Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа искусственного интеллекта

К У Р С О В О Й П Р О Е К Т

Оптимизация SQL-запросов в СУБД Oracle 11g

по дисциплине «Программирование и оптимизация баз данных»

Выполнил
студент гр 3530203/60101 В.К. Фурман

Руководитель
доцент О.Ю. Сабинин

«___» _____ 202__ г.

Санкт-Петербург

2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Оптимизация запросов — важная часть администрирования баз данных. От скорости выполнения зависит скорость работы всего приложения (если база данных используется для какого-либо приложения). К примеру, в интернет-магазине, чем дольше пользователь ждёт, пока загрузятся данные, тем выше вероятность, что он покинет сайт в пользу конкурента. Оптимизация так же помогает избежать траты на излишнюю покупка железа, которое приобретают, чтобы ускорить работу базы данных. Поэтому крайне важно научиться анализировать запросы на возможность их оптимизации.

Целью данной работы является практика оптимизирования запросов в СУБД Oracle. Конкретно — оптимизироваться будет стоимость запросов (cost), которая вычисляется по формуле (1) [3].

$$\text{cost} = \frac{(\#\text{SRds} \cdot \text{sreadtim}) + (\#\text{MRds} \cdot \text{mreadtim}) + (\#\text{cpuCycles}/\text{cpuSpeed})}{\text{sreadtim}}, \quad (1)$$

где

- SRDs — количество одноблочных чтений.
- sreadtim — время на одно одноблочное чтение.
- #MRDs — количество многоблочных чтений.
- mreadtim — время одно на многоблочное чтение.
- #cpuCycles — количество циклов CPU. Включает в себя стоимость CPU–обработки (чистая стоимость CPU) и стоимость извлечения данных (стоимость buffer cache get в CPU).
- cpuSpeed — количество CPU-циклов в секунду.

Порядок выполнения работы будет следующим:

1. Написать SQL-запрос для каждой задачи.
2. Проанализировать планы выполнения, статистики, а также сами запросы на возможные варианты оптимизаций.
3. Использовать как минимум 2 оптимизации на каждый запрос (причём одна оптимизация может быть использована не более, чем дважды за всю работу).
4. Оптимизация может считаться успешной при уменьшении стоимости (cost).

ОПИСАНИЕ ОКРУЖЕНИЯ

Используется база данных Oracle версии `Oracle Database 21c Express Edition Release 21.0.0.0.0 - Production Version 21.3.0.0.0`. Схемы создавались в Portable Database [2] `xepdb1`, поэтому строки подключения имели вид: `CONNECT hr/PASSWORD@xepdb1`.

Стандартные схемы Oracle (`HR`, `OE` и т. д.) были взяты из оффициального репозитория Oracle [1]. Схема `STUDENT` была взята с учебного курса «Программирование и оптимизация баз данных» [5].

Перед каждым выполнением SQL-запроса выполняется очистка Buffer Cache и Shared Pool с помощью первых двух команд ниже. После чего выполняется запрос с выводом плана.

```
alter system flush shared_pool;
alter system flush buffer_cache;
set autotrace on
@task-/*НОМЕР-ЗАДАНИЯ*/.sql
set autotrace off
```

Для схем `HR`, `OE` и `STUDENT` были сгенерированы данные (код генерации и SQL-скрипты для вставки доступны для скачивания в репозитории [4]). После добавления данных выполняется сбор статистики для задействованных таблиц, пример для сбора статистики таблицы `HR.EMPLOYEES` показан ниже.

```
EXEC DBMS_STATS.GATHER_TABLE_STATS('HR', 'EMPLOYEES');
```

Результаты запросов показываются для исходных данных (без сгенерированных), так как некоторые запросы выводят сотни строк со сгенерированными данными. Все планы, включённые в данную работу, показываются для таблиц со сгенерированными данными.

Проверка наличия индексов в таблицах делалась следующим образом:

```
SELECT
  CAST(ind.table_name AS VARCHAR2(36)) AS "Table",
  CAST(ind.index_name AS VARCHAR2(36)) AS "Index name",
  CAST(col.column_name AS VARCHAR2(36)) AS "Column",
  CAST(ind.index_type AS VARCHAR2(36)) AS "Index type"
FROM user_indexes ind
  JOIN user_ind_columns col
    ON ind.index_name = col.index_name
ORDER BY 1, 2, 3, 4;
```

ГЛАВА 1. ЗАДАЧА «ПРОЦЕНТ МУЖЩИН/ЖЕНЩИН В ЗАДАННУЮ ДАТУ (OE)»

1.1. Описание задачи

Используются таблицы схемы OE. Вывести процентное соотношение мужчин и женщин, разместивших заказы в заданную дату. Если один и тот же человек разместил несколько заказов в заданную дату, он должен быть учтён только один раз. В результате должно быть три столбца: дата, процент мужчин и процент женщин.

1.2. Составленный запрос

```
ACCEPT date CHAR PROMPT 'Enter date in format dd-mm-yyyy (e.g. 29-06-2007): '

SELECT
  '&date' AS "Date",
  100
    * COUNT(CASE WHEN oe.customers.gender = 'M' THEN 1 END)
    / COUNT(*)
    AS "Males (%)",
  100
    * COUNT(CASE WHEN oe.customers.gender = 'F' THEN 1 END)
    / COUNT(*)
    AS "Females (%)"
FROM
  oe.customers
    JOIN (
      SELECT DISTINCT customer_id
      FROM oe.orders
      WHERE
        TRUNC(order_date, 'dd')
          = TRUNC(TO_DATE('&date', 'dd-mm-yyyy'), 'dd')
    ) customer_ids
    ON oe.customers.customer_id = customer_ids.customer_id;
```

