# Stored Procedures

1. **Create Vaccinator**

**Graphical user interface, text, application, email

Description automatically generated**

**Testing**

**Graphical user interface, text, application

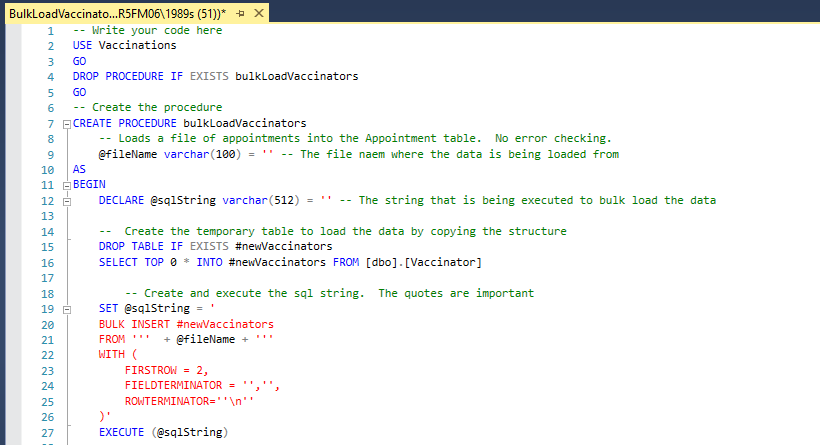
Description automatically generated**

**Graphical user interface, text, application

Description automatically generated**



1. **bulkLoad Vaccinators**

****

**Text

Description automatically generated**

**Testing bulk load, everything is deleted**

**Graphical user interface, application

Description automatically generated**

**Testing bulk load**

**Graphical user interface, text, application

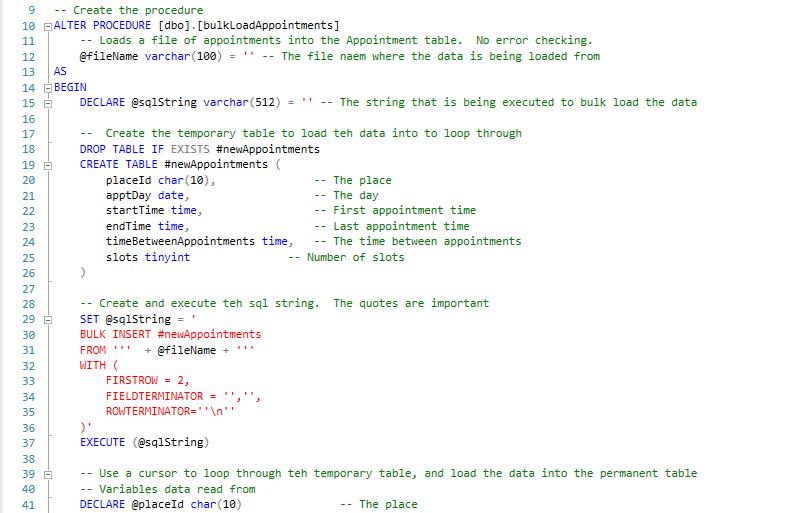
Description automatically generated**

**create Appointment**

**Graphical user interface, text, application, email

Description automatically generated**

**bulkLoad Appointments**

****Graphical user interface, text, application, email

Description automatically generated

**Testing bulk load Appointments ( I’m loading site session August, which is basically original data)**

Graphical user interface, table

Description automatically generated

Prove of bulk load total appointments

Graphical user interface, text, application

Description automatically generated

# Backup

Performing regular database backups is a crucial part of any disaster recovery plan. There are many aspects that should be taken into consideration when planning long term backup strategies. It is necessary to choose the backup type and backup location and to plan how frequently to perform the backups. Additionally, the processes of verification or encryption could also be included in the plan.

