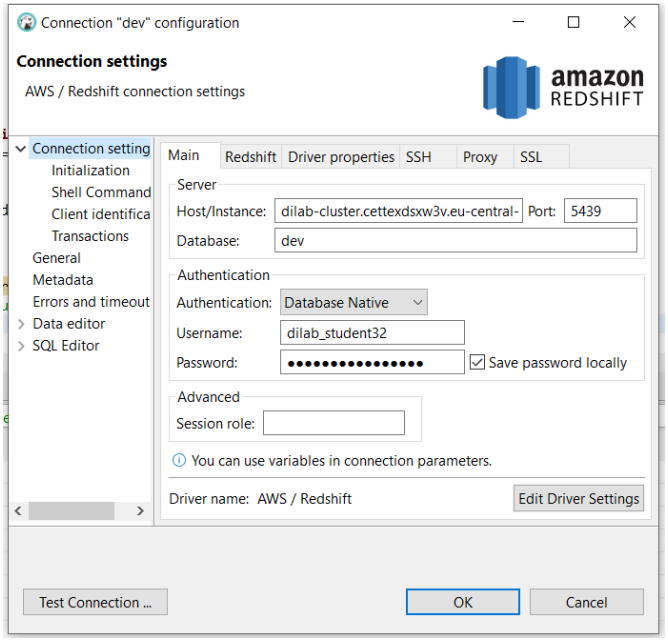
Task_03. Redshift

1. Login to Redshift

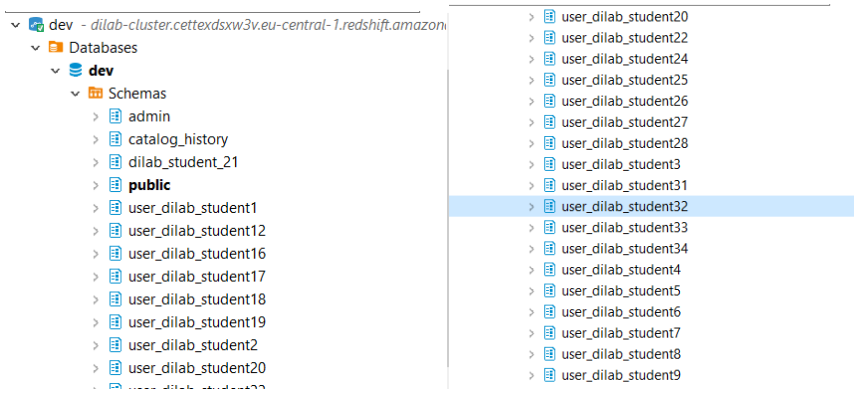
To connect DBeaver to redshift was needed to install driver (no screenshot)

Redshift logged in using login dilab_student32 and password Dilab_student_32 with DBeaver.



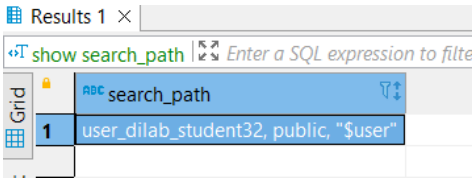
Create schema user_dialab_student32.

```
create schema if not exists user_dilab_student32;
```



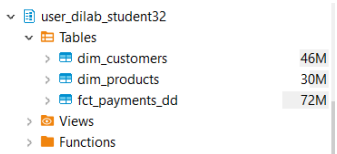
Change search_pass for the user

```
alter user dilab_student32 set search_path to
'$user', 'user_dilab_student32','public';
```



2. Load the tables from S3

Create fact table, dim_products and dim_customers.



Data from S3 bucket was copied for all this tables

```
--copy data from S3
copy user_dilab_student32.fct_payments_dd
from 's3://hanna-yaruk/online_shop/bl_dm/fct_payments/fct_payments.csv'
credentials
'aws_iam_role=arn:aws:iam::            :role/dilab-redshift-role'
region 'eu-central-1'
delimiter ','
csv
DATEFORMAT AS 'auto'
IGNOREHEADER 1;

copy user_dilab_student32.dim_customers
from 's3://hanna-yaruk/online_shop/bl_dm/customers/customers.csv'
credentials
'aws_iam_role=arn:aws:iam::            :role/dilab-redshift-role'
region 'eu-central-1'
delimiter ','
csv
DATEFORMAT AS 'auto'
IGNOREHEADER 1;

copy user_dilab_student32.dim_products
from 's3://hanna-yaruk/online_shop/bl_dm/products/products.csv'
credentials
'aws_iam_role=arn:aws:iam::            :role/dilab-redshift-role'
region 'eu-central-1'
delimiter ','
csv
DATEFORMAT AS 'auto'
IGNOREHEADER 1;
```

Name	Value
Updated Rows	0
Query	copy user_dilab_student32.fct_payments_dd from 's3://hanna-yaruk/online_shop/bl_dm/fct_payments/fct_payments.csv' credentials 'aws_iam_role=arn:aws:iam:: role/dilab-redshift-role' region 'eu-central-1' delimiter ',' csv DATEFORMAT AS 'auto' IGNOREHEADER 1
Finish time	Sun Apr 03 22:59:26 MSK 2022

Name	Value
Updated Rows	0
Query	copy user_dilab_student32.dim_customers from 's3://hanna-yaruk/online_shop/bl_dm/customers/customers.csv' credentials 'aws_iam_role=arn:aws:iam:: role/dilab-redshift-role' region 'eu-central-1' delimiter ',' csv DATEFORMAT AS 'auto' IGNOREHEADER 1
Finish time	Sun Apr 03 23:02:29 MSK 2022

Name	Value
Updated Rows	0
Query	copy user_dilab_student32.dim_products from 's3://hanna-yaruk/online_shop/bl_dm/products/products.csv' credentials 'aws_iam_role=arn:aws:iam:: :role/dilab-redshift-role' region 'eu-central-1' delimiter ',' csv DATEFORMAT AS 'auto' IGNOREHEADER 1
Finish time	Sun Apr 03 23:05:32 MSK 2022

Count the number of rows in the tables

```
select count(*) as fct from fct_payments_dd;
select count(*) as cust from dim_customers;
select count(*) as prod from dim_products;
```

	fct
1	1,000,000

	cust
1	2,000

	prod
1	70

Check the initial compression types, distribution style, sort keys.