Рис.1.1. Запрос для задачи №1

```
Date        Males (%) Females (%)
---------- ---------- -----------
29-06-2007         75          25
```

Рис.1.2. Результат запроса для даты «29-06-2007»

```
-------------------------------------------------------------------------------
| Id  | Operation            | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |           |     1 |    20 |    56   (8)| 00:00:01 |
|   1 |  SORT AGGREGATE      |           |     1 |    20 |            |          |
|*  2 |   HASH JOIN          |           |   379 |  7580 |    56   (8)| 00:00:01 |
|   3 |    VIEW              |           |   379 |  4927 |    29  (14)| 00:00:01 |
|   4 |     HASH UNIQUE      |           |   379 |  4927 |    29  (14)| 00:00:01 |
|*  5 |      TABLE ACCESS FULL| ORDERS    |   385 |  5005 |    28  (11)| 00:00:01 |
|   6 |    TABLE ACCESS FULL | CUSTOMERS | 10319 | 72233 |    27   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

   2 - access("CUSTOMERS"."CUSTOMER_ID"="CUSTOMER_IDS"."CUSTOMER_ID")
   5 - filter(TRUNC(INTERNAL_FUNCTION("ORDER_DATE"),'fmdd')=TO_DATE('
              2007-06-29 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND "CUSTOMER_ID">0)

Note
-----
   - this is an adaptive plan


Statistics
----------------------------------------------------------
       2732  recursive calls
          0  db block gets
       3905  consistent gets
        478  physical reads
          0  redo size
        776  bytes sent via SQL*Net to client
         52  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
        248  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

Рис.1.3. План выполнения запроса

1.3. Первая оптимизация (function-based индекс)

Обратим внимание на то, что в исходном запросе используется сравнение даты, использующее функцию `TRUNC`. Создадим function-based индекс на эту функцию на столбец `order_date`.

```
CREATE INDEX orders_order_date_fnidx ON oe.orders (TRUNC(order_date, 'dd'));
```

Рис.1.4. Создание function-based индекса

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                          | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                     |     1 |    20 |    48   (5)| 00:00:01 |
|   1 |  SORT AGGREGATE                    |                     |     1 |    20 |            |          |
|*  2 |   HASH JOIN                        |                     |   250 |  5000 |    48   (5)| 00:00:01 |
|   3 |    VIEW                            |                     |   250 |  3250 |    20   (5)| 00:00:01 |
|   4 |     HASH UNIQUE                    |                     |   250 |  3500 |    20   (5)| 00:00:01 |
```

```
|*  5 |      TABLE ACCESS BY INDEX ROWID BATCHED| ORDERS                 |   253 |  3542 |    19   (0)| 00:00:01 |
|*  6 |       INDEX RANGE SCAN                  | ORDERS_ORDER_DATE_FNIDX |   101 |       |     1   (0)| 00:00:01 |
|   7 |    TABLE ACCESS FULL                    | CUSTOMERS              | 10319 | 72233 |    27   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------

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

   2 - access("CUSTOMERS"."CUSTOMER_ID"="CUSTOMER_IDS"."CUSTOMER_ID")
   5 - filter("CUSTOMER_ID">0)
   6 - access(TRUNC(INTERNAL_FUNCTION("ORDER_DATE"),'fmdd')=TO_DATE(' 2007-06-29 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss'))


Statistics
----------------------------------------------------------
       2992  recursive calls
          5  db block gets
       4135  consistent gets
        491  physical reads
        852  redo size
        776  bytes sent via SQL*Net to client
         52  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
        271  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

Рис.1.5. План выполнения оптимизированного запроса

Как можно увидеть по плану, cost уменьшился на $\sim 14\%$ ($56 \to 48$), поменялись также статистики (не в лучшую сторону):

- `recursive calls` — $2732 \to 2992$,
- `db block gets` — $0 \to 5$,
- `consistent gets` — $3905 \to 4135$,
- `physical reads` — $478 \to 491$,
- `redo size` — $0 \to 852$,
- `sorts (memory)` — $248 \to 271$.

1.4. Вторая оптимизация (bitmap индекс)

Обратим также внимание на то, что столбец `gender` таблицы `customers` имеет всего 2 значения — «F» и «M», из чего напрашивается желание создать bitmap-индекс на этот столбец.