For systems that contain multiple servers, automation of the process is a must. It is difficult to keep track of all the parameters, details and settings, and that is where the SQL backup policies come in handy. All of the settings could be precisely defined and applied using a single backup policy. Furthermore, the policy can be reused according to your needs or could be set up to run automatically.

Backup can be done “**manually**” through running a simple script

Graphical user interface, text, application, email

Description automatically generated

**Note that it is necessary to create at least one full database backup, in order to be able to perform differential, or transaction log backups.**

Prove

Graphical user interface, application

Description automatically generated

Policy:

**Full backup** should be done automatically or manually every week, because database is very small it takes only a second.

Time: Friday in the end of day (might be Thursday if there is a public holiday on Friday)

**Differential and Log backups** should be done on daily basis. As this backup is only for changes since the last full backup.

**Replication** allows to disseminate data to a number of servers and then distribute the query load among those servers.  
Offline processing. Replication supports manipulating data from database on a machine that is not always connected to the network.  
Redundancy. Replication allows you to build a fail-over database server that’s ready to pick up the processing load at a moment’s notice.

Asynchronous replication in real time to lower the risk of data loss.

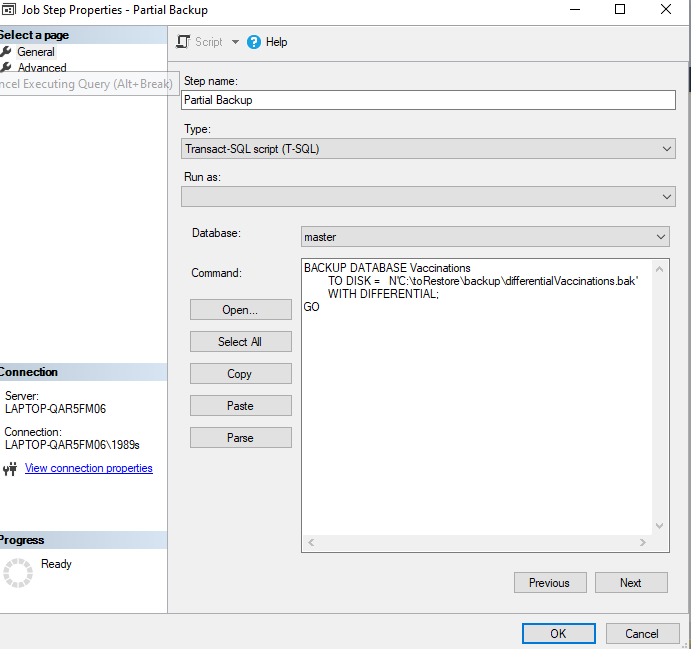
## **Creating and managing SQL backup policies with SQL Server Agent**