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'fct_payments_dd';

select "column", type, encoding, distkey, sortkey
from pg_table_def where schemaname = 'user_dilab_student32' and tablename = 'dim_customers';

select "column", type, encoding, distkey, sortkey
from pg_table_def where schemaname = 'user_dilab_student32' and tablename = 'dim_products';
```

pg_table_def 1

select "column", type, encoding, distkey, sortkey frc | Enter a SQL expression to filter results (use Ct

	column	type	encoding	distkey	sortkey
1	payment_src_id	integer	az64	[ ]	0
2	product_dm_surr_id	integer	az64	[ ]	0
3	employee_dm_surr_id	integer	az64	[ ]	0
4	warehouse_dm_surr_id	integer	az64	[ ]	0
5	discount_dm_surr_id	integer	az64	[ ]	0
6	address_dm_surr_id	integer	az64	[ ]	0
7	customer_dm_surr_id	integer	az64	[ ]	0
8	payment_type_dm_surr_id	integer	az64	[ ]	0
9	product_cost	integer	az64	[ ]	0
10	product_price	integer	az64	[ ]	0
11	quantity	integer	az64	[ ]	0
12	sale_date	date	az64	[ ]	0
13	day_id	date	az64	[ ]	0
14	update_dt	date	az64	[ ]	0

	column	type	encoding	distkey	sortkey
1	customer_dm_surr_id	integer	az64	[ ]	0
2	customer_id	integer	az64	[ ]	0
3	source_system	character varying(256)	lzo	[ ]	0
4	source_entity	character varying(256)	lzo	[ ]	0
5	personal_id	character varying(256)	lzo	[ ]	0
6	first_name	character varying(256)	lzo	[ ]	0
7	last_name	character varying(256)	lzo	[ ]	0
8	email	character varying(256)	lzo	[ ]	0
9	address_id	integer	az64	[ ]	0
10	address	character varying(256)	lzo	[ ]	0
11	city_id	integer	az64	[ ]	0
12	city	character varying(256)	lzo	[ ]	0
13	country_id	integer	az64	[ ]	0
14	country_code	character varying(256)	lzo	[ ]	0
15	country_name	character varying(256)	lzo	[ ]	0
16	region_id	integer	az64	[ ]	0
17	region	character varying(256)	lzo	[ ]	0
18	postal_code	character varying(256)	lzo	[ ]	0
19	phone	character varying(256)	lzo	[ ]	0
20	update_dt	date	az64	[ ]	0

select "column", type, encoding, distkey, sortkey frc | Enter a SQL expression to filter results (use Ctrl+

	column	type	encoding	distkey	sortkey
1	product_dm_surr_id	integer	az64	[ ]	0
2	product_id	integer	az64	[ ]	0
3	source_system	character varying(256)	lzo	[ ]	0
4	source_entity	character varying(256)	lzo	[ ]	0
5	product_name	character varying(256)	lzo	[ ]	0
6	product_desc	character varying(256)	lzo	[ ]	0
7	category_id	integer	az64	[ ]	0
8	category_name	character varying(256)	lzo	[ ]	0
9	product_cost	integer	az64	[ ]	0
10	product_price	integer	az64	[ ]	0
11	total_unit	integer	az64	[ ]	0
12	update_dt	date	az64	[ ]	0

Check distribution style of the tables

```
select "schema", "table", diststyle from SVV_TABLE_INFO
where "schema"='user_dilab_student32' and ("table" = 'fct_payments_dd' or "table" = 'dim_customers' or "table"
= 'dim_products');
```

Results 1

select "schema", "table", diststyle from SVV_TABLE_INFO w | Enter a SQL expressi

	schema	table	diststyle
1	user_dilab_student32	fct_payments_dd	AUTO(EVEN)
2	user_dilab_student32	dim_customers	AUTO(ALL)
3	user_dilab_student32	dim_products	AUTO(ALL)

Initial data were automatically compressed while taking them from csv. There are no distribution or sort keys in the tables. Distribution for small (dim_customers, dim_products) is ALL, for large table (fct_payments_dd) – EVEN.

3. Compression

Check the compression of the tables (previous step)

pg_table_def 1

select "column", type, encoding, distkey, sortkey frc

	column	type	encoding	distkey	sortkey
1	payment_src_id	integer	az64	[ ]	0
2	product_dm_surr_id	integer	az64	[ ]	0
3	employee_dm_surr_id	integer	az64	[ ]	0
4	warehouse_dm_surr_id	integer	az64	[ ]	0
5	discount_dm_surr_id	integer	az64	[ ]	0
6	address_dm_surr_id	integer	az64	[ ]	0
7	customer_dm_surr_id	integer	az64	[ ]	0
8	payment_type_dm_surr_id	integer	az64	[ ]	0
9	product_cost	integer	az64	[ ]	0
10	product_price	integer	az64	[ ]	0
11	quantity	integer	az64	[ ]	0
12	sale_date	date	az64	[ ]	0
13	day_id	date	az64	[ ]	0
14	update_dt	date	az64	[ ]	0

select "column", type, encoding, distkey, sortkey frc

	column	type	encoding	distkey	sortkey
1	product_dm_surr_id	integer	az64	[ ]	0
2	product_id	integer	az64	[ ]	0
3	source_system	character varying(256)	lzo	[ ]	0
4	source_entity	character varying(256)	lzo	[ ]	0
5	product_name	character varying(256)	lzo	[ ]	0
6	product_desc	character varying(256)	lzo	[ ]	0
7	category_id	integer	az64	[ ]	0
8	category_name	character varying(256)	lzo	[ ]	0
9	product_cost	integer	az64	[ ]	0
10	product_price	integer	az64	[ ]	0
11	total_unit	integer	az64	[ ]	0
12	update_dt	date	az64	[ ]	0

Create tables (products and fct_payments) without compression. For this we use *encode raw* for each column and *compupdate off* in copying data.

```sql
--create tables without compression
CREATE TABLE dim_products_withoutcomp (
    product_dm_surr_id integer not null encode RAW,
    product_id       integer not null encode RAW,
    source_system    VARCHAR(256) not null encode RAW,
    source_entity    VARCHAR(256) NOT null encode RAW,
    product_name     VARCHAR(256) NOT null encode RAW,
    product_desc     VARCHAR(256) NOT null encode RAW,
    category_id      integer not null encode RAW,
    category_name    VARCHAR(256) NOT null encode RAW,
    product_cost     integer not null encode RAW,
    product_price    integer not null encode RAW,
    total_unit       integer not null encode RAW,
    update_dt        DATE not null encode RAW
);

CREATE TABLE user_dilab_student32.fct_payments_dd_withoutcomp (
    payment_src_id integer NOT NULL,
    product_dm_surr_id integer NOT null encode RAW,
    employee_dm_surr_id integer NOT null encode RAW,
    warehouse_dm_surr_id integer NOT null encode RAW,
    discount_dm_surr_id integer NOT null encode RAW,
    address_dm_surr_id integer NOT null encode RAW,
    customer_dm_surr_id integer NOT null encode RAW,
    payment_type_dm_surr_id integer NOT null encode RAW,
    product_cost integer NOT null encode RAW,
    product_price integer NOT null encode RAW,
    quantity integer NOT null encode RAW,
    sale_date DATE NOT null encode RAW,
    day_id DATE NOT null encode RAW,
    update_dt DATE NOT null encode RAW
);
```

