Marcin Bernacki

Mateusz Guściora

ADVANCED DATABASES - PROJECT: ONLINE SHOES SHOP

TABLES OF CONTENTS:

1. Assignment 1 - Database model (spec.)

“ Minimal Requirements: 7 Relations. Each Relation should have at least 5 attributes with different types (CHAR, NUMBER, DATE) “, Database Diagram - https://drive.google.com/file/d/1OI7BduTlVQcBkuYUa_ENtEG6vWtUBcXK/view?usp=sharing

2. Assignment 2 - Database workload (spec.)

1. GET TOP ‘X’ CLIENTS FROM SPECIFIC COUNTRY WITH HIGHEST AMOUNT OF PRODUCTS ORDERED FROM SPECIFIC MANUFACTURER
    a. This query will have to search through all orders in the **ORDERS** table, which include products from specified manufacturers, and have been placed by clients from specified countries. It will then select top ‘x’ clients. This will require filtering operations for **ORDERS, CLIENTS, ORDER_PRODUCTS and PRODUCTS** tables which will lead to increased time complexity.

2. GET TOP MOST BOUGHT PRODUCTS FOR MANUFACTURERS FILTER BY COUNTRIES AND CITIES
    a. This query will show best selling products - attribute price/cost - for manufacturer filters in countries and cities. This will require joining and filtering of multiple tables (ORDER, ORDER PRODUCTS, PRODUCTS, MANUFACTURES, ADDRESSES)

3. GET SUMMED ORDERS PRICE FOR EACH CLIENT WITH DATE AND PRICE RANGE
    a. Get summed up price of orders for each client within a specific date and order price range. This query will have to filter table **ORDERS** for each client from the **CLIENTS** table based on input parameters, and calculate the sum of each client orders. This will require many individual filtering operations for each client, which will increase query time complexity.

4. GET SUMMARY OF SALES - INCOME IN GIVEN TIME PERIOD, COUNTRIES AND CITIES
    a. Sum of sold all products with filtering for manufacturers, addresses (countries, cities), group clients in countries and cities, warehouses, dates (years, months)
    (tables: ORDERS, MANUFACTURERS, ADDRESSES, CLIENTS, WAREHOUSES).
    Due to this query, reports can be built, basic analysis of sales.

5. GET MOST ORDERED PRODUCT IN EACH COUNTRY IN THE SPECIFIED TIME PERIOD
    a. This query will have to filter all orders from the **ORDERS** table based on the ordered product and the specified time period. It will then have to sum up the amount of orders of each product from the **PRODUCTS** table for each of the

countries in the **ADDRESSES** table. Calculation of ordered products will require filtering of the **ORDER_PRODUCTS** table for each of the orders.

6. GET CLIENT WITH MOST ORDERS FOR SPECIFIED PRODUCT IN SPECIFIC TIME PERIOD
    a. This query will have to filter all orders for each client. Based on the input parameter it will have to, for each client, sum up all orders of the specified product. It will then have to choose and return the client which ordered the highest amount of specified product in the specified time period. This will require joining and filtering of multiple tables (**ORDER, ORDER_PRODUCTS, PRODUCTS, CLIENTS**)

7. CHANGE DELIVERY ADDRESS OF ALL ORDERS FROM A CLIENT IN THE SPECIFIED TIME PERIOD
    a. This query will filter through all orders from the **ORDERS** table, based on the specified client from the **CLIENTS** table, select all of them in a date range, and then update their address to a new one. Based on whether the new address exists or not, this query will have to find it in the **ADDRESSES** table. If the address is not found, then the query will create it and assign it to found orders.

8. DELETE workers from a specified warehouse whose ‘contract expiration date’ is older than specified day period
    a. This modifying query deletes records - workers who are no longer working. Input parameters allow specifying workplace and time period after which workers will be removed. This query will filter through workers in the **WORKERS** table based on the specified warehouse. It will then check whether the worker's contract expiration date is old enough for him to be removed. Removal of a worker will require removing all records associated with him in tables **WAREHOUSE_WORKERS.**

9. MOVE SPECIFIED AMOUNT OF PRODUCTS TO A WAREHOUSE FROM ANOTHER.
    a. Move selected amounts of product from a warehouse to another. This query will have to filter through **WAREHOUSE_PRODUCTS** table, and based on input parameters subtract products from selected warehouses, remove the tuple if there is no product left. Then the query will have to check whether the product already exists in the selected warehouse, which will require another filtering operation on the **WAREHOUSE_PRODUCTS** table. Based on the output of this operation, new tuples will have to be added, and the existing ones will have to be updated.

3. Assignment 3 - Database model (spec.)

Oracle 19c was chosen as a DBMS due to recommendation. Tables in DBMS have been created according to the scheme with relations. Data was generated in CSV format, and then imported into database tables. One of the objectives was that prepared sample data will reflect the real dependencies between the entities and will try to reflect real life / business scenarios.

Tables:

- ADDRESS (ADDRESS_ID,ADDRESS_COUNTRY,ADDRESS_POSTAL_CODE,ADDRESS_CITY,ADDRESS_STREET,ADDRESS_BUILDING_NUMBER,ADDRESS_APARTAMENT_NUMBER)
  Tables consisting of all addresses used in database records from different tables are related with this table by key ADDRESS_ID. For each record attributes like (country, postal code, city, address street) were generated randomly using python and library Faker and attributes (address building and apartment number were randomly generated). Big number of records in this table is caused by many relations to other tables (client, worker, order, manufactures)

- MANUFACTURERS (MANUFACTURERS_ID,NAME,ADDRESSES_ID,PHONE_NUMBER,TAXPAYER_ID_NUMBER)
  Tables consist of a list of all manufacturers in the company who buys and imports shoes. For each record attributes like (name phone number, taxpayer_id(iban)) was generated randomly using python and library Faker and attributes (address id was randomly selected (without repetitions) from addresses table).

- PRODUCTS (PRODUCT_ID,NAME_PART_1,NAME_PART_2,NAME,MANUFACTURERS_ID,CATEGORY,COLOR,GENDER,MATERIAL,DESCRIPTION)
  Tables consist of a list of all shoes products available to order in shop. For each record attributes like (color, product description(long text), gender) was generated randomly using python and library Faker and attributes (address id, manufactures_id was randomly selected (with repetitions) from tables). Attributes like (name, category, material) was selected from real shoes database taken from website: http://vision.cs.utexas.edu/projects/finegrained/utzap50k/ (31.10.2021)

- WORKERS (WORKERS_ID,NAME,SURNAME,ADDRESSES_ID,EMAIL,PHONE_NUMBER,EMPLOYMENT_DATE,CONTRACT_EXPIRATION_DATE,ROLE)
  Tables consist of a list of all workers (current and fired) working in our shop. For each record attributes like (name surname, email, phone_number) was generated randomly using python and library Faker and attributes (address id was randomly selected (without repetitions) from tables) attributes like (employment date was randomly generated randomly in from 2016-10-01 to 2021-10-01 and contract

expiration date was randomly generated randomly in from 2021-10-01 to 2026-10-01) Worker role was randomly selected from list of roles with given probability ['Scientist','Technician', 'Delivery man','Menager','Office worker','Warehouse Operative','Sortation Operative'], p=[0.05,0.2,0.3,0.05,0.2,0.1,0.1]

- WAREHOUSES (WAREHOUSE_ID,WAREHOUSE_NAME,ADDRESSES_ID)
  Tables consist of a list of all warehouses which our shop uses. For each record attributes like (warehouse_name) were generated randomly using python and library Faker and attributes (address id was randomly selected (without repetitions) from tables).

- WAREHOUSE_WORKERS (WAREHOUSE_WORKER_ID,WAREHOUSE_ID,WORKERS_ID)
  Tables consist of a list of all warehouse workers working in a shop. Only some workers with roles in the list ['Technician', 'Delivery man', 'Warehouse Operative', 'Sortation Operative'] are set to warehouse other workers are set to an office). For each record attributes like (warehouse id was randomly selected (with repetitions) from table warehouses).

- CLIENTS (NAMES_ID,NAME,SURNAME,ADDRESSES_ID,EMAIL,PHONE_NUMBER,ACCOUNT_CREATION_DATE,TYPE)
  Tables consist of a list of all clients accounts in our shop. For each record attributes like (name surname, email, phone_number) were generated randomly using python and library Faker and attributes (address id was randomly selected (without repetitions) from table addresses). (employment date was randomly generated randomly from 2016-10-01 to 2021-10-01). Client type was randomly generated with the given probability described below. Next, having client type tables orders and orders_products was generated.

  Clients_Category and order numbers
  1) No_Purchace_Client 0 orders probability=10%
  2) Average_Client 1-10 orders average count of product < 4 probability=20%
  3) Averge_Max_Client 1-10 orders average count of product > 4 probability=15%
  4) Good_Client 11-20 orders average count of product < 4 probability=20%
  5) Good _Max_Client 11-20 orders average count of product > 4 probability=10%
  6) Premium_Client >21 orders average count of product < 4 probability=15%
  7) Premium_Max_Client >21 orders average count of product > 4 probability=10%

- ORDER (ORDER_ID,CLIENTS_ID,CREATION_DATE,PRICE,ADDRESSES_ID,WAREHOUSE_ID,PAID)
  Tables consist of a list of all orders in our shop. For each client taking into account client type was generated random numbers of orders from the type range described above.
  For each record attributes like (address id and warehouse_id was randomly selected (without repetitions for different clients) from tables). Creation date was randomly

generated randomly from 2016-10-01 to 2021-10-01. Price of the order was a random float from the range(100,1000). Attribute Paid was generated as type boolean with probability [[True, False], p=[0.9,0.1]]

- ORDER_PRODUCT (ORDER_PRODUCT_ID,ORDERS_ID,PRODUCTS_ID, AMOUNT)
  Tables consist of a list of all orders_products in our shop. For each order taking into account client type was generated from random numbers of orders_produts and taking the average number of all orders from a certain client. For each record attribute like (product id data was randomly selected (with repetitions) from table products), Attribute amount was generated randomly by taking into account that average amount of products per client, as described above.

- WAREHOUSE_PRODUCTS (WAREOHUSES_ID, PRODUCTS_ID, PRODUCTS_AMOUNT)
  Tables consist of a list of warehouses id and products id and product_ amount, attribute amount was generated randomly by taking into account that average amount of products per client, as described above.

table1

No	Table name	No rows
1	ORDERS	3 432 926
2	CLIENTS	999 999
3	ADDRESSES	5 000 000
4	PRODUCTS	1 000 000
5	MANUFACTURERS	100 000
6	WORKERS	100 000
7	WAREHOUSES	100 000
8	WAREHOUSES_WORKERS	70 110
9	WAREHOUSES_PRODUCTS	1 000 000
10	ORDER_PRODUCTS	10 295 462

4. Assignment 4 - Database workload (dev.)

The goal of assignment 4 was to create and apply transactions from assignment 3 in SQL language. SQL developer connected to the database was used as a tool. The goal was also to measure time of running each transaction and save this measure.

It should be mentioned that needed backup was created. Backup was created by copying sources, scripts into the external drive.

Function EXECUTE IMMEDIATE 'alter system flush buffer_cache'; was used to flush the buffer cache. Scripts that run functions and transactions were made in SQL developer. And procedures have been written to run chosen transactions. Transactions were written in SQL language using simple clauses.

Running time of each operation is measured with procedure DECLARE t_start TIMESTAMP:=systimstamp;.

Time was being saved in created tables logs
Actual time is being measured at the beginning of the procedure and at the end and then the difference between them is being measured. The results are shown in table2 in distinction for number of measures and highlighted min, max, avg.
Transactions are described above in assignment 2:

table2

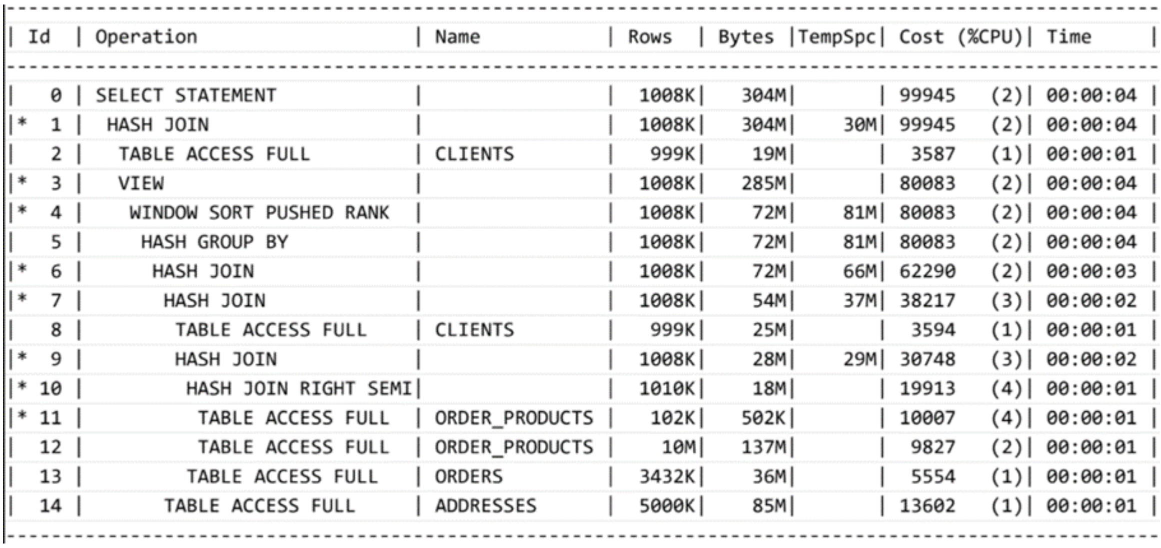
No.	AVG RUNNING TIME	MIN RUNNING TIME	MAX RUNNING TIME	ITERATIONS	DESCRIPTION	CREATION DATE
1	1.0565	0.993	1.354	100	get_top_x_clients_from_countries_with_highest_amount_of_products	14-NOV-21
2	2.97661	2.636	4.504	100	get_top_bought_products_for_manufacturer	13-NOV-21
3	2.1214	2.035	3.169	100	get_summed_orders_price_for_each_client	13-NOV-21
4	1.5459	1.337	2.375	100	get_summary_of_sales	12-NOV-21
5	10.48832	9.905	12.61	100	get_most_ordered_product_in_each_country	12-NOV-21
6	2.75161	2.337	3.932	100	get_client_with_most_orders_for_product	12-NOV-21
7	13.53356	11.076	20.504	100	change_delivery_address_of_orders_from_client	12-NOV-21

5. Assignment 5 - Query plans

There were generated query execution plans in order to optimize working of the queries. They were generated using the function - DBMS_XPLAN. The output for each query contains information about Id, Operation, Name, Rows, Bytes, Cost (%CPU), Time. The analysis and insights for each query are presented below.

1. Get top ‘x’ clients from countries

1. top x clients from countries.pdf - *Query plan table pdf file uploaded to Google Drive*
Plan hash value: 3863526124



```
------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name           | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                | 1008K|   304M|       | 99945   (2)| 00:00:04 |
|*  1 |  HASH JOIN                   |                | 1008K|   304M|    30M| 99945   (2)| 00:00:04 |
|   2 |   TABLE ACCESS FULL          | CLIENTS        |  999K|    19M|       |  3587   (1)| 00:00:01 |
|*  3 |   VIEW                       |                | 1008K|   285M|       | 80083   (2)| 00:00:04 |
|*  4 |    WINDOW SORT PUSHED RANK   |                | 1008K|    72M|    81M| 80083   (2)| 00:00:04 |
|   5 |     HASH GROUP BY            |                | 1008K|    72M|    81M| 80083   (2)| 00:00:04 |
|*  6 |      HASH JOIN               |                | 1008K|    72M|    66M| 62290   (2)| 00:00:03 |
|*  7 |       HASH JOIN              |                | 1008K|    54M|    37M| 38217   (3)| 00:00:02 |
|   8 |        TABLE ACCESS FULL     | CLIENTS        |  999K|    25M|       |  3594   (1)| 00:00:01 |
|*  9 |        HASH JOIN             |                | 1008K|    28M|    29M| 30748   (3)| 00:00:02 |
|* 10 |         HASH JOIN RIGHT SEMI|                | 1010K|    18M|       | 19913   (4)| 00:00:01 |
|* 11 |          TABLE ACCESS FULL   | ORDER_PRODUCTS |  102K|   502K|       | 10007   (4)| 00:00:01 |
|  12 |          TABLE ACCESS FULL   | ORDER_PRODUCTS |   10M|   137M|       |  9827   (2)| 00:00:01 |
|  13 |         TABLE ACCESS FULL    | ORDERS         | 3432K|    36M|       |  5554   (1)| 00:00:01 |
|  14 |       TABLE ACCESS FULL      | ADDRESSES      | 5000K|    85M|       | 13602   (1)| 00:00:01 |
------------------------------------------------------------------------------------------------
```

Whole select statement costs 99945 (%CPU). It stores 304 megabytes of data. It processes 1008 000 rows from database. Statement Join joins some columns from tables and is the most costly operation.
Most costly operations:

- ID 14 - operations on the ADDRESSES table are costly, because in this query we need to filter and group data by country, which are saved as strings. Necessity of string comparison causes overhead.

Predicate Information (identified by operation id):

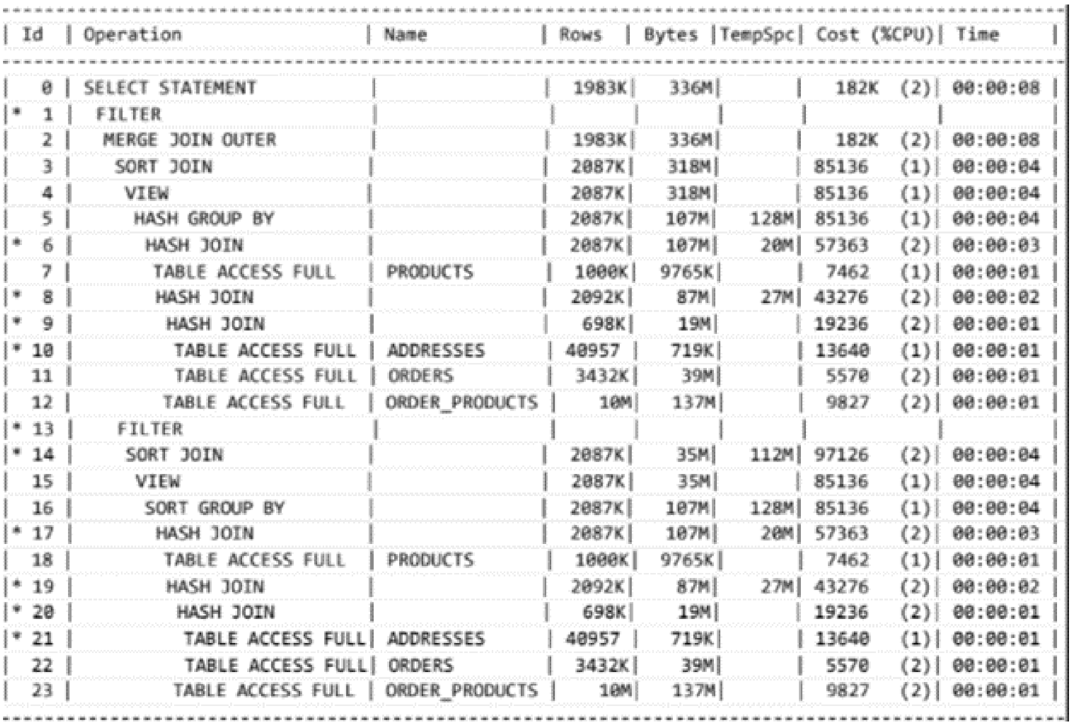
1	access("A"."CLIENT_ID"="CLIENTS"."ID")
3	filter("A"."RNK"<=20)
4	filter(DENSE_RANK() OVER (PARTITION BY "ADDRESSES"."ADDRESS_COUNTRY" ORDER BY SUM("ORDER_PRODUCTS"."AMOUNT") DESC)<=20)
6	access("CLIENTS"."ADDRESSES_ID"="ADDRESSES"."ID")

7	access("ORDERS"."CLIENTS_ID"="CLIENTS"."ID")
9	access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
10	access("ORDER_PRODUCTS"."PRODUCTS_ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
11	filter(MOD("ORDER_PRODUCTS"."PRODUCTS_ID",3)=0)

2. Get top bought products for manufacturers

2. top_bought product for manufacturer.pdf - *Query plan table pdf file uploaded to Google Drive*

Plan hash value: 275409994



```
------------------------------------------------------------------------------------------------
| Id  | Operation                 | Name           | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT          |                | 1983K|  336M|       |   182K  (2)| 00:00:08 |
|*  1 |  FILTER                   |                |      |      |       |            |          |
|   2 |   MERGE JOIN OUTER        |                | 1983K|  336M|       |   182K  (2)| 00:00:08 |
|   3 |    SORT JOIN              |                | 2087K|  318M|       | 85136   (1)| 00:00:04 |
|   4 |     VIEW                  |                | 2087K|  318M|       | 85136   (1)| 00:00:04 |
|   5 |      HASH GROUP BY        |                | 2087K|  107M|   128M| 85136   (1)| 00:00:04 |
|*  6 |       HASH JOIN           |                | 2087K|  107M|    20M| 57363   (2)| 00:00:03 |
|   7 |        TABLE ACCESS FULL  | PRODUCTS       | 1000K| 9765K|       |  7462   (1)| 00:00:01 |
|*  8 |        HASH JOIN          |                | 2092K|   87M|    27M| 43276   (2)| 00:00:02 |
|*  9 |         HASH JOIN         |                |  698K|   19M|       | 19236   (2)| 00:00:01 |
|* 10 |          TABLE ACCESS FULL | ADDRESSES      | 40957 |  719K|       | 13640   (1)| 00:00:01 |
|  11 |          TABLE ACCESS FULL | ORDERS         | 3432K|   39M|       |  5570   (2)| 00:00:01 |
|  12 |         TABLE ACCESS FULL | ORDER_PRODUCTS |   10M|  137M|       |  9827   (2)| 00:00:01 |
|* 13 |    FILTER                 |                |      |      |       |            |          |
|* 14 |     SORT JOIN             |                | 2087K|   35M|   112M| 97126   (2)| 00:00:04 |
|  15 |      VIEW                 |                | 2087K|   35M|       | 85136   (1)| 00:00:04 |
|  16 |       SORT GROUP BY       |                | 2087K|  107M|   128M| 85136   (1)| 00:00:04 |
|* 17 |        HASH JOIN          |                | 2087K|  107M|    20M| 57363   (2)| 00:00:03 |
|  18 |         TABLE ACCESS FULL | PRODUCTS       | 1000K| 9765K|       |  7462   (1)| 00:00:01 |
|* 19 |         HASH JOIN         |                | 2092K|   87M|    27M| 43276   (2)| 00:00:02 |
|* 20 |          HASH JOIN        |                |  698K|   19M|       | 19236   (2)| 00:00:01 |
|* 21 |           TABLE ACCESS FULL| ADDRESSES      | 40957 |  719K|       | 13640   (1)| 00:00:01 |
|  22 |           TABLE ACCESS FULL| ORDERS         | 3432K|   39M|       |  5570   (2)| 00:00:01 |
|  23 |          TABLE ACCESS FULL | ORDER_PRODUCTS |   10M|  137M|       |  9827   (2)| 00:00:01 |
------------------------------------------------------------------------------------------------
```

Whole select statement costs 182 000 (%CPU). It stores 336 megabytes of data. It processes 1983 000 rows from database.Statement Join joins some columns from tables and is the most costly operation.