**Server agent needs to be started**

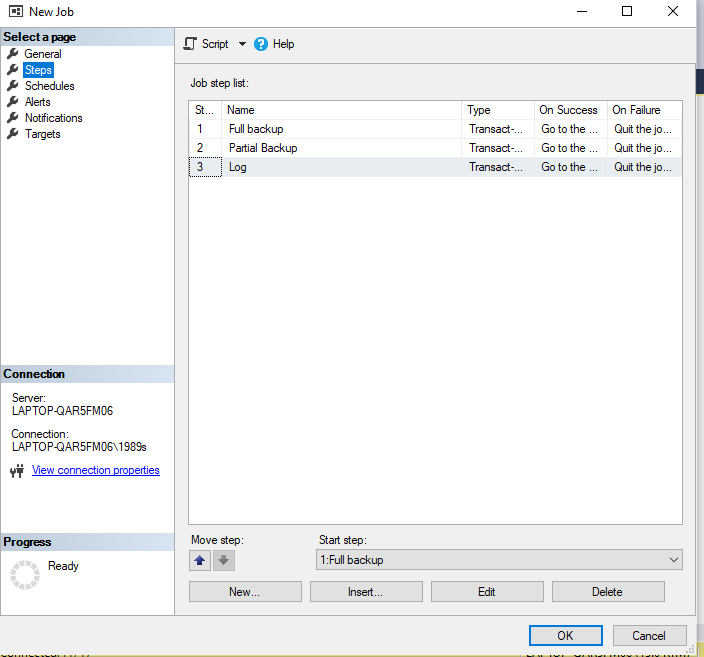
**Text, application

Description automatically generated**

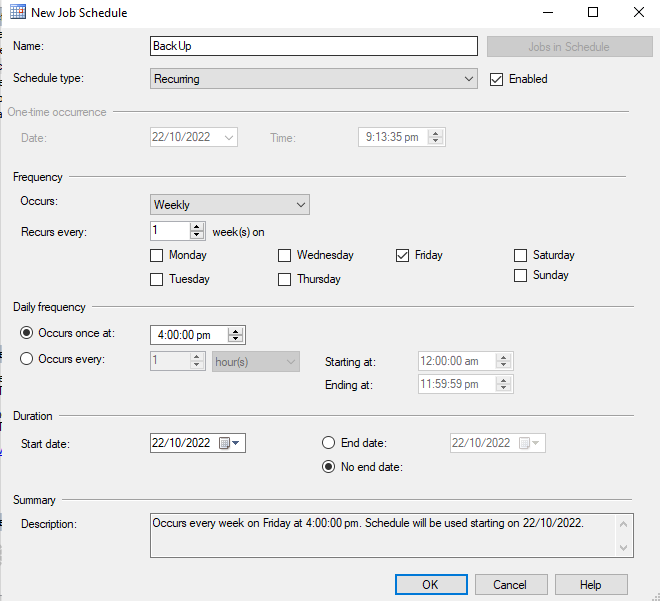
**Example of a Partial backup**



**Steps were created for a Job to run them accordingly.**



**Example of a scheduled backup**



**Scheduled steps of one JOB**

**Graphical user interface, text, application

Description automatically generated**

**Job exists in DB**

Graphical user interface, application

Description automatically generated

**Prove of working JOB**

**Graphical user interface, text, application

Description automatically generated**

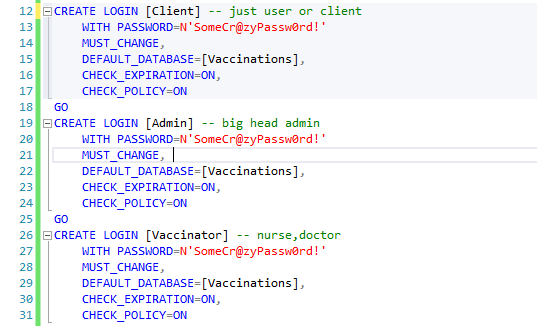
**Prove of Backup**

Graphical user interface, application

Description automatically generated

# User, security, roles

I have created 3 users



Prove of Users

Graphical user interface, text

Description automatically generated



**Three roles have been created that match the usernames.**

Graphical user interface, text, application

Description automatically generated

**Prove of Roles**

A picture containing table

Description automatically generated



**Admin server roles (Applicable only for an admin)**

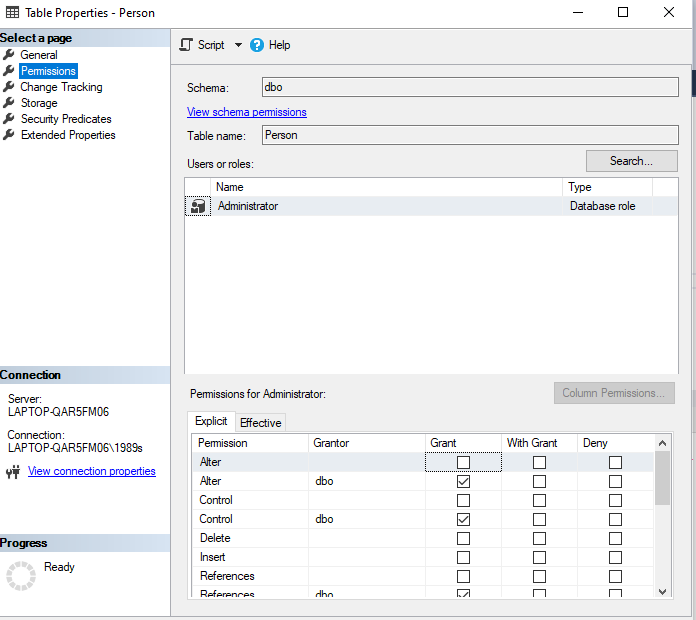
Graphical user interface, text, application, email

