National Cancer Registration and Analysis Service's Cancer Analysis System (CAS) Guidance

Getting started with SQL and extracting data

Lora Frayling and Cong Chen based on a guide by Victoria Coupland and Casey Dunlop

Contents

Introduction

The Cancer Analysis System (CAS) was created and is used by the National Cancer Registration and Analysis Service (NCRAS) to store and analyse cancer registration data and other linked datasets for England and Wales. It was created in 2012 to combine cancer registration data for multiple regions into one central standardised system.

CAS is an Oracle database that uses SQL queries to access our datasets. This guidance has been developed to help new analysts to navigate around the system, write basic SQL queries, and use their own file space. You should aim to work through this document and carry out the tasks outlined to help get to know the system. This guide is designed to be used alongside other CAS guidance documents which can be found on the SVN and in the main NCRAS folder. <<add link>>

To learn more about this type of database management system and how to use SQL there are many free online tools to use such as Code Academy, W3Schools and Khan Academy. If you're part of the NCRAS team, an external SQL training course is also available. To find out more about this talk with your line manager.

If you have any questions or notice any errors in this document, please contact Victoria.Coupland@PHE.GOV.UK.

Loading the data

Please check that the data has been loaded correctly, in particular please check data types of the loaded data against those recorded in the schema provided.

The data types and SQL provided are tailored for the Oracle 11g database on which the CAS sits, you will need to modify schemata and table names described to match those for your data. You may also need to adjust some of the SQL functions to match your dialect.

Writing basic SQL queries

A brief introduction to SQL

An SQL query is a set of instructions or commands to get the data you want out of a database. Most basic queries are structured in the same way:

- What data do you want? (SELECT statement)
- Where does the data come from? (FROM clause)
- What criteria do you want to filter the data by? (WHERE clause)
- How do you want to group or order it? (GROUP BY and ORDER BY clauses)

1. Select and From

Firstly, what data do you want from your table? To ask to see everything in a table you can use the SELECT statement followed by an asterisk (*). Using the * is useful for exploring what data is within a table and how it's structured.

Next, where does the data come from? You then need to specify where the data should be retrieved from using a FROM clause. In the example below we've specified the data should come from SIM_SIM_AV_TUMOUR table which is found in the schema SIMULACRUM.

Every query must be ended with a semi-colon (;).

```
SELECT *
FROM SIMULACRUM.SIM_AV_TUMOUR;
```

Copy and paste the query above into a tab for your database. Then press the green arrow button (or press Ctrl + return while your cursor is inside the query) to run the SQL. The Query Result window should appear when the query has successfully run. This should be showing the entire SIM_AV_TUMOUR table.

i. Next, instead of selecting everything with *, try specifying some field names. You won't always want to see all the columns, specifying the field names shows you only what you'd like to see. Copy and paste the below into the worksheet (and delete the previous query). Here we're selecting the diagnosis date and the cancer site field from the same table as before. Note the comma needed between each field name.

```
SELECT diagnosisdatebest, site_icd10_o2
FROM SIMULACRUM.SIM_AV_TUMOUR;
```

ii. Next, select everything from SIM_AV_TUMOUR (you have the code to do this already). Look for the field names for year of diagnosis, age and sex. Write some SQL that just selects these three fields.

iii. Try selecting everything from the patient table

iv. Try selecting just the date of birth and where they died from the patient table

v. Try selecting everything from the treatment table

vi. Try selecting just the description of the event and the name of the provider from the treatment table

Each of these queries above should retrieve results relatively quickly. In the Query Result window you can see exactly how long it took to 'fetch' the first few rows in seconds. You'll notice for some of the queries below it will take a bit longer for the results be retrieved.