Most costly operations:

- ID 21 - Hash joins on tables PRODUCTS and ORDER_PRODUCTS are costly because of the sheer amount of data that needs processing (10M rows, 235MB processed)

Predicate Information (identified by operation id):

2	filter("B"."MANUFACTURER" IS NULL)

3	access("A"."MANUFACTURER"="B"."MANUFACTURER"(+))
	filter("A"."SUM_PROD"<"B"."SUM_PROD"(+))

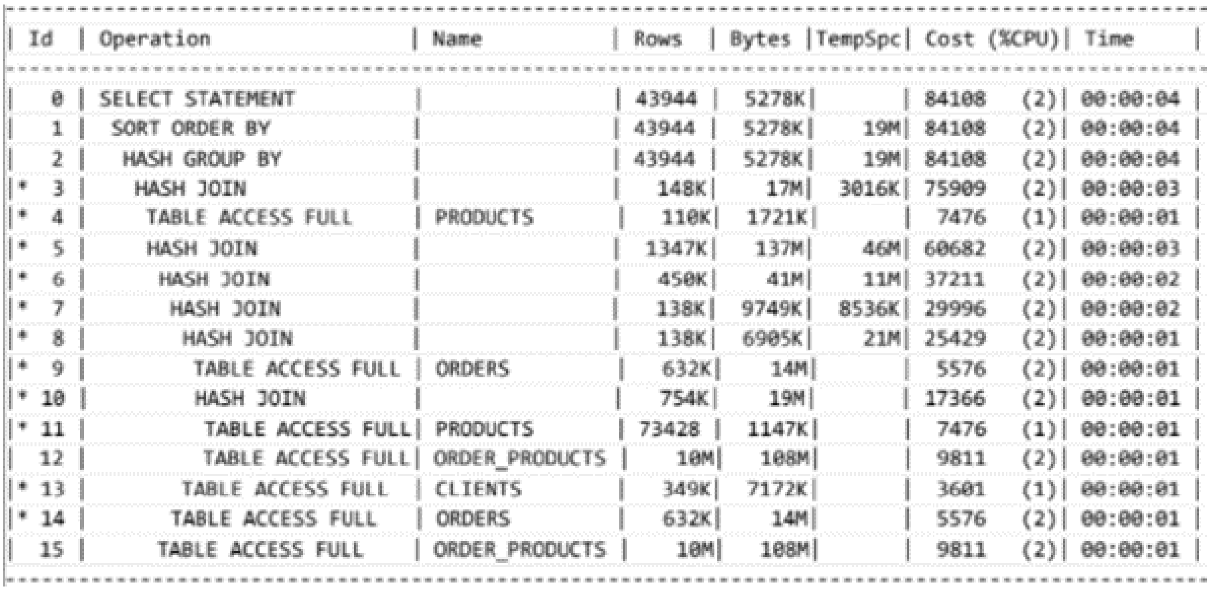
6	access("ORDER_PRODUCTS"."ORDERS_ID"="ITEM_1")

8	access("ORDERS"."ADDRESSES_ID"="ADDRESSES"."ID")
11	access("PRODUCTS"."ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
16	access("ORDER_PRODUCTS"."ORDERS_ID"="ITEM_1")
18	access("ORDERS"."ADDRESSES_ID"="ADDRESSES"."ID")
21	access("PRODUCTS"."ID"="ORDER_PRODUCTS"."PRODUCTS_ID")

3. Get summed orders price for each client

3. summed_orders_price_for_each_client.pdf

Plan hash value: 908707310



```
--------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name           | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
--------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                | 43944 |  5278K|       | 84108   (2)| 00:00:04 |
|   1 |  SORT ORDER BY             |                | 43944 |  5278K|    19M| 84108   (2)| 00:00:04 |
|   2 |   HASH GROUP BY            |                | 43944 |  5278K|    19M| 84108   (2)| 00:00:04 |
|*  3 |    HASH JOIN               |                |  148K|    17M|  3016K| 75909   (2)| 00:00:03 |
|*  4 |     TABLE ACCESS FULL      | PRODUCTS       |  110K|  1721K|       |  7476   (1)| 00:00:01 |
|*  5 |     HASH JOIN              |                | 1347K|   137M|    46M| 60682   (2)| 00:00:03 |
|*  6 |      HASH JOIN             |                |  450K|    41M|    11M| 37211   (2)| 00:00:02 |
|*  7 |       HASH JOIN            |                |  138K|  9749K|  8536K| 29996   (2)| 00:00:02 |
|*  8 |        HASH JOIN           |                |  138K|  6905K|    21M| 25429   (2)| 00:00:01 |
|*  9 |         TABLE ACCESS FULL  | ORDERS         |  632K|    14M|       |  5576   (2)| 00:00:01 |
|* 10 |         HASH JOIN          |                |  754K|    19M|       | 17366   (2)| 00:00:01 |
|* 11 |          TABLE ACCESS FULL| PRODUCTS       | 73428 |  1147K|       |  7476   (1)| 00:00:01 |
|  12 |          TABLE ACCESS FULL| ORDER_PRODUCTS |   10M|   108M|       |  9811   (2)| 00:00:01 |
|* 13 |        TABLE ACCESS FULL   | CLIENTS        |  349K|  7172K|       |  3601   (1)| 00:00:01 |
|* 14 |       TABLE ACCESS FULL    | ORDERS         |  632K|    14M|       |  5576   (2)| 00:00:01 |
|  15 |      TABLE ACCESS FULL     | ORDER_PRODUCTS |   10M|   108M|       |  9811   (2)| 00:00:01 |
--------------------------------------------------------------------------------------------------
```

Whole select statement costs 84108 (%CPU). It stores 5278 megabytes of data. It processes 43944 rows from database. Statement Join joins some columns from tables and is the most costly operation.

Most costly operations:

- ID 4 - Hash group by on tables ORDERS and CLIENTS is costly, because it needs to scan the whole table to find data for the grouping (200K rows, 3531K processed)

Predicate Information (identified by operation id):

2	access("ITEM_1"="CLIENTS"."ID")
5	filter("ORDERS"."PRICE"<=100000 AND "ORDERS"."PRICE">=1 AND "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND "ORDERS"."CREATION_DATE">=TO_DATE(' 1995-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))

4. Get summary of sales

4. summary_of_sales.pdf - *Query plan table pdf file uploaded to Google Drive*

Plan hash value: 879611230

```
| Id  | Operation                  | Name       | Rows   | Bytes |TempSpc| Cost (%CPU)| Time      |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |            |   201K|    14M|       | 50222   (2)| 00:00:02 |
|   1 |  SORT ORDER BY             |            |   201K|    14M|    17M| 50222   (2)| 00:00:02 |
|   2 |   HASH GROUP BY            |            |   201K|    14M|    17M| 50222   (2)| 00:00:02 |
|*  3 |    HASH JOIN RIGHT SEMI    |            |   201K|    14M|  7584K| 42965   (3)| 00:00:02 |
|*  4 |     TABLE ACCESS FULL      | ADDRESSES  |   204K|  5184K|       | 13789   (3)| 00:00:01 |
|*  5 |     HASH JOIN              |            |   201K|     9M|  6096K| 28203   (3)| 00:00:02 |
|   6 |      VIEW                  | VW_GBC_5   |   201K|  3733K|       | 13688   (3)| 00:00:01 |
|   7 |       HASH GROUP BY        |            |   201K|  3733K|   105M| 13688   (3)| 00:00:01 |
|*  8 |        TABLE ACCESS FULL| ORDERS     |  3432K|    62M|       |  5596   (2)| 00:00:01 |
|*  9 |      TABLE ACCESS FULL     | ADDRESSES  |   204K|  6381K|       | 13789   (3)| 00:00:01 |
-----------------------------------------------------------------------------------------------------
```

Whole select statement costs 63342 (%CPU). It stores 2491000 megabytes of data. It processes 71374 rows from database. Statement Join joins some columns from tables and is the most costly operation.

Most costly operations:

- ID 9 - Hash join on the ADDRESSES table is costly, it needs to compare strings (country stored as string) and it needs to scan through whole table to join it

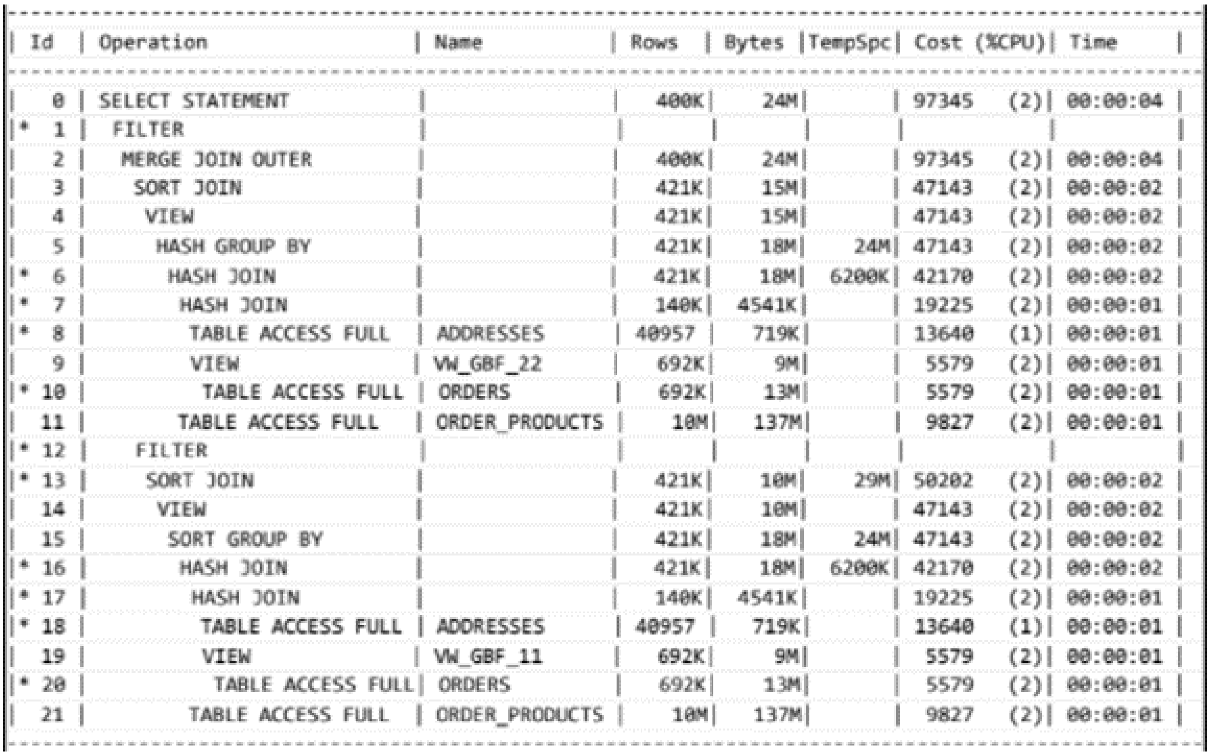
Predicate Information (identified by operation id):

3	access("ADDRESSES"."ADDRESS_COUNTRY"="ADDRESSES"."ADDRESS_COUNTRY")
4	filter(MOD("ADDRESSES"."ID",10000)=0)
5	access("ITEM_1"="ADDRESSES"."ID")
8	filter("ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND "ORDERS"."CREATION_DATE">=TO_DATE(' 1995-04-29 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
9	access("ADDRESSES"."ADDRESS_CITY"="ADDRESSES"."ADDRESS_CITY")
10	filter(MOD("ADDRESSES"."ID",55500)=0)

5. Get most ordered product in each country

5. most_ordered_product_in_each_country.pdf - *Query plan table pdf file uploaded to Google Drive*

Plan hash value: 2046251248



```
-------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name           | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                |  400K|   24M|       | 97345   (2)| 00:00:04 |
|*  1 |  FILTER                    |                |      |      |       |            |          |
|   2 |   MERGE JOIN OUTER         |                |  400K|   24M|       | 97345   (2)| 00:00:04 |
|   3 |    SORT JOIN               |                |  421K|   15M|       | 47143   (2)| 00:00:02 |
|   4 |     VIEW                   |                |  421K|   15M|       | 47143   (2)| 00:00:02 |
|   5 |      HASH GROUP BY         |                |  421K|   18M|    24M| 47143   (2)| 00:00:02 |
|*  6 |       HASH JOIN            |                |  421K|   18M|  6200K| 42170   (2)| 00:00:02 |
|*  7 |        HASH JOIN           |                |  140K| 4541K|       | 19225   (2)| 00:00:01 |
|*  8 |         TABLE ACCESS FULL  | ADDRESSES      | 40957 |  719K|       | 13640   (1)| 00:00:01 |
|   9 |         VIEW               | VW_GBF_22      |  692K|    9M|       |  5579   (2)| 00:00:01 |
|* 10 |          TABLE ACCESS FULL | ORDERS         |  692K|   13M|       |  5579   (2)| 00:00:01 |
|  11 |        TABLE ACCESS FULL   | ORDER_PRODUCTS |   10M|  137M|       |  9827   (2)| 00:00:01 |
|* 12 |    FILTER                  |                |      |      |       |            |          |
|* 13 |     SORT JOIN              |                |  421K|   10M|    29M| 50202   (2)| 00:00:02 |
|  14 |      VIEW                  |                |  421K|   10M|       | 47143   (2)| 00:00:02 |
|  15 |       SORT GROUP BY        |                |  421K|   18M|    24M| 47143   (2)| 00:00:02 |
|* 16 |        HASH JOIN           |                |  421K|   18M|  6200K| 42170   (2)| 00:00:02 |
|* 17 |         HASH JOIN          |                |  140K| 4541K|       | 19225   (2)| 00:00:01 |
|* 18 |          TABLE ACCESS FULL | ADDRESSES      | 40957 |  719K|       | 13640   (1)| 00:00:01 |
|  19 |          VIEW              | VW_GBF_11      |  692K|    9M|       |  5579   (2)| 00:00:01 |
|* 20 |           TABLE ACCESS FULL| ORDERS         |  692K|   13M|       |  5579   (2)| 00:00:01 |
|  21 |         TABLE ACCESS FULL  | ORDER_PRODUCTS |   10M|  137M|       |  9827   (2)| 00:00:01 |
-------------------------------------------------------------------------------------------------
```

Whole select statement costs 97345 (%CPU). It stores 24 megabytes of data. It processes 400 000 rows from database. Statement Join joins some columns from tables and is the most costly operation.

Predicate Information (identified by operation id):

1	filter("B"."COUNTRY" IS NULL)
6	access("ORDER_PRODUCTS"."ORDERS_ID"="ITEM_1")
7	access("ITEM_2"="ADDRESSES"."ID")
8	access("ADDRESSES"."ADDRESS_COUNTRY"="ADDRESS_COUNTRY")
10	filter("from$_subquery$_009"."rowlimit_$$_rownumber"<=1)
11	filter(ROW_NUMBER() OVER (ORDER BY "from$_subquery$_008"."rowlimit_$_0")<=1)
17	filter("ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND "ORDERS"."CREATION_DATE">=TO_DATE(' 1995-04-29 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
19	filter("A"."AMOUNT"<"B"."AMOUNT"(+))
20	access("A"."COUNTRY"="B"."COUNTRY"(+)) filter("A"."COUNTRY"="B"."COUNTRY"(+))
23	access("ORDER_PRODUCTS"."ORDERS_ID"="ITEM_1")
24	access("ITEM_2"="ADDRESSES"."ID")

25	access("ADDRESSES"."ADDRESS_COUNTRY"="ADDRESS_COUNTRY")
27	filter("from$_subquery$_018"."rowlimit_$$_rownumber"<=1)
28	filter(ROW_NUMBER() OVER (ORDER BY "from$_subquery$_017"."rowlimit_$_0")<=1)
34	filter("ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND "ORDERS"."CREATION_DATE">=TO_DATE(' 1995-04-29 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))

6. Get client with most orders for specified products

6. client_with_most_orders_for_products.pdf - *Query plan table pdf file uploaded to Google Drive*

Plan hash value: 926274802

```
| Id  | Operation                 | Name           | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT          |                |   279K|    13M|       | 63810   (2)| 00:00:03 |
|*  1 |  FILTER                   |                |       |       |       |            |          |
|   2 |   MERGE JOIN OUTER        |                |   279K|    13M|       | 63810   (2)| 00:00:03 |
|   3 |    SORT JOIN              |                |   293K|  8893K|       | 31061   (2)| 00:00:02 |
|   4 |     VIEW                  |                |   293K|  8893K|       | 31061   (2)| 00:00:02 |
|   5 |      HASH GROUP BY        |                |   293K|    12M|    15M| 31061   (2)| 00:00:02 |
|*  6 |       HASH JOIN           |                |   293K|    12M|    11M| 27819   (2)| 00:00:02 |
|*  7 |        HASH JOIN          |                |   294K|  8334K|       | 17386   (2)| 00:00:01 |
|*  8 |         TABLE ACCESS FULL | PRODUCTS       | 28643 |   419K|       |  7480   (1)| 00:00:01 |
|   9 |         TABLE ACCESS FULL | ORDER_PRODUCTS |    10M|   137M|       |  9827   (2)| 00:00:01 |
|  10 |        VIEW               | VW_GBF_22      |  3432K|    45M|       |  5586   (2)| 00:00:01 |
|* 11 |         TABLE ACCESS FULL | ORDERS         |  3432K|    62M|       |  5586   (2)| 00:00:01 |
|* 12 |    FILTER                 |                |       |       |       |            |          |
|* 13 |     SORT JOIN             |                |   293K|  5164K|    15M| 32749   (2)| 00:00:02 |
|  14 |      VIEW                 |                |   293K|  5164K|       | 31061   (2)| 00:00:02 |
|  15 |       SORT GROUP BY       |                |   293K|    12M|    15M| 31061   (2)| 00:00:02 |
|* 16 |        HASH JOIN          |                |   293K|    12M|    11M| 27819   (2)| 00:00:02 |
|* 17 |         HASH JOIN         |                |   294K|  8334K|       | 17386   (2)| 00:00:01 |
|* 18 |          TABLE ACCESS FULL| PRODUCTS       | 28643 |   419K|       |  7480   (1)| 00:00:01 |
|  19 |          TABLE ACCESS FULL| ORDER_PRODUCTS |    10M|   137M|       |  9827   (2)| 00:00:01 |
|  20 |         VIEW              | VW_GBF_11      |  3432K|    45M|       |  5586   (2)| 00:00:01 |
|* 21 |          TABLE ACCESS FULL| ORDERS         |  3432K|    62M|       |  5586   (2)| 00:00:01 |
```

Whole select statement costs 63810 (%CPU). It stores 13 megabytes of data. It processes 95 rows from database. Statement Join joins some columns from tables and is the most costly operation.

Predicate Information (identified by operation id):

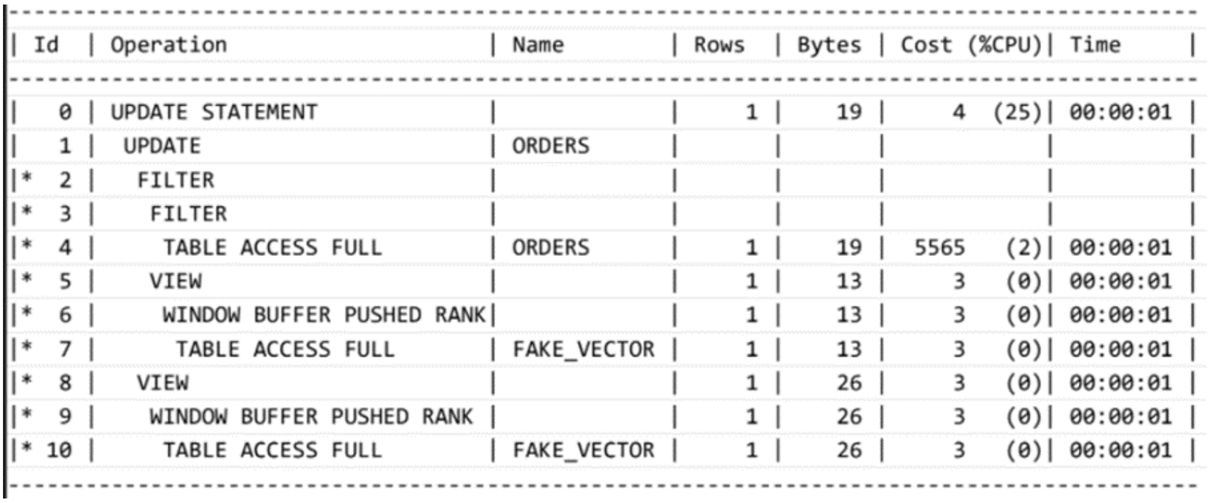
1	filter("B"."PRODUCT" IS NULL)
8	access("ORDER_PRODUCTS"."PRODUCTS_ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
10	filter("ORDER_PRODUCTS"."PRODUCTS_ID"<=100 AND "ORDER_PRODUCTS"."PRODUCTS_ID">=1)
11	filter("ORDER_PRODUCTS"."PRODUCTS_ID"<=100 AND "ORDER_PRODUCTS"."PRODUCTS_ID">=1)
12	access("ORDERS"."ID"="ORDER_PRODUCTS"."ORDERS_ID")
13	filter("ORDERS"."CREATION_DATE">=TO_DATE(' 1995-04-29 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))

14	filter("A"."AMOUNT"<"B"."AMOUNT"(+))
15	access("A"."PRODUCT"="B"."PRODUCT"(+)) filter("A"."PRODUCT"="B"."PRODUCT"(+)) filter("A"."PRODUCT"="B"."PRODUCT"(+))
20	access("ORDER_PRODUCTS"."PRODUCTS_ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
22	filter("ORDER_PRODUCTS"."PRODUCTS_ID"<=100 AND "ORDER_PRODUCTS"."PRODUCTS_ID">=1)
23	filter("ORDER_PRODUCTS"."PRODUCTS_ID"<=100 AND "ORDER_PRODUCTS"."PRODUCTS_ID">=1)
24	access("ORDERS"."ID"="ORDER_PRODUCTS"."ORDERS_ID")
25	filter("ORDERS"."CREATION_DATE">=TO_DATE(' 1995-04-29 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))

7. Change delivery address of orders from clients

7. change delivery address of orders from clients.pdf - *Query plan table pdf file uploaded to Google Drive*

Plan hash value: 4017535720



```
----------------------------------------------------------------------------------------
| Id  | Operation                       | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------
|   0 | UPDATE STATEMENT                |             |     1 |    19 |     4  (25)| 00:00:01 |
|   1 |  UPDATE                         | ORDERS      |       |       |            |          |
|*  2 |   FILTER                        |             |       |       |            |          |
|*  3 |    FILTER                       |             |       |       |            |          |
|*  4 |     TABLE ACCESS FULL           | ORDERS      |     1 |    19 |  5565   (2)| 00:00:01 |
|*  5 |    VIEW                         |             |     1 |    13 |     3   (0)| 00:00:01 |
|*  6 |     WINDOW BUFFER PUSHED RANK|                |     1 |    13 |     3   (0)| 00:00:01 |
|*  7 |      TABLE ACCESS FULL          | FAKE_VECTOR |     1 |    13 |     3   (0)| 00:00:01 |
|*  8 |   VIEW                          |             |     1 |    26 |     3   (0)| 00:00:01 |
|*  9 |    WINDOW BUFFER PUSHED RANK    |             |     1 |    26 |     3   (0)| 00:00:01 |
|* 10 |     TABLE ACCESS FULL           | FAKE_VECTOR |     1 |    26 |     3   (0)| 00:00:01 |
----------------------------------------------------------------------------------------
```

Whole select statement costs 39806 (%CPU). It stores 2491000 megabytes of data. It processes 71374 rows from database. Statement Join joins some columns from tables and is the most costly operation.

Predicate Information (identified by operation id):

2	filter(EXISTS (SELECT 0 FROM (SELECT "Y" "Y",ROW_NUMBER() OVER (ORDER BY NULL) "rowlimit_$$_rownumber" FROM "FAKE_VECTOR" "FAKE_VECTOR" WHERE "X"=:B1) from$_subquery$_006 WHERE "from$_subquery$_006"."rowlimit_$$_rownumber"<=1))
3	filter(NULL IS NOT NULL)
4	filter("ORDERS"."CREATION_DATE"<=TO_DATE(' 1995-04-29 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND "ORDERS"."CREATION_DATE">=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
5	filter("from$_subquery$_006"."rowlimit_$$_rownumber"<=1)
6	filter(ROW_NUMBER() OVER (ORDER BY NULL)<=1)
7	filter("X"=:B1)
8	filter("from$_subquery$_003"."rowlimit_$$_rownumber"<=1)
9	filter(ROW_NUMBER() OVER (ORDER BY NULL)<=1)
10	filter("X"=:B1)

Assignment 6 - Indexes (spec.)

There are 5 proposed indexes:

1. **Bitmap index** for ADDRESSES.Address_country and address_city, because our database contains 243 countries, and repeating city names which are used by 5M addresses. Can be compared with b-tree. Increase in performance in searching for country and city names would make a difference in many of our queries.
2. **Binary Tree** for ADDRESSES.Address_country for comparison with bitmap index
3. **Function index** for extracting year from Orders.data creation to improve searching in rows. We often filter by creation date of orders, and many extraction operations can slow down queries. Using function index to skip this process could improve query running times, because it will skip extraction operations.
4. **IOT** order_products and create this as an index organized table which is well suited for use with narrow tables. Order products table is often used in our queries, so this index might cause improvements in most of our queries.
5. **Bitmap index** for CLIENTS.Type_client, because it is well suited for columns which have a low cardinality of values in them. We have only a few types of clients, which are assigned to millions of clients.

There is a lot of filtering, processing and table access needed in working operations-transactions on addresses, orders, clients, order-product data rows.There are 5 proposed indexes that will be tested and are hoped to improve and optimize transaction working. It is noted that it would be pointless to create a unique type index, because there is mostly not-unique data. It is that given tables in db have low cardinality that’s why it is anticipated that bitmap indexes can be a good choice. Furthermore it is likely that the Binary tree type index, because of the high cardinality of tables, won’t be an improvement but this will be tested.

Assignment 7 - Indexes (dev.)