Description automatically generated

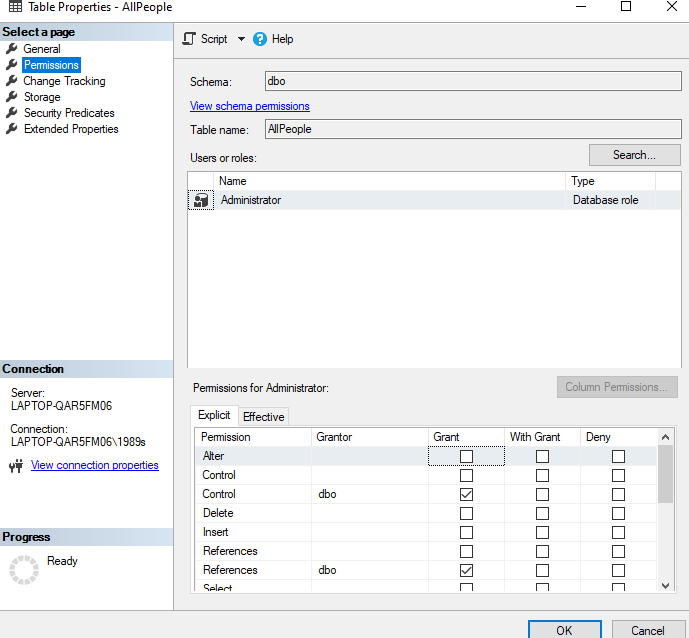
**Admin permissions for a table Person**

Rights or permissions to use the database. My assumptions.

The **administrator** has the right to VIEW the entire database, all tables, he is at the heart of the entire database, but he is not supposed to INSERT, DELETE, UPDATE data from tables. This is due to the fact that this is not part of his responsibilities, and he should not have access to edit data.

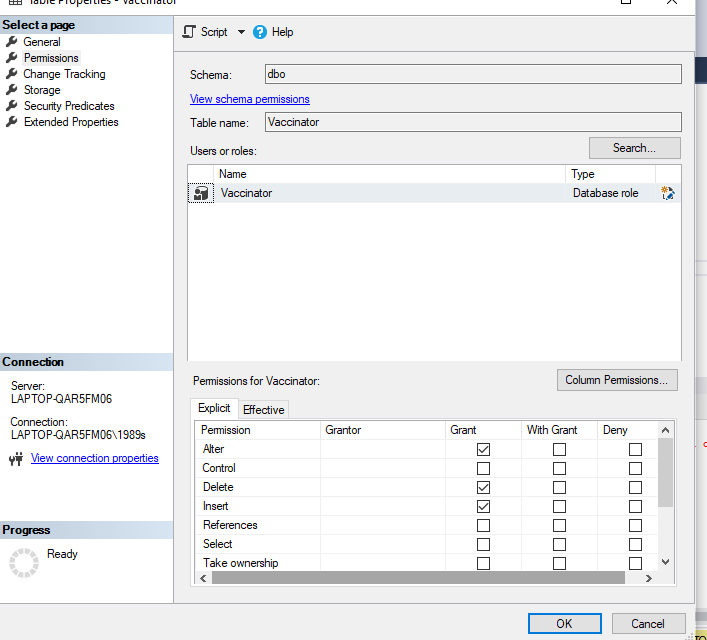
****

**Admin permissions for a table Person**

****

**Vaccinator permissions for table Vaccinator**

As for the staff, or in my case the **vaccinator**, he can view everything, delete or change the table data. But he is forbidden to change the database itself, there is no access to backup and other important data.



