Assignment 9

Introduction

In this assignment, you will use Amazon Redshift, one of AWS' query engine. It operates on data from an S3 data lake. Redshift is built on a strongly relational model, emphasizing the speed of joins across many tables, but extends that model to support hierarchical data such as JSON objects.

Data warehouses:

1. Store historical data that will rarely be updated, although it may be extended by new data.
2. Are optimized for queries over updates.
3. Are optimized for joins across many tables, categorizing data by ever-finer levels of detail.
4. Often store their data by columns rather than rows.

Data lakes:

1. Store its data on object-stores.
2. Hold a wide variety of data, including structured/semistructured formats (JSON, Parquet) and unstructured formats (e.g., text files).
3. Are read in place by query tools, separating data from computation.

In Part 1, you will first set up a Redshift data warehouse, load it with data, and do some typical analyses. In Part 2, you will use Redshift to analyse external files stored in an S3 data lake and explore the performance tradeoffs.

Setup

Preparing Redshift for your first use requires several steps. This is a common activity for data engineers, defining a data warehouse for analysts to consult. A Redshift cluster is typically shared across a group, department, or entire organization. Data engineers and security administrators define the cluster and once it is running and has the correct permissions, business analysts will then simply log in and analyze their data. These analysts will often not call Redshift directly but instead use business intelligence *(https://www.tableau.com/learn/articles/business-intelligence)* tools such as Tableau *(https://www.tableau.com/)* or Power BI *(https://powerbi.microsoft.com/en-ca/)* that in turn call the cluster's API to query the large tables it stores.

Setup 1: Define IAM roles for access permissions

Security is crucial in cloud services.

Given that cloud services are potentially accessible to anyone, anywhere, establishing security policies is a necessary step in configuring a new service. AWS includes a rich set of tools for both authentication and authorization. When creating a Redshift cluster, we need to establish its authority to:

- Manage the virtual machines that run its queries. This authority is defined by a service-linked role *(https://docs.aws.amazon.com/redshift/latest/mgmt/using-service-linked-roles.html)* , predefined by AWS.
- Access all the S3 objects that will be either loaded into Redshift (Part 1 of this assignment) or directly accessed as a data lake (Part 2). This authority is defined by a service role *(https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_create_for-service.html)* , whose permissions we will define.

Both roles are mechanisms for us to delegate authorization to the Redshift service to perform specified actions on our behalf. When we create the Redshift cluster, we will assign it these roles.

CREATE THE REDSHIFT SERVICE-LINKED ROLE, AWSSERVICEROLEFORREDSHIFT

If you have previously worked with Redshift and already have a role named `AWSServiceRoleForRedshift`, you do not need to do this substep and can skip directly to the `RedshiftS3Full` substep.

1. Sign on to your AWS IAM id and navigate to the IAM console by entering `IAM` in the top search bar, then selecting IAM from the dropdown list.
2. Click "Roles" in the left-hand menu.
3. Click the "Create role" button in the upper right-hand corner.

The role creation wizard has four steps.

1. Page 1:
    a. Select `AWS service` (typically the default).
    b. Select the `Redshift` category and `Redshift` as the use case.
    c. Click the `Next` button at the lower right.
2. Page 2:
    a. You can accept the default policy
    b. Click the `Next` button at the lower right.
3. Page 3:
    a. Click the `Create role` button at the lower right.

CREATE THE SERVICE ROLE, REDSHIFTS3FULL

As with the prior substep, begin at the IAM console and press the `Create Role` button.

1. Page 1:
    a. Select `AWS service` (typically the default).
    b. Select the `Redshift` category and `Redshift Customizable` as the use case. **Note that this differs from the above step.**
    c. Press the `Next` button at the lower right.
2. Page 2:
    a. You should see a list of policies with many entries. Enter `s3` in the filter, which should narrow the list to nine or so.
    b. From the filtered list, click the checkbox for `AmazonS3FullAccess`.
    c. Replace `s3` in the filter bar with `glue`.
    d. From this second filtered list, click the checkbox for `AWSGlueConsoleFullAccess`.
    e. You should have **both** policies selected. Click the `Next` button at the lower right.
3. Page 3:
    a. In the "Role name" text box, enter `RedshiftS3Full`.
    b. Click the `Create role` button.

