2_U.pas
SolarDB.mdb
SolarDB_Backup.mdb

Question 3:

Planet_U.pas
Planets.txt
Question3_P.dpr
Question3_P.dproj
Question3_P_Icon.ico
Question3_U.dfm
Question3_U.pas

Question 4:

Question4_P.dpr
Question4_P.dproj
Question4_P_Icon.ico
Question4_U.dfm
Question4_U.pas

QUESTION 1: GENERAL PROGRAMMING SKILLS

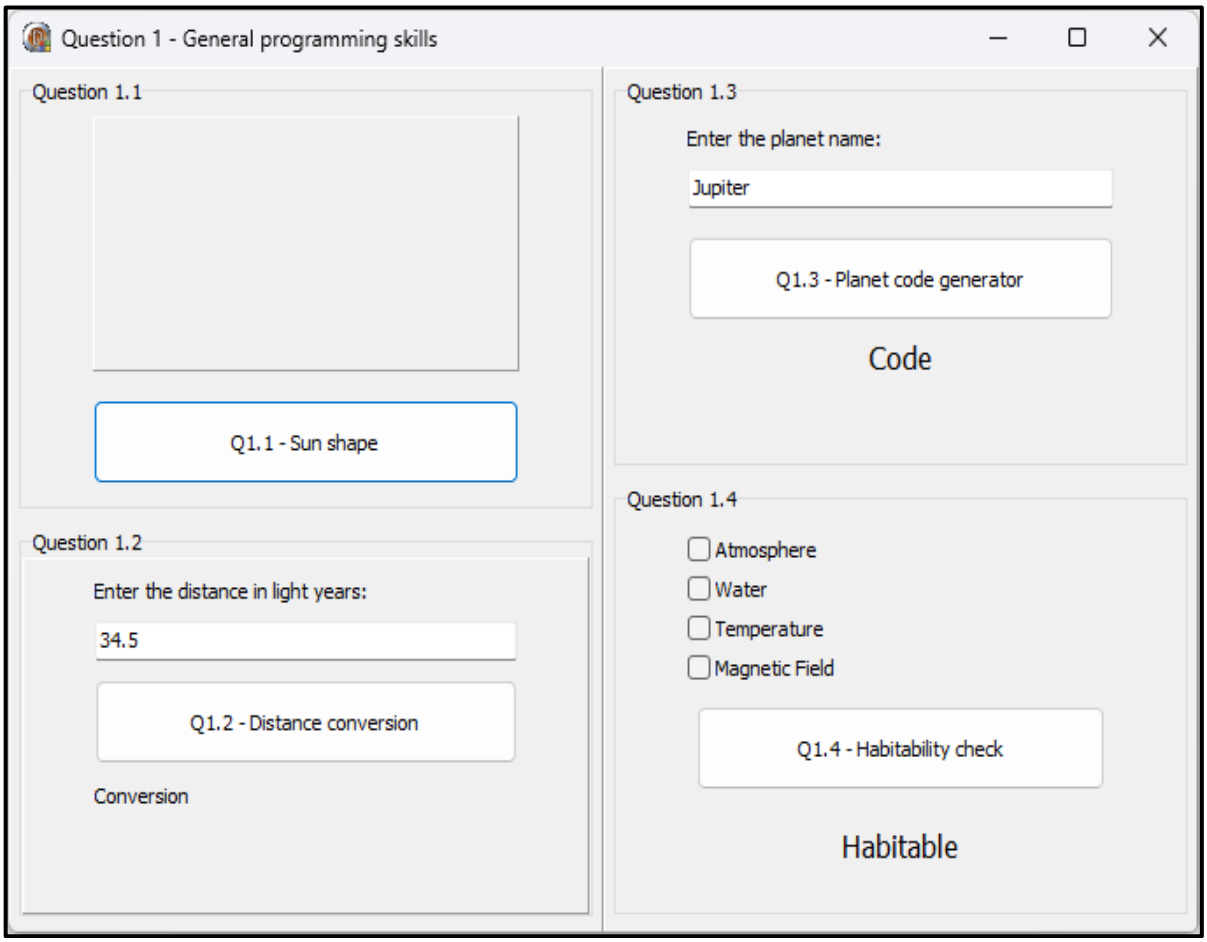
SCENARIO: THE SOLAR SYSTEM

With the renewed interest in space travel and planetary science, the solar system has become a captivating topic of research and exploration. Scientists are developing new ways to study the planets, moons, asteroids and comets that orbit our Sun.

Do the following:

- Open the incomplete program in the **Question 1** folder.
- Enter your name and surname as a comment in the first line of the **Question1_U.pas** file.
- Compile and execute the program. Currently the program has no functionality.

Example of graphical user interface (GUI):



Complete the code for each section of QUESTION 1, as described in QUESTION 1.1 to QUESTION 1.4 that follow.

1.1 **Button [1.1 – Sun shape]**

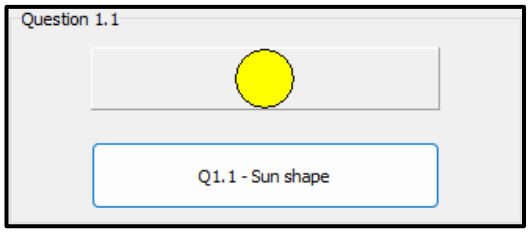
The Sun is the most important celestial object in the solar system. In this question, you will simulate its appearance.

Code the button **btnQ1_1** to **dynamically create** a circular shape that represents the Sun.

The shape must meet the following specifications:

- Correctly assigned to the panel using the Parent property.
- Circular shape.
- 50 x 50 pixels in size.
- Yellow in colour.
- Fill the entire panel **pnlOutput** component. *HINT: Align*

Example of output: *You will not see the yellow in the printed version of the paper.*



(6)

1.2 **Button [1.2 – Distance conversion]**

Light travels vast distances across space. A common measurement used in astronomy is the light year, which represents the distance light travels in one year.

Code the button **btnQ1_2** to do the following:

- Retrieve the value entered in the edit box **edtDistance** component (assumed to be in light years and can contain decimals).
- Convert the value to trillions of kilometers using the formula:

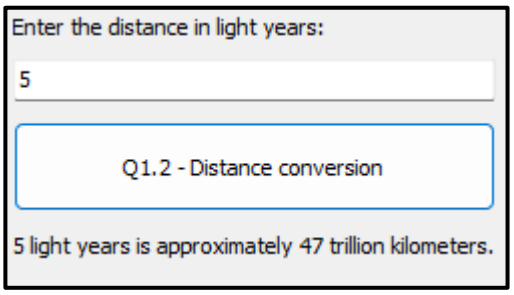
1 light year = 9,461 trillion km.

