NDRS
NATIONAL DISEASE REGISTRATION SERVICE

National Cancer Registration and Analysis Service's Guidance

Getting started with SQL and extracting data

Contents

1. Introduction

The Cancer Analysis System (CAS) was created and is used by the National Disease Registration Service (NDRS) 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 from multiple regions into one central standardised system.

CAS is an Oracle database that uses SQL queries to access our datasets. The Simulacrum is a synthetic dataset modelled on a subset of data stored in the CAS. This guidance has been developed to help new analysts, write basic SQL queries that could be executed against the CAS. 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 Simulacrum guidance documents to support the development of scripts that can be successfully executed against the Simulacrum and CAS datasets. This guide has been adapted from existing NDRS guidance for CAS users, to be relevant to the Simulacrum v2.1.0 dataset.

Answers to these tasks can be found in the Appendix.

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, Khan Academy and W3Schools.

2. Loading the Simulacrum data

The Simulacrum is a series of csv files that can be loaded into any database management or statistical analysis software for querying and analysis. This guidance assumes that the Simulacrum is loaded into a database management or software system that allows SQL queries to be executed against 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 likely 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.

3. 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

i. 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 within a table and how it's structured.

Note: queries should be ended with a semi-colon (;).

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_AV_TUMOUR table which is found in the schema SIMULACRUM.

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

Copy and paste the query above into a new worksheet for your database. Then press the green arrow button to run the SQL (or CTRL+ENTER or F9). The Query Result window should appear when the query has successfully run. This should be showing the entire SIM_AV_TUMOUR table. You could write this query all on one line or split it over multiple lines as above. When it comes to more complex queries it is probably better to split over multiple lines as it is easier to read.

HINT: Useful to go to Tools > Preferences > Database (expand +) > Worksheet and tick the box 'Show query results in new tabs'. This will then open query results in separate windows which helps for clarification and manging your queries.

ii. 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 (you can delete the previous query, add this new one underneath, or open a new worksheet). Here we're selecting the diagnosis date and the description of the cancer site field from the same table as before. Although the field names appear capitalised within the table, you could use lower case when entering the field names within the SQL query to make the layout clearer. If unsure of the full name of a table field, you can prompt SQL to bring up the different options by writing the first few letters e.g. "diagno" which then prompts the different fields starting with these letters to appear in a drop down list (Note: this only works if you've already added your from statement).

iii. Note the comma needed between each field name.

```
SELECT DIAGNOSISDATEBEST, SITE_ICD10_O2_3CHAR
FROM SIMULACRUM.SIM_AV_TUMOUR;
```

iv. 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 gender. Write some SQL that just selects these three fields.

v. Try selecting everything from the patient table.

vi. Try selecting just the vital status date and the location of death from the patient table. Notice the date format of the vital status date field. When you specify a date, it will need to be in this same format (to find out how to change this, go to Useful Tips on page 26).

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 with breast cancer

```
SELECT *
FROM SIMULACRUM.SIM_AV_TUMOUR
WHERE SITE_ICD10_O2_3CHAR = 'C50';
```

ii. Write an SQL query to pull out tumours that were diagnosed on the 16 October 2018.

iii. 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;
```

iv. 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.

v. 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;
```

vi. 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 DATE_FIRST_SURGERY IS NULL;
```

THEN REPLACE THE FINAL LINE WITH:

```
WHERE DATE_FIRST_SURGERY IS NOT NULL;
```

This should now return all DATE_FIRST_SURGEY entries that contain data.

vii. 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');
```

viii. 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. For more on cancer registration and coding see the Cancer Registration training module here.

ix. 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 females as ‘2’.

```
SELECT *
FROM SIMULACRUM.SIM_AV_TUMOUR
WHERE AGE = 34
AND GENDER = '2';
```

x. Run the SQL query above but this time change `AND` to `OR`. What changes?

xi. Write an SQL query that selects patients who were diagnosed on the 1 January 2001 and are female.

