Database Implementation

Relational Databases Assignment 2

Zalán Tóth - 20102768
Computer Forensics and Security Year 2

SETU Waterford
School of Science and Computing
Department of Computing and Mathematics

Waterford City, Ireland
03/12/2023

Table of Contents

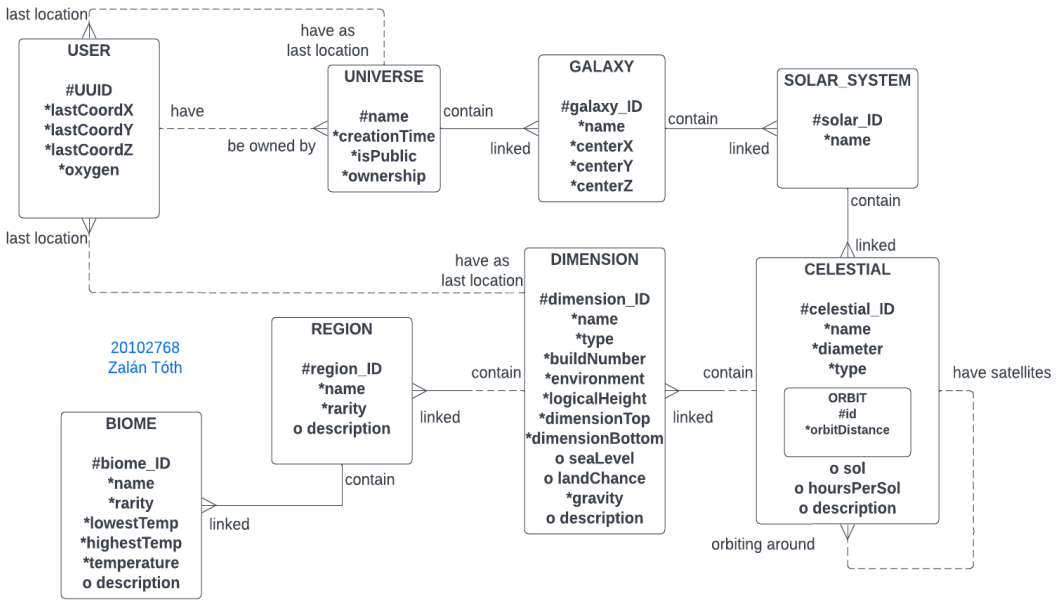
Changes since the Design Document are marked with dark blue in the document.
Removed parts marked as red.

Overall description

A small game developer studio is creating a space exploration game. We are creating a database structure for the planetary positioning design. There are logical parts of the design, that help to locate inside the universes, and there are game-mechanic elements, which connect the user to the game.
On the logical part, we have universes, galaxies, solar systems, and celestials. These elements store location data, so based on the provided data, we must be able to create a map of each universe.
On the game game-mechanic part we have dimensions. A player will always be in a dimension except when choosing a planet. When a player leaves a celestial (on the logical part the player is on a celestial planet, but actually on the game-mechanic part, its location is a dimension), a planetary map will be shown, so the player can choose a destination, and then enter the celestial's atmosphere (In the background all that happens, is that the user changes dimensions). A celestial might have more dimensions. For example, the celestial called Earth might have dimensions called Earth_Site_A, Earth_Site_B, and Earth_Federal_City. Inside a dimension we have regions, and inside regions we have biomes.

Entity-Relationship Diagram

Image format



LucidChart link

https://lucid.app/lucidchart/290a8c00-32f3-40d4-ab2b-70577f25ac3b/edit?invitationId=inv_e6a53526-49f1-44d2-af3c-1cddab164181

Detailed description

User

For the user, we must store information, so we can locate where was the player on the last logout. We need to store the last universe and the dimension and the player's coordinates inside that dimension. We are identifying users based on their UUIDs (a unique string). We also store oxygen value for the player as an integer.