```sql
--copy data from S3
copy user_dilab_student32.dim_products_withoutcomp
from 's3://hanna-yaruk/online_shop/bl_dm/products/products.csv'
credentials
'aws_iam_role=arn:aws:iam::            :role/dilab-redshift-role'
region 'eu-central-1'
delimiter ','
csv
DATEFORMAT AS 'auto'
IGNOREHEADER 1
compupdate off;

copy user_dilab_student32.fct_payments_dd_withoutcomp
from 's3://hanna-yaruk/online_shop/bl_dm/fct_payments/fct_payments.csv'
credentials
'aws_iam_role=arn:aws:iam::            :role/dilab-redshift-role'
region 'eu-central-1'
delimiter ','
csv
DATEFORMAT AS 'auto'
IGNOREHEADER 1
compupdate off;
```

Check compression types, distribution keys, sort keys

```sql
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'dim_products_withoutcomp';

select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'fct_payments_dd_withoutcomp';
```

pg_table_def 1

select "column", type, encoding, distkey, sortkey fro

	column	type	encoding	distkey	sortkey
1	product_dm_surr_id	integer	none	[ ]	0
2	product_id	integer	none	[ ]	0
3	source_system	character varying(256)	none	[ ]	0
4	source_entity	character varying(256)	none	[ ]	0
5	product_name	character varying(256)	none	[ ]	0
6	product_desc	character varying(256)	none	[ ]	0
7	category_id	integer	none	[ ]	0
8	category_name	character varying(256)	none	[ ]	0
9	product_cost	integer	none	[ ]	0
10	product_price	integer	none	[ ]	0
11	total_unit	integer	none	[ ]	0
12	update_dt	date	none	[ ]	0

	column	type	encoding	distkey	sortkey
1	payment_src_id	integer	az64	[ ]	0
2	product_dm_surr_id	integer	none	[ ]	0
3	employee_dm_surr_id	integer	none	[ ]	0
4	warehouse_dm_surr_id	integer	none	[ ]	0
5	discount_dm_surr_id	integer	none	[ ]	0
6	address_dm_surr_id	integer	none	[ ]	0
7	customer_dm_surr_id	integer	none	[ ]	0
8	payment_type_dm_surr_id	integer	none	[ ]	0
9	product_cost	integer	none	[ ]	0
10	product_price	integer	none	[ ]	0
11	quantity	integer	none	[ ]	0
12	sale_date	date	none	[ ]	0
13	day_id	date	none	[ ]	0
14	update_dt	date	none	[ ]	0

Check distribution style of the tables

```
select "schema", "table", diststyle from SVV_TABLE_INFO
where "schema"='user_dilab_student32' and ("table" = 'dim_products_withoutcomp' or "table" =
'fct_payments_dd_withoutcomp');
```

select "schema", "table", diststyle from SVV_TABLE_INFO w

	schema	table	diststyle
1	user_dilab_student32	dim_products_withoutcomp	AUTO(ALL)
2	user_dilab_student32	fct_payments_dd_withoutcomp	AUTO(EVEN)

To identify best compression methods suggested by Redshift use analyze command

```
--analyze compression
select count(*) as fct from fct_payments_dd_withoutcomp;
Analyze compression fct_payments_dd_withoutcomp
comprows 1000000;
```

Results 1

Analyze compression fct_payments_dd_withoutcomp com

	Table	Column	Encoding	Est_reduction_pct
1	fct_payments_dd_withoutcomp	payment_src_id	az64	0.00
2	fct_payments_dd_withoutcomp	product_dm_surr_id	az64	75.20
3	fct_payments_dd_withoutcomp	employee_dm_surr_id	az64	78.13
4	fct_payments_dd_withoutcomp	warehouse_dm_surr_id	az64	87.50
5	fct_payments_dd_withoutcomp	discount_dm_surr_id	az64	87.50
6	fct_payments_dd_withoutcomp	address_dm_surr_id	zstd	85.80
7	fct_payments_dd_withoutcomp	customer_dm_surr_id	az64	62.50
8	fct_payments_dd_withoutcomp	payment_type_dm_surr_id	az64	84.38
9	fct_payments_dd_withoutcomp	product_cost	bytedict	74.98
10	fct_payments_dd_withoutcomp	product_price	bytedict	74.98
11	fct_payments_dd_withoutcomp	quantity	az64	87.50
12	fct_payments_dd_withoutcomp	sale_date	az64	77.59
13	fct_payments_dd_withoutcomp	day_id	az64	77.59
14	fct_payments_dd_withoutcomp	update_dt	az64	99.99

```
select count(*) as prod from dim_products_withoutcomp;
Analyze compression dim_products_withoutcomp
comprows 70;
```

Results 1

Analyze compression dim_products comprows 70

	Table	Column	Encoding	Est_reduction_pct
1	dim_products	product_dm_surr_id	raw	0.00
2	dim_products	product_id	raw	0.00
3	dim_products	source_system	raw	0.00
4	dim_products	source_entity	raw	0.00
5	dim_products	product_name	raw	0.00
6	dim_products	product_desc	raw	0.00
7	dim_products	category_id	raw	0.00
8	dim_products	category_name	raw	0.00
9	dim_products	product_cost	raw	0.00
10	dim_products	product_price	raw	0.00
11	dim_products	total_unit	raw	0.00
12	dim_products	update_dt	raw	0.00

As we see, it’s no sense to use compression on small tables like products. So, we will analyze fact table.

Create table *fct_payments_dd_analyzedcomp*

```sql
--create table fct_payments_dd_analyzedcomp
CREATE TABLE fct_payments_dd_analyzedcomp (
    payment_src_id integer NOT null encode AZ64,
    product_dm_surr_id integer NOT null encode AZ64,
    employee_dm_surr_id integer NOT null encode AZ64,
    warehouse_dm_surr_id integer NOT null encode AZ64,
    discount_dm_surr_id integer NOT null encode AZ64,
    address_dm_surr_id integer NOT null encode ZSTD,
    customer_dm_surr_id integer NOT null encode AZ64,
    payment_type_dm_surr_id integer NOT null encode AZ64,
    product_cost integer NOT null encode bytedict,
    product_price integer NOT null encode bytedict,
    quantity integer NOT null encode AZ64,
    sale_date DATE NOT null encode AZ64,
    day_id DATE NOT null encode AZ64,
    update_dt DATE NOT null encode AZ64
);

