2023

Database Vaccinations Report

COURSEWORK - PORTFOLIO

MARGARITA GRISHECHKINA
ARA | TE PUKENGA

Table of Contents

Stored Procedures

Task

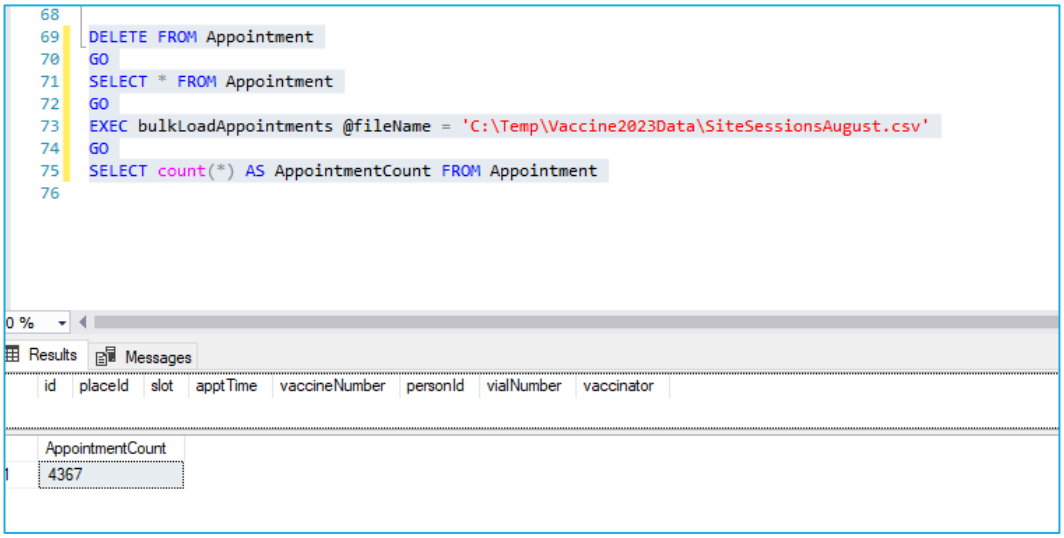
Database developer was required to support a vaccination booking system where individuals may schedule one or multiple appointments, potentially at different locations, depending on their vaccination needs. While the interface handling user input was outside the scope of this project, the responsibility included creating the backend stored procedures to be consumed by that interface.

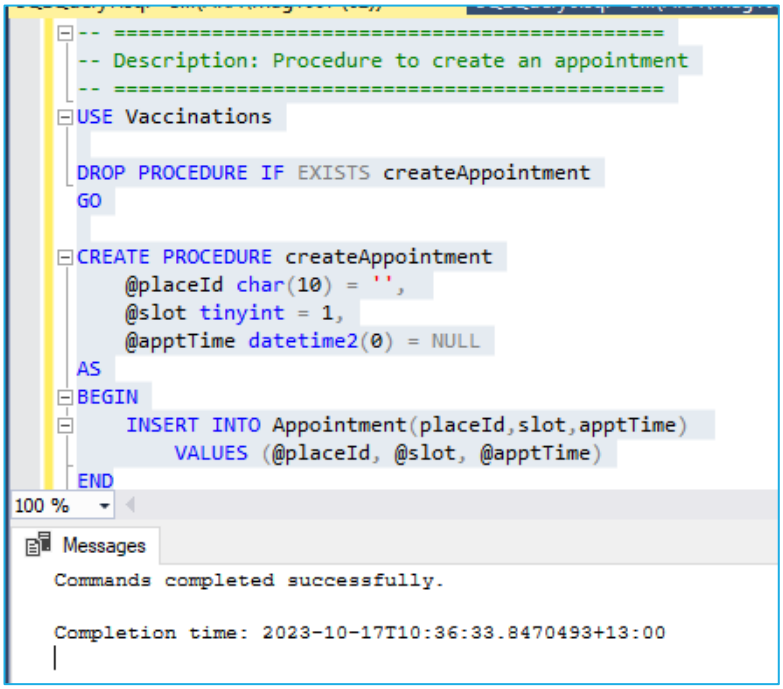
Some stored procedures were inherited, while others—specifically those for bulk loading vaccinators and appointments—were designed and implemented from scratch.

Evidence

Procedure to create an Appointment

A stored procedure was implemented to allow the creation of individual vaccination appointments, enabling integration with the external booking interface.





Procedure to Bulk Load Vaccinator

A dedicated stored procedure was developed to efficiently insert multiple vaccinator records into the database, supporting operational scalability.

```
-- Description: Create Procedure to Bulk Load Vaccinators
-- ============================================

-- Write your code here

USE Vaccinations
GO
DROP PROCEDURE IF EXISTS bulkLoadVaccinators
GO

CREATE PROCEDURE [dbo].[bulkLoadVaccinators]
    -- Loads a file of places into the Vaccination table.  No error checking.
    @fileName varchar(100) = '' -- The file naem where the data is being loaded from
AS
BEGIN
    DECLARE @sqlString varchar(512) = '' -- The string that is being executed to bulk load the data

    --  Create the temporary table to load teh data by copying the structure
    DROP TABLE IF EXISTS #newVaccinators
    SELECT TOP 0 * INTO #newVaccinators FROM Vaccinator

    -- Create and execute teh sql string.  The quotes are important
    SET @sqlString = '
    BULK INSERT #newVaccinators
    FROM '''  + @fileName + '''
    WITH (
        FIRSTROW = 2,
        FIELDTERMINATOR = '','',
        ROWTERMINATOR=''\n''
    )'
    EXECUTE (@sqlString)

    -- Use a cursor to loop through teh temporary table, and load the data into the permanent table
    -- Variables data read from
    DECLARE @iRDNumber char(10)
    DECLARE @preferredName nchar(20)

    -- the cursor declared and opened
    DECLARE newVaccinatorsCursor CURSOR FOR
        SELECT iRDNumber FROM #newVaccinators
    OPEN newVaccinatorsCursor

    -- Loop through the cursor until there is no more data
```

```
    -- Loop through the cursor until there is no more data
    FETCH NEXT FROM newVaccinatorsCursor INTO @iRDNumber
    WHILE @@FETCH_STATUS = 0
        BEGIN
            -- Get teh data from teh temporary table
            SELECT @preferredName = preferredName
                FROM #newVaccinators
                WHERE iRDNumber = @iRDNumber
            -- Run teh procedure that enters the data
            EXEC createVaccinator @iRDNumber=@iRDNumber, @preferredName=@preferredName
            -- Get the next lot of data
            FETCH NEXT FROM newVaccinatorsCursor INTO @iRDNumber
        END

    -- Tidy up
    CLOSE newVaccinatorsCursor
    DEALLOCATE newVaccinatorsCursor
    DROP TABLE #newVaccinators
END
%
Messages
The module 'bulkLoadVaccinators' depends on the missing object 'createVaccinator'. The module will still be created; however, it cannot run successfully until the object exis

Completion time: 2023-10-17T10:48:34.6612265+13:00
```

Procedure to create a New Vaccinator

This procedure allows the insertion of a single vaccinator into the system, ensuring proper validation and data integrity.

```
-- ============================================
-- Description: Procedure to create a New Vaccinator
-- ============================================

USE Vaccinations
GO

DROP PROCEDURE IF EXISTS createVaccinator
GO

CREATE PROCEDURE createVaccinator
    @iRDNumber char(10) = '',
    @preferredName nchar(20) = ''
AS
BEGIN
    INSERT INTO Vaccinator(iRDNumber, preferredName)
    VALUES (@iRDNumber, @preferredName)
END
GO

-- Test the createVaccinator procedure
EXEC createVaccinator @iRDNumber = 'IRD001', @preferredName = 'Dr. Smith'
GO

-- Check if the vaccinator was created successfully
SELECT * FROM Vaccinator WHERE iRDNumber = 'IRD001'
```

```
Results  Messages

(1 row affected)

(1 row affected)

Completion time: 2023-10-17T11:22:00.1493037+13:00
```

Results Messages

	iRDNumber	preferredName
1	IRD001	Dr. Smith

Backup

Task

To ensure continuous availability and data integrity in a high-usage database environment, a comprehensive backup strategy has been defined. This includes policies and implementation procedures for full, differential, and transaction log backups, as well as replication configuration management.