Setup 2: Create and launch a Redshift cluster

Now that you have created the roles that will authorize Redshift access to your objects, you can create a small Redshift cluster consisting of a single machine.

Your cluster for this assignment will need to be located in us-west-2 to allow the use Redshift Spectrum with the sample data located in this region. (Go to the upper right hand of the web console and switch to us-west-2.)

CREATE A CLUSTER SUBNET GROUP

You may get an error below about not having "cluster subnet groups". If so, follow these instructions to add them *(https://docs.aws.amazon.com/redshift/latest/mgmt/managing-cluster-subnet-group-console.html)* .

CREATE THE CLUSTER

In the AWS console, enter `redshift` in the filter bar at the top of the page and select `Amazon Redshift` from the dropdown list.

1. Select `Clusters` from the left-side menu and click `Create cluster` in the upper right.
2. Click the `Free trial` radio button (if it's available to you).
3. Select the `dc2.large` node type.
4. Check the `Generate a password` box (at the bottom of the page).
5. Click the `Create cluster` button.

Wait until the cluster is listed as "Available".

ASSIGN THE IAM ROLES

The final step in configuring your cluster is to assign it the two IAM roles created in the previous step:

1. Select `CLUSTERS` in the left menu.
2. Check the box to the left of your cluster's name. (Remember that the cluster must first have a status of "Available".)
3. In the `Actions` dropdown menu, select `Manage IAM roles`.
4. In the "Manage IAM roles" page, the "Available IAM roles" dropdown will have two entries: `AWSServiceRoleForRedshift` and `RedshiftS3Full`.
5. Select one of the available roles and click the `Associate IAM role` button to its right.
6. Repeat for the second role.
7. Once both roles are listed as "Associated", click the `Save changes` button.

The cluster list will show your cluster status as `Available Modifying`. In fact, you cannot proceed to the next step until the "Modifying" annotation is gone.

Congratulations—you've created the cluster and assigned it permission to read files from S3. The remaining step is to create some tables.

CONTROLLING YOUR COSTS: PAUSE/RESUME THE CLUSTER

A Redshift cluster comprises a group of virtual machines and persistent storage, storing your tables and running your queries. **As long as the machines are running, your account is being charged, whether or not you are performing queries.**

If you are running the free cluster, costs aren't so much of a concern but bear in mind that it is only free for a preset number of hours, so even in this case it makes sense to minimize the number of hours your cluster is running.

You can pause a Redshift cluster, which stops the accumulation of charges. **When you are ending a Redshift session, pause the cluster and then resume it at the start of your next session.**

To pause your cluster:

1. In the Redshift console, click on `CLUSTERS` in the left-side menu.
2. Locate your cluster in the list (it should be the only one) and select the check box to its left.
3. In the `Actions` menu, select `Pause`.
4. In the next dialogue, the radio button (mid-screen) should be set for `Pause now` (the default). Click the `Pause now` button in the lower right to pause your cluster.

Pausing takes about 5 minutes, depending upon the system load. In the list, the cluster's status will proceed from `Pausing` to `Paused`.

To resume your cluster, follow the same process but select the reverse action in the menu:

1. In the Redshift console, click on `CLUSTERS` in the left-side menu.
2. Locate your cluster in the list (it shoud be the only one) and select the check box to its left.
3. In the `Actions` menu, select `Resume`.
4. In the next dialogue, the radio button (mid-screen) should be set for `Resume now` (the default). Click the `Resume now` button in the lower right to resume your cluster.

Resuming often takes a bit longer than pausing, sometimes up to 10 minutes. In the list, the cluster's status will proceed from `Paused` to `Resuming` to `Available`.

Given how long it takes to pause and resume, you will probably only want to do it when you are leaving for an extended period, not when you're just stepping away to get a snack.

Setup 3: Create Redshift tables

The final setup step is defining the Redshift table schemas and loading the tables. We provide the SQL commands that do this.

Get the files

Download `redshift.zip` from CourSys and unzip the file, creating a directory `redshift-files`.

PREPARING AND PROTECTING YOUR AWS ACCOUNT ID

The commands for creating tables require your AWS Account ID, a 12-digit number that uniquely identifies your account. Although knowing this ID insufficient to allow someone to log in to your account, an attacker can use the account ID *(https://rhinosecuritylabs.com/aws/aws-iam-user-enumeration/)* to learn more about your account, opening it to further attack.