Universe

As we can have multiple universes (like parallel universes) we need to store the name of the universe, which has to be unique, as that's the data we use to differentiate between universes. We also must store the creation time in stardate, for that we need a long integer. We also need ownership over the universe itself. It might be owned by the server or a player (we need an attribute for the ownership type and if it’s a player, we need to associate the universe with the user’s UUID). We also need to store a boolean value if the universe is public to all players, or private (whitelisted). A universe has to contain at least 1 galaxy.

Galaxy

A galaxy must be linked to one universe. We are identifying galaxies based on their unique galaxy ID. We also have to store a name for it. To be able to put the galaxy on the map, we need to store its center with X, Y, and Z coordinates. A galaxy must contain at least 1 solar system.

Solar System

A solar system must be linked to one galaxy. We are identifying solar systems based on their unique solar ID. We are storing the solar system's star's ID (which is the center celestial’s celestial ID). We must also store the name of the solar system (like Solar System). A solar system must contain at least 1 celestial.

Celestial

A celestial body is something in space that's orbiting around something else. In this case, a celestial must be linked to one solar system. We are identifying celestials based on their unique celestial ID. We are storing the celestial’s display name. We can have many types of celestials like stars, planets, moons, and satellites. To properly display them on the galactic map, we need to store the diameter of the celestial in kilometers. We need information about what is it orbiting around. For that, we store another celestial's ID or the ID of the galaxy if it's a star. We need to store the average orbit distance in light years. So if a planet is 5 light years away from its orbit (which would be a star in this scenario) then the orbit distance is 5. We also need to allocate space for optional data like sols (solar days), earth hours per sol, and celestial description. Keep in mind that celestials might be orbiting around another celestial, which might be orbiting around another celestial etc., like the Moon around Earth, the Earth around the Sun, and the Sun around the Milky Way's center (black hole).
A celestial might have one or more dimensions depending on its scenario.
Orbits are stored as subtypes.

Dimension

A dimension is a game-mechanic element. It must be linked to one celestial. We are identifying the dimension by its unique dimension ID. We also store a display name for it. We need to store various other values for a dimension. First of all its type. It can be a planet's surface or even a space station. We store this value in a string. We also need to store the environment type, which can be NORMAL or SPACE. We also need to store the sea level in meters if the dimension has one, if not, then the value should be NULL. We need to specify 3 different values in meters for logical height (where should be the logical limit of the dimension, the beginning of space), dimension top (what should be the highest point for the world generator), and dimension bottom (what should be the lowest point for the world generator, this is also the logical bottom). We must store the gravity in m/s^2. If the dimension is a planetary surface we also include a land chance value. It has to be a float between 0 and 1 inclusive or NULL. 1 for a dimension without seas, and 0 for a dimension with only seas. So 0.5 would give 50%-50% for land-sea share. NULL if it's not a planetary surface, like a space station. We need to provide space for an optional description of the dimension. A dimension may contain regions.

Region

A region must be linked to a dimension. We identify a region based on its unique region ID. We need a display name for the region as well. We must store the rarity in an integer. We also wanna allocate space for optional descriptions. A region must contain at least 1 biome.

Biome

A biome must be linked to a region. We identify a biome by its unique biome ID. We need a display name for the biome as well. We must store the rarity as an integer. We need to store temperature data. One for the current temperature in the biome, one for the lowest possible temperature in the biome, and one for the highest possible temperature. All three should be stored in Kelvin (so we need to store only positive values). We also wanna allocate space for an optional description.

Rules

Many rules can be found in the detailed description.

Structural rules

1. Universes must-have attributes for ownership type and the associated user's UUID if owned by a player.
2. A Celestial can have multiple Dimensions associated with it.
3. A Biome must be linked to a Region, which must be linked to a Dimension, which must be linked to a Celestial, which must be linked to a Solar System, which must be linked to a Galaxy, which must be linked to a Universe.
4. Celestials can be of type stars, planets, moons, or satellites.