--insert data
copy user_dilab_student32.fct_payments_dd_analyzedcomp
from 's3://hanna-yaruk/online_shop/bl_dm/fct_payments/fct_payments.csv'
credentials
'aws_iam_role=arn:aws:iam::            :role/dilab-redshift-role'
region 'eu-central-1'
delimiter ','
csv
DATEFORMAT AS 'auto'
IGNOREHEADER 1;

select * from fct_payments_dd_analyzedcomp limit 10;
```

Check compression type in *fct_payments_dd_analyzedcomp* table

```sql
--check compression type
select "column", type, encoding
from pg_table_def where tablename = 'fct_payments_dd_analyzedcomp';
```

pg_table_def 1

select "column", type, encoding from pg_table_def where · Enter a SQL expression to filter results (use

	column	type	encoding
1	payment_src_id	integer	az64
2	product_dm_surr_id	integer	az64
3	employee_dm_surr_id	integer	az64
4	warehouse_dm_surr_id	integer	az64
5	discount_dm_surr_id	integer	az64
6	address_dm_surr_id	integer	zstd
7	customer_dm_surr_id	integer	az64
8	payment_type_dm_surr_id	integer	az64
9	product_cost	integer	bytedict
10	product_price	integer	bytedict
11	quantity	integer	az64
12	sale_date	date	az64
13	day_id	date	az64
14	update_dt	date	az64

Compare the size of the tables - compressed, decompressed and default compressed

```sql
with table_defaultcomp as (
select
  TRIM(name) as table_name,
  TRIM(pg_attribute.attname) AS column_name,
  COUNT(1) AS size
FROM
  svv_diskusage JOIN pg_attribute ON
    svv_diskusage.col = pg_attribute.attnum-1 AND
    svv_diskusage.tbl = pg_attribute.attrelid
where table_name = 'fct_payments_dd'
GROUP BY 1, 2),
table_withoutcomp as (
select
  TRIM(name) as table_name,
  TRIM(pg_attribute.attname) AS column_name,
  COUNT(1) AS size
FROM
  svv_diskusage JOIN pg_attribute ON
    svv_diskusage.col = pg_attribute.attnum-1 AND
    svv_diskusage.tbl = pg_attribute.attrelid
where table_name = 'fct_payments_dd_withoutcomp'
GROUP BY 1, 2),
table_analyzedcomp as (
select
  TRIM(name) as table_name,
  TRIM(pg_attribute.attname) AS column_name,
  COUNT(1) AS size
FROM
  svv_diskusage JOIN pg_attribute ON
    svv_diskusage.col = pg_attribute.attnum-1 AND
    svv_diskusage.tbl = pg_attribute.attrelid
where table_name = 'fct_payments_dd_analyzedcomp'
GROUP BY 1, 2)
select d.column_name,
d."size" as sizemb_default,
w."size" as sizemb_raw,
a."size" as sizemb_analyzed
from table_defaultcomp d
left join table_withoutcomp w
on d.column_name = w.column_name
left join table_analyzedcomp a
on d.column_name = a.column_name
order by d.column_name;
```

Results 1

with table_defaultcomp as (select TRIM(name) as table_n

Enter a SQL expression to filter results (use Ctrl-

	column_name	sizemb_default	sizemb_raw	sizemb_analyzed
1	address_dm_surr_id	4	4	4
2	customer_dm_surr_id	4	4	4
3	day_id	4	4	4
4	discount_dm_surr_id	4	4	4
5	employee_dm_surr_id	4	4	4
6	payment_src_id	4	4	4
7	payment_type_dm_surr_id	4	4	4
8	product_cost	4	4	4
9	product_dm_surr_id	4	4	4
10	product_price	4	4	4
11	quantity	4	4	4
12	sale_date	4	4	4
13	update_dt	4	4	4
14	warehouse_dm_surr_id	4	4	4

The size of compressed or decompressed tables does not differ. In this direct case we can not use compression at all to improve the result.

4. A stored procedure to load the report

Turn off the result caching

```
ALTER USER dilab_student32 SET enable_result_cache_for_session TO off;
```

Name	Value
Updated Rows	0
Query	ALTER USER dilab_student32 SET enable_result_cache_for_session TO off
Finish time	Mon Apr 04 19:30:05 MSK 2022

```
SELECT user, current_user_id;
```

SELECT user, current_user_id | Enter a SQL expre

	current_user	current_user_id
1	dilab_student32	118

Check the activity of the user

```
select * from svl_qlog
where userid = 118
ORDER BY starttime DESC;
```

Start the execution of the select statement for the procedure

```
explain
select personal_id, first_name, last_name, dp.product_name, dp.product_price, sale_date
from fct_payments_dd fpd
left join dim_customers dc on dc.customer_dm_surr_id = fpd.customer_dm_surr_id
left join dim_products dp on dp.product_dm_surr_id = fpd.product_dm_surr_id
where extract(year from sale_date) = '2022' and extract(month from sale_date) = '01' and dp.product_price >
3000
order by dp.product_price, dp.product_name, sale_date;
```

The execution plan is next one:

explain select personal_id, first_name, last_name, dp.produ | Enter a SQL expression to filter results (use Ctrl+Space)

	QUERY PLAN
1	XN Merge (cost=1000000020026.93..1000000020026.94 rows=6 width=92)
2	Merge Key: dp.product_price, dp.product_name, fpd.sale_date
3	-> XN Network (cost=1000000020026.93..1000000020026.94 rows=6 width=92)
4	Send to leader
5	-> XN Sort (cost=1000000020026.93..1000000020026.94 rows=6 width=92)
6	Sort Key: dp.product_price, dp.product_name, fpd.sale_date
7	-> XN Hash Join DS_DIST_ALL_NONE (cost=25.91..20026.85 rows=6 width=92)
8	Hash Cond: ("outer".product_dm_surr_id = "inner".product_dm_surr_id)
9	-> XN Hash Left Join DS_DIST_ALL_NONE (cost=25.00..20025.56 rows=25 width=67)
10	Hash Cond: ("outer".customer_dm_surr_id = "inner".customer_dm_surr_id)
11	-> XN Seq Scan on fct_payments_dd fpd (cost=0.00..20000.00 rows=25 width=12)
12	Filter: ((("date_part"('month'::text, sale_date) = 1) AND ("date_part"('year'::text, sale_date) = 2022))
13	-> XN Hash (cost=20.00..20.00 rows=2000 width=63)
14	-> XN Seq Scan on dim_customers dc (cost=0.00..20.00 rows=2000 width=63)
15	-> XN Hash (cost=0.88..0.88 rows=16 width=33)
16	-> XN Seq Scan on dim_products dp (cost=0.00..0.88 rows=16 width=33)
17	Filter: (product_price > 3000)

Log table for the results (time) of select statement execution (4 runs without 1st):

Number of runs	2	3	4	5
Execution time	74 ms	53 ms	47 ms	54 ms

Check that the result caching is turned off (source_query is NULL):