Keep your AWS Account ID private.

Keeping the ID private isn't trivial because some AWS resource names include it. In particular, `IAM_ROLE` names include your Account ID, such as `arn:aws:iam::AWS_ACCOUNT_ID:role/RedShiftS3Full`. The SQL commands we provide are parameterized and you must instantiate them with your AWS Account ID before running. You protect the ID by storing the instantiated SQL statements in a protected directory, `private`.

To instantiate the SQL commands:

1. Make the Redshift files directory your current directory: `cd redshift-files`
2. To get your Account ID, click on your IAM ID displayed in the upper right corner, with a downward triangle next to it. The dropdown menu will include an entry, `My Account`. This should be a 12-digit number. Copy this number.
3. Create the file `private/aws-id.txt` whose only contents are your Account ID. Save the file.

4. In the `redshift-files` directory, run `instantiate-sql.sh` with no arguments. This will instantiate the SQL statements from the template `sql-statements.tpl`, inserting your AWS Account ID to create `private/sql-statements.sql`.

The `private` directory protects your Account ID by restricting its access to the owner (you) and the `.gitignore` setting prevents its files from being added to the repository and potentially pushed to a public location.

CREATING AND LOADING THE TABLES

With all that preparation, now create the actual tables:

1. Resume your Redshift cluster if it is paused.
2. In the Redshift console, hover over `EDITOR` in the lefthand menu and choose "Query editor v2".
3. If you aren't connected automatically, you should be able to connect (the three-dots menu beside the cluster name), and select "temporary credentials", database "dev" and user name "awsuser".
4. Copy the contents of `private/sql-statements.sql` **up to but not including the section labelled `External tables`** and paste them into the `Query 1` pane of the editor.
5. Click the `Run` button at the top left of the editor pane.

The process will run for a minute or two, defining the table schemas and loading them with data.

IF THE TABLES ALL LOADED

Whew! You're done setting up. You might want to stretch and take a break before starting the next section. If you're going to be away for an extended period, consider pausing the cluster.

IF SOME STATEMENTS FAILED

If you are starting from a completely fresh Redshift cluster, these statements should run without error. In the event an error does occur, here are some tips to debug the problem:

1. The query editor runs each submission as a transaction: A statement only has an effect if every statement in its batch successfully completes. If any statement fails, the effects of prior statements in that transaction will be rolled back.

2. If you want to delete everything and start over, the file `redshift-files/maintenance.sql` has every statement required to clear all tables created in this exercise.

3. If there is an error, it is most likely in a path to one of the S3 objects from which tables are loaded or (for `EXTERNAL` tables) that constitute the table itself.

4. If one of the `LOAD` statements fails, run this query

   ```
   SELECT * FROM pg_catalog.stl_load_errors ORDER BY starttime DESC LIMIT 2;
   ```

   to list the last two input rows that had errors. The most likely problem will be that an S3 object is incorrectly named.

Part 1: Redshift as a data warehouse

With the warehouse set up, we can analyse its data. In this section, given three questions that a business analyst might ask of the data, we will derive Redshift SQL queries answering those questions.

Formatting your results: When the assignment request that you put a result in `answers.txt`, we want text, not a screen shot.

Copy and paste the SQL query first, then copy and paste the short table of results from that query. An answer might look as follows:

```
Question X

Query:

SELECT COUNT(*), in_vancouver
  FROM vancouver_custs
  GROUP BY in_vancouver
  ORDER BY in_vancouver;

Results:

count   in_vancouver
6544    0
3456    1
```

Explore the individual tables

Before the formal analyses, let's familiarize ourselves with the available data. The statements that you just executed built five tables:

- `amenities`: Open Street Map *(https://www.openstreetmap.org/#map=15/49.2767/-122.9182)* data showing the locations and attributes of amenities within the greater Vancouver area. For our purposes, an *amenity* is a place where a customer might spend money (actual data, publicly available).
- `customers`: Customers located across the Canadian provinces and territories (synthetic data).
- `greater_vancouver_prefixes`: For each municipality in the Greater Vancouver area, the three-letter postal code prefixes for addresses in that municipality. Note that some prefixes apply to multiple municipalities (actual data, publicly available).
- `paymentmethods`: Credit and debit card data (synthetic data).
- `purchases`: Purchases by customers at Vancouver amenities using a payment method (synthetic data).