- Round off the light years and result to the nearest whole number.
- Display the result in the label **lblQ1_2** component in the following format:

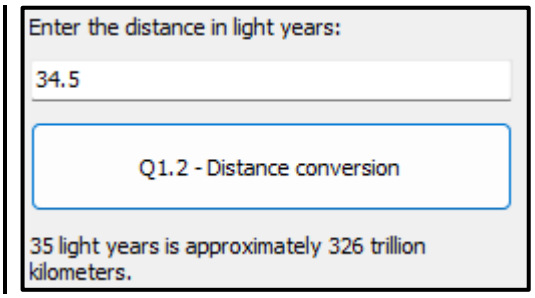
"5 light years is approximately 47 trillion kilometers."

Example of output:

5 light years



34.5 light years



(8)

1.3 **Button [Q1.3 – Planet code generator]**

The Space Database uses a special code for each planet based on the ASCII values of its name.

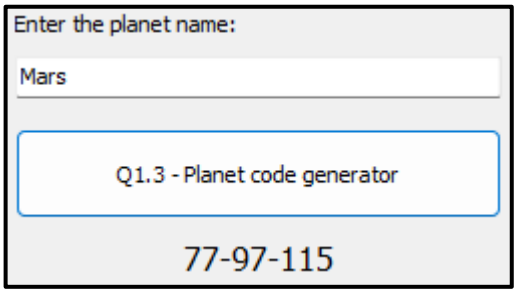
Code the button **btnQ1_3** to do the following:

- Read the name of a planet from the edit box **edtPlanet** component.
- Loop through each character of the name.
- Retrieve the ASCII code of each character.
- If the ASCII code is odd, include it in a string variable.
- Separate the ASCII codes with hyphens.
- Display the final string in the label **lblQ1_3** component.

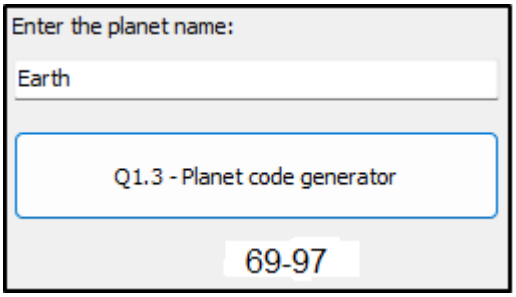
Example:
Input: "Mars"
ASCII values: 77(M), 97(a), 114(r), 115(s)
Output: "77-97-115"

Example of output:

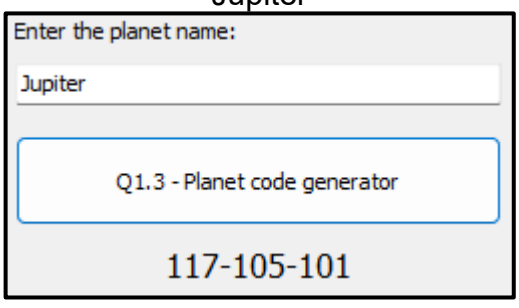
Mars



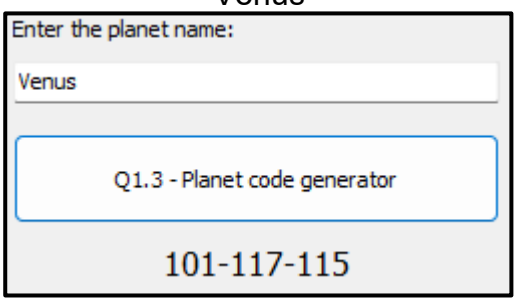
Earth



Jupiter



Venus



(12)

1.4 **Button [Q1.4 – Habitability check]**

To determine if a planet is potentially habitable, scientists consider various factors like atmosphere, presence of water, temperature and a magnetic field.

Each of these characteristics is represented by a checkbox on the form. Select the checkboxes that are applicable to the chosen planet.

Code the button **btnQ1_4** button to do the following:

- Counts the number of selected criteria.
- If 3 or more of the 4 boxes are checked, display:

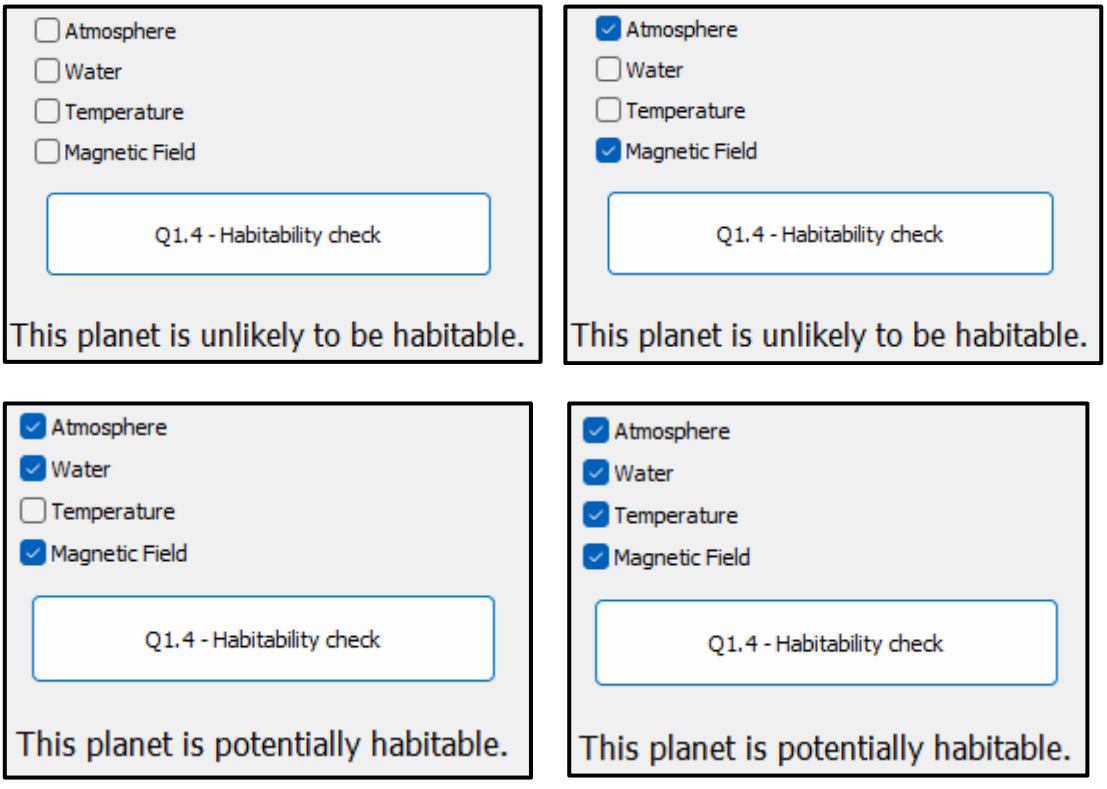
"This planet is potentially habitable."