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.

xii. Sometimes you may want to look for something but not be too specific – like looking for patients with cancer on the first line of their death certificate. To do this you need to use the `LIKE` operator with a wildcard character (`%`). The query below selects patients who have any mention of cancer on their death certificate.

```
SELECT *
FROM SIMULACRUM.SIM_AV_PATIENT
WHERE DEATHCAUSECODE_1A LIKE ('%c%');
```

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 where the text is upper or lower case. Now remove `UPPER`. What happens?

```
SELECT *
FROM SIMULACRUM.SIM_AV_PATIENT
WHERE DEATHCAUSECODE_1A LIKE UPPER('%c%');
```

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;
```

Note: Did you notice how long this query took to run compared to the previous queries? Adding an `ORDER BY` clause to a query will slow it down – keep this in mind when you're writing more complex queries.

Can you change the order from descending (`DESC`) to ascending (`ASC`)? When was the first tumour on the database diagnosed?

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 SQL EXTRACT(YEAR FROM DATE) function may be useful here.

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 retrieve data, filter 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?

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

```
SELECT COUNT (*)
FROM SIMULACRUM.SIM_AV_TUMOUR
WHERE DATE_FIRST_SURGERY IS NULL;
```

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

v. Write a query that counts how many patients are alive.

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

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

```
SELECT COUNT (*)
, COUNT(TUMOURID)
, COUNT(ER_STATUS)
FROM SIMULACRUM.SIM_AV_TUMOUR;
```

Note: It's good practise to include the comma at the front of the statement. By doing this you can easily use a double dash (--) to run the query without using this row.

viii. Does every row have a TUMOURID? Does every row have an ER_STATUS?

ix. What proportion of the patient table has an underlying cause of death coded?

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 `GROUP BY`. Whenever an aggregate function is used (e.g. `COUNT`), all other variables being used, like gender in the example below, must also be in the `GROUP BY` clause. The `ORDER BY` clause is optional.

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

Run the query above. Are there more men or women? Try running the query without `GROUP BY GENDER` – what error message do you see?

What are the ethnicity and gender 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;
```

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?

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?

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

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. HINT: EXTRACT(YEAR FROM DATE) will be helpful here.

In your results you may not want to see all the decimal places. The `ROUND` function can be used to do this.

```
SELECT GENDER, MAX(AGE), ROUND(AVG(AGE), 2)
FROM SIMULACRUM.SIM_AV_TUMOUR
GROUP BY GENDER
ORDER BY GENDER;
```

If you didn't want to see the full name of the field in the results, you can rename it by following the name by `AS` then the new name in your query. Try the below:

```
SELECT GENDER, MAX(AGE), ROUND(AVG(AGE), 2) AS AVG_AGE
FROM SIMULACRUM.SIM_AV_TUMOUR
GROUP BY GENDER
ORDER BY GENDER;
```

You can also use the `TO_CHAR` function to round the number of decimal places – this function changes the number to text. You can also use this same function to display the dates in a different format. In the Simulacrum 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 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 DIAGNOSISDATEBEST BETWEEN ‘2019-01-01’ AND ‘2019-12-
31’;
```

Note: The command is dependent on what else is in the select list. In this query you are counting all patients that are diagnosed with any of these types of tumours.

```
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 DIAGNOSISDATEBEST BETWEEN ‘2019-01-01’ AND ‘2019-12-
31’GROUP BY SITE_ICD10_O2_3CHAR;
```

In this query you are counting how many patients had each type of cancer. This would include anyone who was diagnosed with breast cancer and ovarian cancer in both rows. If you added these results together you would be counting twice anyone that has been diagnosed with multiple tumour types.

What total count did you get for this query? How much does it differ from the previous query?

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. Linking tables together is explained in more detail in section 9.*

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 is used as another variable so will need a comma if it's used in a list. This function has another group (`ELSE`) which is optional and needs to be ended with a new name (`END AS`).

```
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;
```

The formatting/layout isn't important here - it could be written all on one line or split across multiple lines. As you write more complex scripts you may develop a style of how you prefer to format them, but generally it's easier to read code if it's split across multiple lines.

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

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.

Case statements can also be used to derive a new field from multiple fields by using `AND` and `OR`. Run the query below. What is it doing?

```
SELECT PATIENTID
,CASE WHEN DIAGNOSISDATEBEST BETWEEN TO_DATE(‘2019-01-
01’,’YYYY-MM-DD’) AND TO_DATE(‘2019-12-31,’YYYY-MM-DD’)
     AND SITE_ICD10_O2_3CHAR IN ('C50')
THEN ‘BREAST_2019’
ELSE ‘BREAST_OTHER_YEAR’
END AS BREAST_FLAG
FROM SIMULACRUM.SIM_AV_TUMOUR;
```

This function can be useful if you ever need to create flag specific rows in your data. In the example above you might be flagging where the data needs to be removed from your data extract.

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 VITALSTATUS IN ('A')
THEN TO_CHAR(DIAGNOSISDATEBEST, 'DD/MM/YYYY')
END AS DIAGDATEIFA
FROM SIMULACRUM.SIM_AV_TUMOUR
WHERE SITE_ICD10_O2_3CHAR = 'C50'
AND DIAGNOSISDATEBEST BETWEEN TO_DATE(‘2018-01-
01’,’YYYY-MM-DD’) AND TO_DATE(‘2018-12-31,’YYYY-MM-DD’);
```

`DECODE` is similar to `CASE` except that it has less functionality, however it 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 the 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. 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 JOIN`s and `RIGHT JOIN`s. 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.