```
select pid, starttime, substring, source_query from svl_qlog
where userid = 118
ORDER BY starttime DESC;
```

stll_query(+) 1

select pid, starttime, substring, source_query from svl_qlog | Enter a SQL expression to filter results (use Ctrl+Space)

	pid	starttime	substring	source_query
1	1,073,878,141	2022-04-04 18:22:23.399	select * from svl_qlog where userid = 118 ORDER BY starttime	[NULL]
2	1,073,878,141	2022-04-04 18:21:53.923	select * from svl_qlog where userid = 118 ORDER BY starttim	[NULL]
3	1,073,878,141	2022-04-04 18:13:15.925	select userid, query, elapsed, source_query from svl_qlog wh	[NULL]
4	1,073,878,141	2022-04-04 18:10:55.824	select * from svl_qlog where userid = 118 ORDER BY starttime	[NULL]
5	1,073,878,141	2022-04-04 18:09:38.095	select * from svl_qlog where userid = 118 ORDER BY starttime	[NULL]
6	1,073,878,141	2022-04-04 18:06:41.053	select * from svl_qlog where userid = 118 ORDER BY starttime	[NULL]
7	1,073,878,141	2022-04-04 18:06:34.050	select personal_id, first_name, last_name, dp.product_name,	[NULL]
8	1,073,878,141	2022-04-04 18:05:44.773	select * from svl_qlog where userid = 118 ORDER BY starttime	[NULL]
9	1,073,878,141	2022-04-04 18:05:34.178	select personal_id, first_name, last_name, dp.product_name,	[NULL]
10	1,074,233,706	2022-04-04 18:02:12.800	select personal_id, first_name, last_name, dp.product_name,	[NULL]
11	1,074,233,706	2022-04-04 18:02:09.423	select personal_id, first_name, last_name, dp.product_name,	[NULL]
12	1,074,233,706	2022-04-04 18:02:07.617	select personal_id, first_name, last_name, dp.product_name,	[NULL]
13	1,074,233,706	2022-04-04 18:02:06.950	select personal_id, first_name, last_name, dp.product_name,	[NULL]
14	1,074,233,706	2022-04-04 18:02:05.111	select personal_id, first_name, last_name, dp.product_name,	[NULL]
15	1,074,233,706	2022-04-04 18:02:04.009	select personal_id, first_name, last_name, dp.product_name,	[NULL]
16	1,074,233,706	2022-04-04 18:02:02.972	select personal_id, first_name, last_name, dp.product_name,	[NULL]

The results of the first execution are not relevant because of the program loses time for compilation the code. After the compilation (first run) the program will do it faster and uses less compute capacity.

The existing distribution style for the tables is ALL for small tables (like dim_products or dim_customers) and EVEN for big tables like fct_payments_dd. There are no sort keys in all the tables.

5. Optimization

Number of runs	Execution time
2	74 ms
3	53 ms
4	47 ms
5	54 ms

The results of execution plan in p.4 are with auto settings of sort keys and distribution style. We can check it with the command below:

```
select * from SVV_TABLE_INFO where schema =
'user_dilab_student32';
```

select * from SVV_TABLE_INFO where schema = 'user_dilat | Enter a SQL expression to filter results (use Ctrl+Space)

	database	schema	table_id	table	encoded	diststyle	sortkey1	max_varchar	sortkey1_enc	sortkey_num	size
1	dev	user_dilab_student32	146559	dim_products_withoutcomp	N	AUTO(ALL)	AUTO(SORTKEY)	256	[NULL]	0	
2	dev	user_dilab_student32	152361	fct_payments_dd_withoutcomp	Y	AUTO(EVEN)	AUTO(SORTKEY)	0	[NULL]	0	
3	dev	user_dilab_student32	146148	fct_payments_dd	Y, AUTO(ENCODE)	AUTO(EVEN)	AUTO(SORTKEY)	0	[NULL]	0	
4	dev	user_dilab_student32	146161	dim_customers	Y, AUTO(ENCODE)	AUTO(ALL)	AUTO(SORTKEY)	256	[NULL]	0	
5	dev	user_dilab_student32	153791	fct_payments_dd_analyzedcomp	Y	AUTO(EVEN)	AUTO(SORTKEY)	0	[NULL]	0	
6	dev	user_dilab_student32	152251	dim_products	Y, AUTO(ENCODE)	AUTO(ALL)	AUTO(SORTKEY)	256	[NULL]	0	

With Auto settings they can change to more suitable in some period of time. We can switch off the auto optimization and check the results.

```
--change the optimization settings (for they will not change)
ALTER TABLE fct_payments_dd ALTER DISTSTYLE EVEN;
ALTER TABLE dim_customers ALTER DISTSTYLE ALL;
ALTER TABLE dim_products ALTER DISTSTYLE ALL;

ALTER TABLE fct_payments_dd ALTER SORTKEY NONE;
ALTER TABLE dim_customers ALTER SORTKEY NONE;
ALTER TABLE dim_customers ALTER SORTKEY NONE;
```

While setting the results we get the Error

SQL Error [0A000]: ERROR: This table is already SORTKEY NONE

So, we can use no check the results for the tables – they are for nonsorted data.

```
ALTER TABLE fct_payments_dd ALTER SORTKEY (sale_date);
ALTER TABLE dim_customers ALTER SORTKEY (personal_id);
ALTER TABLE dim_customers ALTER SORTKEY (dp.product_name, dp.product_price);
```

Get execute plan

explain select personal_id, first_name, last_name, dp.produ | Enter a SQL expression to filter results (use Ctrl+Space)

	QUERY PLAN
1	XN Merge (cost=1000000020026.93..1000000020026.94 rows=6 width=92)
2	Merge Key: dp.product_price, dp.product_name, fpd.sale_date
3	-> XN Network (cost=1000000020026.93..1000000020026.94 rows=6 width=92)
4	Send to leader
5	-> XN Sort (cost=1000000020026.93..1000000020026.94 rows=6 width=92)
6	Sort Key: dp.product_price, dp.product_name, fpd.sale_date
7	-> XN Hash Join DS_DIST_ALL_NONE (cost=25.91..20026.85 rows=6 width=92)
8	Hash Cond: ("outer".product_dm_surr_id = "inner".product_dm_surr_id)
9	-> XN Hash Left Join DS_DIST_ALL_NONE (cost=25.00..20025.56 rows=25 width=67)
10	Hash Cond: ("outer".customer_dm_surr_id = "inner".customer_dm_surr_id)
11	-> XN Seq Scan on fct_payments_dd fpd (cost=0.00..20000.00 rows=25 width=12)
12	Filter: (("date_part"('month'::text, sale_date) = 1) AND ("date_part"('year'::text, sale_date) = 2022))
13	-> XN Hash (cost=20.00..20.00 rows=2000 width=63)
14	-> XN Seq Scan on dim_customers dc (cost=0.00..20.00 rows=2000 width=63)
15	-> XN Hash (cost=0.88..0.88 rows=16 width=33)
16	-> XN Seq Scan on dim_products dp (cost=0.00..0.88 rows=16 width=33)
17	Filter: (product_price > 3000)

Number of runs	2	3	4	5
Execution time	50 ms	53 ms	54 ms	53 ms

Selected sort keys do not increase the effectiveness.