Each table has a primary key. With one exception, the data is normalized and tables refer to each other via foreign key references. The lone exception is the `tags` field of the `amenities` table, which contains optional tags selected from a list of 139 keywords.

Explore the `CREATE TABLE` statements and run some SQL `SELECT` queries to learn the table contents and their interrelationships. Here are some links to useful references:

- Redshift `SELECT` statement *(https://docs.aws.amazon.com/redshift/latest/dg/r_SELECT_synopsis.html)*
- Redshift SQL functions *(https://docs.aws.amazon.com/redshift/latest/dg/c_SQL_functions.html)*
- Complete Redshift SQL Reference Guide *(https://docs.aws.amazon.com/redshift/latest/dg/cm_chap_SQLCommandRef.html)*

Some queries you might use to familiarize yourself with the tables. Do only as many as you find useful—you will not submit your answers:

- How many customers, purchases, payment methods, and amenities are there?
- What are the types of amenities (`amenity`)?
- What is the range of the purchase dates?
- What are the types (`mtype`) of payment methods?
- How many customers are there from each province and territory?
- What are the average and median purchase amounts?
- What's in `greater_vancouver_prefixes`?

Once you're comfortable working with the tables, proceed to the questions.

QUESTION 1: DEBIT OR CREDIT?

Our first question:

> Do people from Ontario tend to put larger purchases on one payment method?

To answer this, write a query that returns the average purchase amounts that customers from Ontario (province code `ON`) made with credit and debit cards.

In your `answers.txt`, answer this question as Question 1 by organizing the two components:

a. Prose answer to the question

b. Your query used to arrive at this answer

QUESTION 2: WHO SPENDS MORE OVERALL?

Our second question will require more work to answer:

> Consider the three groups of people: people who live in the Vancouver region, visitors from other BC areas, and visitors from outside BC altogether. Which group spent the most per transaction?

To answer this, write a query that returns the following table,

```
From_BC_non_Van From_Van Count Average  Median
...
```

where

- `From_BC_non_Van` is `true` for purchases made by customers living in BC outside Vancouver, `false` for customers from Vancouver or outside BC.
- `From_Van` is `true` for purchases made by customers living in a postal code from the Greater Vancouver region, `false` otherwise.
- `Count` is the number of purchases from customers in this region.
- `Average` is the average amount of these purchases.
- `Median` is the median amount of these purchases.

The table should be ordered by **increasing value of the median.**

With a little thought, you can see that this table will have exactly three rows (which of the four combinations of `From_BC_non_Van` and `From_Van` is impossible?).

The trick for this query is determining which customers live in Greater Vancouver. The `greater_vancouver_prefixes` table has the raw information but its format is inconvenient, listing only the first three letters of postal codes in Greater Vancouver. We want instead a table that indicates whether a given customer lives in the region. We can achieve that effect by creating a view.

The skeleton SQL for creating such a view is

```
CREATE VIEW vancouver_custs AS
WITH
  vprefixes (vp) AS
    (SELECT DISTINCT pcprefix FROM greater_vancouver_prefixes)
SELECT ...
```

The `WITH` clause *(https://docs.aws.amazon.com/redshift/latest/dg/r_WITH_clause.html)* prefixing the `SELECT` creates a temporary table `vprefixes` with a single column `vp` listing all the distinct postal code prefixes in Greater Vancouver. The temporary table can only be used in

the immediately following `SELECT`.

Complete the `SELECT` statement, defining two columns for the `vancouver_custs` view:

- `custid`: The customer ID
- `in_vancouver`: 0 for customers living outside Greater Vancover, 1 for those living in the region

Hint: Use a scalar subquery *(https://docs.aws.amazon.com/redshift/latest/dg/r_scalar_subqueries.html)* with `COUNT` as its field. You should be able to run the query shown above as our example "Question X" and get the same result.

Once you have this view, the `SELECT` statement producing the required table is straightforward.

Answer the question in one sentence and include the statements/results you used to determine this as Question 2.

Answer this question as Question 2 with the three components:

a. An answer to the original question in prose

b. A SQL statement to create the required view

c. A SQL query to support your answer for component a

Querying a semistructured field

Redshift supports semistructured data via its `SUPER` datatype *(https://docs.aws.amazon.com/redshift/latest/dg/r_SUPER_type.html)* . The name is shortened form of "superset" and, as the name implies, a column with this type can contain arbitrary data.