```
CREATE BITMAP INDEX customers_gender_btmidx ON oe.customers (gender);
```

Рис.1.6. Создание bitmap индекса

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name                  | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                       |     1 |    20 |    51   (8)| 00:00:01 |
|   1 |  SORT AGGREGATE               |                       |     1 |    20 |            |          |
|*  2 |   HASH JOIN                   |                       |   379 |  7580 |    51   (8)| 00:00:01 |
|   3 |    VIEW                       |                       |   379 |  4927 |    29  (14)| 00:00:01 |
```

```
|   4 |     HASH UNIQUE                |                          |   379 |  4927 |    29  (14)| 00:00:01 |
|*  5 |      TABLE ACCESS FULL         | ORDERS                   |   385 |  5005 |    28  (11)| 00:00:01 |
|   6 |    VIEW                        | index$_join$_001         | 10319 | 72233 |    22   (0)| 00:00:01 |
|*  7 |     HASH JOIN                  |                          |       |       |            |          |
|   8 |      BITMAP CONVERSION TO ROWIDS|                         | 10319 | 72233 |     1   (0)| 00:00:01 |
|   9 |       BITMAP INDEX FULL SCAN   | CUSTOMERS_GENDER_BTMIDX  |       |       |            |          |
|  10 |      INDEX FAST FULL SCAN      | CUSTOMERS_PK             | 10319 | 72233 |    26   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------

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

   2 - access("CUSTOMERS"."CUSTOMER_ID"="CUSTOMER_IDS"."CUSTOMER_ID")
   5 - filter(TRUNC(INTERNAL_FUNCTION("ORDER_DATE"),'fmdd')=TO_DATE(' 2007-06-29 00:00:00',
              'syyyy-mm-dd hh24:mi:ss') AND "CUSTOMER_ID">0)
   7 - access(ROWID=ROWID)


Statistics
----------------------------------------------------------
       2988  recursive calls
          0  db block gets
       4087  consistent gets
        432  physical reads
          0  redo size
        776  bytes sent via SQL*Net to client
         52  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
        274  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

Рис.1.7. План выполнения оптимизированного запроса

В плане видим, что стоимость (cost) уменьшилась на $\sim$ 8% (56 $\rightarrow$ 51). Посмотрим, как поменялись статистики (в основном, в худшую сторону):

- `recursive calls` — 2732 $\rightarrow$ 2988,
- `consistent gets` — 3905 $\rightarrow$ 4087,
- `physical reads` — 478 $\rightarrow$ 432,
- `sorts (memory)` — 248 $\rightarrow$ 274.

ГЛАВА 2. ЗАДАЧА «ИМЕНА СОТРУДНИКОВ, ВСТРЕЧАЮЩИЕСЯ БОЛЕЕ 2-УХ РАЗ (HR)»

2.1. Описание задачи

Используются таблицы схемы HR. Вывести имена сотрудников, встречающиеся в таблице сотрудников не менее трех раз и не являющиеся именами руководителей подразделений компании или именами непосредственных руководителей кого-либо. В результате должны быть выведены только имена сотрудников, причём каждое — только один раз.

2.2. Составленный запрос

```
SELECT first_name
FROM hr.employees
GROUP BY first_name
HAVING COUNT(*) >= 3
MINUS
SELECT hr.employees.first_name
FROM hr.employees
  JOIN (
    SELECT manager_id
    FROM hr.employees
    WHERE manager_id IS NOT NULL
    UNION ALL
    SELECT manager_id
    FROM hr.departments
    WHERE manager_id IS NOT NULL
  ) manager_ids
  ON hr.employees.employee_id = manager_ids.manager_id;
```

Рис.2.1. Запрос для задачи №2

```
FIRST_NAME
----------
David
Peter
```

Рис.2.2. Результат запроса

```
-------------------------------------------------------------------------------------------
| Id  | Operation               | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
```

```
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                 |     5 |   610 |   107   (4)| 00:00:01 |
|   1 |  MINUS HASH              |                 |       |       |            |          |
|*  2 |   HASH GROUP BY          |                 |     5 |    35 |    17  (18)| 00:00:01 |
|   3 |    INDEX FAST FULL SCAN  | EMP_NAME_IX     | 10107 | 70749 |    14   (0)| 00:00:01 |
|*  4 |   HASH JOIN SEMI         |                 |    23 |   575 |    90   (2)| 00:00:01 |
|   5 |    TABLE ACCESS FULL     | EMPLOYEES       | 10107 |   118K|    68   (0)| 00:00:01 |
|   6 |    VIEW                  |                 | 10117 |   128K|    21   (0)| 00:00:01 |
|   7 |     UNION-ALL            |                 |       |       |            |          |
|*  8 |      INDEX FAST FULL SCAN| EMP_MANAGER_IX  | 10106 | 40424 |    18   (0)| 00:00:01 |
|*  9 |      TABLE ACCESS FULL   | DEPARTMENTS     |    11 |    33 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

   2 - filter(COUNT(*)>=3)
   4 - access("EMPLOYEES"."EMPLOYEE_ID"="MANAGER_IDS"."MANAGER_ID")
   8 - filter("MANAGER_ID" IS NOT NULL)
   9 - filter("MANAGER_ID" IS NOT NULL)


Statistics
----------------------------------------------------------
        526  recursive calls
          0  db block gets
       1189  consistent gets
        420  physical reads
          0  redo size
       2023  bytes sent via SQL*Net to client
         96  bytes received via SQL*Net from client
          6  SQL*Net roundtrips to/from client
         68  sorts (memory)
          0  sorts (disk)
         68  rows processed
```

Рис.2.3. План выполнения запроса

2.3. Первая оптимизация (переписывание запроса)

Попробуем использовать вместо `MINUS` оператор `NOT EXISTS` с подзапросом (рис.2.4).

```
SELECT upper_employees.first_name
FROM hr.employees upper_employees
WHERE NOT EXISTS (
  SELECT hr.employees.first_name
  FROM hr.employees
    JOIN (
      SELECT manager_id
      FROM hr.employees
      WHERE manager_id IS NOT NULL
      UNION ALL
      SELECT manager_id
      FROM hr.departments
      WHERE manager_id IS NOT NULL
    ) manager_ids
    ON hr.employees.employee_id = manager_ids.manager_id
  WHERE hr.employees.first_name = upper_employees.first_name
)
```