Backup Strategy

Full Backups

- A full database backup is scheduled weekly on Sundays at midnight. This timing is chosen to align with reduced system usage and minimal user activity, thus minimizing operational disruptions.
- Each full backup is retained for 4 weeks to allow for historical recovery, long-term data integrity checks, and alignment with regulatory and disaster recovery requirements.

Differential Backups

- Differential backups are scheduled daily from Monday to Saturday at midnight, capturing changes since the last full backup.
- Each differential backup is retained for 7 days, ensuring adequate restore points for the week while optimizing storage use.

Transaction Log Backups

- Transaction logs are backed up every hour to support point-in-time recovery and minimize data loss risk.
- These backups are retained for 48 hours to cover short-term recovery windows without incurring excessive storage costs.

Replication Configuration

- If database replication is enabled, configuration settings are backed up manually following any significant change.
- Script generation using SQL Server Management Studio (SSMS) or Replication Monitor is recommended. These scripts are to be securely stored for rapid recovery in the event of replication failure.

Storage and Retention Policy

- All backups are stored on a separate storage volume to ensure isolation from the primary database environment.
- A dual-storage approach is advised: local (on-premises) and off-site (cloud-based, such as Azure Blob Storage).
- Redundant and geographically separated storage locations are recommended for disaster recovery preparedness.

Tutorial for Using SQL Server Management Studio (SSMS):

1. Open SQL Server Management Studio.
2. Connect to your SQL Server instance.
3. In the Object Explorer, expand the server node and then expand the "Replication" node.
4. Right-click on the replication type you want to script (e.g., "Local Publications" for transactional replication, or "Distributed Transactions" for peer-to-peer replication) and choose "Generate Scripts."
5. Follow the wizard to generate scripts for various replication components. This includes publication, articles, subscriptions, and more.
6. Customize the script generation options as needed, such as scripting for creating, dropping, or altering replication objects.

7. Review and save the generated scripts to a file.
8. Store the script files in a secure location for backup purposes.

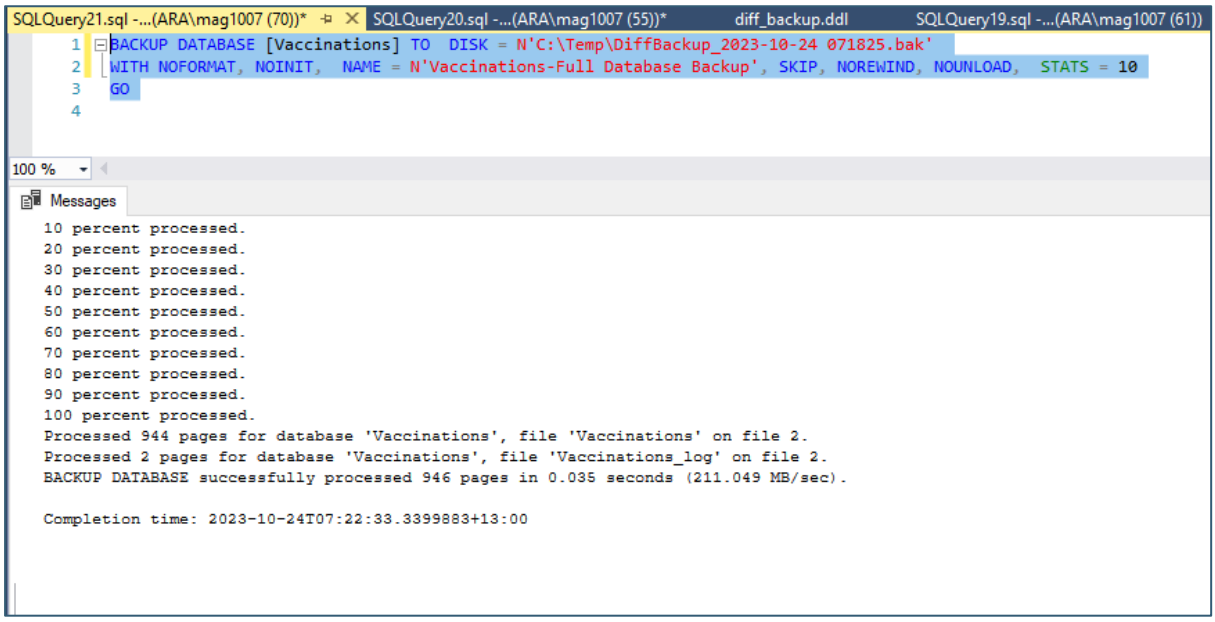
Tutorial for Using Replication Monitor:

1. Open SQL Server Management Studio.
2. Connect to your SQL Server instance.
3. In the Object Explorer, expand the server node and then expand the "Replication" node.
4. Right-click on the "Local Publications" or "Distributed Transactions" node, depending on the type of replication you're using.
5. Select "View Replication Monitor."
6. In Replication Monitor, you can access various functionalities for monitoring and managing replication.
7. To generate scripts, right-click on the publication you want to script and choose "Script Publication as."
8. Select the options for the scripting, including articles, subscriptions, and other configuration settings.
9. Review and save the generated scripts to a file.
10. Store the script files in a secure location for backup purposes.

Backup Implementation Scripts

Full backups script

A SQL script is implemented to automate weekly full backups with timestamped file naming.



Differential backup script

A SQL script to perform differential backups, scheduled daily (Monday–Saturday).

```sql
-- Perform a differential backup of the database
USE Vaccinations;
GO

-- Check if it's a valid day for a differential backup (Monday to Saturday)
DECLARE @CurrentDayOfWeek INT = DATEPART(WEEKDAY, GETDATE());

IF @CurrentDayOfWeek BETWEEN 2 AND 7
BEGIN
    -- Construct the backup file name with date and time
    DECLARE @BackupFileName NVARCHAR(255) = N'C:\Temp\DiffBackup_' + REPLACE(CONVERT(NVARCHAR, GETDATE(), 120), ':', '') + '.bak';

    -- Perform the differential backup
    BACKUP DATABASE Vaccinations
    TO DISK = @BackupFileName
    WITH DIFFERENTIAL;
END;
```

```
Messages
Processed 456 pages for database 'Vaccinations', file 'Vaccinations' on file 1.
Processed 2 pages for database 'Vaccinations', file 'Vaccinations_log' on file 1.
BACKUP DATABASE WITH DIFFERENTIAL successfully processed 458 pages in 0.023 seconds (155.400 MB/sec).

Completion time: 2023-10-24T07:18:25.2623272+13:00
```