Otherwise, display:

"This planet is unlikely to be habitable."

- Output must be shown in label **lblQ1_4** component.

Example output:



(9)

- Enter your name and surname as a comment in the first line of the program file.
- Save your program.
- A printout of the code may be required.

[35]

QUESTION 2: SQL AND DATABASE PROGRAMMING

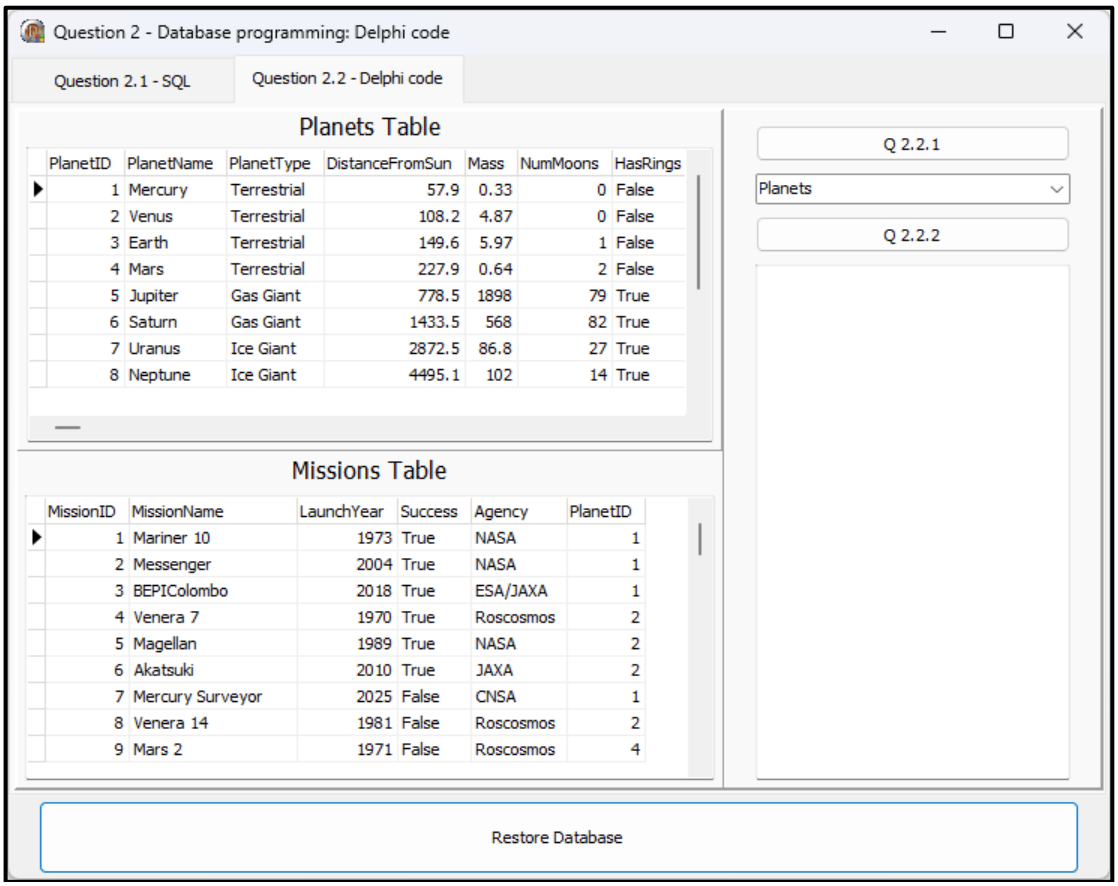
The South African Space Agency is working on an educational initiative to help learners and researchers better understand our solar system. As part of this initiative, a database of the planets and the space missions that have visited them, is being compiled. You are required to assist in developing a system that allows users to query and manage this data.

A database named **SolarDB.mdb** has been provided. It contains two tables: **tblPlanets** and **tblMissions.**

The data pages attached at the end of the question paper provide information on the design of the **SolarDB.mdb** database and its contents.

Do the following:

- Open the incomplete program in the **Question 2** folder.
- Enter your name and surname as a comment in the first line of the **Question2_U.pas** unit file**.**
- Compile and execute the program. The program has limited functionality.
- The contents of the tables are displayed as shown below on the selection of tab sheet **Question 2.2 – Delphi code.**



- Follow the instructions below to complete the code for each section as described in QUESTION 2.1 and QUESTION 2.2.
- Use SQL statements to answer QUESTION 2.1 and Delphi code to answer QUESTION 2.2.