The only column with a `SUPER` type in our system is `amenities.tags`. This column contains an arbitrary collection of field:value pairs, varying from row to row. For example, some amenities have a field, `addr_city`, naming the city in which they are located, but most do not.

You refer to a field as though it were a subcolumn within `tags`, such as `tags.addr_city`. If a field is absent from a row, the value is considered `NULL`:

```
SELECT COUNT(*) FROM amenities WHERE tags.addr_city IS NOT NULL;
```

There are two catches to using field names in queries:

1. Because the field names are not fixed, the parser will not warn you if you misspell one (`tags.add_city`) or refer to one that doesn't exist (`tags.hsduwytusadgha`); you will simply get `NULL` for every row.
2. If your `SELECT` includes both an aggregate function, such as `COUNT`, and a column specifying a field, there are restrictions on the `GROUP BY` clause. We will not use this feature in this exercise.

QUESTION 3: WHO SPENDS MORE ON SUSHI?

Our third question:

> Who spends more at restaurants that serve sushi: locals (residents of Greater Vancouver) or tourists?

To answer this, write a query returning a two-row table with the following two columns:

`avg in_vancouver`

ordered by `in_vancouver`.

Tips:

1. For this simple case, we will only consider restaurants that have a `cuisine` tag and ignore those that don't.
2. You only want to select purchases at *restaurants*. There are non-restaurants in Vancouver that serve sushi.
3. The value in the `cuisine` field might list multiple styles, such as `sushi;udon;sashimi`. You will want to do a partial match using the `ILIKE` operator *(https://docs.aws.amazon.com/redshift/latest/dg/r_patternmatching_condition_like.html)* .
4. The query is simplified if you preface it by a `WITH` clause creating a temporary table `sushi`, whose single column lists the `amenids` of restaurants that have `sushi` in their `cuisine` tag.
5. Take care to spell `cuisine` correctly, otherwise the `sushi` temporary table will be empty.
6. You will also need the `vancouver_custs` view created for Question 2.

Answer the question in one sentence and include the statements/results you used to determine this as Question 3.

Answer this question as Question 3 with the two components:

a. An answer to the original question in prose

b. A SQL query to support your answer for component a

Part 2: Redshift queries to an S3 data lake

In the previous section, we used Redshift to operate on data within the 'data warehouse'-side of the data lake: analysing data that have been cleaned, normalized, and loaded into tidy tables. This represents only a fraction of the data held by typical organizations. The flexibility of a data lake is its ability to operate on data in their original form. I.e., data that have not been cleaned, normalized or loaded and which often come with an irregular structure.

Redshift provides an extension, Redshift Spectrum *(https://aws.amazon.com/blogs/big-data/amazon-redshift-spectrum-extends-data-warehousing-out-to-exabytes-no-loading-required/)* , that allows it to query such data directly without first loading it into Redshift's controlled, relational model. By separating computation from storage, we avoid duplication of data and permit many different systems to access it. For example, an S3 object could be queried by Spectrum and also read by an EMR Spark job.

In AWS, an S3 bucket is a data lake. For this assignment, we will read files directly from S3. You still need to define a Redshift schema for an object in the data lake via a `CREATE EXTERNAL TABLE' statement, specifying a column structure and the location of the object in S3.

Controlling your costs: Spectrum

Where Redshift uses an instance-based billing approach—you pay for every hour your cluster exists, whether you are running queries on it or not—Spectrum uses service-based billing. That is, the cost of a Spectrum query is driven by the data read and the computation consumed.

Each query made to an external tables on S3 is a Spectrum query and is charged. To be clear, Spectrum is an overlay on top of a Redshift cluster. That is, you require an active Redshift cluster to use Spectrum. Therefore, you will want to pause your cluster when you're not running any queries (either standard Redshift query or Spectrum query).

Creating an external schema and two external tables

To create the Redshift descriptors linked to the external S3 objects:

1. Copy the `CREATE EXTERNAL SCHEMA` statement from `redshift-files/private/sql-statements.sql`, paste it into the editor, and run it.
2. Copy each of the two `CREATE EXTERNAL TABLE` statements separately (from the same file) and paste them into the editor before running them.
3. Copy the entire contents of `redshift-files/alter.sql` (which is a single long `ALTER TABLE` statement) into the editor and run it.