2. Where

Now we've selected all the data in a table or in specific columns, you may want to filter the results so you don't see every patient or every tumour. To do this you would use the WHERE clause. This brings back data only where a specific condition is met.

i. Copy and paste the new SQL query below. This query uses the * to select every column in the table, but instead of bringing back every patient, it will only bring back patients diagnosed in 2014

```
SELECT *
FROM SIMULACRUM.SIM_AV_TUMOUR
WHERE diagnosisyear = 2014;
```

ii. Write an SQL query to pull out tumours that were diagnosed on the 16th October 2014

iii. Write an SQL query to pull out patients that died of breast cancer

iv. You don't always need something as specific as the example above, like pulling out one date. The WHERE statement can be used to filter results in different ways including: less or more than a number, within a range, when a field is missing, or within a category. Try running the SQL query the below.

```
SELECT *
FROM SIMULACRUM.SIM_AV_TUMOUR
```

```
WHERE age>90;
```

v. Now try to write and run an SQL query to pull out all tumours where the patient was older than 17 when they were diagnosed

vi. You can also specify within a range using the BETWEEN operator, putting the smaller value first

```
SELECT *
FROM SIMULACRUM.SIM_AV_TUMOUR
WHERE age BETWEEN 18 AND 100;
```

vii. You can use SQL to find cases where a field is missing, or when a field is filled in. Try the query below. Notice the table we're retrieving the data from has changed.

```
SELECT *
FROM SIMULACRUM.SIM_AV_PATIENT
WHERE vitalstatusdate IS NULL;
```

Then replace the final line with:

```
WHERE vitalstatusdate IS NOT NULL;
```

viii. Your filter could look for a variable in a certain category or a list of categories. To do this you would use the IN operator. Run the SQL query below. This is pulling out data for all trachea, bronchus or lung tumours. Note the quotation marks around the codes are required because this datatype is not a number. Dates, text (string), or mixture of text and number variables (VARCHAR) all need quotation marks.

```
SELECT *
FROM SIMULACRUM.SIM_AV_TUMOUR
WHERE site_icd10_o2_3char IN ('C33','C34');
```

ix. Write a query using the NOT IN operator. You could write one where the site_icd10_o2_3char field was not C40 or C41. This would be pulling out all tumours that are not bone cancer.

Note: The codes used above are the International Classification of Diseases v10 (ICD10) codes and are important for understanding national cancer data. The coding systems have changed over time but in our datasets have been converted to ICD10 in data from 1995 onwards.

x. Often you will need to include more than one condition. To do this you would connect your where statements with operators AND and OR. The SQL query below pulls off all

tumours which are diagnosed in women who are 34. Males are coded as 1 and women as 2.

```
SELECT *
FROM SIMULACRUM.SIM_AV_TUMOUR
WHERE age = 34
AND sex = '2';
```

xi. Run the SQL query above but this time change the 'AND' to an 'OR'. What changes?

xii. Write an SQL query that selects patients who were born on the 1st January 1950 and are female. How many are there?

Note: There isn't any limit on how many 'AND's you can put in your query. If you're using more than one condition you may want to consider introducing brackets to avoid potential errors.

xiii. When you're specifying what you want to find in your WHERE clause be aware it is case sensitive. Run the query below. Adding UPPER after LIKE converts all lowercase letters into uppercase, so returns all entries were the text is upper or lower case. Now remove UPPER. What happens?

```
SELECT COUNT (patientid)
FROM SIMULACRUM.SIM_AV_PATIENT
WHERE DEATHCAUSECODE_1A LIKE UPPER('%c50%');
```

3. Ordering

Sometimes you may want to look at the data in a particular order.

i. Run the SQL query below. This would be used when you want to see the most recent data first. When was the most recent tumour diagnosed?

```
SELECT *
FROM SIMULACRUM.SIM_AV_TUMOUR
ORDER BY diagnosisdatebest DESC;
```

ii. Can you change the order from descending (desc) to ascending (asc)? When was the first tumour on the database diagnosed?

iii. You may want to see the results by the most recent diagnosis year (diagnosisyear) and then by diagnosis date. Write an SQL query that would do this.

Note: The ‘order by’ statement is the last part of the command to run. You can also write ‘ORDER BY 1, 2’. This will order by whatever is in column 1 first then column 2.

