Designing a High-Performance SQL Databases for Efficient Behavioral Health Data Analysis

Project Overview

Project Name: Raise and Grow Database Management System

Project Manager: Emmanuel Atangana Seme

Date: March 2021

Project Outlines

1. Introduction

Raise and Grow (RG) is dedicated to supporting adults with developmental disabilities and behavioral challenges by providing high-quality residential care and tailored programs. However, managing client data, staff schedules, program performance, and compliance reporting has become ever more complex with existing systems. This project aims to design and implement a centralized, secure, and scalable database solution. By restructuring data processes and enhancing operational efficiency, this database will empower RG;s management processes to deliver better care, track progress in real-time, and meet compliance requirements with ease.

1.1 Purpose

The purpose of this project is to develop and implement a robust database system for Raise and Grow (RG), a structure in charge of adults with developmental disabilities and behavioral challenges, The database aims to improve the management of behavioral health services. The database will centralize client, staff, and program data, streamline operations, and provide actionable insights to support decision-making. This will enable RG to enhance care quality, reduce inefficiencies, and ensure compliance with regulatory standards.

1.2 Problem Statement

Raise and Grow currently faces challenges with decentralized data management and manual processes, which lead to inefficiencies, errors, and limited insights. Indeed, tracking client progress, managing staff schedules, and monitoring program outcomes are time-consuming and prone to inaccuracies. In addition, preparing compliance reports requires significant effort, diverting valuable resources from main services. Without an integrated database, RG struggles to address these challenges effectively, struggles that impact the quality of care and managerial efficiency.

1.3 Expected Outcome

The success of this project will be measured by its ability to rationalize operations and improve service delivery at RG. Key performance indicators (KPIs) include a significant reduction in time spent on manual data entry and reporting, demonstrated by a 30% improvement in operational efficiency within the first six months. Success will also be evident through enhanced care quality, tracked via a reduction in behavioral incidents and improved client outcomes, as well as timely and accurate compliance reporting with “zero” missed deadlines. Additionally, user’s satisfaction among staff and stakeholders will be monitored, with an expected 85% reporting improved accessibility and usability of the system. Scalability and data security will be validated by the database’s ability to handle increased workloads and maintain data integrity without breaches or downtime.

1.4 Sample Key Questions that can be Addressed by the Database System

As part of the implementation of a robust database for Raise and Grow, the system is designed to solve several critical questions that currently challenge operational efficiency and service delivery. The database will provide centralized data access and real-time insights and enable the organization to answer pressing operational, compliance, and care-related questions:

(a) Type 1 key questions: Client Progress and Behavioral Health

- What are the trends (development) in behavioral health for specific clients, and how effective are their current intervention plans?
- Are there recurring patterns in incidents, and how can these be mitigated?

(b) Type 2 key questions: Program Performance and Effectiveness

- Which programs are delivering the most significant positive outcomes for clients?
- What adjustments are needed in existing programs to enhance their impact?

(c) Type 3 key questions: Staff Management and Scheduling

- Are staff certifications and training up to date to meet program requirements?
- How effectively are staff members matched with clients to maximize compatibility and care quality?

(d) Type 4 key questions: Compliance and Reporting

- Are all compliance reports prepared and submitted on time, with the necessary supporting data?

- What insights can be drawn from compliance data to inform strategic decisions?

(e) Type 5 key questions: Resource Optimization

- Are resources such as staff hours and funding being used efficiently across programs?
- What improvements can be made to balance workloads and ensure consistent coverage?

(f) Type 6 key questions: Client Placement and Compatibility

- How successful have recent client placements been in terms of reduced behavioral incidents and improved client satisfaction?
- Which available placements are best suited to meet the needs of incoming clients?

(g) Type 7 key questions: Operational Efficiency and Growth:

- What bottlenecks in data management have been resolved since implementing the database?
- How can the system scale support the organization's growth and new program development?

2. Project Scope

2.1 Primary Focus Areas

The project will centralize data management by consolidating client, staff, program, incident, and resource data into a single, secure platform, ensuring easy access and efficient handling. It will automate key processes such as incident reporting, progress tracking, and compliance reporting, reducing manual effort and minimizing errors. Behavioral health monitoring will be enhanced through real-time tracking of client behavior patterns and outcomes, enabling timely and effective interventions. The database will be designed to scale with the organization's growth, ensuring flexibility to accommodate new programs and increased data. Additionally, it will support compliance by providing automated tools for accurate and timely reporting to regulatory bodies.

Finally, data security and privacy will be prioritized with role-based access control and robust encryption to safeguard sensitive information.

2.2 Stakeholders

The stakeholders benefiting from this project include clients, whose data will be securely managed to ensure personalized care and better outcomes. Direct Support Professionals (DSPs) will use the system to access client profiles, report incidents, and manage schedules more efficiently. Program managers and specialists will monitor program effectiveness and update behavior plans based on real-time data. Behavior specialists will leverage the system's insights to refine strategies and provide timely interventions. Administrative staff will streamline resource allocation, staff certifications, and compliance reporting. Regulatory bodies like DSHS and DDA will receive accurate and timely reports, ensuring adherence to standards. Finally, the leadership team will utilize database insights for strategic planning and organizational development.

3. RG's Current Challenges

3.1 Decentralized and Manual Data Management Processes

Raise and Grow currently relies on scattered and manual methods for managing critical data, with client, staff, and program information stored in multiple locations and formats. This decentralization results in inconsistencies, data redundancies, and inefficiencies, making it difficult to retrieve accurate information in a timely manner.

3.2 Difficulty in Tracking Client Progress and Staff Performance

The absence of a centralized system creates significant challenges in monitoring client outcomes and evaluating staff effectiveness. Tracking behavioral trends, progress in support plans, and staff compliance with certifications or training schedules is cumbersome and often incomplete, leading to missed opportunities for timely interventions and improvements.

3.3 Inefficiencies in Generating Compliance Reports

Reports for regulatory bodies such as DSHS and DDA are a time-consuming and error-prone process due to the reliance on manual data compilation. These inefficiencies increase the risk of inaccuracies, delays, and potential non-compliance, diverting valuable resources away from core operations.

3.4 Limited Scalability and Adaptability of Current Systems

The existing systems are not designed to scale with the organization's growth or adapt to new programs and evolving needs. As Raise and Grow expands its services, the current infrastructure struggles to accommodate increased client data, staff records, and program requirements, hindering operational efficiency and organizational development.

4. Database Design

4.1 Key Features

The database system will include several essential features to simplify operations and improve efficiency. A centralized data repository will consolidate all clients, staffs, programs, and resource information into a single platform for easy access and management. The system will automate key processes such as incident logging, progress tracking, and compliance reporting, reducing manual workload and minimizing errors. Role-based access control will ensure data security, allowing only authorized users to access sensitive information. Real-time dashboards and reporting tools will provide actionable insights, enabling timely decision-making. The system will also be designed to scale seamlessly, accommodating organizational growth and new program requirements while maintaining high performance.

4.2 Relationships and Schema

The database design will follow a relational structure with well-defined entities and relationships to ensure data consistency and integrity. Key entities include:

- **Clients Table**: Stores personal details, medical history, behavioral profiles, and program placements, linking to incidents and behavior plans.

- **Staff Table**: Maintains staff details, certifications, training records, and program assignments, linked to clients and programs.
- **Programs Table**: Tracks program details, client placements, and associated staff, with relationships to behavior plans and incidents.
- **Table Incidents**: Logs behavioral incidents, linked to specific clients and staff for detailed analysis.
- **Behavior Plans Table**: Stores personalized intervention plans and progress notes, tied to client profiles.
- **Resources Table**: Lists external affiliations, training materials, and related program resources.
- **Compliance Reports Table**: Tracks compliance documentation and reporting timelines, linked to programs and regulatory requirements.

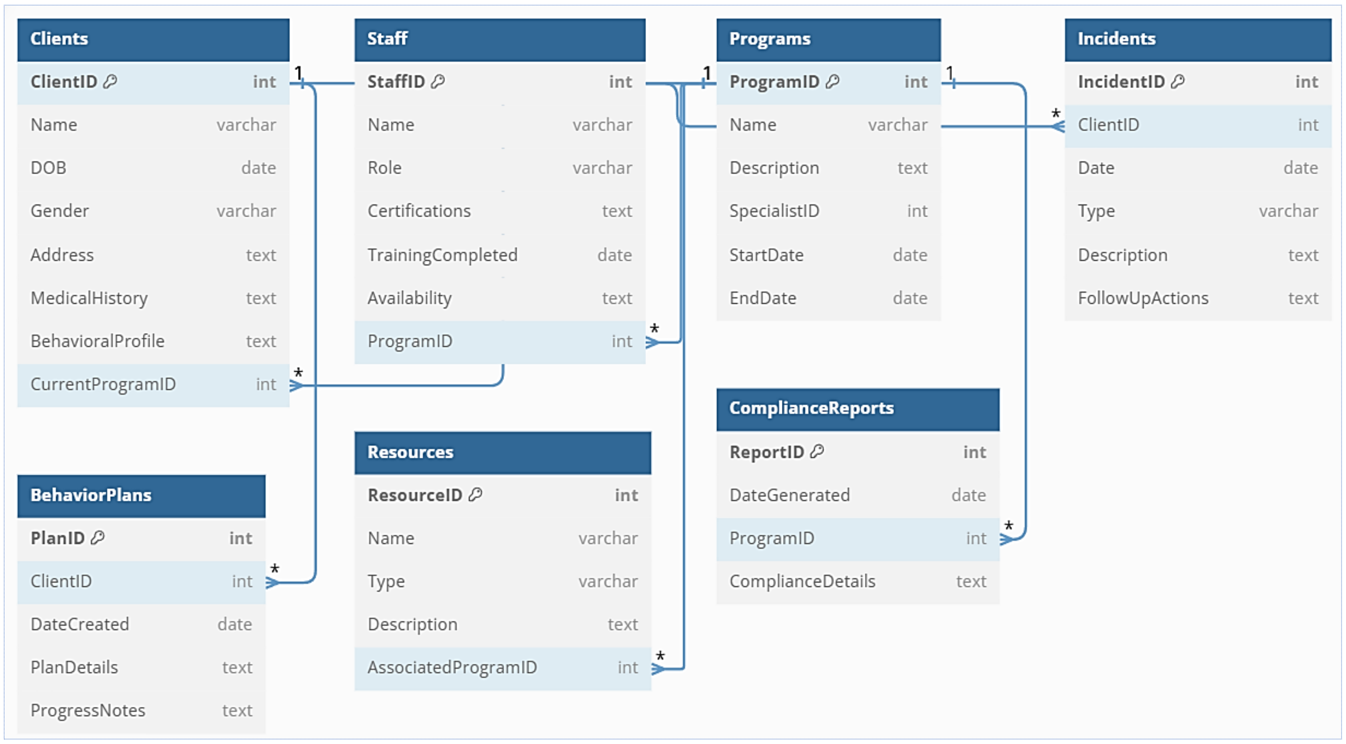
The schema is designed to establish relationships between these entities, ensuring data is interconnected and accessible for comprehensive analysis. For example, a query can retrieve a client's progress, their associated behavior plan, and the staff involved in their care, providing a complete picture in one view. This relational approach supports both operational needs and strategic decision-making.

4.3 Creating an Entity-Relationship Diagram (ERD)

Relationships between entities:

- **Clients** to **Programs**: One client belongs to one program (1:N).
- **Staff** to **Programs**: A staff can work in multiple programs (M:N).
- **Incidents** to **Clients**: Incidents are related to a specific client (1:N).
- **Behavior Plans** to **Clients**: Each client can have multiple behavior plans (1:N).
- **Programs** to **Resources**: Programs may use multiple resources (M:N).
- **Programs** to **Compliance Reports**: Programs generate multiple compliance reports (M:N).

Visualizing the Entity-Relationship Diagram (ERD) using dbdiagram.io Tool



2. Implementation Plan

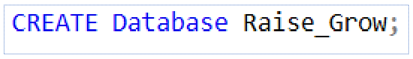
5.1 Requirements Gathering

The first phase of the implementation plan involves gathering and analyzing requirements to ensure the database system meets organizational needs. This step includes collaborating with stakeholders such as program managers, staff, and administrators to identify critical data points like client profiles, program details, incidents, and compliance requirements. The process involves documenting existing workflows, understanding pain points in current data management practices, and defining the desired outcomes, such as improved data accessibility, automated reporting, and streamlined operations. This phase ensures that the database structure, functionality, and features are aligned with operational goals and regulatory obligations.

a. Database Development

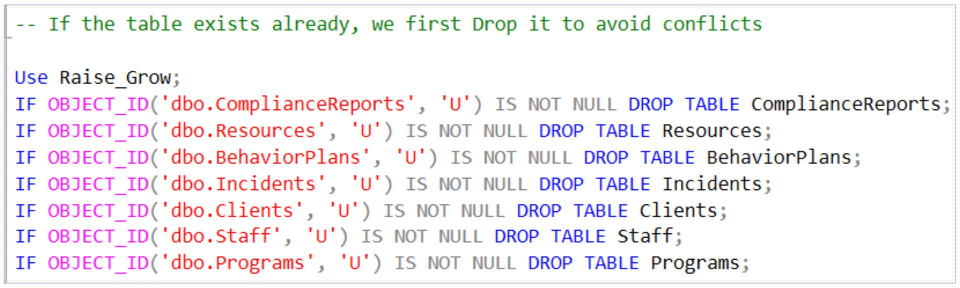
The database development phase focuses on designing and building a robust relational database system based on the requirements gathered. This includes creating the schema with properly structured tables, defining relationships between entities (e.g., clients and programs), and enforcing data integrity through constraints such as primary and foreign keys. During this phase, initial data is populated, and queries, views, and stored procedures are developed to support the system's functionality. The goal is to create a scalable and secure database that meets operational and reporting needs.

- We first create the Database, and we call it “Raise_Grow”



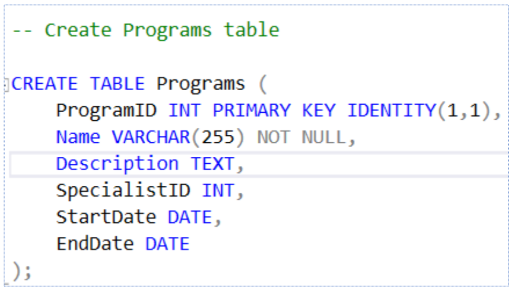
```sql
CREATE Database Raise_Grow;
```

- Then we create each table of the database. We first ensure that these tables are not do yet exist. If the Table already exists, we simply drop it.



```sql
-- If the table exists already, we first Drop it to avoid conflicts

Use Raise_Grow;
IF OBJECT_ID('dbo.ComplianceReports', 'U') IS NOT NULL DROP TABLE ComplianceReports;
IF OBJECT_ID('dbo.Resources', 'U') IS NOT NULL DROP TABLE Resources;
IF OBJECT_ID('dbo.BehaviorPlans', 'U') IS NOT NULL DROP TABLE BehaviorPlans;
IF OBJECT_ID('dbo.Incidents', 'U') IS NOT NULL DROP TABLE Incidents;
IF OBJECT_ID('dbo.Clients', 'U') IS NOT NULL DROP TABLE Clients;
IF OBJECT_ID('dbo.Staff', 'U') IS NOT NULL DROP TABLE Staff;
IF OBJECT_ID('dbo.Programs', 'U') IS NOT NULL DROP TABLE Programs;
```

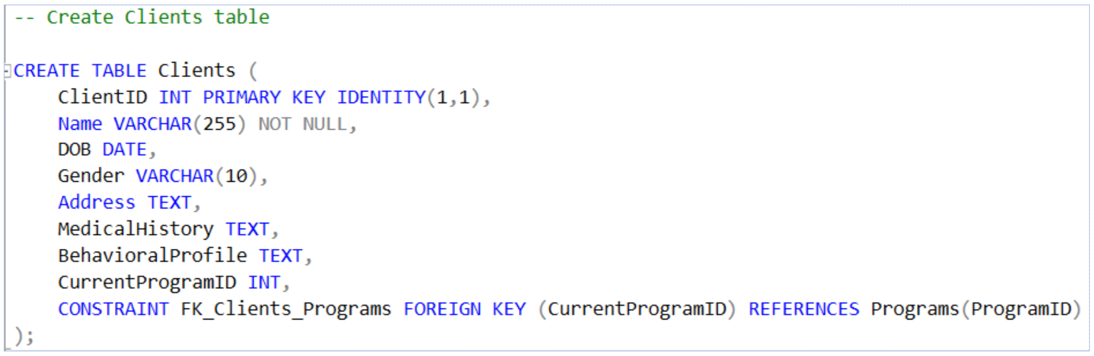
This SQL script checks if specific user-defined tables (ComplianceReports, Resources, BehaviorPlans, Incidents, Clients, Staff, Programs) exist in the Raise_Grow database using the OBJECT_ID function with “U” (indicating user-defined tables). If a table already exists, it is dropped using DROP TABLE. By doing this, we ensure there are no conflicts when recreating or modifying these tables later. This process is commonly used in development or testing to maintain schema consistency.



```sql
-- Create Programs table

CREATE TABLE Programs (
    ProgramID INT PRIMARY KEY IDENTITY(1,1),
    Name VARCHAR(255) NOT NULL,
    Description TEXT,
    SpecialistID INT,
    StartDate DATE,
    EndDate DATE
);
```

(1)

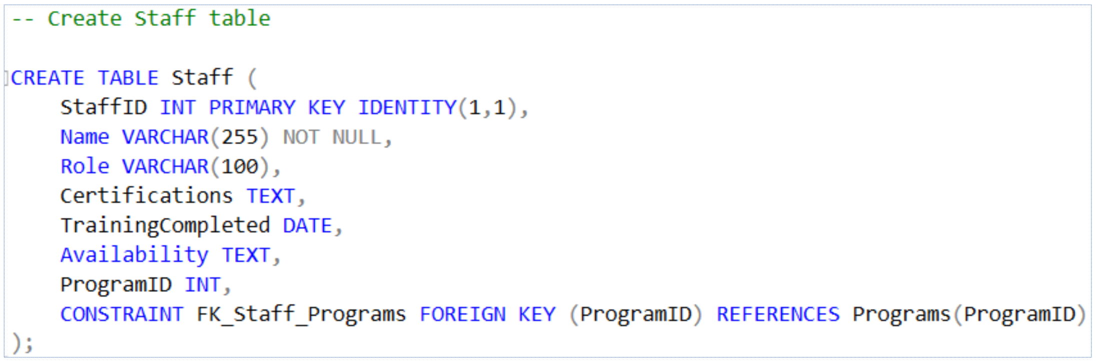
The scripts (1) create a Programs table with a unique ProgramID as the primary key, auto-incrementing with each new entry. The table includes a Name (required, max 255 characters), a Description (long text), a SpecialistID (integer reference), and StartDate and EndDate to define the program's timeline. It is designed to store detailed information about programs, ensuring each entry is uniquely identifiable and organized.



```sql
-- Create Clients table

CREATE TABLE Clients (
    ClientID INT PRIMARY KEY IDENTITY(1,1),
    Name VARCHAR(255) NOT NULL,
    DOB DATE,
    Gender VARCHAR(10),
    Address TEXT,
    MedicalHistory TEXT,
    BehavioralProfile TEXT,
    CurrentProgramID INT,
    CONSTRAINT FK_Clients_Programs FOREIGN KEY (CurrentProgramID) REFERENCES Programs(ProgramID)
);
```

(2)

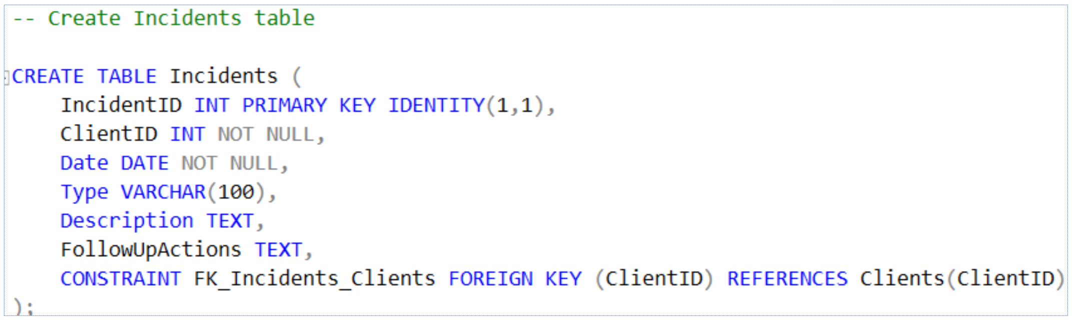
The scripts (2) create a Clients table with a unique ClientID as the primary key, auto-incrementing for each new entry. The table stores client details, including Name (required, max 255 characters), DOB (date of birth), Gender (max 10 characters), Address, MedicalHistory, and BehavioralProfile (all in text format for detailed information). It also includes a CurrentProgramID column as a foreign key referencing the ProgramID in the Programs table, establishing a relationship between clients and the programs they are associated with.



```sql
-- Create Staff table

CREATE TABLE Staff (
    StaffID INT PRIMARY KEY IDENTITY(1,1),
    Name VARCHAR(255) NOT NULL,
    Role VARCHAR(100),
    Certifications TEXT,
    TrainingCompleted DATE,
    Availability TEXT,
    ProgramID INT,
    CONSTRAINT FK_Staff_Programs FOREIGN KEY (ProgramID) REFERENCES Programs(ProgramID)
);
```

(3)

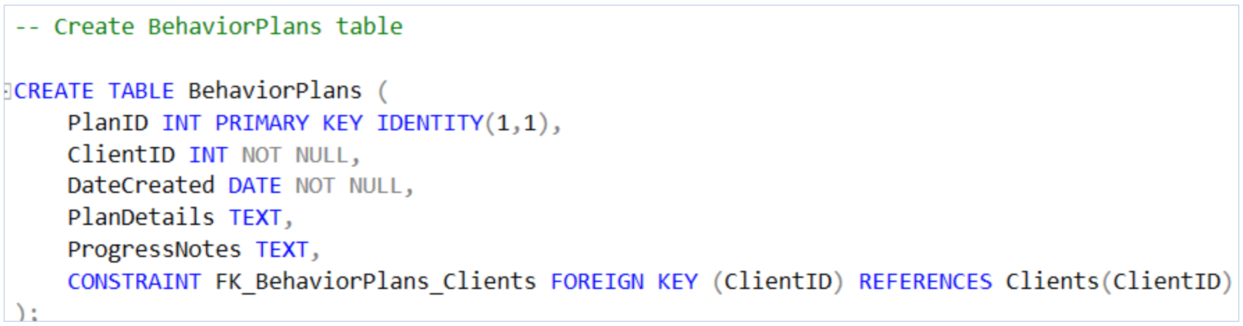
The script (3) creates a Staff table with a unique StaffID as the primary key, auto-incrementing for each new entry. It includes columns for Name (required, max 255 characters), Role (max 100 characters), Certifications (detailed text), TrainingCompleted (date), and Availability (text format). Additionally, it has a ProgramID column as a foreign key referencing the ProgramID in the Programs table, establishing a relationship between staff members and their associated programs. This table captures comprehensive staff details and their program affiliations.



```sql
-- Create Incidents table

CREATE TABLE Incidents (
    IncidentID INT PRIMARY KEY IDENTITY(1,1),
    ClientID INT NOT NULL,
    Date DATE NOT NULL,
    Type VARCHAR(100),
    Description TEXT,
    FollowUpActions TEXT,
    CONSTRAINT FK_Incidents_Clients FOREIGN KEY (ClientID) REFERENCES Clients(ClientID)
);
```

(4)

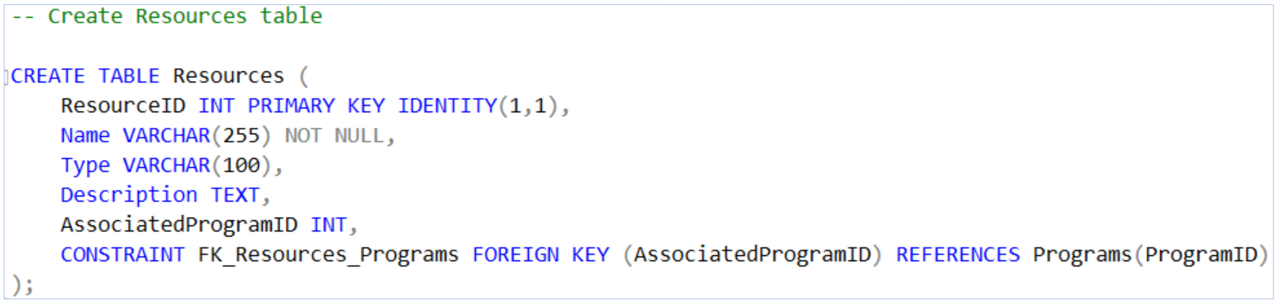
The script (4) creates an Incidents table with a unique IncidentID as the primary key, auto-incrementing for each new record. It includes the ClientID (a required foreign key referencing the Clients table), Date (the incident date, required), Type (description of the incident type, max 100 characters), Description (detailed text about the incident), and FollowUpActions (text describing actions taken after the incident). This table tracks incidents associated with clients, ensuring relational integrity with the Clients table.



```sql
-- Create BehaviorPlans table

CREATE TABLE BehaviorPlans (
    PlanID INT PRIMARY KEY IDENTITY(1,1),
    ClientID INT NOT NULL,
    DateCreated DATE NOT NULL,
    PlanDetails TEXT,
    ProgressNotes TEXT,
    CONSTRAINT FK_BehaviorPlans_Clients FOREIGN KEY (ClientID) REFERENCES Clients(ClientID)
);
```

(5)

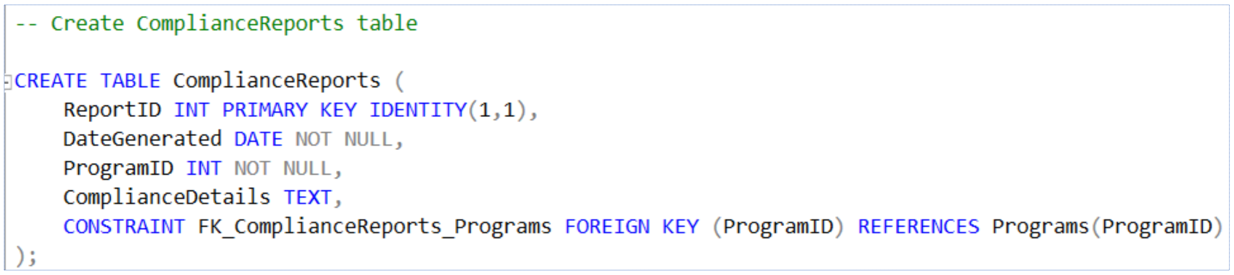
The script (6) creates a BehaviorPlans table with a unique PlanID as the primary key, auto-incrementing for each new record. It includes a ClientID (a required foreign key referencing the Clients table), DateCreated (the date the behavior plan was created, required), PlanDetails (text field for detailed behavior plan information), and ProgressNotes (text field for tracking progress). This table links behavior plans to specific clients, ensuring each plan is associated with the appropriate client in the Clients table.



```sql
-- Create Resources table

CREATE TABLE Resources (
    ResourceID INT PRIMARY KEY IDENTITY(1,1),
    Name VARCHAR(255) NOT NULL,
    Type VARCHAR(100),
    Description TEXT,
    AssociatedProgramID INT,
    CONSTRAINT FK_Resources_Programs FOREIGN KEY (AssociatedProgramID) REFERENCES Programs(ProgramID)
);
```

(6)

The script (6) creates a Resources table with a unique ResourceID as the primary key, auto-incrementing for each new entry. It includes columns for Name (required, max 255 characters), Type (describing the type of resource, max 100 characters), Description (text field for detailed information about the resource), and AssociatedProgramID (a foreign key referencing the ProgramID in the Programs table). This table is designed to store information about various resources and their association with specific programs.



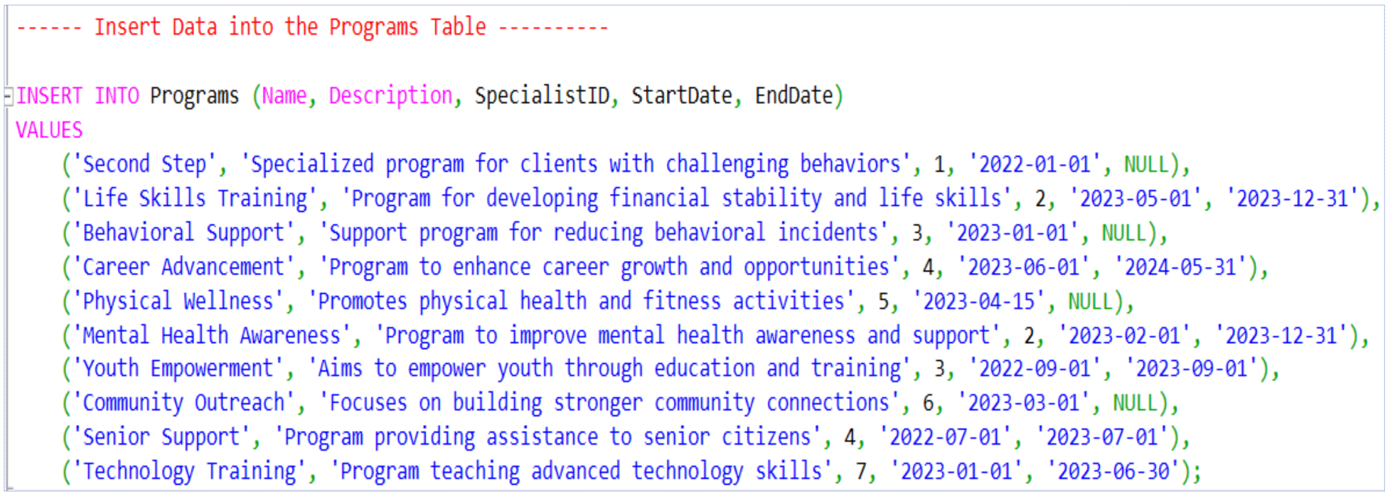
```sql
-- Create ComplianceReports table

CREATE TABLE ComplianceReports (
    ReportID INT PRIMARY KEY IDENTITY(1,1),
    DateGenerated DATE NOT NULL,
    ProgramID INT NOT NULL,
    ComplianceDetails TEXT,
    CONSTRAINT FK_ComplianceReports_Programs FOREIGN KEY (ProgramID) REFERENCES Programs(ProgramID)
);
```

(7)

The script (8) creates a ComplianceReports table with a unique ReportID as the primary key, auto-incrementing for each new record. The table includes DateGenerated (required date of report creation), ProgramID (required foreign key referencing the Programs table to associate the report with a specific program), and ComplianceDetails (a text field for detailed information about the compliance report). This table is designed to store and manage compliance-related reports for various programs.

- Populate the database with the data.

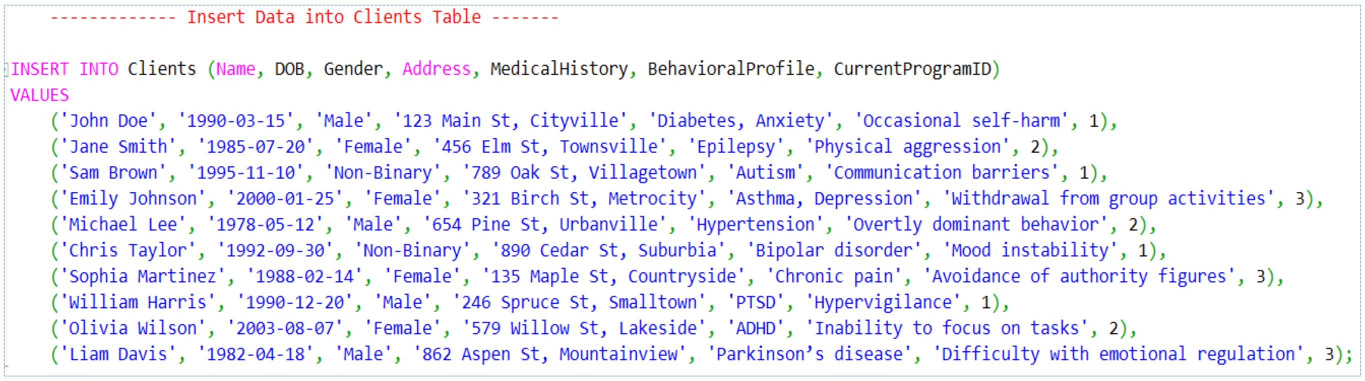
After creating these tables, the next step is to populate them with data using the INSERT INTO statement. Begin by inserting data into the parent tables such as Programs, as they are referenced by foreign keys in other tables. Once the parent tables are populated, proceed to insert data into the dependent tables like Clients, Staff, Incidents, and others, ensuring that the foreign key values correspond to valid entries in the parent tables to maintain referential integrity.



```
------ Insert Data into the Programs Table ----------

INSERT INTO Programs (Name, Description, SpecialistID, StartDate, EndDate)
VALUES
    ('Second Step', 'Specialized program for clients with challenging behaviors', 1, '2022-01-01', NULL),
    ('Life Skills Training', 'Program for developing financial stability and life skills', 2, '2023-05-01', '2023-12-31'),
    ('Behavioral Support', 'Support program for reducing behavioral incidents', 3, '2023-01-01', NULL),
    ('Career Advancement', 'Program to enhance career growth and opportunities', 4, '2023-06-01', '2024-05-31'),
    ('Physical Wellness', 'Promotes physical health and fitness activities', 5, '2023-04-15', NULL),
    ('Mental Health Awareness', 'Program to improve mental health awareness and support', 2, '2023-02-01', '2023-12-31'),
    ('Youth Empowerment', 'Aims to empower youth through education and training', 3, '2022-09-01', '2023-09-01'),
    ('Community Outreach', 'Focuses on building stronger community connections', 6, '2023-03-01', NULL),
    ('Senior Support', 'Program providing assistance to senior citizens', 4, '2022-07-01', '2023-07-01'),
    ('Technology Training', 'Program teaching advanced technology skills', 7, '2023-01-01', '2023-06-30');
```

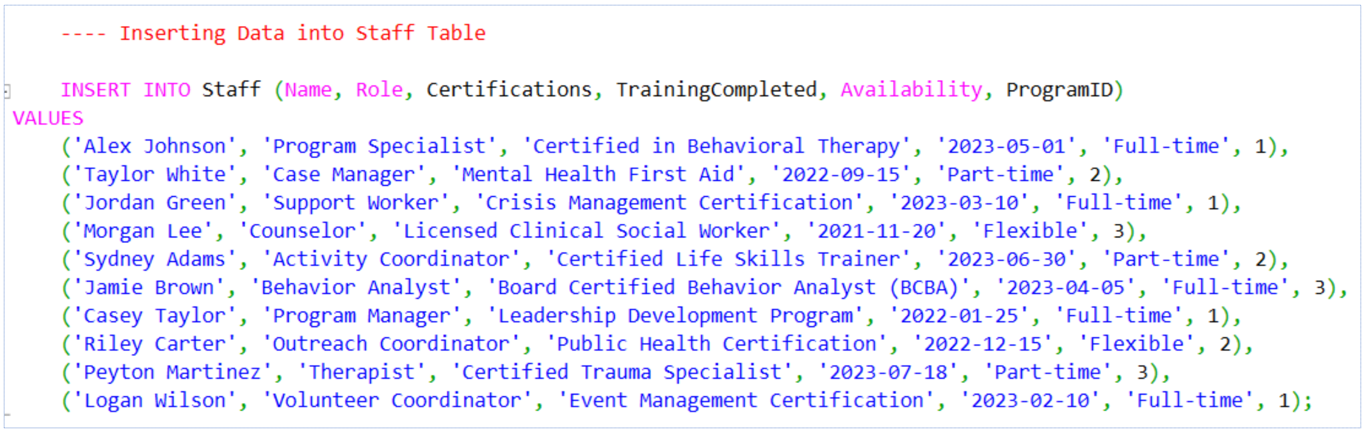
Each row represents a unique program with its name, description, specialist in charge (SpecialistID), start date (StartDate), and optional end date (EndDate). The “NULL” in EndDate indicates ongoing programs without a specified end date.

- ✓ After inserting records into the Programs table, the next step is to insert data into the **Clients** table, as it does not have foreign key dependencies on any other table except Programs.