Log backup script

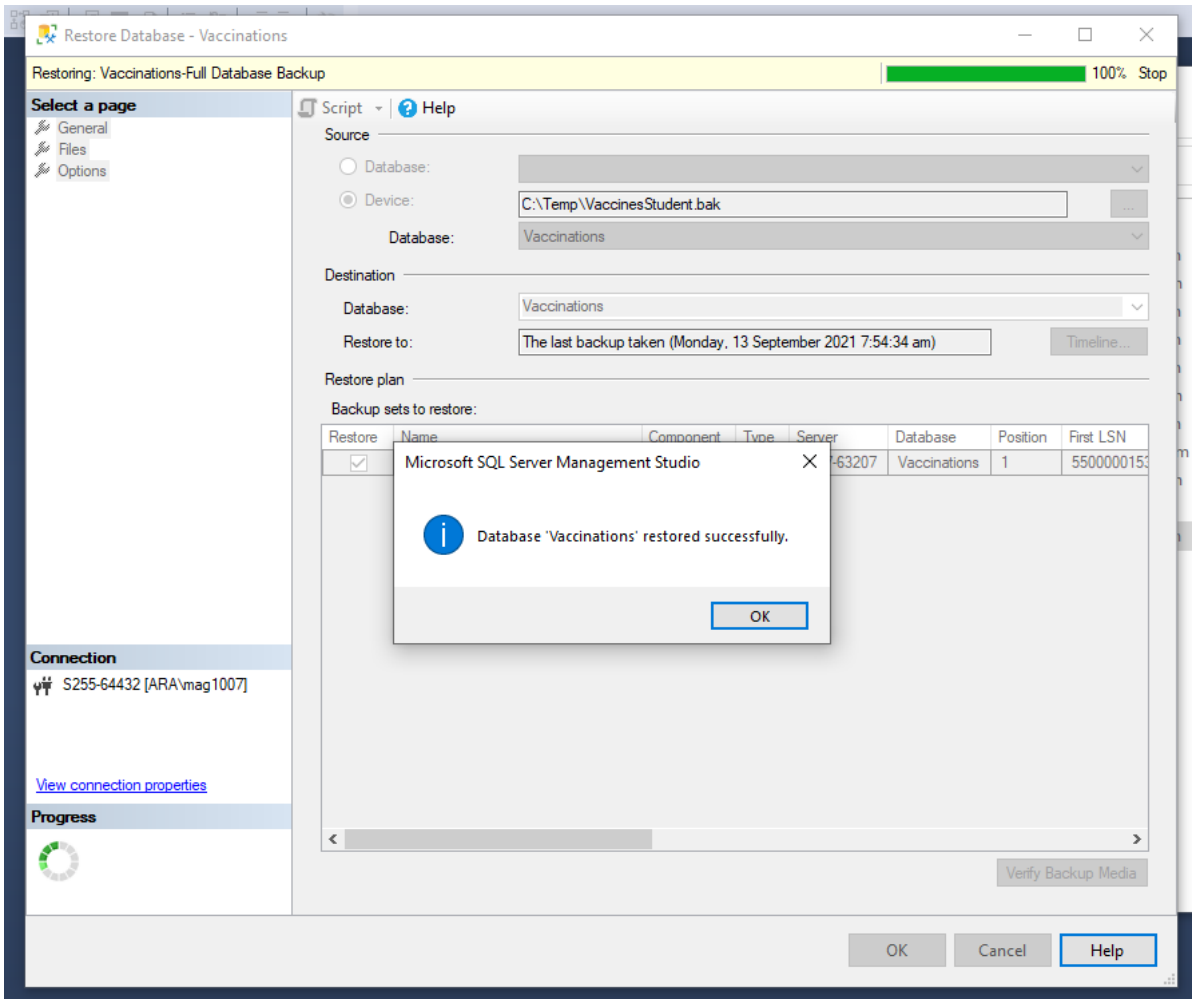
This script enables hourly transaction log backups.

```sql
-- Perform a transaction log backup of the database
USE Vaccinations;
GO

-- Construct the backup file name with date and time
DECLARE @BackupFileName NVARCHAR(255) = N'C:\Temp\LogBackup_' + REPLACE(CONVERT(NVARCHAR, GETDATE(), 120), ':', '') + '.trn';

-- Perform the transaction log backup
BACKUP LOG Vaccinations
TO DISK = @BackupFileName;
```

```
Messages
Processed 560 pages for database 'Vaccinations', file 'Vaccinations_log' on file 1.
BACKUP LOG successfully processed 560 pages in 0.021 seconds (208.333 MB/sec).

Completion time: 2023-10-24T07:23:55.5705854+13:00
```

Restore database backup

To restore the database from a full backup.



Security Policy and Implementation

Task

To maintain the integrity, confidentiality, and availability of the database system, a well-defined user and role-based security policy has been developed. The public does not access the database directly; instead, access is mediated exclusively through stored procedures. This approach minimizes the risk of SQL injection and unauthorized data manipulation.

Users and Roles

Database User Configuration

- A dedicated database user should be created for application-level access.
- This user must be restricted to the EXECUTE permission on a controlled set of stored procedures.
- Direct access to tables is not granted to application users. Instead, data interaction occurs via vetted stored procedures or views.

Application Role

- An application role may be defined and assigned to the application user to streamline permission management.
- Required permissions should be granted to the role rather than directly to individual users to enhance security and auditability.

Rights and Permissions

Execute Permissions

- Only allow EXECUTE permission on specific stored procedures that have been reviewed and sanitized to prevent SQL injection.
- Ensure the scope of permissions is limited to what is strictly necessary for the role's functionality.

No Direct Table Access

- Under no circumstances should users (including application users) have SELECT, INSERT, UPDATE, or DELETE privileges on raw tables.
- Use views or stored procedures to encapsulate data access logic and enforce business rules.

User Access and Authentication Controls

1. Unique Credentials: Each user must have a unique login and password.
2. Strong Password Policies:
    a. Enforce complexity (minimum length, character variety).
    b. Encourage regular password changes.
3. Account Lockout Mechanism:
    a. Lock accounts after repeated failed login attempts to mitigate brute-force attacks.
4. Regular Access Reviews:
    a. Periodically audit user access to ensure alignment with current roles and responsibilities.