```
GROUP BY upper_employees.first_name
HAVING COUNT(*) >= 3;
```

Рис.2.4. Переписанный запрос

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                  | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                |     2 |    28 |   105   (2)| 00:00:01 |
|*  1 |  HASH GROUP BY             |                |     2 |    28 |   105   (2)| 00:00:01 |
|*  2 |   HASH JOIN RIGHT ANTI     |                |  7552 |   103K|   104   (1)| 00:00:01 |
|   3 |    VIEW                    | VW_SQ_1        |    23 |   161 |    90   (2)| 00:00:01 |
|*  4 |     HASH JOIN SEMI         |                |    23 |   575 |    90   (2)| 00:00:01 |
|   5 |      TABLE ACCESS FULL     | EMPLOYEES      | 10107 |   118K|    68   (0)| 00:00:01 |
|   6 |      VIEW                  |                | 10117 |   128K|    21   (0)| 00:00:01 |
|   7 |       UNION-ALL            |                |       |       |            |          |
|*  8 |        INDEX FAST FULL SCAN| EMP_MANAGER_IX | 10106 | 40424 |    18   (0)| 00:00:01 |
|*  9 |        TABLE ACCESS FULL   | DEPARTMENTS    |    11 |    33 |     3   (0)| 00:00:01 |
|  10 |    INDEX FAST FULL SCAN    | EMP_NAME_IX    | 10107 | 70749 |    14   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

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

   1 - filter(COUNT(*)>=3)
   2 - access("ITEM_1"="UPPER_EMPLOYEES"."FIRST_NAME")
   4 - access("EMPLOYEES"."EMPLOYEE_ID"="MANAGER_IDS"."MANAGER_ID")
   8 - filter("MANAGER_ID" IS NOT NULL)
   9 - filter("MANAGER_ID" IS NOT NULL)


Statistics
----------------------------------------------------------
        526  recursive calls
          0  db block gets
       1189  consistent gets
        420  physical reads
          0  redo size
       2023  bytes sent via SQL*Net to client
         96  bytes received via SQL*Net from client
          6  SQL*Net roundtrips to/from client
         68  sorts (memory)
          0  sorts (disk)
         68  rows processed
```

Рис.2.5. План выполнения переписанного запроса

Смотрим, как изменился план. Стоимость (cost) уменьшилась довольно незначительно — на $\sim 2\%$ ($107 \rightarrow 105$). Статистики же не изменились никоим образом.

2.4. Вторая оптимизация (материализованное представление)

Попробуем более масштабную оптимизацию — создадим материализованное представление. Оно может очень сильно ускорить запрос. Недостатком, тем не менее, является то, что данные в представлении нужно поддерживать (при изменении данных в исходной таблице), однако таблица `employees` маловероятно будет меняться часто (так как сотрудники в компании не приходят и уходят каждую

минуту), может быть и такое, что таблица может не меняться днями. Поэтому использование материализованного представления здесь вполне оправдано.

Создание материализованного представления показано на рис.2.6.

```
CREATE MATERIALIZED VIEW employees_names_mview
  BUILD IMMEDIATE
  REFRESH COMPLETE
  AS /* ИСХОДНЫЙ ЗАПРОС */
```

Рис.2.6. Создание материализованного представления `employees_names_mview`

```
-------------------------------------------------------------------------------------------
| Id  | Operation            | Name                 | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |                      |    68 |   476 |     3   (0)| 00:00:01 |
|   1 |  MAT_VIEW ACCESS FULL| EMPLOYEES_NAMES_MVIEW |    68 |   476 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------


Statistics
----------------------------------------------------------
        934  recursive calls
          0  db block gets
       1460  consistent gets
         96  physical reads
          0  redo size
       2239  bytes sent via SQL*Net to client
         96  bytes received via SQL*Net from client
          6  SQL*Net roundtrips to/from client
         81  sorts (memory)
          0  sorts (disk)
         68  rows processed
```

Рис.2.7. План выполнения материализованного представления

Глядя на план, можно увидеть, как сильно изменился cost — на $\sim$ 97% (107 $\rightarrow$ 3). Некоторые статистики, тем не менее, ухудшились (кроме `physical reads` — они улучшились):

- recursive calls — 526 $\rightarrow$ 934,
- consistent gets — 1189 $\rightarrow$ 1460,
- physical reads — 420 $\rightarrow$ 96,
- bytes sent via SQL*Net to client — 2023 $\rightarrow$ 2239,
- sorts (memory) — 68 $\rightarrow$ 81.

ГЛАВА 3. ЗАДАЧА «ВЫВОД ГРУПП С ОТЛИЧНИКАМИ И ДОЛЖНИКАМИ (STUDENT)»

3.1. Описание задачи

Используются таблицы схемы STUDENT. Создать запрос для получения информации о группах в виде, представленном в таблице 3.1.

Таблица 3.1

Требуемый вывод для задачи №3

Группа	Кол-во студентов	Название специальности	Кол-во круглых отличников	Кол-во должников
121				
122				
...				

При подсчёте отличников учесть, что пятёрка могла быть получена со второй попытки. Для всех учитывать только экзамены, предусмотренные учебным планом.

3.2. Составленный запрос

```
WITH
  groups_with_students AS (
    SELECT
      student.study_group.group_num,
      student.speciality.spec_title,
      COUNT(student.student.stud_num) AS students_count
    FROM
      student.study_group
        JOIN student.speciality
          ON student.study_group.spec_num = student.speciality.spec_num
        LEFT JOIN student.student
          ON student.study_group.group_num = student.student.group_num
    GROUP BY
      student.study_group.group_num,
      student.speciality.spec_title
  ),
  student_courses AS (
    SELECT
      student.study_group.group_num,
```

```
      student.student.stud_num,
      student.curriculum.course_num
    FROM student.student
      JOIN student.study_group
        ON student.study_group.group_num = student.student.group_num
      JOIN student.curriculum
        ON student.study_group.spec_num = student.curriculum.spec_num
  ),
  correct_grades AS (
    SELECT
      student_courses.group_num,
      student.grades.stud_num,
      student.grades.course_num,
      FIRST_VALUE(student.grades.grade)
        OVER (
          PARTITION BY student.grades.stud_num, student.grades.course_num
          ORDER BY student.grades.exam_date DESC
        )
        AS last_grade
    FROM student.grades
      JOIN student_courses
        ON student.grades.stud_num = student_courses.stud_num
          AND student.grades.course_num = student_courses.course_num
  ),
  min_grades AS (
    SELECT
      group_num,
      stud_num,
      MIN(last_grade) AS min_grade
    FROM correct_grades
    GROUP BY group_num, stud_num
  )
SELECT
  groups_with_students.group_num AS "Группа",
  groups_with_students.students_count AS "Кол-во студентов",
  groups_with_students.spec_title AS "Кол-во студентов",
  COUNT(CASE WHEN min_grades.min_grade = 5 THEN 1 END) AS "Кол-во круглых отличников",
  COUNT(CASE WHEN min_grades.min_grade = 2 THEN 1 END) AS "Кол-во должников"
FROM groups_with_students
  LEFT JOIN min_grades
    ON min_grades.group_num = groups_with_students.group_num
GROUP BY
  groups_with_students.group_num,
  groups_with_students.students_count,
  groups_with_students.spec_title;
```