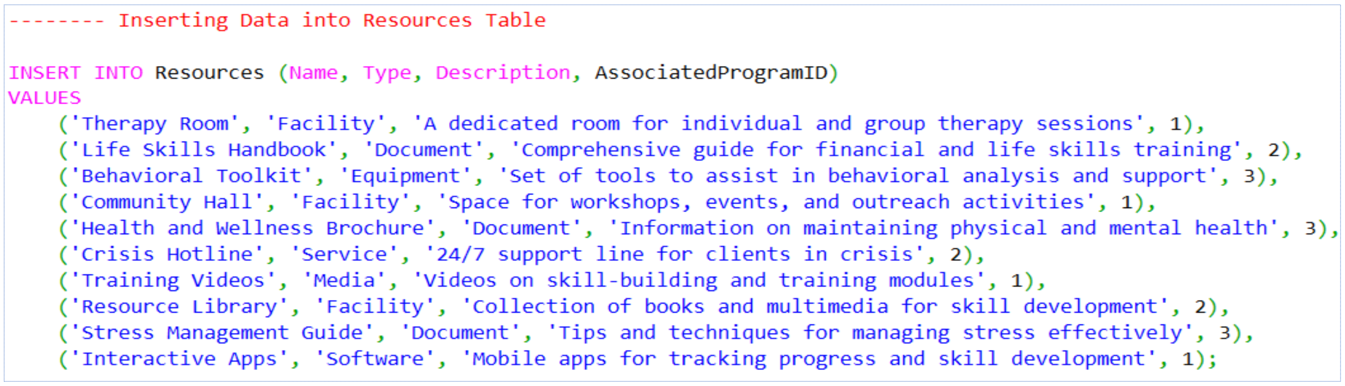
```
    ------------- Insert Data into Clients Table -------

INSERT INTO Clients (Name, DOB, Gender, Address, MedicalHistory, BehavioralProfile, CurrentProgramID)
VALUES
    ('John Doe', '1990-03-15', 'Male', '123 Main St, Cityville', 'Diabetes, Anxiety', 'Occasional self-harm', 1),
    ('Jane Smith', '1985-07-20', 'Female', '456 Elm St, Townsville', 'Epilepsy', 'Physical aggression', 2),
    ('Sam Brown', '1995-11-10', 'Non-Binary', '789 Oak St, Villagetown', 'Autism', 'Communication barriers', 1),
    ('Emily Johnson', '2000-01-25', 'Female', '321 Birch St, Metrocity', 'Asthma, Depression', 'Withdrawal from group activities', 3),
    ('Michael Lee', '1978-05-12', 'Male', '654 Pine St, Urbanville', 'Hypertension', 'Overtly dominant behavior', 2),
    ('Chris Taylor', '1992-09-30', 'Non-Binary', '890 Cedar St, Suburbia', 'Bipolar disorder', 'Mood instability', 1),
    ('Sophia Martinez', '1988-02-14', 'Female', '135 Maple St, Countryside', 'Chronic pain', 'Avoidance of authority figures', 3),
    ('William Harris', '1990-12-20', 'Male', '246 Spruce St, Smalltown', 'PTSD', 'Hypervigilance', 1),
    ('Olivia Wilson', '2003-08-07', 'Female', '579 Willow St, Lakeside', 'ADHD', 'Inability to focus on tasks', 2),
    ('Liam Davis', '1982-04-18', 'Male', '862 Aspen St, Mountainview', 'Parkinson’s disease', 'Difficulty with emotional regulation', 3);
```

- ✓ Next, we insert data into the **Staff** table.



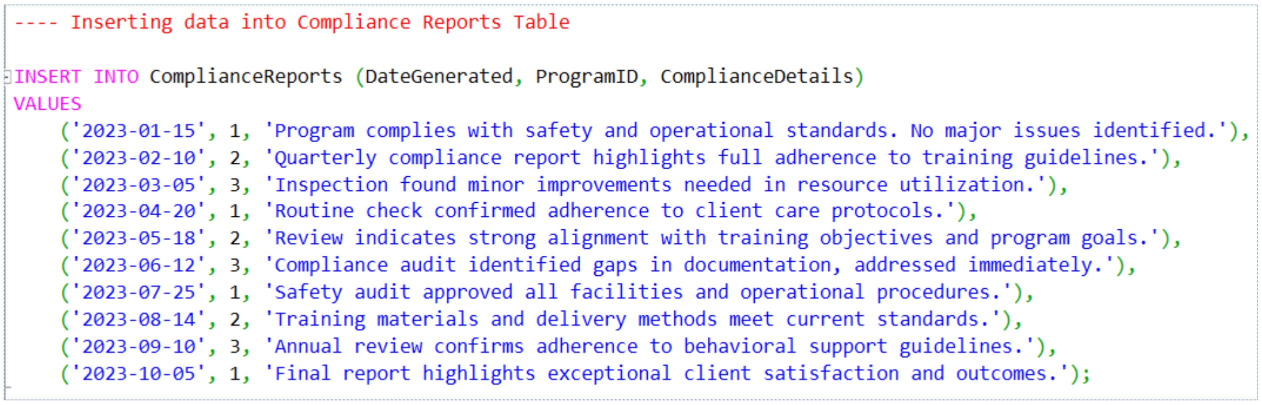
```sql
    ---- Inserting Data into Staff Table

    INSERT INTO Staff (Name, Role, Certifications, TrainingCompleted, Availability, ProgramID)
VALUES
    ('Alex Johnson', 'Program Specialist', 'Certified in Behavioral Therapy', '2023-05-01', 'Full-time', 1),
    ('Taylor White', 'Case Manager', 'Mental Health First Aid', '2022-09-15', 'Part-time', 2),
    ('Jordan Green', 'Support Worker', 'Crisis Management Certification', '2023-03-10', 'Full-time', 1),
    ('Morgan Lee', 'Counselor', 'Licensed Clinical Social Worker', '2021-11-20', 'Flexible', 3),
    ('Sydney Adams', 'Activity Coordinator', 'Certified Life Skills Trainer', '2023-06-30', 'Part-time', 2),
    ('Jamie Brown', 'Behavior Analyst', 'Board Certified Behavior Analyst (BCBA)', '2023-04-05', 'Full-time', 3),
    ('Casey Taylor', 'Program Manager', 'Leadership Development Program', '2022-01-25', 'Full-time', 1),
    ('Riley Carter', 'Outreach Coordinator', 'Public Health Certification', '2022-12-15', 'Flexible', 2),
    ('Peyton Martinez', 'Therapist', 'Certified Trauma Specialist', '2023-07-18', 'Part-time', 3),
    ('Logan Wilson', 'Volunteer Coordinator', 'Event Management Certification', '2023-02-10', 'Full-time', 1);
```

✓ Then, we insert data into the Resources Table



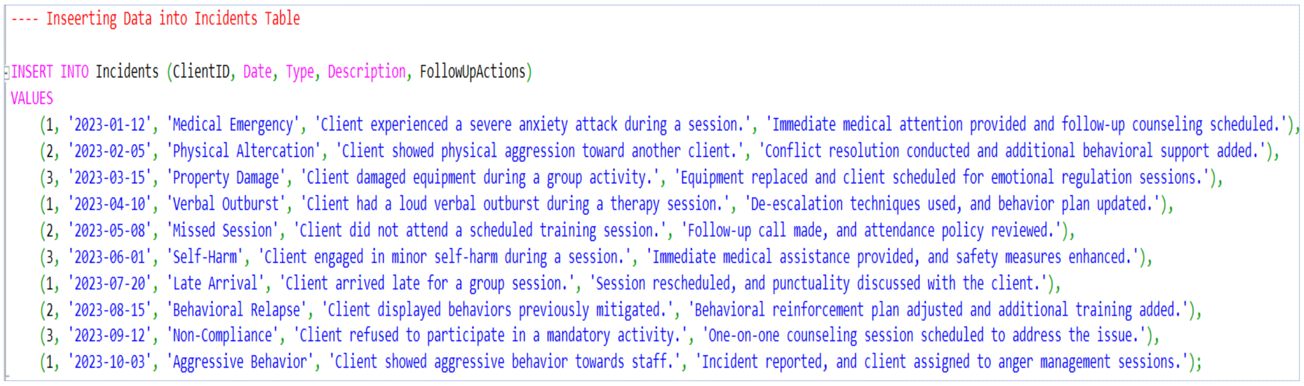
```sql
-------- Inserting Data into Resources Table

INSERT INTO Resources (Name, Type, Description, AssociatedProgramID)
VALUES
    ('Therapy Room', 'Facility', 'A dedicated room for individual and group therapy sessions', 1),
    ('Life Skills Handbook', 'Document', 'Comprehensive guide for financial and life skills training', 2),
    ('Behavioral Toolkit', 'Equipment', 'Set of tools to assist in behavioral analysis and support', 3),
    ('Community Hall', 'Facility', 'Space for workshops, events, and outreach activities', 1),
    ('Health and Wellness Brochure', 'Document', 'Information on maintaining physical and mental health', 3),
    ('Crisis Hotline', 'Service', '24/7 support line for clients in crisis', 2),
    ('Training Videos', 'Media', 'Videos on skill-building and training modules', 1),
    ('Resource Library', 'Facility', 'Collection of books and multimedia for skill development', 2),
    ('Stress Management Guide', 'Document', 'Tips and techniques for managing stress effectively', 3),
    ('Interactive Apps', 'Software', 'Mobile apps for tracking progress and skill development', 1);
```

✓ After inserting resources, the next step is to populate the **ComplianceReports** table.



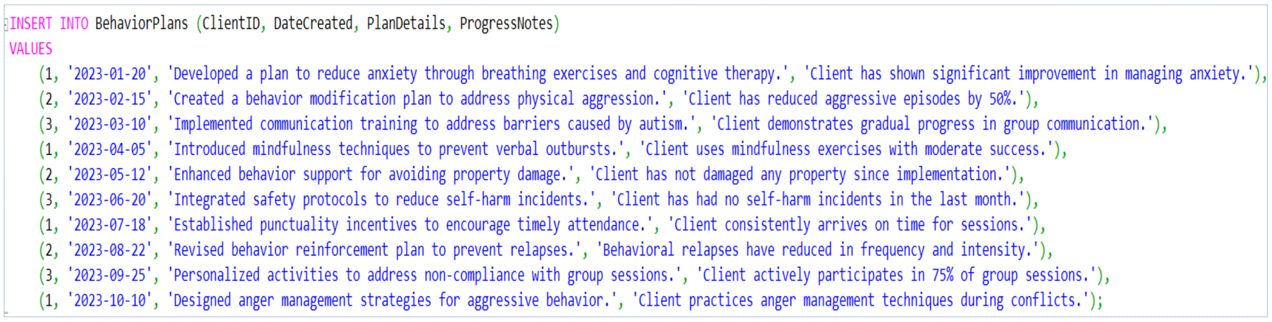
```sql
---- Inserting data into Compliance Reports Table

INSERT INTO ComplianceReports (DateGenerated, ProgramID, ComplianceDetails)
VALUES
    ('2023-01-15', 1, 'Program complies with safety and operational standards. No major issues identified.'),
    ('2023-02-10', 2, 'Quarterly compliance report highlights full adherence to training guidelines.'),
    ('2023-03-05', 3, 'Inspection found minor improvements needed in resource utilization.'),
    ('2023-04-20', 1, 'Routine check confirmed adherence to client care protocols.'),
    ('2023-05-18', 2, 'Review indicates strong alignment with training objectives and program goals.'),
    ('2023-06-12', 3, 'Compliance audit identified gaps in documentation, addressed immediately.'),
    ('2023-07-25', 1, 'Safety audit approved all facilities and operational procedures.'),
    ('2023-08-14', 2, 'Training materials and delivery methods meet current standards.'),
    ('2023-09-10', 3, 'Annual review confirms adherence to behavioral support guidelines.'),
    ('2023-10-05', 1, 'Final report highlights exceptional client satisfaction and outcomes.');
```

✓ Next, we insert data into the **Incidents** table.



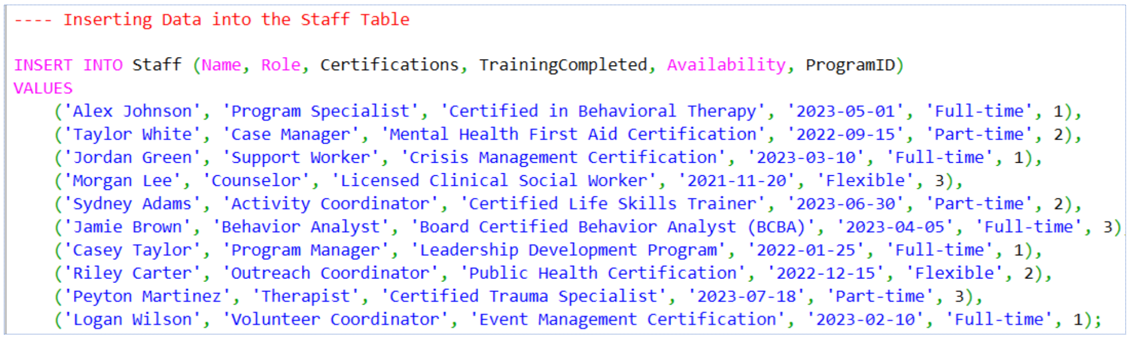
```sql
---- Inseerting Data into Incidents Table

INSERT INTO Incidents (ClientID, Date, Type, Description, FollowUpActions)
VALUES
    (1, '2023-01-12', 'Medical Emergency', 'Client experienced a severe anxiety attack during a session.', 'Immediate medical attention provided and follow-up counseling scheduled.'),
    (2, '2023-02-05', 'Physical Altercation', 'Client showed physical aggression toward another client.', 'Conflict resolution conducted and additional behavioral support added.'),
    (3, '2023-03-15', 'Property Damage', 'Client damaged equipment during a group activity.', 'Equipment replaced and client scheduled for emotional regulation sessions.'),
    (1, '2023-04-10', 'Verbal Outburst', 'Client had a loud verbal outburst during a therapy session.', 'De-escalation techniques used, and behavior plan updated.'),
    (2, '2023-05-08', 'Missed Session', 'Client did not attend a scheduled training session.', 'Follow-up call made, and attendance policy reviewed.'),
    (3, '2023-06-01', 'Self-Harm', 'Client engaged in minor self-harm during a session.', 'Immediate medical assistance provided, and safety measures enhanced.'),
    (1, '2023-07-20', 'Late Arrival', 'Client arrived late for a group session.', 'Session rescheduled, and punctuality discussed with the client.'),
    (2, '2023-08-15', 'Behavioral Relapse', 'Client displayed behaviors previously mitigated.', 'Behavioral reinforcement plan adjusted and additional training added.'),
    (3, '2023-09-12', 'Non-Compliance', 'Client refused to participate in a mandatory activity.', 'One-on-one counseling session scheduled to address the issue.'),
    (1, '2023-10-03', 'Aggressive Behavior', 'Client showed aggressive behavior towards staff.', 'Incident reported, and client assigned to anger management sessions.');
```

- ✓ The Next step is to populate the Behaviors Plans Table with data



```sql
INSERT INTO BehaviorPlans (ClientID, DateCreated, PlanDetails, ProgressNotes)
VALUES
    (1, '2023-01-20', 'Developed a plan to reduce anxiety through breathing exercises and cognitive therapy.', 'Client has shown significant improvement in managing anxiety.'),
    (2, '2023-02-15', 'Created a behavior modification plan to address physical aggression.', 'Client has reduced aggressive episodes by 50%.'),
    (3, '2023-03-10', 'Implemented communication training to address barriers caused by autism.', 'Client demonstrates gradual progress in group communication.'),
    (1, '2023-04-05', 'Introduced mindfulness techniques to prevent verbal outbursts.', 'Client uses mindfulness exercises with moderate success.'),
    (2, '2023-05-12', 'Enhanced behavior support for avoiding property damage.', 'Client has not damaged any property since implementation.'),
    (3, '2023-06-20', 'Integrated safety protocols to reduce self-harm incidents.', 'Client has had no self-harm incidents in the last month.'),
    (1, '2023-07-18', 'Established punctuality incentives to encourage timely attendance.', 'Client consistently arrives on time for sessions.'),
    (2, '2023-08-22', 'Revised behavior reinforcement plan to prevent relapses.', 'Behavioral relapses have reduced in frequency and intensity.'),
    (3, '2023-09-25', 'Personalized activities to address non-compliance with group sessions.', 'Client actively participates in 75% of group sessions.'),
    (1, '2023-10-10', 'Designed anger management strategies for aggressive behavior.', 'Client practices anger management techniques during conflicts.');
```

- ✓ Finally, we insert data into the Staff Table



```sql
---- Inserting Data into the Staff Table

INSERT INTO Staff (Name, Role, Certifications, TrainingCompleted, Availability, ProgramID)
VALUES
    ('Alex Johnson', 'Program Specialist', 'Certified in Behavioral Therapy', '2023-05-01', 'Full-time', 1),
    ('Taylor White', 'Case Manager', 'Mental Health First Aid Certification', '2022-09-15', 'Part-time', 2),
    ('Jordan Green', 'Support Worker', 'Crisis Management Certification', '2023-03-10', 'Full-time', 1),
    ('Morgan Lee', 'Counselor', 'Licensed Clinical Social Worker', '2021-11-20', 'Flexible', 3),
    ('Sydney Adams', 'Activity Coordinator', 'Certified Life Skills Trainer', '2023-06-30', 'Part-time', 2),
    ('Jamie Brown', 'Behavior Analyst', 'Board Certified Behavior Analyst (BCBA)', '2023-04-05', 'Full-time', 3)
    ('Casey Taylor', 'Program Manager', 'Leadership Development Program', '2022-01-25', 'Full-time', 1),
    ('Riley Carter', 'Outreach Coordinator', 'Public Health Certification', '2022-12-15', 'Flexible', 2),
    ('Peyton Martinez', 'Therapist', 'Certified Trauma Specialist', '2023-07-18', 'Part-time', 3),
    ('Logan Wilson', 'Volunteer Coordinator', 'Event Management Certification', '2023-02-10', 'Full-time', 1);
```

With the completion of the data insertion process across all tables, we now move to the **Verification Queries** step. This step ensures that the inserted data is accurate, complete, and adheres to the defined relationships and constraints. By executing targeted SELECT queries, we will validate the integrity of each table and confirm that all foreign key references are functioning as expected, setting the stage for confident use of the database in future operations. We will combine this step with the view procedures.

b. Views and Stored Procedures

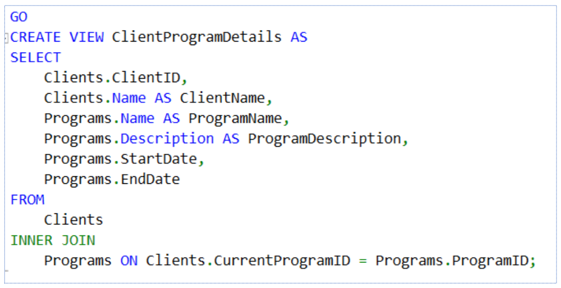
Views and Stored Procedures are essential tools for efficient data management and retrieval. The views and stored will help us also verify and validate the integrity of each table and confirm that all foreign key references are functioning as expected.

The **Views** are virtual tables that provide a simplified, focused, and reusable way to query data from one or more tables. They are particularly useful for presenting complex “joins” queries or frequently used queries in a user-friendly format.

The **Stored Procedures**: These are precompiled SQL scripts stored in the database that perform specific tasks or queries. They enhance database functionality by reducing repetitive query writing, improving performance, and providing security through encapsulation.
This step will enhance data interaction, efficiency, and ensure consistency in accessing and processing information.

❖ Views

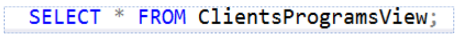
Views are virtual tables that simplify complex queries and provide a user-friendly way to access data. They are particularly useful for creating reports and managing information that is frequently accessed or retrieved. For instance, the following view script lists Clients and their current programs.



```
GO
CREATE VIEW ClientProgramDetails AS
SELECT
    Clients.ClientID,
    Clients.Name AS ClientName,
    Programs.Name AS ProgramName,
    Programs.Description AS ProgramDescription,
    Programs.StartDate,
    Programs.EndDate
FROM
    Clients
INNER JOIN
    Programs ON Clients.CurrentProgramID = Programs.ProgramID;
```

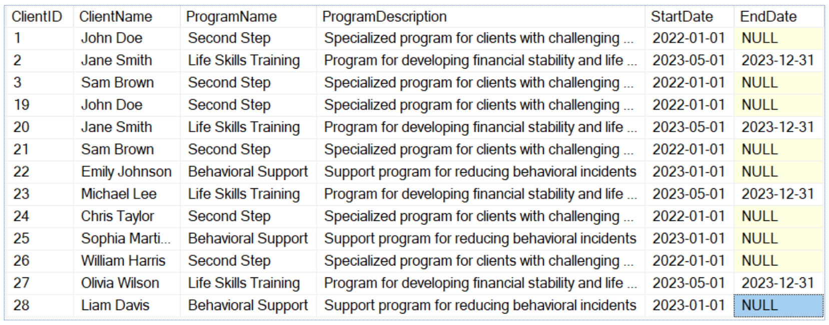
The above script creates a view named “ClientProgramDetails”, which combines data from the Clients and Programs tables using an INNER JOIN on the CurrentProgramID and ProgramID columns. The view selects key columns, including the client's ID and name (ClientID, ClientName), the program's name and description (ProgramName, ProgramDescription), and the program's start and end dates. This view simplifies querying by providing a consolidated dataset of clients and their associated program details.

To retrieve all columns from the ClientProgramDetails view, we can retrieve all columns from the Clients table to view after it has been successfully created:



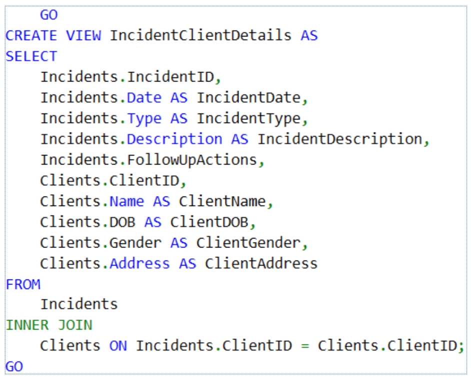
```
SELECT * FROM ClientsProgramsView;
```

And we get the following output:



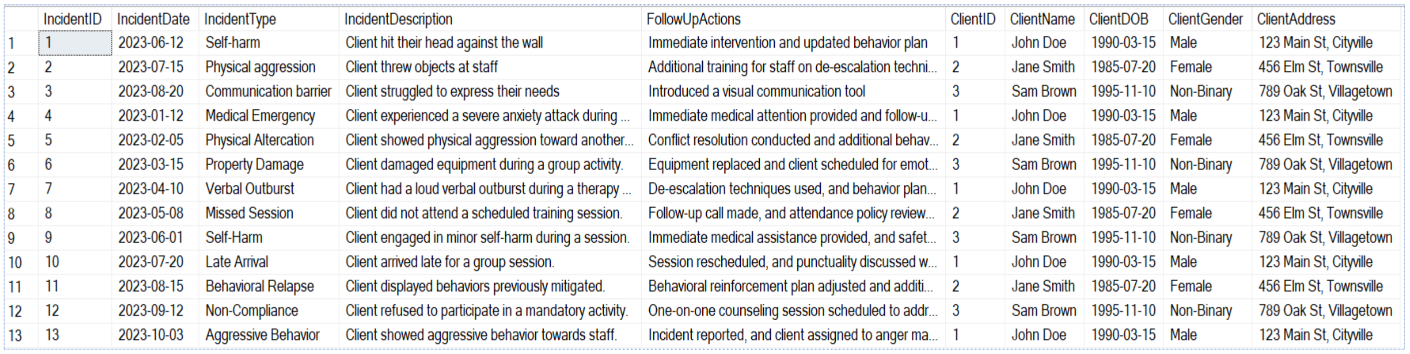
ClientID	ClientName	ProgramName	ProgramDescription	StartDate	EndDate
1	John Doe	Second Step	Specialized program for clients with challenging ...	2022-01-01	NULL
2	Jane Smith	Life Skills Training	Program for developing financial stability and life ...	2023-05-01	2023-12-31
3	Sam Brown	Second Step	Specialized program for clients with challenging ...	2022-01-01	NULL
19	John Doe	Second Step	Specialized program for clients with challenging ...	2022-01-01	NULL
20	Jane Smith	Life Skills Training	Program for developing financial stability and life ...	2023-05-01	2023-12-31
21	Sam Brown	Second Step	Specialized program for clients with challenging ...	2022-01-01	NULL
22	Emily Johnson	Behavioral Support	Support program for reducing behavioral incidents	2023-01-01	NULL
23	Michael Lee	Life Skills Training	Program for developing financial stability and life ...	2023-05-01	2023-12-31
24	Chris Taylor	Second Step	Specialized program for clients with challenging ...	2022-01-01	NULL
25	Sophia Marti...	Behavioral Support	Support program for reducing behavioral incidents	2023-01-01	NULL
26	William Harris	Second Step	Specialized program for clients with challenging ...	2022-01-01	NULL
27	Olivia Wilson	Life Skills Training	Program for developing financial stability and life ...	2023-05-01	2023-12-31
28	Liam Davis	Behavioral Support	Support program for reducing behavioral incidents	2023-01-01	NULL

❖ View of the Incidents and Client Details



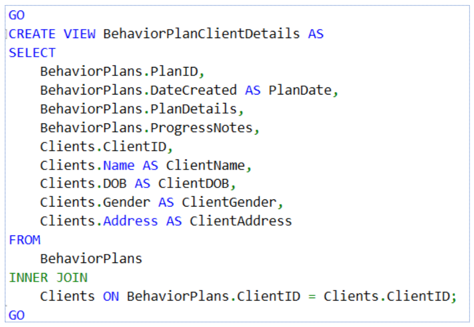
```
    GO
CREATE VIEW IncidentClientDetails AS
SELECT
    Incidents.IncidentID,
    Incidents.Date AS IncidentDate,
    Incidents.Type AS IncidentType,
    Incidents.Description AS IncidentDescription,
    Incidents.FollowUpActions,
    Clients.ClientID,
    Clients.Name AS ClientName,
    Clients.DOB AS ClientDOB,
    Clients.Gender AS ClientGender,
    Clients.Address AS ClientAddress
FROM
    Incidents
INNER JOIN
    Clients ON Incidents.ClientID = Clients.ClientID;
GO
```

✓ To retrieve all data from the view:



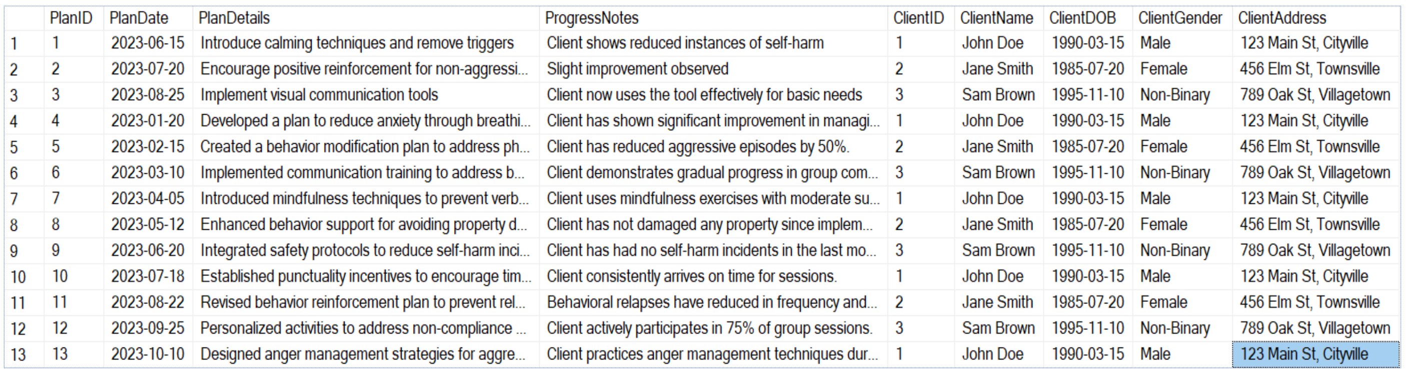
	IncidentID	IncidentDate	IncidentType	IncidentDescription	FollowUpActions	ClientID	ClientName	ClientDOB	ClientGender	ClientAddress
1	1	2023-06-12	Self-harm	Client hit their head against the wall	Immediate intervention and updated behavior plan	1	John Doe	1990-03-15	Male	123 Main St, Cityville
2	2	2023-07-15	Physical aggression	Client threw objects at staff	Additional training for staff on de-escalation techni...	2	Jane Smith	1985-07-20	Female	456 Elm St, Townsville
3	3	2023-08-20	Communication barrier	Client struggled to express their needs	Introduced a visual communication tool	3	Sam Brown	1995-11-10	Non-Binary	789 Oak St, Villagetown
4	4	2023-01-12	Medical Emergency	Client experienced a severe anxiety attack during ...	Immediate medical attention provided and follow-u...	1	John Doe	1990-03-15	Male	123 Main St, Cityville
5	5	2023-02-05	Physical Altercation	Client showed physical aggression toward another...	Conflict resolution conducted and additional behav...	2	Jane Smith	1985-07-20	Female	456 Elm St, Townsville
6	6	2023-03-15	Property Damage	Client damaged equipment during a group activity.	Equipment replaced and client scheduled for emot...	3	Sam Brown	1995-11-10	Non-Binary	789 Oak St, Villagetown
7	7	2023-04-10	Verbal Outburst	Client had a loud verbal outburst during a therapy ...	De-escalation techniques used, and behavior plan...	1	John Doe	1990-03-15	Male	123 Main St, Cityville
8	8	2023-05-08	Missed Session	Client did not attend a scheduled training session.	Follow-up call made, and attendance policy review...	2	Jane Smith	1985-07-20	Female	456 Elm St, Townsville
9	9	2023-06-01	Self-Harm	Client engaged in minor self-harm during a session.	Immediate medical assistance provided, and safet...	3	Sam Brown	1995-11-10	Non-Binary	789 Oak St, Villagetown
10	10	2023-07-20	Late Arrival	Client arrived late for a group session.	Session rescheduled, and punctuality discussed w...	1	John Doe	1990-03-15	Male	123 Main St, Cityville
11	11	2023-08-15	Behavioral Relapse	Client displayed behaviors previously mitigated.	Behavioral reinforcement plan adjusted and additi...	2	Jane Smith	1985-07-20	Female	456 Elm St, Townsville
12	12	2023-09-12	Non-Compliance	Client refused to participate in a mandatory activity.	One-on-one counseling session scheduled to addr...	3	Sam Brown	1995-11-10	Non-Binary	789 Oak St, Villagetown
13	13	2023-10-03	Aggressive Behavior	Client showed aggressive behavior towards staff.	Incident reported, and client assigned to anger ma...	1	John Doe	1990-03-15	Male	123 Main St, Cityville

❖ Behavior Plans and Client Details



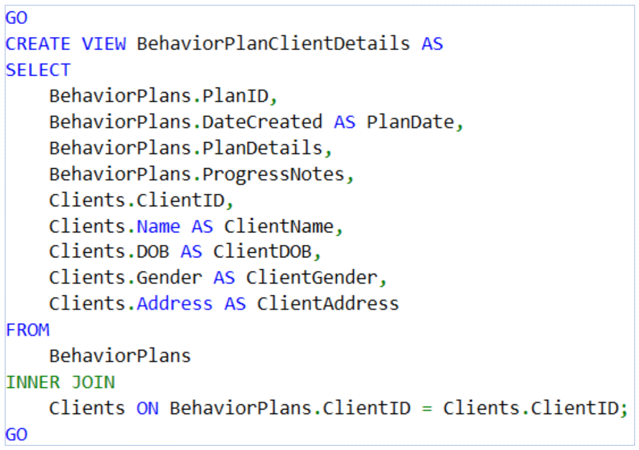
```
GO
CREATE VIEW BehaviorPlanClientDetails AS
SELECT
    BehaviorPlans.PlanID,
    BehaviorPlans.DateCreated AS PlanDate,
    BehaviorPlans.PlanDetails,
    BehaviorPlans.ProgressNotes,
    Clients.ClientID,
    Clients.Name AS ClientName,
    Clients.DOB AS ClientDOB,
    Clients.Gender AS ClientGender,
    Clients.Address AS ClientAddress
FROM
    BehaviorPlans
INNER JOIN
    Clients ON BehaviorPlans.ClientID = Clients.ClientID;
GO
```

✓ To retrieve all data from the view

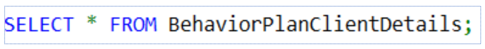


	PlanID	PlanDate	PlanDetails	ProgressNotes	ClientID	ClientName	ClientDOB	ClientGender	ClientAddress
1	1	2023-06-15	Introduce calming techniques and remove triggers	Client shows reduced instances of self-harm	1	John Doe	1990-03-15	Male	123 Main St, Cityville
2	2	2023-07-20	Encourage positive reinforcement for non-aggressi...	Slight improvement observed	2	Jane Smith	1985-07-20	Female	456 Elm St, Townsville
3	3	2023-08-25	Implement visual communication tools	Client now uses the tool effectively for basic needs	3	Sam Brown	1995-11-10	Non-Binary	789 Oak St, Villagetown
4	4	2023-01-20	Developed a plan to reduce anxiety through breathi...	Client has shown significant improvement in managi...	1	John Doe	1990-03-15	Male	123 Main St, Cityville
5	5	2023-02-15	Created a behavior modification plan to address ph...	Client has reduced aggressive episodes by 50%.	2	Jane Smith	1985-07-20	Female	456 Elm St, Townsville
6	6	2023-03-10	Implemented communication training to address b...	Client demonstrates gradual progress in group com...	3	Sam Brown	1995-11-10	Non-Binary	789 Oak St, Villagetown
7	7	2023-04-05	Introduced mindfulness techniques to prevent verb...	Client uses mindfulness exercises with moderate su...	1	John Doe	1990-03-15	Male	123 Main St, Cityville
8	8	2023-05-12	Enhanced behavior support for avoiding property d...	Client has not damaged any property since implem...	2	Jane Smith	1985-07-20	Female	456 Elm St, Townsville
9	9	2023-06-20	Integrated safety protocols to reduce self-harm inci...	Client has had no self-harm incidents in the last mo...	3	Sam Brown	1995-11-10	Non-Binary	789 Oak St, Villagetown
10	10	2023-07-18	Established punctuality incentives to encourage tim...	Client consistently arrives on time for sessions.	1	John Doe	1990-03-15	Male	123 Main St, Cityville
11	11	2023-08-22	Revised behavior reinforcement plan to prevent rel...	Behavioral relapses have reduced in frequency and...	2	Jane Smith	1985-07-20	Female	456 Elm St, Townsville
12	12	2023-09-25	Personalized activities to address non-compliance ...	Client actively participates in 75% of group sessions.	3	Sam Brown	1995-11-10	Non-Binary	789 Oak St, Villagetown
13	13	2023-10-10	Designed anger management strategies for aggre...	Client practices anger management techniques dur...	1	John Doe	1990-03-15	Male	123 Main St, Cityville

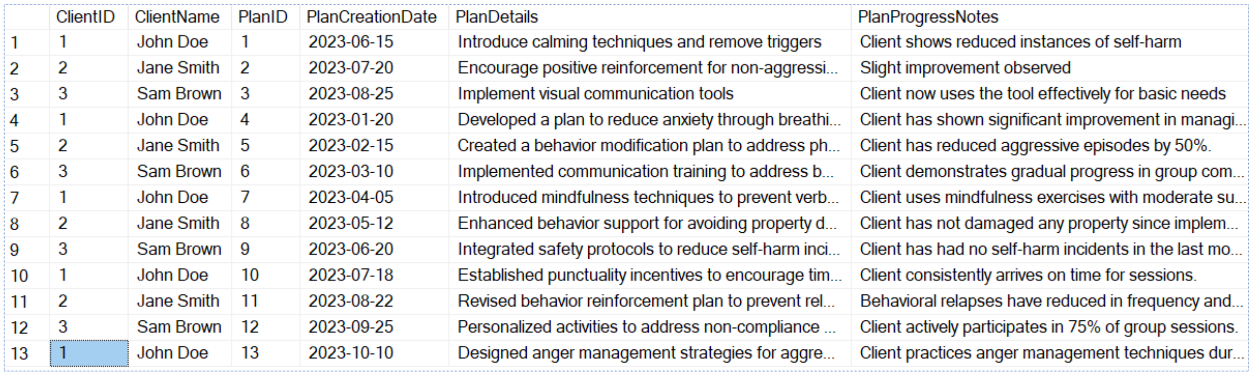
✓ The view Lists behavior plans for each client, including the plan creation date, details, and progress notes.



```
GO
CREATE VIEW BehaviorPlanClientDetails AS
SELECT
    BehaviorPlans.PlanID,
    BehaviorPlans.DateCreated AS PlanDate,
    BehaviorPlans.PlanDetails,
    BehaviorPlans.ProgressNotes,
    Clients.ClientID,
    Clients.Name AS ClientName,
    Clients.DOB AS ClientDOB,
    Clients.Gender AS ClientGender,
    Clients.Address AS ClientAddress
FROM
    BehaviorPlans
INNER JOIN
    Clients ON BehaviorPlans.ClientID = Clients.ClientID;
GO
```

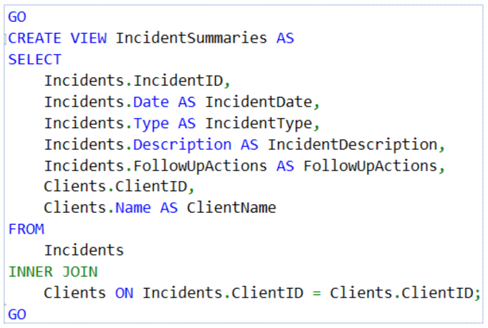


```sql
SELECT * FROM BehaviorPlanClientDetails;
```

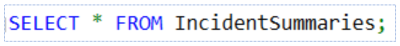


	ClientID	ClientName	PlanID	PlanCreationDate	PlanDetails	PlanProgressNotes
1	1	John Doe	1	2023-06-15	Introduce calming techniques and remove triggers	Client shows reduced instances of self-harm
2	2	Jane Smith	2	2023-07-20	Encourage positive reinforcement for non-aggressi...	Slight improvement observed
3	3	Sam Brown	3	2023-08-25	Implement visual communication tools	Client now uses the tool effectively for basic needs
4	1	John Doe	4	2023-01-20	Developed a plan to reduce anxiety through breathi...	Client has shown significant improvement in managi...
5	2	Jane Smith	5	2023-02-15	Created a behavior modification plan to address ph...	Client has reduced aggressive episodes by 50%.
6	3	Sam Brown	6	2023-03-10	Implemented communication training to address b...	Client demonstrates gradual progress in group com...
7	1	John Doe	7	2023-04-05	Introduced mindfulness techniques to prevent verb...	Client uses mindfulness exercises with moderate su...
8	2	Jane Smith	8	2023-05-12	Enhanced behavior support for avoiding property d...	Client has not damaged any property since implem...
9	3	Sam Brown	9	2023-06-20	Integrated safety protocols to reduce self-harm inci...	Client has had no self-harm incidents in the last mo...
10	1	John Doe	10	2023-07-18	Established punctuality incentives to encourage tim...	Client consistently arrives on time for sessions.
11	2	Jane Smith	11	2023-08-22	Revised behavior reinforcement plan to prevent rel...	Behavioral relapses have reduced in frequency and...
12	3	Sam Brown	12	2023-09-25	Personalized activities to address non-compliance ...	Client actively participates in 75% of group sessions.
13	1	John Doe	13	2023-10-10	Designed anger management strategies for aggre...	Client practices anger management techniques dur...