```sql
ALTER TABLE fct_payments_dd ALTER SORTKEY (sale_date);
ALTER TABLE dim_customers ALTER SORTKEY (customer_dm_surr_id);
ALTER TABLE dim_products ALTER SORTKEY (product_dm_surr_id, product_price);
```

	QUERY PLAN
1	XN Merge (cost=1000000020026.93..1000000020026.94 rows=6 width=92)
2	Merge Key: dp.product_price, dp.product_name, fpd.sale_date
3	-> XN Network (cost=1000000020026.93..1000000020026.94 rows=6 width=92)
4	Send to leader
5	-> XN Sort (cost=1000000020026.93..1000000020026.94 rows=6 width=92)
6	Sort Key: dp.product_price, dp.product_name, fpd.sale_date
7	-> XN Hash Join DS_DIST_ALL_NONE (cost=25.91..20026.85 rows=6 width=92)
8	Hash Cond: ("outer".product_dm_surr_id = "inner".product_dm_surr_id)
9	-> XN Hash Left Join DS_DIST_ALL_NONE (cost=25.00..20025.56 rows=25 width=67)
10	Hash Cond: ("outer".customer_dm_surr_id = "inner".customer_dm_surr_id)
11	-> XN Seq Scan on fct_payments_dd fpd (cost=0.00..20000.00 rows=25 width=12)
12	Filter: (("date_part"('month'::text, sale_date) = 1) AND ("date_part"('year'::text, sale_date) = 2022))
13	-> XN Hash (cost=20.00..20.00 rows=2000 width=63)
14	-> XN Seq Scan on dim_customers dc (cost=0.00..20.00 rows=2000 width=63)
15	-> XN Hash (cost=0.88..0.88 rows=16 width=33)
16	-> XN Seq Scan on dim_products dp (cost=0.00..0.88 rows=16 width=33)
17	Filter: (product_price > 3000)

Number of runs	2	3	4	5
Execution time	62 ms	99 ms	55 ms	63 ms

This sort keys give even worse result than without any keys. So the best way is to leave auto optimization.

```sql
ALTER TABLE fct_payments_dd ALTER SORTKEY AUTO;
ALTER TABLE dim_customers ALTER SORTKEY AUTO;
ALTER TABLE dim_products ALTER SORTKEY AUTO;
```

6. Procedure running

Create table for the report

```
--create table for the report
CREATE TABLE IF NOT EXISTS report (personal_id VARCHAR (256),
first_name VARCHAR (256),
last_name VARCHAR (256),
product_name VARCHAR (256),
product_price integer,
sale_date date)
```

Create and call the procedure

```
--create the procedure for filling the table
CREATE OR REPLACE PROCEDURE reporting(sale_year int, sale_month int, prod_price int)
AS $$
DECLARE
BEGIN
--print the notice of the beginning the procedure
raise notice  'Reporting is started';
--truncate the table
truncate table report;
--print the notice of the truncating
raise notice  'Report table is truncated';
--insert data to the table
insert into report (personal_id, first_name, last_name, product_name, product_price,    sale_date)
select
        personal_id,
        first_name,
        last_name,
        product_name,
        product_price,
        sale_date
from (
        select personal_id, first_name, last_name, dp.product_name, dp.product_price, sale_date
from fct_payments_dd fpd
left join dim_customers dc on dc.customer_dm_surr_id  = fpd.customer_dm_surr_id
left join dim_products dp on dp.product_dm_surr_id = fpd.product_dm_surr_id
where extract(year from sale_date) = sale_year and extract(month from sale_date) = sale_month and dp.product_price > prod_price
order by dp.product_price, dp.product_name, sale_date);
--print the result of the procedure
raise notice 'Report data inserted';
END;
$$
LANGUAGE plpgsql
;

--call the procedure
call reporting(2022, 03, 3000);
```

Check the results of the procedure