Procedural rules

1. When a user logs out, the system must store their last Universe, Dimension, and coordinates within that dimension.
2. Each Universe, Galaxy, Solar System, Celestial, Dimension, Region, and Biome must have a unique ID and a display name.
3. If the Celestial's type is "star", the orbit value has to be the ID of the Galaxy, otherwise the value should be another Celestial's ID.
4. A Dimension's sea level and land chance attribute must be NULL if its type is not a planetary surface.

Programmatic rules

1. Every space- and dimension-related ID has to be unique and stored in a string. We must be able to differentiate between IDs in the sense, that we're capable of telling the type of the ID, just by analyzing it. For example, just by looking at an ID, we must be capable of telling, if that ID belongs to a Celestial, a Galaxy, or a Dimension and which Universe is this entity located in.
   We can differentiate between the IDs if we combine them with the foreign key.
2. The system should ensure that all temperature values are positive since they are stored in Kelvin.
3. We must make sure that the necessary linked elements exist (as shown in structural rule 3) especially when we are creating and deleting an entity.

Normalisation

Check Table Mapping for the result.

1NF - First Normal Form

No multiple values are stored. Primary keys included.

2NF - Second Normal Form

Every non-key attribute is dependent on the key.

3NF - Third Normal Form

Checked for transitive dependency.

Table Mapping

Users

FIELD	TYPE	SIZE	NULL?	DEFAULT	CONSTRAINTS	DESCRIPTION
UUID	VARCHAR	36	NOT NULL		UNIQUE, PK	Unique identifier
LastCoordX	DOUBLE		NOT NULL			Dimension X coordinate
LastCoordY	DOUBLE		NOT NULL			Dimension Y coordinate
LastCoordZ	DOUBLE		NOT NULL			Dimension Z coordinate
Oxygen	SMALLINT		NOT NULL	0		Oxygen level
LastUniverse	VARCHAR	255			FK	Last recorded universe
LastDimension	VARCHAR	255			FK	Last recorded dimension

Universes

FIELD	TYPE	SIZE	NULL?	DEFAULT	CONSTRAINTS	DESCRIPTION
Name	VARCHAR	255	NOT NULL		UNIQUE, PK	Unique identifier
CreationTime	BIGINT		NOT NULL	0		Creation time in Stardate format
isPublic	BOOLEAN		NOT NULL	FALSE		Is the universe accessible to every player?
OwnershipType	VARCHAR	16	NOT NULL	SERVER		Type of Ownership
OwnerUUID	VARCHAR	36				Owner UUID if Ownership type is player

Galaxies

FIELD	TYPE	SIZE	NULL?	DEFAULT	CONSTRAINTS	DESCRIPTION
Galaxy_ID	INT		NOT NULL	AUTO_INCREMENT	UNIQUE, PK	Unique identifier
DisplayName	VARCHAR	255	NOT NULL		UNIQUE	Display name for the Galaxy
CenterX	DOUBLE		NOT NULL			Coordinate X for the Galaxy in the Universe
CenterY	DOUBLE		NOT NULL			Coordinate Y for the Galaxy in the Universe
CenterZ	DOUBLE		NOT NULL			Coordinate Z for the Galaxy in the Universe
UniverseName	VARCHAR	255	NOT NULL		FK	The ID of the Universe where Galaxy is located

SolarSystems

FIELD	TYPE	SIZE	NULL?	DEFAULT	CONSTRAINTS	DESCRIPTION
Solar_ID	INT		NOT NULL	AUTO_INCREMENT	UNIQUE, PK	Unique identifier
DisplayName	VARCHAR	255	NOT NULL		UNIQUE	Display name for the Solar System
Galaxy_ID	INT		NOT NULL		FK	The ID of the Galaxy where the Solar System is located