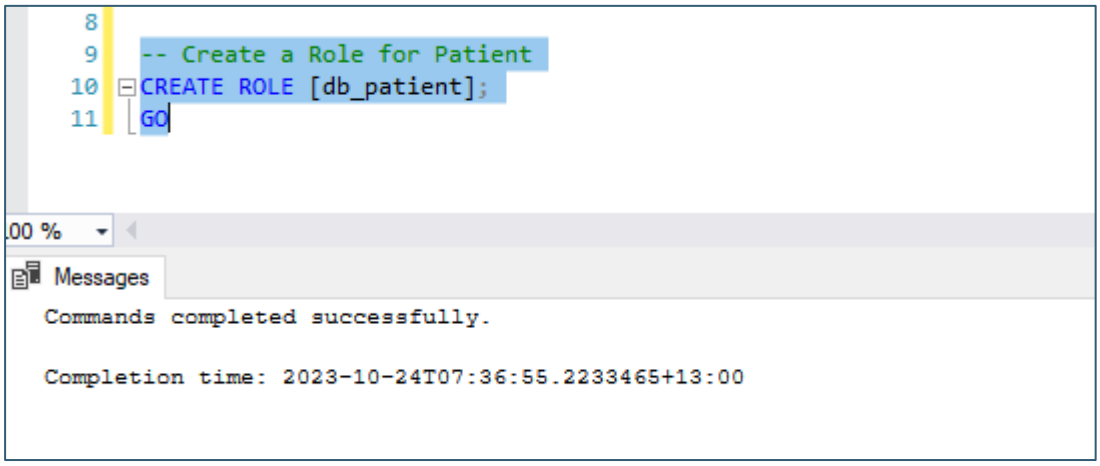
Defined Role Specifications

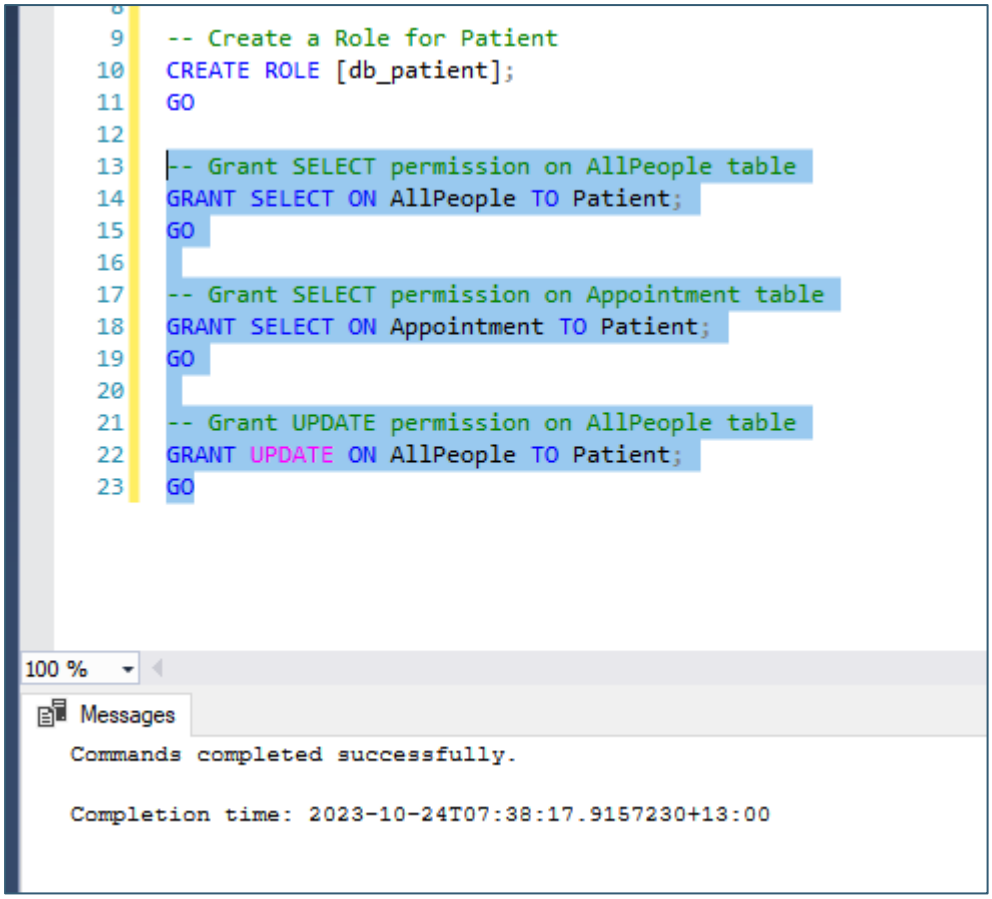
*Each role is configured to enforce **least privilege** principles and ensure compliance with data privacy regulations, including the **New Zealand Privacy Act 2020** and **General Data Protection Regulation (GDPR)**.*

User Role	Assess and functionality
Patient	• View and update personal details; • Access own appointments and vaccination history.
Admin	• Full administrative access to all database objects. • Manage users and their roles.
Vaccinator	• Schedule/manage vaccination appointments. • Update vaccine administration details.
Receptionist	• Manage appointments and view/update patient contact details.
Healthcare Provider	• View and update patient health records and vaccination history.
Reporting Analyst	• Generate and view analytical reports on vaccinations, appointments, and patient data.
Billing/Customer Service	• View billing and insurance data; issue invoices and receipts.
Supervisor/Manager	• Monitor and oversee vaccinators and front desk operations. • Review system-wide performance and scheduling.

Security Script Implementation

Scripts to create roles, assign permissions, and enforce access policies would follow best practices for user isolation and access control. These should be documented thoroughly, validated for injection risks, and maintained with source control.




-- Create a Role for Patient
CREATE ROLE [db_patient];
GO
-- Grant SELECT permission on AllPeople table
GRANT SELECT ON AllPeople TO Patient;
GO
-- Grant SELECT permission on Appointment table
GRANT SELECT ON Appointment TO Patient;
GO
-- Grant UPDATE permission on AllPeople table
GRANT UPDATE ON AllPeople TO Patient;
GO
100 %
Messages
Commands completed successfully.
Completion time: 2023-10-24T07:38:17.9157230+13:00

SQLQuery1.sql - X...(ARA\mag1007 (61))*

```
-- Grant UPDATE permission on AllPeople table
GRANT UPDATE ON AllPeople TO Patient;
GO

-- Create a Role for Admin
CREATE ROLE [db_admin1];
GO

-- Grant CONTROL permission on the database
GRANT CONTROL ON DATABASE::Vaccinations TO Admin;
GO

-- Grant SELECT, INSERT, UPDATE, and DELETE permissions on all tables
GRANT SELECT, INSERT, UPDATE, DELETE ON AllPeople TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON Appointment TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON OldAppointment TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON Place TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON Vaccinator TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON Person TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON vw_AgeAtFirstVaccination TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON vw_DailyVaccination TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON vw_DoneByVaccinators TO Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON vw_MissingBooking TO Admin;
GO
```

100 %

Messages

```
Commands completed successfully.

Completion time: 2023-10-24T07:39:49.3455723+13:00
```

```
-- Create a Role for Billing/Customer Service
CREATE ROLE [db_billingCustomerService];
GO

-- Grant SELECT permission on AllPeople table
GRANT SELECT ON AllPeople TO BillingCustomerService;
GO

-- Grant SELECT permission on Appointment table
GRANT SELECT ON Appointment TO BillingCustomerService;
GO
```

00 %

Messages

```
Commands completed successfully.

Completion time: 2023-10-24T07:43:17.9025414+13:00
```

```
-- Create a Role for Vaccinator
CREATE ROLE [db_vaccinator];
GO
-- Grant SELECT, INSERT, and UPDATE permissions on Appointment table
GRANT SELECT, INSERT, UPDATE ON Appointment TO Vaccinator;
GO

-- Create a Role for Front Desk/Receptionist
CREATE ROLE [db_frontDeskReceptionist];
GO

-- Grant SELECT, INSERT, and UPDATE permissions on Appointment table
GRANT SELECT, INSERT, UPDATE ON Appointment TO FrontDeskReceptionist;
GO

-- Grant SELECT and UPDATE permissions on AllPeople table
GRANT SELECT, UPDATE ON AllPeople TO FrontDeskReceptionist;
GO

-- Create a Role for Healthcare Provider
CREATE ROLE [db_healthcareProvider];
GO

-- Grant SELECT and UPDATE permissions on AllPeople table
GRANT SELECT, UPDATE ON AllPeople TO HealthcareProvider;
GO

-- Grant SELECT and UPDATE permissions on Appointment table
GRANT SELECT, UPDATE ON Appointment TO HealthcareProvider;
GO

-- Create a Role for Reporting Analyst
CREATE ROLE [db_reportingAnalyst];
GO

-- Grant SELECT permission on relevant tables and views
GRANT SELECT ON vw_DailyVaccination TO ReportingAnalyst;
GO
```

```
Messages
Commands completed successfully.

Completion time: 2023-10-24T07:41:58.1745696+13:00
```

```
ALTER ROLE Patient ADD MEMBER guest;
GO
```

```
Messages
Commands completed successfully.

Completion time: 2023-10-24T07:50:48.1534439+13:00
```