```
select * from report;
```

report 1

select * from report | Enter a SQL expression to filter results (use Ctrl+Space)

	personal_id	first_name	last_name	product_name	product_price	sale_date
4	ed932090-a6d5-49ed-b522-cdb9ea969319	r?o	prandini	iphone 7	3,400	2022-03-01
5	80cabdd0-9843-46d7-8683-378979928e26	l?a	woodger	lenovo ideapad 3 15alc6 82ku00ctre	3,230	2022-03-01
6	5118a864-1fdd-41f3-90b9-6c7dac77232f	simplifi?s	wakeham	lenovo ideapad 3 15alc6 82ku00ctre	3,230	2022-03-01
7	d3020400-8a6c-4323-bd4a-8fd0d814aa57	m?lia	batrop	asus tuf gaming f15 fx506hcb-hn144	5,440	2022-03-01
8	3e60470e-b9d4-429e-87c4-604feac38a84	bj?rn	immins	lenovo tab p11 plus 11	3,230	2022-03-01
9	49b330a0-a11e-421a-9a04-85c646016fc1	b?rje	o concannon	iphone 7	3,400	2022-03-01
10	52b51b0e-a3f4-4979-86f2-3baf2bc2af45	c?lia	mckinnon	asus tuf gaming f15 fx506hcb-hn144	5,440	2022-03-01
11	94619751-cb04-4fa1-af62-3cbfa8fc7e56	elo?se	hellin	hp pavilion 15-eh1123nw	4,760	2022-03-01
12	d32590a5-01b9-40f0-afb3-275d9fe9a90f	armstrong	seger	apple macbook pro 14 m1 pro 2021 mkgr3	3,975	2022-03-01
13	5b2f447a-5993-4fdb-a184-6146426b987a	nad?ge	filipson	asus tuf gaming f15 fx506hcb-hn144	5,440	2022-03-01
14	1d624147-3299-4573-b1d4-4a5dc8f76e5f	ru?	biswell	iphone 7	3,400	2022-03-01
15	f6f45535-a5c0-46aa-882d-0d9f43fa2de1	th?r?se	ringer	asus vivobook pro 15 k3500ph-kj103	4,420	2022-03-01
16	b6cf63d5-9be0-44f1-8586-1fa628b71d34	l?andre	wenman	amica ed37618x x-type	6,800	2022-03-01
17	98069956-7c25-4535-a73f-a80d847b0c57	l?ng	merrilees	asus tuf gaming f15 fx506hcb-hn144	5,440	2022-03-01
18	76fa89f8-5ee2-40fa-89d7-01e46f79ba4e	cl?mence	duckwith	apple macbook pro 14 m1 pro 2021 mkgr3	9,010	2022-03-02
19	93013db1-7866-4d4a-9b8b-6cefefb10f49	l?onie	thiem	honor magicbook 16 hym-w56 5301abcm	4,420	2022-03-02
20	2c9a7218-127f-4972-bb18-6dd6de6a8066	jimmie	garnul	jvc lt-50va7110 50	3,450	2022-03-02
21	e35bf4a4-1787-4da0-9b64-eddab75e1c78	ana?l	lakenden	jvc lt-50va7110 50	7,820	2022-03-02
22	d31fb74c-ea8b-49ce-a5a2-d793ff592ea6	simplifi?s	kitchenman	hp pavilion 15-eh1123nw	4,760	2022-03-02
23	75519acf-4aa9-4cb8-a721-f5f21bd8a743	l?andre	martusewicz	asus vivobook pro 15 k3500ph-kj103	4,420	2022-03-02
24	0061bc73-f344-4e49-8349-93c08387e616	margo	duell	apple macbook pro 14 m1 pro 2021 mkgr3	3,975	2022-03-02
25	4c3d98c4-8caf-4123-8f48-941b3e5ba9d8	b?reng?re	riby	apple macbook pro 14 m1 pro 2021 mkgr3	9,010	2022-03-02
26	40e67470-0b97-4499-b4e4-6c8f6845a322	dani?le	beefon	lenovo ideapad 3 15alc6 82ku00ctre	3,230	2022-03-02
27	77db6dbd-77b8-47b1-94d5-8b2cdb36d37a	r?becca	quaintance	jvc lt-50va7110 50	7,820	2022-03-02
28	15654ea1-9250-42dd-92be-ccf0e8ba2ce0	b?atrice	dimmick	lenovo tab p11 plus 11	3,230	2022-03-02

COPY QUESTION

As we see, the file for the table Lineorder_1 is compressed to .gz format. For the table Lineorder_2 data is in four .parquet files. It lets the service to load the information in parallel order, that increases the speed of copying. Also, the size of the loaded files is different – 3.0 GB and 2.268 GB. It also makes the executing time less as we see it on screenshots below.

Name	Type	Last modified	Size	Storage class
lineorder_00.gz	gz	March 30, 2022, 09:15:32 (UTC+03:00)	3.0 GB	Standard

Name	Type	Last modified	Size	Storage class
lineorder_000.parquet	parquet	March 30, 2022, 09:05:18 (UTC+03:00)	567.0 MB	Standard
lineorder_001.parquet	parquet	March 30, 2022, 09:10:53 (UTC+03:00)	568.8 MB	Standard
lineorder_002.parquet	parquet	March 30, 2022, 09:11:01 (UTC+03:00)	568.8 MB	Standard
lineorder_003.parquet	parquet	March 30, 2022, 09:11:07 (UTC+03:00)	567.0 MB	Standard

The time of the copying executing

querytxt	starttime	endtime	diff
copy lineorder_1 from 's3://dilabbucket/files/lineorder_file/' credentials '' region 'eu-central-1' delimiter ';' gzip DATEFORMAT	2022-04-03 10:30:40.091	2022-04-03 10:35:14.899	274,808
copy lineorder_2 from 's3://dilabbucket/files/lineorders/' credentials '' format as parquet	2022-04-03 10:27:03.537	2022-04-03 10:29:03.220	119,683

lineorder_1	2.9G
lineorder_2	2.9G

The tables contain same number of rows

```
select count (*) from lineorder_1;
```

lts 1 | Results 2 | Execution plan - 2 | Executio

t count (*) from lineorder_1 | Enter a SQL expression to filter

count
100,000,008

```
select count (*) from lineorder_2;
```

lts 1 | Results 2 | Execution plan - 2 | Executio

ct count (*) from lineorder_2 | Enter a SQL expression to filter

count
100,000,008

External tables

Create external schema

```
--create external schema
CREATE EXTERNAL SCHEMA if not exists user_dilab_student32_ext
FROM DATA catalog
DATABASE 'hanna_yaruk'
IAM_ROLE 'arn:aws:iam::260586643565:role/dilab-redshift-role';
```

Name	Value
Updated Rows	0
Query	CREATE EXTERNAL SCHEMA if not exists user_dilab_student32_ext
	FROM DATA catalog
	DATABASE 'hanna_yaruk'
	IAM_ROLE 'arn:aws:iam:: :role/dilab-redshift-role'
Finish time	Tue Apr 05 18:14:14 MSK 2022

```
select * from pg_catalog.svv_external_schemas
where schemaname ='user_dilab_student32_ext';
```

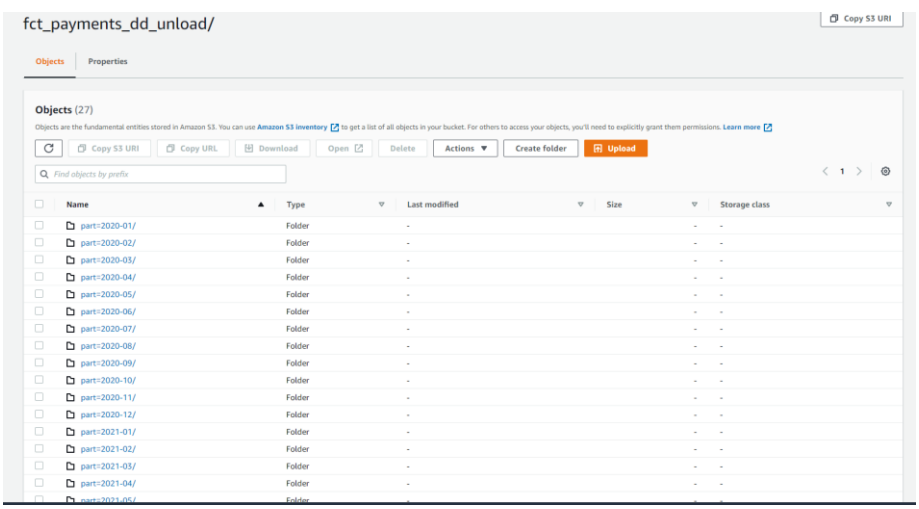
select * from pg_catalog.svv_external_schemas where sche | Enter a SQL expression to filter results (use Ctrl+Space)

	esoid	eskind	schemaname	esowner	databasename	esoptions
1	162943	1	user_dilab_student32_ext	118	hanna_yaruk	{"IAM_ROLE":"arn:aws:iam::260586643565:role/dilab-redshift-role"}

Load data to S3