Question 4: Average purchase per day for the first five days?

Our final question:

> What was the average purchase per day for the first five days of August?

To answer this, write a query that returns the following two-column, five-line table:

```
pdate      avg
2021-08-01 ...
2021-08-02 ...
2021-08-03 ...
2021-08-04 ...
2021-08-05 ...
```

Order the table by increasing date. Use the Redshift `DATE_PART` function *(https://docs.aws.amazon.com/redshift/latest/dg/r_DATE_PART_function.html)* to extract the month and day from the `pdate` field.

We're going to use this query to highlight the massive parallelism potentially available to Spectrum when it reads tables directly from the S3 data lake.

DATA READ BY REDSHIFT FROM AN INTERNAL TABLE

Run the query first as a regular Redshift query on the same `purchases` table you used in Part 1. The results themselves aren't as interesting as how the query is executed by the two query engines.

From the Redshift main page, select "Queries and loads", sort queries by decreasing start time, find the previous query, and select the "query plan" tab. The only part of the plan of interest to us is the bottom row, "Seq Scan on purchases". From that row, note the number of bytes and number of rows that Redshift scanned.

DATA READ BY SPECTRUM FROM AN EXTERNAL TABLE ON S3

Now run the same query, but replace `FROM purchases` with `FROM s3ext.purchases`. This routes the query outside Redshift to the pool of available Spectrum instances, which will read the data from S3 and do most of the processing.

The query result will be identical—the two `purchases` tables have exactly the same data (though in different location). However, if you look at the query plan for this query, the last row will now read "S3 Query Scan purchases" and will only show 5 rows and 120 bytes. This is the *results* of the scan, passed back to Redshift by Spectrum.

It should be obvious the real work for this query was not performed by Redshift but rather Spectrum. And to see how much data Spectrum read , execute the following query of Redshift's internal management table, `pg_catalog.svl_s3query_summary`:

```
SELECT
    external_table_name,
        s3_scanned_bytes,
        s3_scanned_rows,
        avg_request_parallelism,
        max_request_parallelism
  FROM pg_catalog.svl_s3query_summary
  ORDER BY starttime DESC
  LIMIT 1;
```

First note the number of bytes and number of rows Spectrum read.

Also consider the content of the S3 bucket. Navigate your browser to s3://sfu-cmpt-732-redshift *(https://s3.console.aws.amazon.com/s3/buckets/sfu-cmpt-732-redshift?region=us-west-2&tab=objects)* .

How do these values compare with the values for Redshift's query of the same data? What might explain the difference?

The second thing to note is the two parallelism values. The external purchase table is laid out so that *every day can be processed in parallel by a separate Spectrum instance*. To see a more extreme case, rerun the Spectrum query for all 31 days of the month. Now rerun the query on `svl_s3query_summary` but with `LIMIT 2`, so you can compare the statistics for the 5-day and 31-day queries. What was the maximum parallelism you got on this larger query?

Answer this question as Question 4 with the seven components:

a. An answer to the original question in prose

b. A SQL query for Redshift to support your answer for component a

c. What was the bytes / record ratio for Redshift on the 5-day query?

d. What was the bytes / record ratio for Spectrum on the 5-day query?

e. For this purchase dataset, the averages are 57 bytes/line and 968 lines/day. (It may be useful to explore the `sfu-cmpt-732-redshift` bucket *(https://s3.console.aws.amazon.com/s3/buckets/sfu-cmpt-732-redshift?region=us-west-2&tab=objects)* to derive these for yourself.) From these values, what might you infer about how Redshift scans the table? How Spectrum scans the table?

f. Based on these computations and the parallelism results, what properties of a dataset might make it well-suited to loading from S3 into Redshift before querying it?

g. Conversely, what properties of a dataset might make it well-suited to retaining in S3 and querying it using Spectrum?

Cleanup

When you are done the session, remember to Pause the Redshift cluster.

When you are confident that you will not be returning to this exercise, **delete** the Redshift cluster.

Submission

Submit your `answers.txt` file to Assignment 9.

Updated Tue Oct. 31 2023, 17:30 by ggbaker.