Indexing and Performance Optimization

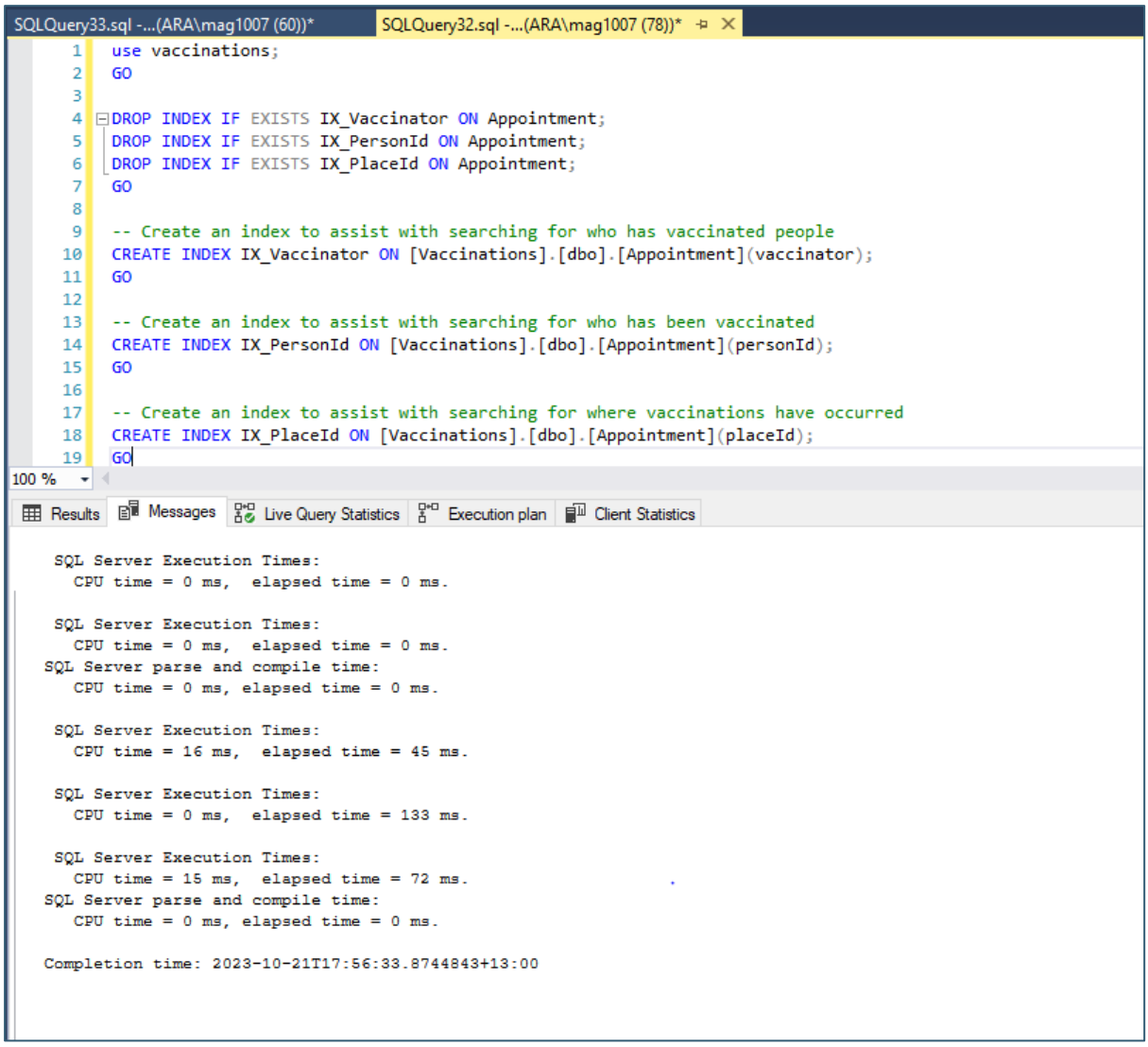
Task

The current database design is optimized primarily for data entry. No views or indexes have yet been implemented to facilitate efficient querying, particularly in relation to vaccinations. This task involves designing and implementing indexes that support commonly executed queries and analysing their impact on overall database performance.

Implementation

Create Indexes

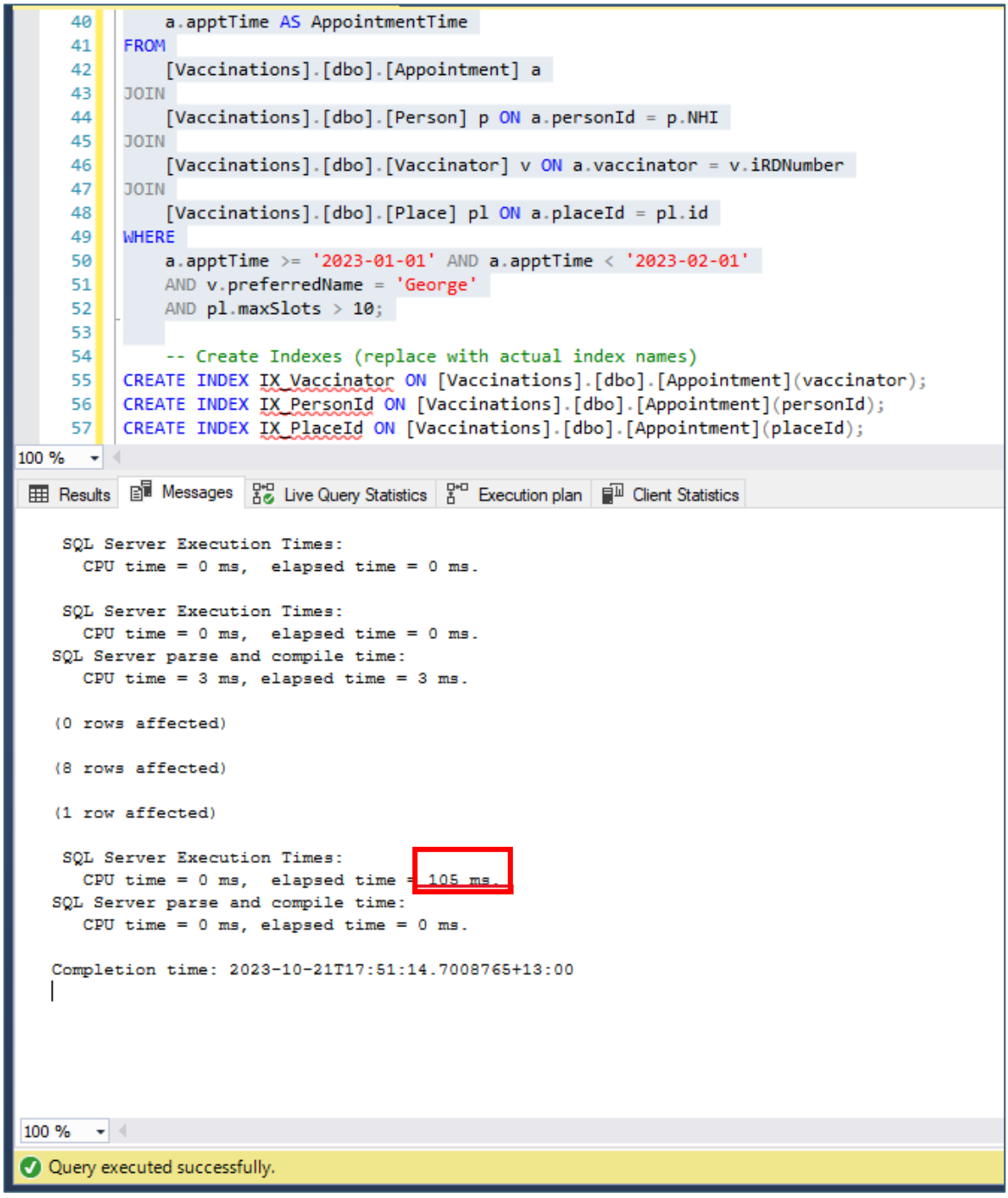
The following indexes were created to improve query performance on frequently searched fields within the Appointment table in the Vaccinations database