Likewise, we create the view of all the other tables. To create the view for the View for Incident Summaries, we use the following script:

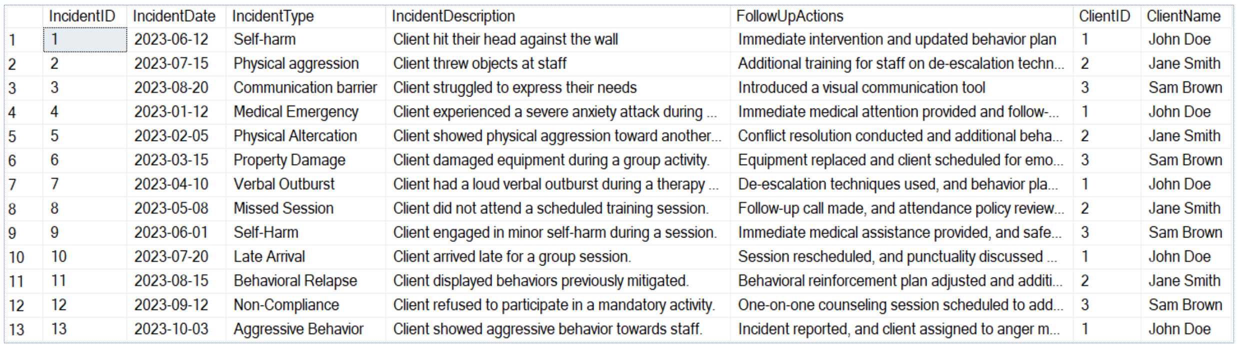


```sql
GO
CREATE VIEW IncidentSummaries AS
SELECT
    Incidents.IncidentID,
    Incidents.Date AS IncidentDate,
    Incidents.Type AS IncidentType,
    Incidents.Description AS IncidentDescription,
    Incidents.FollowUpActions AS FollowUpActions,
    Clients.ClientID,
    Clients.Name AS ClientName
FROM
    Incidents
INNER JOIN
    Clients ON Incidents.ClientID = Clients.ClientID;
GO
```



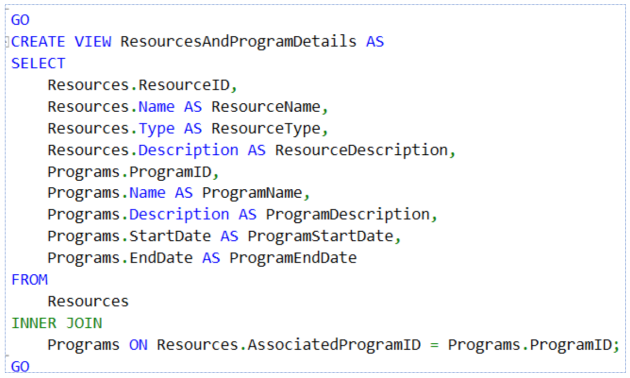
```sql
SELECT * FROM IncidentSummaries;
```

This view simplifies incident reporting by summarizing incident details by client.

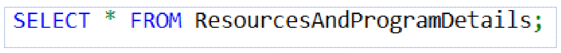


	IncidentID	IncidentDate	IncidentType	IncidentDescription	FollowUpActions	ClientID	ClientName
1	1	2023-06-12	Self-harm	Client hit their head against the wall	Immediate intervention and updated behavior plan	1	John Doe
2	2	2023-07-15	Physical aggression	Client threw objects at staff	Additional training for staff on de-escalation techn...	2	Jane Smith
3	3	2023-08-20	Communication barrier	Client struggled to express their needs	Introduced a visual communication tool	3	Sam Brown
4	4	2023-01-12	Medical Emergency	Client experienced a severe anxiety attack during ...	Immediate medical attention provided and follow-...	1	John Doe
5	5	2023-02-05	Physical Altercation	Client showed physical aggression toward another...	Conflict resolution conducted and additional beha...	2	Jane Smith
6	6	2023-03-15	Property Damage	Client damaged equipment during a group activity.	Equipment replaced and client scheduled for emo...	3	Sam Brown
7	7	2023-04-10	Verbal Outburst	Client had a loud verbal outburst during a therapy ...	De-escalation techniques used, and behavior pla...	1	John Doe
8	8	2023-05-08	Missed Session	Client did not attend a scheduled training session.	Follow-up call made, and attendance policy review...	2	Jane Smith
9	9	2023-06-01	Self-Harm	Client engaged in minor self-harm during a session.	Immediate medical assistance provided, and safe...	3	Sam Brown
10	10	2023-07-20	Late Arrival	Client arrived late for a group session.	Session rescheduled, and punctuality discussed ...	1	John Doe
11	11	2023-08-15	Behavioral Relapse	Client displayed behaviors previously mitigated.	Behavioral reinforcement plan adjusted and additi...	2	Jane Smith
12	12	2023-09-12	Non-Compliance	Client refused to participate in a mandatory activity.	One-on-one counseling session scheduled to add...	3	Sam Brown
13	13	2023-10-03	Aggressive Behavior	Client showed aggressive behavior towards staff.	Incident reported, and client assigned to anger m...	1	John Doe

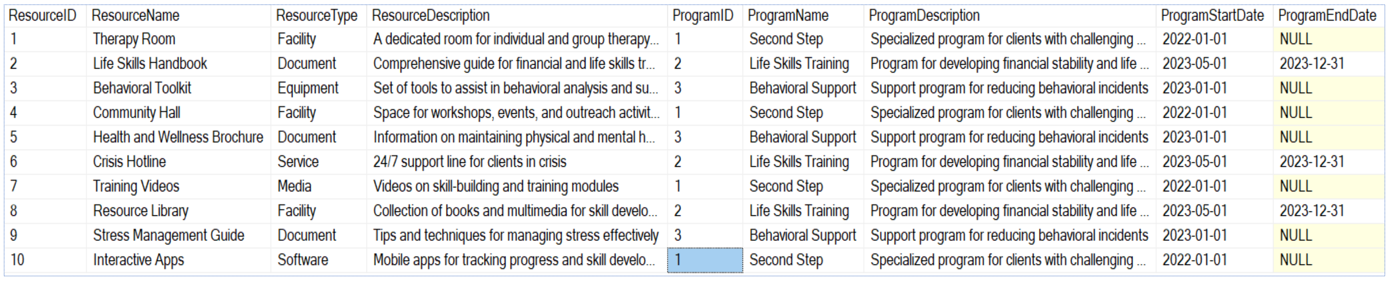
❖ Resources and Program Details View



```sql
GO
CREATE VIEW ResourcesAndProgramDetails AS
SELECT
    Resources.ResourceID,
    Resources.Name AS ResourceName,
    Resources.Type AS ResourceType,
    Resources.Description AS ResourceDescription,
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    Programs.Description AS ProgramDescription,
    Programs.StartDate AS ProgramStartDate,
    Programs.EndDate AS ProgramEndDate
FROM
    Resources
INNER JOIN
    Programs ON Resources.AssociatedProgramID = Programs.ProgramID;
GO
```



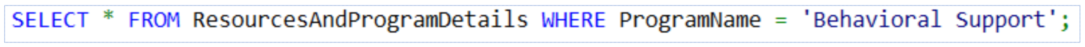
```sql
SELECT * FROM ResourcesAndProgramDetails;
```



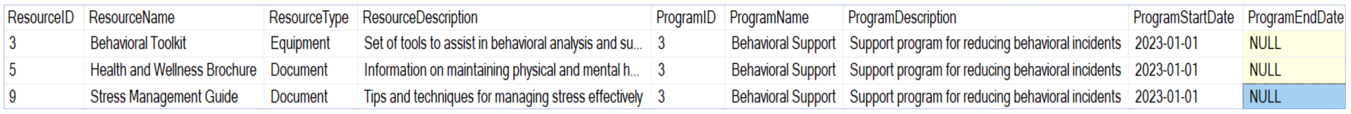
ResourceID	ResourceName	ResourceType	ResourceDescription	ProgramID	ProgramName	ProgramDescription	ProgramStartDate	ProgramEndDate
1	Therapy Room	Facility	A dedicated room for individual and group therapy...	1	Second Step	Specialized program for clients with challenging ...	2022-01-01	NULL
2	Life Skills Handbook	Document	Comprehensive guide for financial and life skills tr...	2	Life Skills Training	Program for developing financial stability and life ...	2023-05-01	2023-12-31
3	Behavioral Toolkit	Equipment	Set of tools to assist in behavioral analysis and su...	3	Behavioral Support	Support program for reducing behavioral incidents	2023-01-01	NULL
4	Community Hall	Facility	Space for workshops, events, and outreach activit...	1	Second Step	Specialized program for clients with challenging ...	2022-01-01	NULL
5	Health and Wellness Brochure	Document	Information on maintaining physical and mental h...	3	Behavioral Support	Support program for reducing behavioral incidents	2023-01-01	NULL
6	Crisis Hotline	Service	24/7 support line for clients in crisis	2	Life Skills Training	Program for developing financial stability and life ...	2023-05-01	2023-12-31
7	Training Videos	Media	Videos on skill-building and training modules	1	Second Step	Specialized program for clients with challenging ...	2022-01-01	NULL
8	Resource Library	Facility	Collection of books and multimedia for skill develo...	2	Life Skills Training	Program for developing financial stability and life ...	2023-05-01	2023-12-31
9	Stress Management Guide	Document	Tips and techniques for managing stress effectively	3	Behavioral Support	Support program for reducing behavioral incidents	2023-01-01	NULL
10	Interactive Apps	Software	Mobile apps for tracking progress and skill develo...	1	Second Step	Specialized program for clients with challenging ...	2022-01-01	NULL

The view provides information about resources and the programs they are associated with, including resource type and descriptions

✓ To filter resources for a specific program:



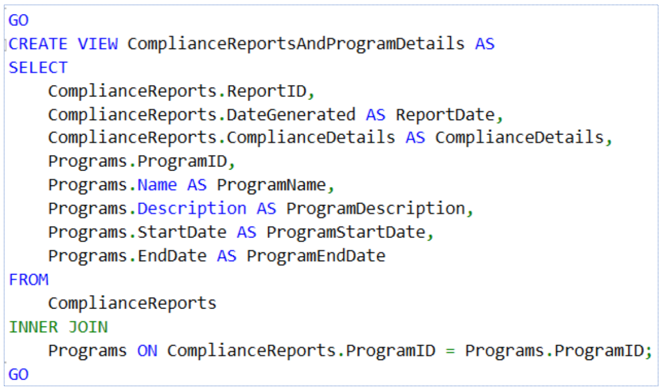
```sql
SELECT * FROM ResourcesAndProgramDetails WHERE ProgramName = 'Behavioral Support';
```



ResourceID	ResourceName	ResourceType	ResourceDescription	ProgramID	ProgramName	ProgramDescription	ProgramStartDate	ProgramEndDate
3	Behavioral Toolkit	Equipment	Set of tools to assist in behavioral analysis and su...	3	Behavioral Support	Support program for reducing behavioral incidents	2023-01-01	NULL
5	Health and Wellness Brochure	Document	Information on maintaining physical and mental h...	3	Behavioral Support	Support program for reducing behavioral incidents	2023-01-01	NULL
9	Stress Management Guide	Document	Tips and techniques for managing stress effectively	3	Behavioral Support	Support program for reducing behavioral incidents	2023-01-01	NULL

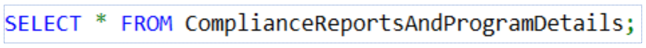
This view makes it easy to analyze the relationship between resources and their associated programs

❖ Compliance Reports and Program Details

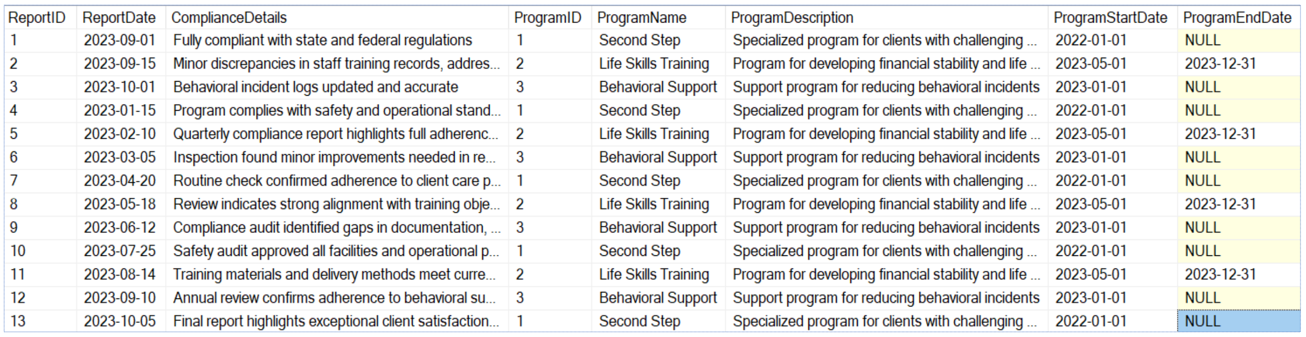


```sql
GO
CREATE VIEW ComplianceReportsAndProgramDetails AS
SELECT
    ComplianceReports.ReportID,
    ComplianceReports.DateGenerated AS ReportDate,
    ComplianceReports.ComplianceDetails AS ComplianceDetails,
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    Programs.Description AS ProgramDescription,
    Programs.StartDate AS ProgramStartDate,
    Programs.EndDate AS ProgramEndDate
FROM
    ComplianceReports
INNER JOIN
    Programs ON ComplianceReports.ProgramID = Programs.ProgramID;
GO
```

This view provides a comprehensive dataset that shows compliance reports along with the details of the associated programs, making it easy to track compliance status across different programs.



```sql
SELECT * FROM ComplianceReportsAndProgramDetails;
```



ReportID	ReportDate	ComplianceDetails	ProgramID	ProgramName	ProgramDescription	ProgramStartDate	ProgramEndDate
1	2023-09-01	Fully compliant with state and federal regulations	1	Second Step	Specialized program for clients with challenging ...	2022-01-01	NULL
2	2023-09-15	Minor discrepancies in staff training records, addres...	2	Life Skills Training	Program for developing financial stability and life ...	2023-05-01	2023-12-31
3	2023-10-01	Behavioral incident logs updated and accurate	3	Behavioral Support	Support program for reducing behavioral incidents	2023-01-01	NULL
4	2023-01-15	Program complies with safety and operational stand...	1	Second Step	Specialized program for clients with challenging ...	2022-01-01	NULL
5	2023-02-10	Quarterly compliance report highlights full adherenc...	2	Life Skills Training	Program for developing financial stability and life ...	2023-05-01	2023-12-31
6	2023-03-05	Inspection found minor improvements needed in re...	3	Behavioral Support	Support program for reducing behavioral incidents	2023-01-01	NULL
7	2023-04-20	Routine check confirmed adherence to client care p...	1	Second Step	Specialized program for clients with challenging ...	2022-01-01	NULL
8	2023-05-18	Review indicates strong alignment with training obje...	2	Life Skills Training	Program for developing financial stability and life ...	2023-05-01	2023-12-31
9	2023-06-12	Compliance audit identified gaps in documentation, ...	3	Behavioral Support	Support program for reducing behavioral incidents	2023-01-01	NULL
10	2023-07-25	Safety audit approved all facilities and operational p...	1	Second Step	Specialized program for clients with challenging ...	2022-01-01	NULL
11	2023-08-14	Training materials and delivery methods meet curre...	2	Life Skills Training	Program for developing financial stability and life ...	2023-05-01	2023-12-31
12	2023-09-10	Annual review confirms adherence to behavioral su...	3	Behavioral Support	Support program for reducing behavioral incidents	2023-01-01	NULL
13	2023-10-05	Final report highlights exceptional client satisfaction...	1	Second Step	Specialized program for clients with challenging ...	2022-01-01	NULL

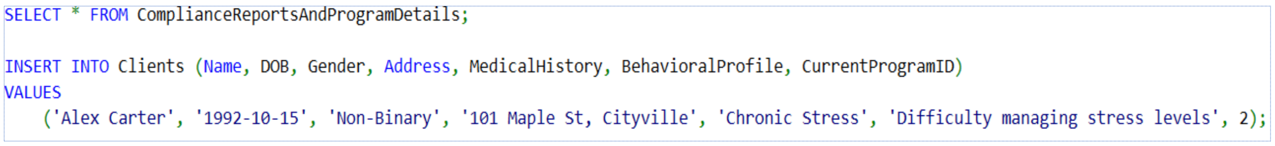
c. Testing and Deployment

Through the process of testing, we ensure that the database system meets all functional, performance, and security requirements. For that step, we perform a variety of tests should:

- ✓ Functional Testing: We verify that all tables, views, and stored procedures work as expected.
- ✓ We do the Test CRUD (Create, Read, Update, Delete) operations on each table.
- ✓ Validate relationships between tables, such as foreign key constraints.
- ✓ Test all views by running SELECT queries to ensure they return accurate data.

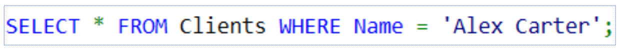
5.4.1 A few series of Tests

First, we insert a new client into the Clients table and assign them to an existing program using the CurrentProgramID.

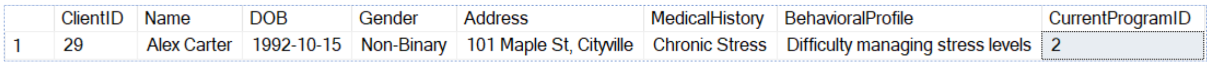


```
SELECT * FROM ComplianceReportsAndProgramDetails;

INSERT INTO Clients (Name, DOB, Gender, Address, MedicalHistory, BehavioralProfile, CurrentProgramID)
VALUES
    ('Alex Carter', '1992-10-15', 'Non-Binary', '101 Maple St, Cityville', 'Chronic Stress', 'Difficulty managing stress levels', 2);
```

To confirm the client has been added and assigned to the correct program:

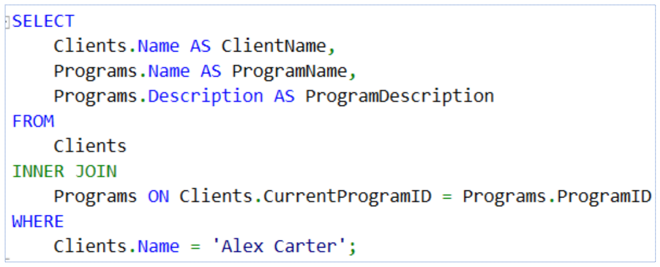


```
SELECT * FROM Clients WHERE Name = 'Alex Carter';
```



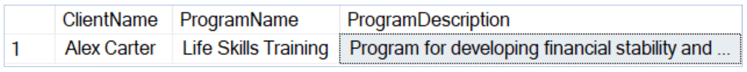
	ClientID	Name	DOB	Gender	Address	MedicalHistory	BehavioralProfile	CurrentProgramID
1	29	Alex Carter	1992-10-15	Non-Binary	101 Maple St, Cityville	Chronic Stress	Difficulty managing stress levels	2

To verify the program details assigned to the client:



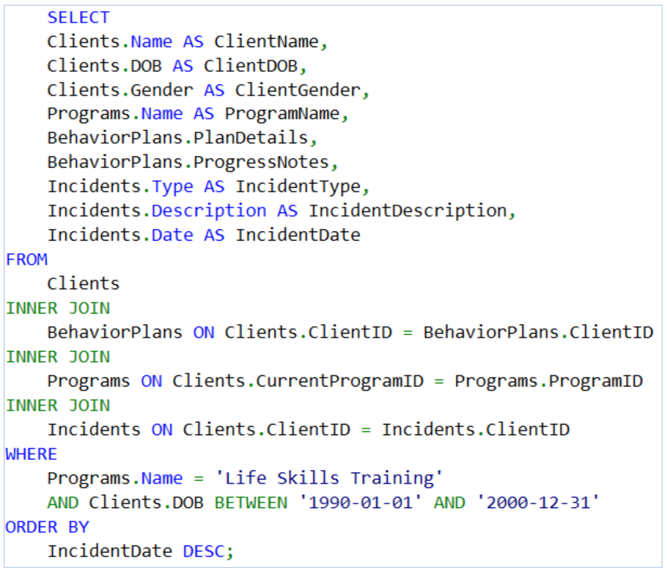
```
SELECT
    Clients.Name AS ClientName,
    Programs.Name AS ProgramName,
    Programs.Description AS ProgramDescription
FROM
    Clients
INNER JOIN
    Programs ON Clients.CurrentProgramID = Programs.ProgramID
WHERE
    Clients.Name = 'Alex Carter';
```

This test ensures the new client is successfully added and properly linked to an existing program via the CurrentProgramID.



	ClientName	ProgramName	ProgramDescription
1	Alex Carter	Life Skills Training	Program for developing financial stability and ...

To test the Performance of the database, we run at least one complex query. For instance, we can run a query involving multiple joins and large datasets.

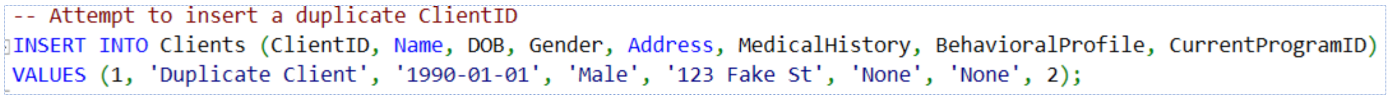


```sql
    SELECT
    Clients.Name AS ClientName,
    Clients.DOB AS ClientDOB,
    Clients.Gender AS ClientGender,
    Programs.Name AS ProgramName,
    BehaviorPlans.PlanDetails,
    BehaviorPlans.ProgressNotes,
    Incidents.Type AS IncidentType,
    Incidents.Description AS IncidentDescription,
    Incidents.Date AS IncidentDate
FROM
    Clients
INNER JOIN
    BehaviorPlans ON Clients.ClientID = BehaviorPlans.ClientID
INNER JOIN
    Programs ON Clients.CurrentProgramID = Programs.ProgramID
INNER JOIN
    Incidents ON Clients.ClientID = Incidents.ClientID
WHERE
    Programs.Name = 'Life Skills Training'
    AND Clients.DOB BETWEEN '1990-01-01' AND '2000-12-31'
ORDER BY
    IncidentDate DESC;
```

- **Data Integrity Testing**

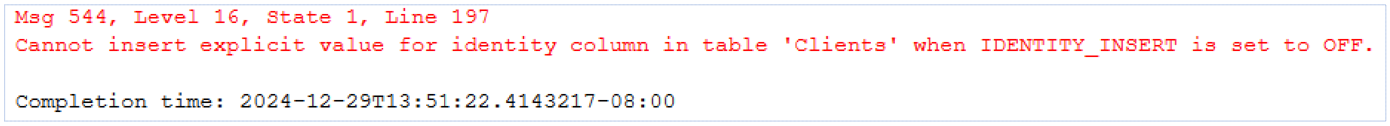
Data integrity testing ensures that the database maintains accuracy, consistency, and validity of the data. The focus is to verify that the implemented constraints (like primary keys, foreign keys, unique constraints, and NOT NULL constraints) are working correctly and that no invalid data can be inserted.

- Primary Key Constraint Test



```sql
-- Attempt to insert a duplicate ClientID
INSERT INTO Clients (ClientID, Name, DOB, Gender, Address, MedicalHistory, BehavioralProfile, CurrentProgramID)
VALUES (1, 'Duplicate Client', '1990-01-01', 'Male', '123 Fake St', 'None', 'None', 2);
```

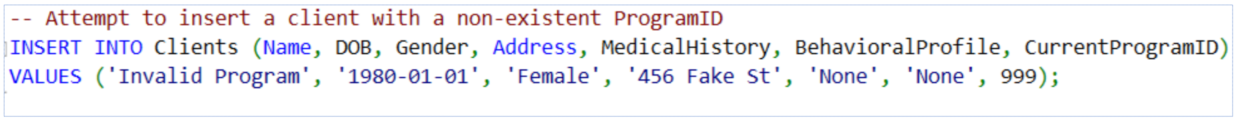
We obtain the following error message:



```
Msg 544, Level 16, State 1, Line 197
Cannot insert explicit value for identity column in table 'Clients' when IDENTITY_INSERT is set to OFF.

Completion time: 2024-12-29T13:51:22.4143217-08:00
```

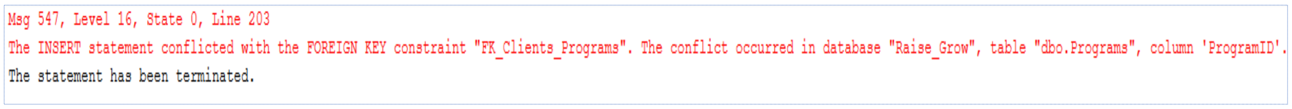
The error indicates that you are attempting to explicitly insert a value into an identity column (ClientID in this case) while the IDENTITY_INSERT setting for the table is turned off. By default, the identity column automatically generates values, so explicit inserts are not allowed unless you temporarily enable IDENTITY_INSERT.

- **Foreign Key Constraint Testing :** Ensure invalid CurrentProgramID values cannot be inserted into the Clients table.



```sql
-- Attempt to insert a client with a non-existent ProgramID
INSERT INTO Clients (Name, DOB, Gender, Address, MedicalHistory, BehavioralProfile, CurrentProgramID)
VALUES ('Invalid Program', '1980-01-01', 'Female', '456 Fake St', 'None', 'None', 999);
```

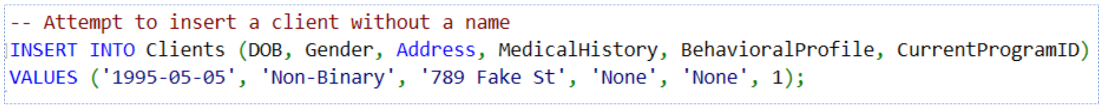
The output shows the following error:



```
Msg 547, Level 16, State 0, Line 203
The INSERT statement conflicted with the FOREIGN KEY constraint "FK_Clients_Programs". The conflict occurred in database "Raise_Grow", table "dbo.Programs", column 'ProgramID'.
The statement has been terminated.
```

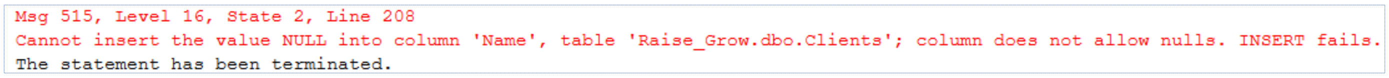
○ NOT NULL Constraint Test

This test verifies that NOT NULL columns cannot have null values.



```sql
-- Attempt to insert a client without a name
INSERT INTO Clients (DOB, Gender, Address, MedicalHistory, BehavioralProfile, CurrentProgramID)
VALUES ('1995-05-05', 'Non-Binary', '789 Fake St', 'None', 'None', 1);
```

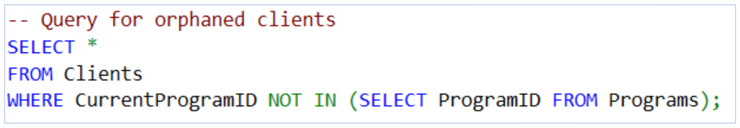
The output shows the following error preventing duplicate entries.



```
Msg 515, Level 16, State 2, Line 208
Cannot insert the value NULL into column 'Name', table 'Raise_Grow.dbo.Clients'; column does not allow nulls. INSERT fails.
The statement has been terminated.
```

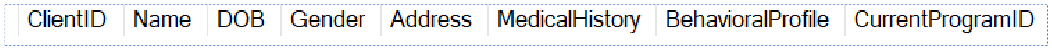
○ Data Consistency Test

This test checks for orphaned records (e.g., clients without a valid program).



```sql
-- Query for orphaned clients
SELECT *
FROM Clients
WHERE CurrentProgramID NOT IN (SELECT ProgramID FROM Programs);
```

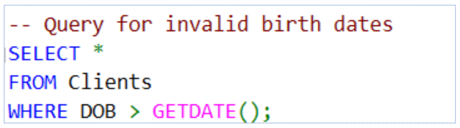
The query returns no results if referential integrity is maintained (which is an expected output)



ClientID	Name	DOB	Gender	Address	MedicalHistory	BehavioralProfile	CurrentProgramID

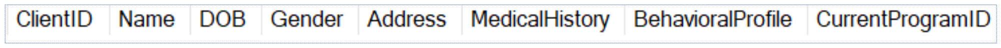
○ Data Accuracy Test

This test validates data with domain rules, such as DOB being in the past.



```sql
-- Query for invalid birth dates
SELECT *
FROM Clients
WHERE DOB > GETDATE();
```

The query returns no results as expected.

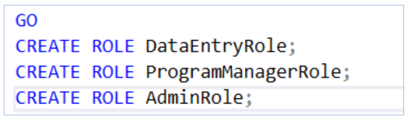


ClientID	Name	DOB	Gender	Address	MedicalHistory	BehavioralProfile	CurrentProgramID

The purpose of these tests is to ensure that data remains accurate and consistent, constraints are properly implemented and enforced, and relationships between tables are valid. Conducting these tests after data insertion and periodically during database use helps maintain data integrity and ensures the database functions as intended.

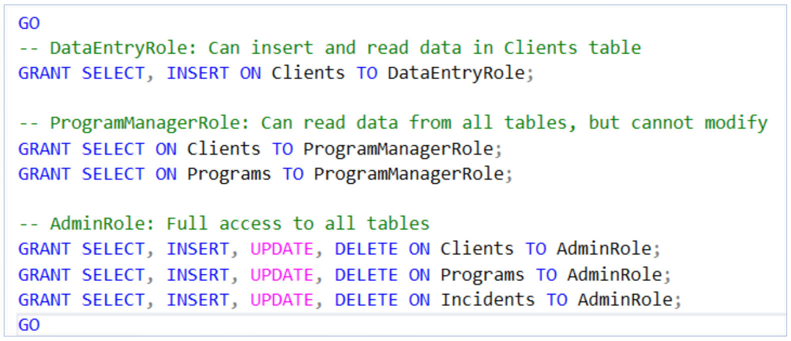
○ Security Testing

The first security test is to Validate “Role-Based Access Control”, that is, ensure that users have appropriate permissions based on their roles and are restricted from accessing unauthorized data or performing unauthorized actions. For this, we create or define roles with specific access rights. For example:



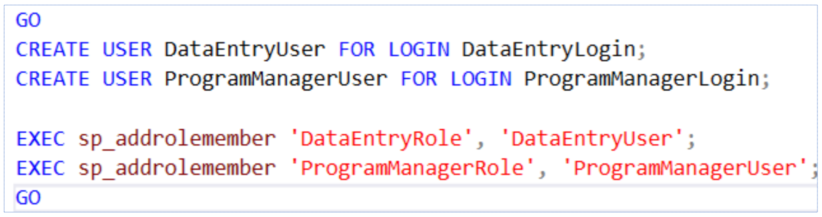
```sql
GO
CREATE ROLE DataEntryRole;
CREATE ROLE ProgramManagerRole;
CREATE ROLE AdminRole;
```

Then, we assign or grant permission to each role on the database objects:



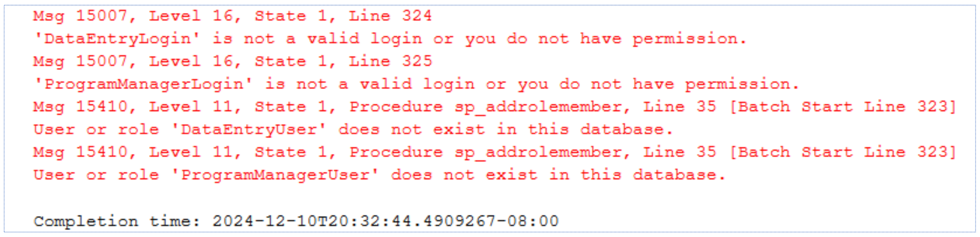
```sql
GO
-- DataEntryRole: Can insert and read data in Clients table
GRANT SELECT, INSERT ON Clients TO DataEntryRole;

-- ProgramManagerRole: Can read data from all tables, but cannot modify
GRANT SELECT ON Clients TO ProgramManagerRole;
GRANT SELECT ON Programs TO ProgramManagerRole;

-- AdminRole: Full access to all tables
GRANT SELECT, INSERT, UPDATE, DELETE ON Clients TO AdminRole;
GRANT SELECT, INSERT, UPDATE, DELETE ON Programs TO AdminRole;
GRANT SELECT, INSERT, UPDATE, DELETE ON Incidents TO AdminRole;
GO
```

Then, we assign Roles to Users, that is, we add database users and assign them to roles:



```sql
GO
CREATE USER DataEntryUser FOR LOGIN DataEntryLogin;
CREATE USER ProgramManagerUser FOR LOGIN ProgramManagerLogin;

EXEC sp_addrolemember 'DataEntryRole', 'DataEntryUser';
EXEC sp_addrolemember 'ProgramManagerRole', 'ProgramManagerUser';
GO
```

We obtain the following error message



```
Msg 15007, Level 16, State 1, Line 324
'DataEntryLogin' is not a valid login or you do not have permission.
Msg 15007, Level 16, State 1, Line 325
'ProgramManagerLogin' is not a valid login or you do not have permission.
Msg 15410, Level 11, State 1, Procedure sp_addrolemember, Line 35 [Batch Start Line 323]
User or role 'DataEntryUser' does not exist in this database.
Msg 15410, Level 11, State 1, Procedure sp_addrolemember, Line 35 [Batch Start Line 323]
User or role 'ProgramManagerUser' does not exist in this database.

Completion time: 2024-12-10T20:32:44.4909267-08:00
```

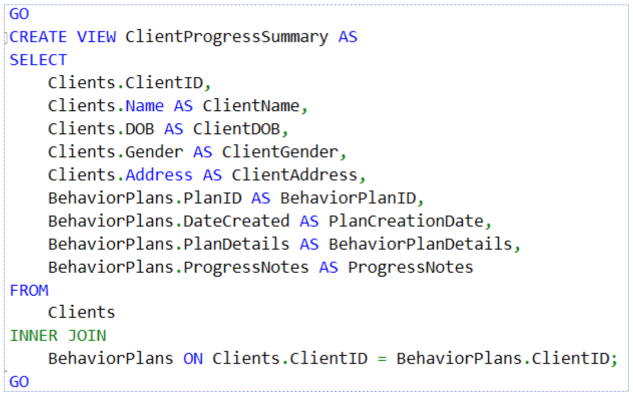
3. Sample Problems Scenarios that could be addressed Using this Database

The database is designed to address various operational challenges and support advanced analytics by providing structured, queryable data. By leveraging SQL's powerful features, the database enables users to extract valuable insights, automate processes, and enhance decision-making. This section presents real-world scenarios where SQL techniques such as joins, filtering, subqueries, and aggregate functions solve complex problems, demonstrating the database's utility in streamlining operations and driving data-driven outcomes.