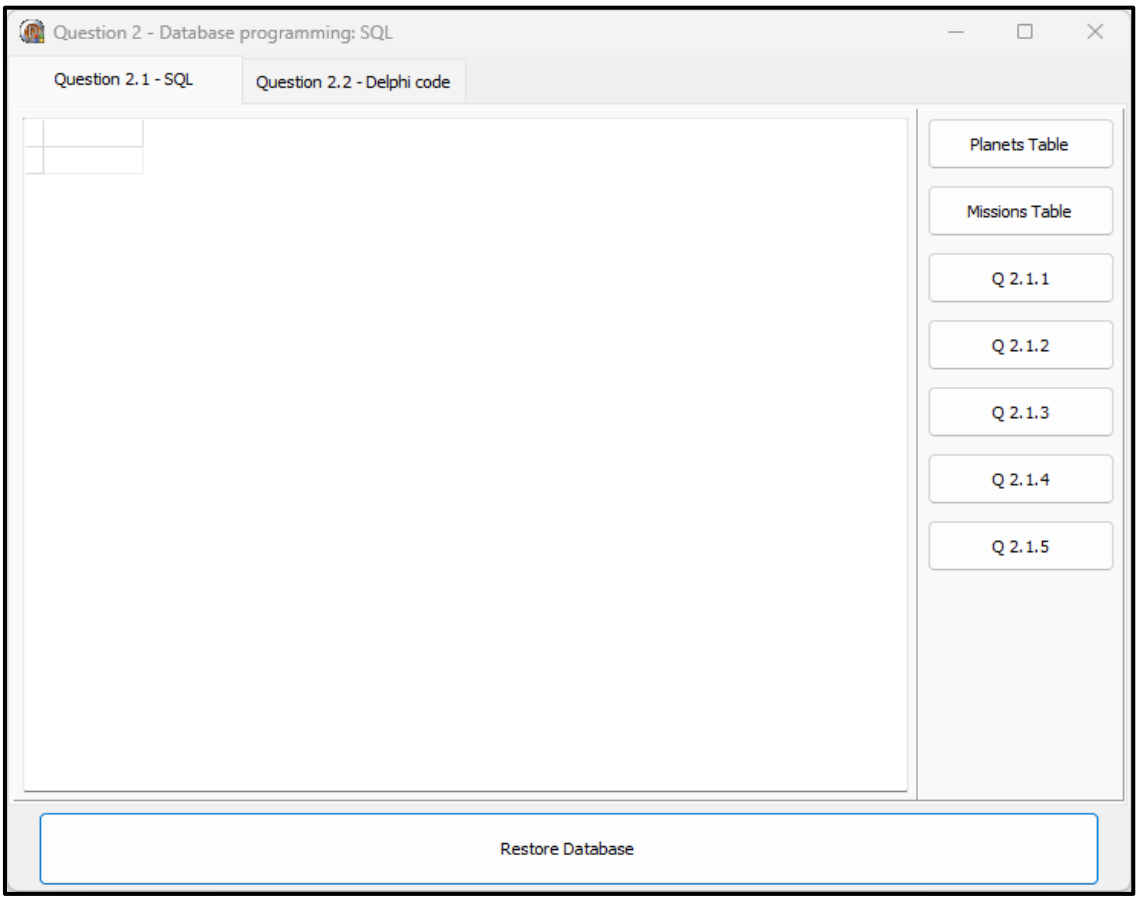
NOTE:

- The 'Restore database' button is provided to restore the data contained in the database to the original content.
- The content of the database is password-protected, i.e. you will NOT be able to gain access to the content of the database using Microsoft Access.
- Code is provided to link the GUI components to the database. Do NOT change any of the code provided.
- THREE variables are declared as public variables, as described in the table below.

Variable	Data type	Description
tblPlanets	TADOTable	Refers to the data stored in the table tblPlanets
tblMissions	TADOTable	Refers to the data stored in the table tblMissions
qryInfo	TADOQuery	Query component that will query the two tables tblPlanets and tblMissions

2.1 Tab Sheet [Question 2.1 - SQL]

Example of graphical user interface (GUI) for QUESTION 2.1.



NOTE:

- Use ONLY SQL code to answer QUESTION 2.1.1 to QUESTION 2.1.5.
- Code to execute the SQL statements and display the results of the queries is provided. The SQL statements assigned to the variables **sSQL1, sSQL2, sSQL3, sSQL4,** and **sSQL5** are incomplete.

Complete the SQL statements to perform the tasks described in QUESTION 2.1.1 to QUESTION 2.1.5 below.

2.1.1 **Button [Q2.1.1]**

Write SQL code to display each PlanetType and the **total number of moons** for that type. Use the alias TotalMoons for the calculated column. The list must be sorted in descending order of moons.

Example of output:

PlanetType	TotalMoons
Gas Giant	161
Ice Giant	41
Terrestrial	3

(6)

2.1.2 **Button [Q2.1.2]**

Write SQL code to display the MissionName, LaunchYear, Success and PlanetName for all successful missions to planets that are classified as Gas Giants.

Use both the tblPlanets and tblMissions tables.

Example of output:

MissionName	LaunchYear	Success	PlanetName
JUICE	2023	True	Jupiter
Galileo	1989	True	Jupiter
New Horizons	2006	True	Jupiter
Pioneer 11	1973	True	Saturn
Cassini-Huygens	1997	True	Saturn
Bosveld Beacon	2032	True	Saturn
Saturn Dreamer	2027	True	Saturn
Zeus Pioneer	2028	True	Jupiter

(6)

2.1.3 **Button [Q2.1.3]**

A new mission called “Karoo Voyager” was launched in 2025 by the SAASA agency to the planet Jupiter. The mission was successful.

Write SQL code to insert a new record into the tblMissions table with the following details:

- MissionName: Karoo Voyager
- LaunchYear: 2025
- Success: True
- Agency: SAASA
- PlanetID: 5

(4)

2.1.4 **Button [Q2.1.4]**

A failed mission to Neptune by NASA has now been successfully completed by SpaceX in the current year.

Write Delphi code to do the following:

- Dynamically retrieve the current year from the system date and store it in a string variable named sYear.

Write SQL code to update the mission:
- Agency field = 'SpaceX'
- Success field = TRUE
- LaunchYear field = to the value in sYear

Only update the record(s) where:

- PlanetID = 8 (Neptune)
- Agency = 'NASA'
- Success = False

Note: The year 2025 must not be hardcoded. The SQL statement must use the sYear variable so that the code will work correctly regardless of the current year.

Example of the output:

▶	25	Neptune Orbiter	2025	True	SpaceX	8

(6)

2.1.5 **Button [Q2.1.5]**

Write SQL code to display the Agency and the **total number of successful missions**. Use the alias TotalSuccess for the calculated column. Only display agencies that have completed 3 or more successful missions.