4. Counting

In the examples above we’ve covered how to pull out data, filter it and sort it. In SQL, there are also aggregate functions such as COUNT. Aggregate functions combine multiple rows of values into a single value. When using functions like COUNT, be careful to think about exactly what you’re counting - rows, tumours, patients, unique patients or a subset of one of these.

i. Count how many rows are in the tumour table.

```
SELECT COUNT(*)
FROM SIMULACRUM.SIM_AV_TUMOUR;
```

ii. Count how many patients there are in the patient table.

iii. Compare your answers to the last two questions. Why are they different?

iv. You can also be more specific with your counts by only selecting by certain criteria.

```
SELECT COUNT(*)
FROM SIMULACRUM.SIM_AV_PATIENT
WHERE vitalstatusdate IS NOT NULL;
```

v. Run the query above to count how many rows have a null deathdate, and then write another query to count how many rows have a deathdate filled in. Check it adds up to the total number of rows.

vi. Write a query that counts how many patients are alive. Did you get the same answer as the number of patients with a null death date? If not, any ideas why not?

vii. Write a query to count how many tumours were diagnosed in patients aged over 100.

viii. You can also count particular data items. Sometimes data items are null (blank) and these are not counted when you do this. Below is a query that counts how many rows there are, how many rows with a tumour id, and how many rows with an er_status

```
SELECT COUNT (*) ,COUNT(tumourid) , COUNT(er_status)
FROM SIMULACRUM.SIM_AV_TUMOUR;
```

ix. Does every row have a tumour id? Does every row have an er_status?

x. What proportion of the treatment table has an imaging description?

5. Counting in groups

You may want to know the total count split by a variable, like the number of tumours diagnosed by gender. For this you would use the GROUP BY function. Note whenever an aggregate function like COUNT is used, all other variables being counted must also be in the GROUP BY clause. ORDER BY can be used to place the groups in order, however is not necessary for grouping.

```
SELECT sex, COUNT(*)
FROM SIMULACRUM.SIM_AV_TUMOUR
GROUP BY sex
ORDER BY sex;
```

i. Run the query above. Are there more men or women?

ii. What are the age and sex breakdown of lung cancer patients?

6. Other aggregate functions (avg, min, max)

Counting is an aggregate function. There are many other useful aggregate functions like sum (SUM), average (AVG), minimum (MIN) and maximum (MAX).

```
SELECT MAX(age), SUM(age), COUNT(age), AVG(age)
FROM SIMULACRUM.SIM_AV_TUMOUR;
```

i. Run the query above. How old is the oldest person on the database diagnosed with cancer? Do you think this is a data quality problem? Can you find out what sort of cancer they had? What is the average age of a cancer patient?

ii. These aggregate functions can also be combined with the group by function e.g. the maximum age by gender. Try writing this into a query. How old is the oldest women?

iii. Write an SQL query to find the average age of men with cancer and the average age of women with cancer.

iv. Have people gotten older? Write a query that looks at how many cancers there are by year of diagnosis, and the average age by year of diagnosis. Can you spot some real data quality problems in the early years?

v. In your results you may not want to see all the decimal places. The TO_CHAR function can be used round values to a number of decimal places. The number of decimal places is determined by the number of decimal places given in the number in the function: '9,999.9' gives 1 decimal place, '9,999,99' gives 2, etc.

```
SELECT diagnosisyear, MAX(age), TO_CHAR (AVG(age),'9,999.99')
FROM SIMULACRUM.SIM_AV_TUMOUR
GROUP BY diagnosisyear
ORDER BY diagnosisyear;
```

vi. If you want to extract parts from a date, e.g. year, month, day, you can use the EXTRACT function, which will produce a new field with the extracted values. If you didn't want to see the full name of the field in the results, you can rename it by following the EXTRACT function with AS then the new name in your query. Try the below:

```
SELECT EXTRACT(YEAR FROM diagnosisdatebest) AS diagnosisyear,  MAX(age),
TO_CHAR(AV(age),'9,999.99')
        FROM SIMULACRUM.SIM_AV_TUMOUR
        GROUP BY EXTRACT(YEAR FROM diagnosisdatebest)
        ORDER BY diagnosisyear;
```

vii. You can also use the TO_CHAR function to display the dates is a different format. In CAS the default date format is 'yyyy/mm/dd'.

```
SELECT diagnosisdatebest
          , TO_CHAR (diagnosisdatebest,'yyyy/mm/dd')
          , TO_CHAR (diagnosisdatebest,'dd/mm/yyyy')
          , TO_CHAR (diagnosisdatebest,'FMMonth DD, YYYY')
FROM SIMULACRUM.SIM_AV_TUMOUR;
```

Note: This can also be changed in under Tools, then Preferences, Database (NLS).