6.1) Monitoring Client Progress

Scenario: Case managers need a summary of clients' behavioral progress, including details of their behavior plans and progress notes.

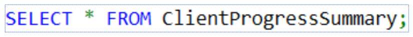
Solution: Use a query or view to consolidate information from the Clients and BehaviorPlans tables.



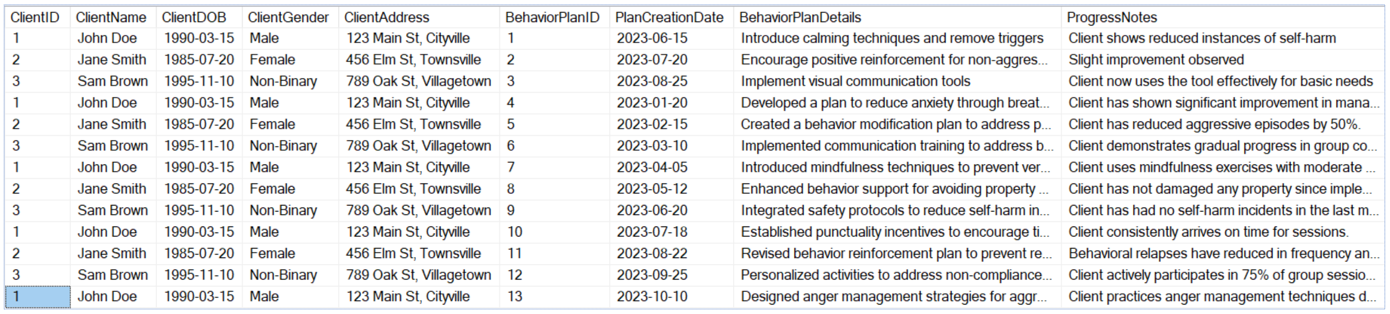
```
GO
CREATE VIEW ClientProgressSummary AS
SELECT
    Clients.ClientID,
    Clients.Name AS ClientName,
    Clients.DOB AS ClientDOB,
    Clients.Gender AS ClientGender,
    Clients.Address AS ClientAddress,
    BehaviorPlans.PlanID AS BehaviorPlanID,
    BehaviorPlans.DateCreated AS PlanCreationDate,
    BehaviorPlans.PlanDetails AS BehaviorPlanDetails,
    BehaviorPlans.ProgressNotes AS ProgressNotes
FROM
    Clients
INNER JOIN
    BehaviorPlans ON Clients.ClientID = BehaviorPlans.ClientID;
GO
```

Usage:

- To view all client progress summaries:

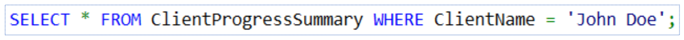


```
SELECT * FROM ClientProgressSummary;
```

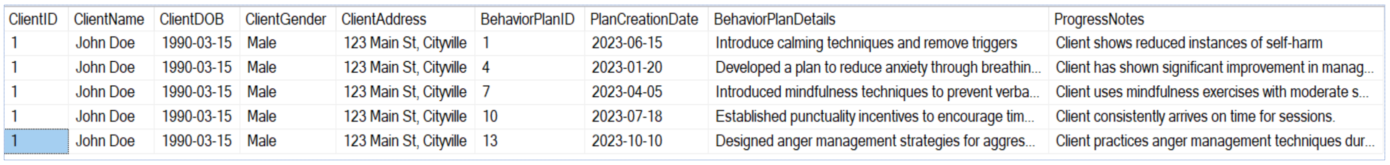


ClientID	ClientName	ClientDOB	ClientGender	ClientAddress	BehaviorPlanID	PlanCreationDate	BehaviorPlanDetails	ProgressNotes
1	John Doe	1990-03-15	Male	123 Main St, Cityville	1	2023-06-15	Introduce calming techniques and remove triggers	Client shows reduced instances of self-harm
2	Jane Smith	1985-07-20	Female	456 Elm St, Townsville	2	2023-07-20	Encourage positive reinforcement for non-aggres...	Slight improvement observed
3	Sam Brown	1995-11-10	Non-Binary	789 Oak St, Villagetown	3	2023-08-25	Implement visual communication tools	Client now uses the tool effectively for basic needs
1	John Doe	1990-03-15	Male	123 Main St, Cityville	4	2023-01-20	Developed a plan to reduce anxiety through breat...	Client has shown significant improvement in mana...
2	Jane Smith	1985-07-20	Female	456 Elm St, Townsville	5	2023-02-15	Created a behavior modification plan to address p...	Client has reduced aggressive episodes by 50%.
3	Sam Brown	1995-11-10	Non-Binary	789 Oak St, Villagetown	6	2023-03-10	Implemented communication training to address b...	Client demonstrates gradual progress in group co...
1	John Doe	1990-03-15	Male	123 Main St, Cityville	7	2023-04-05	Introduced mindfulness techniques to prevent ver...	Client uses mindfulness exercises with moderate ...
2	Jane Smith	1985-07-20	Female	456 Elm St, Townsville	8	2023-05-12	Enhanced behavior support for avoiding property ...	Client has not damaged any property since imple...
3	Sam Brown	1995-11-10	Non-Binary	789 Oak St, Villagetown	9	2023-06-20	Integrated safety protocols to reduce self-harm in...	Client has had no self-harm incidents in the last m...
1	John Doe	1990-03-15	Male	123 Main St, Cityville	10	2023-07-18	Established punctuality incentives to encourage ti...	Client consistently arrives on time for sessions.
2	Jane Smith	1985-07-20	Female	456 Elm St, Townsville	11	2023-08-22	Revised behavior reinforcement plan to prevent re...	Behavioral relapses have reduced in frequency an...
3	Sam Brown	1995-11-10	Non-Binary	789 Oak St, Villagetown	12	2023-09-25	Personalized activities to address non-compliance...	Client actively participates in 75% of group sessio...
1	John Doe	1990-03-15	Male	123 Main St, Cityville	13	2023-10-10	Designed anger management strategies for aggr...	Client practices anger management techniques d...

- To filter progress for a specific client:



```
SELECT * FROM ClientProgressSummary WHERE ClientName = 'John Doe';
```

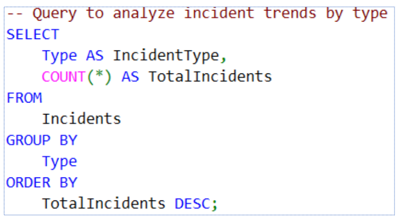


ClientID	ClientName	ClientDOB	ClientGender	ClientAddress	BehaviorPlanID	PlanCreationDate	BehaviorPlanDetails	ProgressNotes
1	John Doe	1990-03-15	Male	123 Main St, Cityville	1	2023-06-15	Introduce calming techniques and remove triggers	Client shows reduced instances of self-harm
1	John Doe	1990-03-15	Male	123 Main St, Cityville	4	2023-01-20	Developed a plan to reduce anxiety through breathin...	Client has shown significant improvement in manag...
1	John Doe	1990-03-15	Male	123 Main St, Cityville	7	2023-04-05	Introduced mindfulness techniques to prevent verba...	Client uses mindfulness exercises with moderate s...
1	John Doe	1990-03-15	Male	123 Main St, Cityville	10	2023-07-18	Established punctuality incentives to encourage tim...	Client consistently arrives on time for sessions.
1	John Doe	1990-03-15	Male	123 Main St, Cityville	13	2023-10-10	Designed anger management strategies for aggres...	Client practices anger management techniques dur...

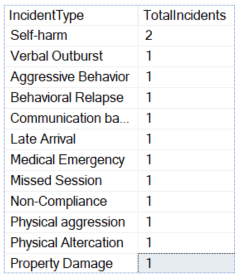
6.2) Incident Trends Analysis

Scenario: The program manager needs to identify clients with frequent incidents and analyze common types of incidents across programs to improve interventions and resource allocation.

- **Query to analyze incident trends by type**

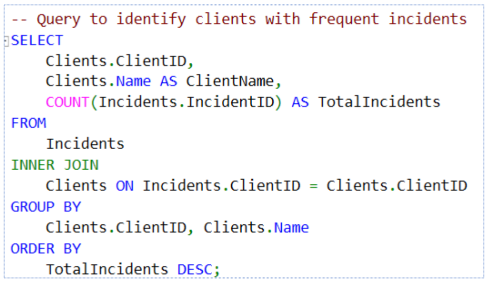


```
-- Query to analyze incident trends by type
SELECT
    Type AS IncidentType,
    COUNT(*) AS TotalIncidents
FROM
    Incidents
GROUP BY
    Type
ORDER BY
    TotalIncidents DESC;
```

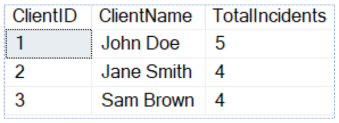


IncidentType	TotalIncidents
Self-harm	2
Verbal Outburst	1
Aggressive Behavior	1
Behavioral Relapse	1
Communication ba...	1
Late Arrival	1
Medical Emergency	1
Missed Session	1
Non-Compliance	1
Physical aggression	1
Physical Altercation	1
Property Damage	1

- **Query to identify clients with frequent incidents**



```
-- Query to identify clients with frequent incidents
SELECT
    Clients.ClientID,
    Clients.Name AS ClientName,
    COUNT(Incidents.IncidentID) AS TotalIncidents
FROM
    Incidents
INNER JOIN
    Clients ON Incidents.ClientID = Clients.ClientID
GROUP BY
    Clients.ClientID, Clients.Name
ORDER BY
    TotalIncidents DESC;
```



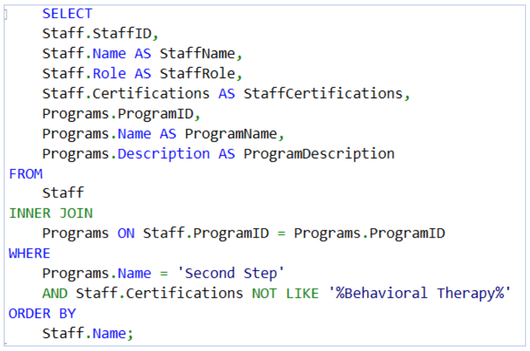
ClientID	ClientName	TotalIncidents
1	John Doe	5
2	Jane Smith	4
3	Sam Brown	4

6.3) Staff Certification Tracking

Scenario: Ensure that staff members working in specialized programs (e.g., Second Step) hold the required certifications for their roles.

Solution:

We query the Staff and Programs tables to verify if certifications match program requirements by identifying missing or mismatched certifications

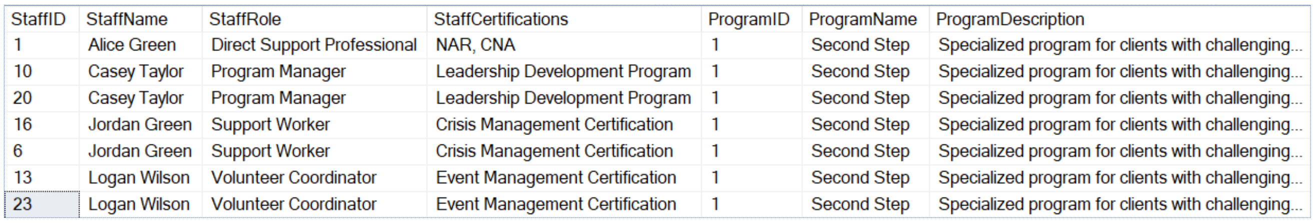


```
    SELECT
    Staff.StaffID,
    Staff.Name AS StaffName,
    Staff.Role AS StaffRole,
    Staff.Certifications AS StaffCertifications,
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    Programs.Description AS ProgramDescription
FROM
    Staff
INNER JOIN
    Programs ON Staff.ProgramID = Programs.ProgramID
WHERE
    Programs.Name = 'Second Step'
    AND Staff.Certifications NOT LIKE '%Behavioral Therapy%'
ORDER BY
    Staff.Name;
```

Explanation:

- **Joins**: The query joins the Staff table with the Programs table using the ProgramID foreign key to identify which staff members are assigned to which programs.
- **Filter**: It filters for staff assigned to the specialized program Second Step.

- It checks if staff certifications do not include the required certification (e.g., %Behavioral Therapy%).
- **Columns**: Displays staff details (StaffID, Name, Role, Certifications) alongside their assigned program details (ProgramID, ProgramName, ProgramDescription).
- **Ordering**: Results are ordered alphabetically by staff name for readability.



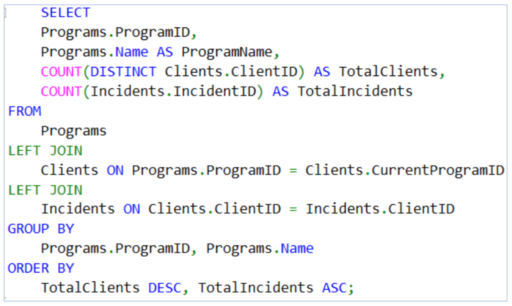
StaffID	StaffName	StaffRole	StaffCertifications	ProgramID	ProgramName	ProgramDescription
1	Alice Green	Direct Support Professional	NAR, CNA	1	Second Step	Specialized program for clients with challenging...
10	Casey Taylor	Program Manager	Leadership Development Program	1	Second Step	Specialized program for clients with challenging...
20	Casey Taylor	Program Manager	Leadership Development Program	1	Second Step	Specialized program for clients with challenging...
16	Jordan Green	Support Worker	Crisis Management Certification	1	Second Step	Specialized program for clients with challenging...
6	Jordan Green	Support Worker	Crisis Management Certification	1	Second Step	Specialized program for clients with challenging...
13	Logan Wilson	Volunteer Coordinator	Event Management Certification	1	Second Step	Specialized program for clients with challenging...
23	Logan Wilson	Volunteer Coordinator	Event Management Certification	1	Second Step	Specialized program for clients with challenging...

6.4) Evaluating Program Effectiveness

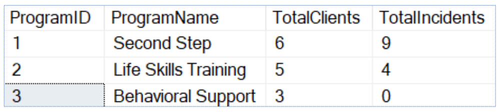
Scenario: Determine program effectiveness by evaluating client engagement. This involves analyzing the number of clients enrolled in each program and the frequency of incidents associated with those clients.

Solution:

Query the Clients, Programs, and Incidents tables to correlate program enrollment with the number of incidents, providing insights into client engagement and program outcomes.



```sql
    SELECT
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    COUNT(DISTINCT Clients.ClientID) AS TotalClients,
    COUNT(Incidents.IncidentID) AS TotalIncidents
FROM
    Programs
LEFT JOIN
    Clients ON Programs.ProgramID = Clients.CurrentProgramID
LEFT JOIN
    Incidents ON Clients.ClientID = Incidents.ClientID
GROUP BY
    Programs.ProgramID, Programs.Name
ORDER BY
    TotalClients DESC, TotalIncidents ASC;
```



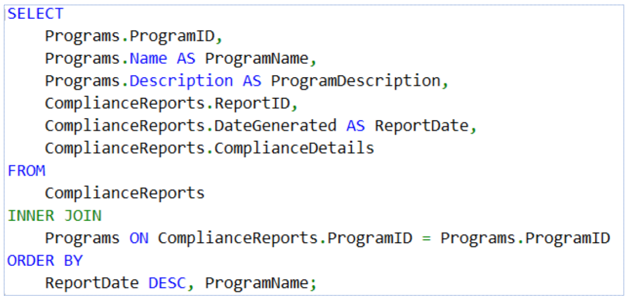
ProgramID	ProgramName	TotalClients	TotalIncidents
1	Second Step	6	9
2	Life Skills Training	5	4
3	Behavioral Support	3	0

Explanation:

- **Joins**: LEFT JOIN between Programs and Clients to matches programs with the clients enrolled in them.
- LEFT JOIN between Clients and Incidents to correlate clients with their associated incident reports.
- **Aggregations**: COUNT(DISTINCT Clients.ClientID): Counts the total number of unique clients enrolled in each program.

- COUNT(Incidents.IncidentID): Counts the total number of incidents reported for clients in each program.
- **Grouping**: Groups the results by ProgramID and ProgramName to aggregate data for each program.
- **Ordering**: Sorts programs by the highest number of clients (TotalClients DESC) and then by the lowest number of incidents (TotalIncidents ASC), emphasizing programs with high engagement and low issues.

Solution:

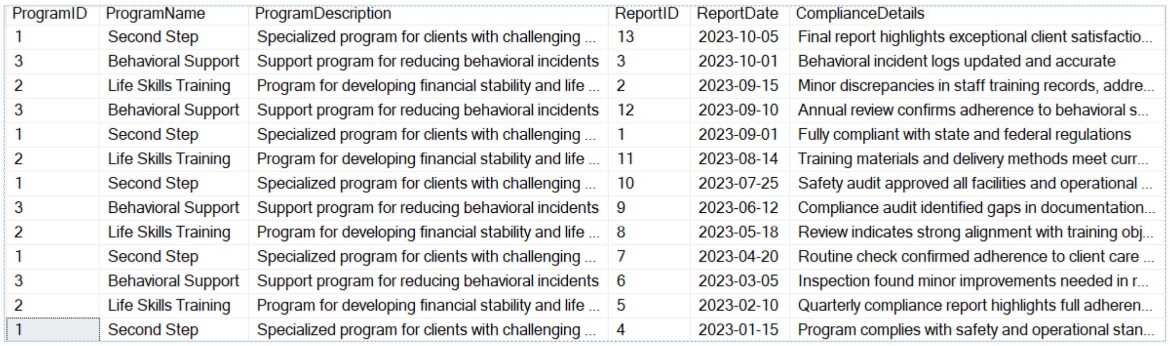


```
SELECT
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    Programs.Description AS ProgramDescription,
    ComplianceReports.ReportID,
    ComplianceReports.DateGenerated AS ReportDate,
    ComplianceReports.ComplianceDetails
FROM
    ComplianceReports
INNER JOIN
    Programs ON ComplianceReports.ProgramID = Programs.ProgramID
ORDER BY
    ReportDate DESC, ProgramName;
```

Explanation:

- **Joins**: Joins the ComplianceReports table with the Programs table using the ProgramID to associate each compliance report with the corresponding program.
- **Selected Columns**: From Programs, includes ProgramID, ProgramName, and ProgramDescription to provide context for each program.
- From ComplianceReports, includes ReportID, ReportDate, and ComplianceDetails to display detailed compliance information.
- **Ordering**: Results are sorted by the most recent compliance reports (ReportDate DESC) and then alphabetically by program name (ProgramName).

The output is as follows:



ProgramID	ProgramName	ProgramDescription	ReportID	ReportDate	ComplianceDetails
1	Second Step	Specialized program for clients with challenging ...	13	2023-10-05	Final report highlights exceptional client satisfactio...
3	Behavioral Support	Support program for reducing behavioral incidents	3	2023-10-01	Behavioral incident logs updated and accurate
2	Life Skills Training	Program for developing financial stability and life ...	2	2023-09-15	Minor discrepancies in staff training records, addre...
3	Behavioral Support	Support program for reducing behavioral incidents	12	2023-09-10	Annual review confirms adherence to behavioral s...
1	Second Step	Specialized program for clients with challenging ...	1	2023-09-01	Fully compliant with state and federal regulations
2	Life Skills Training	Program for developing financial stability and life ...	11	2023-08-14	Training materials and delivery methods meet curr...
1	Second Step	Specialized program for clients with challenging ...	10	2023-07-25	Safety audit approved all facilities and operational ...
3	Behavioral Support	Support program for reducing behavioral incidents	9	2023-06-12	Compliance audit identified gaps in documentation...
2	Life Skills Training	Program for developing financial stability and life ...	8	2023-05-18	Review indicates strong alignment with training obj...
1	Second Step	Specialized program for clients with challenging ...	7	2023-04-20	Routine check confirmed adherence to client care ...
3	Behavioral Support	Support program for reducing behavioral incidents	6	2023-03-05	Inspection found minor improvements needed in r...
2	Life Skills Training	Program for developing financial stability and life ...	5	2023-02-10	Quarterly compliance report highlights full adheren...
1	Second Step	Specialized program for clients with challenging ...	4	2023-01-15	Program complies with safety and operational stan...

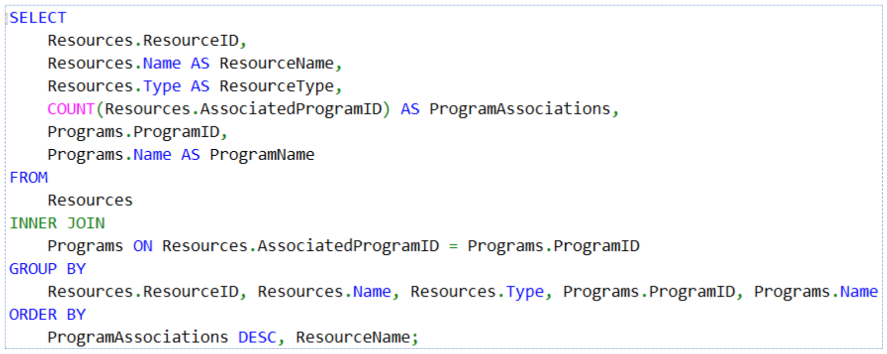
6.5) Resource Utilization

Scenario:

Identify the resources most commonly associated with programs to analyze their utilization and ensure efficient allocation.

Solution:

We query the Resources and Programs tables to find associations between resources and programs, and count how often each resource is linked to programs.



```
SELECT
    Resources.ResourceID,
    Resources.Name AS ResourceName,
    Resources.Type AS ResourceType,
    COUNT(Resources.AssociatedProgramID) AS ProgramAssociations,
    Programs.ProgramID,
    Programs.Name AS ProgramName
FROM
    Resources
INNER JOIN
    Programs ON Resources.AssociatedProgramID = Programs.ProgramID
GROUP BY
    Resources.ResourceID, Resources.Name, Resources.Type, Programs.ProgramID, Programs.Name
ORDER BY
    ProgramAssociations DESC, ResourceName;
```

Explanation:

- **Joins**: Combines the Resources table with the Programs table using AssociatedProgramID to determine which resources are associated with which programs.
- **Aggregations**: Uses COUNT(Resources.AssociatedProgramID) to calculate how many times a resource is associated with programs.
- **Grouping**: Groups the results by ResourceID, ResourceName, ResourceType, and program details (ProgramID, ProgramName) to aggregate data by resource and program association.
- **Ordering**: Sorts the results by the number of program associations (ProgramAssociations DESC) and then alphabetically by resource name for clarity.

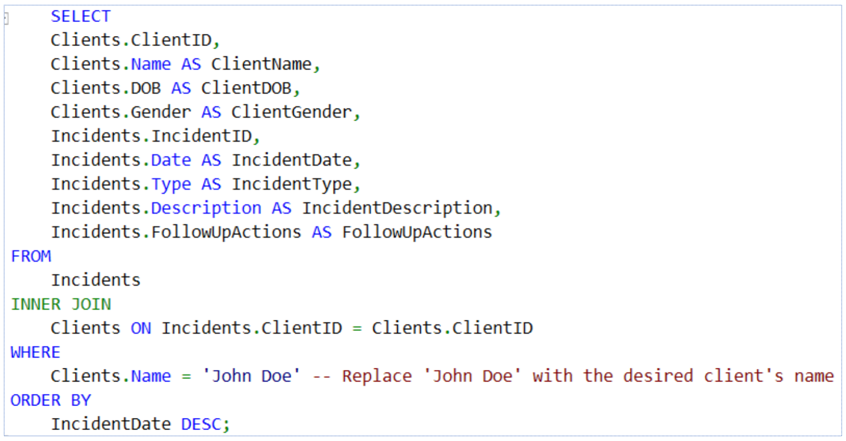
6.6) Tracking Client Incidents and Follow-Ups

Scenario:

Caseworkers need a detailed history of incidents and the corresponding follow-up actions for a specific client to monitor progress and address recurring issues.

Solution:

We query the Incidents and Clients tables to retrieve incident records and follow-up actions for a specific client.



```sql
    SELECT
    Clients.ClientID,
    Clients.Name AS ClientName,
    Clients.DOB AS ClientDOB,
    Clients.Gender AS ClientGender,
    Incidents.IncidentID,
    Incidents.Date AS IncidentDate,
    Incidents.Type AS IncidentType,
    Incidents.Description AS IncidentDescription,
    Incidents.FollowUpActions AS FollowUpActions
FROM
    Incidents
INNER JOIN
    Clients ON Incidents.ClientID = Clients.ClientID
WHERE
    Clients.Name = 'John Doe' -- Replace 'John Doe' with the desired client's name
ORDER BY
    IncidentDate DESC;
```

Explanation:

- **Joins**: Links the Incidents table with the Clients table using the ClientID foreign key to associate incidents with client details.
- **Columns**: Displays client information (ClientID, Name, DOB, Gender) along with incident details (IncidentID, Date, Type, Description) and follow-up actions.
- **Filters**: Filters the results to show data for a specific client by matching the client's name in the WHERE clause.
- **Ordering**: Sorts incidents in descending order of date (IncidentDate DESC) to show the most recent incidents first.

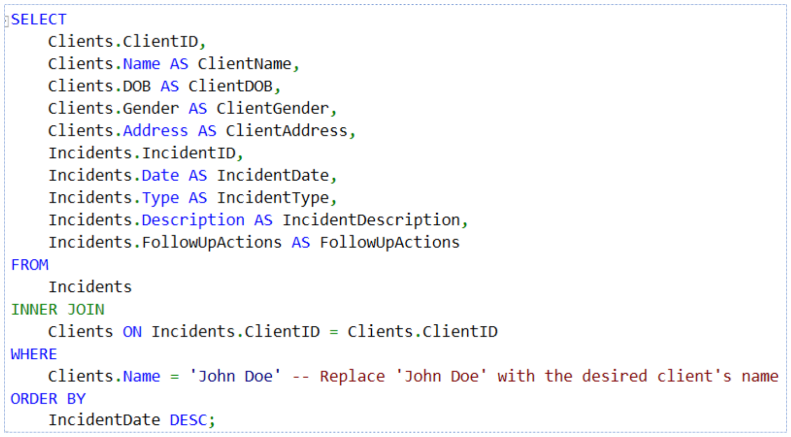
6.7) Tracking Client Incidents and Follow-Ups

Scenario:

Caseworkers require a detailed history of incidents and the corresponding follow-up actions for a specific client to track progress and manage recurring issues effectively.

Solution:

Use the following query to retrieve incident records and follow-up actions for a specific client by filtering data from the Incidents and Clients tables.



```sql
SELECT
    Clients.ClientID,
    Clients.Name AS ClientName,
    Clients.DOB AS ClientDOB,
    Clients.Gender AS ClientGender,
    Clients.Address AS ClientAddress,
    Incidents.IncidentID,
    Incidents.Date AS IncidentDate,
    Incidents.Type AS IncidentType,
    Incidents.Description AS IncidentDescription,
    Incidents.FollowUpActions AS FollowUpActions
FROM
    Incidents
INNER JOIN
    Clients ON Incidents.ClientID = Clients.ClientID
WHERE
    Clients.Name = 'John Doe' -- Replace 'John Doe' with the desired client's name
ORDER BY
    IncidentDate DESC;
```

Explanation:

- **Joins**: The INNER JOIN links the Incidents table with the Clients table based on the ClientID, ensuring that each incident is associated with its corresponding client.
- **Columns**: Includes client details like ClientID, Name, DOB, Gender, and Address.
- Lists incident details such as IncidentID, IncidentDate, IncidentType, Description, and FollowUpActions.
- **Filters**: The WHERE clause filters data to focus on a specific client by name (Clients.Name = 'John Doe'). This can be modified to filter by ClientID if needed.
- **Ordering**: Sorts results by IncidentDate in descending order (DESC) to display the most recent incidents first.

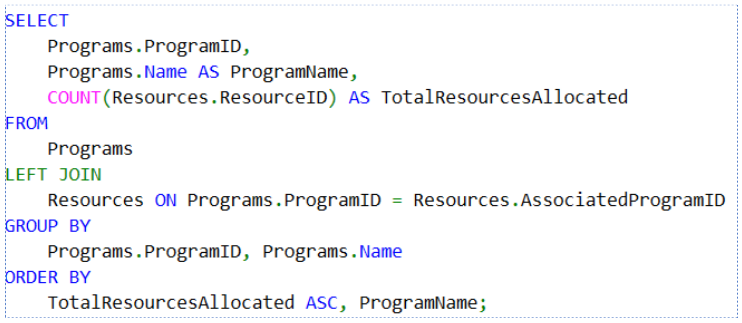
6.8) Resource Allocation Efficiency

Scenario:

Analyze how many resources are allocated to each program and identify potential gaps to improve resource allocation.

Solution:

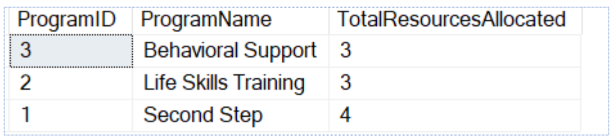
Use a grouped query to count the number of resources allocated to each program, providing insights into resource distribution.



```sql
SELECT
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    COUNT(Resources.ResourceID) AS TotalResourcesAllocated
FROM
    Programs
LEFT JOIN
    Resources ON Programs.ProgramID = Resources.AssociatedProgramID
GROUP BY
    Programs.ProgramID, Programs.Name
ORDER BY
    TotalResourcesAllocated ASC, ProgramName;
```

Explanation:

- **Joins**: A LEFT JOIN is used to include all programs, even those without allocated resources, ensuring potential gaps are identified.
- **Aggregations**: COUNT(Resources.ResourceID): Counts the number of resources allocated to each program.
- **Grouping**: Groups results by ProgramID and ProgramName to aggregate the resource count for each program.
- **Ordering**: Results are sorted in ascending order of Total Resources allocated to highlight programs with fewer resources, and then alphabetically by program name.



ProgramID	ProgramName	TotalResourcesAllocated
3	Behavioral Support	3
2	Life Skills Training	3
1	Second Step	4

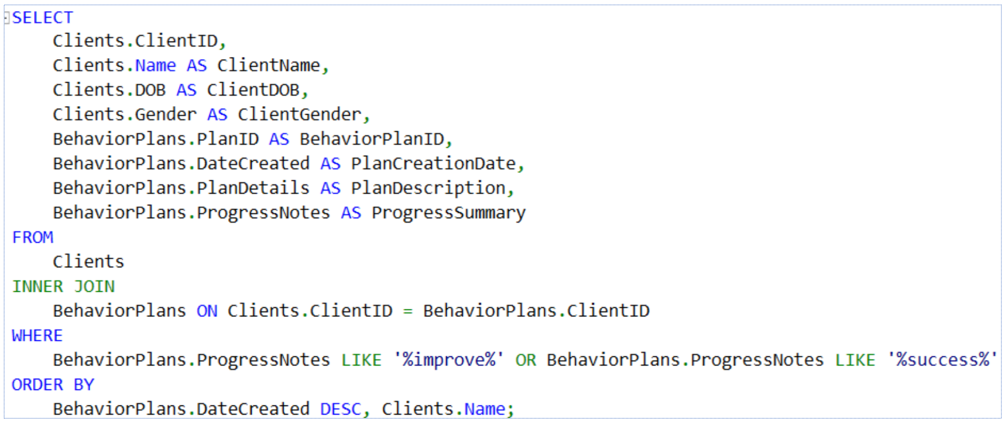
6.9) Behavior Plan Effectiveness

Scenario:

Measure the effectiveness of behavior plans by evaluating the progress notes of all clients, highlighting significant improvements.

Solution:

Use the following SQL query to analyze progress notes for all clients and summarize improvements based on their behavior plans.



```
SELECT
    Clients.ClientID,
    Clients.Name AS ClientName,
    Clients.DOB AS ClientDOB,
    Clients.Gender AS ClientGender,
    BehaviorPlans.PlanID AS BehaviorPlanID,
    BehaviorPlans.DateCreated AS PlanCreationDate,
    BehaviorPlans.PlanDetails AS PlanDescription,
    BehaviorPlans.ProgressNotes AS ProgressSummary
FROM
    Clients
INNER JOIN
    BehaviorPlans ON Clients.ClientID = BehaviorPlans.ClientID
WHERE
    BehaviorPlans.ProgressNotes LIKE '%improve%' OR BehaviorPlans.ProgressNotes LIKE '%success%'
ORDER BY
    BehaviorPlans.DateCreated DESC, Clients.Name;
```

Explanation:

- **Joins**: The INNER JOIN links the Clients table with the BehaviorPlans table using ClientID to associate clients with their behavior plans.
- **Filters**: The WHERE clause filters progress notes for keywords like "improve" or "success" to identify plans showing positive progress.
- **Columns**: Includes client details (ClientID, Name, DOB, Gender) and behavior plan details (PlanID, PlanCreationDate, PlanDescription, ProgressSummary).
- **Ordering**: Sorts results by the most recent plans (PlanCreationDate DESC) and then alphabetically by client name.

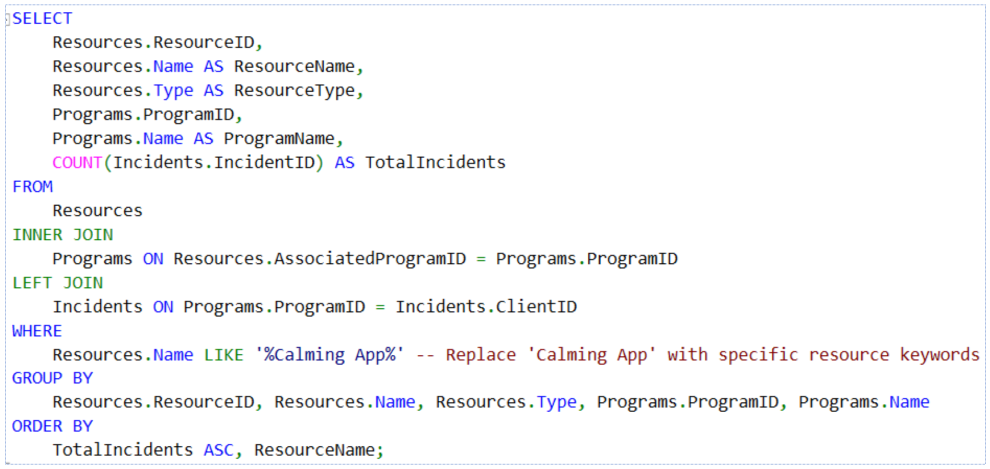
6.10) Incident and Resource Correlation

Scenario:

Evaluate if specific resources (e.g., calming apps) correlate with a reduction in incidents by analyzing the relationship between resource allocation and client incidents.

Solution:

Combine data from the Resources and Incidents tables, linking them through associated programs and analyzing the number of incidents for programs using specific resources.

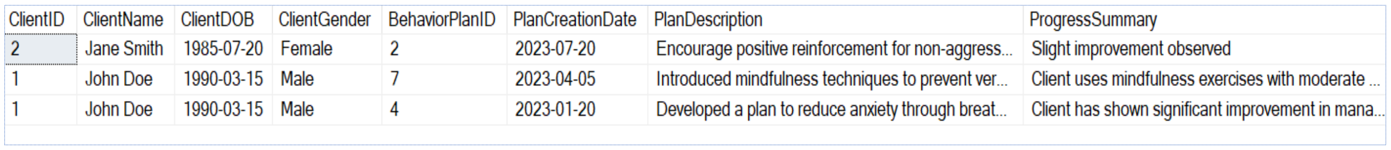


```
SELECT
    Resources.ResourceID,
    Resources.Name AS ResourceName,
    Resources.Type AS ResourceType,
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    COUNT(Incidents.IncidentID) AS TotalIncidents
FROM
    Resources
INNER JOIN
    Programs ON Resources.AssociatedProgramID = Programs.ProgramID
LEFT JOIN
    Incidents ON Programs.ProgramID = Incidents.ClientID
WHERE
    Resources.Name LIKE '%Calming App%' -- Replace 'Calming App' with specific resource keywords
GROUP BY
    Resources.ResourceID, Resources.Name, Resources.Type, Programs.ProgramID, Programs.Name
ORDER BY
    TotalIncidents ASC, ResourceName;
```

Explanation:

- **Joins**: Combines Resources and Programs using AssociatedProgramID to identify which resources are linked to each program.
- Links Programs with Incidents to analyze incidents associated with the programs.
- **Filters**: The WHERE clause filters for specific resources, such as calming apps, by using keywords (e.g., %Calming App%).
- **Aggregations**: COUNT(Incidents.IncidentID) calculates the total number of incidents for each program using a specific resource.
- **Grouping**: Groups results by ResourceID, ResourceName, ResourceType, and program details to provide insights into resource utilization and incident trends.
- **Ordering**: Sorts by the total number of incidents in ascending order (TotalIncidents ASC) to highlight resources correlated with fewer incidents.

