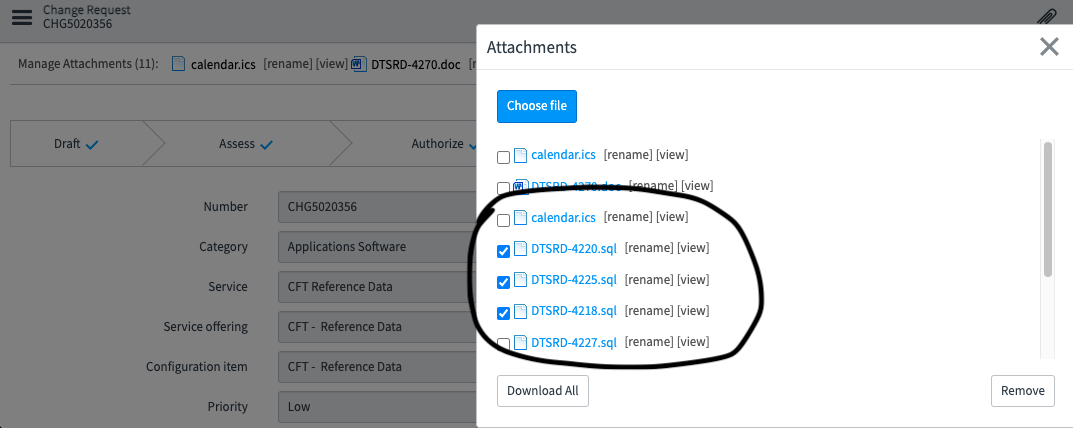
I have been reviewing some of the database tasks that the team perform, one of which includes manual updates to Production data through scheduled Change Requests.

These tasks can be time consuming for the team and require, in some cases, many manual steps. **All** manual manipulation of a Production database carries risk, and should **only be performed when necessary**, not as a routine task. We need to streamline these requests/processes, with a view to having them automated in a pipeline – removing all the manual steps.

As a result of my investigation, you will begin to see changes to some of the Change Requests that you are assigned, and I want to step you through these changes and the processes that you need to follow going forward. Additionally, I will be working through an example of this process in the Show & Tell tomorrow (20/03/2025).

Firstly, all data changes will be scripted by the requester – *as part of the PlatOps team, you are not expected to create any SQL scripts*. While the SQL may also be contained in the body of the request, each CR will have one (or more) SQL files attached to it. These files can be accessed at the top of the CR and should be downloaded to your laptop – this is what you will be running against the appropriate database.



*Ideally*, each CR should only contain 1 SQL script for each Server/database that needs to be updated. So, for refdata changes, this would be 2 scripts, one for [rd\_professional] and one for [rd\_user\_profile]. However, currently, I am aware that the services team combine multiple similar tickets into one CR, resulting in multiple SQL scripts (as with the above example) – even though those changes are all for the same database. This practice should change over time, particularly when the database changes are automated through a pipeline, but for not it will remain as 1 script per incident/ticket.

# Services Team

I have provided a template to the services team (current version included at the end of this document) that they are to use as the basis for each change and customise with the appropriate SQL statements.

The *services team* *developer* of the script MUST fully test the script on a lower environment, ensuring it is executed as a complete script, and not individual statements. Variables should be used in place of hard coding for any data that needs to be altered for testing the script. For example, the server name, and any other data such as email address or Id that needs to be used in production but does not exist on the test system.

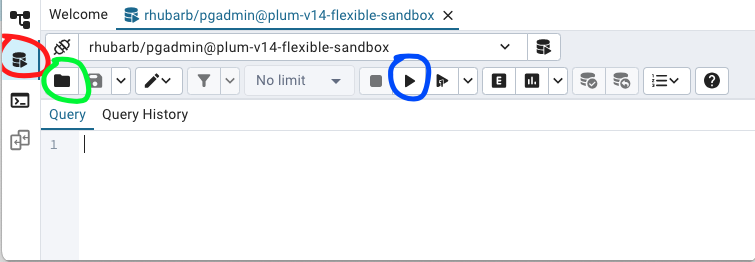
Once successfully tested, the only changes to be made to the script are to be the variable values, before the script is attached to the Change Request. The script(s) will be run as-is by PlatOps, without any manual review – it is the developers’ responsibility to ensure the script runs to completion without manual intervention or changes.

# Running scripts

I’m assuming that all these CR changes are completed using pgAdmin, if not then please let me know. The latest version of pgAdmin is 9.1, available [here](https://www.pgadmin.org/download/).

## PgAdmin - Query Tool

Use **pgAdmin** and connect to the correct Server/Database using **Query Tool Workspace** – highlighted below in **red**.



 Once connected, open the first SQL file, using the folder icon highlighted in **green**. **It is really important to note here that NO SQL should be copied and pasted into the pgAdmin windows** – everything to be run against a Production database should be from a saved script. Ad-hoc, typed or copy/pasted SQL has a high risk of being incorrect and not repeatable or traceable, so even in an emergency, SQL should be saved into a script and then run.