Рис.3.1. Запрос для задачи №3

Группа	Кол-во студентов	Кол-во студентов	Кол-во круглых отличников	Кол-во должников
121	3	СИСТЕМЫ АВТОМАТИЧЕСКОГО УПРАВЛЕНИЯ	1	1
122	2	СИСТЕМЫ АВТОМАТИЧЕСКОГО УПРАВЛЕНИЯ	1	0
123	2	ЭКОНОМИКА ПРЕДПРИЯТИЙ	0	1
124	2	ЭКОНОМИКА ПРЕДПРИЯТИЙ	0	0

Рис.3.2. Результат запроса

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name         | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |              |  1898 |   170K|       |   373   (4)| 00:00:01 |
|   1 |  HASH GROUP BY                 |              |  1898 |   170K|       |   373   (4)| 00:00:01 |
|*  2 |   HASH JOIN OUTER              |              | 47128 |  4234K|       |   370   (3)| 00:00:01 |
|   3 |    VIEW                        |              |  1898 |   137K|       |    77   (6)| 00:00:01 |
|   4 |     HASH GROUP BY              |              |  1898 |   140K|       |    77   (6)| 00:00:01 |
|*  5 |      HASH JOIN OUTER           |              | 19983 |  1483K|       |    75   (3)| 00:00:01 |
|   6 |       MERGE JOIN               |              |   671 | 44286 |       |     6  (17)| 00:00:01 |
|   7 |        TABLE ACCESS BY INDEX ROWID| SPECIALITY   |     4 |   236 |       |     2   (0)| 00:00:01 |
|   8 |         INDEX FULL SCAN        | SYS_C0010097 |     4 |       |       |     1   (0)| 00:00:01 |
|*  9 |        SORT JOIN               |              |   671 |  4697 |       |     4  (25)| 00:00:01 |
|  10 |         TABLE ACCESS FULL      | STUDY_GROUP  |   671 |  4697 |       |     3   (0)| 00:00:01 |
|  11 |       TABLE ACCESS FULL        | STUDENT      | 19983 |   195K|       |    68   (0)| 00:00:01 |
|  12 |    VIEW                        |              | 16661 |   292K|       |   293   (2)| 00:00:01 |
|  13 |     HASH GROUP BY              |              | 16661 |   113K|       |   293   (2)| 00:00:01 |
|  14 |      VIEW                      |              | 16661 |   113K|       |   291   (2)| 00:00:01 |
|  15 |       WINDOW SORT              |              | 16661 |   715K|   864K|   291   (2)| 00:00:01 |
|* 16 |        HASH JOIN               |              | 16661 |   715K|       |   101   (2)| 00:00:01 |
|  17 |         INDEX FULL SCAN        | PK_UCHEB     |     5 |    35 |       |     1   (0)| 00:00:01 |
|* 18 |         HASH JOIN              |              | 22411 |   809K|       |    99   (2)| 00:00:01 |
|* 19 |          HASH JOIN             |              | 19983 |   331K|       |    72   (2)| 00:00:01 |
|  20 |           TABLE ACCESS FULL    | STUDY_GROUP  |   671 |  4697 |       |     3   (0)| 00:00:01 |
|  21 |           TABLE ACCESS FULL    | STUDENT      | 19983 |   195K|       |    68   (0)| 00:00:01 |
|  22 |          TABLE ACCESS FULL     | GRADES       | 22430 |   438K|       |    27   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------

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

   2 - access("MIN_GRADES"."GROUP_NUM"(+)="GROUPS_WITH_STUDENTS"."GROUP_NUM")
   5 - access("STUDY_GROUP"."GROUP_NUM"="STUDENT"."GROUP_NUM"(+))
   9 - access("STUDY_GROUP"."SPEC_NUM"="SPECIALITY"."SPEC_NUM")
       filter("STUDY_GROUP"."SPEC_NUM"="SPECIALITY"."SPEC_NUM")
  16 - access("GRADES"."COURSE_NUM"="CURRICULUM"."COURSE_NUM" AND
              "STUDY_GROUP"."SPEC_NUM"="CURRICULUM"."SPEC_NUM")
  18 - access("GRADES"."STUD_NUM"="STUDENT"."STUD_NUM")
  19 - access("STUDY_GROUP"."GROUP_NUM"="STUDENT"."GROUP_NUM")


Statistics
----------------------------------------------------------
       2471  recursive calls
         23  db block gets
       4264  consistent gets
        610  physical reads
       4052  redo size
      38650  bytes sent via SQL*Net to client
        536  bytes received via SQL*Net from client
         46  SQL*Net roundtrips to/from client
        246  sorts (memory)
          0  sorts (disk)
        671  rows processed
```

Рис.3.3. План выполнения запроса

3.3. Первая оптимизация (bitmap индекс)

Заметим, что в таблице `student` столбец `group_num` имеет тенденцию хранить похожие значения (так как для каждой группы N-ое количество студентов учится в ней). Отсюда делаем вывод, что логично было бы создать bitmap индекс на этот столбец, что мы и сделаем.