The output is displayed below:



ClientID	ClientName	ClientDOB	ClientGender	BehaviorPlanID	PlanCreationDate	PlanDescription	ProgressSummary
2	Jane Smith	1985-07-20	Female	2	2023-07-20	Encourage positive reinforcement for non-aggress...	Slight improvement observed
1	John Doe	1990-03-15	Male	7	2023-04-05	Introduced mindfulness techniques to prevent ver...	Client uses mindfulness exercises with moderate ...
1	John Doe	1990-03-15	Male	4	2023-01-20	Developed a plan to reduce anxiety through breat...	Client has shown significant improvement in mana...

These scenarios illustrate how the database can be leveraged to address real-world operational challenges, streamline decision-making, and improve service delivery.

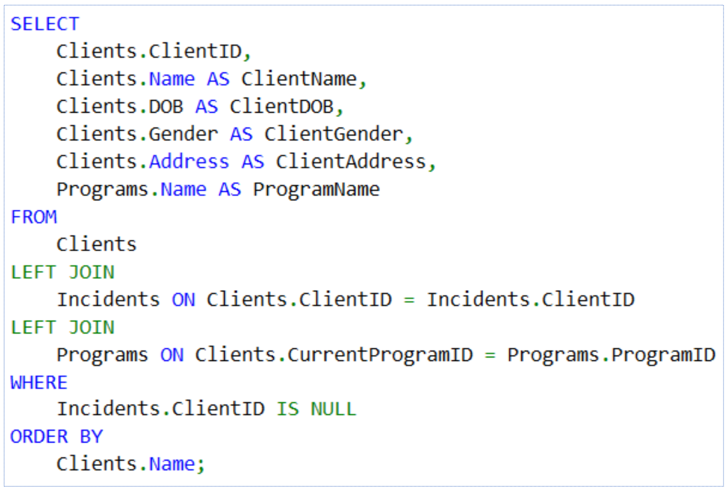
6.11) Identify Clients with No Incidents

Scenario:

Case managers need to identify clients who have not been involved in any reported incidents, allowing them to focus resources on clients requiring more support.

Solution:

Use a query to find clients in the Clients table who do not have matching entries in the Incidents table.



```sql
SELECT
    Clients.ClientID,
    Clients.Name AS ClientName,
    Clients.DOB AS ClientDOB,
    Clients.Gender AS ClientGender,
    Clients.Address AS ClientAddress,
    Programs.Name AS ProgramName
FROM
    Clients
LEFT JOIN
    Incidents ON Clients.ClientID = Incidents.ClientID
LEFT JOIN
    Programs ON Clients.CurrentProgramID = Programs.ProgramID
WHERE
    Incidents.ClientID IS NULL
ORDER BY
    Clients.Name;
```

Explanation:

- **Joins**: LEFT JOIN connects Clients with Incidents to include all clients, even those without incident records.
- LEFT JOIN links Clients with Programs to include program details for each client.
- **Filters**: WHERE Incidents.ClientID IS NULL: Ensures only clients with no matching incidents in the Incidents table are included.
- **Columns**: Includes client details (ClientID, Name, DOB, Gender, Address) and their associated program name for context.
- **Ordering**: Sorts the results alphabetically by client name (Clients.Name).

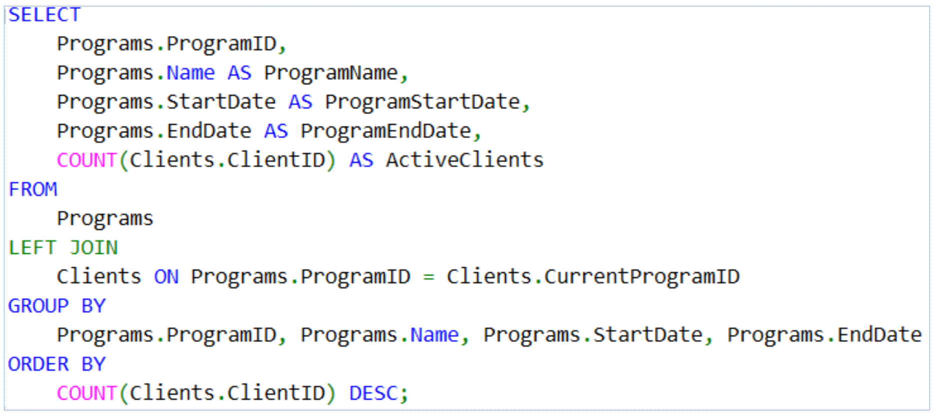
6.12) Program Utilization Trends

Scenario:

The administration needs to monitor the number of active clients enrolled in each program over time to track utilization trends and allocate resources effectively.

Solution:

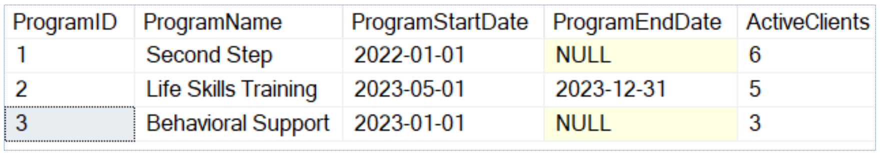
Use a query to group clients by program and evaluate their enrollment trends over time.



```sql
SELECT
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    Programs.StartDate AS ProgramStartDate,
    Programs.EndDate AS ProgramEndDate,
    COUNT(Clients.ClientID) AS ActiveClients
FROM
    Programs
LEFT JOIN
    Clients ON Programs.ProgramID = Clients.CurrentProgramID
GROUP BY
    Programs.ProgramID, Programs.Name, Programs.StartDate, Programs.EndDate
ORDER BY
    COUNT(Clients.ClientID) DESC;
```

Explanation:

- **COUNT(Clients.ClientID)**: Counts the number of active clients in each program.
- The alias ActiveClients is used for display but cannot be referenced in the ORDER BY clause directly.
- **GROUP BY**: Groups the data by program details to calculate the number of clients per program.
- **ORDER BY**: Orders the results by the count of active clients (COUNT(Clients.ClientID) DESC), showing the most utilized programs first.



ProgramID	ProgramName	ProgramStartDate	ProgramEndDate	ActiveClients
1	Second Step	2022-01-01	NULL	6
2	Life Skills Training	2023-05-01	2023-12-31	5
3	Behavioral Support	2023-01-01	NULL	3

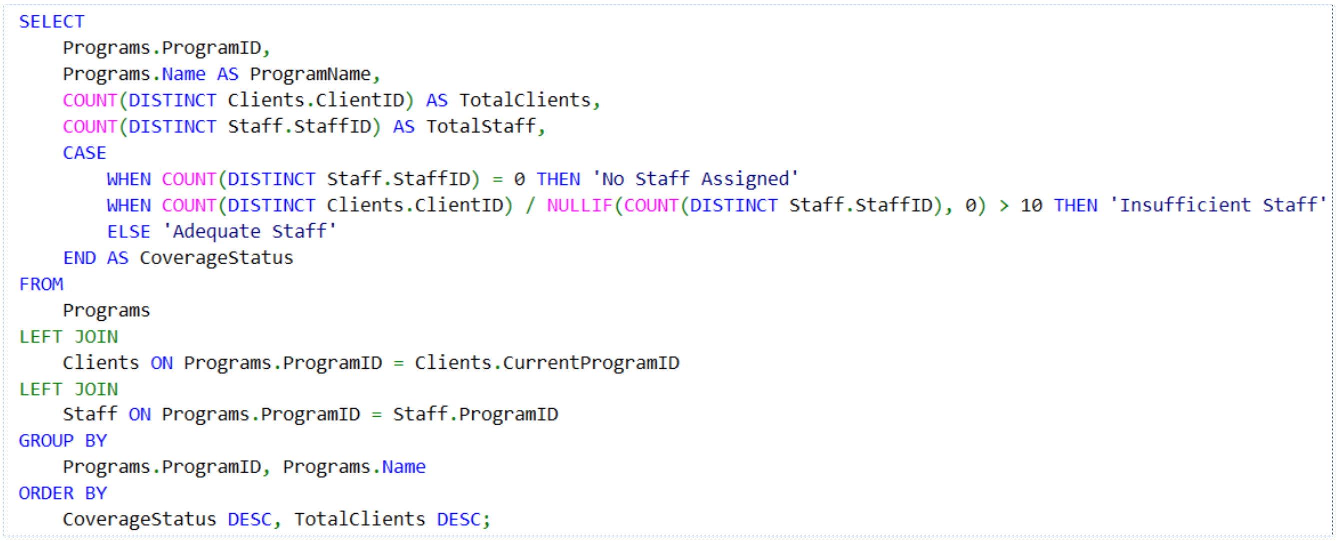
6.13) Staffing Gaps

Scenario:

Identify programs with insufficient staff coverage by analyzing the client-to-staff ratio for each program. This helps in reallocating staff to ensure adequate coverage.

Solution:

Use a query to compare the number of clients and staff assigned to each program, highlighting programs with low staff-to-client ratios.

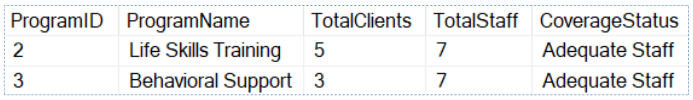


```
SELECT
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    COUNT(DISTINCT Clients.ClientID) AS TotalClients,
    COUNT(DISTINCT Staff.StaffID) AS TotalStaff,
    CASE
        WHEN COUNT(DISTINCT Staff.StaffID) = 0 THEN 'No Staff Assigned'
        WHEN COUNT(DISTINCT Clients.ClientID) / NULLIF(COUNT(DISTINCT Staff.StaffID), 0) > 10 THEN 'Insufficient Staff'
        ELSE 'Adequate Staff'
    END AS CoverageStatus
FROM
    Programs
LEFT JOIN
    Clients ON Programs.ProgramID = Clients.CurrentProgramID
LEFT JOIN
    Staff ON Programs.ProgramID = Staff.ProgramID
GROUP BY
    Programs.ProgramID, Programs.Name
ORDER BY
    CoverageStatus DESC, TotalClients DESC;
```

Explanation:

- **Joins**: LEFT JOIN links Programs to Clients and Staff, ensuring all programs are included, even those without assigned clients or staff.
- **Aggregations**: COUNT(DISTINCT Clients.ClientID): Counts the number of unique clients enrolled in each program.
- COUNT(DISTINCT Staff.StaffID): Counts the number of unique staff members assigned to each program.
- **Coverage Status**: Uses a CASE statement to categorize programs:
  - 'No Staff Assigned': Programs with zero staff.
  - 'Insufficient Staff': Programs with a client-to-staff ratio exceeding 10.
  - 'Adequate Staff': Programs with sufficient staff coverage.
- **NULLIF**: Prevents division by zero by replacing zero staff count with NULL.
- **Ordering**: Sorts by CoverageStatus (to prioritize gaps) and then by the number of clients (TotalClients).

The output is as follows:



ProgramID	ProgramName	TotalClients	TotalStaff	CoverageStatus
2	Life Skills Training	5	7	Adequate Staff
3	Behavioral Support	3	7	Adequate Staff

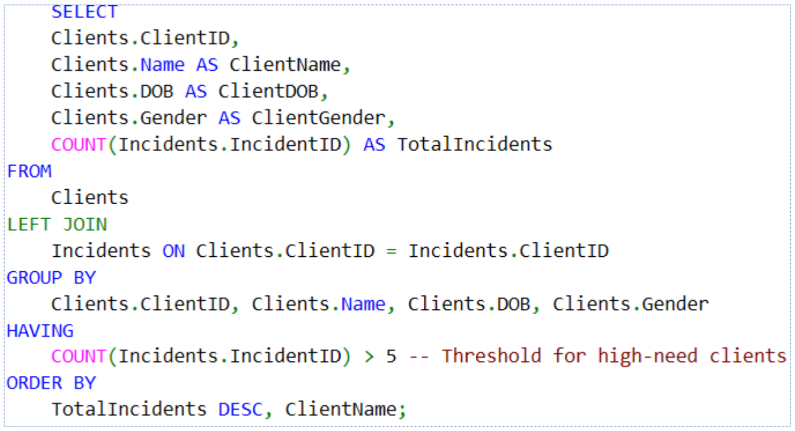
6.14) High-Need Clients

Scenario:

Identify clients with frequent incidents to prioritize resource allocation and targeted interventions.

Solution:

Use a query to count the number of incidents reported for each client and identify those with the highest frequency of occurrences.

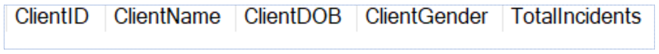


```sql
    SELECT
    Clients.ClientID,
    Clients.Name AS ClientName,
    Clients.DOB AS ClientDOB,
    Clients.Gender AS ClientGender,
    COUNT(Incidents.IncidentID) AS TotalIncidents
FROM
    Clients
LEFT JOIN
    Incidents ON Clients.ClientID = Incidents.ClientID
GROUP BY
    Clients.ClientID, Clients.Name, Clients.DOB, Clients.Gender
HAVING
    COUNT(Incidents.IncidentID) > 5 -- Threshold for high-need clients
ORDER BY
    TotalIncidents DESC, ClientName;
```

Explanation:

- **Joins**: LEFT JOIN links the Clients table with the Incidents table, ensuring all clients are included in the analysis, even those with no incidents.
- **Aggregations**: COUNT(Incidents.IncidentID): Counts the number of incidents associated with each client.
- **Grouping**: Groups data by ClientID, ClientName, DOB, and Gender to calculate incident counts for each client.
- **HAVING Clause**: Filters the results to show only clients with more than 5 incidents (adjustable threshold).
- **Ordering**: Sorts the results by the total number of incidents in descending order, prioritizing clients with the most incidents.

The output as follows:



ClientID	ClientName	ClientDOB	ClientGender	TotalIncidents

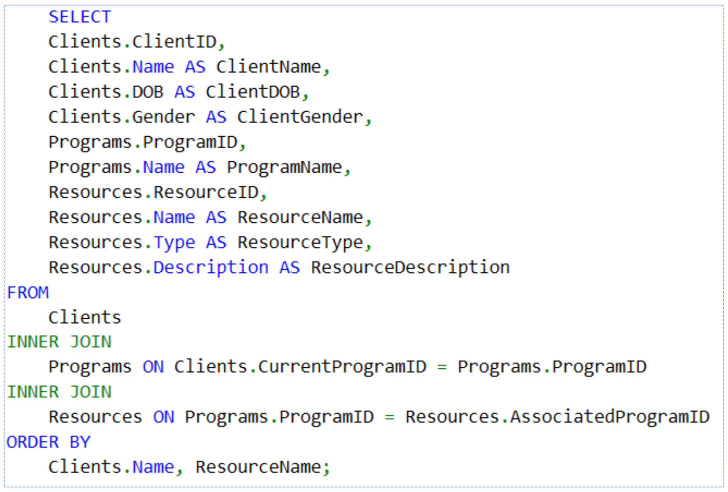
6.15) Resource Utilization by Client

Scenario:

Track which resources are most frequently used by specific clients within their assigned programs to evaluate individual resource engagement.

Solution:

Use a query to join the Resources table with the Clients table via their assigned programs, identifying resources utilized by each client.

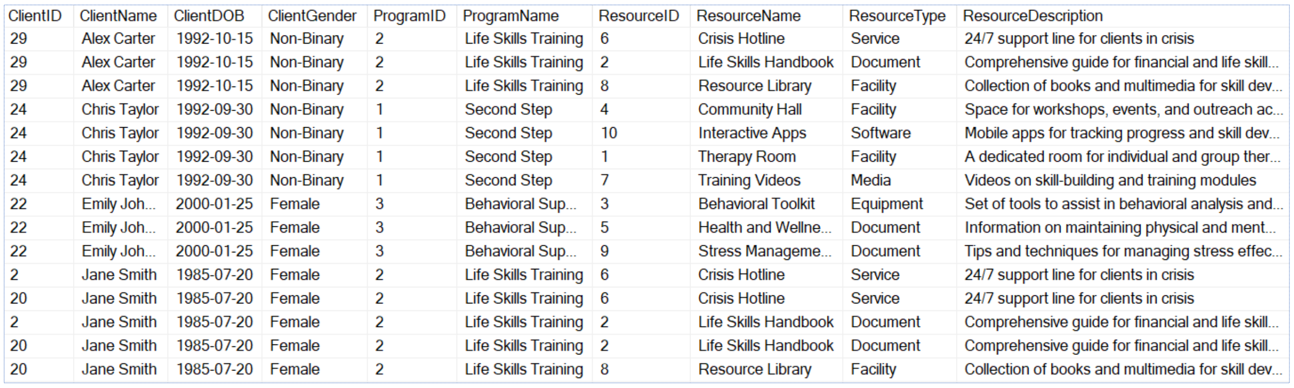


```
    SELECT
    Clients.ClientID,
    Clients.Name AS ClientName,
    Clients.DOB AS ClientDOB,
    Clients.Gender AS ClientGender,
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    Resources.ResourceID,
    Resources.Name AS ResourceName,
    Resources.Type AS ResourceType,
    Resources.Description AS ResourceDescription
FROM
    Clients
INNER JOIN
    Programs ON Clients.CurrentProgramID = Programs.ProgramID
INNER JOIN
    Resources ON Programs.ProgramID = Resources.AssociatedProgramID
ORDER BY
    Clients.Name, ResourceName;
```

Explanation:

- **Joins**: INNER JOIN connects Clients to Programs using CurrentProgramID to link clients to their assigned programs.
- Another INNER JOIN links Programs to Resources via AssociatedProgramID to identify resources used in each program.
- **Selected Columns**: Includes client details (ClientID, ClientName, DOB, Gender).
- Includes program details (ProgramID, ProgramName) to identify the assigned program for each client.
- Includes resource details (ResourceID, ResourceName, ResourceType, ResourceDescription) to show which resources are available to the client.
- **Ordering**: Results are sorted alphabetically by client name and then by resource name for clarity.

A portion of output is as follows:



ClientID	ClientName	ClientDOB	ClientGender	ProgramID	ProgramName	ResourceID	ResourceName	ResourceType	ResourceDescription
29	Alex Carter	1992-10-15	Non-Binary	2	Life Skills Training	6	Crisis Hotline	Service	24/7 support line for clients in crisis
29	Alex Carter	1992-10-15	Non-Binary	2	Life Skills Training	2	Life Skills Handbook	Document	Comprehensive guide for financial and life skill...
29	Alex Carter	1992-10-15	Non-Binary	2	Life Skills Training	8	Resource Library	Facility	Collection of books and multimedia for skill dev...
24	Chris Taylor	1992-09-30	Non-Binary	1	Second Step	4	Community Hall	Facility	Space for workshops, events, and outreach ac...
24	Chris Taylor	1992-09-30	Non-Binary	1	Second Step	10	Interactive Apps	Software	Mobile apps for tracking progress and skill dev...
24	Chris Taylor	1992-09-30	Non-Binary	1	Second Step	1	Therapy Room	Facility	A dedicated room for individual and group ther...
24	Chris Taylor	1992-09-30	Non-Binary	1	Second Step	7	Training Videos	Media	Videos on skill-building and training modules
22	Emily Joh...	2000-01-25	Female	3	Behavioral Sup...	3	Behavioral Toolkit	Equipment	Set of tools to assist in behavioral analysis and...
22	Emily Joh...	2000-01-25	Female	3	Behavioral Sup...	5	Health and Wellne...	Document	Information on maintaining physical and ment...
22	Emily Joh...	2000-01-25	Female	3	Behavioral Sup...	9	Stress Manageme...	Document	Tips and techniques for managing stress effec...
2	Jane Smith	1985-07-20	Female	2	Life Skills Training	6	Crisis Hotline	Service	24/7 support line for clients in crisis
20	Jane Smith	1985-07-20	Female	2	Life Skills Training	6	Crisis Hotline	Service	24/7 support line for clients in crisis
2	Jane Smith	1985-07-20	Female	2	Life Skills Training	2	Life Skills Handbook	Document	Comprehensive guide for financial and life skill...
20	Jane Smith	1985-07-20	Female	2	Life Skills Training	2	Life Skills Handbook	Document	Comprehensive guide for financial and life skill...
20	Jane Smith	1985-07-20	Female	2	Life Skills Training	8	Resource Library	Facility	Collection of books and multimedia for skill dev...

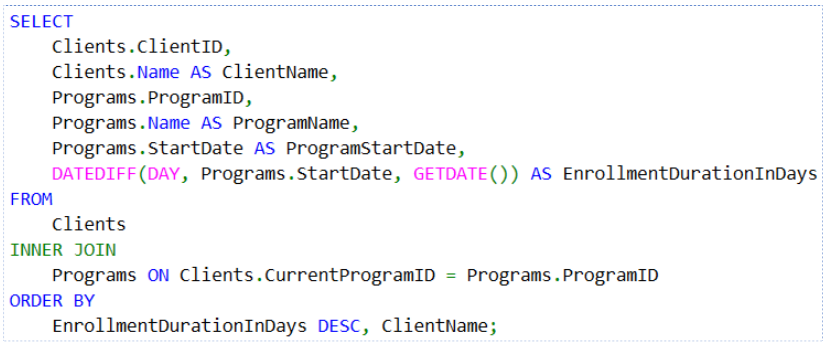
6.16) Longest Tenure in Programs

Scenario:

Identify the clients who have been enrolled in their current programs for the longest duration to recognize loyalty or evaluate long-term outcomes.

Solution:

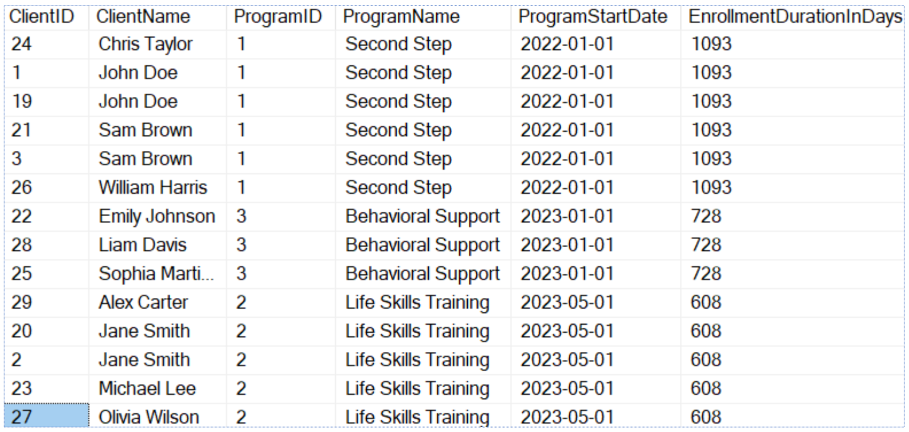
Use a query to calculate the duration of enrollment for each client in their current program and sort by the longest duration.



```sql
SELECT
    Clients.ClientID,
    Clients.Name AS ClientName,
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    Programs.StartDate AS ProgramStartDate,
    DATEDIFF(DAY, Programs.StartDate, GETDATE()) AS EnrollmentDurationInDays
FROM
    Clients
INNER JOIN
    Programs ON Clients.CurrentProgramID = Programs.ProgramID
ORDER BY
    EnrollmentDurationInDays DESC, ClientName;
```

Explanation:

- **Joins**: INNER JOIN connects Clients to Programs using CurrentProgramID to associate clients with their current programs.
- **Selected Columns**: Includes client details (ClientID, ClientName).
- Includes program details (ProgramID, ProgramName, ProgramStartDate) to provide context about the assigned program.
- Calculates the enrollment duration (EnrollmentDurationInDays) using DATEDIFF, which measures the number of days between the program's start date and the current date.
- **Ordering**: Sorts results in descending order of EnrollmentDurationInDays to display the longest-tenured clients first.



ClientID	ClientName	ProgramID	ProgramName	ProgramStartDate	EnrollmentDurationInDays
24	Chris Taylor	1	Second Step	2022-01-01	1093
1	John Doe	1	Second Step	2022-01-01	1093
19	John Doe	1	Second Step	2022-01-01	1093
21	Sam Brown	1	Second Step	2022-01-01	1093
3	Sam Brown	1	Second Step	2022-01-01	1093
26	William Harris	1	Second Step	2022-01-01	1093
22	Emily Johnson	3	Behavioral Support	2023-01-01	728
28	Liam Davis	3	Behavioral Support	2023-01-01	728
25	Sophia Marti...	3	Behavioral Support	2023-01-01	728
29	Alex Carter	2	Life Skills Training	2023-05-01	608
20	Jane Smith	2	Life Skills Training	2023-05-01	608
2	Jane Smith	2	Life Skills Training	2023-05-01	608
23	Michael Lee	2	Life Skills Training	2023-05-01	608
27	Olivia Wilson	2	Life Skills Training	2023-05-01	608

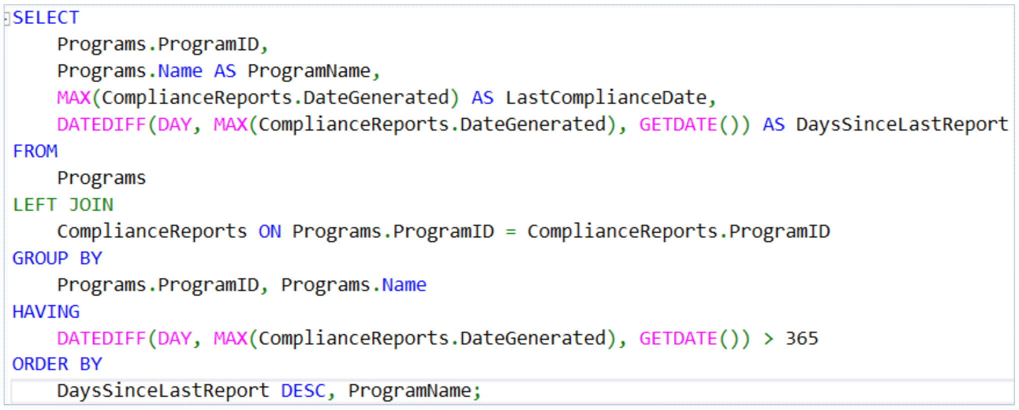
6.17) Compliance Monitoring

Scenario:

Identify programs with overdue compliance reports by finding programs where the last compliance report was generated over a year ago. This helps ensure timely updates and adherence to compliance standards.

Solution:

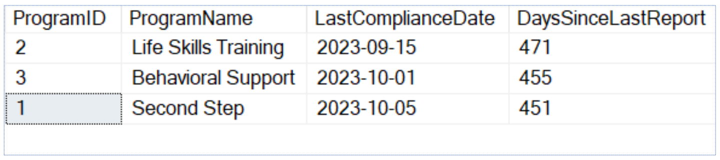
Use a query to compare the most recent compliance report date for each program with the current date and flag overdue reports.



```
SELECT
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    MAX(ComplianceReports.DateGenerated) AS LastComplianceDate,
    DATEDIFF(DAY, MAX(ComplianceReports.DateGenerated), GETDATE()) AS DaysSinceLastReport
FROM
    Programs
LEFT JOIN
    ComplianceReports ON Programs.ProgramID = ComplianceReports.ProgramID
GROUP BY
    Programs.ProgramID, Programs.Name
HAVING
    DATEDIFF(DAY, MAX(ComplianceReports.DateGenerated), GETDATE()) > 365
ORDER BY
    DaysSinceLastReport DESC, ProgramName;
```

Explanation:

- **Joins**: LEFT JOIN connects Programs to ComplianceReports to include all programs, even those without recent reports.
- **Aggregations**: MAX(ComplianceReports.DateGenerated): Finds the most recent compliance report date for each program.
- **Calculated Columns**: DATEDIFF(DAY, MAX(ComplianceReports.DateGenerated), GETDATE()): Calculates the number of days since the most recent report.
- **Filters**: The HAVING clause filters programs where the last compliance report was generated more than 365 days ago (overdue reports).
- **Grouping**: Groups results by ProgramID and ProgramName to aggregate data for each program.
- **Ordering**: Sorts results by the number of days since the last report in descending order, prioritizing the most overdue programs.



ProgramID	ProgramName	LastComplianceDate	DaysSinceLastReport
2	Life Skills Training	2023-09-15	471
3	Behavioral Support	2023-10-01	455
1	Second Step	2023-10-05	451

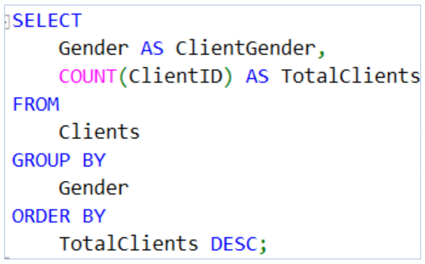
6.18) Clients by Gender

Scenario:

Analyze the distribution of clients based on gender to identify demographic patterns and guide program planning or resource allocation.

Solution:

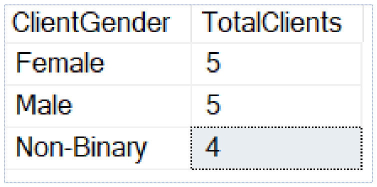
Group clients by gender and count the total number of clients in each group.



```
SELECT
    Gender AS ClientGender,
    COUNT(ClientID) AS TotalClients
FROM
    Clients
GROUP BY
    Gender
ORDER BY
    TotalClients DESC;
```

Explanation:

- **Grouping**: Groups the clients by the Gender column to calculate the distribution of clients across different genders.
- **Aggregations**: COUNT(ClientID): Counts the total number of clients in each gender group.
- **Ordering**: Sorts the results by the total number of clients in descending order (TotalClients DESC) to display the largest groups first.
- **Columns**: ClientGender: Shows the gender category (e.g., Male, Female, Non-Binary).
- TotalClients: Displays the number of clients in each gender group.



ClientGender	TotalClients
Female	5
Male	5
Non-Binary	4

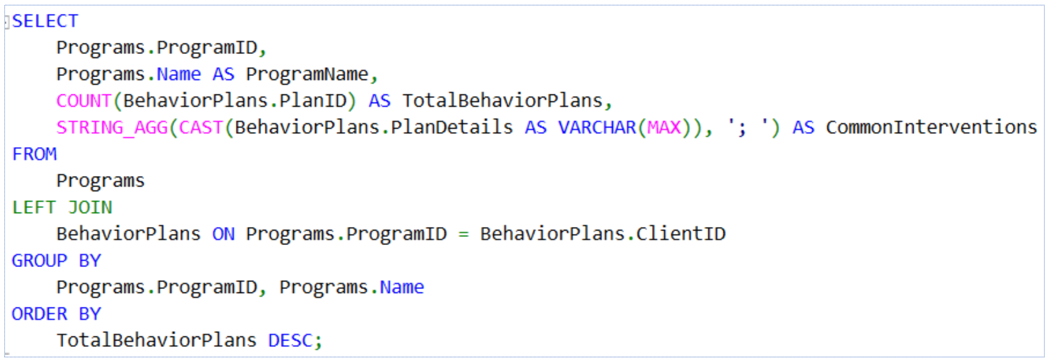
6.19) Behavioral Trends by Program

Scenario:

Analyze trends in behavior plans for each program to understand which interventions are commonly applied and their frequency.

Solution:

Join the BehaviorPlans table with the Programs table to group behavior plans by program and count the number of plans applied to each program.

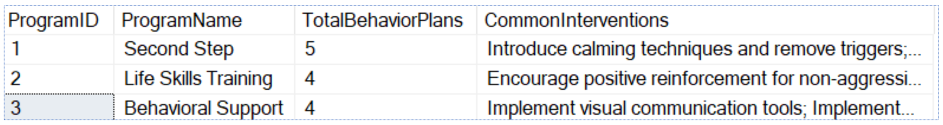


```
SELECT
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    COUNT(BehaviorPlans.PlanID) AS TotalBehaviorPlans,
    STRING_AGG(CAST(BehaviorPlans.PlanDetails AS VARCHAR(MAX)), '; ') AS CommonInterventions
FROM
    Programs
LEFT JOIN
    BehaviorPlans ON Programs.ProgramID = BehaviorPlans.ClientID
GROUP BY
    Programs.ProgramID, Programs.Name
ORDER BY
    TotalBehaviorPlans DESC;
```

Explanation:

- **Data Type Conversion**: CAST(BehaviorPlans.PlanDetails AS VARCHAR(MAX)): Converts the TEXT column PlanDetails to a compatible VARCHAR(MAX) type for use with STRING_AGG.
- **Functionality**: STRING_AGG: Concatenates all behavior plan details for each program into a single string, separated by ;.
- **Grouping**: Groups by ProgramID and ProgramName to aggregate behavior plan trends for each program.
- **Ordering**: Sorts results by the number of behavior plans in descending order (TotalBehaviorPlans DESC).

The output:



ProgramID	ProgramName	TotalBehaviorPlans	CommonInterventions
1	Second Step	5	Introduce calming techniques and remove triggers;...
2	Life Skills Training	4	Encourage positive reinforcement for non-aggressi...
3	Behavioral Support	4	Implement visual communication tools; Implement...

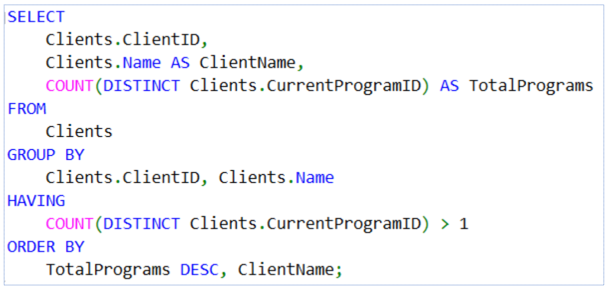
6.20) Client Overlap Between Programs

Scenario:

Identify clients who have participated in multiple programs over time to evaluate cross-program engagement or transitions.

Solution:

Query the Clients table to detect client IDs that appear under multiple CurrentProgramID values by grouping and counting unique program associations.



```
SELECT
    Clients.ClientID,
    Clients.Name AS ClientName,
    COUNT(DISTINCT Clients.CurrentProgramID) AS TotalPrograms
FROM
    Clients
GROUP BY
    Clients.ClientID, Clients.Name
HAVING
    COUNT(DISTINCT Clients.CurrentProgramID) > 1
ORDER BY
    TotalPrograms DESC, ClientName;
```

Explanation:

- **Grouping**: Groups clients by ClientID and ClientName to aggregate their program associations.
- **Aggregations**: COUNT(DISTINCT Clients.CurrentProgramID): Counts the number of unique programs each client has participated in.
- **HAVING Clause**: Filters results to include only clients who have been associated with more than one program (COUNT(DISTINCT Clients.CurrentProgramID) > 1).
- **Ordering**: Sorts results by the total number of programs in descending order (TotalPrograms DESC) and then alphabetically by client name.

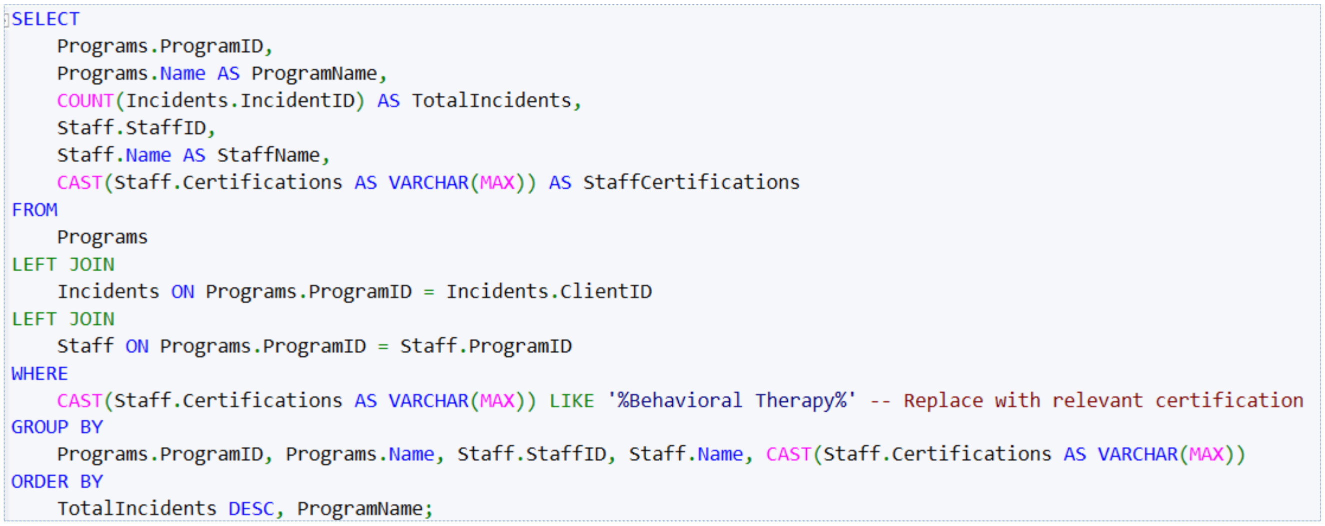
6.21) Specialized Training Effectiveness

Scenario:

Evaluate whether staff with specialized training are assigned to programs with higher needs by analyzing staff certifications and incident rates in their assigned programs.