Development phase contains puting indexes into the database. Indexes were put in statements are the following:

- Bitmap index
- Binary tree
- Composite bitmap and binary index for multiple columns (adresses.adress_cities, adresses.adress_country AND products.materials and products.gender)
- Composite binary index for multiple columns (adress_cities, adress_country)
- Index organized table (IOT) on ORDER_PRODUCTS_TABLE

There were provided experiments which contained comparison of indexes, comparison of running times with and without indexes. And testing indexes in different statements-transaction. For example - Index IOT was tested on different statestments. And testing different indexes on querie-statestmants. (For example on query 4 we test index bitmap and IOT)

The query plans of the statements of the transactions with indexes are presented below:

Query Plans

Indexes query plans on GET_SUMMARY_OF_SALES

Without indexes

```
 -------------------------------------------------------------------------------------------
 | Id  | Operation              | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------,----------------------------------------
 |   0 | SELECT STATEMENT       |           |   201K|    14M|       | 50222   (2)| 00:00:02 |
 |   1 |  SORT ORDER BY         |           |   201K|    14M|    17M| 50222   (2)| 00:00:02 |
 |   2 |   HASH GROUP BY        |           |   201K|    14M|    17M| 50222   (2)| 00:00:02 |
 |*  3 |    HASH JOIN RIGHT SEMI |           |   201K|    14M|  7584K| 42965   (3)| 00:00:02 |
 |*  4 |     TABLE ACCESS FULL   | ADDRESSES |   204K|  5184K|       | 13789   (3)| 00:00:01 |
 |*  5 |     HASH JOIN           |           |   201K|     9M|  6096K| 28203   (3)| 00:00:02 |
 |   6 |      VIEW               | VW_GBC_5  |   201K|  3733K|       | 13688   (3)| 00:00:01 |
 |   7 |       HASH GROUP BY     |           |   201K|  3733K|   105M| 13688   (3)| 00:00:01 |
 |*  8 |        TABLE ACCESS FULL| ORDERS    |  3432K|    62M|       |  5596   (2)| 00:00:01 |
 |*  9 |      TABLE ACCESS FULL  | ADDRESSES |   204K|  6381K|       | 13789   (3)| 00:00:01 |
 -------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("ADDRESSES"."ADDRESS_CITY"="ADDRESSES"."ADDRESS_CITY")
   4 - filter("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR "ADDRESSES"."ADDRESS_COUNTRY"='Malta'
              OR "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR "ADDRESSES"."ADDRESS_COUNTRY"='Poland'
              OR "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Saudi Arabia' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
   5 - access("ITEM_1"="ADDRESSES"."ID")
   8 - filter("ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss') AND "ORDERS"."CREATION_DATE">=TO_DATE(' 1995-04-29 00:00:00',
           'syyyy-mm-dd hh24:mi:ss'))
9 - filter("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR
           "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR
           "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR "ADDRESSES"."ADDRESS_COUNTRY"='Malta'
           OR "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR "ADDRESSES"."ADDRESS_COUNTRY"='Poland'
           OR "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR
           "ADDRESSES"."ADDRESS_COUNTRY"='Saudi Arabia' OR
           "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
```

With Separate Bitmap Indexes on columns address_country and address_city

```
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name                           | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |                                |  201K|    14M|       | 47534   (2)| 00:00:02 |
|   1 |  SORT ORDER BY                         |                                |  201K|    14M|    17M| 47534   (2)| 00:00:02 |
|   2 |   HASH GROUP BY                        |                                |  201K|    14M|    17M| 47534   (2)| 00:00:02 |
|*  3 |    HASH JOIN RIGHT SEMI                |                                |  201K|    14M|  7584K| 40277   (2)| 00:00:02 |
|*  4 |     VIEW                               | index$_join$_006               |  204K|  5184K|       | 12130   (2)| 00:00:01 |
|*  5 |      HASH JOIN                         |                                |      |       |       |            |          |
|   6 |       INLIST ITERATOR                  |                                |      |       |       |            |          |
|   7 |        BITMAP CONVERSION TO ROWIDS     |                                |  204K|  5184K|       |   100   (0)| 00:00:01 |
|*  8 |         BITMAP INDEX SINGLE VALUE      | ADDRESS_COUNTRY_BITMAP_INDEX   |      |       |       |            |          |
|   9 |       BITMAP CONVERSION TO ROWIDS      |                                |  204K|  5184K|       |  3158   (1)| 00:00:01 |
|  10 |        BITMAP INDEX FULL SCAN          | ADDRESS_CITY_BITMAP_INDEX      |      |       |       |            |          |
|* 11 |     HASH JOIN                          |                                |  201K|     9M|  6096K| 27173   (2)| 00:00:02 |
|  12 |      VIEW                              | VW_GBC_5                       |  201K|  3733K|       | 13688   (3)| 00:00:01 |
|  13 |       HASH GROUP BY                    |                                |  201K|  3733K|   105M| 13688   (3)| 00:00:01 |
|* 14 |        TABLE ACCESS FULL               | ORDERS                         | 3432K|    62M|       |  5596   (2)| 00:00:01 |
|  15 |      INLIST ITERATOR                   |                                |      |       |       |            |          |
|  16 |       TABLE ACCESS BY INDEX ROWID BATCHED| ADDRESSES                    |  204K|  6381K|       | 12760   (1)| 00:00:01 |
|  17 |        BITMAP CONVERSION TO ROWIDS     |                                |      |       |       |            |          |
|* 18 |         BITMAP INDEX SINGLE VALUE      | ADDRESS_COUNTRY_BITMAP_INDEX   |      |       |       |            |          |
-------------------------------------------------------------------------------------------------------------------

   Predicate Information (identified by operation id):
   ---------------------------------------------------

 3 - access("ADDRESSES"."ADDRESS_CITY"="ADDRESSES"."ADDRESS_CITY")
 4 - filter("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR
           "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR
           "ADDRESSES"."ADDRESS_COUNTRY"='Malta' OR "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR
           "ADDRESSES"."ADDRESS_COUNTRY"='Poland' OR "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR
           "ADDRESSES"."ADDRESS_COUNTRY"='Saudi Arabia' OR "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
 5 - access(ROWID=ROWID)
 8 - access("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR
            "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR
            "ADDRESSES"."ADDRESS_COUNTRY"='Malta' OR "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR
            "ADDRESSES"."ADDRESS_COUNTRY"='Poland' OR "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR
            "ADDRESSES"."ADDRESS_COUNTRY"='Saudi Arabia' OR "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
11 - access("ITEM_1"="ADDRESSES"."ID")
14 - filter("ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
            "ORDERS"."CREATION_DATE">=TO_DATE(' 1995-04-29 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
18 - access("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR
            "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR
            "ADDRESSES"."ADDRESS_COUNTRY"='Malta' OR "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR
            "ADDRESSES"."ADDRESS_COUNTRY"='Poland' OR "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR
            "ADDRESSES"."ADDRESS_COUNTRY"='Saudi Arabia' OR "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
```

Separate B-tree indexes on columns address_country and address_city

```
                      -------------------------------------------------------------------------------------------
                      | Id  | Operation                | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
                      -------------------------------------------------------------------------------------------
                      |   0 | SELECT STATEMENT         |           |  201K|    14M|       | 50222   (2)| 00:00:02 |
                      |   1 |  SORT ORDER BY           |           |  201K|    14M|    17M| 50222   (2)| 00:00:02 |
                      |   2 |   HASH GROUP BY          |           |  201K|    14M|    17M| 50222   (2)| 00:00:02 |
                      |*  3 |    HASH JOIN RIGHT SEMI  |           |  201K|    14M|  7584K| 42965   (3)| 00:00:02 |
                      |*  4 |     TABLE ACCESS FULL    | ADDRESSES |  204K|  5184K|       | 13789   (3)| 00:00:01 |
                      |*  5 |     HASH JOIN            |           |  201K|     9M|  6096K| 28203   (3)| 00:00:02 |
                      |   6 |      VIEW                | VW_GBC_5  |  201K|  3733K|       | 13688   (3)| 00:00:01 |
                      |   7 |       HASH GROUP BY      |           |  201K|  3733K|   105M| 13688   (3)| 00:00:01 |
                      |*  8 |        TABLE ACCESS FULL| ORDERS    | 3432K|    62M|       |  5596   (2)| 00:00:01 |
                      |*  9 |      TABLE ACCESS FULL   | ADDRESSES |  204K|  6381K|       | 13789   (3)| 00:00:01 |
                      -------------------------------------------------------------------------------------------

   Predicate Information (identified by operation id):
   ---------------------------------------------------
3 - access("ADDRESSES"."ADDRESS_CITY"="ADDRESSES"."ADDRESS_CITY")
4 - filter("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR
          "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR
          "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR "ADDRESSES"."ADDRESS_COUNTRY"='Malta'
          OR "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR "ADDRESSES"."ADDRESS_COUNTRY"='Poland'
          OR "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR
          "ADDRESSES"."ADDRESS_COUNTRY"='Saudi Arabia' OR
          "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
5 - access("ITEM_1"="ADDRESSES"."ID")
8 - filter("ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd
          hh24:mi:ss') AND "ORDERS"."CREATION_DATE">=TO_DATE(' 1995-04-29 00:00:00',
          'syyyy-mm-dd hh24:mi:ss'))
9 - filter("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR
          "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR
          "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR "ADDRESSES"."ADDRESS_COUNTRY"='Malta'
OR "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR "ADDRESSES"."ADDRESS_COUNTRY"='Poland'
OR "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR
"ADDRESSES"."ADDRESS_COUNTRY"='Saudi Arabia' OR
"ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
```

Composite B-tree index on columns address_country and address_city

```
-----------------------------------------------------------------------------------------------------------
| Id  | Operation                 | Name                          | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT          |                               |   113K|  8531K|       | 35397   (2)| 00:00:02 |
|   1 |  SORT ORDER BY            |                               |   113K|  8531K|    17M| 35397   (2)| 00:00:02 |
|   2 |   HASH GROUP BY           |                               |   113K|  8531K|    17M| 35397   (2)| 00:00:02 |
|*  3 |    HASH JOIN RIGHT SEMI   |                               |   201K|    14M|  7584K| 30184   (3)| 00:00:02 |
|   4 |     INLIST ITERATOR       |                               |       |       |       |            |          |
|*  5 |      INDEX RANGE SCAN     | ADDRESS_CITY_COMP_BTREE_INDEX |   204K|  5184K|       |  1009   (1)| 00:00:01 |
|*  6 |     HASH JOIN             |                               |   201K|     9M|  6096K| 28203   (3)| 00:00:02 |
|   7 |      VIEW                 | VW_GBC_5                      |   201K|  3733K|       | 13688   (3)| 00:00:01 |
|   8 |       HASH GROUP BY       |                               |   201K|  3733K|   105M| 13688   (3)| 00:00:01 |
|*  9 |        TABLE ACCESS FULL| ORDERS                          |  3432K|    62M|       |  5596   (2)| 00:00:01 |
|* 10 |      TABLE ACCESS FULL    | ADDRESSES                     |   204K|  6381K|       | 13789   (3)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("ADDRESSES"."ADDRESS_CITY"="ADDRESSES"."ADDRESS_CITY")
   5 - access("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Malta' OR "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Poland' OR "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Saudi Arabia' OR "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
   6 - access("ITEM_1"="ADDRESSES"."ID")
   9 - filter("ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "ORDERS"."CREATION_DATE">=TO_DATE(' 1995-04-29 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
  10 - filter("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Malta' OR "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Poland' OR "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Saudi Arabia' OR "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
```

Composite Bitmap index on columns address_country and address_city

```
-----------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                 | Name                            | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                          |                                 |   113K|  8531K|       | 47750   (1)| 00:00:02 |
|   1 |  SORT ORDER BY                            |                                 |   113K|  8531K|    17M| 47750   (1)| 00:00:02 |
|   2 |   HASH GROUP BY                           |                                 |   113K|  8531K|    17M| 47750   (1)| 00:00:02 |
|*  3 |    HASH JOIN RIGHT SEMI                   |                                 |   201K|    14M|  7584K| 42537   (1)| 00:00:02 |
|   4 |     INLIST ITERATOR                       |                                 |       |       |       |            |          |
|   5 |      BITMAP CONVERSION TO ROWIDS          |                                 |   204K|  5184K|       | 13575   (1)| 00:00:01 |
|*  6 |       BITMAP INDEX RANGE SCAN             | ADDRESS_CITY_COMP_BITMAP_INDEX |       |       |       |            |          |
|*  7 |     HASH JOIN                             |                                 |   201K|     9M|  6096K| 27989   (2)| 00:00:02 |
|   8 |      VIEW                                 | VW_GBC_5                        |   201K|  3733K|       | 13688   (3)| 00:00:01 |
|   9 |       HASH GROUP BY                       |                                 |   201K|  3733K|   105M| 13688   (3)| 00:00:01 |
|* 10 |        TABLE ACCESS FULL                  | ORDERS                          |  3432K|    62M|       |  5596   (2)| 00:00:01 |
|  11 |      INLIST ITERATOR                      |                                 |       |       |       |            |          |
|  12 |       TABLE ACCESS BY INDEX ROWID BATCHED| ADDRESSES                       |   204K|  6381K|       | 13575   (1)| 00:00:01 |
|  13 |        BITMAP CONVERSION TO ROWIDS        |                                 |       |       |       |            |          |
|* 14 |         BITMAP INDEX RANGE SCAN           | ADDRESS_CITY_COMP_BITMAP_INDEX |       |       |       |            |          |
-----------------------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("ADDRESSES"."ADDRESS_CITY"="ADDRESSES"."ADDRESS_CITY")
   6 - access("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Malta' OR "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR "ADDRESSES"."ADDRESS_COUNTRY"='Poland'
              OR "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR "ADDRESSES"."ADDRESS_COUNTRY"='Saudi Arabia' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
       filter("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Malta' OR "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR "ADDRESSES"."ADDRESS_COUNTRY"='Poland'
              OR "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR "ADDRESSES"."ADDRESS_COUNTRY"='Saudi Arabia' OR
```

```
            "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
 7 - access("ITEM_1"="ADDRESSES"."ID")
10 - filter("ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
            "ORDERS"."CREATION_DATE">=TO_DATE(' 1995-04-29 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
14 - access("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR
            "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR
            "ADDRESSES"."ADDRESS_COUNTRY"='Malta' OR "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR "ADDRESSES"."ADDRESS_COUNTRY"='Poland'
            OR "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR "ADDRESSES"."ADDRESS_COUNTRY"='Saudi Arabia' OR
            "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
     filter("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR
            "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR
            "ADDRESSES"."ADDRESS_COUNTRY"='Malta' OR "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR "ADDRESSES"."ADDRESS_COUNTRY"='Poland'
            OR "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR "ADDRESSES"."ADDRESS_COUNTRY"='Saudi Arabia' OR
            "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
```

Indexes query plans on GET_SUMMED_ORDERS_PRICE_FOR_EACH_CLIENT

Without indexes

```
----------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name            | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |                 | 43944 |  5278K|       | 84108   (2)| 00:00:04 |
|   1 |  SORT ORDER BY              |                 | 43944 |  5278K|    19M| 84108   (2)| 00:00:04 |
|   2 |   HASH GROUP BY             |                 | 43944 |  5278K|    19M| 84108   (2)| 00:00:04 |
|*  3 |    HASH JOIN                |                 |   148K|    17M|  3016K| 75909   (2)| 00:00:03 |
|*  4 |     TABLE ACCESS FULL       | PRODUCTS        |   110K|  1721K|       |  7476   (1)| 00:00:01 |
|*  5 |     HASH JOIN               |                 |  1347K|   137M|    46M| 60682   (2)| 00:00:03 |
|*  6 |      HASH JOIN              |                 |   450K|    41M|    11M| 37211   (2)| 00:00:02 |
|*  7 |       HASH JOIN             |                 |   138K|  9749K|  8536K| 29996   (2)| 00:00:02 |
|*  8 |        HASH JOIN            |                 |   138K|  6905K|    21M| 25429   (2)| 00:00:01 |
|*  9 |         TABLE ACCESS FULL   | ORDERS          |   632K|    14M|       |  5576   (2)| 00:00:01 |
|* 10 |         HASH JOIN           |                 |   754K|    19M|       | 17366   (2)| 00:00:01 |
|* 11 |          TABLE ACCESS FULL| PRODUCTS        | 73428 |  1147K|       |  7476   (1)| 00:00:01 |
|  12 |          TABLE ACCESS FULL| ORDER_PRODUCTS  |    10M|   108M|       |  9811   (2)| 00:00:01 |
|* 13 |        TABLE ACCESS FULL    | CLIENTS         |   349K|  7172K|       |  3601   (1)| 00:00:01 |
|* 14 |       TABLE ACCESS FULL     | ORDERS          |   632K|    14M|       |  5576   (2)| 00:00:01 |
|  15 |      TABLE ACCESS FULL      | ORDER_PRODUCTS  |    10M|   108M|       |  9811   (2)| 00:00:01 |
----------------------------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("PRODUCTS"."ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
   4 - filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='F')
   5 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
   6 - access("ORDERS"."CLIENTS_ID"="CLIENTS"."ID")
   7 - access("CLIENTS"."ID"="ORDERS"."CLIENTS_ID")
   8 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
   9 - filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss') AND "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd
```

```
            hh24:mi:ss'))
10 - access("PRODUCTS"."ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
11 - filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='M')
13 - filter("CLIENTS"."TYPE_CLIENT"='Good_Client' OR
            "CLIENTS"."TYPE_CLIENT"='Premium_Client')
14 - filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd
            hh24:mi:ss') AND "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd
            hh24:mi:ss'))
```

With separate bitmap index on products gender and products material

```
---------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                               | Name                            | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
---------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |                                | 43944 |  5278K|       | 77829   (2)| 00:00:04 |
|   1 |  SORT ORDER BY                         |                                | 43944 |  5278K|    19M| 77829   (2)| 00:00:04 |
|   2 |   HASH GROUP BY                        |                                | 43944 |  5278K|    19M| 77829   (2)| 00:00:04 |
|*  3 |    HASH JOIN                           |                                |   148K|    17M|  3016K| 69630   (2)| 00:00:03 |
|*  4 |     VIEW                               | index$_join$_014               |   110K|  1721K|       |  4502   (2)| 00:00:01 |
|*  5 |      HASH JOIN                         |                                |       |       |       |            |          |
|*  6 |       HASH JOIN                        |                                |       |       |       |            |          |
|   7 |        BITMAP CONVERSION TO ROWIDS     |                                |   110K|  1721K|       |    33   (0)| 00:00:01 |
|*  8 |         BITMAP INDEX SINGLE VALUE      | PRODUCTS_GENDER_BITMAP_INDEX   |       |       |       |            |          |
|   9 |        BITMAP CONVERSION TO ROWIDS     |                                |   110K|  1721K|       |    48   (0)| 00:00:01 |
|* 10 |         BITMAP INDEX SINGLE VALUE      | PRODUCTS_MATERIAL_BITMAP_INDEX |       |       |       |            |          |
|  11 |       INDEX FAST FULL SCAN             | PRODUCTS_PK                    |   110K|  1721K|       |  2367   (1)| 00:00:01 |
|* 12 |     HASH JOIN                          |                                |  1347K|   137M|    46M| 57377   (2)| 00:00:03 |
|* 13 |      HASH JOIN                         |                                |   450K|    41M|    11M| 33906   (2)| 00:00:02 |
|* 14 |       HASH JOIN                        |                                |   138K|  9749K|  8536K| 26691   (2)| 00:00:02 |
|* 15 |        HASH JOIN                       |                                |   138K|  6905K|    21M| 22123   (2)| 00:00:01 |
|* 16 |         TABLE ACCESS FULL              | ORDERS                         |   632K|    14M|       |  5576   (2)| 00:00:01 |
|* 17 |         HASH JOIN                      |                                |   754K|    19M|       | 14061   (3)| 00:00:01 |
|* 18 |          VIEW                          | index$_join$_007               | 73428 |  1147K|       |  4170   (2)| 00:00:01 |
|* 19 |           HASH JOIN                    |                                |       |       |       |            |          |
|* 20 |            HASH JOIN                   |                                |       |       |       |            |          |
|  21 |             BITMAP CONVERSION TO ROWIDS|                                | 73428 |  1147K|       |    22   (0)| 00:00:01 |
|* 22 |              BITMAP INDEX SINGLE VALUE | PRODUCTS_GENDER_BITMAP_INDEX   |       |       |       |            |          |
|  23 |             BITMAP CONVERSION TO ROWIDS|                                | 73428 |  1147K|       |    48   (0)| 00:00:01 |
|* 24 |              BITMAP INDEX SINGLE VALUE | PRODUCTS_MATERIAL_BITMAP_INDEX |       |       |       |            |          |
|  25 |            INDEX FAST FULL SCAN        | PRODUCTS_PK                    | 73428 |  1147K|       |  2367   (1)| 00:00:01 |
|  26 |          TABLE ACCESS FULL             | ORDER_PRODUCTS                 |    10M|   108M|       |  9811   (2)| 00:00:01 |
|* 27 |        TABLE ACCESS FULL               | CLIENTS                        |   349K|  7172K|       |  3601   (1)| 00:00:01 |
|* 28 |       TABLE ACCESS FULL                | ORDERS                         |   632K|    14M|       |  5576   (2)| 00:00:01 |
|  29 |      TABLE ACCESS FULL                 | ORDER_PRODUCTS                 |    10M|   108M|       |  9811   (2)| 00:00:01 |
-------------------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("PRODUCTS"."ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
   4 - filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='F')
   5 - access(ROWID=ROWID)
   6 - access(ROWID=ROWID)
   8 - access("PRODUCTS"."GENDER"='F')
  10 - access("PRODUCTS"."MATERIAL"='Leather')
  12 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
  13 - access("ORDERS"."CLIENTS_ID"="CLIENTS"."ID")
  14 - access("CLIENTS"."ID"="ORDERS"."CLIENTS_ID")
  15 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
  16 - filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
  17 - access("PRODUCTS"."ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
  18 - filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='M')
  19 - access(ROWID=ROWID)
  20 - access(ROWID=ROWID)
  22 - access("PRODUCTS"."GENDER"='M')
  24 - access("PRODUCTS"."MATERIAL"='Leather')
  27 - filter("CLIENTS"."TYPE_CLIENT"='Good_Client' OR "CLIENTS"."TYPE_CLIENT"='Premium_Client')
  28 - filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
```

With composite bitmap index on products gender and products material

```
-----------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                             | Name                                      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |                                           | 43944 |  5278K|       | 75878   (2)| 00:00:03 |
|   1 |  SORT ORDER BY                        |                                           | 43944 |  5278K|    19M| 75878   (2)| 00:00:03 |
|   2 |   HASH GROUP BY                       |                                           | 43944 |  5278K|    19M| 75878   (2)| 00:00:03 |
|*  3 |    HASH JOIN                          |                                           |   148K|    17M|  3016K| 67678   (2)| 00:00:03 |
|*  4 |     VIEW                              | index$_join$_014                          |   110K|  1721K|       |  3395   (2)| 00:00:01 |
|*  5 |      HASH JOIN                        |                                           |       |       |       |            |          |
|   6 |       BITMAP CONVERSION TO ROWIDS     |                                           |   110K|  1721K|       |    35   (0)| 00:00:01 |
|*  7 |        BITMAP INDEX SINGLE VALUE      | PRODUCTS_GEND_MATERIAL_COMP_BITMAP_INDEX |       |       |       |            |          |
|   8 |       INDEX FAST FULL SCAN            | PRODUCTS_PK                               |   110K|  1721K|       |  2367   (1)| 00:00:01 |
|*  9 |     HASH JOIN                         |                                           |  1347K|   137M|    46M| 56532   (2)| 00:00:03 |
|* 10 |      HASH JOIN                        |                                           |   450K|    41M|    11M| 33062   (2)| 00:00:02 |
|* 11 |       HASH JOIN                       |                                           |   138K|  9749K|  8536K| 25847   (2)| 00:00:02 |
|* 12 |        HASH JOIN                      |                                           |   138K|  6905K|    21M| 21279   (2)| 00:00:01 |
|* 13 |         TABLE ACCESS FULL             | ORDERS                                    |   632K|    14M|       |  5576   (2)| 00:00:01 |
|* 14 |         HASH JOIN                     |                                           |   754K|    19M|       | 13216   (3)| 00:00:01 |
|* 15 |          VIEW                         | index$_join$_007                          | 73428 |  1147K|       |  3326   (2)| 00:00:01 |
|* 16 |           HASH JOIN                   |                                           |       |       |       |            |          |
|  17 |            BITMAP CONVERSION TO ROWIDS|                                           | 73428 |  1147K|       |    24   (0)| 00:00:01 |
|* 18 |             BITMAP INDEX SINGLE VALUE | PRODUCTS_GEND_MATERIAL_COMP_BITMAP_INDEX |       |       |       |            |          |
|  19 |            INDEX FAST FULL SCAN       | PRODUCTS_PK                               | 73428 |  1147K|       |  2367   (1)| 00:00:01 |
|  20 |          TABLE ACCESS FULL            | ORDER_PRODUCTS                            |    10M|   108M|       |  9811   (2)| 00:00:01 |
|* 21 |        TABLE ACCESS FULL              | CLIENTS                                   |   349K|  7172K|       |  3601   (1)| 00:00:01 |
|* 22 |       TABLE ACCESS FULL               | ORDERS                                    |   632K|    14M|       |  5576   (2)| 00:00:01 |
|  23 |      TABLE ACCESS FULL                | ORDER_PRODUCTS                            |    10M|   108M|       |  9811   (2)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

 3 - access("PRODUCTS"."ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
 4 - filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='F')
 5 - access(ROWID=ROWID)
 7 - access("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='F')
 9 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
10 - access("ORDERS"."CLIENTS_ID"="CLIENTS"."ID")
```

```
11 - access("CLIENTS"."ID"="ORDERS"."CLIENTS_ID")
12 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
13 - filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
            "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
14 - access("PRODUCTS"."ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
15 - filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='M')
16 - access(ROWID=ROWID)
18 - access("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='M')
21 - filter("CLIENTS"."TYPE_CLIENT"='Good_Client' OR "CLIENTS"."TYPE_CLIENT"='Premium_Client')
22 - filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
            "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
```