Permissions for AllPeople table

**Graphical user interface

Description automatically generated**

**Client role/ permissions**

The average user or **client** will not have any access to the database at all, only to a view specially created by me for this situation, client can see only place, slot and name of the Vaccinator. He can only see view and see updates to this view.

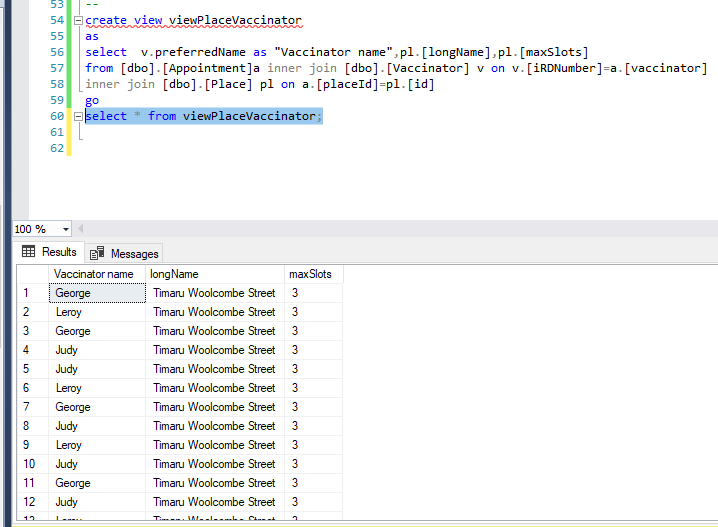
Screens of rights or permissions. Please be aware that I included limited tables due to time limit.

**Special view is created**

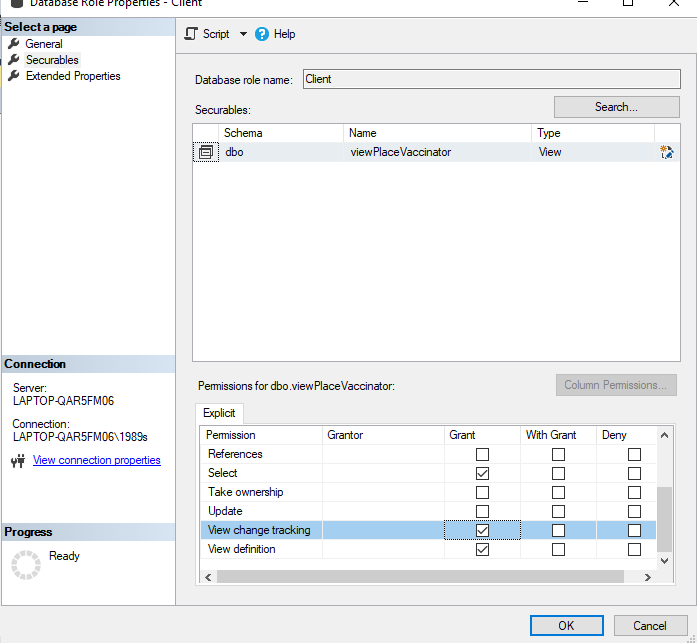
**Text

Description automatically generated**

**Prove of view**

****

Permissions for Client



# Indexing

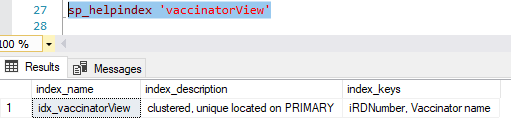
* **Index 1** Assist with searching for who has vaccinated people