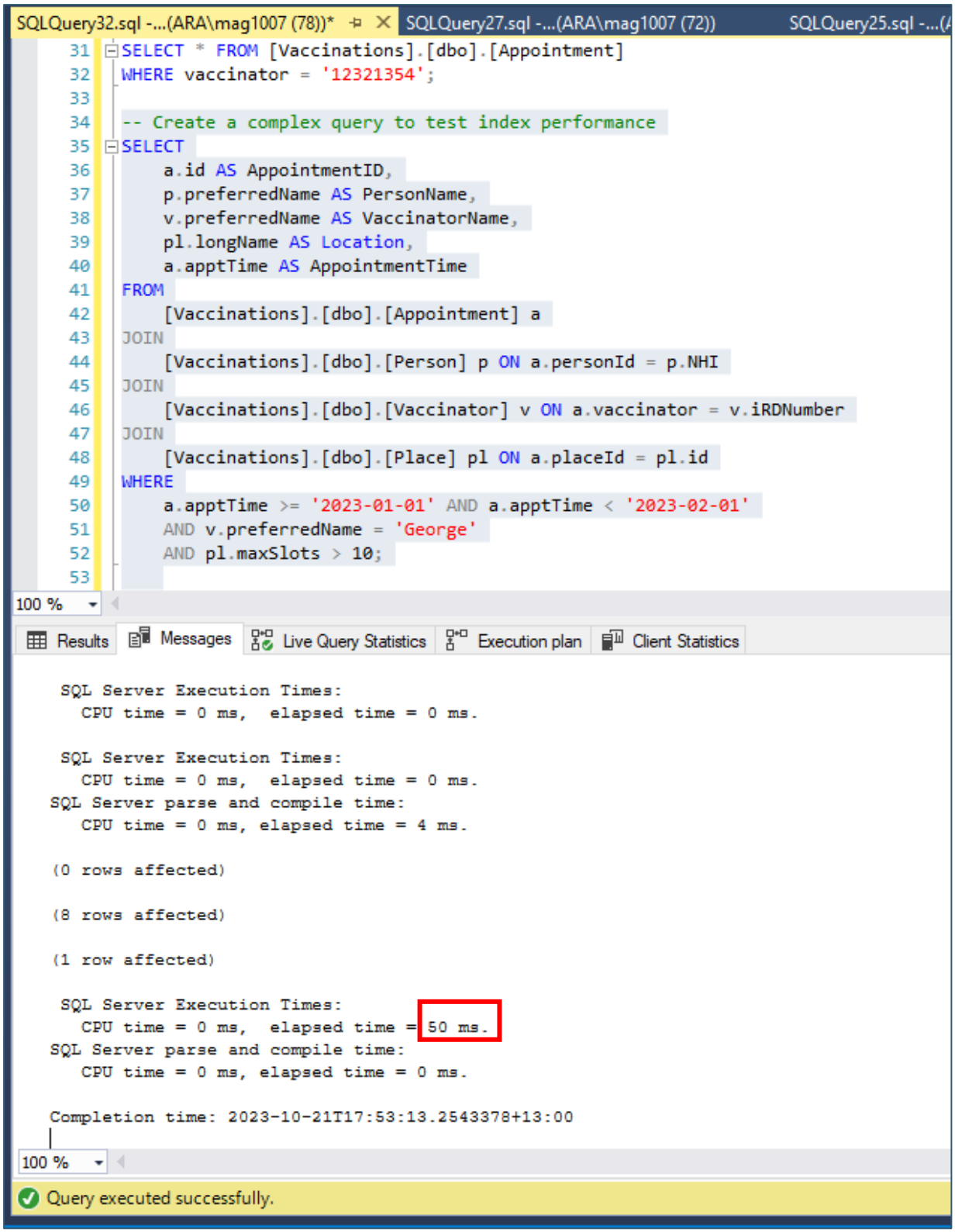
Testing

To test indexes, you should begin by creating a complex query and executing it to observe variations in server execution time.

Before indexing



After indexing



Impact on Database Performance and Recommendations

Indexing plays a critical role in improving query performance, particularly in read-heavy databases. As demonstrated in the screenshots, the inclusion of indexes significantly reduced query execution time—from over 100ms in some cases down to around 50ms or less.

Benefits Observed

- *Faster Query Performance*
  Execution times improved notably after indexing key columns (vaccinator, personId, placeId) in the Appointment table.
- *Improved Join Efficiency*
  Complex joins involving Person, Vaccinator, and Place were executed faster due to indexed join keys.
- *Better Filter Operations*
  ndexed columns used in WHERE clauses (e.g., apptTime, preferredName) contributed to more efficient filtering.

Potential Drawbacks

- *Insert/Update Overhead*
  Every time a row is inserted or updated, the associated indexes must also be maintained, which may slow down write operations.
- *Storage Usage*
  Indexes consume additional disk space, which can become significant with large datasets.
- *Maintenance Costs*
  Indexes may require periodic reorganization or rebuilding to maintain optimal performance.
- *Query Plan Complexity*
  Too many indexes can confuse the query optimizer, leading to suboptimal execution plans.

Alternative or Complementary Approaches to Indexing

1. **Covering Indexes**
   Create indexes that include all columns required by frequently executed queries. This reduces the need for lookups in the base table, further improving performance.
2. **Query Optimization**
   Refactor SQL queries for efficiency—avoid unnecessary joins, filter early, and ensure the query logic is streamlined.
3. **Caching Strategies**
   For high-traffic, read-heavy applications, consider implementing an in-memory caching layer (e.g., Redis or Memcached) to store frequently accessed data.
4. **Partitioning and Archiving**
   If the database grows significantly, consider partitioning tables or archiving older data to reduce the volume of active data scanned by queries.
5. **Regular Monitoring**
   Use tools like SQL Server Management Studio (SSMS) Execution Plans and Statistics to regularly monitor performance and adjust indexes accordingly.

Views

View 1: Daily Vaccination in each place.

Task Description

This view is designed to monitor and summarize daily vaccination activity across all vaccination locations. It helps track both:

- The maximum number of vaccinations that could be administered per location per day.
- The actual number of vaccinations that were booked or administered on that specific date.

This information is useful for capacity planning, performance monitoring, and operational reporting in a vaccination program.

Implementation

```
/* create view to  showseach vaccination place, with the total number of vaccinations
that could happen each day, and the number of vaccinations actually happening or booked each day. */

CREATE VIEW vw_DailyVaccination
AS
SELECT
    ap.placeId AS VaccinationPlace,
    CONVERT(DATE, ap.apptTime) AS testDates,
    COUNT(*) AS NumberOfPossibleVaccines,
    COUNT(ap.vaccineNumber) AS ActualNumberOfVaccines
FROM
    Appointment ap
GROUP BY
    ap.placeId, CONVERT(DATE, ap.apptTime)

GO

-- test view

SELECT *
FROM vw_DailyVaccination
ORDER BY VaccinationPlace, testDates;
```

100 %

Results | Messages

	VaccinationPlace	testDates	NumberOfPossibleVaccines	ActualNumberOfVaccines
67	Tim_Red	2021-08-29	27	26
68	Tim_Red	2021-08-30	54	47
69	Tim_Red	2021-08-31	54	51
70	Tim_Wool	2021-08-02	87	77
71	Tim_Wool	2021-08-03	87	83
72	Tim_Wool	2021-08-04	87	80
73	Tim_Wool	2021-08-05	87	81
74	Tim_Wool	2021-08-06	58	53
75	Tim_Wool	2021-08-09	87	78
76	Tim_Wool	2021-08-10	87	82
77	Tim_Wool	2021-08-11	87	78
78	Tim_Wool	2021-08-12	87	81
79	Tim_Wool	2021-08-13	58	50
80	Tim_Wool	2021-08-16	87	81
81	Tim_Wool	2021-08-17	87	77
82	Tim_Wool	2021-08-18	87	78

View 2: Missing Bookings

Task Description

This view identifies individuals who have **missed their vaccination appointments** (i.e. no vialNumber recorded) and **do not have a rescheduled or later booking** in the system. The view includes their contact details to assist healthcare staff in **follow-up communication** via their preferred method (text, phone, or email).