Celestials

FIELD	TYPE	SIZE	NULL?	DEFAULT	CONSTRAINTS	DESCRIPTION
Celestial_ID	INT		NOT NULL	AUTO_INCREMENT	UNIQUE, PK	Unique identifier
DisplayName	VARCHAR	255	NOT NULL		UNIQUE	Display name for the Celestial
Diameter	DOUBLE					Diameter of the Celestial in kilometres
Type	VARCHAR	64	NOT NULL			Type of the Celestial (ex.: Planet)
Sol	SMALLINT					Amount of sols (days) each year
HoursPerSol	DOUBLE					Amount of hours per sol
Description	TEXT					Description of the Celestial
Solar_ID	INT		NOT NULL		FK	The ID of the Solar System where the Celestial is located

Orbits

FIELD	TYPE	SIZE	NULL?	DEFAULT	CONSTRAINTS	DESCRIPTION
Celestial_ID	INT		NOT NULL	AUTO_INCREMENT	UNIQUE, PK	Unique identifier tied to the Celestial ID in Celestials
OrbitDistance	DOUBLE	32, 16	NOT NULL			Distance in light-years from the Orbit
CelestialOrbit	INT				FK	The ID of the Celestial which this Celestial is orbiting around. If NULL, then it is orbiting around the center of the Galaxy.

Dimensions

FIELD	TYPE	SIZE	NULL?	DEFAULT	CONSTRAINTS	DESCRIPTION
Dimension_ID	INT		NOT NULL	AUTO_INCREMENT	UNIQUE, PK	Unique identifier
DisplayName	VARCHAR	255	NOT NULL		UNIQUE	Display name for the Dimension
Type	VARCHAR	255	NOT NULL			Type of the dimension
Environment	VARCHAR	16	NOT NULL	NORMAL		Environment type
BuildNumber	INT		NOT NULL			Versioning
LogicalHeight	DOUBLE		NOT NULL			Logical height for the dimension (in metres)
DimensionTop	DOUBLE		NOT NULL			Generator top for the dimension (in metres)
DimensionBottom	DOUBLE		NOT NULL			Generator bottom for the dimension (in metres)
SeaLevel	DOUBLE					Sea level in metres
Gravity	DOUBLE		NOT NULL			Gravity value (m/s^2)
LandChance	DOUBLE	8,7				Chance for land
Description	TEXT					Description of the dimension
Celestial_ID	INT		NOT NULL		FK	The ID of the Celestial where the Dimension is located

Regions

FIELD	TYPE	SIZE	NULL?	DEFAULT	CONSTRAINTS	DESCRIPTION
Region_ID	INT		NOT NULL	AUTO_INCREMENT	UNIQUE, PK	Unique identifier
DisplayName	VARCHAR	255	NOT NULL		UNIQUE	Display name for the Region
Rarity	INT		NOT NULL			Rarity
Description	TEXT		NOT NULL			Description of the region
Dimension_ID	INT		NOT NULL		FK	The ID of the Dimension where the Region is located

Biomes

FIELD	TYPE	SIZE	NULL?	DEFAULT	CONSTRAINTS	DESCRIPTION
Biome_ID	INT		NOT NULL	AUTO_INCREMENT	UNIQUE, PK	Unique identifier
DisplayName	VARCHAR	255	NOT NULL		UNIQUE	Display name for the Biome
Rarity	INT		NOT NULL			Rarity
LowestTemp	DOUBLE		NOT NULL			Lowest possible temperature in the biome in Kelvin
HighestTemp	DOUBLE		NOT NULL			Highest possible temperature in the biome in Kelvin
Temperature	DOUBLE		NOT NULL			Current temperature in the biome in Kelvin
Description	TEXT		NOT NULL			Description of the region
Region_ID	INT		NOT NULL		FK	The ID of the Region where the Biome is located