IOT query plans on GET_SUMMED_ORDERS_PRICE_FOR_EACH_CLIENT

Without IOT on ORDER_PRODUCTS

```
---------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name          | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
---------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |               | 43944 |  5278K|       | 84108   (2)| 00:00:04 |
|   1 |  SORT ORDER BY              |               | 43944 |  5278K|    19M| 84108   (2)| 00:00:04 |
|   2 |   HASH GROUP BY             |               | 43944 |  5278K|    19M| 84108   (2)| 00:00:04 |
|*  3 |    HASH JOIN                |               |   148K|    17M|  3016K| 75909   (2)| 00:00:03 |
|*  4 |     TABLE ACCESS FULL       | PRODUCTS      |   110K|  1721K|       |  7476   (1)| 00:00:01 |
|*  5 |     HASH JOIN               |               |  1347K|   137M|    46M| 60682   (2)| 00:00:03 |
|*  6 |      HASH JOIN              |               |   450K|    41M|    11M| 37211   (2)| 00:00:02 |
|*  7 |       HASH JOIN             |               |   138K|  9749K|  8536K| 29996   (2)| 00:00:02 |
|*  8 |        HASH JOIN            |               |   138K|  6905K|    21M| 25429   (2)| 00:00:01 |
|*  9 |         TABLE ACCESS FULL   | ORDERS        |   632K|    14M|       |  5576   (2)| 00:00:01 |
|* 10 |         HASH JOIN           |               |   754K|    19M|       | 17366   (2)| 00:00:01 |
|* 11 |          TABLE ACCESS FULL| PRODUCTS      | 73428 |  1147K|       |  7476   (1)| 00:00:01 |
|  12 |          TABLE ACCESS FULL| ORDER_PRODUCTS |    10M|   108M|       |  9811   (2)| 00:00:01 |
|* 13 |        TABLE ACCESS FULL    | CLIENTS       |   349K|  7172K|       |  3601   (1)| 00:00:01 |
|* 14 |       TABLE ACCESS FULL     | ORDERS        |   632K|    14M|       |  5576   (2)| 00:00:01 |
|  15 |      TABLE ACCESS FULL      | ORDER_PRODUCTS |    10M|   108M|       |  9811   (2)| 00:00:01 |
---------------------------------------------------------------------------------------------------
```

```
   Predicate Information (identified by operation id):
   ---------------------------------------------------
3 - access("PRODUCTS"."ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
4 - filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='F')
5 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
6 - access("ORDERS"."CLIENTS_ID"="CLIENTS"."ID")
7 - access("CLIENTS"."ID"="ORDERS"."CLIENTS_ID")
8 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
9 - filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd
       hh24:mi:ss') AND "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd
```

```
             hh24:mi:ss'))
  10 - access("PRODUCTS"."ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
  11 - filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='M')
  13 - filter("CLIENTS"."TYPE_CLIENT"='Good_Client' OR
             "CLIENTS"."TYPE_CLIENT"='Premium_Client')
  14 - filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd
             hh24:mi:ss') AND "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd
             hh24:mi:ss'))
```

With IOT on ORDER_PRODUCTS

```
------------------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name                       | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                            | 43944 |  6565K|       |   318K  (2)| 00:00:13 |
|   1 |  SORT ORDER BY                 |                            | 43944 |  6565K|  1134M|   318K  (2)| 00:00:13 |
|   2 |   HASH GROUP BY                |                            | 43944 |  6565K|  1134M|   318K  (2)| 00:00:13 |
|*  3 |    HASH JOIN                   |                            |  7113K|  1038M|       |   139K  (2)| 00:00:06 |
|*  4 |     TABLE ACCESS FULL          | PRODUCTS                   | 73428 |  1147K|       |  7476   (1)| 00:00:01 |
|*  5 |     HASH JOIN                  |                            |    84M|    10G|   411M|   130K  (2)| 00:00:06 |
|   6 |      INDEX FAST FULL SCAN      | ORDER_PRODUCTS_INDEX_PK |    11M|   281M|       | 10521   (1)| 00:00:01 |
|*  7 |      HASH JOIN                 |                            |  4679K|   495M|    21M| 72019   (1)| 00:00:03 |
|*  8 |       TABLE ACCESS FULL        | ORDERS                     |   632K|    14M|       |  5576   (2)| 00:00:01 |
|*  9 |       HASH JOIN                |                            |  1441K|   119M|    11M| 58563   (1)| 00:00:03 |
|* 10 |        TABLE ACCESS FULL       | CLIENTS                    |   349K|  7172K|       |  3601   (1)| 00:00:01 |
|* 11 |        HASH JOIN               |                            |  1441K|    90M|    21M| 49071   (1)| 00:00:02 |
|* 12 |         TABLE ACCESS FULL      | ORDERS                     |   632K|    14M|       |  5576   (2)| 00:00:01 |
|* 13 |         HASH JOIN              |                            |  1441K|    57M|  3016K| 38707   (1)| 00:00:02 |
|* 14 |          TABLE ACCESS FULL    | PRODUCTS                   |   110K|  1721K|       |  7476   (1)| 00:00:01 |
|  15 |          INDEX FAST FULL SCAN| ORDER_PRODUCTS_INDEX_PK |    11M|   281M|       | 10521   (1)| 00:00:01 |
------------------------------------------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("PRODUCTS"."ID"="ORDER_PRODUCTS_INDEX"."PRODUCTS_ID")
   4 - filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='M')
   5 - access("ORDER_PRODUCTS_INDEX"."ORDERS_ID"="ORDERS"."ID")
   7 - access("CLIENTS"."ID"="ORDERS"."CLIENTS_ID")
   8 - filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
   9 - access("ORDERS"."CLIENTS_ID"="CLIENTS"."ID")
  10 - filter("CLIENTS"."TYPE_CLIENT"='Good_Client' OR "CLIENTS"."TYPE_CLIENT"='Premium_Client')
  11 - access("ORDER_PRODUCTS_INDEX"."ORDERS_ID"="ORDERS"."ID")
  12 - filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
  13 - access("PRODUCTS"."ID"="ORDER_PRODUCTS_INDEX"."PRODUCTS_ID")
  14 - filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='F')
```

There were compared running times of statements with and without indexes. The measured times improved with indexes: BITMAP, COMP B-TREE, COMP BITMAP. With the rest of indexes improvement an improvement has not been noted. The results are presented and compared below.

Table 1:

Tests based on query “GET_SUMMARY_OF_SALES” 100 Iterations					
	AVG TIME	**MIN TIME**	**MAX TIME**	**DIFF AVG**	**% DIFF**
BASE	1.5459	1.337	2.375	-----------	-----------
BITMAP	1.09164	0.994	1.621	-0.45426	-29.38%
COMP B-TREE	1.01046	0.883	1.819	-0.53544	-34.64%
COMP BITMAP	0.79979	0.685	1.197	-0.74611	-48.26%

Indexes on columns Addresses.address_country and Addresses.address_city

Table 2:

Tests based on query “GET_SUMMED_ORDERS_PRICE_FOR_CLIENTS” 100 Iterations					
	AVG TIME	**MIN TIME**	**MAX TIME**	**DIFF AVG**	**% DIFF**
BASE	2.21578	2.062	3.009	-----------	-----------
BITMAP	2.51526	2.261	3.336	0.29948	13.52%
COMP BITMAP	2.46789	2.148	3.107	0.25211	11.38%

Indexes on columns Products.material and Products.gender

Table 3:

Tests based on query “GET_SUMMED_ORDERS_PRICE_FOR_CLIENTS” 100 Iterations					
	AVG TIME	**MIN TIME**	**MAX TIME**	**DIFF AVG**	**% DIFF**
BASE	2.21578	2.062	3.009	-----------	-----------
IOT	17.127	16.026	19.208	14.91122	672.96%

ORDER_PRODUCTS generated as index organized table IOT

Conclusions

There were experiments with different types of indexes and comparison between indexes and comparison with running times before indexes. We've got highest time savings using composite binary indexes on table ADDRESSES linking address_country and address_city together.

Index IOT on order_products did not help improve running times. All measured times were longer with IOT.

Function index would be inadequate for any of our queries.

In GET_SUMMED_ORDERS_PRICE_FOR_CLIENTS with bitmap indexes Running times are longer but query costs are smaller.

Assignment 8 - Partitions (spec.)

The concept of partitioning is based on the idea of dividing large objects into smaller ones. Which in the context of databases means the decomposition of very large tables and indexes into smaller and more manageable parts called partitions.

There are 3 proposed partitions:

- **Partition type - Range** where DATE would be the partitioning key. We are expecting improvements in all filters which use some kind of date period.
- **Partition type - List** value CLIENTS.CLIENT_TYPE. These columns have only few different values, using list partitioned table with them as the partitioning key could improve times of many of all our queries that only filter by one of their value
- **Partition type - Hash**, where ADDRESS.CITY would be the partitioning key.

It would be an experiment with partitions between hash partitions and value list partitions.

- Another experiment would be Composite partition - List partitions by country and subpartitions on cities with hash function.
- Another experiment would be comparison between partition type composite on PRODUCT.material and Product.Gender partition type value list PRODUCT.MATERIAL and PRODUCT.GENDER.
- Another experiment will be another disc (disc C: and disc D:) for parallel DBMS execution, to test the difference in execution times between parallel and sequential execution

Assignment 9 - Partitions (dev.)

Partitions were done to decomposition of very large tables and indexes into smaller and more manageable parts called partitions in order to improve efficiency of database operation. Partitions were created in oracle sql developer by editing and rewriting existing queries. The following partitions were created:

- Range type partitions on table ORDERS with partitioning interval of 1 year, query “GET_SUMMED_ORDERS_PRICE_FOR_EACH_CLIENT”.
- List type partitions on table CLIENTS.CLIENT_TYPE, query “GET_SUMMED_ORDERS_PRICE_FOR_EACH_CLIENT
- Hash type partitions on table ADDRESSES with ADDRESS_CITY used as partitioning key, query “GET_TOP_X_CLIENTS_FROM_COUNTRIES_WITH_HIGHEST_AMOUNT_OF_PRODUCTS”
- List type partitions on table ADDRESSES with ADDRESS_COUNTRY used as partitioning key, query “GET_TOP_X_CLIENTS_FROM_COUNTRIES_WITH_HIGHEST_AMOUNT_OF_PRODUCTS”
- Composite type partitions on table ADDRESSES with ADDRESS_COUNTRY used as list partition and ADDRESS_CITY used as hash subpartition key, query “GET_TOP_X_CLIENTS_FROM_COUNTRIES_WITH_HIGHEST_AMOUNT_OF_PRODUCTS”.

Query plans, comparisons of the query plans and the comparison of running times were presented below.

Query plans

GET_SUMMED_ORDERS_PRICE_FOR_EACH_CLIENT
Without partitioning:

```
----------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name           | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                | 43944 |  5278K|       | 84108   (2)| 00:00:04 |
|   1 |  SORT ORDER BY             |                | 43944 |  5278K|    19M| 84108   (2)| 00:00:04 |
|   2 |   HASH GROUP BY            |                | 43944 |  5278K|    19M| 84108   (2)| 00:00:04 |
|*  3 |    HASH JOIN               |                |   148K|    17M|  3016K| 75909   (2)| 00:00:03 |
|*  4 |     TABLE ACCESS FULL      | PRODUCTS       |   110K|  1721K|       |  7476   (1)| 00:00:01 |
|*  5 |     HASH JOIN              |                |  1347K|   137M|    46M| 60682   (2)| 00:00:03 |
|*  6 |      HASH JOIN             |                |   450K|    41M|    11M| 37211   (2)| 00:00:02 |
|*  7 |       HASH JOIN            |                |   138K|  9749K|  8536K| 29996   (2)| 00:00:02 |
|*  8 |        HASH JOIN           |                |   138K|  6905K|    21M| 25429   (2)| 00:00:01 |
|*  9 |         TABLE ACCESS FULL  | ORDERS         |   632K|    14M|       |  5576   (2)| 00:00:01 |
|* 10 |         HASH JOIN          |                |   754K|    19M|       | 17366   (2)| 00:00:01 |
|* 11 |          TABLE ACCESS FULL | PRODUCTS       | 73428 |  1147K|       |  7476   (1)| 00:00:01 |
|  12 |          TABLE ACCESS FULL | ORDER_PRODUCTS |    10M|   108M|       |  9811   (2)| 00:00:01 |
|* 13 |        TABLE ACCESS FULL   | CLIENTS        |   349K|  7172K|       |  3601   (1)| 00:00:01 |
|* 14 |       TABLE ACCESS FULL    | ORDERS         |   632K|    14M|       |  5576   (2)| 00:00:01 |
|  15 |      TABLE ACCESS FULL     | ORDER_PRODUCTS |    10M|   108M|       |  9811   (2)| 00:00:01 |
----------------------------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("PRODUCTS"."ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
   4 - filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='F')
   5 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
   6 - access("ORDERS"."CLIENTS_ID"="CLIENTS"."ID")
   7 - access("CLIENTS"."ID"="ORDERS"."CLIENTS_ID")
   8 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
   9 - filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss') AND "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss'))
  10 - access("PRODUCTS"."ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
  11 - filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='M')
  13 - filter("CLIENTS"."TYPE_CLIENT"='Good_Client' OR
              "CLIENTS"."TYPE_CLIENT"='Premium_Client')
  14 - filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss') AND "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss'))
```

Range type partitions on table ORDERS with partitioning interval of 1 year,

```
-----------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                         | Name                      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     | Pstart| Pstop |
-----------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                  |                           | 43944 |  5278K|       | 77737   (1)| 00:00:04 |       |       |
|   1 |  SORT ORDER BY                    |                           | 43944 |  5278K|    19M| 77737   (1)| 00:00:04 |       |       |
|   2 |   HASH GROUP BY                   |                           | 43944 |  5278K|    19M| 77737   (1)| 00:00:04 |       |       |
|*  3 |    HASH JOIN                      |                           |   148K|    17M|  3016K| 69538   (2)| 00:00:03 |       |       |
|*  4 |     TABLE ACCESS FULL             | PRODUCTS                  |   110K|  1721K|       |  7476   (1)| 00:00:01 |       |       |
|*  5 |     HASH JOIN                     |                           |  1347K|   137M|    46M| 54311   (2)| 00:00:03 |       |       |
|*  6 |      HASH JOIN                    |                           |   450K|    41M|    11M| 30385   (2)| 00:00:02 |       |       |
|*  7 |       HASH JOIN                   |                           |   138K|  9749K|  8536K| 26811   (2)| 00:00:02 |       |       |
|*  8 |        HASH JOIN                  |                           |   138K|  6905K|    21M| 22243   (2)| 00:00:01 |       |       |
|   9 |         PARTITION RANGE ITERATOR|                           |   632K|    14M|       |  1936   (2)| 00:00:01 |     3 |     4 |
|* 10 |          TABLE ACCESS FULL        | ORDERS_RANGE_PARTITIONED |   632K|    14M|       |  1936   (2)| 00:00:01 |     3 |     4 |
|* 11 |         HASH JOIN                 |                           |   754K|    19M|       | 17821   (2)| 00:00:01 |       |       |
|* 12 |          TABLE ACCESS FULL        | PRODUCTS                  | 73428 |  1147K|       |  7476   (1)| 00:00:01 |       |       |
|  13 |          INDEX FAST FULL SCAN     | ORDER_PRODUCTS_INDEX_PK  |    10M|   108M|       | 10266   (2)| 00:00:01 |       |       |
|* 14 |        TABLE ACCESS FULL          | CLIENTS                   |   349K|  7172K|       |  3601   (1)| 00:00:01 |       |       |
|  15 |       PARTITION RANGE ITERATOR   |                           |   632K|    14M|       |  1936   (2)| 00:00:01 |     3 |     4 |
|* 16 |        TABLE ACCESS FULL          | ORDERS_RANGE_PARTITIONED |   632K|    14M|       |  1936   (2)| 00:00:01 |     3 |     4 |
|  17 |      INDEX FAST FULL SCAN         | ORDER_PRODUCTS_INDEX_PK  |    10M|   108M|       | 10266   (2)| 00:00:01 |       |       |
-----------------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("PRODUCTS"."ID"="ORDER_PRODUCTS_INDEX"."PRODUCTS_ID")
   4 - filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='F')
   5 - access("ORDER_PRODUCTS_INDEX"."ORDERS_ID"="ORDERS_RANGE_PARTITIONED"."ID")
   6 - access("ORDERS_RANGE_PARTITIONED"."CLIENTS_ID"="CLIENTS"."ID")
   7 - access("CLIENTS"."ID"="ORDERS_RANGE_PARTITIONED"."CLIENTS_ID")
```

```
 8 - access("ORDER_PRODUCTS_INDEX"."ORDERS_ID"="ORDERS_RANGE_PARTITIONED"."ID")
10 - filter("ORDERS_RANGE_PARTITIONED"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
            "ORDERS_RANGE_PARTITIONED"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
11 - access("PRODUCTS"."ID"="ORDER_PRODUCTS_INDEX"."PRODUCTS_ID")
12 - filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='M')
14 - filter("CLIENTS"."TYPE_CLIENT"='Good_Client' OR "CLIENTS"."TYPE_CLIENT"='Premium_Client')
16 - filter("ORDERS_RANGE_PARTITIONED"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
            "ORDERS_RANGE_PARTITIONED"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
```

List type partitions on table CLIENTS.CLIENT_TYPE:

```
---------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name                        | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     | Pstart| Pstop |
---------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                             | 43944 |  5278K|       | 82621   (2)| 00:00:04 |       |       |
|   1 |  SORT ORDER BY                 |                             | 43944 |  5278K|    19M| 82621   (2)| 00:00:04 |       |       |
|   2 |   HASH GROUP BY                |                             | 43944 |  5278K|    19M| 82621   (2)| 00:00:04 |       |       |
|*  3 |    HASH JOIN                   |                             |  148K|    17M|  3016K| 74422   (2)| 00:00:03 |       |       |
|*  4 |     TABLE ACCESS FULL          | PRODUCTS                    |  110K|  1721K|       |  7476   (1)| 00:00:01 |       |       |
|*  5 |     HASH JOIN                  |                             |  1347K|   137M|    46M| 59195   (2)| 00:00:03 |       |       |
|*  6 |      HASH JOIN                 |                             |  450K|    41M|    11M| 35269   (2)| 00:00:02 |       |       |
|*  7 |       HASH JOIN                |                             |  138K|  9749K|  8536K| 28055   (2)| 00:00:02 |       |       |
|*  8 |        HASH JOIN               |                             |  138K|  6905K|    21M| 25884   (2)| 00:00:02 |       |       |
|*  9 |         TABLE ACCESS FULL      | ORDERS                      |  632K|    14M|       |  5576   (2)| 00:00:01 |       |       |
|* 10 |         HASH JOIN              |                             |  754K|    19M|       | 17821   (2)| 00:00:01 |       |       |
|* 11 |          TABLE ACCESS FULL     | PRODUCTS                    | 73428 |  1147K|       |  7476   (1)| 00:00:01 |       |       |
|  12 |          INDEX FAST FULL SCAN| ORDER_PRODUCTS_INDEX_PK     |   10M|   108M|       | 10266   (2)| 00:00:01 |       |       |
|  13 |        PARTITION LIST INLIST |                             |  349K|  7172K|       |  1204   (1)| 00:00:01 |KEY(I) |KEY(I) |
|  14 |         TABLE ACCESS FULL      | CLIENTS_LIST_PARTITIONED |  349K|  7172K|       |  1204   (1)| 00:00:01 |KEY(I) |KEY(I) |
|* 15 |       TABLE ACCESS FULL        | ORDERS                      |  632K|    14M|       |  5576   (2)| 00:00:01 |       |       |
|  16 |      INDEX FAST FULL SCAN      | ORDER_PRODUCTS_INDEX_PK     |   10M|   108M|       | 10266   (2)| 00:00:01 |       |       |
---------------------------------------------------------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("PRODUCTS"."ID"="ORDER_PRODUCTS_INDEX"."PRODUCTS_ID")
   4 - filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='F')
   5 - access("ORDER_PRODUCTS_INDEX"."ORDERS_ID"="ORDERS"."ID")
   6 - access("ORDERS"."CLIENTS_ID"="CLIENTS_LIST_PARTITIONED"."ID")
   7 - access("CLIENTS_LIST_PARTITIONED"."ID"="ORDERS"."CLIENTS_ID")
   8 - access("ORDER_PRODUCTS_INDEX"."ORDERS_ID"="ORDERS"."ID")
   9 - filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
            "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
10 - access("PRODUCTS"."ID"="ORDER_PRODUCTS_INDEX"."PRODUCTS_ID")
11 - filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='M')
15 - filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
            "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
```

List type partitions on table PRODUCTS.MATERIAL:

```
----------------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                   | Name                         | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     | Pstart| Pstop |
----------------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                            |                              |  8101 |  3014K|       | 76695   (3)| 00:00:03 |       |       |
|   1 |  SORT ORDER BY                              |                              |  8101 |  3014K|    62M| 76695   (3)| 00:00:03 |       |       |
|   2 |   HASH GROUP BY                             |                              |  8101 |  3014K|    62M| 76695   (3)| 00:00:03 |       |       |
|   3 |    NESTED LOOPS                             |                              |   158K|    57M|       | 63719   (3)| 00:00:03 |       |       |
|   4 |     NESTED LOOPS                            |                              |  1402K|    57M|       | 63719   (3)| 00:00:03 |       |       |
|*  5 |      HASH JOIN                              |                              |  1402K|   315M|   105M| 63600   (3)| 00:00:03 |       |       |
|*  6 |       HASH JOIN                             |                              |   468K|   100M|    21M| 37177   (4)| 00:00:02 |       |       |
|*  7 |        TABLE ACCESS FULL                    | ORDERS                       |   632K|    14M|       |  5576   (2)| 00:00:01 |       |       |
|*  8 |        HASH JOIN                            |                              |   144K|    27M|    11M| 29058   (5)| 00:00:02 |       |       |
|*  9 |         TABLE ACCESS FULL                   | CLIENTS                      |   349K|  7172K|       |  3601   (1)| 00:00:01 |       |       |
|* 10 |         HASH JOIN                           |                              |   144K|    24M|    21M| 23593   (5)| 00:00:01 |       |       |
|* 11 |          TABLE ACCESS FULL                  | ORDERS                       |   632K|    14M|       |  5576   (2)| 00:00:01 |       |       |
|  12 |          NESTED LOOPS                       |                              |   785K|   116M|       | 10681  (10)| 00:00:01 |       |       |
|  13 |           NESTED LOOPS                      |                              |    10M|   116M|       | 10681  (10)| 00:00:01 |       |       |
|  14 |            TABLE ACCESS FULL                | ORDER_PRODUCTS               |    10M|   108M|       |  9811   (2)| 00:00:01 |       |       |
|* 15 |            INDEX UNIQUE SCAN                | PRODUCTS_LIST_PARTITIONED_PK |     1 |       |       |     0   (0)| 00:00:01 |       |       |
|* 16 |           TABLE ACCESS BY GLOBAL INDEX ROWID| PRODUCTS_LIST_PARTITIONED    |     1 |   145 |       |     0   (0)| 00:00:01 |    32 |    32 |
|  17 |       TABLE ACCESS FULL                     | ORDER_PRODUCTS               |    10M|   108M|       |  9811   (2)| 00:00:01 |       |       |
|* 18 |      INDEX UNIQUE SCAN                      | PRODUCTS_LIST_PARTITIONED_PK |     1 |       |       |     0   (0)| 00:00:01 |       |       |
|* 19 |     TABLE ACCESS BY GLOBAL INDEX ROWID      | PRODUCTS_LIST_PARTITIONED    |     1 |   145 |       |     0   (0)| 00:00:01 |    32 |    32 |
----------------------------------------------------------------------------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

   5 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
   6 - access("ORDERS"."CLIENTS_ID"="CLIENTS"."ID")
   7 - filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND "ORDERS"."CREATION_DATE"<=TO_DATE('
              2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
   8 - access("CLIENTS"."ID"="ORDERS"."CLIENTS_ID")
   9 - filter("CLIENTS"."TYPE_CLIENT"='Good_Client' OR "CLIENTS"."TYPE_CLIENT"='Premium_Client')
  10 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
  11 - filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND "ORDERS"."CREATION_DATE"<=TO_DATE('
              2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
  15 - access("PRODUCTS_LIST_PARTITIONED"."ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
  16 - filter("PRODUCTS_LIST_PARTITIONED"."MATERIAL"='Leather' AND "PRODUCTS_LIST_PARTITIONED"."GENDER"='M')
  18 - access("PRODUCTS_LIST_PARTITIONED"."ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
  19 - filter("PRODUCTS_LIST_PARTITIONED"."MATERIAL"='Leather' AND "PRODUCTS_LIST_PARTITIONED"."GENDER"='F')
```