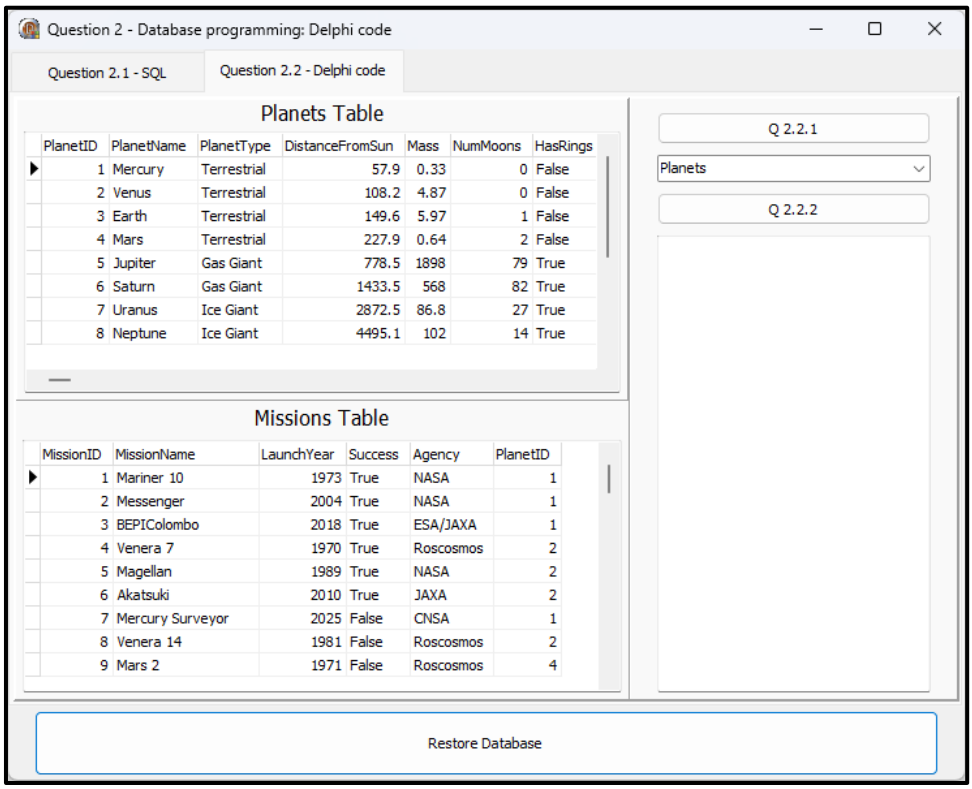
Example of the output:
Note: The following output will only appear if QUESTIONS 2.1.3 and 2.1.4 have been successfully run:

	Agency	TotalSuccess
▶	NASA	13
	SAASA	4
	SpaceX	3

(6)

2.2 Tab Sheet [Question 2.2 – Delphi Code]

Example of graphical user interface (GUI) for QUESTION 2.2.



NOTE:

- Use ONLY Delphi programming code to answer QUESTION 2.2.1 and QUESTION 2.2.2.
- NO marks will be awarded for SQL statements in QUESTION 2.2.
- Use the global variables provided: tblPlanets and tblMissions.

IMPORTANT:
Before attempting any code in this section, you must click the "Restore Database" button to reset the data to its original state. Failure to do so may result in incorrect outputs.

2.2.1 Button [Q2.2.1]

Write Delphi code to count the number of records in each of the two database tables:

- tblPlanets
- tblMissions

Display the output in the rich edit **redOutput** component, each on a new line.

Example of the output:

Solar System
Number of planets: 8
Number of missions: 56

(2)

2.2.2 **Button [Q2.2.2]**

A combo box named cmbPlanets contains the list of planet names from the tblPlanets table.

Write Delphi code that will do the following when a planet is selected and the Q2.2.2 button is clicked:

- Find the PlanetID that matches the selected planet name
- Loop through tblMissions to only show successful missions to that planet
- If no missions are found, display:

"No successful missions to [PlanetName]"

- Display a heading for the correct planet name in the rich edit **redOutput** component. The heading must be bold.
- Display the following details for each mission in the rich edit **redOutput** component, one per line:

MissionName (Agency, LaunchYear)

Example of the output for the following planets:

Mercury

Mercury
Mariner 10 (NASA, 1973)
Messenger (NASA, 2004)
BEPIColombo (ESA/JAXA, 2018)

Venus

Venus
Venera 7 (Roscosmos, 1970)
Magellan (NASA, 1989)
Akatsuki (JAXA, 2010)
Venus Express (ESA, 2005)

Saturn

Saturn
Pioneer 11 (NASA, 1973)
Cassini-Huygens (NASA/ESA, 1997)
Bosveld Beacon (SAASA, 2015)
Saturn Dreamer (JAXA, 2003)

Neptune

Neptune
Voyager 2 (Neptune) (NASA, 1977)
Neptune One (CNSA, 2030)

(10)

- Enter your name and surname as a comment in the first line of the program file.
- Save your program.
- A printout of the code may be required.

[40]

QUESTION 3: OBJECT-ORIENTED PROGRAMMING

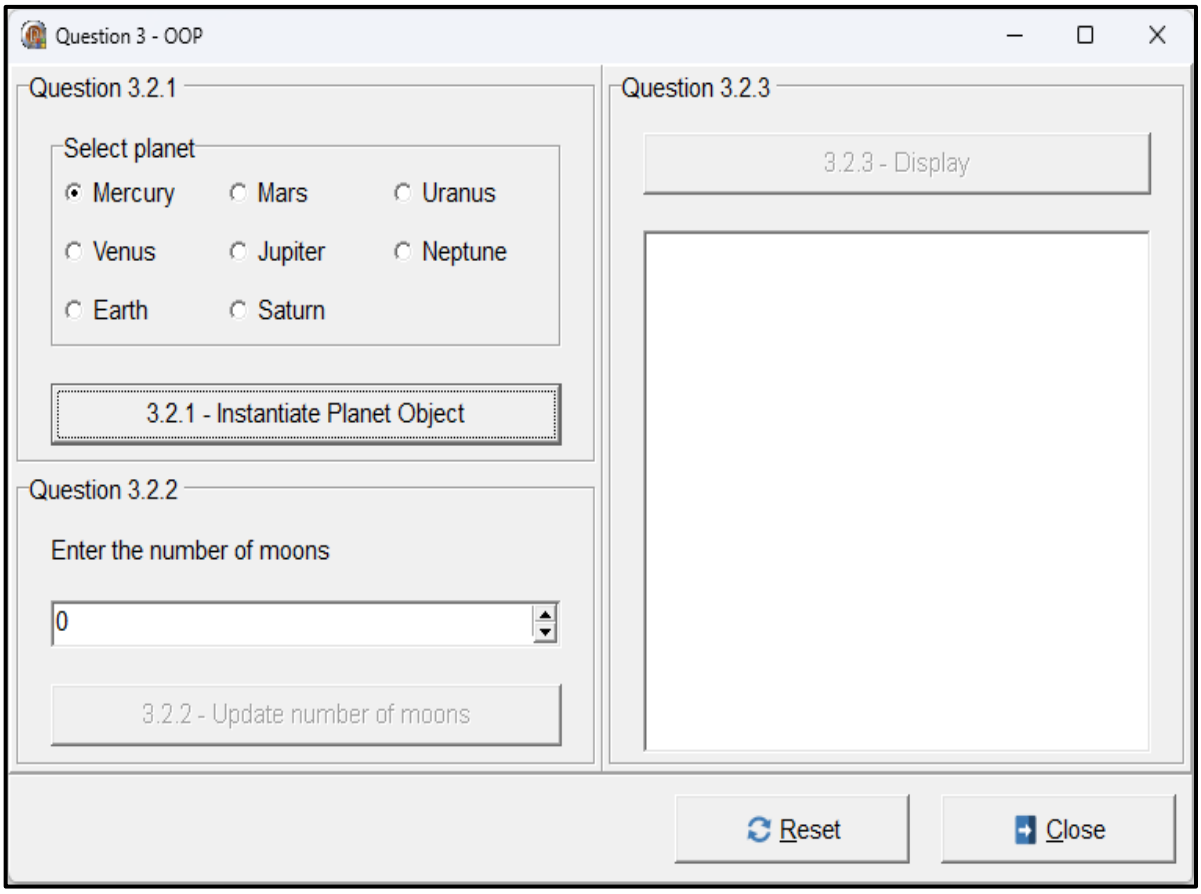
You have been tasked by the International Planetary Research Division to assist with managing and processing information about the planets in our solar system.