SQL statements

DDL - Data Definition

Creation

```sql
CREATE DATABASE Space;
USE Space;

CREATE TABLE Universes (
    Name VARCHAR(255) UNIQUE NOT NULL PRIMARY KEY,
    CreationTime BIGINT NOT NULL DEFAULT 0,
    IsPublic BOOLEAN NOT NULL DEFAULT FALSE,
    OwnershipType VARCHAR(16) NOT NULL DEFAULT 'SERVER',
    OwnerUUID VARCHAR(36)
    -- OwnerUUID foreign key will be added after Users table creation
);

CREATE TABLE Galaxies (
    Galaxy_ID INT UNIQUE NOT NULL AUTO_INCREMENT PRIMARY KEY,
    DisplayName VARCHAR(255) UNIQUE NOT NULL,
    CenterX DOUBLE NOT NULL,
    CenterY DOUBLE NOT NULL,
    CenterZ DOUBLE NOT NULL,
    UniverseName VARCHAR(255) NOT NULL,
    FOREIGN KEY (UniverseName) REFERENCES Universes(Name)
);

CREATE TABLE SolarSystems (
    Solar_ID INT UNIQUE NOT NULL AUTO_INCREMENT PRIMARY KEY,
    DisplayName VARCHAR(255) UNIQUE NOT NULL,
    Galaxy_ID INT NOT NULL,
    FOREIGN KEY (Galaxy_ID) REFERENCES Galaxies(Galaxy_ID)
);

CREATE TABLE Celestials (
    Celestial_ID INT UNIQUE NOT NULL AUTO_INCREMENT PRIMARY KEY,
    DisplayName VARCHAR(N255) UNIQUE NOT NULL,
    Diameter DOUBLE,
    Type VARCHAR(64) NOT NULL,
    Sol SMALLINT,
    HoursPerSol DOUBLE,
    Description TEXT,
```

```
    Solar_ID INT NOT NULL,
    FOREIGN KEY (Solar_ID) REFERENCES SolarSystems(Solar_ID)
);

CREATE TABLE Orbits (
    Celestial_ID INT UNIQUE NOT NULL AUTO_INCREMENT PRIMARY KEY
REFERENCES Celestials(Celestial_ID),
    OrbitDistance DOUBLE NOT NULL,
    CelestialOrbit INT,
    FOREIGN KEY (CelestialOrbit) REFERENCES Celestials(Celestial_ID)
);

CREATE TABLE Dimensions (
    Dimension_ID INT UNIQUE NOT NULL AUTO_INCREMENT PRIMARY KEY,
    DisplayName VARCHAR(255) UNIQUE NOT NULL,
    Type VARCHAR(255) NOT NULL,
    Environment VARCHAR(16) NOT NULL DEFAULT 'NORMAL',
    BuildNumber INT NOT NULL,
    LogicalHeight DOUBLE NOT NULL,
    DimensionTop DOUBLE NOT NULL,
    DimensionBottom DOUBLE NOT NULL,
    SeaLevel DOUBLE,
    Gravity DOUBLE NOT NULL,
    LandChance DOUBLE(8,7),
    Description TEXT,
    Celestial_ID INT NOT NULL,
    FOREIGN KEY (Celestial_ID) REFERENCES Celestials(Celestial_ID)
);

CREATE TABLE Regions (
    Region_ID INT UNIQUE NOT NULL AUTO_INCREMENT PRIMARY KEY,
    DisplayName VARCHAR(255) UNIQUE NOT NULL,
    Rarity INT NOT NULL,
    Description TEXT,
    Dimension_ID INT NOT NULL,
    FOREIGN KEY (Dimension_ID) REFERENCES Dimensions(Dimension_ID)
);

CREATE TABLE Biomes (
    Biome_ID INT UNIQUE NOT NULL AUTO_INCREMENT PRIMARY KEY,
    DisplayName VARCHAR(255) UNIQUE NOT NULL,
    Rarity INT NOT NULL,
    LowestTemp DOUBLE NOT NULL,
```

```
    HighestTemp DOUBLE NOT NULL,
    Temperature DOUBLE NOT NULL,
    Description TEXT,
    Region_ID INT NOT NULL,
    FOREIGN KEY (Region_ID) REFERENCES Regions(Region_ID)
);

CREATE TABLE Users (
    UUID VARCHAR(36) UNIQUE NOT NULL PRIMARY KEY,
    LastCoordX DOUBLE NOT NULL,
    LastCoordY DOUBLE NOT NULL,
    LastCoordZ DOUBLE NOT NULL,
    Oxygen SMALLINT NOT NULL DEFAULT 0,
    LastUniverse VARCHAR(255),
    LastDimension INT,
    FOREIGN KEY (LastUniverse) REFERENCES Universes(Name),
    FOREIGN KEY (LastDimension) REFERENCES Dimensions(Dimension_ID)
);

-- Add foreign key to Universes (OwnerUUID)
ALTER TABLE Universes
ADD FOREIGN KEY (OwnerUUID) REFERENCES Users(UUID);
```