Combined all partitioned tables:

```
----------------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                   | Name                         | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     | Pstart| Pstop |
----------------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                            |                              |  8101 | 3014K|       | 67928   (3)| 00:00:03 |       |       |
|   1 |  SORT ORDER BY                              |                              |  8101 | 3014K|    62M| 67928   (3)| 00:00:03 |       |       |
|   2 |   HASH GROUP BY                             |                              |  8101 | 3014K|    62M| 67928   (3)| 00:00:03 |       |       |
|   3 |    NESTED LOOPS                             |                              |  158K|    57M|       | 54952   (3)| 00:00:03 |       |       |
|   4 |     NESTED LOOPS                            |                              | 1402K|    57M|       | 54952   (3)| 00:00:03 |       |       |
|*  5 |      HASH JOIN                              |                              | 1402K|   315M|   105M| 54833   (3)| 00:00:03 |       |       |
|*  6 |       HASH JOIN                             |                              |  468K|   100M|    21M| 27954   (4)| 00:00:02 |       |       |
|   7 |        PARTITION RANGE ITERATOR             |                              |  632K|    14M|       |  1936   (2)| 00:00:01 |     3 |     4 |
|*  8 |         TABLE ACCESS FULL                   | ORDERS_RANGE_PARTITIONED     |  632K|    14M|       |  1936   (2)| 00:00:01 |     3 |     4 |
|*  9 |        HASH JOIN                            |                              |  144K|    27M|    11M| 23476   (5)| 00:00:01 |       |       |
|  10 |         PARTITION LIST INLIST               |                              |  349K|  7172K|       |  1204   (1)| 00:00:01 |KEY(I) |KEY(I) |
|  11 |          TABLE ACCESS FULL                  | CLIENTS_LIST_PARTITIONED     |  349K|  7172K|       |  1204   (1)| 00:00:01 |KEY(I) |KEY(I) |
|* 12 |         HASH JOIN                           |                              |  144K|    24M|    21M| 20408   (6)| 00:00:01 |       |       |
|  13 |          PARTITION RANGE ITERATOR           |                              |  632K|    14M|       |  1936   (2)| 00:00:01 |     3 |     4 |
|* 14 |           TABLE ACCESS FULL                 | ORDERS_RANGE_PARTITIONED     |  632K|    14M|       |  1936   (2)| 00:00:01 |     3 |     4 |
|  15 |          NESTED LOOPS                       |                              |  785K|   116M|       | 11136   (9)| 00:00:01 |       |       |
|  16 |           NESTED LOOPS                      |                              |   10M|   116M|       | 11136   (9)| 00:00:01 |       |       |
|  17 |            INDEX FAST FULL SCAN             | ORDER_PRODUCTS_INDEX_PK      |   10M|   108M|       | 10266   (2)| 00:00:01 |       |       |
|* 18 |            INDEX UNIQUE SCAN                | PRODUCTS_LIST_PARTITIONED_PK |     1 |       |       |     0   (0)| 00:00:01 |       |       |
|* 19 |           TABLE ACCESS BY GLOBAL INDEX ROWID| PRODUCTS_LIST_PARTITIONED    |     1 |   145 |       |     0   (0)| 00:00:01 |    32 |    32 |
|  20 |       INDEX FAST FULL SCAN                  | ORDER_PRODUCTS_INDEX_PK      |   10M|   108M|       | 10266   (2)| 00:00:01 |       |       |
|* 21 |      INDEX UNIQUE SCAN                      | PRODUCTS_LIST_PARTITIONED_PK |     1 |       |       |     0   (0)| 00:00:01 |       |       |
|* 22 |     TABLE ACCESS BY GLOBAL INDEX ROWID      | PRODUCTS_LIST_PARTITIONED    |     1 |   145 |       |     0   (0)| 00:00:01 |    32 |    32 |
----------------------------------------------------------------------------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

   5 - access("ORDER_PRODUCTS_INDEX"."ORDERS_ID"="ORDERS_RANGE_PARTITIONED"."ID")
   6 - access("ORDERS_RANGE_PARTITIONED"."CLIENTS_ID"="CLIENTS_LIST_PARTITIONED"."ID")
   8 - filter("ORDERS_RANGE_PARTITIONED"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "ORDERS_RANGE_PARTITIONED"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
   9 - access("CLIENTS_LIST_PARTITIONED"."ID"="ORDERS_RANGE_PARTITIONED"."CLIENTS_ID")
  12 - access("ORDER_PRODUCTS_INDEX"."ORDERS_ID"="ORDERS_RANGE_PARTITIONED"."ID")
  14 - filter("ORDERS_RANGE_PARTITIONED"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "ORDERS_RANGE_PARTITIONED"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
  18 - access("PRODUCTS_LIST_PARTITIONED"."ID"="ORDER_PRODUCTS_INDEX"."PRODUCTS_ID")
  19 - filter("PRODUCTS_LIST_PARTITIONED"."MATERIAL"='Leather' AND "PRODUCTS_LIST_PARTITIONED"."GENDER"='M')
  21 - access("PRODUCTS_LIST_PARTITIONED"."ID"="ORDER_PRODUCTS_INDEX"."PRODUCTS_ID")
  22 - filter("PRODUCTS_LIST_PARTITIONED"."MATERIAL"='Leather' AND "PRODUCTS_LIST_PARTITIONED"."GENDER"='F')
```

GET_TOP_X_CLIENTS_FROM_COUNTRIES_WITH_HIGHEST_AMOUNT_OF_PRODUCTS

Without any partitions:

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation                             | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |                 |   215 | 68155 | 33047   (2)| 00:00:02 |
|   1 |  NESTED LOOPS                         |                 |   215 | 68155 | 33047   (2)| 00:00:02 |
|   2 |   NESTED LOOPS                        |                 |   215 | 68155 | 33047   (2)| 00:00:02 |
|*  3 |    VIEW                               |                 |   215 | 63855 | 32617   (2)| 00:00:02 |
|*  4 |     WINDOW SORT PUSHED RANK           |                 |   215 | 16125 | 32617   (2)| 00:00:02 |
|   5 |      HASH GROUP BY                    |                 |   215 | 16125 | 32617   (2)| 00:00:02 |
|   6 |       NESTED LOOPS                    |                 |   215 | 16125 | 32616   (2)| 00:00:02 |
|   7 |        NESTED LOOPS                   |                 |   215 | 16125 | 32616   (2)| 00:00:02 |
|   8 |         NESTED LOOPS                  |                 |   215 | 12255 | 32186   (2)| 00:00:02 |
|   9 |          NESTED LOOPS                 |                 |   215 |  6450 | 31756   (2)| 00:00:02 |
|* 10 |           HASH JOIN RIGHT SEMI        |                 |   216 |  4104 | 31324   (2)| 00:00:02 |
|  11 |            VIEW                       | VW_NSO_1        |    21 |   105 | 21419   (1)| 00:00:01 |
|* 12 |             HASH JOIN                 |                 |    21 |  1113 | 21419   (1)| 00:00:01 |
|* 13 |              HASH JOIN                |                 |     2 |    86 | 13949   (2)| 00:00:01 |
|* 14 |               TABLE ACCESS FULL       | ADDRESSES       |     2 |    64 | 13674   (2)| 00:00:01 |
|  15 |               TABLE ACCESS FULL       | MANUFACTURERS   |   100K|  1074K|   274   (1)| 00:00:01 |
|  16 |              TABLE ACCESS FULL        | PRODUCTS        |  1000K|  9765K|  7462   (1)| 00:00:01 |
|  17 |            TABLE ACCESS FULL          | ORDER_PRODUCTS  |    10M|   137M|  9827   (2)| 00:00:01 |
|  18 |           TABLE ACCESS BY INDEX ROWID| ORDERS          |     1 |    11 |     2   (0)| 00:00:01 |
|* 19 |            INDEX UNIQUE SCAN          | ORDERS_PK       |     1 |       |     1   (0)| 00:00:01 |
|  20 |          TABLE ACCESS BY INDEX ROWID  | CLIENTS         |     1 |    27 |     2   (0)| 00:00:01 |
|* 21 |           INDEX UNIQUE SCAN           | CLIENTS_PK      |     1 |       |     1   (0)| 00:00:01 |
|* 22 |         INDEX UNIQUE SCAN             | ADDRESSES_PK    |     1 |       |     1   (0)| 00:00:01 |
|  23 |        TABLE ACCESS BY INDEX ROWID    | ADDRESSES       |     1 |    18 |     2   (0)| 00:00:01 |
|* 24 |    INDEX UNIQUE SCAN                  | CLIENTS_PK      |     1 |       |     1   (0)| 00:00:01 |
|  25 |   TABLE ACCESS BY INDEX ROWID         | CLIENTS         |     1 |    20 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("A"."RNK"<=20)
   4 - filter(DENSE_RANK() OVER ( PARTITION BY "ADDRESSES"."ADDRESS_COUNTRY" ORDER BY
              SUM("ORDER_PRODUCTS"."AMOUNT") DESC )<=20)
  10 - access("ORDER_PRODUCTS"."PRODUCTS_ID"="ID")
  12 - access("MANUFACTURERS"."ID"="PRODUCTS"."MANUFACTURERS_ID")
  13 - access("ADDRESSES"."ID"="MANUFACTURERS"."ADDRESSES_ID")
  14 - filter(("ADDRESSES"."ADDRESS_CITY"='Michaelmouth' OR "ADDRESSES"."ADDRESS_CITY"='New
              Michael' OR "ADDRESSES"."ADDRESS_CITY"='West Anthony') AND
              ("ADDRESSES"."ADDRESS_COUNTRY"='Austria' OR "ADDRESSES"."ADDRESS_COUNTRY"='Korea' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Mexico'))
  19 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
  21 - access("ORDERS"."CLIENTS_ID"="CLIENTS"."ID")
  22 - access("CLIENTS"."ADDRESSES_ID"="ADDRESSES"."ID")
  24 - access("A"."CLIENT_ID"="CLIENTS"."ID")
```

Hash type partitions on table ADDRESSES with ADDRESS_CITY used as partitioning key

```
------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                 | Name                          | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                          |                               |   215 | 68155 | 20037   (2)| 00:00:01 |       |       |
|   1 |  NESTED LOOPS                             |                               |   215 | 68155 | 20037   (2)| 00:00:01 |       |       |
|   2 |   NESTED LOOPS                            |                               |   215 | 68155 | 20037   (2)| 00:00:01 |       |       |
|*  3 |    VIEW                                   |                               |   215 | 63855 | 19607   (2)| 00:00:01 |       |       |
|*  4 |     WINDOW SORT PUSHED RANK               |                               |   215 | 17200 | 19607   (2)| 00:00:01 |       |       |
|   5 |      HASH GROUP BY                        |                               |   215 | 17200 | 19607   (2)| 00:00:01 |       |       |
|   6 |       NESTED LOOPS                        |                               |   215 | 17200 | 19606   (2)| 00:00:01 |       |       |
|   7 |        NESTED LOOPS                       |                               |   215 | 17200 | 19606   (2)| 00:00:01 |       |       |
|   8 |         NESTED LOOPS                      |                               |   215 | 13330 | 19176   (2)| 00:00:01 |       |       |
|   9 |          NESTED LOOPS                     |                               |   215 |  7525 | 18745   (2)| 00:00:01 |       |       |
|* 10 |           HASH JOIN                       |                               |   216 |  5184 | 18313   (2)| 00:00:01 |       |       |
|  11 |            NESTED LOOPS                   |                               |    21 |   210 |  7969   (1)| 00:00:01 |       |       |
|  12 |             VIEW                          | VW_NSO_1                      |    21 |   105 |  7947   (1)| 00:00:01 |       |       |
|  13 |              HASH UNIQUE                  |                               |    21 |  1113 |            |          |       |       |
|* 14 |               HASH JOIN                   |                               |    21 |  1113 |  7947   (1)| 00:00:01 |       |       |
|* 15 |                HASH JOIN                  |                               |     2 |    86 |   477   (2)| 00:00:01 |       |       |
|  16 |                 PARTITION HASH INLIST     |                               |     2 |    64 |   202   (1)| 00:00:01 |KEY(I) |KEY(I) |
|* 17 |                  TABLE ACCESS FULL        | ADDRESSES_HASH_PARTITIONED    |     2 |    64 |   202   (1)| 00:00:01 |KEY(I) |KEY(I) |
|  18 |                 TABLE ACCESS FULL         | MANUFACTURERS                 |   100K|  1074K|   274   (1)| 00:00:01 |       |       |
|  19 |                TABLE ACCESS FULL          | PRODUCTS                      |  1000K|  9765K|  7462   (1)| 00:00:01 |       |       |
|* 20 |             INDEX UNIQUE SCAN             | PRODUCTS_PK                   |     1 |     5 |     1   (0)| 00:00:01 |       |       |
|  21 |            INDEX FAST FULL SCAN           | ORDER_PRODUCTS_INDEX_PK       |    10M|   137M| 10266   (2)| 00:00:01 |       |       |
|  22 |           TABLE ACCESS BY INDEX ROWID     | ORDERS                        |     1 |    11 |     2   (0)| 00:00:01 |       |       |
|* 23 |            INDEX UNIQUE SCAN              | ORDERS_PK                     |     1 |       |     1   (0)| 00:00:01 |       |       |
|  24 |          TABLE ACCESS BY INDEX ROWID      | CLIENTS                       |     1 |    27 |     2   (0)| 00:00:01 |       |       |
|* 25 |           INDEX UNIQUE SCAN               | CLIENTS_PK                    |     1 |       |     1   (0)| 00:00:01 |       |       |
|* 26 |         INDEX UNIQUE SCAN                 | ADDRESSES_HASH_PARTITIONED_PK |     1 |       |     1   (0)| 00:00:01 |       |       |
|  27 |        TABLE ACCESS BY GLOBAL INDEX ROWID| ADDRESSES_HASH_PARTITIONED    |     1 |    18 |     2   (0)| 00:00:01 | ROWID | ROWID |
|* 28 |    INDEX UNIQUE SCAN                      | CLIENTS_PK                    |     1 |       |     1   (0)| 00:00:01 |       |       |
|  29 |   TABLE ACCESS BY INDEX ROWID             | CLIENTS                       |     1 |    20 |     2   (0)| 00:00:01 |       |       |
------------------------------------------------------------------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("A"."RNK"<=20)
   4 - filter(DENSE_RANK() OVER ( PARTITION BY "ADDRESSES_HASH_PARTITIONED"."ADDRESS_COUNTRY" ORDER BY
              SUM("ORDER_PRODUCTS_INDEX"."AMOUNT") DESC )<=20)
  10 - access("PRODUCTS"."ID"="ORDER_PRODUCTS_INDEX"."PRODUCTS_ID")
  14 - access("MANUFACTURERS"."ID"="PRODUCTS"."MANUFACTURERS_ID")
  15 - access("ADDRESSES_HASH_PARTITIONED"."ID"="MANUFACTURERS"."ADDRESSES_ID")
  17 - filter(("ADDRESSES_HASH_PARTITIONED"."ADDRESS_CITY"='Michaelmouth' OR "ADDRESSES_HASH_PARTITIONED"."ADDRESS_CITY"='New
              Michael' OR "ADDRESSES_HASH_PARTITIONED"."ADDRESS_CITY"='West Anthony') AND
              ("ADDRESSES_HASH_PARTITIONED"."ADDRESS_COUNTRY"='Austria' OR "ADDRESSES_HASH_PARTITIONED"."ADDRESS_COUNTRY"='Korea' OR
              "ADDRESSES_HASH_PARTITIONED"."ADDRESS_COUNTRY"='Mexico'))
  20 - access("PRODUCTS"."ID"="ID")
  23 - access("ORDER_PRODUCTS_INDEX"."ORDERS_ID"="ORDERS"."ID")
  25 - access("ORDERS"."CLIENTS_ID"="CLIENTS"."ID")
  26 - access("CLIENTS"."ADDRESSES_ID"="ADDRESSES_HASH_PARTITIONED"."ID")
  28 - access("A"."CLIENT_ID"="CLIENTS"."ID")
```

List type partitions on table ADDRESSES with ADDRESS_COUNTRY used as partitioning key

```
----------------------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                        | Name                           | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     | Pstart| Pstop |
----------------------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                                 |                                |  6564 |  2032K|       | 31698   (2)| 00:00:02 |       |       |
|*  1 |  HASH JOIN                                       |                                |  6564 |  2032K|       | 31698   (2)| 00:00:02 |       |       |
|*  2 |   VIEW                                           |                                |  6564 |  1903K|       | 28104   (2)| 00:00:02 |       |       |
|*  3 |    WINDOW SORT PUSHED RANK                       |                                |  6564 |  1307K|  1392K| 28104   (2)| 00:00:02 |       |       |
|   4 |     HASH GROUP BY                                |                                |  6564 |  1307K|  1392K| 28104   (2)| 00:00:02 |       |       |
|   5 |      NESTED LOOPS                                |                                |  6564 |  1307K|       | 27810   (2)| 00:00:02 |       |       |
|   6 |       NESTED LOOPS                               |                                |  6564 |  1307K|       | 27810   (2)| 00:00:02 |       |       |
|*  7 |        HASH JOIN                                 |                                |  6564 |   397K|       | 27809   (2)| 00:00:02 |       |       |
|*  8 |         HASH JOIN                                |                                |  6564 |   224K|       | 24207   (2)| 00:00:01 |       |       |
|*  9 |          HASH JOIN                               |                                |  6576 |   154K|       | 18626   (2)| 00:00:01 |       |       |
|* 10 |           HASH JOIN RIGHT SEMI                   |                                |   640 |  6400 |       |  8282   (1)| 00:00:01 |       |       |
|  11 |            VIEW                                  | VW_NSO_1                       |   640 |  3200 |       |  7754   (1)| 00:00:01 |       |       |
|* 12 |             HASH JOIN                            |                                |   640 |   182K|       |  7754   (1)| 00:00:01 |       |       |
|  13 |              NESTED LOOPS SEMI                   |                                |    65 | 18330 |       |   284   (5)| 00:00:01 |       |       |
|  14 |               TABLE ACCESS FULL                  | MANUFACTURERS                  |   100K|  1074K|       |   274   (1)| 00:00:01 |       |       |
|* 15 |               TABLE ACCESS BY GLOBAL INDEX ROWID| ADDRESSES_LIST_PARTITIONED     |     1 |   271 |       |     0   (0)| 00:00:01 | ROWID | ROWID |
|* 16 |                INDEX UNIQUE SCAN                 | ADDRESSES_LIST_PARTITIONED_PK |     1 |       |       |     0   (0)| 00:00:01 |       |       |
|  17 |              TABLE ACCESS FULL                   | PRODUCTS                       |  1000K|  9765K|       |  7462   (1)| 00:00:01 |       |       |
|  18 |            INDEX FAST FULL SCAN                  | PRODUCTS_PK                    |  1000K|  4882K|       |   520   (2)| 00:00:01 |       |       |
|  19 |           INDEX FAST FULL SCAN                   | ORDER_PRODUCTS_INDEX_PK        |    10M|   137M|       | 10266   (2)| 00:00:01 |       |       |
|  20 |          TABLE ACCESS FULL                       | ORDERS                         |  3432K|    36M|       |  5554   (1)| 00:00:01 |       |       |
|  21 |         TABLE ACCESS FULL                        | CLIENTS                        |   999K|    25M|       |  3594   (1)| 00:00:01 |       |       |
|* 22 |        INDEX UNIQUE SCAN                         | ADDRESSES_LIST_PARTITIONED_PK |     1 |       |       |     0   (0)| 00:00:01 |       |       |
|  23 |       TABLE ACCESS BY GLOBAL INDEX ROWID         | ADDRESSES_LIST_PARTITIONED     |     1 |   142 |       |     0   (0)| 00:00:01 | ROWID | ROWID |
|  24 |   TABLE ACCESS FULL                              | CLIENTS                        |   999K|    19M|       |  3587   (1)| 00:00:01 |       |       |
----------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("A"."CLIENT_ID"="CLIENTS"."ID")
   2 - filter("A"."RNK"<=20)
   3 - filter(DENSE_RANK() OVER ( PARTITION BY "ADDRESSES_LIST_PARTITIONED"."ADDRESS_COUNTRY" ORDER BY SUM("ORDER_PRODUCTS_INDEX"."AMOUNT") DESC
              )<=20)
   7 - access("ORDERS"."CLIENTS_ID"="CLIENTS"."ID")
   8 - access("ORDER_PRODUCTS_INDEX"."ORDERS_ID"="ORDERS"."ID")
   9 - access("PRODUCTS"."ID"="ORDER_PRODUCTS_INDEX"."PRODUCTS_ID")
  10 - access("PRODUCTS"."ID"="ID")
  12 - access("MANUFACTURERS"."ID"="PRODUCTS"."MANUFACTURERS_ID")
  15 - filter(("ADDRESSES_LIST_PARTITIONED"."ADDRESS_COUNTRY"='Austria' OR "ADDRESSES_LIST_PARTITIONED"."ADDRESS_COUNTRY"='Korea' OR
              "ADDRESSES_LIST_PARTITIONED"."ADDRESS_COUNTRY"='Mexico') AND ("ADDRESSES_LIST_PARTITIONED"."ADDRESS_CITY"='Michaelmouth' OR
              "ADDRESSES_LIST_PARTITIONED"."ADDRESS_CITY"='New Michael' OR "ADDRESSES_LIST_PARTITIONED"."ADDRESS_CITY"='West Anthony'))
  16 - access("ADDRESSES_LIST_PARTITIONED"."ID"="MANUFACTURERS"."ADDRESSES_ID")
  22 - access("CLIENTS"."ADDRESSES_ID"="ADDRESSES_LIST_PARTITIONED"."ID")
```

Composite type partitions on table ADDRESSES with ADDRESS_COUNTRY used as list partition and ADDRESS_CITY used as hash subpartition key