This is crucial for maintaining vaccination coverage and ensuring no individuals are left behind due to missed appointments.

Implementation

```
16
17  /* A view showing all people that have missed a booking, and do not have a later booking.
18   We will want to contact these people, so include names and contact details. */
19
20  CREATE VIEW vw_MissingBooking
21   AS
22   SELECT
23       NHI AS personId,
24       preferredName, eMail, phone, preferredContactMethod
25   FROM
26       Person p
27   INNER JOIN Appointment ap ON p.NHI = ap.personId
28   WHERE ap.vialNumber is NULL;
29
30   GO
31
32  select * from vw_MissingBooking;
33
34   GO
```

100 %

Results | Messages

	personId	preferredName	eMail	phone	preferredContactMethod
1	ABC0156	Shanell	ABC0156@havemydata.nz	8670164	text
2	AEG0500	Brock	AEG0500@havemydata.nz	7710508	text
3	AER0512	Carter	AER0512@havemydata.nz	7590520	phone
4	AFI0576	Mae	AFI0576@havemydata.nz	7640584	phone
5	AGH0632	Marietta	AGH0632@havemydata.nz	6770640	phone
6	AHA0736	Jesica	AHA0736@havemydata.nz	6560744	phone
7	AIT0848	Gena	AIT0848@havemydata.nz	8280856	text
8	AJM1020	Maxwell	AJM1020@havemydata.nz	6911028	text
9	AKE1100	Larraine	AKE1100@havemydata.nz	7311108	phone
10	AKM1060	Santa	AKM1060@havemydata.nz	7081068	phone
11	AKM1104	Judi	AKM1104@havemydata.nz	6511112	phone
12	ALB1172	Jan	ALB1172@havemydata.nz	7821180	text
13	ALR1156	Emogene	ALR1156@havemydata.nz	8171164	phone
14	AMH1276	Marietta	AMH1276@havemydata.nz	8421284	text
15	AMO1296	Elma	AMO1296@havemydata.nz	8921304	text
16	ANB1448	Daisey	ANB1448@havemydata.nz	6811456	phone
17	ANQ1420	Angila	ANQ1420@havemydata.nz	7811428	text
18	AOU1512	Carter	AOU1512@havemydata.nz	6741520	text
19	API1648	Marietta	API1648@havemydata.nz	7551656	phone
20	AWD2324	Venetta	AWD2324@havemydata.nz	8512332	text

View 3: Patient Age at First Vaccine

Task Description

This view categorizes patients by their age (in decades) at the time of their first vaccination. It is used to analyse age distribution and trends in vaccination uptake. Only patients with a valid vaccine record are included, ensuring the analysis is based on actual vaccination data. This breakdown helps with targeted public health strategies, such as prioritizing specific age groups for follow-up or campaigns.

Implementation

```
25  CREATE VIEW vw_AgeAtFirstVaccination
26  AS
27  SELECT
28      FLOOR(DATEDIFF(YEAR, dateOfBirth, apptTime) / 10.0) * 10 AS AgeDecadeCategory,
29      NHI AS patientNHI,
30      PreferredName AS patientName,
31      VialNumber AS firstVaccineVial,
32      ApptTime as firstVaccineTime,
33      DATEDIFF(YEAR, dateOfBirth, apptTime) AS AgeAtFirstVaccine
34  FROM
35      Person p
36  INNER JOIN Appointment ap ON p.NHI = ap.PersonId
37  WHERE
38      NHI IN (SELECT DISTINCT PersonId FROM Appointment) and VialNumber is not NULL;
39
40  GO
41
42  -- test view
43  select * from vw_AgeAtFirstVaccination
44  ORDER BY AgeDecadeCategory;
45
46  GO
47
48
```

100 %

Results | Messages

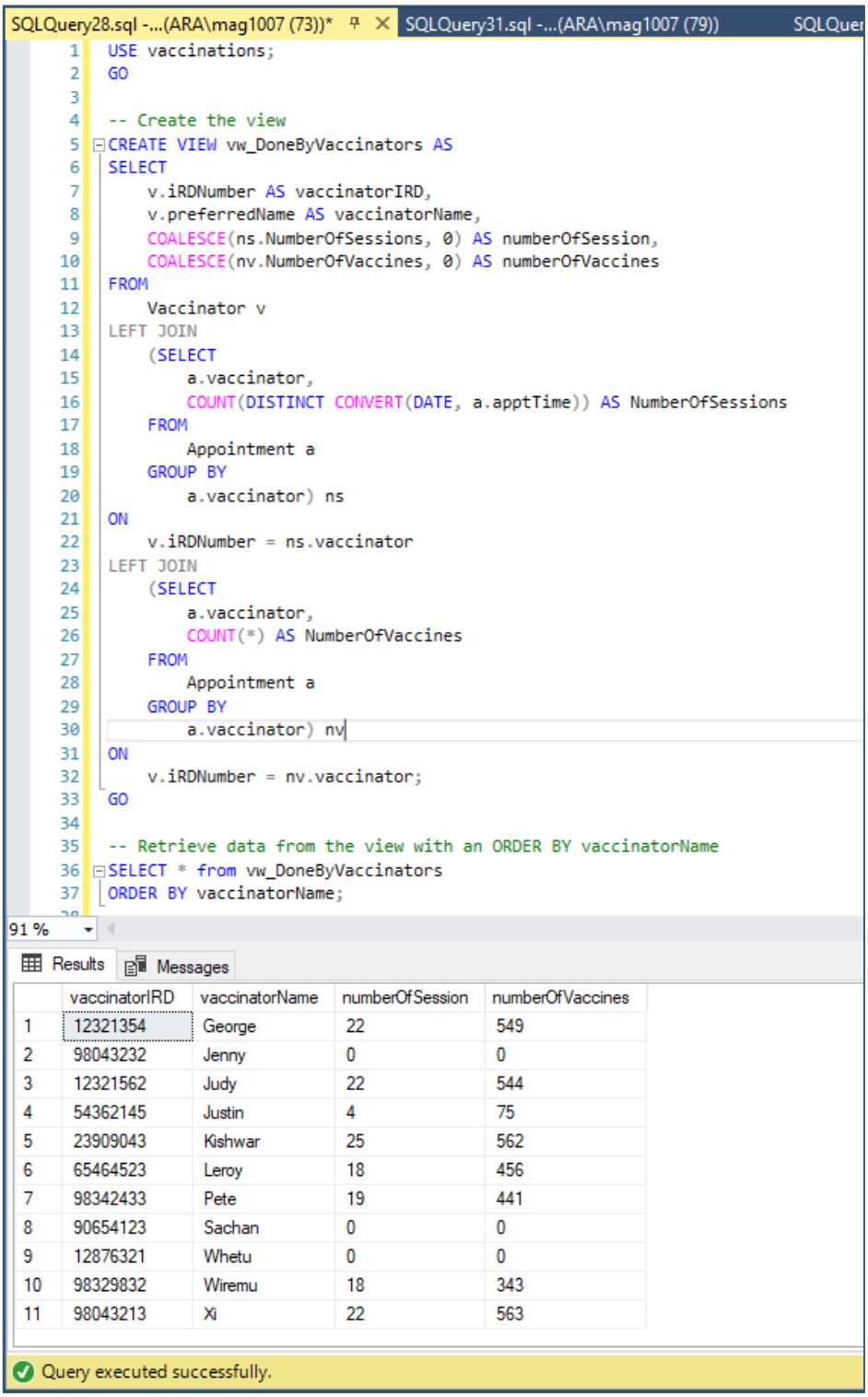
	AgeDecadeCategory	patientNHI	patientName	firstVaccineVial	firstVaccineTime	AgeAtFirstVaccine
1	30	ACD0304	Carol	SC0002028	2021-08-11 10:00:00	36
2	30	ADY0392	Todd	SC0004257	2021-08-12 15:40:00	36
3	30	AFK0600	Emogene	SC0003754	2021-08-13 12:15:00	36
4	30	AFZ0584	Era	SC0003963	2021-08-24 14:45:00	34
5	30	AHQ0724	Elma	SC0003664	2021-08-10 14:30:00	34
6	30	AHH0776	Josephina	SC0005136	2021-08-17 10:00:00	37
7	30	AJD1024	Shanell	SC0003225	2021-08-31 15:30:00	39
8	30	AJS0936	Kenya	SC0004885	2021-08-12 07:45:00	37
9	30	ALM1164	Maxwell	SC0002617	2021-08-20 11:45:00	37
10	30	AOE1520	Era	SC0004893	2021-08-12 08:45:00	38
11	30	AOI1504	Carol	SC0004385	2021-08-23 12:40:00	34
12	30	APW1624	Kenya	SC0001573	2021-08-03 13:15:00	39
13	30	ASF1948	Angila	SC0002904	2021-08-25 15:30:00	35
14	30	ASJ1872	Marietta	SC0002109	2021-08-12 09:30:00	38
15	30	ASN1912	Kenna	SC0004924	2021-08-12 12:30:00	34
16	30	AUY2124	Yelena	SC0002025	2021-08-11 09:45:00	36
17	30	AVM2208	Jesica	SC0004890	2021-08-12 08:15:00	36
18	30	AVM2228	Salome	SC0004507	2021-08-04 07:45:00	35
19	30	AWH2352	Jesica	SC0004016	2021-08-26 11:30:00	37
20	30	AXM2404	Bea	SC0004930	2021-08-12 13:15:00	35
21	30	AYD2496	Angila	SC0005271	2021-08-20 08:45:00	36