DML - Data Manipulation

Inserts

```
USE Space;

INSERT INTO Users (UUID, LastCoordX, LastCoordY, LastCoordZ, Oxygen,
LastUniverse, LastDimension) VALUES
('22158454-6f80-46ff-9eaf-2b26fad71700', 10.0, 20.0, 30.0, 100, NULL,
NULL),
('0fb0a9fd-6202-446e-a51f-6be1046b4d77', 40.0, 50.0, 60.0, 80, NULL,
NULL),
('74ebdbd4-70a3-4ac5-a01e-8564b658cdc1', 45.0, 0.0, 60.0, 260, NULL,
NULL),
('c96792ac-7aea-4f16-975e-535a20a2791a', 70.0, 0.0, 60.0, 300, NULL,
NULL),
('f7c77d99-9f15-4a66-a87d-c4a51ef30d19', 40.0, 0.0, 60.0, 115, NULL,
NULL),
('29874407-b60b-44d8-8440-4e6b5a45a1a2', 70.0, 80.0, 90.0, 60, NULL,
NULL);

INSERT INTO Universes (Name, CreationTime, IsPublic, OwnershipType,
OwnerUUID) VALUES
('m44', 0, TRUE, 'SERVER', NULL),
('p88', 5834262234632, FALSE, 'PLAYER',
'22158454-6f80-46ff-9eaf-2b26fad71700');


INSERT INTO Galaxies (Galaxy_ID, DisplayName, CenterX, CenterY,
CenterZ, UniverseName) VALUES
(DEFAULT, 'Milky Way', 0.0, 0.0, 0.0, 'm44'),
(DEFAULT, 'Arcadia', 12514.0, 507.0, 12514.0, 'p88');

INSERT INTO SolarSystems (Solar_ID, DisplayName, Galaxy_ID) VALUES
(DEFAULT, 'Solar System', 1),
(DEFAULT, 'Elementia', 2);

INSERT INTO Celestials (Celestial_ID, DisplayName, Diameter, Type, Sol,
HoursPerSol, Description, Solar_ID) VALUES
(DEFAULT, 'Sun', 4262, 'Star', NULL, NULL, 'Center of the Solar
System', 1),
(DEFAULT, 'Mercury', 4262, 'Planet', 150, 7, 'A planet', 1),
(DEFAULT, 'Venus', 236236, 'Planet', 200, 15, 'A planet', 1),
```

```
(DEFAULT, 'Earth', 64363, 'Planet', 365, 24, 'The home of the
Humanity', 1),
(DEFAULT, 'Mars', 32626, 'Planet', 400, 24.6, 'Red planet', 1),
(DEFAULT, 'Jupiter', 32626323236, 'Planet', 600, 50, 'Gas planet', 1),
(DEFAULT, 'Uranus', 3626236, 'Planet', 1200, 30, 'A planet', 1),
(DEFAULT, 'Saturnus', 2356623, 'Planet', 1800, 66, 'A planet', 1),
(DEFAULT, 'Neptunus', 125125, 'Planet', 2256, 110, 'A planet', 1),
(DEFAULT, 'Moon', 12445, 'Moon', NULL, NULL, 'Moon of Earth', 1),
(DEFAULT, 'Europa', 2143, 'Moon', NULL, NULL, 'Moon of Jupiter', 1),
(DEFAULT, 'International Space Station', NULL, 'Satellite', NULL, NULL,
'Space Station of Humanity', 1),
(DEFAULT, 'Harmonia', 24141252156.56, 'Star', NULL, NULL, 'Center of
Elementia', 2);

INSERT INTO Orbits (Celestial_ID, OrbitDistance, CelestialOrbit) VALUES
(1, 40, NULL), -- Sun
(2, 0.000000587, 1), -- Mercury
(3, 0.000005587, 1), -- Venus
(4, 0.000015587, 1), -- Earth