7. Distinct

The DISTINCT clause is used with the SELECT command to remove duplicates. This is useful for counting e.g. unique patients. To do this you would need to select only patientid.

i. Try running the query below. What happens when you remove distinct? What are the results showing you?

```
SELECT COUNT (DISTINCT patientid)
FROM SIMULACRUM.SIM_AV_TUMOUR
WHERE site_icd10_o2_3char IN ('C50','C53','C54','C55','C56')
AND EXTRACT(YEAR FROM diagnosisdatebest) = 2014;
```

Note: The command is dependent on what else is in the select list. If there is another variable e.g. patientid and diagnosiscode it would give distinct patientid for each diagnosiscode. If you added these results together you would be double counting anyone with both tumours.

ii. What happens when you add the additional field in the query below. Try adding a geography field. What are the results showing you?

```
SELECT COUNT (DISTINCT patientid), site_icd10_o2_3char
FROM SIMULACRUM.SIM_AV_TUMOUR
WHERE site_icd10_o2_3char IN ('C50','C53','C54','C55','C56')
AND EXTRACT(YEAR FROM diagnosisdatebest)= 2014
GROUP BY site_icd10_o2_3char;
```

Note: If you are linking and duplicate rows are produced you shouldn't really use DISTINCT to account for this. It may well indicate an issue with the linkage which could need to be resolved.

8. Case statements

You may need to make a new field based on values in another field that splits the data into different groups – like age groups. The function that can do this is CASE WHEN. Notice it has another group (ELSE) and needs to be ended with a the command END. To create a name for this field you then write AS and then the new name.

```
SELECT age,
CASE
        WHEN age <25 THEN 'Under 25'
        WHEN age <70 THEN '25 - 69'
        ELSE '70+'
        END AS agegroup
FROM SIMULACRUM.SIM_AV_TUMOUR;
```

i. Run the above SQL query. Can you amend it so that it groups people into under 40s, 40 – 80, and over 80s?

ii. Write an SQL query using the CREG_CODE field that returns, for every tumour, 'In ECRIC region' or 'Not in ECRIC region'. The ECRIC code is Y0401.

iii. Case statements can also be used to derive a new field from multiple fields by using AND and OR. Try writing a query that splits that data that combined 2 or more fields. This function can be useful if you ever need to create flags in your data.

iv. In your case statement you may want to use multiple variable types. For example, you may want to specify a date if there is a value in another field. To do this you would need to change the variable type using the TO_CHAR function.

```
SELECT patientid,
CASE WHEN sex IN (2) THEN TO_CHAR(diagnosisdatebest, 'DD/MM/YYY')
        END AS diagdateiffemale
 FROM SIMULACRUM.SIM_AV_TUMOUR
 WHERE site_icd10_o2_3char = 'C50' AND EXTRACT(YEAR FROM
diagnosisdatebest)= 2014;
```

The DECODE function is similar to the CASE statement except that it has less functionally, but is shorter to write if you need to use very long non-complex case statements.

9. Linking/joining tables

You may need to use data from more than one table. To do this you would link or join tables together. Whenever you link tables you'll need to think carefully about the relationship between the tables you're joining. How two tables link will depend on the tables you're joining. One of the following situations could happen:

- For every row in the first table, there is exactly one row in the second table. This should be what happens with look up tables e.g. a table with ICD codes links to the cancer type in text.
- For some rows in the first table, there are no rows in the second table. For example, you might have a table of all cancers and a table of all surgeries. Some cancers won't have had surgery.
- For some rows in the first table, there is more than one row in the second table. For example, some cancers have had more than one surgery.
- For some rows in the first table, there are no rows but for others there is more than one.