Solution:

Query the Staff, Programs, and Incidents tables to compare staff certifications with the incident rates of the programs they are assigned to.

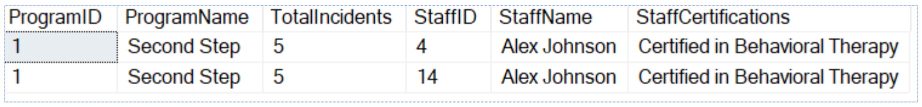


```sql
SELECT
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    COUNT(Incidents.IncidentID) AS TotalIncidents,
    Staff.StaffID,
    Staff.Name AS StaffName,
    CAST(Staff.Certifications AS VARCHAR(MAX)) AS StaffCertifications
FROM
    Programs
LEFT JOIN
    Incidents ON Programs.ProgramID = Incidents.ClientID
LEFT JOIN
    Staff ON Programs.ProgramID = Staff.ProgramID
WHERE
    CAST(Staff.Certifications AS VARCHAR(MAX)) LIKE '%Behavioral Therapy%' -- Replace with relevant certification
GROUP BY
    Programs.ProgramID, Programs.Name, Staff.StaffID, Staff.Name, CAST(Staff.Certifications AS VARCHAR(MAX))
ORDER BY
    TotalIncidents DESC, ProgramName;
```

Explanation:

- **Data Type Conversion**: CAST(Staff.Certifications AS VARCHAR(MAX)): Converts the TEXT column to VARCHAR(MAX), allowing for operations like LIKE or comparisons.
- **Filter**: WHERE CAST(Staff.Certifications AS VARCHAR(MAX)) LIKE '%Behavioral Therapy%': Filters for staff with certifications relevant to program needs.

- **Grouping**: Groups by program and staff details to aggregate incident counts and evaluate alignment.
- **Ordering**: Sorts by the number of incidents (TotalIncidents DESC) and then by program name.



ProgramID	ProgramName	TotalIncidents	StaffID	StaffName	StaffCertifications
1	Second Step	5	4	Alex Johnson	Certified in Behavioral Therapy
1	Second Step	5	14	Alex Johnson	Certified in Behavioral Therapy

6.22) Clients and Their Assigned Staff

Scenario Description:

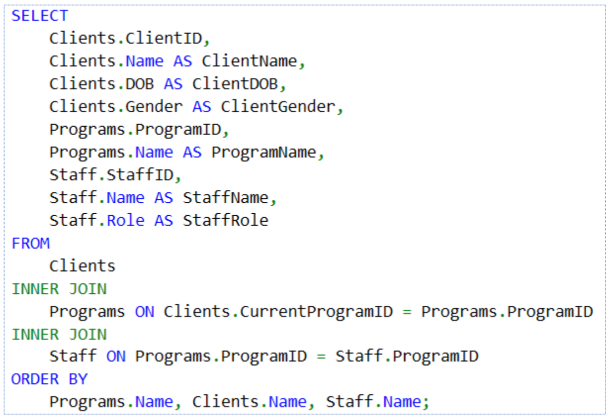
The management wants a report that lists all clients along with the staff members assigned to their respective programs. This report evaluates whether clients are adequately supported by program staff.

Solution Using INNER JOIN:

We will use an INNER JOIN to combine the Clients, Staff, and Programs tables. The linking is performed through:

- CurrentProgramID in the Clients table.
- ProgramID in the Staff table.

A portion of the output is:

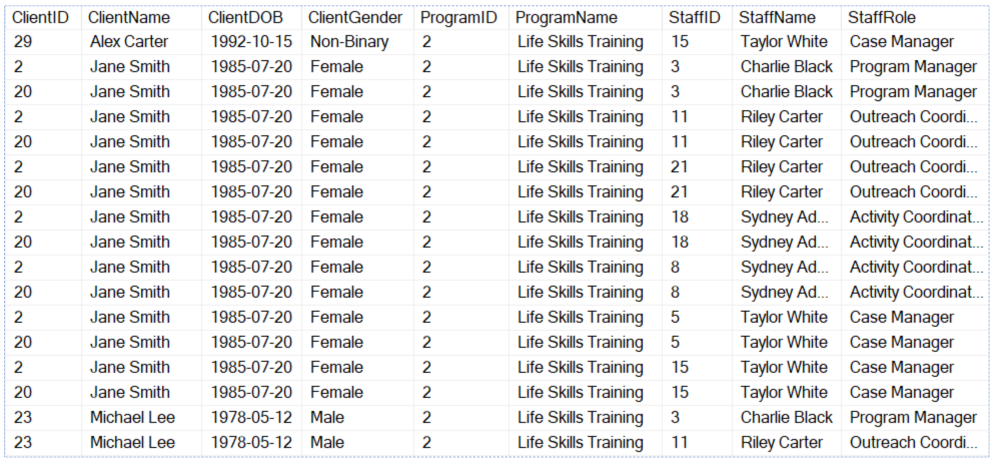


```
SELECT
    Clients.ClientID,
    Clients.Name AS ClientName,
    Clients.DOB AS ClientDOB,
    Clients.Gender AS ClientGender,
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    Staff.StaffID,
    Staff.Name AS StaffName,
    Staff.Role AS StaffRole
FROM
    Clients
INNER JOIN
    Programs ON Clients.CurrentProgramID = Programs.ProgramID
INNER JOIN
    Staff ON Programs.ProgramID = Staff.ProgramID
ORDER BY
    Programs.Name, Clients.Name, Staff.Name;
```

Breakdown of the Query:

1. **Selected Columns**:
   - **From Clients Table**:
     - ClientID, Name (aliased as ClientName), DOB, and Gender provide key demographic information about each client.
   - **From Programs Table**:
     - ProgramID and Name (aliased as ProgramName) identify the program to which each client is assigned.

   - **From Staff Table**:
      - StaffID, Name (aliased as StaffName), and Role provide details about the staff members supporting each program.
2. **Joins**:
   - **INNER JOIN Clients and Programs**:
      - Links the Clients table with the Programs table using CurrentProgramID in Clients and ProgramID in Programs. This ensures each client is matched to their assigned program.
   - **INNER JOIN Programs and Staff**:
      - Links the Programs table with the Staff table using ProgramID in both tables. This ensures each program is matched to its assigned staff members.
3. **Ordering**:
   - **Programs.Name**: Groups and sorts results by program name.
   - **Clients.Name**: Sorts clients alphabetically within each program.
   - **Staff.Name**: Sorts staff members alphabetically within each program-client group.
4. **Aliases**:
   - Columns are aliased for better readability in the result set (e.g., Programs.Name AS ProgramName).



ClientID	ClientName	ClientDOB	ClientGender	ProgramID	ProgramName	StaffID	StaffName	StaffRole
29	Alex Carter	1992-10-15	Non-Binary	2	Life Skills Training	15	Taylor White	Case Manager
2	Jane Smith	1985-07-20	Female	2	Life Skills Training	3	Charlie Black	Program Manager
20	Jane Smith	1985-07-20	Female	2	Life Skills Training	3	Charlie Black	Program Manager
2	Jane Smith	1985-07-20	Female	2	Life Skills Training	11	Riley Carter	Outreach Coordi...
20	Jane Smith	1985-07-20	Female	2	Life Skills Training	11	Riley Carter	Outreach Coordi...
2	Jane Smith	1985-07-20	Female	2	Life Skills Training	21	Riley Carter	Outreach Coordi...
20	Jane Smith	1985-07-20	Female	2	Life Skills Training	21	Riley Carter	Outreach Coordi...
2	Jane Smith	1985-07-20	Female	2	Life Skills Training	18	Sydney Ad...	Activity Coordinat...
20	Jane Smith	1985-07-20	Female	2	Life Skills Training	18	Sydney Ad...	Activity Coordinat...
2	Jane Smith	1985-07-20	Female	2	Life Skills Training	8	Sydney Ad...	Activity Coordinat...
20	Jane Smith	1985-07-20	Female	2	Life Skills Training	8	Sydney Ad...	Activity Coordinat...
2	Jane Smith	1985-07-20	Female	2	Life Skills Training	5	Taylor White	Case Manager
20	Jane Smith	1985-07-20	Female	2	Life Skills Training	5	Taylor White	Case Manager
2	Jane Smith	1985-07-20	Female	2	Life Skills Training	15	Taylor White	Case Manager
20	Jane Smith	1985-07-20	Female	2	Life Skills Training	15	Taylor White	Case Manager
23	Michael Lee	1978-05-12	Male	2	Life Skills Training	3	Charlie Black	Program Manager
23	Michael Lee	1978-05-12	Male	2	Life Skills Training	11	Riley Carter	Outreach Coordi...

6.23) Scenario: Clients Without Incidents

Scenario:

Management requires a report that lists all clients, including those without any recorded incidents. This report can help identify clients who may need less intervention or those for whom incidents might not have been properly logged.

Solution Using LEFT JOIN:

A LEFT JOIN is used to combine the Clients table with the Incidents table, ensuring that all clients are included in the result, even if they do not have matching records in the Incidents table.

```sql
SELECT
    Clients.ClientID,
    Clients.Name AS ClientName,
    Clients.DOB AS ClientDOB,
    Clients.Gender AS ClientGender,
    CAST(Clients.Address AS VARCHAR(MAX)) AS ClientAddress,
    Programs.Name AS ProgramName,
    COUNT(Incidents.IncidentID) AS TotalIncidents
FROM
    Clients
LEFT JOIN
    Programs ON Clients.CurrentProgramID = Programs.ProgramID
LEFT JOIN
    Incidents ON Clients.ClientID = Incidents.ClientID
GROUP BY
    Clients.ClientID, Clients.Name, Clients.DOB, Clients.Gender, CAST(Clients.Address AS VARCHAR(MAX)), Programs.Name
ORDER BY
    TotalIncidents ASC, Clients.Name;
```

Explanation of the Query:

1. **Joins**:
   - **LEFT JOIN Clients and Programs**:
     - Links the Clients table with the Programs table via CurrentProgramID, ensuring all clients are associated with their respective programs.
   - **LEFT JOIN Clients and Incidents**:
     - Links the Clients table with the Incidents table via ClientID, ensuring all clients are included even if they have no recorded incidents.
2. **Selected Columns**:
   - **From Clients**:
     - Includes demographic and contact details (ClientID, Name, DOB, Gender, Address).
   - **From Programs**:
     - Includes the name of the program (ProgramName) to which the client is assigned.
   - **From Incidents**:
     - Uses COUNT(Incidents.IncidentID) to calculate the total number of incidents for each client. Clients without incidents will have a count of 0.
3. **Grouping**:
   - Groups results by all Clients and Programs columns to aggregate incident counts for each client.

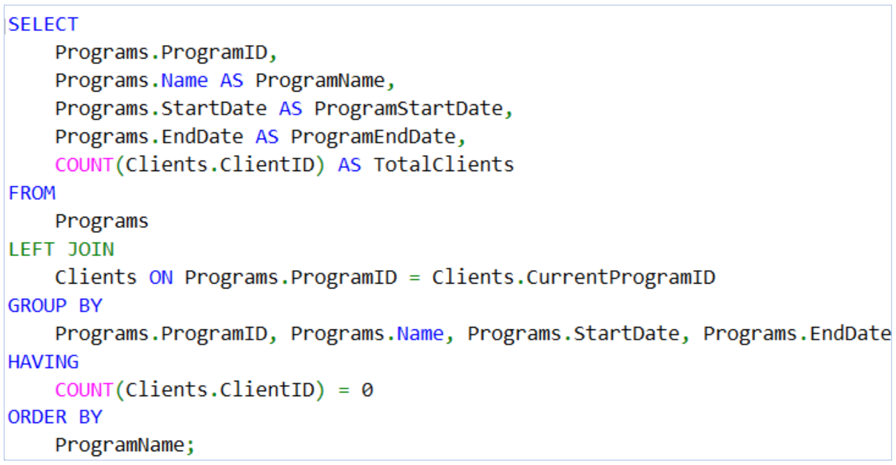
4. **Ordering**:
   - Sorts results by TotalIncidents in ascending order to prioritize clients with no incidents, and then alphabetically by client name.

6.24) Scenario: Programs Without Assigned Clients

Management seeks to identify programs that currently have no enrolled clients. This allows them to pinpoint underutilized programs that may require reevaluation, additional resources, or potential closure.

Solution Using RIGHT JOIN:

A RIGHT JOIN is used to ensure that all programs are included in the results, even if there are no clients assigned to them. Programs without assigned clients will appear with NULL values for client details.



```
SELECT
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    Programs.StartDate AS ProgramStartDate,
    Programs.EndDate AS ProgramEndDate,
    COUNT(Clients.ClientID) AS TotalClients
FROM
    Programs
LEFT JOIN
    Clients ON Programs.ProgramID = Clients.CurrentProgramID
GROUP BY
    Programs.ProgramID, Programs.Name, Programs.StartDate, Programs.EndDate
HAVING
    COUNT(Clients.ClientID) = 0
ORDER BY
    ProgramName;
```

Explanation of the Query:

1. **Joins**:
   - **LEFT JOIN**:
     - Combines the Programs table with the Clients table using ProgramID in Programs and CurrentProgramID in Clients.
     - Ensures all programs are included, even if no clients are enrolled.
2. **Columns**:
   - **From Programs**:
     - Includes ProgramID, Name (aliased as ProgramName), StartDate, and EndDate to provide key program details.
   - **From Clients**:
     - COUNT(Clients.ClientID) calculates the total number of clients assigned to each program. Programs with no clients will have a count of 0.

3. **Grouping**:
    - Groups results by program details (ProgramID, Name, StartDate, EndDate) to aggregate client counts for each program.
4. **Filter**:
    - **HAVING COUNT(Clients.ClientID) = 0**:
        - Filters results to include only programs with no enrolled clients.
5. **Ordering**:
    - Sorts the results alphabetically by ProgramName for clarity.

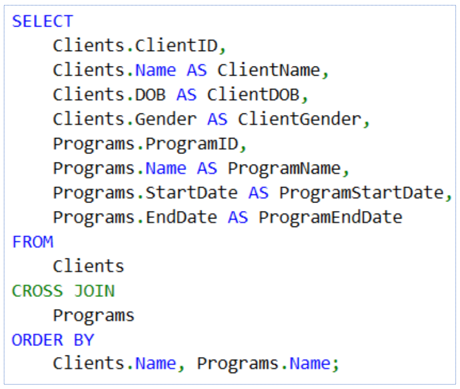
6.25) Scenario: Generating All Possible Client-Program Combinations

Scenario Description:

Management needs to generate all possible client-to-program assignments to simulate enrollment scenarios or analyze compatibility for future assignments. This helps in planning and optimizing resource allocation.

Solution Using CROSS JOIN:

A CROSS JOIN combines each row from the Clients' table with every row from the Programs table, producing all possible combinations of clients and programs.



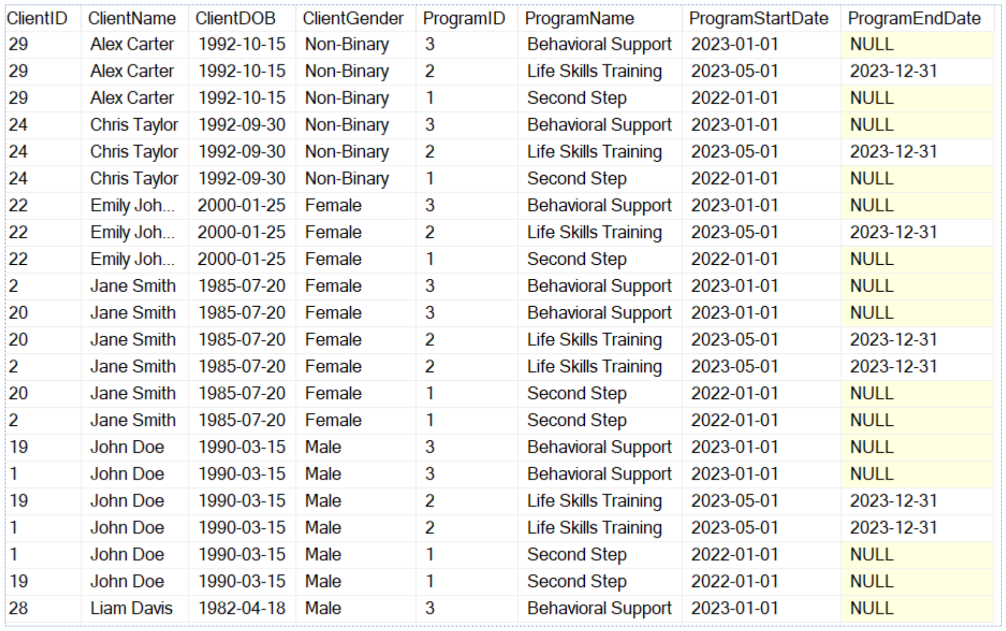
```
SELECT
    Clients.ClientID,
    Clients.Name AS ClientName,
    Clients.DOB AS ClientDOB,
    Clients.Gender AS ClientGender,
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    Programs.StartDate AS ProgramStartDate,
    Programs.EndDate AS ProgramEndDate
FROM
    Clients
CROSS JOIN
    Programs
ORDER BY
    Clients.Name, Programs.Name;
```

Explanation of the Query:

1. **CROSS JOIN**:
    - Combines every row from the Clients table with every row from the Programs table.
    - Produces a Cartesian product, resulting in all possible client-program combinations.
2. **Selected Columns**:
    - **From Clients**:

        - Includes ClientID, Name (aliased as ClientName), DOB, and Gender to provide client details.
    - **From Programs**:
        - Includes ProgramID, Name (aliased as ProgramName), StartDate, and EndDate to provide program details.
3. **Ordering**:
    - Sorts results alphabetically by ClientName and then by ProgramName for better readability.

A part of the outcome is:



ClientID	ClientName	ClientDOB	ClientGender	ProgramID	ProgramName	ProgramStartDate	ProgramEndDate
29	Alex Carter	1992-10-15	Non-Binary	3	Behavioral Support	2023-01-01	NULL
29	Alex Carter	1992-10-15	Non-Binary	2	Life Skills Training	2023-05-01	2023-12-31
29	Alex Carter	1992-10-15	Non-Binary	1	Second Step	2022-01-01	NULL
24	Chris Taylor	1992-09-30	Non-Binary	3	Behavioral Support	2023-01-01	NULL
24	Chris Taylor	1992-09-30	Non-Binary	2	Life Skills Training	2023-05-01	2023-12-31
24	Chris Taylor	1992-09-30	Non-Binary	1	Second Step	2022-01-01	NULL
22	Emily Joh...	2000-01-25	Female	3	Behavioral Support	2023-01-01	NULL
22	Emily Joh...	2000-01-25	Female	2	Life Skills Training	2023-05-01	2023-12-31
22	Emily Joh...	2000-01-25	Female	1	Second Step	2022-01-01	NULL
2	Jane Smith	1985-07-20	Female	3	Behavioral Support	2023-01-01	NULL
20	Jane Smith	1985-07-20	Female	3	Behavioral Support	2023-01-01	NULL
20	Jane Smith	1985-07-20	Female	2	Life Skills Training	2023-05-01	2023-12-31
2	Jane Smith	1985-07-20	Female	2	Life Skills Training	2023-05-01	2023-12-31
20	Jane Smith	1985-07-20	Female	1	Second Step	2022-01-01	NULL
2	Jane Smith	1985-07-20	Female	1	Second Step	2022-01-01	NULL
19	John Doe	1990-03-15	Male	3	Behavioral Support	2023-01-01	NULL
1	John Doe	1990-03-15	Male	3	Behavioral Support	2023-01-01	NULL
19	John Doe	1990-03-15	Male	2	Life Skills Training	2023-05-01	2023-12-31
1	John Doe	1990-03-15	Male	2	Life Skills Training	2023-05-01	2023-12-31
1	John Doe	1990-03-15	Male	1	Second Step	2022-01-01	NULL
19	John Doe	1990-03-15	Male	1	Second Step	2022-01-01	NULL
28	Liam Davis	1982-04-18	Male	3	Behavioral Support	2023-01-01	NULL

6.26) Scenario: Identifying Clients Without Programs and Programs Without Clients

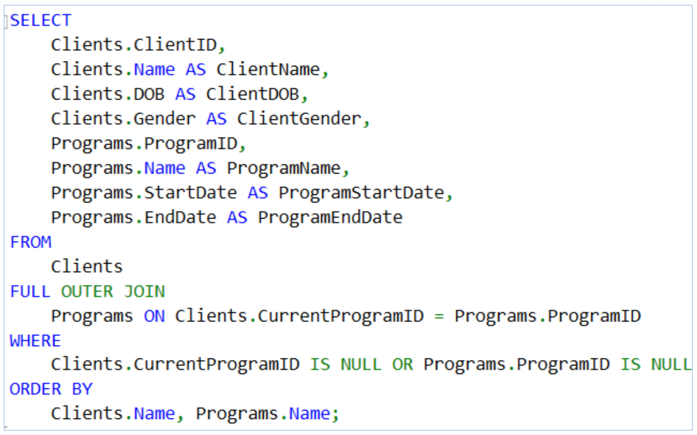
Scenario Description:

Management wants to:

1. Identify clients who are not currently assigned to any program.
2. Identify programs that currently have no clients enrolled. This ensures no client is overlooked, and no program is underutilized.

Solution Using FULL OUTER JOIN:

A FULL OUTER JOIN combines all rows from the Clients and Programs tables, showing matches where they exist and NULL values for unmatched rows in either table.

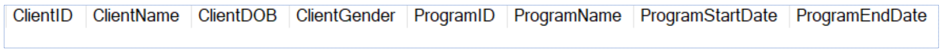


```
SELECT
    Clients.ClientID,
    Clients.Name AS ClientName,
    Clients.DOB AS ClientDOB,
    Clients.Gender AS ClientGender,
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    Programs.StartDate AS ProgramStartDate,
    Programs.EndDate AS ProgramEndDate
FROM
    Clients
FULL OUTER JOIN
    Programs ON Clients.CurrentProgramID = Programs.ProgramID
WHERE
    Clients.CurrentProgramID IS NULL OR Programs.ProgramID IS NULL
ORDER BY
    Clients.Name, Programs.Name;
```

Explanation of the Query:

1. **FULL OUTER JOIN**:
    - Combines all rows from the Clients and Programs tables.
    - Rows with matches are included, and NULL values are shown for unmatched rows.
2. **Filter**:
    - **WHERE Clients.CurrentProgramID IS NULL**:
        - Identifies programs without clients.
    - **WHERE Programs.ProgramID IS NULL**:
        - Identifies clients without programs.
3. **Selected Columns**:
        - **From Clients**: Includes ClientID, Name (aliased as ClientName), DOB, and Gender.
        - **From Programs**: Includes ProgramID, Name (aliased as ProgramName), StartDate, and EndDate.
4. **Ordering**:
    - Sorts results alphabetically by ClientName and ProgramName for clarity.

The output is :



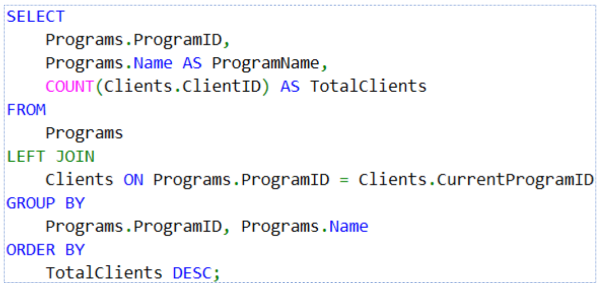
ClientID	ClientName	ClientDOB	ClientGender	ProgramID	ProgramName	ProgramStartDate	ProgramEndDate

6.27) Program Enrollment

Scenario Description:

Management seeks to know how many clients are enrolled in each program to assess program popularity and workload distribution.

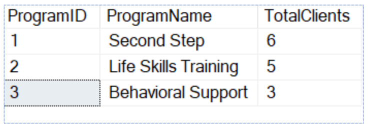
SQL Query for Program Enrollment:



```sql
SELECT
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    COUNT(Clients.ClientID) AS TotalClients
FROM
    Programs
LEFT JOIN
    Clients ON Programs.ProgramID = Clients.CurrentProgramID
GROUP BY
    Programs.ProgramID, Programs.Name
ORDER BY
    TotalClients DESC;
```

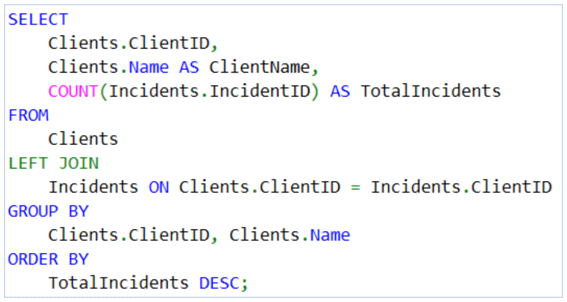
Explanation:

- Counts the number of clients (COUNT(Clients.ClientID)) enrolled in each program.
- Groups by ProgramID and ProgramName to aggregate the client count per program.
- Includes programs with zero clients using LEFT JOIN.



ProgramID	ProgramName	TotalClients
1	Second Step	6
2	Life Skills Training	5
3	Behavioral Support	3

(i). Count Total Incidents for Each Client:

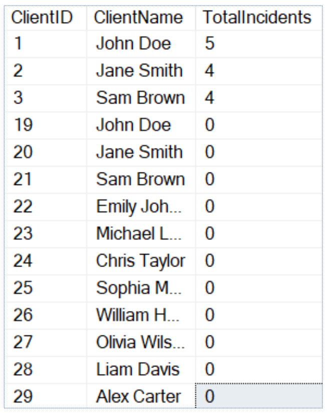


```sql
SELECT
    Clients.ClientID,
    Clients.Name AS ClientName,
    COUNT(Incidents.IncidentID) AS TotalIncidents
FROM
    Clients
LEFT JOIN
    Incidents ON Clients.ClientID = Incidents.ClientID
GROUP BY
    Clients.ClientID, Clients.Name
ORDER BY
    TotalIncidents DESC;
```

Explanation:

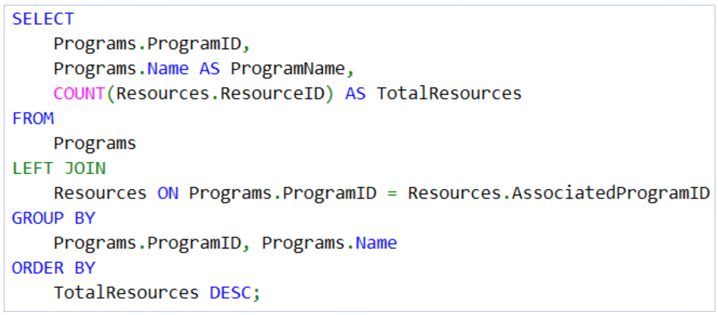
- Counts the total number of incidents (COUNT(Incidents.IncidentID)) per client.
- Groups by ClientID and ClientName to summarize incidents for each client.
- Includes clients without incidents using LEFT JOIN.

The output is



ClientID	ClientName	TotalIncidents
1	John Doe	5
2	Jane Smith	4
3	Sam Brown	4
19	John Doe	0
20	Jane Smith	0
21	Sam Brown	0
22	Emily Joh...	0
23	Michael L...	0
24	Chris Taylor	0
25	Sophia M...	0
26	William H...	0
27	Olivia Wils...	0
28	Liam Davis	0
29	Alex Carter	0

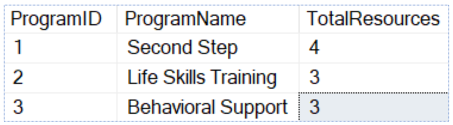
(ii). Count Clients by Gender:



```
SELECT
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    COUNT(Resources.ResourceID) AS TotalResources
FROM
    Programs
LEFT JOIN
    Resources ON Programs.ProgramID = Resources.AssociatedProgramID
GROUP BY
    Programs.ProgramID, Programs.Name
ORDER BY
    TotalResources DESC;
```

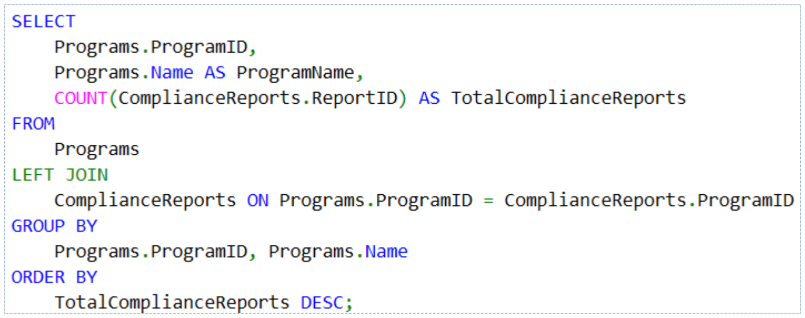
Explanation:

- Counts the total number of resources (COUNT(Resources.ResourceID)) assigned to each program.
- Groups by ProgramID and ProgramName to aggregate resource counts.
- Includes programs without resources using LEFT JOIN



ProgramID	ProgramName	TotalResources
1	Second Step	4
2	Life Skills Training	3
3	Behavioral Support	3

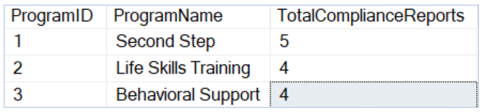
(iii). Count Compliance Reports Per Program:



```
SELECT
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    COUNT(ComplianceReports.ReportID) AS TotalComplianceReports
FROM
    Programs
LEFT JOIN
    ComplianceReports ON Programs.ProgramID = ComplianceReports.ProgramID
GROUP BY
    Programs.ProgramID, Programs.Name
ORDER BY
    TotalComplianceReports DESC;
```

Explanation:

- Counts the total number of compliance reports (COUNT(ComplianceReports.ReportID)) per program.
- Groups by ProgramID and ProgramName to summarize reports for each program.
- Includes programs without compliance reports using LEFT JOIN.



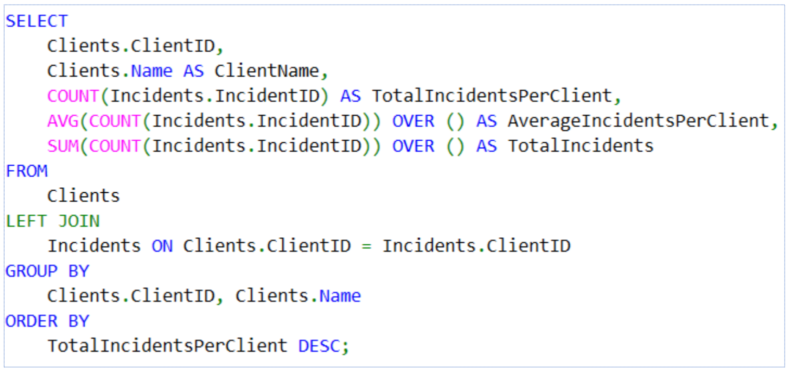
ProgramID	ProgramName	TotalComplianceReports
1	Second Step	5
2	Life Skills Training	4
3	Behavioral Support	4

6.28) Calculating the Average Number of Incidents Per Client and Total Incidents

Scenario Description:

Management needs to analyze the average number of incidents per client and the total number of incidents across all clients. This provides insights into overall behavioral trends and client-specific patterns.

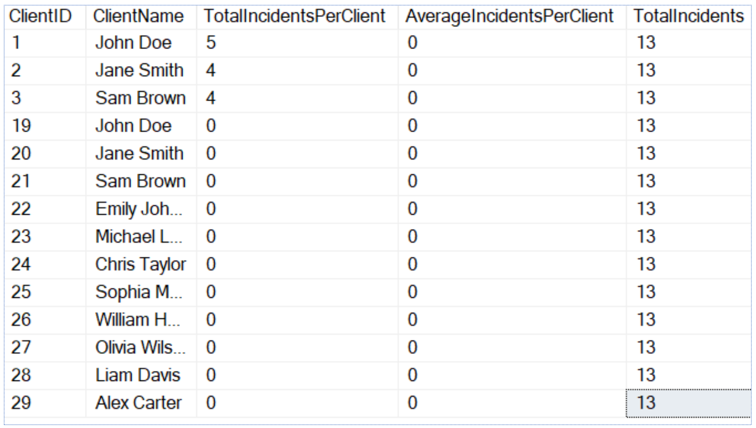
Solution:



```sql
SELECT
    Clients.ClientID,
    Clients.Name AS ClientName,
    COUNT(Incidents.IncidentID) AS TotalIncidentsPerClient,
    AVG(COUNT(Incidents.IncidentID)) OVER () AS AverageIncidentsPerClient,
    SUM(COUNT(Incidents.IncidentID)) OVER () AS TotalIncidents
FROM
    Clients
LEFT JOIN
    Incidents ON Clients.ClientID = Incidents.ClientID
GROUP BY
    Clients.ClientID, Clients.Name
ORDER BY
    TotalIncidentsPerClient DESC;
```

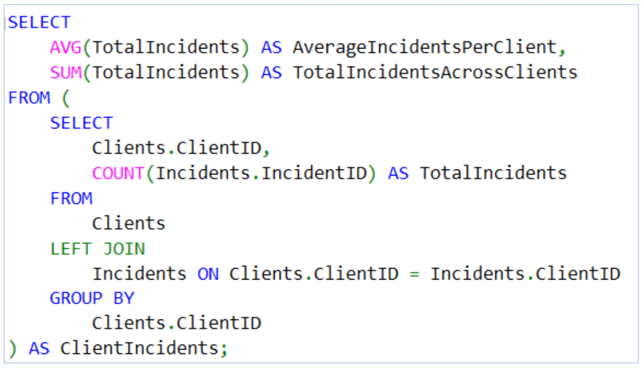
Explanation of the Query:

1. **Columns**:
   - Clients.ClientID, Clients.Name: Identifies each client.
   - COUNT(Incidents.IncidentID): Counts the total number of incidents for each client.
   - AVG(COUNT(Incidents.IncidentID)) OVER (): Calculates the average number of incidents per client using a window function.
   - SUM(COUNT(Incidents.IncidentID)) OVER (): Computes the total number of incidents across all clients.
2. **LEFT JOIN**:
   - Ensures all clients are included, even if they have no incidents.
3. **Grouping**:
   - Groups by ClientID and ClientName to calculate the incident count for each client.
4. **Ordering**:
   - Sorts by TotalIncidentsPerClient in descending order to prioritize clients with the most incidents.



ClientID	ClientName	TotalIncidentsPerClient	AverageIncidentsPerClient	TotalIncidents
1	John Doe	5	0	13
2	Jane Smith	4	0	13
3	Sam Brown	4	0	13
19	John Doe	0	0	13
20	Jane Smith	0	0	13
21	Sam Brown	0	0	13
22	Emily Joh...	0	0	13
23	Michael L...	0	0	13
24	Chris Taylor	0	0	13
25	Sophia M...	0	0	13
26	William H...	0	0	13
27	Olivia Wils...	0	0	13
28	Liam Davis	0	0	13
29	Alex Carter	0	0	13

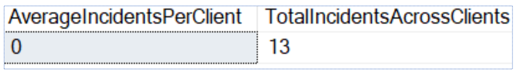
Alternative Query for Overall Averages and Totals:



```sql
SELECT
    AVG(TotalIncidents) AS AverageIncidentsPerClient,
    SUM(TotalIncidents) AS TotalIncidentsAcrossClients
FROM (
    SELECT
        Clients.ClientID,
        COUNT(Incidents.IncidentID) AS TotalIncidents
    FROM
        Clients
    LEFT JOIN
        Incidents ON Clients.ClientID = Incidents.ClientID
    GROUP BY
        Clients.ClientID
) AS ClientIncidents;
```

Explanation of Alternative Query:

1. **Subquery**:
   - Calculates TotalIncidents for each client in the subquery.
2. **Outer Query**:
   - Uses AVG to compute the average number of incidents per client.
   - Uses SUM to compute the total number of incidents across all clients.