Once the SQL script is opened, it is to be run *as a complete script* – not as individual statements – using F5 or the Execute Script button highlighted in **blue**.

A template has been provided to the services team to use when raising the CRs – I’ve attached the template. You will see it contains only 2 sections. The first *Anonymous Block* performs a check that the database and server are as expected. There are 2 variables that the service team will populate, containing the correct production server name and the database name. When the script runs, these variables will be checked against the current connection. If the current connection is not as expected an exception will be raised and the script execution will be stopped. If the connection is correct, you will see a message like the one below and execution will continue.

NOTICE:  CORRECT Server and DB : Server [plum-v14-flexible-sandbox], Database [rhubarb]

The rest of the script will contain a single Anonymous Blocks (AB). An AB can have one of 2 possible outcomes – all SQL completes, and the transaction is committed to the database, or an exception is raised, and all SQL changes are rolled back.

The template has been designed so that the output should list the Jira Ticket and the Step number to match to the CR. An example of the expected output is below.

NOTICE: DTSRD-4220 Step 4.1 : Rows UPDATED by query=1

NOTICE: DTSRD-4220 Step 4.2 : Rows UPDATED by query=1

NOTICE: DTSRD-4220 Step 4.3 : Rows UPDATED by query=1

NOTICE: DTSRD-4220 SUCCESSFUL COMPLETION, all changes COMMITTED.

The following is an example of the expected output when one of the queries fails:

NOTICE: DTSRD-4220 Step 4.1 : Rows UPDATED by query=1

NOTICE: DTSRD-4220 Step 4.2 : ERROR RAISED, all changes ROLLED BACK

ERROR: P0001

ERROR:Change Failed [P0001]DTSRD-4220 Step 4.2 : ERROR wrong number of rows altered --> 0

SQL state: P0001

Detail: ERROR:Change Failed [P0001]DTSRD-4220 Step 4.2 : ERROR wrong number of rows altered --> 0

Context: PL/pgSQL function inline\_code\_block line 45 at RAISE

The output from each script is then to be copied into the CR.

## PgAdmin - PLSQL Tool

For those that prefer to use PSQL, this is also available in pgAdmin, highlighted below in **red**.

A screenshot of a computer

AI-generated content may be incorrect.

The command to run a script in PSQL is [\i <filename including full path>], as highlighted in **blue.** You will see the same script output as outlined previously.

There is a difference in using PSQL – the script output contains the name of the file in each line, so the output looks more cluttered and is not as easy to read. This can be minimised by cd-ing to the appropriate folder before running the script. In this way, only the script name is included in the output, and not the full path.

A screenshot of a computer program

AI-generated content may be incorrect.

## DBeaver

Some of the teams use DBeaver in place of pgAdmin. Once connected, open the first SQL file, using the File Menu item highlighted in **green**. As with pgAdmin, **it is really important to note here that NO SQL should be copied and pasted into the windows.**

Once the SQL script is opened, it is to be run *as a complete script* – not as individual statements – using the Execute SQL Script button highlighted in **blue**.

A screenshot of a computer

AI-generated content may be incorrect.

The Notify statements and any other messages will appear in the Output window. These messages are to be verified as correct (if testing) or copied into the CR as proof of output.

A screenshot of a computer program

AI-generated content may be incorrect.

If you have any questions, about this email, the templates, SQL, or databases in general then please feel free to contact me by teams/email/slack, either here or in Kainos. If you want me to screen share as you use the templates for the first few times, I can make myself available with a bit of notice. I don’t work on HMCTS every day, so you may need to contact me on Kainos. I can generally log onto HMCTS anytime I’m needed.

If you receive any database change requests that are not using the templates, then please let me know and I can discuss this with the team raising the requests. It may take some time to identify and reach all the teams requesting changes.

## SQL Template

/\*

SQL Template for Database Data Changes

Author : Donna Hull

Date : 02/04/2025

Version : 0.1

Notes : Each SQL Script is to contain TWO Anonymous blocks only. The first

for the Server/DB check and the second to perform the ALL work required

for the change. This second block can contain as many statements/actions

as are required to complete the change, but MUST contain The DECLARE/BEGIN

at the start of the block and the EXCEPTION/END at the end of the block.

Each individual step must include a RAISE NOTICE at the end of the step

giving details of the step action and/or output as confirmation of what was

done.

Testing of the script MUST be run by the developer on a lower environment.

Regardless of the tool used to run the script, the FULL script must be run AS

A SCRIPT, not individual statements.

Once developed and tested, any variables, such as server name, etc that were set

for testing purposes, must be updated to appropriate production values. The script

is then to be attached to the Change Request WITHOUT any changes to the rest of the

script. PlatOps will run the supplied script, without any manual review. It is the

developers responsibility to ensure the script runs to completion without manual

intervention or changes.

If multiple incidents are being run in one change window, then one script is required

per incident.