The organisation has provided a data file **Planets.txt** containing important planetary data. You are required to use object-oriented programming principles in Delphi to represent and manipulate this data using a class named TPlanet.

Do the following:

- Open the incomplete program in the **Question 3** folder.
- Open the incomplete object class **Planet_U.pas**.
- Enter your full name as a comment in the first line of both the **Planet_U.pas** file and the **Question3_U.pas** file.
- Compile and execute the program. The program has limited functionality currently.
- Do NOT remove or change any provided code.

Example of the graphical user interface (GUI):



Complete the code as specified in QUESTION 3.1 and QUESTION 3.2 that follow.

NOTE: You are NOT allowed to add any additional attributes or user-defined methods, unless explicitly stated in the question.

Open the incomplete object class **Planet_U.pas.**

3.1 The provided incomplete class (**TPlanet**) contains the declaration of seven attributes that describe the **objPlanet** object.

ATTRIBUTE	DESCRIPTION
fName	String value containing the planet's name
fType	String value containing the planet's type
fDiameter	Real value containing planet's diameter in km
fMass	Real value containing planet's mass in 10^{24} kg
fDistance	Real value containing the planet's distance to the sun in million km
fMoons	Integer value containing the amount of moons orbiting the planet
fRings	Boolean value indicating whether the planet has rings

Complete the code in the object class as described in QUESTION 3.1.1 to QUESTION 3.1.6 below.

3.1.1 Write code for a constructor method named **Create** that receives the following parameters:

- Planet Name
- Planet Type
- Diameter
- Mass
- Distance from the Sun
- Number of Moons
- Whether the planet has rings (Boolean)

Assign these parameter values to the corresponding class attributes. (4)

3.1.2 Write code for a method named **getDistance** that returns a string in the following format:

Distance: <TAB space> [Distance] million km from the Sun.

Round the distance to 1 decimal place. (3)

3.1.3 As scientific technology advances, new moons may be discovered orbiting the planets.

Write code for a mutator method named **setMoons** that receives an integer parameter representing the number of newly discovered moons. This value must be added to the existing fMoons attribute. (2)

3.1.4 Write code for a method named **calcDensity** to calculate and return the density of the planet using the following formula:

$$\text{Density} = \frac{\text{Mass} \times 10^{24}}{\frac{4}{3} \times \pi \times \left(\frac{\text{Diameter}}{2}\right)^3}$$

NOTE:
- The mass is given in units of 10^{24} kg.
- Assume the planet is spherical.
- Return the result as a real value. (7)

3.1.5 Write code for a method named **hasRings** that returns a Boolean value indicating whether the planet has rings. (2)

3.1.6 Write code for a **toString** method that returns a string in the following format:

[PlanetName]

Type: <TAB space> [Type]
Diameter: <TAB space> [Diameter]
Mass: <TAB space> [Mass]
Moons: <TAB space> [Moons]

See Question 3.2.3 for final output. (3)

3.2 An incomplete program has been supplied in the **Question 3** folder. The program contains code for the object class to be accessible and declares an object variable named **objPlanet.**

A text file named Planets.txt is provided, containing the following fields:

PlanetName,PlanetType,Diameter,Mass,DistanceFromSun,NumberOfMoons, HasRings

Text file contents:

```
Mercury,Terrestrial,4879,0.33,57.9,0,False
Venus,Terrestrial,12104,4.87,108.2,0,False
Earth,Terrestrial,12756,5.97,149.6,1,False
Mars,Terrestrial,6792,0.64,227.9,2,False
Jupiter,Gas Giant,142984,1898,778.5,79,True
Saturn,Gas Giant,120536,568,1433.5,82,True
Uranus,Ice Giant,51118,86.8,2872.5,27,True
Neptune,Ice Giant,49528,102,4495.1,14,True
```

Write code to perform the tasks described in QUESTION 3.2.1 to QUESTION 3.2.3 below.

3.2.1 **Button [3.2.1 – Create Planet Object]**

Write code to do the following:

- Extract the planet name from the **rgpPlanets** component.
- Test to see if the text file exists. If the text file cannot be found, display a suitable message and close the application.
- Loop through the text file to search for the correct planet.
- If the planet is not found, display a suitable message.
- If the planet is found, extract all the required values for the planet and instantiate the new Planet object.
- Display a message once the object has been created.

Message = Planet instantiated (15)

3.2.2 **Button [3.2.2 – Update Number of Moons]**

A spin edit component named **sedMoons** is provided.

Write code to add the number of moons for the current planet object using the value from sedMoons. (1)

3.2.3 **Button [3.2.3 – Display]**

Write code to display information about the selected planet using the object's methods. The output must appear in the rich edit **redOutput** component. Use #9 characters where necessary to align the output.

- Call the **toString** method and display the result.
- Call the **getDistance** method and display the result on a new line.
- Use the **hasRings** method to determine whether the planet has rings and display the message as follows:

Rings: Yes or Rings: No

- Call the **calcDensity** method and display (rounded to 2 decimal places) the result in the following format:

Density: <TAB space> [density value]

Example output:

Mercury with no added moons

```
Mercury

Type:       Terrestrial
Diameter:   4879
Mass:       0.33
Moons:      0
Distance:   57.9 million km from the Sun.
Rings:      No
Density:    5426538183344.24
```

Saturn with 10 new moons

```
Saturn

Type:       Gas Giant
Diameter:   120536
Mass:       568
Moons:      92
Distance:   1433.5 million km from the Sun.
Rings:      Yes
Density:    619440201620.72
```

Earth with no added moons

```
Earth

Type:       Terrestrial
Diameter:   12756
Mass:       5.97
Moons:      1
Distance:   149.6 million km from the Sun.
Rings:      No
Density:    5493285577295.12
```

Uranus with 50 new moons

```
Uranus

Type:       Ice Giant
Diameter:   51118
Mass:       86.8
Moons:      77
Distance:   2872.5 million km from the Sun.
Rings:      Yes
Density:    1241079328229.34
```

- Enter your name and surname as a comment in the first line of the program file.
- Save your program.
- A printout of the code may be required.