View 4 Number of sessions & vaccines by all vaccinators.

Task Description

This view displays the total number of unique vaccination sessions and the total number of vaccines administered by each vaccinator. It also includes vaccinators who have not yet administered any vaccines or conducted any sessions, ensuring complete visibility of all registered vaccinators.

Implementation



Trigger

Task description

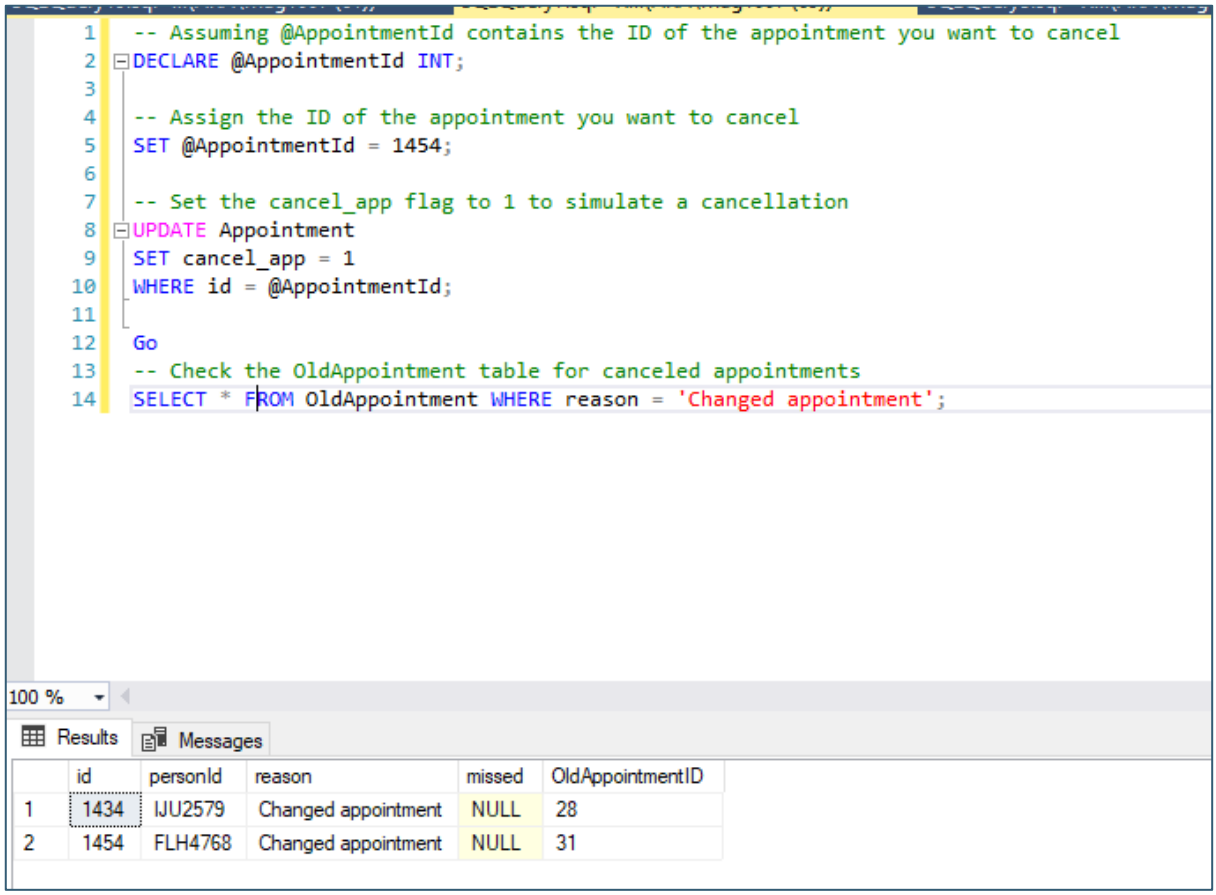
Develop a trigger that automatically populates the OldAppointment table when an appointment is cancelled. The inserted record should include the cancellation reason marked as “Changed appointment”.

Implementation

The following trigger is created to run after an update on the Appointment table. It checks if the cancel_app column was updated and set to 1, indicating a cancellation. If so, the appointment details are inserted into the OldAppointment table.

```
use Vaccinations;
GO
-- Create a trigger to populate OldAppointment when an appointment is canceled
CREATE OR ALTER TRIGGER Trigger_CancelAppointment
ON dbo.Appointment
AFTER UPDATE
AS
BEGIN
    IF UPDATE(cancel_app) -- Check if the cancel_app column is updated
    BEGIN
        DECLARE @AppointmentID INT;

        -- Get the ID of the updated appointment
        SELECT @AppointmentID = id FROM inserted;

        -- Check if the appointment is canceled (cancel_app = 1)
        IF (SELECT cancel_app FROM dbo.Appointment WHERE id = @AppointmentID) = 1
        BEGIN
            -- Insert the canceled appointment into OldAppointment with the reason "Changed appointment"
            INSERT INTO OldAppointment (id, personId, reason, missed)
            SELECT id, personId, 'Changed appointment', NULL
            FROM dbo.Appointment
            WHERE id = @AppointmentID;
        END
    END
END;
```

```
Messages
Commands completed successfully.

Completion time: 2023-10-24T07:07:45.6567079+13:00
```



```sql
-- Assuming @AppointmentId contains the ID of the appointment you want to cancel
DECLARE @AppointmentId INT;

-- Assign the ID of the appointment you want to cancel
SET @AppointmentId = 1454;

-- Set the cancel_app flag to 1 to simulate a cancellation
UPDATE Appointment
SET cancel_app = 1
WHERE id = @AppointmentId;

Go
-- Check the OldAppointment table for canceled appointments
SELECT * FROM OldAppointment WHERE reason = 'Changed appointment';
```

100 %

Results | Messages

	id	personId	reason	missed	OldAppointmentID
1	1434	IJU2579	Changed appointment	NULL	28
2	1454	FLH4768	Changed appointment	NULL	31