```
-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name                  | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |                       |  1898 |   170K|       |   338   (4)| 00:00:01 |
|   1 |  HASH GROUP BY                         |                       |  1898 |   170K|       |   338   (4)| 00:00:01 |
|*  2 |   HASH JOIN OUTER                      |                       | 47128 |  4234K|       |   335   (3)| 00:00:01 |
|   3 |    VIEW                                |                       |  1898 |   137K|       |    59   (7)| 00:00:01 |
|   4 |     HASH GROUP BY                      |                       |  1898 |   140K|       |    59   (7)| 00:00:01 |
|*  5 |      HASH JOIN OUTER                   |                       | 19983 |  1483K|       |    57   (4)| 00:00:01 |
|   6 |       MERGE JOIN                       |                       |   671 | 44286 |       |     6  (17)| 00:00:01 |
|   7 |        TABLE ACCESS BY INDEX ROWID     | SPECIALITY            |     4 |   236 |       |     2   (0)| 00:00:01 |
|   8 |         INDEX FULL SCAN                | SYS_C0010097          |     4 |       |       |     1   (0)| 00:00:01 |
|*  9 |        SORT JOIN                       |                       |   671 |  4697 |       |     4  (25)| 00:00:01 |
|  10 |         TABLE ACCESS FULL              | STUDY_GROUP           |   671 |  4697 |       |     3   (0)| 00:00:01 |
|  11 |       VIEW                             | index$_join$_004      | 19983 |   195K|       |    51   (2)| 00:00:01 |
|* 12 |        HASH JOIN                       |                       |       |       |       |            |          |
|  13 |         BITMAP CONVERSION TO ROWIDS    |                       | 19983 |   195K|       |    11   (0)| 00:00:01 |
|  14 |          BITMAP INDEX FULL SCAN        | STUDENT_GROUP_NUM_IDX |       |       |       |            |          |
|  15 |         INDEX FAST FULL SCAN           | SYS_C0010070          | 19983 |   195K|       |    49   (0)| 00:00:01 |
|  16 |    VIEW                                |                       | 16661 |   292K|       |   276   (3)| 00:00:01 |
|  17 |     HASH GROUP BY                      |                       | 16661 |   113K|       |   276   (3)| 00:00:01 |
|  18 |      VIEW                              |                       | 16661 |   113K|       |   274   (2)| 00:00:01 |
|  19 |       WINDOW SORT                      |                       | 16661 |   715K|   864K|   274   (2)| 00:00:01 |
|* 20 |        HASH JOIN                       |                       | 16661 |   715K|       |    83   (3)| 00:00:01 |
|  21 |         INDEX FULL SCAN                | PK_UCHEB              |     5 |    35 |       |     1   (0)| 00:00:01 |
|* 22 |         HASH JOIN                      |                       | 22411 |   809K|       |    82   (3)| 00:00:01 |
|* 23 |          HASH JOIN                     |                       | 19983 |   331K|       |    54   (2)| 00:00:01 |
|  24 |           TABLE ACCESS FULL            | STUDY_GROUP           |   671 |  4697 |       |     3   (0)| 00:00:01 |
|  25 |           VIEW                         | index$_join$_006      | 19983 |   195K|       |    51   (2)| 00:00:01 |
|* 26 |            HASH JOIN                   |                       |       |       |       |            |          |
|  27 |             BITMAP CONVERSION TO ROWIDS|                       | 19983 |   195K|       |    11   (0)| 00:00:01 |
|  28 |              BITMAP INDEX FULL SCAN    | STUDENT_GROUP_NUM_IDX |       |       |       |            |          |
|  29 |             INDEX FAST FULL SCAN       | SYS_C0010070          | 19983 |   195K|       |    49   (0)| 00:00:01 |
|  30 |          TABLE ACCESS FULL             | GRADES                | 22430 |   438K|       |    27   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------------

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

   2 - access("MIN_GRADES"."GROUP_NUM"(+)="GROUPS_WITH_STUDENTS"."GROUP_NUM")
   5 - access("STUDY_GROUP"."GROUP_NUM"="STUDENT"."GROUP_NUM"(+))
   9 - access("STUDY_GROUP"."SPEC_NUM"="SPECIALITY"."SPEC_NUM")
       filter("STUDY_GROUP"."SPEC_NUM"="SPECIALITY"."SPEC_NUM")
  12 - access(ROWID=ROWID)
  20 - access("GRADES"."COURSE_NUM"="CURRICULUM"."COURSE_NUM" AND
              "STUDY_GROUP"."SPEC_NUM"="CURRICULUM"."SPEC_NUM")
  22 - access("GRADES"."STUD_NUM"="STUDENT"."STUD_NUM")
  23 - access("STUDY_GROUP"."GROUP_NUM"="STUDENT"."GROUP_NUM")
  26 - access(ROWID=ROWID)


Statistics
----------------------------------------------------------

       2446  recursive calls
         22  db block gets
       3852  consistent gets
        414  physical reads
       3832  redo size
      38650  bytes sent via SQL*Net to client
        536  bytes received via SQL*Net from client
         46  SQL*Net roundtrips to/from client
        242  sorts (memory)
          0  sorts (disk)
        671  rows processed
```

Рис.3.4. План выполнения оптимизированного запроса

Смотрим план выполнения. Наблюдаем уменьшение стоимости (cost) запроса — на ~ 10% (373 → 338). Посмотрим, как поменялись статистики (в лучшую сторону):

- `recursive calls` — 2471 → 2446,
- `db block gets` — 23 → 22,
- `consistent gets` — 4264 → 3852,
- `physical reads` — 610 → 414,
- `redo size` — 4052 → 3832,
- `sorts (memory)` — 246 → 242.

3.4. Вторая оптимизация (индексно-организованная таблица)

Обратим внимание на таблицу `student` — в ней нам нужны лишь поля `stud_num` и `group_num`. Так как остальные поля не используются, логично создать индексно-организованную таблицу `student_lookup` (рис.3.5).