(8)

[45]

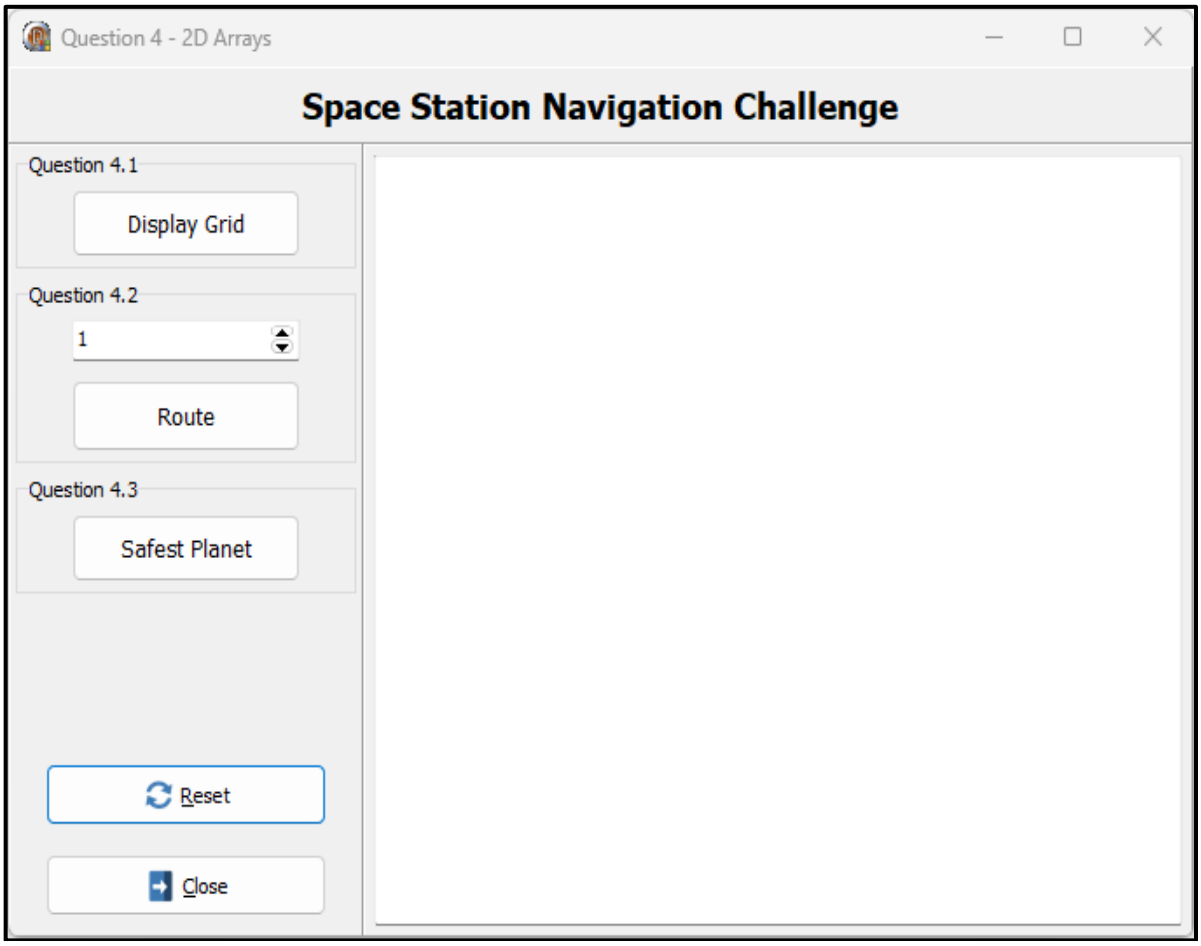
QUESTION 4: PROBLEM-SOLVING PROGRAMMING

The International Space Agency has developed a virtual navigation system for spacecraft moving between planets in the solar system.

Do the following:

- Open the incomplete program in the **Question 4** folder.
- Enter your full name as a comment in in the first line of the **Question4_u.pas** file.
- Compile and execute the program. The program has limited functionality currently.
- Do NOT remove or change any provided code.

Example of the graphical user interface (GUI):



The following have been provided in the program:

- A preloaded single dimensional array named **arrPlanets** has been instantiated for you. This array holds the planet names.
- A preloaded two-dimensional array named **ar2Solar** has been instantiated for you. This array stores the data of an 8 x 8 grid for the solar system map.

Complete the code for each section of QUESTION 4, as described in QUESTION 4.1 to QUESTION 4.3 below.

4.1 **Button [4.1 – Display Grid]**

The array **ar2Solar** is a preloaded 2D array of char, representing an 8x8 map of the solar system.

Each cell contains one of the following characters:

- A capital letter indicating a planet:
  'Y' Mercury
  'V' Venus
  'E' Earth
  'M' Mars
  'J' Jupiter
  'S' Saturn
  'U' Uranus
  'N' Neptune
- An asterisk (*) represents an asteroid
- A hyphen (-) represents an empty space

Write code to display the contents of the solar map in a neatly formatted grid in the rich edit **redOutput** component. Each row must begin with the row number, and the column numbers must appear as a header.

Example of the output:

NOTE*: Display the contents exactly as shown, including row and column labels.*

```
   1 2 3 4 5 6 7 8
1  Y - - * - - - E
2  * * - - * * - -
3  - - J - - * - -
4  - * * * - - - -
5  S - - U - N - -
6  - * - - - - * -
7  - - - * * * * -
8  V - - * * - - M
```

(8)

4.2 **Button [4.2 - Route]**

The user selects a column number using a spin edit **sedColumn** component.

Write Delphi code to determine whether the selected column from the 2D array **ar2Solar** is clear of asteroids ('*'). A column is considered clear if it contains no asterisks ('*').

Display the appropriate message in the rich edit **redOutput** component.

Example of the output:

If Column 1 was selected:

```
Route 1 is not clear.
```

If Column 8 was selected:

```
Route 8 is clear.
```

(7)

4.3 **Button [4.3 – Safest Planet]**

A planet in the grid is considered "safe" if it is far away from all asteroids. You are required to find the planet that is the furthest away from its nearest asteroid.