It's very easy to make a mistake when joining tables. For example:

```
SELECT *
FROM SIMULACRUM.SIM_AV_PATIENT, SIMULACRUM.SIM_AV_TUMOUR;
```

i. Run the query above. It should look like the patient table and the tumour table side by side. How many rows are there in the patient table? How many rows are there in the tumour table? How many rows does query above give you? Can you work out what the query has done? Comparing the patient IDs from the patient part of the table and the tumour part of the table might help.

   This is called a Cartesian cross product. Every row of the patient table has been stuck next to every row of the tumour table, even if they have nothing in common which is incorrect. There needs to be criterion to join on.

ii. Run the query below. This is joining the tables on the patientid by a left join by the command LEFT OUTER JOIN. Compare the patient ID columns. Count how many rows the query is returning.

```
SELECT *
FROM SIMULACRUM.SIM_AV_PATIENT
LEFT OUTER JOIN SIMULACRUM.SIM_AV_TUMOUR
ON SIMULACRUM.SIM_AV_PATIENT.patientid =
SIMULACRUM.SIM_AV_TUMOUR.patientid;
```

Note: A left join joins the two tables, and selects all rows in the first table that match those in the second on the join criterion plus any rows in the first table that don't match. In the example above the tables are matching on patientid.

There are also inner joins and right joins. An inner join selects all rows in the first table that match those in the second on the join criterion only.

A right join selects all rows in the first table that match those in the second on the join criteria plus any rows in the second table that don't match.

To find out more about all joins click here.

iii. Instead of writing out the table name each time you need to join two tables you can give the table an abbreviated name or a single letter. In the query below SIM_AV_PATIENT is named 'p' and SIM_AV_TUMOUR is named 't'.

```
SELECT *
FROM SIMULACRUM.SIM_AV_PATIENT p
LEFT OUTER JOIN SIMULACRUM.SIM_AV_TUMOUR t
ON p.patientid = t.patientid;
```

iv. Run the query below. Can you then count how many tumours there are for each deprivation quintile? Try looking at the queries above if you need a reminder.

```
SELECT tumourid, laterality
FROM SIMULACRUM.SIM_AV_TUMOUR;
```

v. You'll need a lookup table to make sense of the basisofdiagnosis code. Run the query below. This is a lookup table to turn the basis of diagnosis code into a short description.

```
SELECT *
FROM zlaterality;
```

In the previous examples, the tables we've used have all been retrieved from the CASREF01 database. This lookup table is in another database (CAS1702) under a User called SPRINGMVC3. Using the @ symbol we can connect to a different database than the one we are running the query in. Note that this only works between casref01 and a monthly snapshot. You cannot link between two monthly snapshots.

vi. Now join the two tables together. For every tumour ID you should have the words as well as the codes. Can you see how the two tables have been joined together?

```
SELECT TUMOURID, laterality, SHORTDESC
FROM SIMULACRUM.SIM_AV_TUMOUR t
LEFT OUTER JOIN zlaterality z
ON t.basisofdiagnosis = z.zbasisid;
```

10. Subqueries

In some cases, you'll need to run one query to get a table, and then do further queries on that table. Two common ways of doing this are by using a WITH clause or using nested queries. For most queries, the approach you use will be down to personal preference. However, as queries become more complex, the WITH clause may be more efficient in retrieving data.