(5, 0.00002345, 1), -- Mars
(6, 0.000015587, 1), -- Jupiter
(7, 0.000045587, 1), -- Uranus
(8, 0.000065587, 1), -- Saturnus
(9, 0.000115587, 1), -- Neptunus
(10, 0.0000000052, 4), -- Moon
(11, 0.0000000142, 6), -- Europa
(12, 0.0000000012, 4), -- International Space Station
(13, 56, NULL); -- Harmonia

INSERT INTO Dimensions (Dimension_ID, DisplayName, Type, Environment,
BuildNumber, LogicalHeight, DimensionTop, DimensionBottom, SeaLevel,
Gravity, LandChance, Description, Celestial_ID) VALUES
(DEFAULT, 'Surface of Earth', 'SURFACE', 'NORMAL', 1001, 256.0, 128.0,
-128.0, 62.0, 9.8, 0.5, 'Earth surface', 4),
(DEFAULT, 'Surface of the Moon', 'SURFACE', 'NORMAL', 343, 256.0,
128.0, -128.0, 62.0, 9.8, 0.5, 'Moon surface', 10),
(DEFAULT, 'Space Station for Earth', 'SATELLITE', 'NORMAL', 51, 128.0,
64.0, -64.0, NULL, 1.6, NULL, 'Space Station interior', 1);

INSERT INTO Regions (Region_ID, DisplayName, Rarity, Description,
Dimension_ID) VALUES
(DEFAULT, 'Cheesy', 5, 'Very cheesy', 2),
(DEFAULT, 'Dry', 1, 'A vast dry region', 1),
```

```
(DEFAULT, 'Temperate', 2, 'A vast temperate region', 1);

INSERT INTO Biomes (Biome_ID, DisplayName, Rarity, LowestTemp,
HighestTemp, Temperature, Description, Region_ID) VALUES
(DEFAULT, 'Valleys', 4, 250.0, 300.0, 270.0, 'Cheesy valleys', 1),
(DEFAULT, 'Craters', 2, 250.0, 300.0, 279.0, 'Cheesy craters', 1),
(DEFAULT, 'Desert Oasis', 3, 293.15, 318.15, 303.15, 'A rare oasis in
the midst of a vast desert', 2),
(DEFAULT, 'Dune Sea', 5, 303.15, 323.15, 313.15, NULL, 2),
(DEFAULT, 'Canyon Maze', 4, 288.15, 308.15, 298.15, NULL, 2),
(DEFAULT, 'Dust Plains', 3, 298.15, 318.15, 308.15, NULL, 2),
(DEFAULT, 'Rocky Badlands', 2, 283.15, 313.15, 293.15, NULL, 2),
(DEFAULT, 'Red Desert', 4, 308.15, 328.15, 318.15, 'A red-colored
desert with extreme temperatures', 2),
(DEFAULT, 'Green Forest', 3, 283.15, 298.15, 291.15, 'Lush, green
forests with diverse wildlife', 3),
(DEFAULT, 'Meadow Fields', 2, 281.15, 295.15, 288.15, 'Meadows with
colorful flowers', 3),
(DEFAULT, 'Blue Lakes', 4, 278.15, 293.15, 285.15, 'Clear blue lakes
surrounded by nature', 3),
(DEFAULT, 'Misty Valley', 5, 280.15, 291.15, 286.15, 'A valley', 3),
(DEFAULT, 'Fruit Orchards', 3, 282.15, 297.15, 289.15, 'Orchards rich
with various fruits', 3),
(DEFAULT, 'Sunflower Plains', 2, 283.15, 298.15, 290.15, 'Vast plains
with tall sunflowers', 3);
```