```
UNLOAD ('select *, CAST(TO_CHAR(day_id, ''YYYY-MM'') AS varchar) as part from fct_payments_dd')
TO 's3://hanna-yaruk/online_shop/bl_dm/fct_payments_dd_unload/'
IAM_ROLE 'arn:aws:iam::260586643565:role/dilab-redshift-role'
CSV DELIMITER AS ','
PARTITION BY (part);
```

Data is loaded by month



Create partitioned external table

```sql
CREATE external TABLE  user_dilab_student32_ext.ext_student32_partitioned
   (payment_src_id integer,
    product_dm_surr_id integer,
    employee_dm_surr_id integer,
    warehouse_dm_surr_id integer,
    discount_dm_surr_id integer,
    address_dm_surr_id integer,
    customer_dm_surr_id integer,
    payment_type_dm_surr_id integer,
    product_cost integer,
    product_price integer,
    quantity integer,
    sale_date DATE,
    day_id DATE,
    update_dt DATE
   )
partitioned by (part varchar)
row format delimited
fields terminated by ','
stored as textfile
location 's3://hanna-yaruk/online_shop/bl_dm/fct_payments_dd_unload/';
```

The table is empty. We need to alter the table according to the partitions.

```sql
--add partitions to the table
ALTER TABLE user_dilab_student32_ext.ext_student32_partitioned
ADD IF NOT EXISTS PARTITION (part='2020-01')
LOCATION 's3://hanna-yaruk/online_shop/bl_dm/fct_payments_dd_unload/';

ALTER TABLE user_dilab_student32_ext.ext_student32_partitioned
ADD IF NOT EXISTS PARTITION (part='2020-02')
LOCATION 's3://hanna-yaruk/online_shop/bl_dm/fct_payments_dd_unload/';

ALTER TABLE user_dilab_student32_ext.ext_student32_partitioned
ADD IF NOT EXISTS PARTITION (part='2020-03')
LOCATION 's3://hanna-yaruk/online_shop/bl_dm/fct_payments_dd_unload/';

ALTER TABLE user_dilab_student32_ext.ext_student32_partitioned
ADD IF NOT EXISTS PARTITION (part='2020-04')
LOCATION 's3://hanna-yaruk/online_shop/bl_dm/fct_payments_dd_unload/';

ALTER TABLE user_dilab_student32_ext.ext_student32_partitioned
ADD IF NOT EXISTS PARTITION (part='2020-05')
LOCATION 's3://hanna-yaruk/online_shop/bl_dm/fct_payments_dd_unload/';

ALTER TABLE user_dilab_student32_ext.ext_student32_partitioned
ADD IF NOT EXISTS PARTITION (part='2020-06')
LOCATION 's3://hanna-yaruk/online_shop/bl_dm/fct_payments_dd_unload/';

ALTER TABLE user_dilab_student32_ext.ext_student32_partitioned
ADD IF NOT EXISTS PARTITION (part='2020-07')
LOCATION 's3://hanna-yaruk/online_shop/bl_dm/fct_payments_dd_unload/';

ALTER TABLE user_dilab_student32_ext.ext_student32_partitioned
ADD IF NOT EXISTS PARTITION (part='2020-08')
LOCATION 's3://hanna-yaruk/online_shop/bl_dm/fct_payments_dd_unload/';

ALTER TABLE user_dilab_student32_ext.ext_student32_partitioned
ADD IF NOT EXISTS PARTITION (part='2020-09')
LOCATION 's3://hanna-yaruk/online_shop/bl_dm/fct_payments_dd_unload/';

ALTER TABLE user_dilab_student32_ext.ext_student32_partitioned
```

Verify data in partitioned external table

Results 1

SELECT count(red.*) AS redshift_table, count(ext.*) AS exte

	redshift_table	external_table	case
1	989,010	989,010	OK

```
--verify data in partitioned external table
SELECT count(r.*) AS redshift_table, count(e.*) AS external_table,
    CASE WHEN ((redshift_table - external_table) = 0)
        THEN 'OK'
        ELSE 'Not OK'
        END
FROM user_dilab_student32.fct_payments_dd AS r
INNER JOIN user_dilab_student32_ext.ext_student32_partitioned AS e
ON r.payment_src_id = e.payment_src_id
AND r.product_dm_surr_id = e.product_dm_surr_id
AND r.employee_dm_surr_id = e.employee_dm_surr_id
AND r.warehouse_dm_surr_id = e.warehouse_dm_surr_id
AND r.discount_dm_surr_id = e.discount_dm_surr_id
AND r.address_dm_surr_id = e.address_dm_surr_id
AND r.customer_dm_surr_id = e.customer_dm_surr_id
AND r.payment_type_dm_surr_id = e.payment_type_dm_surr_id
AND r.product_cost = e.product_cost
AND r.product_price = e.product_price
AND r.quantity = e.quantity
AND r.sale_date = e.sale_date
WHERE TO_CHAR(r.day_id, 'YYYY-MM') = '2021-09';
```

Examine explain with a WHERE clause containing the partition

```
EXPLAIN
SELECT * FROM user_dilab_student32_ext.ext_student32_partitioned
WHERE part = '2021-09' ;
```

EXPLAIN SELECT * FROM user_dilab_student32_ext.ext_stu

	QUERY PLAN
1	XN Partition Loop (cost=0.00..1200000012.50 rows=50000000000 width=458)
2	-> XN Seq Scan PartitionInfo of user_dilab_student32_ext.ext_student32_partitioned (cost=0.00..12.50 rows=5 width=402)
3	Filter: ((part)::text = '2021-09'::text)
4	-> XN S3 Query Scan ext_student32_partitioned (cost=0.00..200000000.00 rows=10000000000 width=56)
5	-> S3 Seq Scan user_dilab_student32_ext.ext_student32_partitioned location:"s3://hanna-yaruk/online_shop/bl_dm/fct_payments_dd_unload" format:TEXT (cost=0.00..100000000.00 rows=10000000000 width=56)

Scan is just on the partition, that contains the needed period – without any additional movings.