```
-----------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name                               | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-----------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |                                    |   215 | 68155 | 20106   (2)| 00:00:01 |       |       |
|   1 |  NESTED LOOPS                            |                                    |   215 | 68155 | 20106   (2)| 00:00:01 |       |       |
|   2 |   NESTED LOOPS                           |                                    |   215 | 68155 | 20106   (2)| 00:00:01 |       |       |
|*  3 |    VIEW                                  |                                    |   215 | 63855 | 19676   (2)| 00:00:01 |       |       |
|*  4 |     WINDOW SORT PUSHED RANK              |                                    |   215 | 17200 | 19676   (2)| 00:00:01 |       |       |
|   5 |      HASH GROUP BY                       |                                    |   215 | 17200 | 19676   (2)| 00:00:01 |       |       |
|   6 |       NESTED LOOPS                       |                                    |   215 | 17200 | 19675   (2)| 00:00:01 |       |       |
|   7 |        NESTED LOOPS                      |                                    |   215 | 17200 | 19675   (2)| 00:00:01 |       |       |
|   8 |         NESTED LOOPS                     |                                    |   215 | 13330 | 19244   (2)| 00:00:01 |       |       |
|   9 |          NESTED LOOPS                    |                                    |   215 |  7525 | 18814   (2)| 00:00:01 |       |       |
|* 10 |           HASH JOIN                      |                                    |   216 |  5184 | 18382   (2)| 00:00:01 |       |       |
|  11 |            NESTED LOOPS                  |                                    |    21 |   210 |  8037   (1)| 00:00:01 |       |       |
|  12 |             VIEW                         | VW_NSO_1                           |    21 |   105 |  8015   (1)| 00:00:01 |       |       |
|  13 |              HASH UNIQUE                 |                                    |    21 |  1113 |            |          |       |       |
|* 14 |               HASH JOIN                  |                                    |    21 |  1113 |  8015   (1)| 00:00:01 |       |       |
|* 15 |                HASH JOIN                 |                                    |     2 |    86 |   546   (2)| 00:00:01 |       |       |
|  16 |                 PARTITION LIST INLIST    |                                    |     2 |    64 |   271   (2)| 00:00:01 |KEY(I) |KEY(I) |
|  17 |                  PARTITION HASH INLIST   |                                    |     2 |    64 |   271   (2)| 00:00:01 |KEY(I) |KEY(I) |
|* 18 |                   TABLE ACCESS FULL      | ADDRESSES_COMPOSITE_PARTITIONED    |     2 |    64 |   271   (2)| 00:00:01 |KEY(I) |KEY(I) |
|  19 |                 TABLE ACCESS FULL        | MANUFACTURERS                      |   100K|  1074K|   274   (1)| 00:00:01 |       |       |
|  20 |                TABLE ACCESS FULL         | PRODUCTS                           |  1000K|  9765K|  7462   (1)| 00:00:01 |       |       |
|* 21 |             INDEX UNIQUE SCAN            | PRODUCTS_PK                        |     1 |     5 |     1   (0)| 00:00:01 |       |       |
|  22 |            INDEX FAST FULL SCAN          | ORDER_PRODUCTS_INDEX_PK            |    10M|   137M| 10266   (2)| 00:00:01 |       |       |
|  23 |           TABLE ACCESS BY INDEX ROWID    | ORDERS                             |     1 |    11 |     2   (0)| 00:00:01 |       |       |
|* 24 |            INDEX UNIQUE SCAN             | ORDERS_PK                          |     1 |       |     1   (0)| 00:00:01 |       |       |
|  25 |          TABLE ACCESS BY INDEX ROWID     | CLIENTS                            |     1 |    27 |     2   (0)| 00:00:01 |       |       |
|* 26 |           INDEX UNIQUE SCAN              | CLIENTS_PK                         |     1 |       |     1   (0)| 00:00:01 |       |       |
|* 27 |         INDEX UNIQUE SCAN                | ADDRESSES_COMPOSITE_PARTITIONED_PK |     1 |       |     1   (0)| 00:00:01 |       |       |
|  28 |        TABLE ACCESS BY GLOBAL INDEX ROWID| ADDRESSES_COMPOSITE_PARTITIONED    |     1 |    18 |     2   (0)| 00:00:01 | ROWID | ROWID |
|* 29 |    INDEX UNIQUE SCAN                     | CLIENTS_PK                         |     1 |       |     1   (0)| 00:00:01 |       |       |
|  30 |   TABLE ACCESS BY INDEX ROWID            | CLIENTS                            |     1 |    20 |     2   (0)| 00:00:01 |       |       |
-----------------------------------------------------------------------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("A"."RNK"<=20)
   4 - filter(DENSE_RANK() OVER ( PARTITION BY "ADDRESSES_COMPOSITE_PARTITIONED"."ADDRESS_COUNTRY" ORDER BY
              SUM("ORDER_PRODUCTS_INDEX"."AMOUNT") DESC )<=20)
  10 - access("PRODUCTS"."ID"="ORDER_PRODUCTS_INDEX"."PRODUCTS_ID")
  14 - access("MANUFACTURERS"."ID"="PRODUCTS"."MANUFACTURERS_ID")
  15 - access("ADDRESSES_COMPOSITE_PARTITIONED"."ID"="MANUFACTURERS"."ADDRESSES_ID")
  18 - filter("ADDRESSES_COMPOSITE_PARTITIONED"."ADDRESS_CITY"='Michaelmouth' OR "ADDRESSES_COMPOSITE_PARTITIONED"."ADDRESS_CITY"='New
              Michael' OR "ADDRESSES_COMPOSITE_PARTITIONED"."ADDRESS_CITY"='West Anthony')
  21 - access("PRODUCTS"."ID"="ID")
  24 - access("ORDER_PRODUCTS_INDEX"."ORDERS_ID"="ORDERS"."ID")
  26 - access("ORDERS"."CLIENTS_ID"="CLIENTS"."ID")
  27 - access("CLIENTS"."ADDRESSES_ID"="ADDRESSES_COMPOSITE_PARTITIONED"."ID")
  29 - access("A"."CLIENT_ID"="CLIENTS"."ID")
```

Comparison of query plans

Table1 Comparison of query plans

Name/Transaction	**“GET_SUMMED_ORDERS_PRICE_[…]”**	**“GET_TOP_X_CLIENTS_[...] ”**
Query plans	Rows: 43944 Bytes:5278k Cost: 84108	Rows: 1008k Bytes: 304M Cost: 99945
Query plans with **Partitions**	1) Range type partition Rows: 4826 Bytes: 805K Cost: 71003 2) List type partition (Client types) Rows: 81200 Bytes:18M Cost: 74230 3) List type partition (Product materials) Rows: 8101 Bytes: 3014K Cost: 76695 4) Combined partitions Rows: 8101 Bytes:3014K Cost: 67928	1) Hash type partition Rows: 215 Bytes: 68155 Cost: 20037 2) List type partition Rows:6564 Bytes: 2032K Cost: 31698 3) Composite type partition Rows: 215 Bytes: 68155 Cost: 20106

The table shows key details from Query plans, Query plans with indexes and Query plans with Partitions. Comparing CPU % Cost without partitions and with partitions, it can be concluded that using partitions for all tested partition’s type and transaction is effective and reasonable.

Comparison CPU % Cost on the Query - “GET_SUMMED_ORDERS_PRICE_[…]” for query, partition type range is better than other partition types for this example (cost 71003 vs 74230 vs 74230). As can be expected, usage of all partitioned tables at once (Combined partitions) yielded even better results.

Comparison CPU % Cost on the Query - “GET_TOP_X_CLIENTS_[...] ” for queries with Partition types Hash and Composite measured cost is better than List type partition (20037 vs 20106 vs 31698).

Running time comparison

Tests based on query “GET_SUMMED_ORDERS_PRICE_FOR_CLIENTS” 100 Iterations					
	AVG TIME	MIN TIME	MAX TIME	DIFF AVG	% DIFF
BASE	2.05217	2.012	2.239	-----------	-----------
LIST (CLIENT_TYPE)	1.97535	1.931	2.129	-0.07682	-3.74%
LIST(MATERIAL)(10 ITER)	18.4284	18.162	19.49	16.37623	798.00%
RANGE	2.65822	2.597	2.999	0.60605	29.53%
COMBINED (10 ITER)	19.6193	1.835	23.609	17.56713	856.03%

Analyzing average running times of the query (saved in logs) - “GET_SUMMED_ORDERS_PRICE_[…]” with and without partitions it can be concluded that, in this case, adding list type partition (list client_type) was the most successful partitioning method. Other two partitions gave big differences in time measures.

Tests based on query “GET TOP X CLIENTS IN COUNTRIES” 1000 Iterations					
	AVG TIME	MIN TIME	MAX TIME	DIFF AVG	% DIFF
BASE	0.834536	0.802	1.19	-----------	-----------
LIST	0.49756	0.477	0.76	-0.336976	-40.38%
HASH	0.507668	0.48	0.82	-0.326868	-39.17%
COMPOSITE	0.550339	0.482	0.994	-0.284197	-34.05%

Analyzing running times of the query (saved in logs) - “GET_TOP_X_CLIENTS_[...] ” with and without partitions it can be concluded that, with partitions running times were shorter. List partition and Hash partition was the most successful partitioning method.

Conclusions

The greatest improvement for “GET_SUMMED_ORDERS_PRICE_[…]” was partition type list (client type) which improved cost and the average running time.

For “GET TOP X CLIENTS IN COUNTRIES " all suggested partitions improved cost and running time especially list and hash type partition.

We couldn’t reliably measure times of list type partition using product.material and all partitions combined, for unknown reason after the first run which returned times in between 1-3 seconds, all later runs were returning times around 10x greater, despite the query and query plans staying the same.

Out of all partitioning methods the list partitioning method proved to be best for tested queries, yielding highest time savings.

Assignment 10 - Columnar store (spec.)

Given columns were chosen to be changed to columnar store from row oriented system to vertically - columnar oriented in order to improve the speed of selected queries. In a row oriented system whole rows are loaded during query execution, even if only one or two columns are needed by query. Oracle In-Memory Columnar Storage allows loading of only selected columns instead of whole rows for queries. It can save time and computational space for operations.

Tables/columns proposed to represent in columnar storage :

GET_SUMMED_ORDERS_PRICE_FOR_EACH_CLIENT

```
----------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name           | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |                | 43944 |  5278K|       | 84108   (2)| 00:00:04 |
|   1 |  SORT ORDER BY              |                | 43944 |  5278K|    19M| 84108   (2)| 00:00:04 |
|   2 |   HASH GROUP BY             |                | 43944 |  5278K|    19M| 84108   (2)| 00:00:04 |
|*  3 |    HASH JOIN                |                |   148K|    17M|  3016K| 75909   (2)| 00:00:03 |
|*  4 |     TABLE ACCESS FULL       | PRODUCTS       |   110K|  1721K|       |  7476   (1)| 00:00:01 |
|*  5 |     HASH JOIN               |                |  1347K|   137M|    46M| 60682   (2)| 00:00:03 |
|*  6 |      HASH JOIN              |                |   450K|    41M|    11M| 37211   (2)| 00:00:02 |
|*  7 |       HASH JOIN             |                |   138K|  9749K|  8536K| 29996   (2)| 00:00:02 |
|*  8 |        HASH JOIN            |                |   138K|  6905K|    21M| 25429   (2)| 00:00:01 |
|*  9 |         TABLE ACCESS FULL   | ORDERS         |   632K|    14M|       |  5576   (2)| 00:00:01 |
|* 10 |         HASH JOIN           |                |   754K|    19M|       | 17366   (2)| 00:00:01 |
|* 11 |          TABLE ACCESS FULL| PRODUCTS       | 73428 |  1147K|       |  7476   (1)| 00:00:01 |
|  12 |          TABLE ACCESS FULL| ORDER_PRODUCTS |    10M|   108M|       |  9811   (2)| 00:00:01 |
|* 13 |        TABLE ACCESS FULL    | CLIENTS        |   349K|  7172K|       |  3601   (1)| 00:00:01 |
|* 14 |       TABLE ACCESS FULL     | ORDERS         |   632K|    14M|       |  5576   (2)| 00:00:01 |
|  15 |      TABLE ACCESS FULL      | ORDER_PRODUCTS |    10M|   108M|       |  9811   (2)| 00:00:01 |
----------------------------------------------------------------------------------------------------
```

Predicate Information (identified by operation id):

```
---------------------------------------------------

   3 - access("PRODUCTS"."ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
   4 - filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='F')
   5 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
   6 - access("ORDERS"."CLIENTS_ID"="CLIENTS"."ID")
   7 - access("CLIENTS"."ID"="ORDERS"."CLIENTS_ID")
   8 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
   9 - filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss') AND "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss'))
  10 - access("PRODUCTS"."ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
  11 - filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='M')
  13 - filter("CLIENTS"."TYPE_CLIENT"='Good_Client' OR
              "CLIENTS"."TYPE_CLIENT"='Premium_Client')
  14 - filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss') AND "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss'))
```

As can be seen in the above query plan, GET_SUMMED_ORDERS_PRICE_FOR_EACH_CLIENT relies only on a few columns from tables CLIENTS, ORDERS and PRODUCTS. ORDER_PRODUCTS is used fully due to it being a many to many linking table.
Due to that, columns listed below will be changed to be stored as In-Memory Compression Unit (IMCU):

- CLIENTS.TYPE_CLIENT
- PRODUCTS.MATERIAL
- ORDERS.CREATION_DATE
- ORDERS.PRICE

ORDERS.PRICE is added here, as it isn’t listed in above query plan and it is used to calculate summed orders price.
This query also has a lot of hash joins, which will be a good opportunity to experiment with in-memory join groups for above tables and ORDER_PRODUCTS table.

GET_SUMMARY_OF_SALES

```
---------------------------------------------------------------------------------------------
| Id  | Operation               | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------,----------------------------------------
|   0 | SELECT STATEMENT        |           |   201K|    14M|       | 50222   (2)| 00:00:02 |
|   1 |  SORT ORDER BY          |           |   201K|    14M|    17M| 50222   (2)| 00:00:02 |
|   2 |   HASH GROUP BY         |           |   201K|    14M|    17M| 50222   (2)| 00:00:02 |
|*  3 |    HASH JOIN RIGHT SEMI |           |   201K|    14M|  7584K| 42965   (3)| 00:00:02 |
|*  4 |     TABLE ACCESS FULL   | ADDRESSES |   204K|  5184K|       | 13789   (3)| 00:00:01 |
|*  5 |     HASH JOIN           |           |   201K|     9M|  6096K| 28203   (3)| 00:00:02 |
|   6 |      VIEW               | VW_GBC_5  |   201K|  3733K|       | 13688   (3)| 00:00:01 |
|   7 |       HASH GROUP BY     |           |   201K|  3733K|   105M| 13688   (3)| 00:00:01 |
|*  8 |        TABLE ACCESS FULL| ORDERS    |  3432K|    62M|       |  5596   (2)| 00:00:01 |
|*  9 |      TABLE ACCESS FULL  | ADDRESSES |   204K|  6381K|       | 13789   (3)| 00:00:01 |
---------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("ADDRESSES"."ADDRESS_CITY"="ADDRESSES"."ADDRESS_CITY")
   4 - filter("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR "ADDRESSES"."ADDRESS_COUNTRY"='Malta'
              OR "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR "ADDRESSES"."ADDRESS_COUNTRY"='Poland'
              OR "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Saudi Arabia' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
   5 - access("ITEM_1"="ADDRESSES"."ID")
   8 - filter("ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss') AND "ORDERS"."CREATION_DATE">=TO_DATE(' 1995-04-29 00:00:00',
              'syyyy-mm-dd hh24:mi:ss'))
   9 - filter("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR "ADDRESSES"."ADDRESS_COUNTRY"='Malta'
              OR "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR "ADDRESSES"."ADDRESS_COUNTRY"='Poland'
              OR "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Saudi Arabia' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
```

As can be seen in the above query plan, GET_SUMMARY_OF_SALES relies columns from tables ADDRESSES and ORDERS can be used as columnar storage. Columns listed below will be changed to be stored as IMCU:

- ADDRESSES.ADDRESS_COUNTRY
- ORDERS.CREATION_DATE
- ORDERS.PRICE

ORDERS.PRICE is added here, as it isn't listed in above query plan and it is used to calculate summed orders price for countries.

GET_TOP_X_CLIENTS_FROM_COUNTRIES_WITH_HIGHEST_AMOUNT_OF_PRODUCTS

```
--------------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |                 |   215 | 68155 | 33047   (2)| 00:00:02 |
|   1 |  NESTED LOOPS                          |                 |   215 | 68155 | 33047   (2)| 00:00:02 |
|   2 |   NESTED LOOPS                         |                 |   215 | 68155 | 33047   (2)| 00:00:02 |
|*  3 |    VIEW                                |                 |   215 | 63855 | 32617   (2)| 00:00:02 |
|*  4 |     WINDOW SORT PUSHED RANK            |                 |   215 | 16125 | 32617   (2)| 00:00:02 |
|   5 |      HASH GROUP BY                     |                 |   215 | 16125 | 32617   (2)| 00:00:02 |
|   6 |       NESTED LOOPS                     |                 |   215 | 16125 | 32616   (2)| 00:00:02 |
|   7 |        NESTED LOOPS                    |                 |   215 | 16125 | 32616   (2)| 00:00:02 |
|   8 |         NESTED LOOPS                   |                 |   215 | 12255 | 32186   (2)| 00:00:02 |
|   9 |          NESTED LOOPS                  |                 |   215 |  6450 | 31756   (2)| 00:00:02 |
|* 10 |           HASH JOIN RIGHT SEMI         |                 |   216 |  4104 | 31324   (2)| 00:00:02 |
|  11 |            VIEW                        | VW_NSO_1        |    21 |   105 | 21419   (1)| 00:00:01 |
|* 12 |             HASH JOIN                  |                 |    21 |  1113 | 21419   (1)| 00:00:01 |
|* 13 |              HASH JOIN                 |                 |     2 |    86 | 13949   (2)| 00:00:01 |
|* 14 |               TABLE ACCESS FULL        | ADDRESSES       |     2 |    64 | 13674   (2)| 00:00:01 |
|  15 |               TABLE ACCESS FULL        | MANUFACTURERS   |   100K|  1074K|   274   (1)| 00:00:01 |
|  16 |              TABLE ACCESS FULL       | PRODUCTS        |  1000K|  9765K|  7462   (1)| 00:00:01 |
|  17 |            TABLE ACCESS FULL         | ORDER_PRODUCTS  |    10M|   137M|  9827   (2)| 00:00:01 |
|  18 |           TABLE ACCESS BY INDEX ROWID| ORDERS          |      1 |    11 |      2   (0)| 00:00:01 |
|* 19 |            INDEX UNIQUE SCAN         | ORDERS_PK       |      1 |       |      1   (0)| 00:00:01 |
|  20 |          TABLE ACCESS BY INDEX ROWID | CLIENTS         |      1 |    27 |      2   (0)| 00:00:01 |
|* 21 |           INDEX UNIQUE SCAN          | CLIENTS_PK      |      1 |       |      1   (0)| 00:00:01 |
|* 22 |         INDEX UNIQUE SCAN            | ADDRESSES_PK    |      1 |       |      1   (0)| 00:00:01 |
|  23 |        TABLE ACCESS BY INDEX ROWID   | ADDRESSES       |      1 |    18 |      2   (0)| 00:00:01 |
|* 24 |    INDEX UNIQUE SCAN                 | CLIENTS_PK      |      1 |       |      1   (0)| 00:00:01 |
|  25 |   TABLE ACCESS BY INDEX ROWID        | CLIENTS         |      1 |    20 |      2   (0)| 00:00:01 |
---------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("A"."RNK"<=20)
   4 - filter(DENSE_RANK() OVER ( PARTITION BY "ADDRESSES"."ADDRESS_COUNTRY" ORDER BY
              SUM("ORDER_PRODUCTS"."AMOUNT") DESC )<=20)
  10 - access("ORDER_PRODUCTS"."PRODUCTS_ID"="ID")
  12 - access("MANUFACTURERS"."ID"="PRODUCTS"."MANUFACTURERS_ID")
  13 - access("ADDRESSES"."ID"="MANUFACTURERS"."ADDRESSES_ID")
  14 - filter(("ADDRESSES"."ADDRESS_CITY"='Michaelmouth' OR "ADDRESSES"."ADDRESS_CITY"='New
              Michael' OR "ADDRESSES"."ADDRESS_CITY"='West Anthony') AND
              ("ADDRESSES"."ADDRESS_COUNTRY"='Austria' OR "ADDRESSES"."ADDRESS_COUNTRY"='Korea' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Mexico'))
  19 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
  21 - access("ORDERS"."CLIENTS_ID"="CLIENTS"."ID")
  22 - access("CLIENTS"."ADDRESSES_ID"="ADDRESSES"."ID")
  24 - access("A"."CLIENT_ID"="CLIENTS"."ID")
```

Above query works mostly on tables MANUFACTURERS and PRODUCTS, there are also two hash joins that are quite costly. Due to that below columns will be stored in memory:

- MANUFACTURERS .ADDRESS_ID
- ADDRESSES.ADDRESS_CITY

Two hash joins (No.12 and 13 in the query plan) might get optimized with in-memory join groups, comparing results with and without join groups will be one of our experiments.

Proposed experiments and comparison:

- comparison original cost and time of operations and with tables/columns in columnar store
- comparison between:
    - running times with and without in-memory join groups
    - different parameters for in-memory columnar storage

Assignment 11 - Columnar Store (dev.)

GET_SUMMED_ORDERS_PRICE_FOR_EACH_CLIENT
Without columnar store

```
----------------------------------------------------------------------------------------------------
| Id  | Operation                 | Name           | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT          |                | 43944 |  5278K|       | 84108   (2)| 00:00:04 |
|   1 |  SORT ORDER BY            |                | 43944 |  5278K|    19M| 84108   (2)| 00:00:04 |
|   2 |   HASH GROUP BY           |                | 43944 |  5278K|    19M| 84108   (2)| 00:00:04 |
|*  3 |    HASH JOIN              |                |   148K|    17M|  3016K| 75909   (2)| 00:00:03 |
|*  4 |     TABLE ACCESS FULL     | PRODUCTS       |   110K|  1721K|       |  7476   (1)| 00:00:01 |
|*  5 |     HASH JOIN             |                |  1347K|   137M|    46M| 60682   (2)| 00:00:03 |
|*  6 |      HASH JOIN            |                |   450K|    41M|    11M| 37211   (2)| 00:00:02 |
|*  7 |       HASH JOIN           |                |   138K|  9749K|  8536K| 29996   (2)| 00:00:02 |
|*  8 |        HASH JOIN          |                |   138K|  6905K|    21M| 25429   (2)| 00:00:01 |
|*  9 |         TABLE ACCESS FULL | ORDERS         |   632K|    14M|       |  5576   (2)| 00:00:01 |
|* 10 |         HASH JOIN         |                |   754K|    19M|       | 17366   (2)| 00:00:01 |
|* 11 |          TABLE ACCESS FULL| PRODUCTS       | 73428 |  1147K|       |  7476   (1)| 00:00:01 |
|  12 |          TABLE ACCESS FULL| ORDER_PRODUCTS |    10M|   108M|       |  9811   (2)| 00:00:01 |
|* 13 |        TABLE ACCESS FULL  | CLIENTS        |   349K|  7172K|       |  3601   (1)| 00:00:01 |
|* 14 |       TABLE ACCESS FULL   | ORDERS         |   632K|    14M|       |  5576   (2)| 00:00:01 |
|  15 |      TABLE ACCESS FULL    | ORDER_PRODUCTS |    10M|   108M|       |  9811   (2)| 00:00:01 |
----------------------------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("PRODUCTS"."ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
   4 - filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='F')
   5 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
   6 - access("ORDERS"."CLIENTS_ID"="CLIENTS"."ID")
   7 - access("CLIENTS"."ID"="ORDERS"."CLIENTS_ID")
   8 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
   9 - filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss') AND "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss'))
  10 - access("PRODUCTS"."ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
  11 - filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='M')
  13 - filter("CLIENTS"."TYPE_CLIENT"='Good_Client' OR
              "CLIENTS"."TYPE_CLIENT"='Premium_Client')
  14 - filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss') AND "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss'))
```

With columnar store

```
---------------------------------------------------------------------------------------------------------------
| Id  | Operation                            | Name             | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
---------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |                  | 43645 |  5242K|       | 84101   (2)| 00:00:04 |
|   1 |  SORT ORDER BY                       |                  | 43645 |  5242K|    19M| 84101   (2)| 00:00:04 |
|   2 |   HASH GROUP BY                      |                  | 43645 |  5242K|    19M| 84101   (2)| 00:00:04 |
|*  3 |    HASH JOIN                         |                  |   148K|    17M|  3016K| 75901   (2)| 00:00:03 |
   |* 4 |     TABLE ACCESS INMEMORY FULL       | PRODUCTS       |  110K|  1721K|       |  7476   (1)| 00:00:01 |
   |* 5 |     HASH JOIN                        |                | 1347K|   137M|    46M| 60675   (2)| 00:00:03 |
   |* 6 |      HASH JOIN                       |                |  450K|    41M|    11M| 37204   (2)| 00:00:02 |
   |  7 |       JOIN FILTER CREATE             | :BF0000        |  138K|  9749K|       | 29989   (2)| 00:00:02 |
   |* 8 |        HASH JOIN                     |                |  138K|  9749K|  8536K| 29989   (2)| 00:00:02 |
   |  9 |         JOIN FILTER CREATE           | :BF0001        |  138K|  9749K|       | 29989   (2)| 00:00:02 |
   |* 10 |          HASH JOIN                   |                |  138K|  6905K|    21M| 25429   (2)| 00:00:01 |
   |* 11 |           TABLE ACCESS INMEMORY FULL | ORDERS         |  632K|    14M|       |  5576   (2)| 00:00:01 |
   |* 12 |           HASH JOIN                  |                |  754K|    19M|       | 17366   (2)| 00:00:01 |
   |* 13 |            TABLE ACCESS INMEMORY FULL| PRODUCTS       | 73428 |  1147K|       |  7476   (1)| 00:00:01 |
   |  14 |            TABLE ACCESS FULL         | ORDER_PRODUCTS |   10M|   108M|       |  9811   (2)| 00:00:01 |
   |  15 |         JOIN FILTER USE              | :BF0001        |  345K|  7077K|       |  3601   (1)| 00:00:01 |
   |* 16 |          TABLE ACCESS INMEMORY FULL  | CLIENTS        |  345K|  7077K|       |  3601   (1)| 00:00:01 |
   |  17 |       JOIN FILTER USE                | :BF0000        |  632K|    14M|       |  5576   (2)| 00:00:01 |
   |* 18 |        TABLE ACCESS INMEMORY FULL    | ORDERS         |  632K|    14M|       |  5576   (2)| 00:00:01 |
   |  19 |      TABLE ACCESS FULL               | ORDER_PRODUCTS |   10M|   108M|       |  9811   (2)| 00:00:01 |
   ---------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("PRODUCTS"."ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
   4 - inmemory("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='F')
       filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='F')
   5 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
   6 - access("ORDERS"."CLIENTS_ID"="CLIENTS"."ID")
   8 - access("CLIENTS"."ID"="ORDERS"."CLIENTS_ID")
  10 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
  11 - inmemory("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss')
              AND "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
       filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
  12 - access("PRODUCTS"."ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
  13 - inmemory("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='M')
       filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='M')
  16 - inmemory(("CLIENTS"."TYPE_CLIENT"='Good_Client' OR "CLIENTS"."TYPE_CLIENT"='Premium_Client')
              AND SYS_OP_BLOOM_FILTER(:BF0001,"CLIENTS"."ID"))
       filter(("CLIENTS"."TYPE_CLIENT"='Good_Client' OR "CLIENTS"."TYPE_CLIENT"='Premium_Client') AND
              SYS_OP_BLOOM_FILTER(:BF0001,"CLIENTS"."ID"))
  18 - inmemory("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss')
              AND "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              SYS_OP_BLOOM_FILTER(:BF0000,"ORDERS"."CLIENTS_ID"))
       filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              SYS_OP_BLOOM_FILTER(:BF0000,"ORDERS"."CLIENTS_ID"))
```

GET_SUMMARY_OF_SALES

Without columnar store

```
 -----------------------------------------------------------------------------------------
 | Id  | Operation              | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
------------------------------------------------,-----------------------------------------
 |   0 | SELECT STATEMENT       |           |   201K|    14M|       | 50222   (2)| 00:00:02 |
 |   1 |  SORT ORDER BY         |           |   201K|    14M|    17M| 50222   (2)| 00:00:02 |
 |   2 |   HASH GROUP BY        |           |   201K|    14M|    17M| 50222   (2)| 00:00:02 |
 |*  3 |    HASH JOIN RIGHT SEMI |           |   201K|    14M|  7584K| 42965   (3)| 00:00:02 |
 |*  4 |     TABLE ACCESS FULL  | ADDRESSES |   204K|  5184K|       | 13789   (3)| 00:00:01 |
 |*  5 |     HASH JOIN          |           |   201K|     9M|  6096K| 28203   (3)| 00:00:02 |
 |   6 |      VIEW              | VW_GBC_5  |   201K|  3733K|       | 13688   (3)| 00:00:01 |
 |   7 |       HASH GROUP BY    |           |   201K|  3733K|   105M| 13688   (3)| 00:00:01 |
                  |*  8 |        TABLE ACCESS FULL| ORDERS    |  3432K|    62M|       |  5596   (2)| 00:00:01 |
                  |*  9 |      TABLE ACCESS FULL  | ADDRESSES |   204K|  6381K|       | 13789   (3)| 00:00:01 |
                  ---------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("ADDRESSES"."ADDRESS_CITY"="ADDRESSES"."ADDRESS_CITY")
   4 - filter("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR "ADDRESSES"."ADDRESS_COUNTRY"='Malta'
              OR "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR "ADDRESSES"."ADDRESS_COUNTRY"='Poland'
              OR "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Saudi Arabia' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
   5 - access("ITEM_1"="ADDRESSES"."ID")
   8 - filter("ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss') AND "ORDERS"."CREATION_DATE">=TO_DATE(' 1995-04-29 00:00:00',
              'syyyy-mm-dd hh24:mi:ss'))
   9 - filter("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR "ADDRESSES"."ADDRESS_COUNTRY"='Malta'
              OR "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR "ADDRESSES"."ADDRESS_COUNTRY"='Poland'
              OR "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Saudi Arabia' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
```

With columnar store