The example below uses the WITH clause to find out how many patients were diagnosed with breast cancer in 2014.

i. Firstly, you can write a query to get all the breast cancer in 2014:

```
SELECT *
FROM simulacrum.sim_av_tumour
WHERE site_icd10_o2_3char = 'C50'
AND EXTRACT(YEAR FROM diagnosisdatebest) = 2014;
```

ii. You can amend this query so instead of all treatments, it returns one row per patient:

```
SELECT DISTINCT patientid
FROM simulacrum.sim_av_tumour
WHERE site_icd10_o2_3char = 'C50'
AND EXTRACT(YEAR FROM diagnosisdatebest) = 2014;
```

iii. Or to give you more information on how many tumours each patient had:

```
SELECT patientid, COUNT(*)
FROM simulacrum.sim_av_tumour
WHERE site_icd10_o2_3char = 'C50'
AND EXTRACT(YEAR FROM diagnosisdatebest) = 2014
GROUP BY patientid;
```

iv. Now you want to write a query on this query. One way to do this is to use 'WITH [tablename] as' at the start. You can call this table anything you want.

```
WITH breastpatients AS
(SELECT patientid, count(*)
FROM simulacrum.sim_av_tumour
WHERE site_icd10_o2_3char = 'C50'
AND EXTRACT(YEAR FROM diagnosisdatebest) = 2014
GROUP BY patientid)
```

```
SELECT * FROM breastpatients;
```

v. Now you can do the query below to see how many there are:

```
WITH breastpatients AS
(SELECT patientid, count(*)
FROM simulacrum.sim_av_tumour
WHERE site_icd10_o2_3char = 'C50'
AND EXTRACT(YEAR FROM diagnosisdatebest) = 2014
GROUP BY patientid)
SELECT COUNT(*)
FROM breastpatients;
```

vi. Then you can start doing things like seeing how many tumours had 1 surgery, how many had 2 surgeries, etc. It's easiest if you give the COUNT(*) column a proper name using 'AS'

```
WITH breastpatients AS
(SELECT patientid, count(*) as tumourcount
FROM simulacrum.sim_av_tumour
WHERE site_icd10_o2_3char = 'C50'
AND EXTRACT(YEAR FROM diagnosisdatebest) = 2014
GROUP BY patientid)SELECT tumourcount, count(*)
FROM breastpatients
GROUP BY tumourcount
ORDER BY tumourcount asc;
```

Multiple tables can be strung together in this way if needed. Each can retrieve data from any of the tables above it. They can be structured like the below:

```
WITH [table1] AS ([query])      -- first query
, [table2] AS ([query])         -- second query
, [table3] AS ([query])         -- third query (etc)
[query]                         -- final query
```

vii. We can also do subqueries by using nested queries. If the previous query was written as a nested query, it would look like this:

```
SELECT tumourcount, count(*)
FROM
(SELECT patientid, count(*) as tumourcount
FROM simulacrum.sim_av_tumour
WHERE site_icd10_o2_3char = 'C50'
AND EXTRACT(YEAR FROM diagnosisdatebest) = 2014
GROUP BY patientid) breastpatients
GROUP BY tumourcount
ORDER BY tumourcount asc;
```

The first query is nested within the second. In the same way as above, we're asking the results from the second query to come from the results table of the first query. This table

has been called tumourswithsurgery but could be called a single letter. Multiple tables can be nested within a query.

viii. What is the difference between ‘how many patients had a breast cancer aged 65’ and ‘how many patients aged 65 have breast cancer’? Can you write an SQL query for the second question?

11. More advanced functions

There are many more advanced functions you can use in SQL. Below we’ll briefly discuss some of the more useful ones you may need.

Row_number and Rank
Row number is a function that assigns a unique number to each row, or each row in a partition – like numbering by patiend id (giving each patient a new number), or numbering by patientid and tumourid (giving each tumour that a patient has a new number). Rank is similar but assigns a ranked number to each row, or each row in a partition. These can be useful when dealing with duplicates. To learn about Row_number and Rank, go to the oracle webpage here and here.

Regular expressions
The regular expression function is a way of searching for things in a flexible way. For example if you wanted to look for any cancer site that contains the word ‘skin’ or any provider with ‘royal’ in the name. To learn about regular expressions, go to the oracle webpage here.

Pivot and Unpivot
The pivot function is a way of turning a data table with the data in rows into their own columns. Unpivot can turn the results back, from columns into rows. If you need to do considerable data manipulation you should consider using a statistical package such as Stata to do this more easily. To learn about pivots, go to the oracle webpage here.