Graphical user interface, text, application, email

Description automatically generated

**Prove**

**Index created**



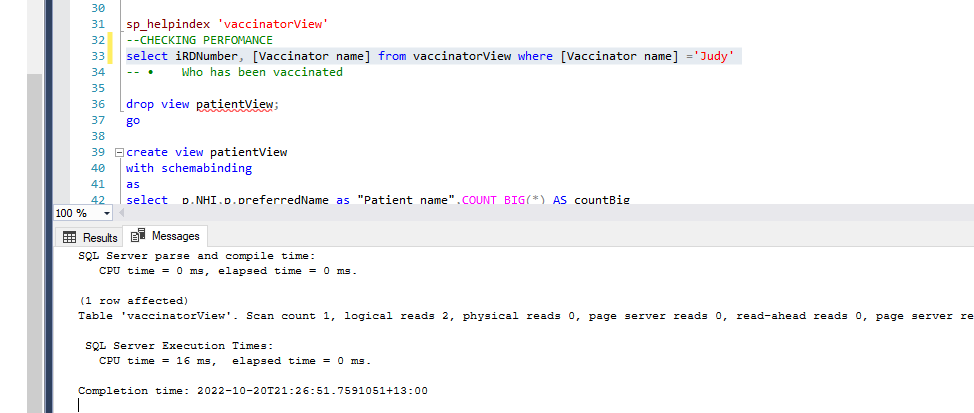
**Before index and view were created**

select iRDNumber, [Vaccinator name] from vaccinatorView where [Vaccinator name] ='Judy'

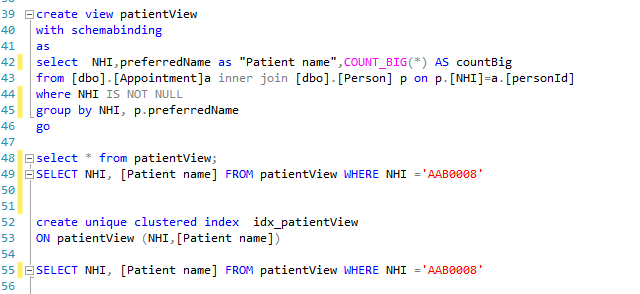
Graphical user interface, text, application

Description automatically generated

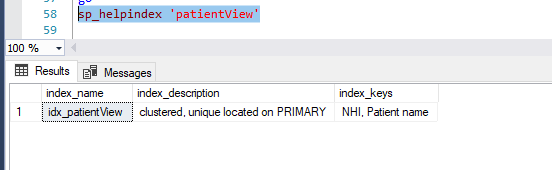
**After**

****

* **Index 2 Who has been vaccinated**



**Prove of index**



**Performance without index and view**

SELECT NHI, [Patient name] FROM patientView WHERE NHI ='AAB0008'

Graphical user interface, text, application, email

Description automatically generated

**Performance after**

Graphical user interface, text, application

Description automatically generated

* **Index 3 Where vaccinations have occurred**

Graphical user interface, text, application, email

Description automatically generated

**Prove of index**

Graphical user interface, text, application

Description automatically generated

**Performance before index and view were created**

**Graphical user interface, text, application, email

Description automatically generated**

**Performance after**

**Graphical user interface, text, application, email

Description automatically generated**

**Comment on what these indexes may do to database performance and propose an alternative solution to indexes.**

Those indexes speed up searches/queries as it can be seen

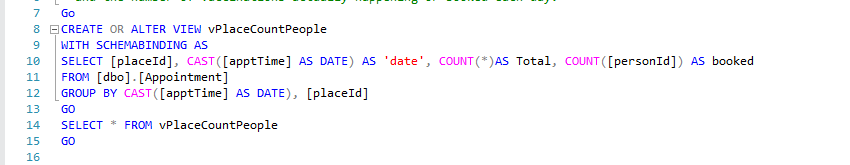
Index is s performance tool that is small, fast, and optimized for quick lookups.