```
             ------------------------------------------------------------------------------------------------------
```

```
| Id  | Operation                         | Name       | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                  |            |   201K|    14M|       | 50222   (2)| 00:00:02 |
|   1 |  SORT ORDER BY                    |            |   201K|    14M|    17M| 50222   (2)| 00:00:02 |
|   2 |   HASH GROUP BY                   |            |   201K|    14M|    17M| 50222   (2)| 00:00:02 |
|*  3 |    HASH JOIN RIGHT SEMI           |            |   201K|    14M|  7584K| 42965   (3)| 00:00:02 |
|*  4 |     TABLE ACCESS INMEMORY FULL    | ADDRESSES  |   204K|  5184K|       | 13789   (3)| 00:00:01 |
|*  5 |     HASH JOIN                     |            |   201K|     9M|  6096K| 28203   (3)| 00:00:02 |
|   6 |      VIEW                         | VW_GBC_5   |   201K|  3733K|       | 13688   (3)| 00:00:01 |
|   7 |       HASH GROUP BY               |            |   201K|  3733K|   105M| 13688   (3)| 00:00:01 |
|*  8 |        TABLE ACCESS INMEMORY FULL| ORDERS     |  3432K|    62M|       |  5596   (2)| 00:00:01 |
|*  9 |      TABLE ACCESS INMEMORY FULL   | ADDRESSES  |   204K|  6381K|       | 13789   (3)| 00:00:01 |
------------------------------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("ADDRESSES"."ADDRESS_CITY"="ADDRESSES"."ADDRESS_CITY")
   4 - inmemory("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR "ADDRESSES"."ADDRESS_COUNTRY"='Malta' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR "ADDRESSES"."ADDRESS_COUNTRY"='Poland' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR "ADDRESSES"."ADDRESS_COUNTRY"='Saudi
              Arabia' OR "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
       filter("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR "ADDRESSES"."ADDRESS_COUNTRY"='Malta' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR "ADDRESSES"."ADDRESS_COUNTRY"='Poland' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR "ADDRESSES"."ADDRESS_COUNTRY"='Saudi
              Arabia' OR "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
   5 - access("ITEM_1"="ADDRESSES"."ID")
   8 - inmemory("ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss') AND "ORDERS"."CREATION_DATE">=TO_DATE(' 1995-04-29 00:00:00', 'syyyy-mm-dd
```

```
           hh24:mi:ss'))
    filter("ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd
           hh24:mi:ss') AND "ORDERS"."CREATION_DATE">=TO_DATE(' 1995-04-29 00:00:00', 'syyyy-mm-dd
           hh24:mi:ss'))
9 - inmemory("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR
           "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR
           "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR "ADDRESSES"."ADDRESS_COUNTRY"='Malta' OR
           "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR "ADDRESSES"."ADDRESS_COUNTRY"='Poland' OR
           "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR "ADDRESSES"."ADDRESS_COUNTRY"='Saudi
           Arabia' OR "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
    filter("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR
           "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR
           "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR "ADDRESSES"."ADDRESS_COUNTRY"='Malta' OR
           "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR "ADDRESSES"."ADDRESS_COUNTRY"='Poland' OR
           "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR "ADDRESSES"."ADDRESS_COUNTRY"='Saudi
           Arabia' OR "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
```

GET_TOP_X_CLIENTS_FROM_COUNTRIES_WITH_HIGHEST_AMOUNT_OF_PRODUCTS

Without columnar store

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation                             | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |                |   215 | 68155 | 33047   (2)| 00:00:02 |
|   1 |  NESTED LOOPS                         |                |   215 | 68155 | 33047   (2)| 00:00:02 |
|   2 |   NESTED LOOPS                        |                |   215 | 68155 | 33047   (2)| 00:00:02 |
|*  3 |    VIEW                               |                |   215 | 63855 | 32617   (2)| 00:00:02 |
|*  4 |     WINDOW SORT PUSHED RANK           |                |   215 | 16125 | 32617   (2)| 00:00:02 |
|   5 |      HASH GROUP BY                    |                |   215 | 16125 | 32617   (2)| 00:00:02 |
|   6 |       NESTED LOOPS                    |                |   215 | 16125 | 32616   (2)| 00:00:02 |
|   7 |        NESTED LOOPS                   |                |   215 | 16125 | 32616   (2)| 00:00:02 |
|   8 |         NESTED LOOPS                  |                |   215 | 12255 | 32186   (2)| 00:00:02 |
|   9 |          NESTED LOOPS                 |                |   215 |  6450 | 31756   (2)| 00:00:02 |
|* 10 |           HASH JOIN RIGHT SEMI        |                |   216 |  4104 | 31324   (2)| 00:00:02 |
|  11 |            VIEW                       | VW_NSO_1       |    21 |   105 | 21419   (1)| 00:00:01 |
|* 12 |             HASH JOIN                 |                |    21 |  1113 | 21419   (1)| 00:00:01 |
|* 13 |              HASH JOIN                |                |     2 |    86 | 13949   (2)| 00:00:01 |
|* 14 |               TABLE ACCESS FULL       | ADDRESSES      |     2 |    64 | 13674   (2)| 00:00:01 |
|  15 |               TABLE ACCESS FULL       | MANUFACTURERS  |   100K|  1074K|   274   (1)| 00:00:01 |
|  16 |              TABLE ACCESS FULL        | PRODUCTS       |  1000K|  9765K|  7462   (1)| 00:00:01 |
|  17 |            TABLE ACCESS FULL          | ORDER_PRODUCTS |    10M|   137M|  9827   (2)| 00:00:01 |
|  18 |           TABLE ACCESS BY INDEX ROWID| ORDERS         |     1 |    11 |     2   (0)| 00:00:01 |
|* 19 |            INDEX UNIQUE SCAN          | ORDERS_PK      |     1 |       |     1   (0)| 00:00:01 |
|  20 |          TABLE ACCESS BY INDEX ROWID | CLIENTS        |     1 |    27 |     2   (0)| 00:00:01 |
|* 21 |           INDEX UNIQUE SCAN           | CLIENTS_PK     |     1 |       |     1   (0)| 00:00:01 |
|* 22 |         INDEX UNIQUE SCAN             | ADDRESSES_PK   |     1 |       |     1   (0)| 00:00:01 |
|  23 |        TABLE ACCESS BY INDEX ROWID    | ADDRESSES      |     1 |    18 |     2   (0)| 00:00:01 |
|* 24 |    INDEX UNIQUE SCAN                  | CLIENTS_PK     |     1 |       |     1   (0)| 00:00:01 |
|  25 |   TABLE ACCESS BY INDEX ROWID         | CLIENTS        |     1 |    20 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("A"."RNK"<=20)
   4 - filter(DENSE_RANK() OVER ( PARTITION BY "ADDRESSES"."ADDRESS_COUNTRY" ORDER BY
              SUM("ORDER_PRODUCTS"."AMOUNT") DESC )<=20)
  10 - access("ORDER_PRODUCTS"."PRODUCTS_ID"="ID")
  12 - access("MANUFACTURERS"."ID"="PRODUCTS"."MANUFACTURERS_ID")
  13 - access("ADDRESSES"."ID"="MANUFACTURERS"."ADDRESSES_ID")
  14 - filter(("ADDRESSES"."ADDRESS_CITY"='Michaelmouth' OR "ADDRESSES"."ADDRESS_CITY"='New
              Michael' OR "ADDRESSES"."ADDRESS_CITY"='West Anthony') AND
              ("ADDRESSES"."ADDRESS_COUNTRY"='Austria' OR "ADDRESSES"."ADDRESS_COUNTRY"='Korea' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Mexico'))
  19 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
  21 - access("ORDERS"."CLIENTS_ID"="CLIENTS"."ID")
  22 - access("CLIENTS"."ADDRESSES_ID"="ADDRESSES"."ID")
  24 - access("A"."CLIENT_ID"="CLIENTS"."ID")
```

With columnar store

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                                   | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                            |                  |   215 | 68155 | 33047   (2)| 00:00:02 |
|   1 |  NESTED LOOPS                               |                  |   215 | 68155 | 33047   (2)| 00:00:02 |
|   2 |   NESTED LOOPS                              |                  |   215 | 68155 | 33047   (2)| 00:00:02 |
|*  3 |    VIEW                                     |                  |   215 | 63855 | 32617   (2)| 00:00:02 |
|*  4 |     WINDOW SORT PUSHED RANK                 |                  |   215 | 16125 | 32617   (2)| 00:00:02 |
|   5 |      HASH GROUP BY                          |                  |   215 | 16125 | 32617   (2)| 00:00:02 |
|   6 |       NESTED LOOPS                          |                  |   215 | 16125 | 32616   (2)| 00:00:02 |
|   7 |        NESTED LOOPS                         |                  |   215 | 16125 | 32616   (2)| 00:00:02 |
|   8 |         NESTED LOOPS                        |                  |   215 | 12255 | 32186   (2)| 00:00:02 |
|   9 |          NESTED LOOPS                       |                  |   215 |  6450 | 31756   (2)| 00:00:02 |
           |* 10 |           HASH JOIN RIGHT SEMI           |                |   216 |  4104 | 31324   (2)| 00:00:02 |
           |  11 |            VIEW                          | VW_NSO_1       |    21 |   105 | 21419   (1)| 00:00:01 |
           |* 12 |             HASH JOIN                    |                |    21 |  1113 | 21419   (1)| 00:00:01 |
           |* 13 |              HASH JOIN                   |                |     2 |    86 | 13949   (2)| 00:00:01 |
           |  14 |               JOIN FILTER CREATE         | :BF0000        |     2 |    64 | 13674   (2)| 00:00:01 |
           |* 15 |                TABLE ACCESS INMEMORY FULL| ADDRESSES      |     2 |    64 | 13674   (2)| 00:00:01 |
           |  16 |               JOIN FILTER USE            | :BF0000        |   100K|  1074K|   274   (1)| 00:00:01 |
           |* 17 |                TABLE ACCESS INMEMORY FULL| MANUFACTURERS  |   100K|  1074K|   274   (1)| 00:00:01 |
           |  18 |              TABLE ACCESS FULL           | PRODUCTS       |  1000K|  9765K|  7462   (1)| 00:00:01 |
           |  19 |            TABLE ACCESS FULL             | ORDER_PRODUCTS |    10M|   137M|  9827   (2)| 00:00:01 |
           |  20 |           TABLE ACCESS BY INDEX ROWID    | ORDERS         |     1 |    11 |     2   (0)| 00:00:01 |
           |* 21 |            INDEX UNIQUE SCAN             | ORDERS_PK      |     1 |       |     1   (0)| 00:00:01 |
           |  22 |          TABLE ACCESS BY INDEX ROWID     | CLIENTS        |     1 |    27 |     2   (0)| 00:00:01 |
           |* 23 |           INDEX UNIQUE SCAN              | CLIENTS_PK     |     1 |       |     1   (0)| 00:00:01 |
           |* 24 |         INDEX UNIQUE SCAN                | ADDRESSES_PK   |     1 |       |     1   (0)| 00:00:01 |
           |  25 |        TABLE ACCESS BY INDEX ROWID       | ADDRESSES      |     1 |    18 |     2   (0)| 00:00:01 |
           |* 26 |    INDEX UNIQUE SCAN                     | CLIENTS_PK     |     1 |       |     1   (0)| 00:00:01 |
           |  27 |   TABLE ACCESS BY INDEX ROWID            | CLIENTS        |     1 |    20 |     2   (0)| 00:00:01 |
           -----------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("A"."RNK"<=20)
   4 - filter(DENSE_RANK() OVER ( PARTITION BY "ADDRESSES"."ADDRESS_COUNTRY" ORDER BY
              SUM("ORDER_PRODUCTS"."AMOUNT") DESC )<=20)
  10 - access("ORDER_PRODUCTS"."PRODUCTS_ID"="ID")
  12 - access("MANUFACTURERS"."ID"="PRODUCTS"."MANUFACTURERS_ID")
  13 - access("ADDRESSES"."ID"="MANUFACTURERS"."ADDRESSES_ID")
  15 - inmemory(("ADDRESSES"."ADDRESS_CITY"='Michaelmouth' OR "ADDRESSES"."ADDRESS_CITY"='New
              Michael' OR "ADDRESSES"."ADDRESS_CITY"='West Anthony') AND
              ("ADDRESSES"."ADDRESS_COUNTRY"='Austria' OR "ADDRESSES"."ADDRESS_COUNTRY"='Korea' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Mexico'))
```

```
       filter(("ADDRESSES"."ADDRESS_CITY"='Michaelmouth' OR "ADDRESSES"."ADDRESS_CITY"='New
              Michael' OR "ADDRESSES"."ADDRESS_CITY"='West Anthony') AND
              ("ADDRESSES"."ADDRESS_COUNTRY"='Austria' OR "ADDRESSES"."ADDRESS_COUNTRY"='Korea' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Mexico'))
  17 - inmemory(SYS_OP_BLOOM_FILTER(:BF0000,"MANUFACTURERS"."ADDRESSES_ID"))
       filter(SYS_OP_BLOOM_FILTER(:BF0000,"MANUFACTURERS"."ADDRESSES_ID"))
  21 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
  23 - access("ORDERS"."CLIENTS_ID"="CLIENTS"."ID")
  24 - access("CLIENTS"."ADDRESSES_ID"="ADDRESSES"."ID")
  26 - access("A"."CLIENT_ID"="CLIENTS"."ID")
```

Columnar store and In Memory Join Groups were used on chosen transactions in order to test whether better running times are possible (lower cost of operations and better time).

Comparing query plans for chosen queries with and without columnar store method it can be said that no improvement was registered. There was no registered change for the number of bytes, rows and Cost %cpu. The running time of queries changed for the worse.

Running time comparison

Measured times were presented in tables below. They were measured for queries: GET_SUMMED_ORDERS_PRICE_FOR_CLIENTS", "GET_SUMMARY_OF_SALES", "GET TOP X CLIENTS IN COUNTRIES". Running times were calculated based on 50 iterations. Average measured time was similar for different methods (without and with columnar store))

Times improved for both, low and high memcompress for "GET TOP X CLIENTS IN COUNTRIES" but the difference is negligible and could just be a result of a chance.

Tests based on query "GET_SUMMED_ORDERS_PRICE_FOR_CLIENTS" 50 Iterations					
	AVG TIME	**MIN TIME**	**MAX TIME**	**DIFF AVG**	**% DIFF**
BASE	17.26134	16.091	37.65	-----------	-----------
MEMCOMPRESS LOW	15.54418	15.099	16.146	-1.71716	-9.95%
MEMCOMPRESS HIGH	14.69904	14.159	16.423	-2.5623	-14.84%

Tests based on query “GET_SUMMARY_OF_SALES” 50 Iterations					
	AVG TIME	**MIN TIME**	**MAX TIME**	**DIFF AVG**	**% DIFF**
BASE	10.89988	10.599	13.04	-----------	-----------
MEMCOMPRESS LOW	10.11864	9.794	10.642	-0.78124	-7.17%
MEMCOMPRESS HIGH	10.2111	9.843	12.662	-0.68878	-6.32%

Tests based on query “GET TOP X CLIENTS IN COUNTRIES” 50 Iterations					
	AVG TIME	**MIN TIME**	**MAX TIME**	**DIFF AVG**	**% DIFF**
BASE	10.17358	9.851	10.591	-----------	-----------
MEMCOMPRESS LOW	9.6034	9.367	10.25	-0.57018	-5.60%
MEMCOMPRESS HIGH	10.51178	9.425	32.003	0.3382	3.32%

Assignment 12 - Summary (spec.)

All optimization methods were used in order to improve the working of databases, experiment with them and learn about their function. Queries were optimized using indexes, partitions and columnar storage.
Table created to summarize works and experiments from previous chapters is presented below. Table presents key measurements: average execution times and cost of queries for optimization methods and particular Transactions.

Indexes method improved some of the queries. The measured times improved with indexes: BITMAP, COMP B-TREE, COMP BITMAP. Best results were given for GET SUMMED ORDERS PRICE FOR EACH CLIENT WITH DATE AND PRICE RANGE and GET SUMMARY OF SALES - INCOME IN GIVEN TIME PERIOD, COUNTRIES AND CITIES both types COMPOSITE BITMAP.

Partition methods were used on two different Queries with different types of Partitions. The greatest improvement for “GET_SUMMED_ORDERS_PRICE_[…]” was partition type list (client type) which improved cost and the average running time. For “GET TOP X CLIENTS IN COUNTRIES '' all suggested partitions improved cost and running time especially list and hash type partition.

Columnar storage method although run several times didn’t give promising results. Table contains best from “worst” measurements. Comparing query plans for chosen queries with and without columnar store method it can be said that no improvement was registered. There was no registered change for the number of bytes, rows and Cost %cpu. The running time of queries changed for the worse.

TRANSACTION / METHOD	BASE - NO METHOD	INDEXES	PARTITIONS	COLUMNAR STORAGE
GET TOP ‘X’ CLIENTS FROM SPECIFIC COUNTRY WITH HIGHEST AMOUNT OF PRODUCTS ORDERED FROM SPECIFIC MANUFACTURER	avg time: 0.834536 costs: 33047 (% cpu)		best - list avg time: 0.49756 costs: 31698 best - hash avg time: 0.507668 cost: 20037 (% cpu)	best - memcompress low avg time: 9.6034 costs: 33047 (%cpu)
GET SUMMED ORDERS PRICE FOR EACH CLIENT WITH DATE AND PRICE RANGE	avg time: 2.21578 costs: 84108 (% cpu)	best - COMPOSITE BITMAP avg time: 2.46789 costs: 75878 (% cpu)	best - List (Client types) avg time: 1.97535 costs: 74230 (%cpu)	best - memcompress high avg time: 14.69904 costs: 84101 (%cpu)
GET SUMMARY OF SALES - INCOME IN GIVEN TIME PERIOD, COUNTRIES AND CITIES	avg time: 1.5459 costs: 50222 (% cpu)	best - COMPOSITE BITMAP avg time: 0.79979 costs: 47750 (% cpu)		best - memcompress low avg time: 10.11864 costs: 50222 (%cpu)
GET MOST ORDERED PRODUCT IN EACH COUNTRY IN THE SPECIFIED TIME PERIOD	avg time: 10.48832 costs: 97345 (% cpu)			
GET CLIENT WITH MOST ORDERS FOR SPECIFIED PRODUCT IN SPECIFIC TIME PERIOD	avg time: 2.75161 costs: 63810 (% cpu)			

Percentage improvement in query running times in relation to base running times

QUERY	INDEXES	PARTITIONS	COLUMNAR STORAGE
GET TOP ‘X’ CLIENTS FROM SPECIFIC COUNTRY WITH HIGHEST AMOUNT OF PRODUCTS ORDERED FROM SPECIFIC MANUFACTURER	---------	-40.38%	-5.60%
GET MOST ORDERED PRODUCT IN EACH COUNTRY IN THE SPECIFIED TIME PERIOD	1.46%	---------	---------
GET SUMMED ORDERS PRICE FOR EACH CLIENT WITH DATE AND PRICE RANGE	11.38%	-3.74%	-14.84%
GET SUMMARY OF SALES - INCOME IN GIVEN TIME PERIOD, COUNTRIES AND CITIES	-48.26%	---------	-7.17%

Queries highlighted in green in the above table have shown good improvements with optimization methods tested by us. Using those methods together could yield significant time savings for the queries highlighted above. In the development stage queries listed below will be tested with combination of the most successful optimization methods:

- GET TOP ‘X’ CLIENTS FROM SPECIFIC COUNTRY WITH HIGHEST AMOUNT OF PRODUCTS
  - Hash partition
  - Columnar store with low memory compression
- GET SUMMED ORDERS PRICE FOR EACH CLIENT WITH DATE AND PRICE RANGE
  - List partition using column CLIENTS.CLIENT_TYPE
  - Columnar store with high memory compression
- GET SUMMARY OF SALES - INCOME IN GIVEN TIME PERIOD, COUNTRIES AND CITIES
  - Composite Bitmap Index on columns ADDRESSES.ADDRESS_COUNTRY and ADDRESSES.ADDRESS_CITY
  - Columnar store with low memory compression

Assignment 13 - Summary (dev.)

The aim of the project was to create a database for an online shoe store and also to experiment with different optimization methods. Database was created using oracle database sql developer. After creating a generic database schema, transactions/operations running on the database were created. Then query plans, showing query performance data, were generated from oracle. The results were then saved in documentation. Later, the optimization methods were introduced and experimentation was made. Optimization methods were indexes, partitions and columnar stores. Last step was to summarize and compare given results and to repeat tests for best methods and draw conclusions.

QUERY PLANS

GET_SUMMED_ORDERS_PRICE_FOR_EACH_CLIENT

Without optimization

```
----------------------------------------------------------------------------------------------------
| Id  | Operation                 | Name           | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT          |                | 43944 |  5278K|       | 84108   (2)| 00:00:04 |
|   1 |  SORT ORDER BY            |                | 43944 |  5278K|    19M| 84108   (2)| 00:00:04 |
|   2 |   HASH GROUP BY           |                | 43944 |  5278K|    19M| 84108   (2)| 00:00:04 |
|*  3 |    HASH JOIN              |                |   148K|    17M|  3016K| 75909   (2)| 00:00:03 |
|*  4 |     TABLE ACCESS FULL     | PRODUCTS       |   110K|  1721K|       |  7476   (1)| 00:00:01 |
|*  5 |     HASH JOIN             |                |  1347K|   137M|    46M| 60682   (2)| 00:00:03 |
|*  6 |      HASH JOIN            |                |   450K|    41M|    11M| 37211   (2)| 00:00:02 |
|*  7 |       HASH JOIN           |                |   138K|  9749K|  8536K| 29996   (2)| 00:00:02 |
|*  8 |        HASH JOIN          |                |   138K|  6905K|    21M| 25429   (2)| 00:00:01 |
|*  9 |         TABLE ACCESS FULL | ORDERS         |   632K|    14M|       |  5576   (2)| 00:00:01 |
|* 10 |         HASH JOIN         |                |   754K|    19M|       | 17366   (2)| 00:00:01 |
|* 11 |          TABLE ACCESS FULL| PRODUCTS       | 73428 |  1147K|       |  7476   (1)| 00:00:01 |
|  12 |          TABLE ACCESS FULL| ORDER_PRODUCTS |    10M|   108M|       |  9811   (2)| 00:00:01 |
|* 13 |        TABLE ACCESS FULL  | CLIENTS        |   349K|  7172K|       |  3601   (1)| 00:00:01 |
|* 14 |       TABLE ACCESS FULL   | ORDERS         |   632K|    14M|       |  5576   (2)| 00:00:01 |
|  15 |      TABLE ACCESS FULL    | ORDER_PRODUCTS |    10M|   108M|       |  9811   (2)| 00:00:01 |
----------------------------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("PRODUCTS"."ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
   4 - filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='F')
   5 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
   6 - access("ORDERS"."CLIENTS_ID"="CLIENTS"."ID")
   7 - access("CLIENTS"."ID"="ORDERS"."CLIENTS_ID")
   8 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
   9 - filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss') AND "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss'))
  10 - access("PRODUCTS"."ID"="ORDER_PRODUCTS"."PRODUCTS_ID")
  11 - filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='M')
  13 - filter("CLIENTS"."TYPE_CLIENT"='Good_Client' OR
              "CLIENTS"."TYPE_CLIENT"='Premium_Client')
  14 - filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss') AND "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss'))
```

Optimized