Listagg
The listagg function is a way of concatenating values from multiple rows into a single string, according to ORDER BY expression. This is useful for listing all distinct values in a field into one for example concatenating all chemotherapy drug names used into a single string for each cancer type. To learn more about Listagg, go to the oracle page here.

Creating your own tables

Your file space

Each CAS user has permissions to create their own tables in their own file space in the Snapshots and CASREF01 databases. While each user has a limited amount of space, it should be sufficient for most users. If you run out of space you may need to delete old tables you have created and possibly empty your recycle bin.

Creating a table in your file space
To create a table in your own file space using data from CAS, open a new tab for the database you would like the table to be stored. In the tab, write the query to select the data you want. Then add brackets around this query, and in the top line use CREATE TABLE followed by a name for your table (no spaces). Like the below:

```
CREATE TABLE table01 nologging AS
(SELECT patientid
FROM SIMULACRUM.SIM_AV_PATIENT
WHERE diagnosisyear BETWEEN 2014 AND 2015);
```

Clicking run on the query will create a table called table01, which can be found under ‘Tables’ in the database tab where the query has been run in . You can then use the data in this table as you would the other main tables. Note: always including ‘nologging’ in order to speed up the query.

Deleting a table from your filespace
To delete a table from your filespace you should use the DROP TABLE function.

```
DROP TABLE table01 PURGE;
```

You could also right click on your table, click DROP and then Apply. By ticking Purge (or including it in your code) your table will be unrecoverable, by not ticking purge it will go into the recycle bin.

Giving and revoking other CAS users access your tables
You may need to give another CAS user access to your tables. To do this you would use the GRANT function.

```
GRANT SELECT, ALTER ON analysisyourname.table_01 TO analysisotheruser;
```

This will allow the user to view and alter your table.

Important note: Please be aware of the permission level of the person you are granting table access to and do not give them access to fields they would not normally be able to see.

You can revoke table access using the command below. REVOKE SELECT, ALTER ON analysisyourname.table_01 TO analysisotheruser;

Importing data into your file space

You may not need to do this too often but if you do instructions are below. If you are uploading big datasets or lookup tables then please speak with Victoria.coupland@phe.gov.uk in the first instance.

i. Firstly, you'll need to get your data in the right format.
SQL developer will accept csv, excel (.xls), or plain text (.tsv) but not xlsx. You'll need to save your file in the correct format for the upload to be successful: a csv file is where the fields are separated by *commas* while a text file may have deliminators (values that separate the fields) that are spaces " ", pipes "|", or semicolons ";".

ii. Once you have your data in the right format log into the tunnel and then SQL developer. Right click on 'Tables' in the space that you would like you table to appear. This can be in CASREF or the Snapshots. Then on Click on Import Data. Click on Browse and find your file and click 'open'.

iii. Look at the data preview. Does it look right? You'll need to specify if it has headers here. If the data doesn't look right click cancel check the file format you have used is correct. If your dates have swapped day and month around you'll be able to change the date type later. Click next.

iv. Give your table a name (no spaces) and click next again. Here you can choose the columns you need at this point. If you need all you can don't change anything here. Click next.

v. In this window you can verify the data you are going to upload for each column. You change the column names, the datatype and the size. Size is the maximum field length, i.e. how many characters a field is allowed to have. SQL developer will automatically predict the datatype and maximum field length by looking at the first 20 rows of your file so these could be incorrect. Look at each of your columns to check what's been added and that there are no error flags that appear in the Data window below.

vi. There are multiple reasons why your data may show an error flag. Here are a few examples:

- If the first few rows look like dates and then a text word appears. This is because SQL developer has assumed all data in the column are dates. A way to avoid this is to either correct the errors in the original file or change the columns into the 'varchar2' datatype and correct the errors later (e.g. using to_date function).
- If there is a higher number of characters in the field than specified in the column field length. To correct this you'll need to increase the set field length of the column. If you don't know your data well, you can always set a high number that will capture everything. Note, if the first few rows of your table are blank they SQL could have predicted the field length to 0.