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 AV_PATIENT is named ‘AVP’ and AV_TUMOUR is named ‘AVT’.

```
SELECT *
FROM SIMULACRUM.SIM_AV_PATIENT AVP
LEFT OUTER JOIN SIMULACRUM.SIM_AV_TUMOUR AVT
ON AVP.PATIENTID = AVT.PATIENTID;
```

Run the query below.

iii. Can you then count how many tumours there are for each behaviour type? Try looking at the queries above if you need a reminder.

```
SELECT TUMOURID, BEHAVIOUR_ICD10_O2
FROM SIMULACRUM.SIM_AV_TUMOUR;
```

You’ll need a lookup table to make sense of the BEHAVIOUR_ICD10_O2 code. In the previous examples, the tables we’ve used have all been retrieved from the SIMULACRUM schema. For the purposes of this query, we assume the lookup tables provided with the Simulacrum have been loaded into the same database is under a schema called LOOKUPS. Run the query below. This lookup table turns the behaviour code into a short description.

```
SELECT *
FROM LOOKUPS.Z_BEHAVIOUR;
```

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, BEHAVIOUR_ICD10_O2, DESCRIPTION
FROM SIMULACRUM.SIM_AV_TUMOUR AVT
LEFT OUTER JOIN LOOKUPS.Z_BEHAVIOUR ZB
ON AVT. BEHAVIOUR_ICD10_O2 = ZB.CODE;
```

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 tumours had surgery in 2010.

Firstly, you can write a query to get all tests of EGFR:

```
SELECT *
FROM SIMULACRUM.SIM_AV_GENE
WHERE GENE_DESC='EGFR'
AND EXTRACT(YEAR FROM DATE_OVERALL_TS)IN ('2018',’2019’);
```

You can amend this query so instead of all records, it returns the unique IDS of all tumours tested:

```
SELECT DISTINCT TUMOURID
SIMULACRUM.SIM_AV_GENE
WHERE GENE_DESC='EGFR'
AND EXTRACT(YEAR FROM DATE_OVERALL_TS) IN ('2018',’2019’);
```

Or to give you more information on the results of the EGFR tests:

```
SELECT OVERALL_TS, COUNT(*)
FROM SIMULACRUM.SIM_AV_GENE
WHERE GENE_DESC='EGFR'
AND     EXTRACT(YEAR     FROM     DATE_OVERALL_TS)     IN
('2018',’2019’)GROUP BY OVERALL_TS;
```

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 EGFRRESULTS AS
(SELECT OVERALL_TS, COUNT(*)
FROM SIMULACRUM.SIM_AV_GENE
WHERE GENE_DESC='EGFR'
AND EXTRACT(YEAR FROM DATE_OVERALL_TS) IN ('2018',’2019’)
GROUP BY OVERALL_TS
)

SELECT * FROM EGFRRESULTS;
```

Now you can do the query below to restrict to only normal and abnormal results:

```
WITH EGFRRESULTS AS
(SELECT OVERALL_TS, COUNT(*)
FROM SIMULACRUM.SIM_AV_GENE
WHERE GENE_DESC='EGFR'
AND EXTRACT(YEAR FROM DATE_OVERALL_TS) IN ('2018',’2019’)
GROUP BY OVERALL_TS
)


SELECT *
FROM EGFRRESULTS
WHERE OVERALL_TS IN ('a:abnormal', 'b:normal') ;
```

Then you can start doing things like seeing how many tumours had 1 gene tested, how many had 2 genes tested, etc. It’s easiest if you give the `COUNT(*)` column a proper name using `AS`

```
WITH TUMOURSWITHGENETESTS AS
(SELECT TUMOURID, COUNT(*) AS GENESCOUNT
FROM SIMULACRUM.SIM_AV_GENE
WHERE
EXTRACT(YEAR FROM DATE_OVERALL_TS) IN ('2018',’2019’)
GROUP BY TUMOURID)

SELECT GENESCOUNT, COUNT(*)
FROM TUMOURSWITHGENETESTS
GROUP BY GENESCOUNTORDER BY GENESCOUNT 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)
[select * from table3]         -- final query
```

Note: You only use the `WITH` in the first table of the query. The rest of the tables will begin with a comma (as if in a list).