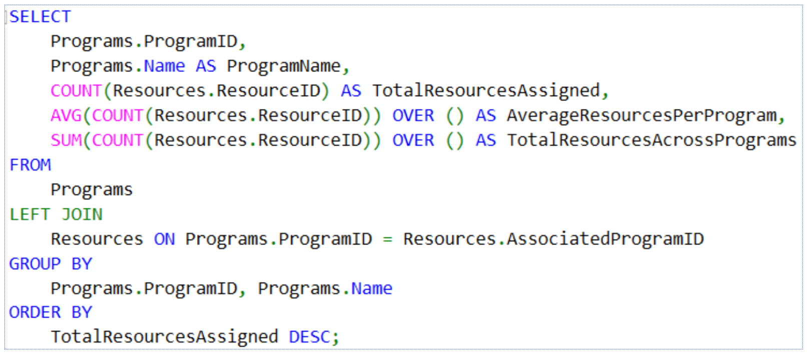
AverageIncidentsPerClient	TotalIncidentsAcrossClients
0	13

6.29) Scenario: Calculate Total and Average Resources Assigned to Programs

Scenario Description:

Management needs to evaluate resource distribution by calculating the total number of resources assigned to programs and the average number of resources per program. This helps identify under-resourced or over-resourced programs.

Solution:

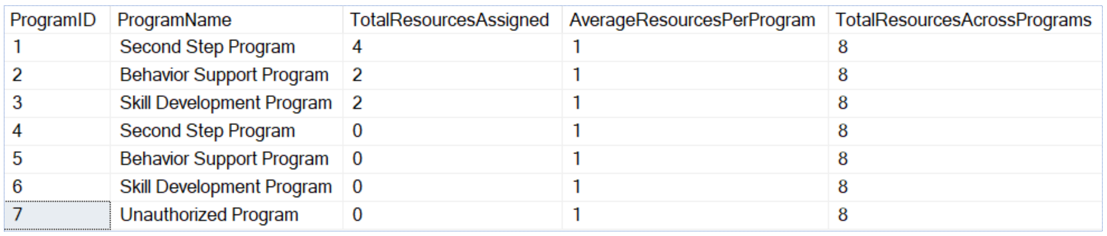


```sql
SELECT
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    COUNT(Resources.ResourceID) AS TotalResourcesAssigned,
    AVG(COUNT(Resources.ResourceID)) OVER () AS AverageResourcesPerProgram,
    SUM(COUNT(Resources.ResourceID)) OVER () AS TotalResourcesAcrossPrograms
FROM
    Programs
LEFT JOIN
    Resources ON Programs.ProgramID = Resources.AssociatedProgramID
GROUP BY
    Programs.ProgramID, Programs.Name
ORDER BY
    TotalResourcesAssigned DESC;
```

- **Programs.ProgramID, Programs.Name**: Identifies each program and its name.
- **COUNT(Resources.ResourceID)**: Counts the total number of resources assigned to each program.

- **AVG(COUNT(Resources.ResourceID)) OVER ()**: Calculates the average number of resources assigned across all programs using a window function.
- **SUM(COUNT(Resources.ResourceID)) OVER ()**: Computes the total number of resources assigned across all programs.
- **LEFT JOIN**: Ensures all programs are included in the analysis, even if they have no resources assigned.
- **Grouping**: Groups by ProgramID and ProgramName to aggregate resource counts for each program.
- **Ordering**: Sorts results by TotalResourcesAssigned in descending order to highlight programs with the highest number of resources.

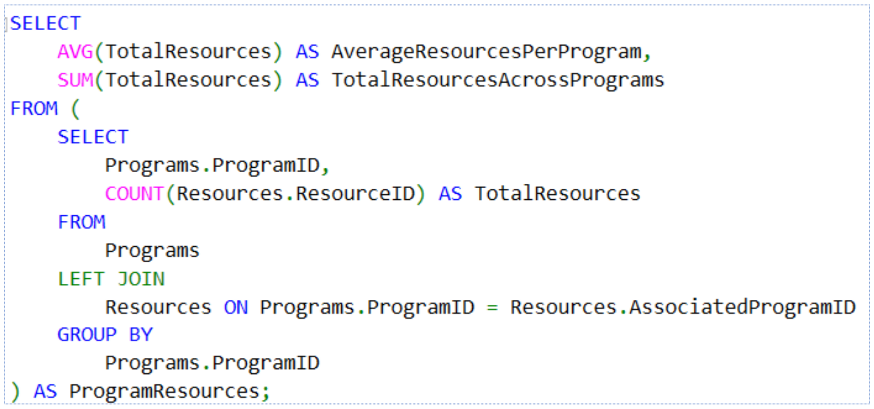
The output is as follows:



ProgramID	ProgramName	TotalResourcesAssigned	AverageResourcesPerProgram	TotalResourcesAcrossPrograms
1	Second Step Program	4	1	8
2	Behavior Support Program	2	1	8
3	Skill Development Program	2	1	8
4	Second Step Program	0	1	8
5	Behavior Support Program	0	1	8
6	Skill Development Program	0	1	8
7	Unauthorized Program	0	1	8

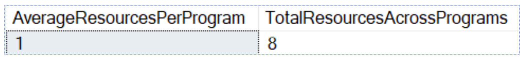
Alternative Query for Overall Metrics Only:

Suppose we need only the total and average resources without program-specific details:



```
SELECT
    AVG(TotalResources) AS AverageResourcesPerProgram,
    SUM(TotalResources) AS TotalResourcesAcrossPrograms
FROM (
    SELECT
        Programs.ProgramID,
        COUNT(Resources.ResourceID) AS TotalResources
    FROM
        Programs
    LEFT JOIN
        Resources ON Programs.ProgramID = Resources.AssociatedProgramID
    GROUP BY
        Programs.ProgramID
) AS ProgramResources;
```

The output is:



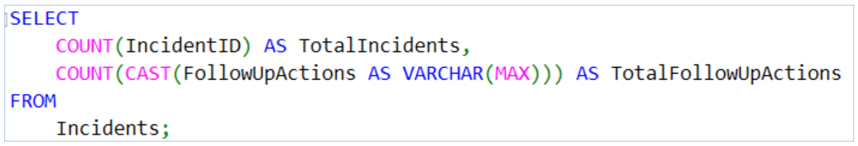
AverageResourcesPerProgram	TotalResourcesAcrossPrograms
1	8

6.30) Scenario: Summing Incident Follow-Ups

Scenario Description:

Management seeks to analyze the total number of follow-up actions taken for incidents across all clients to assess responsiveness and intervention efforts.

Solution:



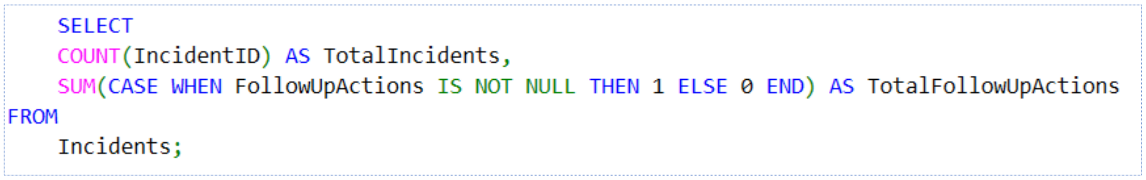
```sql
SELECT
    COUNT(IncidentID) AS TotalIncidents,
    COUNT(CAST(FollowUpActions AS VARCHAR(MAX))) AS TotalFollowUpActions
FROM
    Incidents;
```

Explanation:

- **CAST(FollowUpActions AS VARCHAR(MAX))**: Converts the TEXT column FollowUpActions to VARCHAR(MAX), which is compatible with the COUNT function.
- **COUNT(IncidentID)**: Counts the total number of incidents in the Incidents table.
- **COUNT(CAST(FollowUpActions AS VARCHAR(MAX)))**: Counts the number of non-NULL FollowUpActions entries after converting them to VARCHAR(MAX).

Alternative Query:

We can simply Count Non-NULL Follow-Up Actions without casting



```sql
    SELECT
    COUNT(IncidentID) AS TotalIncidents,
    SUM(CASE WHEN FollowUpActions IS NOT NULL THEN 1 ELSE 0 END) AS TotalFollowUpActions
FROM
    Incidents;
```

6.31) Comprehensive Report on Client and Program Metrics Using Aggregate Functions

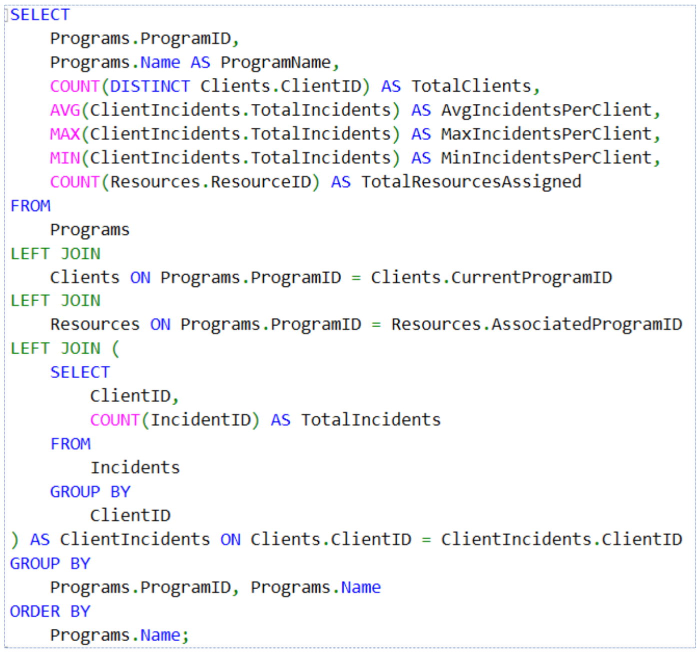
Scenario Description:

Management requires a detailed summary report for each program, including:

1. Total number of clients enrolled.
2. Average number of incidents per client.
3. Maximum and minimum incidents counts among clients.
4. Total resources assigned to each program.

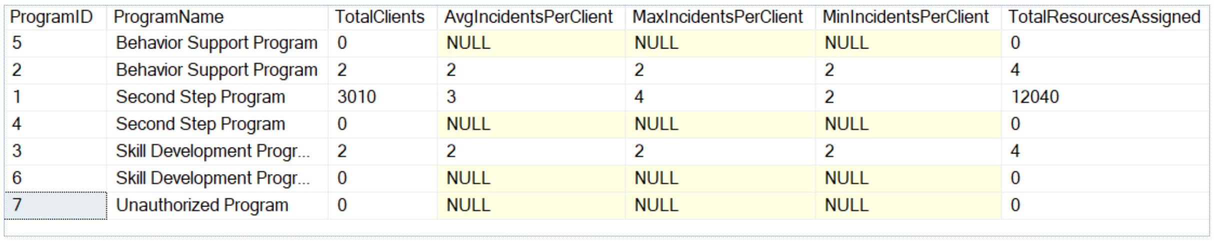
This report combines multiple aggregate functions (COUNT, AVG, MIN, MAX, SUM) to analyze program utilization and client behavior.

Solution:



```sql
SELECT
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    COUNT(DISTINCT Clients.ClientID) AS TotalClients,
    AVG(ClientIncidents.TotalIncidents) AS AvgIncidentsPerClient,
    MAX(ClientIncidents.TotalIncidents) AS MaxIncidentsPerClient,
    MIN(ClientIncidents.TotalIncidents) AS MinIncidentsPerClient,
    COUNT(Resources.ResourceID) AS TotalResourcesAssigned
FROM
    Programs
LEFT JOIN
    Clients ON Programs.ProgramID = Clients.CurrentProgramID
LEFT JOIN
    Resources ON Programs.ProgramID = Resources.AssociatedProgramID
LEFT JOIN (
    SELECT
        ClientID,
        COUNT(IncidentID) AS TotalIncidents
    FROM
        Incidents
    GROUP BY
        ClientID
) AS ClientIncidents ON Clients.ClientID = ClientIncidents.ClientID
GROUP BY
    Programs.ProgramID, Programs.Name
ORDER BY
    Programs.Name;
```

The output is the following:



ProgramID	ProgramName	TotalClients	AvgIncidentsPerClient	MaxIncidentsPerClient	MinIncidentsPerClient	TotalResourcesAssigned
5	Behavior Support Program	0	NULL	NULL	NULL	0
2	Behavior Support Program	2	2	2	2	4
1	Second Step Program	3010	3	4	2	12040
4	Second Step Program	0	NULL	NULL	NULL	0
3	Skill Development Progr...	2	2	2	2	4
6	Skill Development Progr...	0	NULL	NULL	NULL	0
7	Unauthorized Program	0	NULL	NULL	NULL	0

Explanation:

- **Aggregate Functions**: **COUNT(DISTINCT Clients.ClientID)**:
    - Counts the total number of unique clients enrolled in each program.
    - **AVG(ClientIncidents.TotalIncidents)**: Calculates the average number of incidents per client within each program.
    - **MAX(ClientIncidents.TotalIncidents)**: Identifies the client with the maximum number of incidents in the program.
    - **MIN(ClientIncidents.TotalIncidents)**: Identifies the client with the minimum number of incidents in the program.
    - **COUNT(Resources.ResourceID)**: Counts the total number of resources assigned to each program.
    - **Joins**: **Programs to Clients (LEFT JOIN)**: Links programs to their enrolled clients using ProgramID and CurrentProgramID.

        - **Programs to Resources (LEFT JOIN)**: Links programs to their assigned resources using ProgramID and AssociatedProgramID.
    - **Clients to Incidents (LEFT JOIN with Subquery)**:
        - Aggregates incident counts per client in a subquery and links it to clients.
    - **Subquery**: The subquery calculates the total number of incidents (COUNT(IncidentID)) for each client and groups by ClientID.
    - **Grouping**: Groups by Programs.ProgramID and Programs.Name to aggregate data for each program.
    - **Ordering**: Sorts results alphabetically by ProgramName.

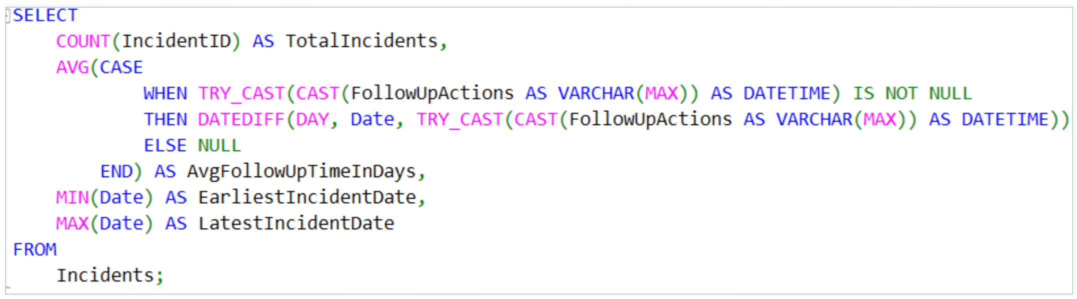
6.32) Scenario: Incident and Follow-Up Analysis

Scenario Description:

Management requires an analysis of all incidents to:

1. Determine the total number of incidents.
2. Calculate the average time taken for follow-up actions.
3. Identify the earliest and latest incident dates.

Solution :



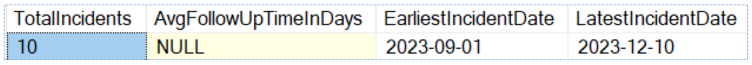
```
SELECT
    COUNT(IncidentID) AS TotalIncidents,
    AVG(CASE
            WHEN TRY_CAST(CAST(FollowUpActions AS VARCHAR(MAX)) AS DATETIME) IS NOT NULL
            THEN DATEDIFF(DAY, Date, TRY_CAST(CAST(FollowUpActions AS VARCHAR(MAX)) AS DATETIME))
            ELSE NULL
        END) AS AvgFollowUpTimeInDays,
    MIN(Date) AS EarliestIncidentDate,
    MAX(Date) AS LatestIncidentDate
FROM
    Incidents;
```

Explanation:

- **Conversion to VARCHAR(MAX)**: **CAST(FollowUpActions AS VARCHAR(MAX))**: Converts the TEXT column to VARCHAR(MAX) to make it compatible with TRY_CAST.
- **TRY_CAST**: **TRY_CAST(... AS DATETIME)**: Attempts to convert the VARCHAR(MAX) value to DATETIME. Invalid values are treated as NULL.
- **CASE Statement**: Ensures that only valid follow-up dates are included in the DATEDIFF calculation.
- **Aggregate Functions**: **AVG**: Calculates the average time in days between the incident date (Date) and valid follow-up actions.

- **COUNT**: Counts the total number of incidents.
- **MIN and MAX**: Identify the earliest and latest incident dates.

The output is

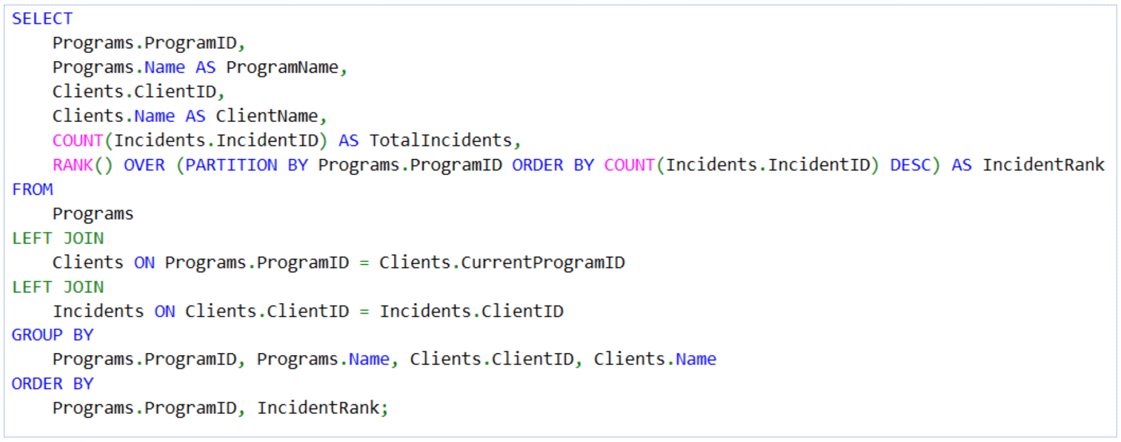


TotalIncidents	AvgFollowUpTimeInDays	EarliestIncidentDate	LatestIncidentDate
10	NULL	2023-09-01	2023-12-10

6.33) Scenario: Ranking Clients by the Number of Incidents Within Each Program

Scenario Description:

Management requires a report to identify the most incident-prone clients within each program. Clients should be ranked based on the number of incidents they have within their respective programs.



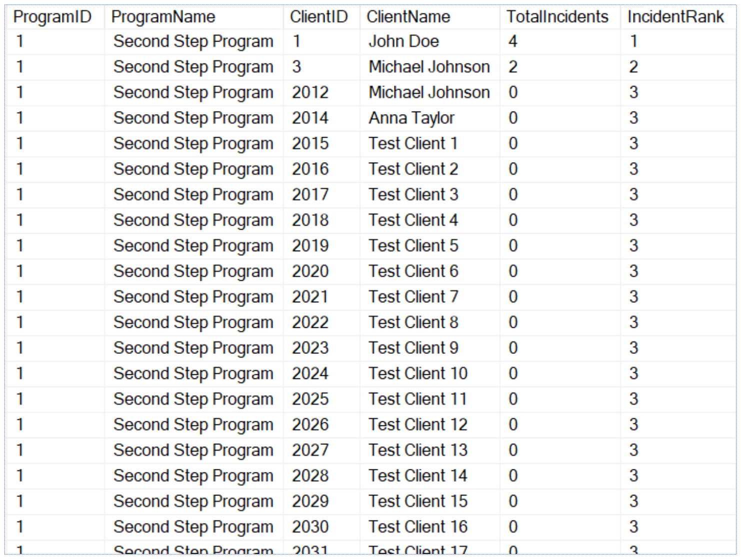
```
SELECT
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    Clients.ClientID,
    Clients.Name AS ClientName,
    COUNT(Incidents.IncidentID) AS TotalIncidents,
    RANK() OVER (PARTITION BY Programs.ProgramID ORDER BY COUNT(Incidents.IncidentID) DESC) AS IncidentRank
FROM
    Programs
LEFT JOIN
    Clients ON Programs.ProgramID = Clients.CurrentProgramID
LEFT JOIN
    Incidents ON Clients.ClientID = Incidents.ClientID
GROUP BY
    Programs.ProgramID, Programs.Name, Clients.ClientID, Clients.Name
ORDER BY
    Programs.ProgramID, IncidentRank;
```

Explanation:

- **Joins**: **LEFT JOIN Programs and Clients**:
  - Links the Programs table to the Clients table using ProgramID and CurrentProgramID.
  - **LEFT JOIN Clients and Incidents**: Links the Clients table to the Incidents table using ClientID.
  - **Selected Columns**: **From Programs**: Includes ProgramID and Name to identify the program.
  - **From Clients**: Includes ClientID and Name to identify the client.
  - **From Incidents**: Counts the number of incidents for each client using COUNT(Incidents.IncidentID).
- **RANK() Function**: **PARTITION BY Programs.ProgramID**:
  - Groups the ranking by program, so clients are ranked within their respective programs.

        - **ORDER BY COUNT(Incidents.IncidentID) DESC**: Orders the clients within each program by the number of incidents in descending order.
    - **Grouping**: Groups by Programs.ProgramID, Programs.Name, Clients.ClientID, and Clients.Name to calculate incident counts per client.
    - **Ordering**: Sorts by Programs.ProgramID and IncidentRank for better readability.

A portion of the output is as follows:

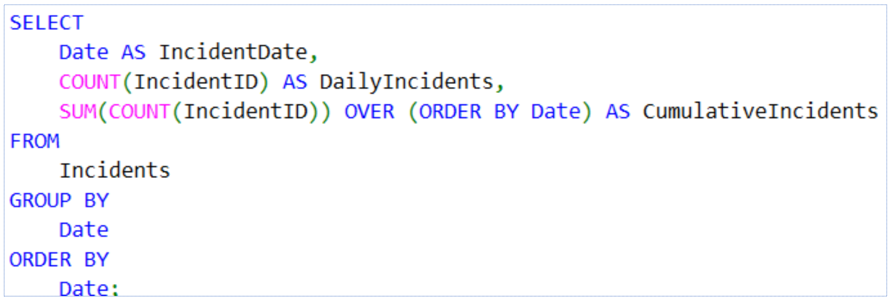


ProgramID	ProgramName	ClientID	ClientName	TotalIncidents	IncidentRank
1	Second Step Program	1	John Doe	4	1
1	Second Step Program	3	Michael Johnson	2	2
1	Second Step Program	2012	Michael Johnson	0	3
1	Second Step Program	2014	Anna Taylor	0	3
1	Second Step Program	2015	Test Client 1	0	3
1	Second Step Program	2016	Test Client 2	0	3
1	Second Step Program	2017	Test Client 3	0	3
1	Second Step Program	2018	Test Client 4	0	3
1	Second Step Program	2019	Test Client 5	0	3
1	Second Step Program	2020	Test Client 6	0	3
1	Second Step Program	2021	Test Client 7	0	3
1	Second Step Program	2022	Test Client 8	0	3
1	Second Step Program	2023	Test Client 9	0	3
1	Second Step Program	2024	Test Client 10	0	3
1	Second Step Program	2025	Test Client 11	0	3
1	Second Step Program	2026	Test Client 12	0	3
1	Second Step Program	2027	Test Client 13	0	3
1	Second Step Program	2028	Test Client 14	0	3
1	Second Step Program	2029	Test Client 15	0	3
1	Second Step Program	2030	Test Client 16	0	3
1	Second Step Program	2031	Test Client 17	0	3

6.34) Scenario: Cumulative Incident Count by Date

Scenario Description:

Management needs to analyze the cumulative number of incidents over time to understand trends and identify peaks in incident frequency.

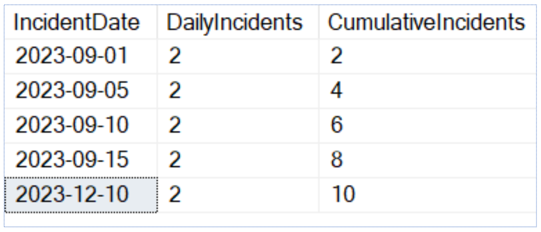


```
SELECT
    Date AS IncidentDate,
    COUNT(IncidentID) AS DailyIncidents,
    SUM(COUNT(IncidentID)) OVER (ORDER BY Date) AS CumulativeIncidents
FROM
    Incidents
GROUP BY
    Date
ORDER BY
    Date;
```

Explanation:

- **COUNT(IncidentID)**: Counts the total number of incidents for each date.
- **SUM(COUNT(IncidentID)) OVER (ORDER BY Date)**: Computes the cumulative sum of incidents up to each date using a window function.
- **GROUP BY Date**: Groups incidents by their occurrence date.

- **Ordering**: Sorts the results by Date to show trends over time.



IncidentDate	DailyIncidents	CumulativeIncidents
2023-09-01	2	2
2023-09-05	2	4
2023-09-10	2	6
2023-09-15	2	8
2023-12-10	2	10

Scenario: Percentile of Resources Assigned to Each Program

Scenario Description:

Management seeks to understand how resource allocation compares across programs by calculating percentiles for the number of resources assigned to each program.

6.35) Scenario: Average Number of Incidents Per Client Across Programs

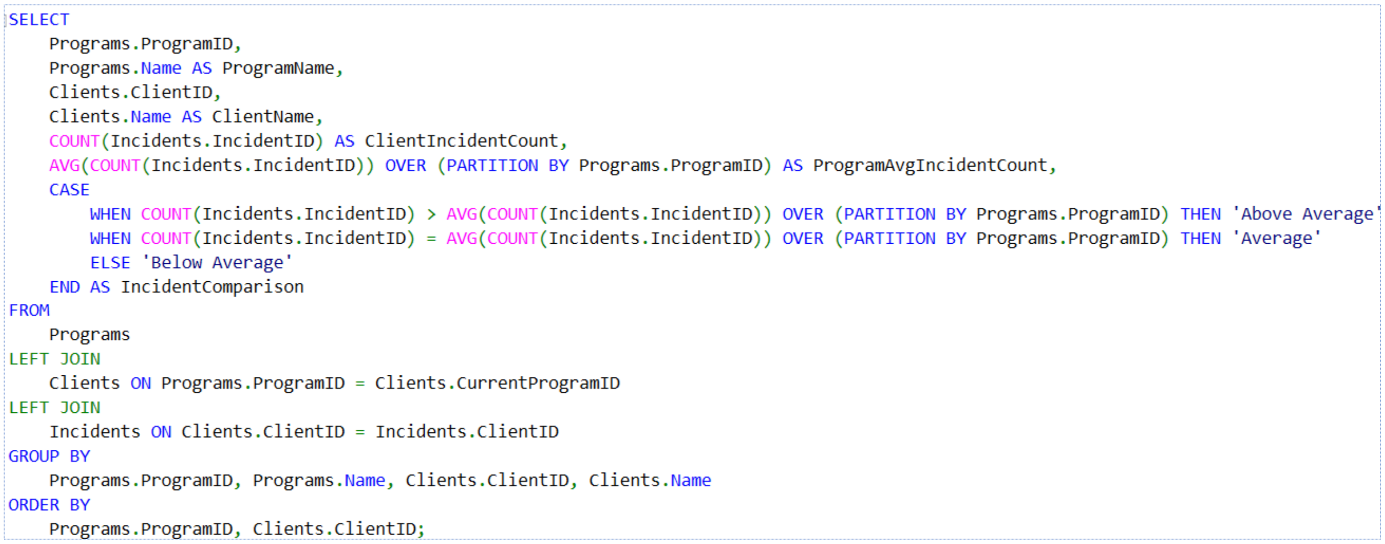
Scenario Description:

Management needs a report that calculates:

1. The average number of incidents per client for each program.
2. Each client's incident count compared to the program's average.

This helps identify clients with unusually high or low incident counts relative to their program peers.

Solution:



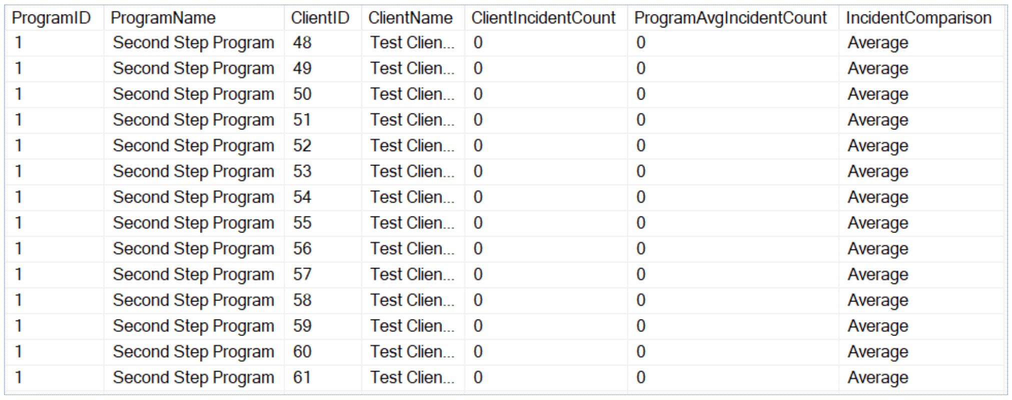
```sql
SELECT
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    Clients.ClientID,
    Clients.Name AS ClientName,
    COUNT(Incidents.IncidentID) AS ClientIncidentCount,
    AVG(COUNT(Incidents.IncidentID)) OVER (PARTITION BY Programs.ProgramID) AS ProgramAvgIncidentCount,
    CASE
        WHEN COUNT(Incidents.IncidentID) > AVG(COUNT(Incidents.IncidentID)) OVER (PARTITION BY Programs.ProgramID) THEN 'Above Average'
        WHEN COUNT(Incidents.IncidentID) = AVG(COUNT(Incidents.IncidentID)) OVER (PARTITION BY Programs.ProgramID) THEN 'Average'
        ELSE 'Below Average'
    END AS IncidentComparison
FROM
    Programs
LEFT JOIN
    Clients ON Programs.ProgramID = Clients.CurrentProgramID
LEFT JOIN
    Incidents ON Clients.ClientID = Incidents.ClientID
GROUP BY
    Programs.ProgramID, Programs.Name, Clients.ClientID, Clients.Name
ORDER BY
    Programs.ProgramID, Clients.ClientID;
```

Explanation:

- **Joins**: **LEFT JOIN Programs and Clients**:
  - Links the Programs table to the Clients table using ProgramID and CurrentProgramID.
  - **LEFT JOIN Clients and Incidents**: Links the Clients table to the Incidents table using ClientID.

- **Aggregate Functions**: **COUNT(Incidents.IncidentID)**:
  - Counts the number of incidents for each client.
- **AVG(COUNT(Incidents.IncidentID)) OVER (PARTITION BY Programs.ProgramID)**:
  - Calculates the average number of incidents per client for each program using a window function.
- **CASE Statement**: Compares each client's incident count to the program's average:
  - **Above Average**: If the client's incident count is greater than the program's average.
  - **Average**: If the client's incident count equals the program's average.
  - **Below Average**: If the client's incident count is less than the program's average.
- **Grouping**: Groups by ProgramID, ProgramName, ClientID, and ClientName to calculate the client-level and program-level metrics.
- **Ordering**: Sorts the results by ProgramID and ClientID for clarity.

A portion of the output is:



ProgramID	ProgramName	ClientID	ClientName	ClientIncidentCount	ProgramAvgIncidentCount	IncidentComparison
1	Second Step Program	48	Test Clien...	0	0	Average
1	Second Step Program	49	Test Clien...	0	0	Average
1	Second Step Program	50	Test Clien...	0	0	Average
1	Second Step Program	51	Test Clien...	0	0	Average
1	Second Step Program	52	Test Clien...	0	0	Average
1	Second Step Program	53	Test Clien...	0	0	Average
1	Second Step Program	54	Test Clien...	0	0	Average
1	Second Step Program	55	Test Clien...	0	0	Average
1	Second Step Program	56	Test Clien...	0	0	Average
1	Second Step Program	57	Test Clien...	0	0	Average
1	Second Step Program	58	Test Clien...	0	0	Average
1	Second Step Program	59	Test Clien...	0	0	Average
1	Second Step Program	60	Test Clien...	0	0	Average
1	Second Step Program	61	Test Clien...	0	0	Average

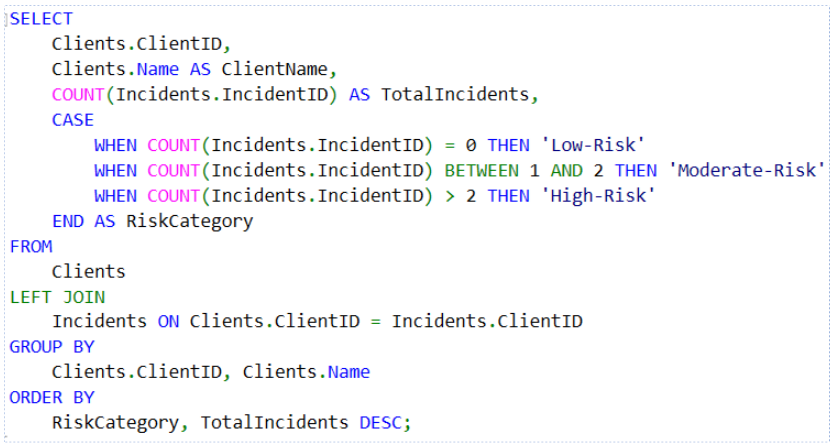
6.36) Categorizing Clients Based on Incident Count

Scenario Description:

Management requires a classification of clients into risk categories based on their total number of incidents:

- **Low-Risk**: 0 incidents.
- **Moderate-Risk**: 1–2 incidents.
- **High-Risk**: More than 2 incidents.

This helps prioritize clients for interventions and resource allocation.

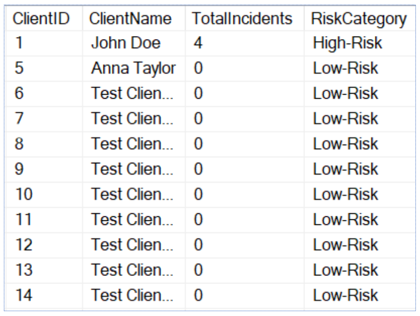


```
SELECT
    Clients.ClientID,
    Clients.Name AS ClientName,
    COUNT(Incidents.IncidentID) AS TotalIncidents,
    CASE
        WHEN COUNT(Incidents.IncidentID) = 0 THEN 'Low-Risk'
        WHEN COUNT(Incidents.IncidentID) BETWEEN 1 AND 2 THEN 'Moderate-Risk'
        WHEN COUNT(Incidents.IncidentID) > 2 THEN 'High-Risk'
    END AS RiskCategory
FROM
    Clients
LEFT JOIN
    Incidents ON Clients.ClientID = Incidents.ClientID
GROUP BY
    Clients.ClientID, Clients.Name
ORDER BY
    RiskCategory, TotalIncidents DESC;
```

Explanation:

- **Joins**: **LEFT JOIN**: Ensures all clients are included, even if they have no incidents.
- **COUNT(Incidents.IncidentID)**: Counts the total number of incidents associated with each client.
- **CASE Statement**: Implements conditional logic to classify clients:
    - **Low-Risk**: Clients with 0 incidents.
    - **Moderate-Risk**: Clients with 1–2 incidents.
    - **High-Risk**: Clients with more than 2 incidents.
- **Grouping**: Groups by ClientID and ClientName to calculate incident counts and classify each client.
- **Ordering**: Sorts by RiskCategory and then by TotalIncidents in descending order for easier prioritization.

A portion of the output is as follows:

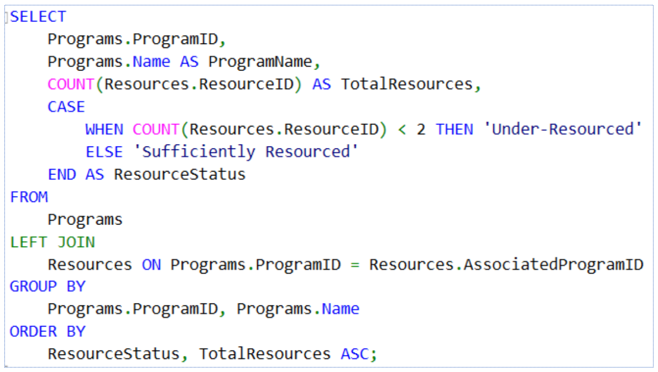


ClientID	ClientName	TotalIncidents	RiskCategory
1	John Doe	4	High-Risk
5	Anna Taylor	0	Low-Risk
6	Test Clien...	0	Low-Risk
7	Test Clien...	0	Low-Risk
8	Test Clien...	0	Low-Risk
9	Test Clien...	0	Low-Risk
10	Test Clien...	0	Low-Risk
11	Test Clien...	0	Low-Risk
12	Test Clien...	0	Low-Risk
13	Test Clien...	0	Low-Risk
14	Test Clien...	0	Low-Risk

6.37) Flagging Programs with Insufficient Resources

Scenario Description:

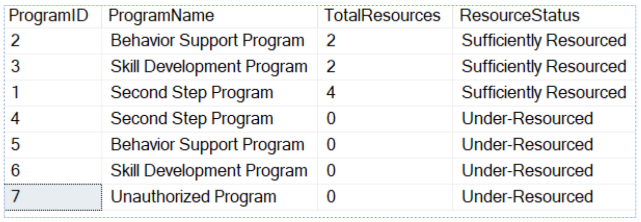
Management wants to identify programs with fewer than two resources and flag them as "Under-Resourced."



```sql
SELECT
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    COUNT(Resources.ResourceID) AS TotalResources,
    CASE
        WHEN COUNT(Resources.ResourceID) < 2 THEN 'Under-Resourced'
        ELSE 'Sufficiently Resourced'
    END AS ResourceStatus
FROM
    Programs
LEFT JOIN
    Resources ON Programs.ProgramID = Resources.AssociatedProgramID
GROUP BY
    Programs.ProgramID, Programs.Name
ORDER BY
    ResourceStatus, TotalResources ASC;
```

Explanation:

- **Joins**: **LEFT JOIN** ensures that all programs are included, even if they have no resources assigned.
- **COUNT(Resources.ResourceID)**: Counts the total number of resources assigned to each program.
- **CASE Statement**: Flags programs with fewer than two resources as **Under-Resourced**.
- Flags others as **Sufficiently Resourced**.
- **Grouping**: Groups by ProgramID and ProgramName to calculate resource counts per program.
- **Ordering**: Sorts by ResourceStatus and TotalResources in ascending order for clarity.