- If your column names have spaces in them. If your original data had a column called ‘Date of Diagnosis’ you will have to change the name to ‘Date_of_Diagnosis’ or ‘Dateofdiagnosis’.
- If you have dates in the wrong format. You can choose the date format for your column from the format dropdown menu.

vii. If your data looks ok click Next and then Finish. You’ll see a loading bar and then either a pop up window that says Upload Successful or Upload Failed.

viii. If the upload failed: SQL Developer only checks on the first 100 rows, so even if there is success for the verification it could fail to upload after this point because of an issue in a later row. If this happens you can press cancel, and then look at your original data to spot the reason for the error. You’ll need to go back to the beginning of the upload process and remember any other changes you previously made in the verification stage.

ix. If the upload is successful: you should be able to see your new table in the database you loaded it into under Tables. You may need to refresh the connections before it appears (click the blue arrows symbol in the connections panel).

x. Keep a local file copy of any scratch table data you create so you can re-load it if necessary.

Other useful tips

Below are some other useful to tips to get you started in SQL.

Annotating script

When writing an SQL script, its best practice to include a title and some notes explaining the purpose of the script, when it was created and notes to explain each stage in your query. These annotations will be invaluable when you come to look at your script later and for other people to be able to use and understand your script. Notes can be added by using a double dash (--) or /* at the beginning and */ at the end.
We'd recommend something like this:

```
/* Counting the number of patients who were diagnosed with lung cancer in 2014, by gender and age
CREATED BY: Vicki Coupland
DATE: 09.02.2018
*/
-- 1: Selecting variables
```

These annotating functions can also be used when you want to keep in sections of your script in that are no longer being used.

Customising your tab display

It is possible to customise your SQL developer query tab, such as having numbered lines (useful for being able to find errors), changing the colour of text when it is a function/field/string (useful for reading script more easily), and using a different date format permanently (to save time if frequently converting). To do this, click on Tools then Preferences then Code Editor (Line Gutter and PL/SQL Syntax Colours) or Database (NLS).

Problem solving errors

Sometimes a query will not run and will come back with an error message. These error messages will indicate a line number to trace back to look for your error. If you're unsure, these error messages are usually common so can be googled to find answers. Websites like Stack Overflow and the Oracle website are very useful for finding solutions.

Optimising the running time of queries

To learn about how to write SQL in the most efficient way to run queries more quickly, take a look at this presentation from the CAS developer team. If you need more help contact the CAS developer team (see contacts list). <<need to update this bit>>

Exporting data out

To select all the data quickly you can click control+shift+copy which will include both the headers and the data. Using ctrl+copy will only copy the data. Alternatively, you can export data using the export function in SQL. Right click any column and select the export option.

Please be aware of exactly where you are exporting the data whenever you take data out of CAS as it could be patient identifiable. Only save data to places that have been designated as secure for patient identifiable data (PID). Your line manager will be able to advise you on where to save this type of data. <<refer to 'accessing data' policy>>

Cancelling queries

You may run a query that will not finish (the loading sign will not disappear) and needs to be cancelled. This can be done by clicking the red cross next the loading sign; depending on what you are running this could be instant or take some time. In some cases the query even after clicking cancel will continue to run in the background and slow down your next queries. If this happens the query will need to be cancelled by the CAS developer team. If you feel your query will need to be cancelled by a developer please contact the CAS analytical lead (Vicki Coupland) in the first instance.

Closing SQL developer

When you close SQL developer it will ask you if you would like to commit changes or roll back changes. Click commit unless you have created a table you do not want to save.

Oracle SQL vs. Microsoft SQL

Microsoft SQL is another relational database management system that is similar to Oracle but has some differences in the command language used. There is a separate SACT database which uses Microsoft SQL and contains the most recent SACT data. If you need to write queries using Microsoft SQL please contact the SACT dataset lead. A list of current leads can be found <<here>>

Writing complex script

If you’re writing a complex script and coming across issues that your line manager/colleagues can’t solve, then there are colleagues in the NCRAS team that have advanced SQL and/or CAS knowledge who may be able to help:

- simulacrum@healthdatainsight.org.uk