If this query was written as a nested query, it would look like this:

```
SELECT GENESCOUNT, COUNT(*)
FROM
(SELECT TUMOURID, COUNT(*) AS GENESCOUNT
FROM SIMULACRUM.SIM_AV_GENE
WHERE
EXTRACT(YEAR FROM DATE_OVERALL_TS) IN ('2018','2019')
GROUP BY TUMOURID) TUMOURSWITHGENETEST
GROUP BY GENESCOUNT
ORDER BY GENESCOUNT 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 TUMOURSWITHGENETESTS but could be called a single letter. Multiple tables can be nested within a query.

What is the difference between 'how many cancers had a gene test in 2019' and 'how many cancers diagnosed in 2019 had a gene test'? Can you write an SQL query for the second question? Hint: you can use a `JOIN` to do this.

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 in your dataset, or each row in a section of your dataset. For example, you could number by PATIENTID which would give each patient a different row number (if there were two records for that patient they would have the same number), or you could number by PATIENTID and TUMOURID which would give each tumour that a patient had a different row number. `RANK` is similar but assigns a ranked number to each row, or each row in a partition/section of your dataset. 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 in SQL 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. However, if you're only exploring data then Excel would be preferable. 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 PIVOT and UNPIVOT, 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 an 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.

4. Creating your own tables

Your file space

Each user has a limited amount of space to create their own tables. If you run out of space, you may need to delete old tables you have created and possibly empty your recycle bin.

You can check how full your file space is with the following code:

```
SELECT UTQ.BYTES/1024/1024, UTQ.MAX_BYTES/1024/1024, UTQ.*
FROM USER_TS_QUOTAS UTQ;
SELECT * FROM USER_RECYCLEBIN ORDER BY ORIGINAL_NAME,
OBJECT_NAME;
SELECT US.BYTES/1024/1024 AS BYTES_MB, UT.COMPRESSION,
UT.COMPRESS_FOR, UT.*
FROM USER_TABLES UT
INNER JOIN USER_SEGMENTS US
  ON UT.TABLE_NAME = US.SEGMENT_NAME
ORDER BY US.BYTES/1024/1024 DESC;
```

Creating a table in your file space

To create a table in your own file space using data from the Simulacrum, open a new worksheet for the database you would like the table to be stored. In the tab, write the query to select the data you want from the Simulacrum. 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 TABLE_01 NOLOGGING AS
(SELECT PATIENTID
SIMULACRUM.SIM_AV_TUMOUR    WHERE    EXTRACT(YEAR    FROM
DIAGNOSISDATEBEST) BETWEEN 2018 AND 2019);
```

Note: Always include `NOLOGGING` *as this will speed up your query.*

Clicking run on the query will create a table called TABLE_01. It can be found in the 'Tables' section of the snapshot you ran your query in. (You might need to right click on the 'Tables' tab and press 'Refresh' for your new table to appear). You can then use the data in this table as you would the other main tables.

Deleting a table from your file space

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

```
DROP TABLE TABLE_01 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 access your tables

****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 may need to give another individual user access to your tables. To do this you would use the `GRANT` function. This will allow the user to view and alter your table.

```
GRANT   SELECT,   ALTER   ON   ANALYSISYOURNAME.TABLE_01   TO
ANALYSISOTHERUSER;
```

You can revoke table access using the command below.

```
REVOKE SELECT, ALTER ON ANALYSISYOURNAME.TABLE_01 FROM
ANALYSISOTHERUSER;
```

Importing data into your file space

You may not need to do this too often but if you do instructions are below.

Firstly, you'll need to get your data in the right format. SQL developer will accept csv, excel (.xls or .xlsx) or plain text (.tsv). 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 delimitators (values that separate the fields) that are spaces " ", pipes "|", or semicolons ";".

Right click on 'Tables' in the schema space that you would like you table to appear. Then on Click on Import Data. Click on Browse and find your file and click 'open'.

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.

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.

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.

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 the 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 drop down menu.

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.

If the upload failed: SQL Developer automatically only checks on the first 100 rows (unless you unticked this box during the upload), 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.

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

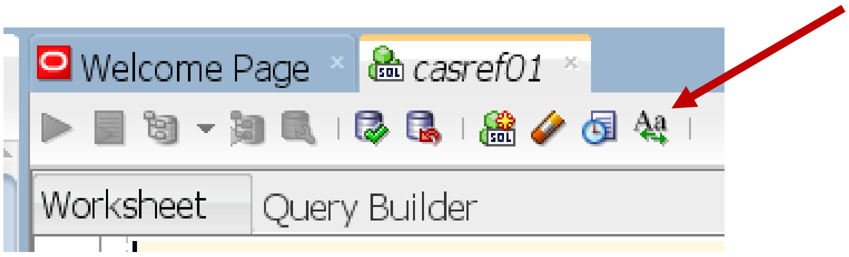
Keep a local file copy of any scratch table data you create so you can re-load it if necessary.

5. Useful tips

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

Capitals

Select the Upper/Lower/InitCap function to the right-hand side of the toolbar to automatically change any text from capital to lower case, or vice versa



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: Jane Smith
DATE: 09.02.2024
*/
-- 1: Selecting variables
```

The 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 worksheet 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.