```
DROP TABLE student_lookup PURGE;
CREATE TABLE student_lookup (
  stud_num,
  group_num,
  CONSTRAINT student_lookup_pk PRIMARY KEY (stud_num)
)
ORGANIZATION INDEX
AS
  SELECT stud_num, group_num
  FROM student.student;
```

Рис.3.5. Создание индексно-организованной таблицы `student_lookup`

После чего в исходном запросе (рис.3.1) нам необходимо изменить все вхождения `student.student` на `student.student_lookup` (для краткости изложения, код приводиться не будет, так как нужно поменять всего лишь 2 строки).

```
-----------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name              | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                   |  1898 |   170K|       |   324   (5)| 00:00:01 |
|   1 |  HASH GROUP BY                 |                   |  1898 |   170K|       |   324   (5)| 00:00:01 |
|*  2 |   HASH JOIN OUTER              |                   | 47128 |  4234K|       |   321   (4)| 00:00:01 |
|   3 |    VIEW                        |                   |  1898 |   137K|       |    24  (13)| 00:00:01 |
|   4 |     HASH GROUP BY              |                   |  1898 |   177K|       |    24  (13)| 00:00:01 |
|*  5 |      HASH JOIN OUTER           |                   | 19983 |  1873K|       |    22   (5)| 00:00:01 |
|   6 |       MERGE JOIN               |                   |   671 | 44286 |       |     6  (17)| 00:00:01 |
|   7 |        TABLE ACCESS BY INDEX ROWID| SPECIALITY        |     4 |   236 |       |     2   (0)| 00:00:01 |
|   8 |         INDEX FULL SCAN        | SYS_C0010097      |     4 |       |       |     1   (0)| 00:00:01 |
|*  9 |        SORT JOIN               |                   |   671 |  4697 |       |     4  (25)| 00:00:01 |
|  10 |         TABLE ACCESS FULL      | STUDY_GROUP       |   671 |  4697 |       |     3   (0)| 00:00:01 |
|  11 |       INDEX FAST FULL SCAN     | STUDENT_LOOKUP_PK | 19983 |   585K|       |    16   (0)| 00:00:01 |
|  12 |    VIEW                        |                   | 16661 |   292K|       |   296   (3)| 00:00:01 |
|  13 |     HASH GROUP BY              |                   | 16661 |   113K|       |   296   (3)| 00:00:01 |
|  14 |      VIEW                      |                   | 16661 |   113K|       |   295   (3)| 00:00:01 |
|  15 |       WINDOW SORT              |                   | 16661 |  1041K|  1192K|   295   (3)| 00:00:01 |
|* 16 |        HASH JOIN               |                   | 16661 |  1041K|       |    35  (12)| 00:00:01 |
```

```
|  17 |         INDEX FULL SCAN           | PK_UCHEB          |     5 |    35 |       |     1   (0)| 00:00:01 |
|* 18 |         HASH JOIN                 |                   | 22411 |  1247K|       |    33  (10)| 00:00:01 |
|  19 |          TABLE ACCESS FULL        | STUDY_GROUP       |   671 |  4697 |       |     3   (0)| 00:00:01 |
|  20 |          NESTED LOOPS             |                   | 22411 |  1094K|       |    30  (10)| 00:00:01 |
|  21 |           TABLE ACCESS FULL       | GRADES            | 22430 |   438K|       |    27   (0)| 00:00:01 |
|* 22 |           INDEX UNIQUE SCAN       | STUDENT_LOOKUP_PK |     1 |    30 |       |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------------------

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

   2 - access("MIN_GRADES"."GROUP_NUM"(+)="GROUPS_WITH_STUDENTS"."GROUP_NUM")
   5 - access("STUDY_GROUP"."GROUP_NUM"="STUDENT_LOOKUP"."GROUP_NUM"(+))
   9 - access("STUDY_GROUP"."SPEC_NUM"="SPECIALITY"."SPEC_NUM")
       filter("STUDY_GROUP"."SPEC_NUM"="SPECIALITY"."SPEC_NUM")
  16 - access("GRADES"."COURSE_NUM"="CURRICULUM"."COURSE_NUM" AND
              "STUDY_GROUP"."SPEC_NUM"="CURRICULUM"."SPEC_NUM")
  18 - access("STUDY_GROUP"."GROUP_NUM"="STUDENT_LOOKUP"."GROUP_NUM")
  22 - access("GRADES"."STUD_NUM"="STUDENT_LOOKUP"."STUD_NUM")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)


Statistics
----------------------------------------------------------
       2441  recursive calls
         23  db block gets
       4389  consistent gets
        404  physical reads
       3832  redo size
      38650  bytes sent via SQL*Net to client
        536  bytes received via SQL*Net from client
         46  SQL*Net roundtrips to/from client
        242  sorts (memory)
          0  sorts (disk)
        671  rows processed
```

Рис.3.6. План выполнения оптимизированного запроса

Посмотрим на план выполнения. Видим уменьшение стоимости (cost) запроса — на $\sim 13\%$ ($373 \to 324$). Посмотрим, как поменялись статистики (в лучшую сторону, за исключением `consistent gets`):

- `recursive calls` — $2471 \to 2441$,
- `consistent gets` — $4264 \to 4389$,
- `physical reads` — $610 \to 404$,
- `redo size` — $4052 \to 3832$,
- `sorts (memory)` — $246 \to 242$.

ГЛАВА 4. ЗАДАЧА «КЛИЕНТ, СДЕЛАВШИЙ ПОКУПКУ НА МАКСИМАЛЬНУЮ СУММУ (SH)»

4.1. Описание задачи

Используются таблицы схемы SH. Вывести фамилию и имя клиента, сделавшего покупки через интернет (channel_desc = "Internet") или партнёров (channel_desc = "Partners") не по акции (promo_category = "NO PROMOTION #") на максимальную сумму в заданном году.

4.2. Составленный запрос