To calculate how far a planet is from an asteroid:

- Count how many rows apart they are. This is the absolute difference between the two row numbers. *HINT use Delphi's Abs function.*
- Count how many columns apart they are. This is the absolute difference between the two column numbers.
- Add the two values together to get the distance.

For example:

> If a planet is in Row 2, Column 5 and an asteroid is in Row 6, Column 3, then the distance is:
>
> $$|6 - 2| + |3 - 5| = 4 + 2 = 6 \text{ units}$$

Write code to:

- Loop through the 2D array **ar2Solar** to locate every planet in the grid.
- For each planet, calculate its distance to every asteroid and determine the closest asteroid to that planet.
- After checking all the planets, select the one that is furthest from its nearest asteroid.
- Display the **name of the safest planet**, its position (row and column), and the **distance to its nearest asteroid** in the rich edit **redOutput** component.

Use the provided array arrPlanets[1..8] to get the full name of the planet based on its character.

Example output:

```
The safest planet is Earth at Row 1, Column 8.
Closest asteroid is 3 units away.
```

(15)

- Enter your name and surname as a comment in the first line of the program file.
- Save your program.
- A printout of the code may be required.

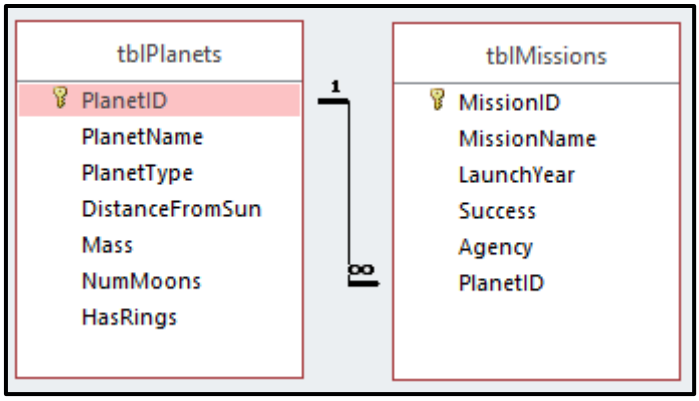
[30]

TOTAL: 150

DATABASE INFORMATION QUESTION 2:

The database **SolarDB** consists of the tables: **tblPlanets** and **tblMissions**.

The following one-to-many relationship with referential integrity exists between the two tables in the database:



The design of the database tables is as follows:

Table: **tblPlanets** – This table contains details of the various planets in our solar system.

Field Name	Data Type	Description
PlanetID	AutoNumber (PK)	Unique identifier for each planet
PlanetName	Text (20)	Name of the planet
PlanetType	Text (15)	Type of planet (e.g., Terrestrial, Gas Giant)
DistanceFromSun	Number	Distance from the Sun in million km
Mass	Number	Mass of the planet in 10^24 kg
NumMoons	Number	Number of moons the planet has
HasRings	Yes/No	Indicates if the planet has rings

Example of the records in the **tblPlanets** table:

PlanetID	PlanetName	PlanetType	DistanceFromSun	Mass	NumMoons	HasRings
1	Mercury	Terrestrial	57.9	0.33	0	☐
2	Venus	Terrestrial	108.2	4.87	0	☐
3	Earth	Terrestrial	149.6	5.97	1	☐
4	Mars	Terrestrial	227.9	0.64	2	☐
5	Jupiter	Gas Giant	778.5	1898	79	☑
6	Saturn	Gas Giant	1433.5	568	82	☑
7	Uranus	Ice Giant	2872.5	86.8	27	☑
8	Neptune	Ice Giant	4495.1	102	14	☑

Table: **tblMissions** – This table contains details of the various missions to the different planets.

Field Name	Data Type	Description
MissionID	AutoNumber (PK)	Unique identifier for each mission
MissionName	Text (30)	Name of the mission
LaunchYear	Number	The year the mission was launched
Success	Yes/No	Indicates if the mission was successful
Agency	Text (20)	The organisation responsible for the mission
PlanetID	Number (FK)	Foreign key linked to tblPlanets.PlanetID

Example of the first 40 records in the **tblMissions** table:

MissionID	MissionName	LaunchYear	Success	Agency	PlanetID
1	Mariner 10	1973	☑	NASA	1
2	Messenger	2004	☑	NASA	1
3	BEPIColombo	2018	☑	ESA/JAXA	1
4	Venera 7	1970	☑	Roscosmos	2
5	Magellan	1989	☑	NASA	2
6	Akatsuki	2010	☑	JAXA	2
7	Mercury Surveyor	2025	☐	CNSA	1
8	Venera 14	1981	☐	Roscosmos	2
9	Mars 2	1971	☐	Roscosmos	4
10	Viking 1	1975	☑	NASA	4
11	Viking 2	1975	☑	NASA	4
12	Curiosity	2011	☑	NASA	4
13	Perseverance	2020	☑	NASA	4
14	Mars Observer	1992	☐	NASA	4
15	Tianwen-1	2020	☑	CNSA	4
16	RedDawn	2025	☑	SpaceX	4
17	Mangalyaan	2013	☑	ISRO	4
18	JUICE	2023	☑	ESA	5
19	Galileo	1989	☑	NASA	5
20	New Horizons	2006	☑	NASA	5
21	Jupiter Vision	2026	☐	CNSA	5
22	Pioneer 11	1973	☑	NASA	6
23	Cassini-Huygens	1997	☑	NASA/ESA	6
24	Saturn Probe X	2025	☐	SpaceX	6
25	Neptune Orbiter	1999	☐	NASA	8
26	Voyager 2 (Neptune)	1977	☑	NASA	8
27	Voyager 2 (Uranus)	1977	☑	NASA	7
28	Uranus Explorer	2025	☐	NASA	7
29	Pluto Express	2024	☐	NASA	5
30	Mercury Express	2023	☐	ISRO	1
31	Venus Express	2005	☑	ESA	2
32	Zhurong 2	2026	☐	CNSA	4
33	Shenzhou Venus	2027	☐	CNSA	2
34	Asteroid Probe Z	2022	☐	JAXA	5
35	GalacticEye	2030	☑	SAASA	4
36	SA-Voyager	2031	☐	SAASA	5
37	Bosveld Beacon	2015	☑	SAASA	6
38	Athena 1	2033	☐	SAASA	3
39	SolarScope I	2029	☑	SAASA	3
40	Soviet Jupiter 1	1960	☐	Roscosmos	5

PLANNING PAGE 1