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).

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.

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.

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

6. Appendix: Results

Below are results to tasks 1-10. The answers below are correct when using Simulacrum v2.1.0 .

1. This section was about exploring the tables and fields.

2.
   i. This section was about exploring the `WHERE` clause.
   ii. This section was about exploring the `WHERE` clause.
   iii. This section was about exploring the `WHERE` clause.
   iv. This section was about exploring the `WHERE` clause.
   v. This section was about exploring the `WHERE` clause.
   vi. This section was about exploring the `WHERE` clause.
   vii. This section was about exploring the `WHERE` clause.
   viii. This section was about exploring the `WHERE` clause.
   ix. This section was about exploring the `WHERE` clause.
   x. Using `AND` retrieves data for anyone aged 34 and is also female. Using `OR` retrieves data for anyone aged 34 (male or female) or is female (any age).
   xi. Use: `WHERE GENDER = 2 AND DIAGNOSISDATEBEST = '2001/01/01';` 311
   xii. Removing `UPPER` should show patients with a death cause code including only lowercase 'c'. There shouldn't be any of these.

3.
   i. 2019.
   ii. 2016
   iii.
```
SELECT EXTRACT(YEAR FROM DIAGNOSISDATEBEST) AS
DIAGNOSISYEAR, DIAGNOSISDATEBEST
FROM SIMULACRUM.SIM_AV_TUMOUR
ORDER BY DIAGNOSISYEAR, DIAGNOSISDATEBEST;
```

4.
   i. 1,995,570
   ii. Use the `SIM_AV_PATIENT` table: 1,871,605
   iii. This is because the first is counting tumours and the second is counting patients. There are more tumours than patients as patients can have more than one cancer diagnosis.

- iv. 1,270,485 + 725,085 = 1,995,570. This total should be the same as the answer above.
- v. Use the `SIM_AV_PATIENT` table: `WHERE VITALSTATUS = 'A'`: 1,232,237
- vi. Use the `SIM_AV_TUMOUR` table: `WHERE AGE >100`: 781
- vii. 1,995,570; 1,995,570; 176,313
- viii. Yes; No.
- ix. Use the `SIM_AV_PATIENT` table: `WHERE DEATHCAUSECODE_UNDERLYING IS NOT NULL`; 623,087; 33.3%

5.
- i. More women
- ii. You should have data for 3 groups (1,2,9) for gender. You should have data for 23 ethnicity groups and a null group D.

6.
- i. Oldest = 107; Use: `WHERE AGE = 107. Lung cancer`; Average = 67.1.
- ii. Use the `MAX` select function
- iii. Use the `AVG` select function and the group by clause; 69 for men vs 65 for women.
- iv. You should have data for years beginning from 2016 to 2019, the number of tumours, the maximum age, and the average age. The average age should remain similar over time.

7.
- i. 63,044; 64,704. Some patients have more than one diagnosis in this group.
- ii. You should have data for the 5 cancer types with the number of patients. Summing these counts gives 63,222.

8. This section was about using case statements.

9.
- i. The query has put every row from the AV_TUMOUR table against every row of the AV_PATIENT table. 1,995,570; 1,871,605; 3,734,918,789,850
- ii. Use `COUNT` in select statement; 1,995570.
- iii. Use `COUNT` in the select statement and `GROUP BY` (BEHAVIOUR_ICD10_O2). You should have data for 7 behaviour codes codes (0-3,5,6 and 9) with the number of tumours for each.

10.

viii. The first is the year of the test, while the second is year of diagnosis. Use:

```
SELECT COUNT (DISTINCT AVT.TUMOURID)
FROM SIMULACRUM.SIM_AV_TUMOUR AVT
LEFT JOIN SIMULACRUM.SIM_AV_GENE AVTG
ON AVT.TUMOURID = AVTG.TUMOURID
WHERE EXTRACT(YEAR FROM AVT.DIAGNOSISYEAR) = '2019'
AND GENE_DESC IS NOT NULL;
= 83698.
```