**Alternative solution or how to increase performance**

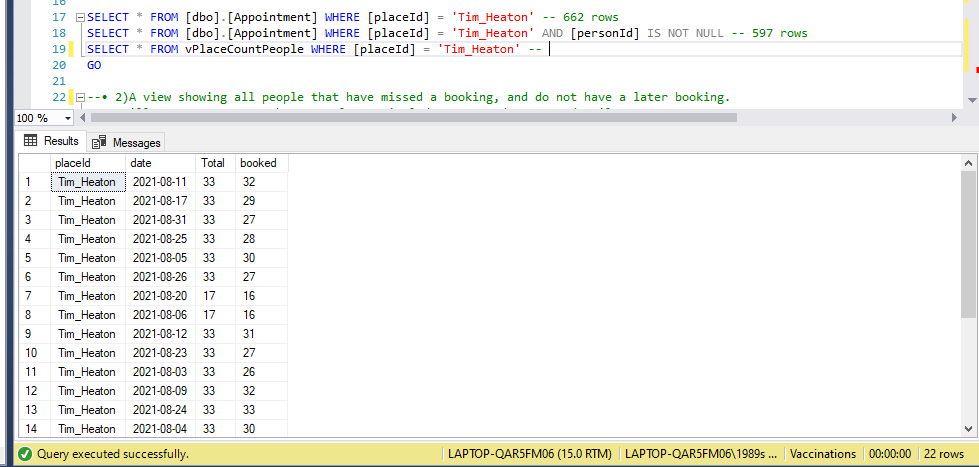
* Need to make sure that queries which rely on Indexes run frequently, otherwise there is no point in creating indexes.
* Another thing to consider when indexing a database table is the size of the table. If the table is small with less than 1000 pages, no performance enhancement can be gained from indexing that table, as SQL Server Query Optimizer will prefer scanning the whole table rather than examining the SQL index and try to create the best possible plan. In other words, this index on the small table will not be used and will have overhead on the table as it should be maintained when the table is changed.
* It is recommended to start indexing the table by creating a clustered index, that covers the column(s) called very frequently, which will convert it from the heap table to a sorted clustered table, then create the required non-clustered indexes that cover the remaining queries in the system. In this way, the non-clustered indexes will be built over the SQL Server clustered index and the pointers on the leaf-level nodes of the non-clustered indexes will also point to the location of the row in the sorted clustered index.
* Selecting the proper storage criteria for the index during the design phase will help in improving the query performance by increasing the I/O performance. For instance, creating the non-clustered index on a filegroup located in a different disk drive than the disk drive where the main table is created will improve the performance of the queries that use that non-clustered index, as it will not be affected by the concurrent reading of the data and SQL index pages, that are spread across different disks, which will be performed on different disk drives.

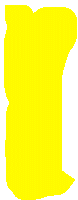
# Views

**1)A view that shows each vaccination place, with the total number of vaccinations that could happen each day, and the number of vaccinations actually happening or booked each day.**



**View 1 prove**

****



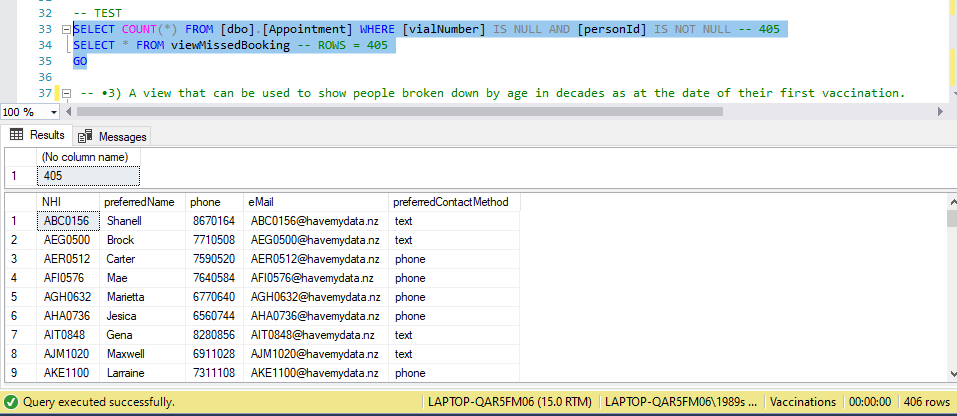
**2) A view showing all people that have missed a booking, and do not have a later booking. We will want to contact these people, so include names and contact details.**