\*/

-- Check DB/Server Name

-- IMPORTANT : This section is NOT to be removed. All changes are to include a Server and database check

-- Set the ServerName and DBName variables as appropriate for your change

DO -- Anonymous Block

$ServerCheck$

DECLARE

correctServer Varchar(100) := 'plum-v14-flexible-sandbox'; --<ServerName>

correctDBName Varchar(100) := 'rhubarb'; --<DBName>

currentServer Varchar(100);

currentDatabase Varchar(100);

BEGIN

select replace(setting,'-data',''), current\_database()

into currentServer,currentDatabase

from pg\_settings

where name='azure.customer\_resource\_group';

if currentServer <> correctServer THEN

RAISE EXCEPTION

USING DETAIL = 'Wrong SERVER! Server [' || currentServer|| '] should be [' || correctServer|| ']';

END IF; if currentDatabase <> correctDBName THEN

RAISE EXCEPTION

USING DETAIL = 'Wrong DB! DB [' || currentDatabase || '] should be [' || correctDBName|| ']';

END IF;

RAISE NOTICE E'CORRECT Server and DB : Server [%], Database [%]\n',currentServer,currentDatabase;

END

$ServerCheck$;

--DTSRD-4220 - example with 3 linked sql statements

DO

$CR$

DECLARE

TicketNumber varchar(10) := 'DTSRD-4220';

AffectedRows integer;

StatusText varchar(20);

--<include any script variables here>

FirstId int := 1;

SecondId int := 4;

ThirdId int := 3;

--SecondId = 4 to see example of successful completion,

--SecondId = 2 to see example of failure

BEGIN

-- Step 4.1

StatusText='Step 4.1';

UPDATE dh\_test

SET notes = 'Testing Script second update fails'

WHERE id = FirstId;

GET DIAGNOSTICS AffectedRows = ROW\_COUNT;

if AffectedRows <> 1 then

RAISE EXCEPTION '% % : ERROR wrong number of rows altered --> %',TicketNumber,StatusText,AffectedRows;

end if;

RAISE NOTICE '% % : Rows UPDATED by query=%', TicketNumber,StatusText, AffectedRows ;

-- Step 4.2

StatusText='Step 4.2';

UPDATE dh\_test

SET notes = 'Testing Script failing step'

WHERE id = SecondId;

GET DIAGNOSTICS AffectedRows = ROW\_COUNT;

if AffectedRows <> 1 then

RAISE EXCEPTION '% % : ERROR wrong number of rows altered --> %',TicketNumber,StatusText,AffectedRows;

end if;

RAISE NOTICE '% % : Rows UPDATED by query=%', TicketNumber,StatusText, AffectedRows ;

-- Step 4.3

StatusText='Step 4.3';

UPDATE dh\_test

SET notes = 'Testing Script only gets executed if second completes'

WHERE id = ThirdId;

GET DIAGNOSTICS AffectedRows = ROW\_COUNT;

if AffectedRows <> 1 then

RAISE EXCEPTION '% % : ERROR wrong number of rows altered --> %',TicketNumber,StatusText,AffectedRows;

end if;

RAISE NOTICE '% % : Rows UPDATED by query=%', TicketNumber,StatusText, AffectedRows ;

-- <add more Steps here if required>

RAISE NOTICE E'% SUCCESSFUL COMPLETION, all changes COMMITTED.\n',TicketNumber;

EXCEPTION

WHEN others THEN

RAISE NOTICE '% % : ERROR RAISED, all changes ROLLED BACK', TicketNumber,StatusText;

RAISE EXCEPTION

USING DETAIL = 'ERROR:Change Failed [' || SQLSTATE|| ']' || SQLERRM;

--transaction is rolled back;

END

$CR$;

/\*

Example Output of Script with Failing Step 4.2

NOTICE: CORRECT Server and DB : Server [plum-v14-flexible-sandbox], Database [rhubarb]

NOTICE: DTSRD-4220 Step 4.1 : Rows UPDATED by query=1

NOTICE: DTSRD-4220 Step 4.2 : ERROR RAISED, all changes ROLLED BACK

ERROR: P0001

ERROR:Change Failed [P0001]DTSRD-4220 Step 4.2 : ERROR wrong number of rows altered --> 0

SQL state: P0001

Detail: ERROR:Change Failed [P0001]DTSRD-4220 Step 4.2 : ERROR wrong number of rows altered --> 0

Context: PL/pgSQL function inline\_code\_block line 56 at RAISE

Example Output of Script with Successful Step 4.2

NOTICE: CORRECT Server and DB : Server [plum-v14-flexible-sandbox], Database [rhubarb]

NOTICE: DTSRD-4220 Step 4.1 : Rows UPDATED by query=1

NOTICE: DTSRD-4220 Step 4.2 : Rows UPDATED by query=1

NOTICE: DTSRD-4220 Step 4.3 : Rows UPDATED by query=1

NOTICE: DTSRD-4220 SUCCESSFUL COMPLETION, all changes COMMITTED.

\*/