Basic Selects

```
SELECT * FROM Users;
SELECT * FROM Universes;
SELECT * FROM Galaxies;
SELECT * FROM SolarSystems;
SELECT * FROM Celestials;
SELECT * FROM Orbits;
SELECT * FROM Dimensions;
SELECT * FROM Regions;
SELECT * FROM Biomes;
```

Advanced Selects

Arithmetic, Aliases, Where

```
-- Average oxygen level among users
SELECT
    AVG(Oxygen) AS AverageOxygenLevel
FROM
    Users;

-- Maximum oxygen level among users
SELECT
    MAX(Oxygen) AS MaximumOxygenLevel
FROM
    Users;

-- Minimum oxygen level among users
SELECT
    MIN(Oxygen) AS MinimumOxygenLevel
FROM
    Users;

-- Counting number of planets
SELECT
    COUNT(*) AS NumberOfPlanets
FROM
    Celestials
WHERE
    Type = 'Planet';
```

Joins

```
-- Getting the orbit for Stars which is the Galaxy
SELECT
    c.Celestial_ID,
    c.DisplayName AS CelestialName,
    g.Galaxy_ID AS OrbitGalaxyID,
    g.DisplayName AS OrbitGalaxyName
FROM
    Celestials AS c
JOIN
    SolarSystems AS ss ON c.Solar_ID = ss.Solar_ID
JOIN
    Galaxies AS g ON ss.Galaxy_ID = g.Galaxy_ID
```

```
LEFT JOIN
    Orbits AS o ON c.Celestial_ID = o.Celestial_ID
WHERE
    o.CelestialOrbit IS NULL;

-- Getting the Celestial and its corresponding SolarSystem and Galaxy
SELECT
    c.DisplayName AS Celestial,
    ss.DisplayName AS SolarSystem,
    g.DisplayName AS Galaxy
FROM
    Celestials c
JOIN
    SolarSystems ss ON c.Solar_ID = ss.Solar_ID
JOIN
    Galaxies g ON ss.Galaxy_ID = g.Galaxy_ID;

-- Connecting Biomes with their corresponding Region
SELECT
    b.Biome_ID,
    b.DisplayName AS BiomeName,
    r.Region_ID,
    r.DisplayName AS RegionName
FROM
    Biomes b
JOIN
    Regions r ON b.Region_ID = r.Region_ID;
```

Group functions

```
-- Ordering Biomes by their displayname
SELECT
    *
FROM
    Biomes
GROUP BY
    DisplayName;


-- Ordering Biomes by temperature (Shown in Celsius instead of Kelvin)
SELECT
    Biome_ID,
    DisplayName,
```

```
    Temperature-272.15 AS TemperatureInCelsuis
FROM
    Biomes
GROUP BY
    TemperatureInCelsuis;

-- Getting moons and satellites and ordering them by diameter in
descending order.
SELECT
    Celestial_ID,
    DisplayName,
    Diameter,
    Type
FROM
    Celestials
WHERE
    Type IN ('Moon', 'Satellite')
ORDER BY
    Diameter DESC;
```