Text

Description automatically generated

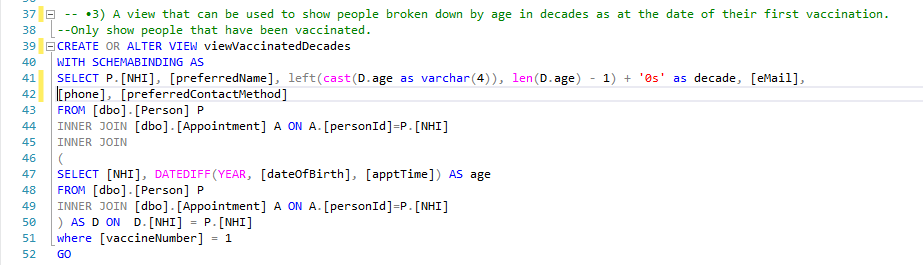
**View prove**

**testing**

****



**3) A view that can be used to show people broken down by age in decades as at the date of their first vaccination. Only show people that have been vaccinated.**

****

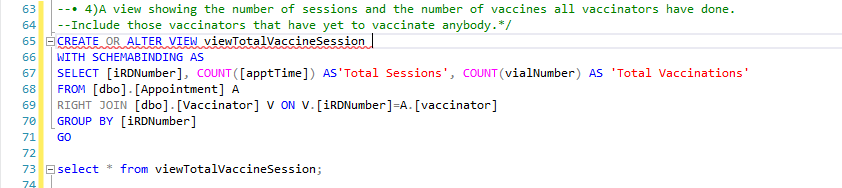
**View prove**

**Graphical user interface, text, application, email

Description automatically generated**



**4) A view showing the number of sessions and the number of vaccines all vaccinators have done. Include those vaccinators that have yet to vaccinate anybody.**

****

**View prove**

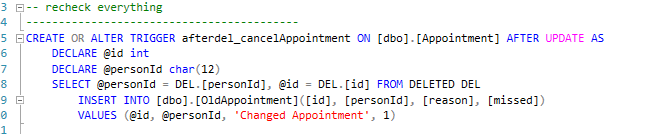
**Table

Description automatically generated**

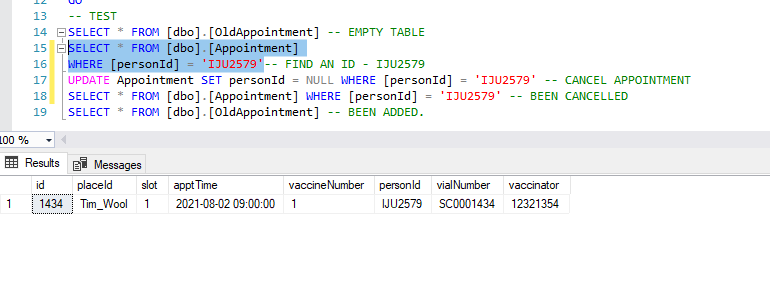


# Trigger

* When someone cancels an appointment, the table for Old Appointments is populated, with the reason being “Changed appointment.”



**Appointment is clearly existed and it is in the table Appointment**



**Appointment for particular person is deleted by me, and now it does not exist in the table Appointment**

Graphical user interface, text, application

Description automatically generated

**Appointment now in the table Old Appointment, which means trigger is working**

Graphical user interface, text, application

Description automatically generated