```
-----------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                            | Name                      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     | Pstart| Pstop |
-----------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |                           | 28568 |  3431K|       | 87265   (1)| 00:00:04 |       |       |
|   1 |  SORT ORDER BY                       |                           | 28568 |  3431K|    27M| 87265   (1)| 00:00:04 |       |       |
|   2 |   HASH GROUP BY                      |                           | 28568 |  3431K|    27M| 87265   (1)| 00:00:04 |       |       |
|*  3 |    HASH JOIN                         |                           |   208K|    24M|  2976K| 78550   (2)| 00:00:04 |       |       |
|*  4 |     TABLE ACCESS INMEMORY FULL       | PRODUCTS                  |   108K|  1698K|       |  7475   (1)| 00:00:01 |       |       |
|*  5 |     HASH JOIN                        |                           |  1924K|   196M|    66M| 60067   (2)| 00:00:03 |       |       |
|*  6 |      HASH JOIN                       |                           |   642K|    58M|    10M| 35153   (2)| 00:00:02 |       |       |
|   7 |       JOIN FILTER CREATE             | :BF0000                   |   131K|  9237K|       | 27968   (2)| 00:00:02 |       |       |
|*  8 |        HASH JOIN                     |                           |   131K|  9237K|  8088K| 27968   (2)| 00:00:02 |       |       |
|   9 |         JOIN FILTER CREATE           | :BF0001                   |   131K|  9237K|       | 27968   (2)| 00:00:02 |       |       |
|* 10 |          HASH JOIN                   |                           |   131K|  6542K|    21M| 25810   (2)| 00:00:02 |       |       |
|* 11 |           TABLE ACCESS INMEMORY FULL | ORDERS                    |   631K|    14M|       |  5576   (2)| 00:00:01 |       |       |
|* 12 |           HASH JOIN                  |                           |   714K|    18M|       | 17821   (2)| 00:00:01 |       |       |
|* 13 |            TABLE ACCESS INMEMORY FULL| PRODUCTS                  | 69586 |  1087K|       |  7475   (1)| 00:00:01 |       |       |
|  14 |            INDEX FAST FULL SCAN      | ORDER_PRODUCTS_INDEX_PK   |    10M|   108M|       | 10266   (2)| 00:00:01 |       |       |
|  15 |         JOIN FILTER USE              | :BF0001                   |   354K|  7278K|       |  1204   (1)| 00:00:01 |       |       |
|  16 |          PARTITION LIST INLIST       |                           |   354K|  7278K|       |  1204   (1)| 00:00:01 |KEY(I) |KEY(I) |
|* 17 |           TABLE ACCESS INMEMORY FULL | CLIENTS_LIST_PARTITIONED  |   354K|  7278K|       |  1204   (1)| 00:00:01 |KEY(I) |KEY(I) |
|  18 |       JOIN FILTER USE                | :BF0000                   |   631K|    14M|       |  5576   (2)| 00:00:01 |       |       |
|* 19 |        TABLE ACCESS INMEMORY FULL    | ORDERS                    |   631K|    14M|       |  5576   (2)| 00:00:01 |       |       |
|  20 |      INDEX FAST FULL SCAN            | ORDER_PRODUCTS_INDEX_PK   |    10M|   108M|       | 10266   (2)| 00:00:01 |       |       |
-----------------------------------------------------------------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("PRODUCTS"."ID"="ORDER_PRODUCTS_INDEX"."PRODUCTS_ID")
   4 - inmemory("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='F')
```

```
      filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='F')
   5 - access("ORDER_PRODUCTS_INDEX"."ORDERS_ID"="ORDERS"."ID")
   6 - access("ORDERS"."CLIENTS_ID"="CLIENTS_LIST_PARTITIONED"."ID")
   8 - access("CLIENTS_LIST_PARTITIONED"."ID"="ORDERS"."CLIENTS_ID")
  10 - access("ORDER_PRODUCTS_INDEX"."ORDERS_ID"="ORDERS"."ID")
  11 - inmemory("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
       filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
  12 - access("PRODUCTS"."ID"="ORDER_PRODUCTS_INDEX"."PRODUCTS_ID")
  13 - inmemory("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='M')
       filter("PRODUCTS"."MATERIAL"='Leather' AND "PRODUCTS"."GENDER"='M')
  17 - inmemory(SYS_OP_BLOOM_FILTER(:BF0001,"CLIENTS_LIST_PARTITIONED"."ID"))
       filter(SYS_OP_BLOOM_FILTER(:BF0001,"CLIENTS_LIST_PARTITIONED"."ID"))
  19 - inmemory("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              SYS_OP_BLOOM_FILTER(:BF0000,"ORDERS"."CLIENTS_ID"))
       filter("ORDERS"."CREATION_DATE">=TO_DATE(' 2020-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              SYS_OP_BLOOM_FILTER(:BF0000,"ORDERS"."CLIENTS_ID"))
```

GET_SUMMARY_OF_SALES

Without optimization

```
---------------------------------------------------------------------------------------------
| Id  | Operation                | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------,---------------------------------------
|   0 | SELECT STATEMENT         |           |   201K|    14M|       | 50222   (2)| 00:00:02 |
|   1 |  SORT ORDER BY           |           |   201K|    14M|    17M| 50222   (2)| 00:00:02 |
|   2 |   HASH GROUP BY          |           |   201K|    14M|    17M| 50222   (2)| 00:00:02 |
|*  3 |    HASH JOIN RIGHT SEMI |           |   201K|    14M|  7584K| 42965   (3)| 00:00:02 |
|*  4 |     TABLE ACCESS FULL    | ADDRESSES |   204K|  5184K|       | 13789   (3)| 00:00:01 |
|*  5 |     HASH JOIN            |           |   201K|     9M|  6096K| 28203   (3)| 00:00:02 |
|   6 |      VIEW                | VW_GBC_5  |   201K|  3733K|       | 13688   (3)| 00:00:01 |
|   7 |       HASH GROUP BY      |           |   201K|  3733K|   105M| 13688   (3)| 00:00:01 |
|*  8 |        TABLE ACCESS FULL| ORDERS    |  3432K|    62M|       |  5596   (2)| 00:00:01 |
|*  9 |      TABLE ACCESS FULL   | ADDRESSES |   204K|  6381K|       | 13789   (3)| 00:00:01 |
---------------------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("ADDRESSES"."ADDRESS_CITY"="ADDRESSES"."ADDRESS_CITY")
   4 - filter("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR "ADDRESSES"."ADDRESS_COUNTRY"='Malta'
              OR "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR "ADDRESSES"."ADDRESS_COUNTRY"='Poland'
              OR "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Saudi Arabia' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
   5 - access("ITEM_1"="ADDRESSES"."ID")
   8 - filter("ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss') AND "ORDERS"."CREATION_DATE">=TO_DATE(' 1995-04-29 00:00:00',
              'syyyy-mm-dd hh24:mi:ss'))
   9 - filter("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR "ADDRESSES"."ADDRESS_COUNTRY"='Malta'
              OR "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR "ADDRESSES"."ADDRESS_COUNTRY"='Poland'
              OR "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Saudi Arabia' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
```

Optimized

```
------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name                            | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |                                 |   129K|  9753K|       | 46398   (1)| 00:00:02 |
|   1 |  SORT ORDER BY                           |                                 |   129K|  9753K|    11M| 46398   (1)| 00:00:02 |
|   2 |   HASH GROUP BY                          |                                 |   129K|  9753K|    11M| 46398   (1)| 00:00:02 |
|*  3 |    HASH JOIN RIGHT SEMI                  |                                 |   129K|  9753K|  7400K| 41721   (1)| 00:00:02 |
|   4 |     INLIST ITERATOR                      |                                 |       |       |       |            |          |
|   5 |      BITMAP CONVERSION TO ROWIDS         |                                 |   199K|  5060K|       | 13472   (1)| 00:00:01 |
|*  6 |       BITMAP INDEX RANGE SCAN            | ADDRESS_CITY_COMP_BITMAP_INDEX  |       |       |       |            |          |
|*  7 |     HASH JOIN                            |                                 |   129K|  6460K|  3928K| 27499   (2)| 00:00:02 |
|   8 |      VIEW                                | VW_GBC_5                        |   129K|  2406K|       | 13416   (3)| 00:00:01 |
|   9 |       HASH GROUP BY                      |                                 |   129K|  2406K|   105M| 13416   (3)| 00:00:01 |
|* 10 |        TABLE ACCESS INMEMORY FULL        | ORDERS                          |  3432K|    62M|       |  5596   (2)| 00:00:01 |
|  11 |      INLIST ITERATOR                     |                                 |       |       |       |            |          |
|  12 |       TABLE ACCESS BY INDEX ROWID BATCHED| ADDRESSES                       |   199K|  6228K|       | 13472   (1)| 00:00:01 |
|  13 |        BITMAP CONVERSION TO ROWIDS       |                                 |       |       |       |            |          |
|* 14 |         BITMAP INDEX RANGE SCAN          | ADDRESS_CITY_COMP_BITMAP_INDEX  |       |       |       |            |          |
------------------------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("ADDRESSES"."ADDRESS_CITY"="ADDRESSES"."ADDRESS_CITY")
   6 - access("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Malta' OR "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR "ADDRESSES"."ADDRESS_COUNTRY"='Poland'
              OR "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR "ADDRESSES"."ADDRESS_COUNTRY"='Saudi Arabia' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
       filter("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Malta' OR "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR "ADDRESSES"."ADDRESS_COUNTRY"='Poland'
              OR "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR "ADDRESSES"."ADDRESS_COUNTRY"='Saudi Arabia' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
   7 - access("ITEM_1"="ADDRESSES"."ID")
  10 - inmemory("ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "ORDERS"."CREATION_DATE">=TO_DATE(' 1995-04-29 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
       filter("ORDERS"."CREATION_DATE"<=TO_DATE(' 2021-10-30 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "ORDERS"."CREATION_DATE">=TO_DATE(' 1995-04-29 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
  14 - access("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Malta' OR "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR "ADDRESSES"."ADDRESS_COUNTRY"='Poland'
              OR "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR "ADDRESSES"."ADDRESS_COUNTRY"='Saudi Arabia' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
       filter("ADDRESSES"."ADDRESS_COUNTRY"='Finland' OR "ADDRESSES"."ADDRESS_COUNTRY"='Haiti' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Israel' OR "ADDRESSES"."ADDRESS_COUNTRY"='Lithuania' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Malta' OR "ADDRESSES"."ADDRESS_COUNTRY"='Myanmar' OR "ADDRESSES"."ADDRESS_COUNTRY"='Poland'
              OR "ADDRESSES"."ADDRESS_COUNTRY"='Sao Tome and Principe' OR "ADDRESSES"."ADDRESS_COUNTRY"='Saudi Arabia' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Thailand')
```

GET_TOP_X_CLIENTS_FROM_COUNTRIES_WITH_HIGHEST_AMOUNT_OF_PRODUCTS

Without optimization

```
--------------------------------------------------------------------------------------------------------
| Id  | Operation                             | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |                 |   838 |   259K| 38032   (2)| 00:00:02 |
|   1 |  NESTED LOOPS                         |                 |   838 |   259K| 38032   (2)| 00:00:02 |
|   2 |   NESTED LOOPS                        |                 |   838 |   259K| 38032   (2)| 00:00:02 |
|*  3 |    VIEW                               |                 |   838 |   243K| 36355   (2)| 00:00:02 |
|*  4 |     WINDOW SORT PUSHED RANK           |                 |   838 | 62850 | 36355   (2)| 00:00:02 |
|   5 |      HASH GROUP BY                    |                 |   838 | 62850 | 36355   (2)| 00:00:02 |
|   6 |       NESTED LOOPS                    |                 |   838 | 62850 | 36354   (2)| 00:00:02 |
|   7 |        NESTED LOOPS                   |                 |   838 | 62850 | 36354   (2)| 00:00:02 |
|   8 |         NESTED LOOPS                  |                 |   838 | 47766 | 34677   (2)| 00:00:02 |
|   9 |          NESTED LOOPS                 |                 |   838 | 25140 | 33000   (2)| 00:00:02 |
|* 10 |           HASH JOIN RIGHT SEMI        |                 |   838 | 15922 | 31323   (2)| 00:00:02 |
|  11 |            VIEW                       | VW_NSO_1        |    76 |   380 | 21418   (1)| 00:00:01 |
|* 12 |             HASH JOIN                 |                 |    76 |  4028 | 21418   (1)| 00:00:01 |
|* 13 |              HASH JOIN                |                 |     7 |   301 | 13949   (2)| 00:00:01 |
|* 14 |               TABLE ACCESS FULL       | ADDRESSES       |     7 |   224 | 13674   (2)| 00:00:01 |
|  15 |               TABLE ACCESS FULL       | MANUFACTURERS   | 97406 |  1046K|   274   (1)| 00:00:01 |
|  16 |              TABLE ACCESS FULL        | PRODUCTS        |   992K|  9687K|  7462   (1)| 00:00:01 |
|  17 |            TABLE ACCESS FULL          | ORDER_PRODUCTS  |    10M|   137M|  9826   (2)| 00:00:01 |
|  18 |           TABLE ACCESS BY INDEX ROWID| ORDERS          |     1 |    11 |     2   (0)| 00:00:01 |
|* 19 |            INDEX UNIQUE SCAN          | ORDERS_PK       |     1 |       |     1   (0)| 00:00:01 |
|  20 |          TABLE ACCESS BY INDEX ROWID  | CLIENTS         |     1 |    27 |     2   (0)| 00:00:01 |
|* 21 |           INDEX UNIQUE SCAN           | CLIENTS_PK      |     1 |       |     1   (0)| 00:00:01 |
|* 22 |         INDEX UNIQUE SCAN             | ADDRESSES_PK    |     1 |       |     1   (0)| 00:00:01 |
|  23 |        TABLE ACCESS BY INDEX ROWID    | ADDRESSES       |     1 |    18 |     2   (0)| 00:00:01 |
|* 24 |    INDEX UNIQUE SCAN                  | CLIENTS_PK      |     1 |       |     1   (0)| 00:00:01 |
|  25 |   TABLE ACCESS BY INDEX ROWID         | CLIENTS         |     1 |    20 |     2   (0)| 00:00:01 |
--------------------------------------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("A"."RNK"<=20)
   4 - filter(DENSE_RANK() OVER ( PARTITION BY "ADDRESSES"."ADDRESS_COUNTRY" ORDER BY
              SUM("ORDER_PRODUCTS"."AMOUNT") DESC )<=20)
  10 - access("ORDER_PRODUCTS"."PRODUCTS_ID"="ID")
  12 - access("MANUFACTURERS"."ID"="PRODUCTS"."MANUFACTURERS_ID")
  13 - access("ADDRESSES"."ID"="MANUFACTURERS"."ADDRESSES_ID")
  14 - filter(("ADDRESSES"."ADDRESS_CITY"='Michaelmouth' OR "ADDRESSES"."ADDRESS_CITY"='New
              Michael' OR "ADDRESSES"."ADDRESS_CITY"='West Anthony') AND
              ("ADDRESSES"."ADDRESS_COUNTRY"='Austria' OR "ADDRESSES"."ADDRESS_COUNTRY"='Korea' OR
              "ADDRESSES"."ADDRESS_COUNTRY"='Mexico'))
  19 - access("ORDER_PRODUCTS"."ORDERS_ID"="ORDERS"."ID")
  21 - access("ORDERS"."CLIENTS_ID"="CLIENTS"."ID")
  22 - access("CLIENTS"."ADDRESSES_ID"="ADDRESSES"."ID")
  24 - access("A"."CLIENT_ID"="CLIENTS"."ID")
```

Optimized

```
---------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                   | Name                          | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
---------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                            |                               |   831 |   257K| 25027   (1)| 00:00:01 |       |       |
|   1 |  NESTED LOOPS                               |                               |   831 |   257K| 25027   (1)| 00:00:01 |       |       |
|   2 |   NESTED LOOPS                              |                               |   831 |   257K| 25027   (1)| 00:00:01 |       |       |
|*  3 |    VIEW                                     |                               |   831 |   241K| 23364   (2)| 00:00:01 |       |       |
|*  4 |     WINDOW SORT PUSHED RANK                 |                               |   831 | 66480 | 23364   (2)| 00:00:01 |       |       |
|   5 |      HASH GROUP BY                          |                               |   831 | 66480 | 23364   (2)| 00:00:01 |       |       |
|   6 |       NESTED LOOPS                          |                               |   831 | 66480 | 23363   (2)| 00:00:01 |       |       |
|   7 |        NESTED LOOPS                         |                               |   831 | 66480 | 23363   (2)| 00:00:01 |       |       |
|   8 |         NESTED LOOPS                        |                               |   831 | 51522 | 21700   (2)| 00:00:01 |       |       |
|   9 |          NESTED LOOPS                       |                               |   831 | 29085 | 20037   (2)| 00:00:01 |       |       |
|* 10 |           HASH JOIN                         |                               |   832 | 19968 | 18373   (2)| 00:00:01 |       |       |
|  11 |            NESTED LOOPS                     |                               |    81 |   810 |  8029   (1)| 00:00:01 |       |       |
|  12 |             VIEW                            | VW_NSO_1                      |    81 |   405 |  7946   (1)| 00:00:01 |       |       |
|  13 |              HASH UNIQUE                    |                               |    81 |  4293 |            |          |       |       |
|* 14 |               HASH JOIN                     |                               |    81 |  4293 |  7946   (1)| 00:00:01 |       |       |
|* 15 |                HASH JOIN                    |                               |     8 |   344 |   477   (2)| 00:00:01 |       |       |
|  16 |                 JOIN FILTER CREATE          | :BF0000                       |     8 |   256 |   202   (1)| 00:00:01 |       |       |
|  17 |                  PARTITION HASH INLIST      |                               |     8 |   256 |   202   (1)| 00:00:01 |KEY(I) |KEY(I) |
|* 18 |                   TABLE ACCESS INMEMORY FULL| ADDRESSES_HASH_PARTITIONED    |     8 |   256 |   202   (1)| 00:00:01 |KEY(I) |KEY(I) |
|  19 |                 JOIN FILTER USE             | :BF0000                       | 97406 |  1046K|   274   (1)| 00:00:01 |       |       |
|* 20 |                  TABLE ACCESS INMEMORY FULL | MANUFACTURERS                 | 97406 |  1046K|   274   (1)| 00:00:01 |       |       |
|  21 |                TABLE ACCESS FULL            | PRODUCTS                      |   992K|  9687K|  7462   (1)| 00:00:01 |       |       |
|* 22 |             INDEX UNIQUE SCAN               | PRODUCTS_PK                   |     1 |     5 |     1   (0)| 00:00:01 |       |       |
|  23 |            INDEX FAST FULL SCAN             | ORDER_PRODUCTS_INDEX_PK       |    10M|   137M| 10266   (2)| 00:00:01 |       |       |
|  24 |           TABLE ACCESS BY INDEX ROWID       | ORDERS                        |     1 |    11 |     2   (0)| 00:00:01 |       |       |
|* 25 |            INDEX UNIQUE SCAN                | ORDERS_PK                     |     1 |       |     1   (0)| 00:00:01 |       |       |
|  26 |          TABLE ACCESS BY INDEX ROWID        | CLIENTS                       |     1 |    27 |     2   (0)| 00:00:01 |       |       |
|* 27 |           INDEX UNIQUE SCAN                 | CLIENTS_PK                    |     1 |       |     1   (0)| 00:00:01 |       |       |
|* 28 |         INDEX UNIQUE SCAN                   | ADDRESSES_HASH_PARTITIONED_PK |     1 |       |     1   (0)| 00:00:01 |       |       |
|  29 |        TABLE ACCESS BY GLOBAL INDEX ROWID   | ADDRESSES_HASH_PARTITIONED    |     1 |    18 |     2   (0)| 00:00:01 | ROWID | ROWID |
|* 30 |    INDEX UNIQUE SCAN                        | CLIENTS_PK                    |     1 |       |     1   (0)| 00:00:01 |       |       |
|  31 |   TABLE ACCESS BY INDEX ROWID               | CLIENTS                       |     1 |    20 |     2   (0)| 00:00:01 |       |       |
-------------------------------------------------------------------------------------------------------------------------------------------

     Predicate Information (identified by operation id):
     ---------------------------------------------------

        3 - filter("A"."RNK"<=20)
        4 - filter(DENSE_RANK() OVER ( PARTITION BY "ADDRESSES_HASH_PARTITIONED"."ADDRESS_COUNTRY" ORDER BY
                   SUM("ORDER_PRODUCTS_INDEX"."AMOUNT") DESC )<=20)
       10 - access("PRODUCTS"."ID"="ORDER_PRODUCTS_INDEX"."PRODUCTS_ID")
       14 - access("MANUFACTURERS"."ID"="PRODUCTS"."MANUFACTURERS_ID")
       15 - access("ADDRESSES_HASH_PARTITIONED"."ID"="MANUFACTURERS"."ADDRESSES_ID")
       18 - inmemory(("ADDRESSES_HASH_PARTITIONED"."ADDRESS_CITY"='Michaelmouth' OR "ADDRESSES_HASH_PARTITIONED"."ADDRESS_CITY"='New
                   Michael' OR "ADDRESSES_HASH_PARTITIONED"."ADDRESS_CITY"='West Anthony') AND
                   ("ADDRESSES_HASH_PARTITIONED"."ADDRESS_COUNTRY"='Austria' OR "ADDRESSES_HASH_PARTITIONED"."ADDRESS_COUNTRY"='Korea'
     OR
                   "ADDRESSES_HASH_PARTITIONED"."ADDRESS_COUNTRY"='Mexico'))
            filter(("ADDRESSES_HASH_PARTITIONED"."ADDRESS_CITY"='Michaelmouth' OR "ADDRESSES_HASH_PARTITIONED"."ADDRESS_CITY"='New
                   Michael' OR "ADDRESSES_HASH_PARTITIONED"."ADDRESS_CITY"='West Anthony') AND
                   ("ADDRESSES_HASH_PARTITIONED"."ADDRESS_COUNTRY"='Austria' OR "ADDRESSES_HASH_PARTITIONED"."ADDRESS_COUNTRY"='Korea'
     OR
                   "ADDRESSES_HASH_PARTITIONED"."ADDRESS_COUNTRY"='Mexico'))
       20 - inmemory(SYS_OP_BLOOM_FILTER(:BF0000,"MANUFACTURERS"."ADDRESSES_ID"))
            filter(SYS_OP_BLOOM_FILTER(:BF0000,"MANUFACTURERS"."ADDRESSES_ID"))
       22 - access("PRODUCTS"."ID"="ID")
       25 - access("ORDER_PRODUCTS_INDEX"."ORDERS_ID"="ORDERS"."ID")
       27 - access("ORDERS"."CLIENTS_ID"="CLIENTS"."ID")
       28 - access("CLIENTS"."ADDRESSES_ID"="ADDRESSES_HASH_PARTITIONED"."ID")
       30 - access("A"."CLIENT_ID"="CLIENTS"."ID")
```

Running times and relative running times difference comparison

QUERY	BASE AVG TIME	FINAL AVG TIME	DIFF AVG	INDEXES % DIFF	PARTITIONS % DIFF	COLUMNAR % DIFF	FINAL % DIFF
GET SUMMED ORDERS PRICE FOR EACH CLIENT WITH DATE AND PRICE RANGE	15.82048	15.032988	-0.787492	------------	-3.74%	-14.84%	-4.98%
GET SUMMARY OF SALES - INCOME IN GIVEN TIME PERIOD, COUNTRIES AND CITIES	10.17094	3.324416	-6.846524	-48.26%	---------	-7.17%	-67.31%
GET TOP ‘X’ CLIENTS FROM SPECIFIC COUNTRY WITH HIGHEST AMOUNT OF PRODUCTS ORDERED FROM SPECIFIC MANUFACTURER	11.27488	5.755068	-5.519812	---------	-40.38%	-5.60%	-48.96%

Cost relative difference comparison

QUERY	INDEXES COST % DIFF	PARTITIONS COST % DIFF	COLUMNAR COST % DIFF	FINAL COST % DIFF
GET SUMMED ORDERS PRICE FOR EACH CLIENT WITH DATE AND PRICE RANGE	------------	-1.77%	-0.01%	3.75%
GET SUMMARY OF SALES - INCOME IN GIVEN TIME PERIOD, COUNTRIES AND CITIES	-4.92%	---------	0.00%	-7.61%
GET TOP ‘X’ CLIENTS FROM SPECIFIC COUNTRY WITH HIGHEST AMOUNT OF PRODUCTS ORDERED FROM SPECIFIC MANUFACTURER	---------	-39.37%	0.00%	-34.19%

Rows number relative difference comparison

QUERY	INDEXES ROWS % DIFF	PARTITIONS ROWS % DIFF	COLUMNAR ROWS % DIFF	FINAL ROWS % DIFF
GET SUMMED ORDERS PRICE FOR EACH CLIENT WITH DATE AND PRICE RANGE	------------	0.00%	-0.68%	-34.99%
GET SUMMARY OF SALES - INCOME IN GIVEN TIME PERIOD, COUNTRIES AND CITIES	-43.78%	---------	0.00%	-30.34%
GET TOP ‘X’ CLIENTS FROM SPECIFIC COUNTRY WITH HIGHEST AMOUNT OF PRODUCTS ORDERED FROM SPECIFIC MANUFACTURER	---------	0.00%	0.00%	-0.84%

CONCLUSIONS

The whole phase was about picking the best set of improvements and applying it and checking if applying all improvements improves results further. Analyzing results after testing the best methods it can be stated that improvements were still observed.

Analyzing running times it can be concluded that greatest improvement was observed for query GET SUMMARY OF SALES - INCOME IN GIVEN TIME PERIOD, COUNTRIES AND CITIES. Optimization methods used for this query were Indexes and Columnar storage. Quite big improvement were observed with query GET TOP ‘X’ CLIENTS FROM SPECIFIC COUNTRY WITH HIGHEST AMOUNT OF PRODUCTS ORDERED FROM SPECIFIC MANUFACTURER using Partitions and Columnar storage as optimization methods.

Analyzing running cost in CPU % usage it can be concluded that best improvement - that is decreasing Cost was observed for GET TOP ‘X’ CLIENTS FROM SPECIFIC COUNTRY WITH HIGHEST AMOUNT OF PRODUCTS ORDERED FROM SPECIFIC MANUFACTURER using Partitions and Columnar Storage as optimizing methods. Cost was ~34% less than without optimizing methods.

Improvements in number of rows - smaller the better - was observed for queries GET SUMMED ORDERS PRICE FOR EACH CLIENT WITH DATE AND PRICE RANGE with optimizing methods: partitions and columnar storage and for GET SUMMARY OF SALES - INCOME IN GIVEN TIME PERIOD, COUNTRIES AND CITIES with Indexes and Columnar store as the optimizing methods. The improvements were as follows: 34.99% and 30.34% less than without optimizing methods.

Database workload processing speed for these queries was successfully improved using above optimization methods.