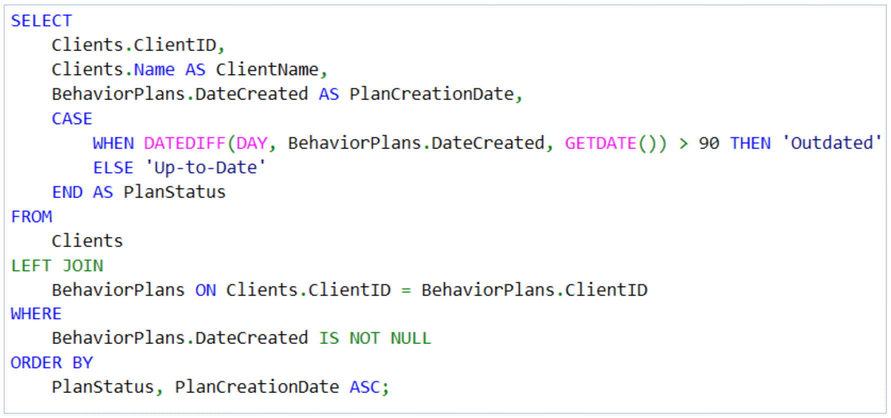


ProgramID	ProgramName	TotalResources	ResourceStatus
2	Behavior Support Program	2	Sufficiently Resourced
3	Skill Development Program	2	Sufficiently Resourced
1	Second Step Program	4	Sufficiently Resourced
4	Second Step Program	0	Under-Resourced
5	Behavior Support Program	0	Under-Resourced
6	Skill Development Program	0	Under-Resourced
7	Unauthorized Program	0	Under-Resourced

6.38) Flagging Clients Who Have Not Updated Behavior Plans

Scenario Description:

Identify clients whose behavior plans were created more than 90 days ago and flag them as "Outdated."

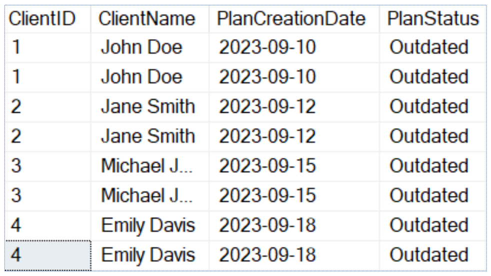


```sql
SELECT
    Clients.ClientID,
    Clients.Name AS ClientName,
    BehaviorPlans.DateCreated AS PlanCreationDate,
    CASE
        WHEN DATEDIFF(DAY, BehaviorPlans.DateCreated, GETDATE()) > 90 THEN 'Outdated'
        ELSE 'Up-to-Date'
    END AS PlanStatus
FROM
    Clients
LEFT JOIN
    BehaviorPlans ON Clients.ClientID = BehaviorPlans.ClientID
WHERE
    BehaviorPlans.DateCreated IS NOT NULL
ORDER BY
    PlanStatus, PlanCreationDate ASC;
```

Explanation:

- **Joins**: **LEFT JOIN** ensures that all clients are included, even if they have no behavior plans.
- **DATEDIFF(DAY, BehaviorPlans.DateCreated, GETDATE())**: Calculates the number of days since the behavior plan was created.
- **CASE Statement**: Flags plans created more than 90 days ago as **Outdated**.
- Flags others as **Up-to-Date**.
- **Filter**: **WHERE BehaviorPlans.DateCreated IS NOT NULL** ensures only clients with behavior plans are included.
- **Ordering**: Sorts by PlanStatus and PlanCreationDate for clarity.

The output is



ClientID	ClientName	PlanCreationDate	PlanStatus
1	John Doe	2023-09-10	Outdated
1	John Doe	2023-09-10	Outdated
2	Jane Smith	2023-09-12	Outdated
2	Jane Smith	2023-09-12	Outdated
3	Michael J...	2023-09-15	Outdated
3	Michael J...	2023-09-15	Outdated
4	Emily Davis	2023-09-18	Outdated
4	Emily Davis	2023-09-18	Outdated

6.39) Comprehensive Program and Client Analysis

Scenario Description:

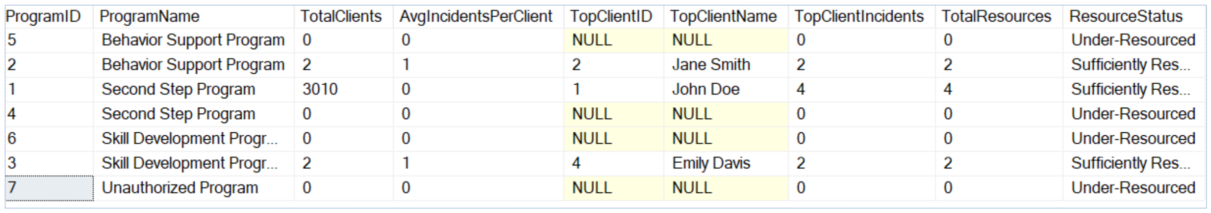
Management needs a detailed report covering:

1. Total clients and their average number of incidents per program.
2. Clients with the most incidents in each program.
3. Programs with the fewest resources flagged as "Under-Resourced."

This query combines grouping, ranking, subqueries, joins, and conditional logic for a comprehensive analysis.

```
WITH ClientIncidentCounts AS (
  SELECT
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    Clients.ClientID,
    Clients.Name AS ClientName,
    COUNT(Incidents.IncidentID) AS TotalIncidents
  FROM
    Programs
  LEFT JOIN
    Clients ON Programs.ProgramID = Clients.CurrentProgramID
  LEFT JOIN
    Incidents ON Clients.ClientID = Incidents.ClientID
  GROUP BY
    Programs.ProgramID, Programs.Name, Clients.ClientID, Clients.Name
),
ProgramClientSummary AS (
  SELECT
    ProgramID,
    ProgramName,
    COUNT(ClientID) AS TotalClients,
    AVG(TotalIncidents) AS AvgIncidentsPerClient
  FROM
    ClientIncidentCounts
  GROUP BY
    ProgramID, ProgramName
),
TopClientsByProgram AS (
  SELECT
    ProgramID,
    ProgramName,
    ClientID,
    ClientName,
    TotalIncidents,
    RANK() OVER (PARTITION BY ProgramID ORDER BY TotalIncidents DESC) AS IncidentRank
  FROM
    ClientIncidentCounts
),
ProgramResourceSummary AS (
  SELECT
    Programs.ProgramID,
    Programs.Name AS ProgramName,
    COUNT(Resources.ResourceID) AS TotalResources,
    CASE
      WHEN COUNT(Resources.ResourceID) < 2 THEN 'Under-Resourced'
      ELSE 'Sufficiently Resourced'
    END AS ResourceStatus
  FROM
    Programs
  LEFT JOIN
    Resources ON Programs.ProgramID = Resources.AssociatedProgramID
  GROUP BY
    Programs.ProgramID, Programs.Name
)
SELECT
  pcs.ProgramID,
  pcs.ProgramName,
  pcs.TotalClients,
  pcs.AvgIncidentsPerClient,
  tcbp.ClientID AS TopClientID,
  tcbp.ClientName AS TopClientName,
  tcbp.TotalIncidents AS TopClientIncidents,
  prs.TotalResources,
  prs.ResourceStatus
FROM
  ProgramClientSummary pcs
LEFT JOIN
  TopClientsByProgram tcbp ON pcs.ProgramID = tcbp.ProgramID AND tcbp.IncidentRank = 1
LEFT JOIN
  ProgramResourceSummary prs ON pcs.ProgramID = prs.ProgramID
ORDER BY
  pcs.ProgramName;
```

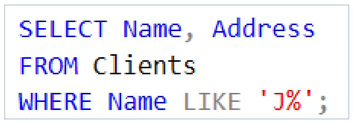
The output is



ProgramID	ProgramName	TotalClients	AvgIncidentsPerClient	TopClientID	TopClientName	TopClientIncidents	TotalResources	ResourceStatus
5	Behavior Support Program	0	0	NULL	NULL	0	0	Under-Resourced
2	Behavior Support Program	2	1	2	Jane Smith	2	2	Sufficiently Res...
1	Second Step Program	3010	0	1	John Doe	4	4	Sufficiently Res...
4	Second Step Program	0	0	NULL	NULL	0	0	Under-Resourced
6	Skill Development Progr...	0	0	NULL	NULL	0	0	Under-Resourced
3	Skill Development Progr...	2	1	4	Emily Davis	2	2	Sufficiently Res...
7	Unauthorized Program	0	0	NULL	NULL	0	0	Under-Resourced

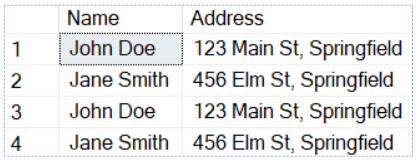
6.40) Using LIKE

Scenario: Find clients with names starting with "J".



```
SELECT Name, Address
FROM Clients
WHERE Name LIKE 'J%';
```

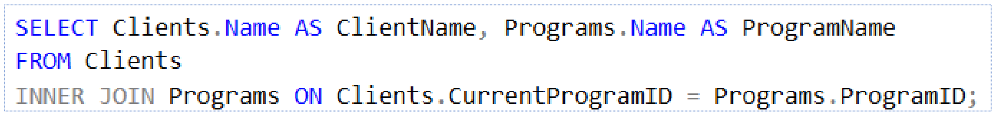
The output is



	Name	Address
1	John Doe	123 Main St, Springfield
2	Jane Smith	456 Elm St, Springfield
3	John Doe	123 Main St, Springfield
4	Jane Smith	456 Elm St, Springfield

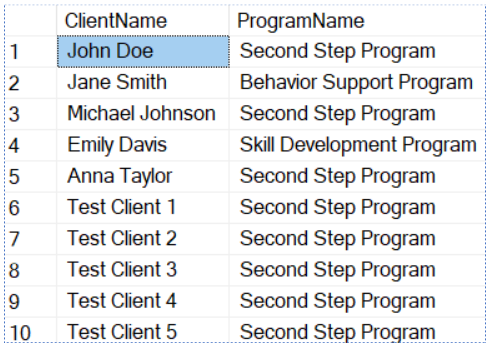
6.41) Using INNER JOIN

Scenario: List all clients and their current program details.



```
SELECT Clients.Name AS ClientName, Programs.Name AS ProgramName
FROM Clients
INNER JOIN Programs ON Clients.CurrentProgramID = Programs.ProgramID;
```

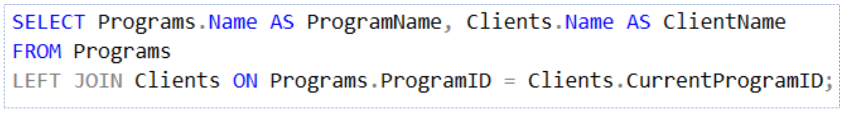
An INNER JOIN combines rows from the Clients and Programs tables based on a matching condition. The condition specified is: "Clients.CurrentProgramID = Programs.ProgramID". This ensures that only rows where a CurrentProgramID in the Clients table matches a ProgramID in the Programs table are included in the result. The portion of the output is



	ClientName	ProgramName
1	John Doe	Second Step Program
2	Jane Smith	Behavior Support Program
3	Michael Johnson	Second Step Program
4	Emily Davis	Skill Development Program
5	Anna Taylor	Second Step Program
6	Test Client 1	Second Step Program
7	Test Client 2	Second Step Program
8	Test Client 3	Second Step Program
9	Test Client 4	Second Step Program
10	Test Client 5	Second Step Program

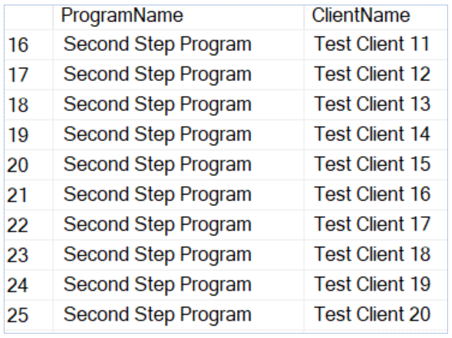
6.42) Using LEFT JOIN

Scenario: Find all programs and the clients associated with them, including programs without any clients.



```
SELECT Programs.Name AS ProgramName, Clients.Name AS ClientName
FROM Programs
LEFT JOIN Clients ON Programs.ProgramID = Clients.CurrentProgramID;
```

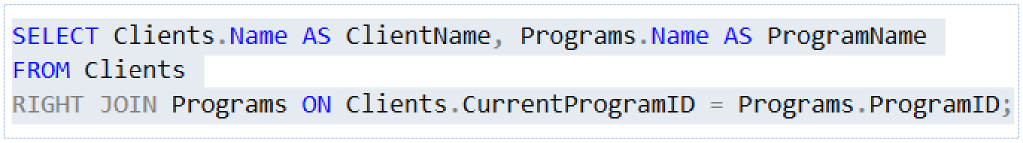
This SQL query uses a RIGHT JOIN to combine data from the Clients and Programs tables, ensuring that all rows from the Programs table are included, even if they do not have a matching record in the Clients table. The portion of the result is shown below:



	ProgramName	ClientName
16	Second Step Program	Test Client 11
17	Second Step Program	Test Client 12
18	Second Step Program	Test Client 13
19	Second Step Program	Test Client 14
20	Second Step Program	Test Client 15
21	Second Step Program	Test Client 16
22	Second Step Program	Test Client 17
23	Second Step Program	Test Client 18
24	Second Step Program	Test Client 19
25	Second Step Program	Test Client 20

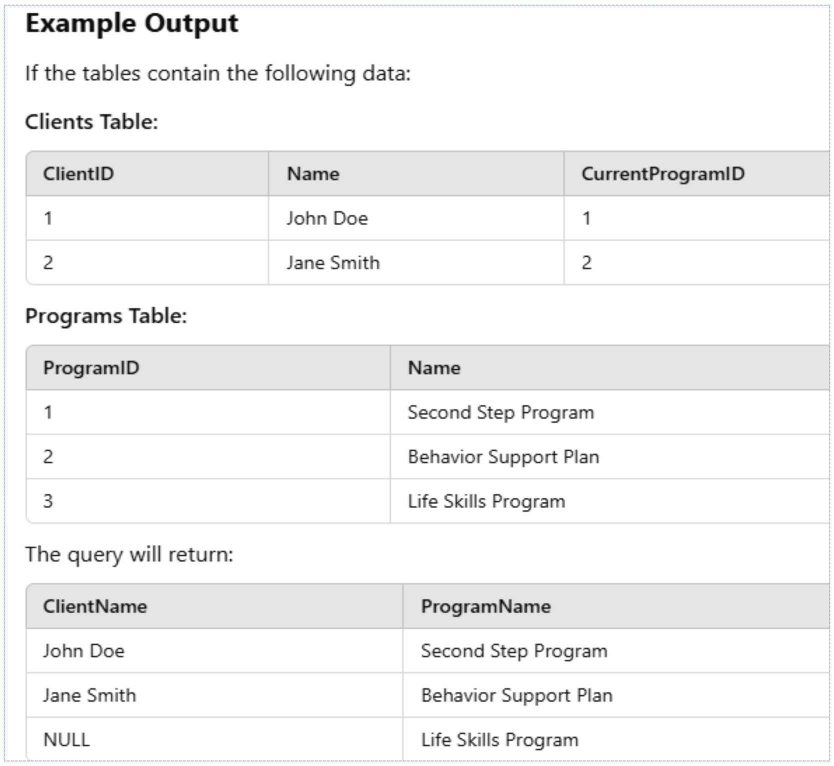
d. Using RIGHT JOIN

Scenario: List all clients and their assigned programs, even if a client is not currently assigned to a program.



```
SELECT Clients.Name AS ClientName, Programs.Name AS ProgramName
FROM Clients
RIGHT JOIN Programs ON Clients.CurrentProgramID = Programs.ProgramID;
```

The A RIGHT JOIN ensures that **all rows from the Programs table** appear in the result, even if there is no matching CurrentProgramID in the Clients table. If no client is associated with a particular program, columns from the Clients table will show NULL.


Example Output

If the tables contain the following data:

Clients Table:

ClientID	Name	CurrentProgramID
1	John Doe	1
2	Jane Smith	2

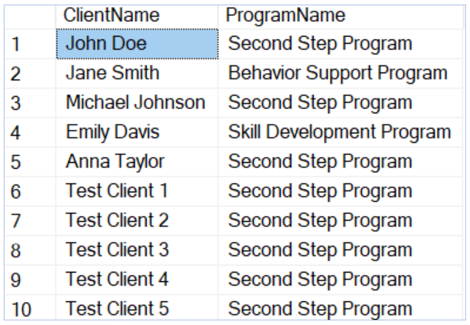
Programs Table:

ProgramID	Name
1	Second Step Program
2	Behavior Support Plan
3	Life Skills Program

The query will return:

ClientName	ProgramName
John Doe	Second Step Program
Jane Smith	Behavior Support Plan
NULL	Life Skills Program

The portion of the output is :



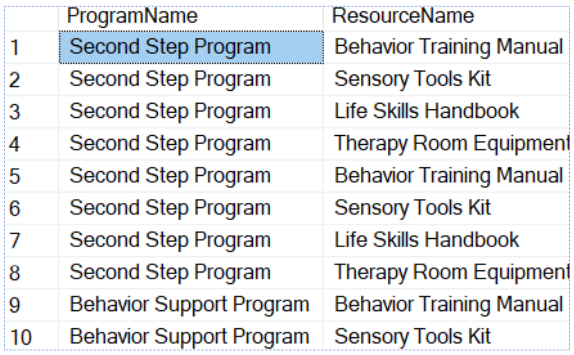
	ClientName	ProgramName
1	John Doe	Second Step Program
2	Jane Smith	Behavior Support Program
3	Michael Johnson	Second Step Program
4	Emily Davis	Skill Development Program
5	Anna Taylor	Second Step Program
6	Test Client 1	Second Step Program
7	Test Client 2	Second Step Program
8	Test Client 3	Second Step Program
9	Test Client 4	Second Step Program
10	Test Client 5	Second Step Program

6.43) Using CROSS JOIN

Scenario: Create a combination of all programs and all resources to analyze potential program-resource assignments.

```
SELECT Programs.Name AS ProgramName, Resources.Name AS ResourceName
FROM Programs
CROSS JOIN Resources;
```

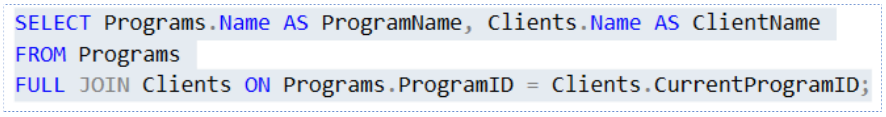
The A CROSS JOIN generates all possible combinations of rows from the two tables. If the Programs table has 3 rows and the Resources table has 4 rows, the result will have 3×4 = 12 rows. The result is:



	ProgramName	ResourceName
1	Second Step Program	Behavior Training Manual
2	Second Step Program	Sensory Tools Kit
3	Second Step Program	Life Skills Handbook
4	Second Step Program	Therapy Room Equipment
5	Second Step Program	Behavior Training Manual
6	Second Step Program	Sensory Tools Kit
7	Second Step Program	Life Skills Handbook
8	Second Step Program	Therapy Room Equipment
9	Behavior Support Program	Behavior Training Manual
10	Behavior Support Program	Sensory Tools Kit

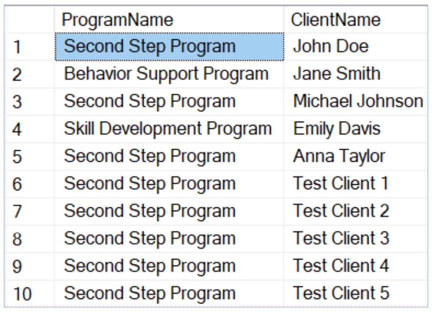
6.44) Using FULL JOIN

Scenario: Show all programs and clients, including unassigned programs or clients.



```sql
SELECT Programs.Name AS ProgramName, Clients.Name AS ClientName
FROM Programs
FULL JOIN Clients ON Programs.ProgramID = Clients.CurrentProgramID;
```

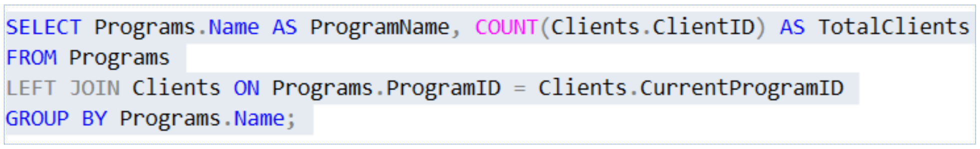
The Full Join combines rows from both tables. It includes all rows from the Programs table, even if there are no matching rows in the Clients table, and also all rows from the Clients table, even if there are no matching rows in the Programs table. If no match exists, the corresponding columns from the unmatched table will be NULL. The portion of the output is shown below:



	ProgramName	ClientName
1	Second Step Program	John Doe
2	Behavior Support Program	Jane Smith
3	Second Step Program	Michael Johnson
4	Skill Development Program	Emily Davis
5	Second Step Program	Anna Taylor
6	Second Step Program	Test Client 1
7	Second Step Program	Test Client 2
8	Second Step Program	Test Client 3
9	Second Step Program	Test Client 4
10	Second Step Program	Test Client 5

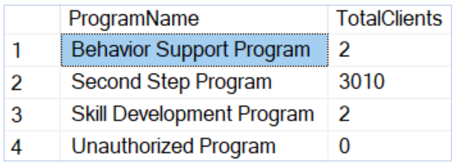
6.45) Using COUNT

Scenario: Count the total number of clients enrolled in each program.



```sql
SELECT Programs.Name AS ProgramName, COUNT(Clients.ClientID) AS TotalClients
FROM Programs
LEFT JOIN Clients ON Programs.ProgramID = Clients.CurrentProgramID
GROUP BY Programs.Name;
```

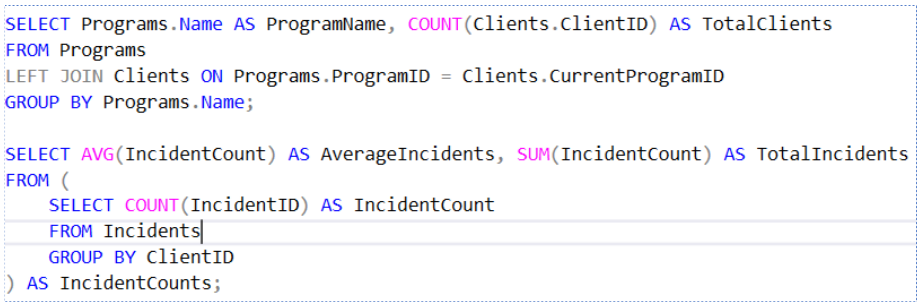
The output is



	ProgramName	TotalClients
1	Behavior Support Program	2
2	Second Step Program	3010
3	Skill Development Program	2
4	Unauthorized Program	0

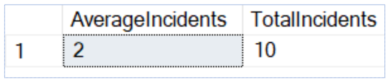
6.46) Using AVERAGE, SUM

Scenario: Calculate the average number of incidents per client and the total number of incidents recorded.



```sql
SELECT Programs.Name AS ProgramName, COUNT(Clients.ClientID) AS TotalClients
FROM Programs
LEFT JOIN Clients ON Programs.ProgramID = Clients.CurrentProgramID
GROUP BY Programs.Name;

SELECT AVG(IncidentCount) AS AverageIncidents, SUM(IncidentCount) AS TotalIncidents
FROM (
    SELECT COUNT(IncidentID) AS IncidentCount
    FROM Incidents
    GROUP BY ClientID
) AS IncidentCounts;
```

This query calculates the average number of incidents per client as well as the total number of incidents. The output is displayed below:

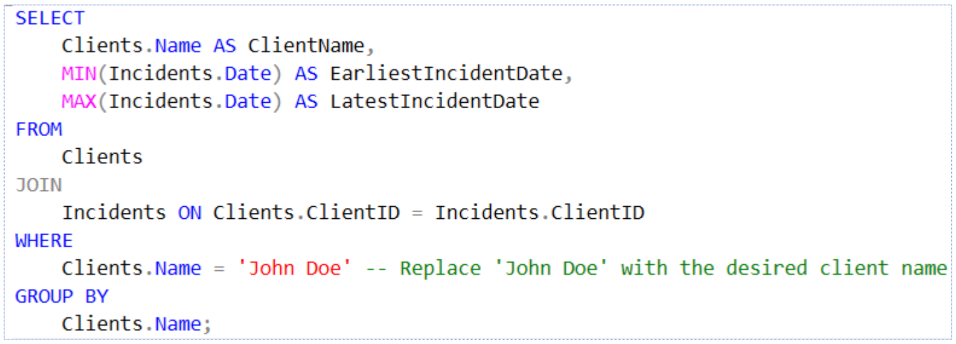


	AverageIncidents	TotalIncidents
1	2	10

6.47) Using Aggregate Functions

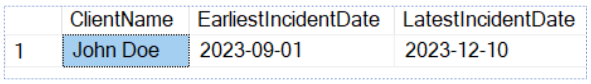
Scenario: Finding the Earliest and Latest Incident Dates for a Given Client named John Doe

This query helps in understanding the history of a client, which can be critical for creating tailored behavior plans or tracking progress over time.



```sql
SELECT
    Clients.Name AS ClientName,
    MIN(Incidents.Date) AS EarliestIncidentDate,
    MAX(Incidents.Date) AS LatestIncidentDate
FROM
    Clients
JOIN
    Incidents ON Clients.ClientID = Incidents.ClientID
WHERE
    Clients.Name = 'John Doe' -- Replace 'John Doe' with the desired client name
GROUP BY
    Clients.Name;
```

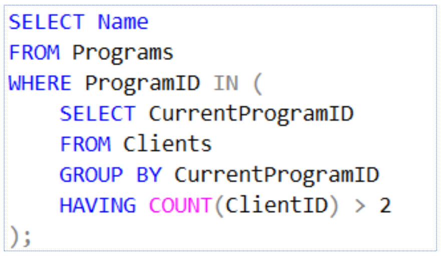
The output is as follows:



	ClientName	EarliestIncidentDate	LatestIncidentDate
1	John Doe	2023-09-01	2023-12-10

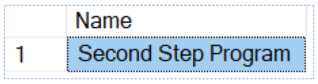
15. Using a Subquery

Scenario: List of all programs with more than 2 clients enrolled.



```sql
SELECT Name
FROM Programs
WHERE ProgramID IN (
    SELECT CurrentProgramID
    FROM Clients
    GROUP BY CurrentProgramID
    HAVING COUNT(ClientID) > 2
);
```

The result is the following:



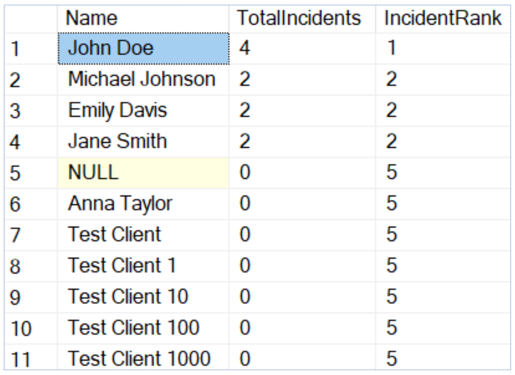
	Name
1	Second Step Program

6.48) Window Functions

Scenario: Rank clients by the number of incidents they have.

```sql
SELECT Name,
       COUNT(Incidents.IncidentID) AS TotalIncidents,
       RANK() OVER (ORDER BY COUNT(Incidents.IncidentID) DESC) AS IncidentRank
FROM Clients
LEFT JOIN Incidents ON Clients.ClientID = Incidents.ClientID
GROUP BY Clients.Name;
```

The portion of results is displayed below:



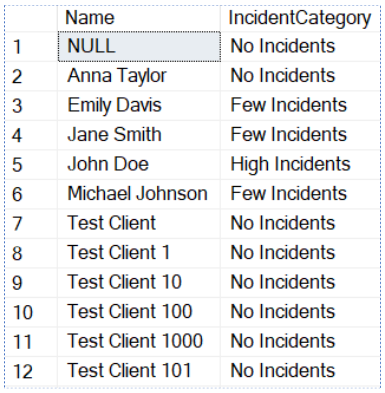
	Name	TotalIncidents	IncidentRank
1	John Doe	4	1
2	Michael Johnson	2	2
3	Emily Davis	2	2
4	Jane Smith	2	2
5	NULL	0	5
6	Anna Taylor	0	5
7	Test Client	0	5
8	Test Client 1	0	5
9	Test Client 10	0	5
10	Test Client 100	0	5
11	Test Client 1000	0	5

6.49) Using IF

Scenario: Categorize clients based on the number of incidents they have.

```sql
SELECT Name,
       CASE
           WHEN COUNT(Incidents.IncidentID) = 0 THEN 'No Incidents'
           WHEN COUNT(Incidents.IncidentID) <= 3 THEN 'Few Incidents'
           ELSE 'High Incidents'
       END AS IncidentCategory
FROM Clients
LEFT JOIN Incidents ON Clients.ClientID = Incidents.ClientID
GROUP BY Clients.Name;
```

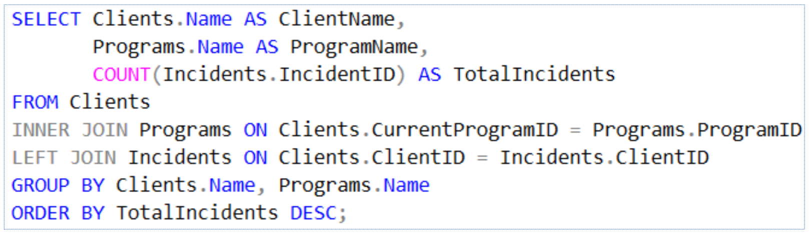
The portion of the output is the following:



	Name	IncidentCategory
1	NULL	No Incidents
2	Anna Taylor	No Incidents
3	Emily Davis	Few Incidents
4	Jane Smith	Few Incidents
5	John Doe	High Incidents
6	Michael Johnson	Few Incidents
7	Test Client	No Incidents
8	Test Client 1	No Incidents
9	Test Client 10	No Incidents
10	Test Client 100	No Incidents
11	Test Client 1000	No Incidents
12	Test Client 101	No Incidents

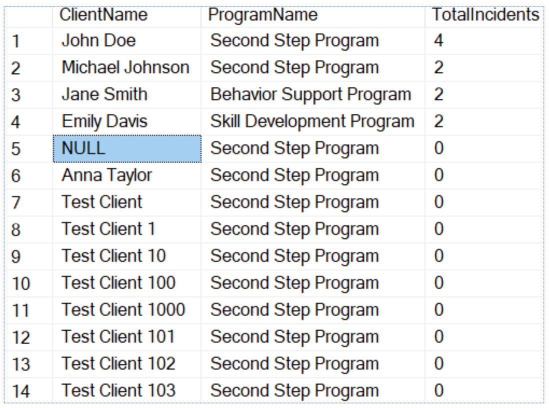
6.50) 1Using Complex Queries

Scenario: List of clients with their program details and the count of incidents, ordered by the number of incidents.



```
SELECT Clients.Name AS ClientName,
       Programs.Name AS ProgramName,
       COUNT(Incidents.IncidentID) AS TotalIncidents
FROM Clients
INNER JOIN Programs ON Clients.CurrentProgramID = Programs.ProgramID
LEFT JOIN Incidents ON Clients.ClientID = Incidents.ClientID
GROUP BY Clients.Name, Programs.Name
ORDER BY TotalIncidents DESC;
```

The partial of the results is the following:



	ClientName	ProgramName	TotalIncidents
1	John Doe	Second Step Program	4
2	Michael Johnson	Second Step Program	2
3	Jane Smith	Behavior Support Program	2
4	Emily Davis	Skill Development Program	2
5	NULL	Second Step Program	0
6	Anna Taylor	Second Step Program	0
7	Test Client	Second Step Program	0
8	Test Client 1	Second Step Program	0
9	Test Client 10	Second Step Program	0
10	Test Client 100	Second Step Program	0
11	Test Client 1000	Second Step Program	0
12	Test Client 101	Second Step Program	0
13	Test Client 102	Second Step Program	0
14	Test Client 103	Second Step Program	0

7. Expected Benefits

The implementation of this project brings numerous advantages that span across operational efficiency, decision-making, quality of care, regulatory compliance, and scalability. By leveraging optimized processes, data-driven insights, and advanced tools, organizations can streamline workflows, enhance patient outcomes, and meet both current and future needs. The following subsections outline the specific benefits anticipated from this initiative.

7.1 Efficiency

Efficiency is one of the core benefits of this project, as it aims to streamline workflows and automate repetitive tasks. By minimizing manual interventions, the likelihood of errors is reduced, allowing for faster and more accurate data processing. The optimized processes ensure that resources such as time and effort are utilized effectively, freeing up staff to focus on higher-priority tasks. Additionally, improved data retrieval mechanisms make it easier for users to access critical information promptly, thereby enhancing overall productivity and operational performance.

7.2 Decision-Making

Improved decision-making is another significant outcome, driven by the availability of accurate, real-time data. The integration of advanced analytics and reporting tools provides actionable insights that enable organizations to make data-driven decisions confidently. Predictive models further support proactive strategies, helping to identify trends, patterns, and areas requiring intervention. With enhanced visibility into key performance metrics, stakeholders can make timely and informed decisions that drive organizational success and improve outcomes.

7.3 Care Quality

The project directly contributes to improved care quality by ensuring the accuracy and reliability of patient data. Streamlined data management enables healthcare providers to access comprehensive and up-to-date records, leading to better diagnoses and treatment plans. Predictive analytics also play a vital role in identifying at-risk patients, allowing for timely interventions and preventive care. By reducing administrative burdens, healthcare professionals can dedicate more time to patient care, ultimately enhancing satisfaction and outcomes for both patients and providers.

7.4 Compliance

Compliance with regulatory requirements is a critical benefit of this project, ensuring adherence to industry standards and legal frameworks such as HIPAA, GDPR, and other applicable guidelines. Robust data management practices, including secure access controls and audit trails, provide transparency and accountability in operations. The system helps mitigate risks associated with non-compliance, such as penalties and reputational damage, by maintaining data integrity and enabling accurate reporting. This proactive approach to compliance builds trust and confidence among stakeholders.

7.5 Efficiency

Efficiency is one of the core benefits of this project, as it aims to streamline workflows and automate repetitive tasks. By minimizing manual interventions, the likelihood of errors is reduced, allowing for faster and more accurate data processing. The optimized processes ensure that resources such as time and effort are utilized effectively, freeing up staff to focus on higher-priority tasks. Additionally, improved data retrieval mechanisms make it easier for users to access critical information promptly, thereby enhancing overall productivity and operational performance.

7.6 Scalability

Scalability ensures that the project can adapt to growing organizational demands and evolving technologies. The infrastructure is designed to handle increasing data volumes and accommodate more users without compromising performance. The system's flexibility allows seamless integration with other tools and platforms, ensuring future compatibility and functionality expansion. This ability to scale effectively prepares the organization for long-term growth, enabling it to meet current challenges while being ready to embrace future innovations and opportunities.

8. General Conclusion

This project initiative serves as a comprehensive solution to current challenges while laying a strong foundation for future success. It empowers Raise and Grow organization to remain competitive, responsive, and efficient in an ever-evolving environment, driving meaningful and sustainable outcomes.