```
SELECT
  sh.customers.cust_last_name AS "Surname",
  sh.customers.cust_first_name AS "Name",
  SUM(sh.sales.amount_sold * sh.sales.quantity_sold) AS "Spent"
FROM sh.sales
  JOIN sh.customers
    ON sh.sales.cust_id = sh.customers.cust_id
  JOIN sh.channels
    ON sh.sales.channel_id = sh.channels.channel_id
  JOIN sh.promotions
    ON sh.sales.promo_id = sh.promotions.promo_id
WHERE
  sh.channels.channel_desc IN ('Internet', 'Partners')
    AND sh.promotions.promo_name = 'NO PROMOTION #'
    AND TRUNC(sh.sales.time_id, 'YEAR')
      = TRUNC(TO_DATE('1998', 'yyyy'), 'YEAR')
GROUP BY
  sh.customers.cust_id,
  sh.customers.cust_first_name,
  sh.customers.cust_last_name
ORDER BY "Spent" DESC
FETCH FIRST 1 ROWS ONLY;
```

Рис.4.1. Запрос для задачи №4

```
Surname                                  Name                      Spent
---------------------------------------- -------------------- ----------
Bakerman                                 Marvel                56243,93
```

Рис.4.2. Результат запроса для года 1998

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name       | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |            |     1 |    73 |  1059  (11)| 00:00:01 |       |       |
|   1 |  SORT ORDER BY             |            |     1 |    73 |  1059  (11)| 00:00:01 |       |       |
|*  2 |   VIEW                     |            |     1 |    73 |  1058  (11)| 00:00:01 |       |       |
|*  3 |    WINDOW SORT PUSHED RANK |            |  1149 |   100K|  1058  (11)| 00:00:01 |       |       |
|   4 |     HASH GROUP BY          |            |  1149 |   100K|  1058  (11)| 00:00:01 |       |       |
|*  5 |      HASH JOIN             |            |  1149 |   100K|  1056  (11)| 00:00:01 |       |       |
|*  6 |       HASH JOIN            |            |  1149 | 80430 |   632  (17)| 00:00:01 |       |       |
|   7 |        MERGE JOIN CARTESIAN|            |     2 |    84 |    20   (0)| 00:00:01 |       |       |
|*  8 |         TABLE ACCESS FULL  | PROMOTIONS |     1 |    29 |    17   (0)| 00:00:01 |       |       |
|   9 |         BUFFER SORT        |            |     2 |    26 |     3   (0)| 00:00:01 |       |       |
|* 10 |          TABLE ACCESS FULL | CHANNELS   |     2 |    26 |     3   (0)| 00:00:01 |       |       |
|  11 |        PARTITION RANGE ALL |            |  9188 |   251K|   612  (17)| 00:00:01 |     1 |    28 |
|* 12 |         TABLE ACCESS FULL  | SALES      |  9188 |   251K|   612  (17)| 00:00:01 |     1 |    28 |
|  13 |       TABLE ACCESS FULL    | CUSTOMERS  | 55500 |  1083K|   423   (1)| 00:00:01 |       |       |
-------------------------------------------------------------------------------------------------------

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

   2 - filter("from$_subquery$_008"."rowlimit_$$_rownumber"<=1)
   3 - filter(ROW_NUMBER() OVER ( ORDER BY SUM("SALES"."AMOUNT_SOLD"*"SALES"."QUANTITY_SOLD")
              DESC )<=1)
   5 - access("SALES"."CUST_ID"="CUSTOMERS"."CUST_ID")
   6 - access("SALES"."PROMO_ID"="PROMOTIONS"."PROMO_ID" AND
              "SALES"."CHANNEL_ID"="CHANNELS"."CHANNEL_ID")
   8 - filter("PROMOTIONS"."PROMO_NAME"='NO PROMOTION #')
  10 - filter("CHANNELS"."CHANNEL_DESC"='Internet' OR "CHANNELS"."CHANNEL_DESC"='Partners')
  12 - filter(TRUNC(INTERNAL_FUNCTION("SALES"."TIME_ID"),'fmyear')=TRUNC(TO_DATE('1998','yyyy'),'
              fmyear'))


Statistics
----------------------------------------------------------
       4230  recursive calls
          4  db block gets
      10016  consistent gets
       3582  physical reads
        792  redo size
        735  bytes sent via SQL*Net to client
         52  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
        374  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

Рис.4.3. План выполнения запроса

4.3. Первая оптимизация (индексно-организованная таблица)

В таблице `customers` довольно большое количество столбцов (больше 10), однако в нашем запросе используются лишь 3, отсюда появляется идея создать индексно-организованную таблицу `customer_lookup`, содержащую эти столбцы. Это мы и сделаем (рис.4.4).

```
CREATE TABLE customer_lookup (
  cust_id,
  cust_last_name,
  cust_first_name,
  CONSTRAINT customers_iot_pk PRIMARY KEY (cust_id)
)
ORGANIZATION INDEX
AS
  SELECT
    cust_id,
    cust_last_name,
```

```
    cust_first_name
  FROM sh.customers;
```

Рис.4.4. Создание индексно-организованной таблицы `customer_lookup`

Как и в предыдущей задаче, нам нужно всего лишь заменить все вхождения `sh.customer` на `sh.customer_lookup`.

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name             | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                  |     1 |    73 |   635  (17)| 00:00:01 |       |       |
|   1 |  SORT ORDER BY             |                  |     1 |    73 |   635  (17)| 00:00:01 |       |       |
|*  2 |   VIEW                     |                  |     1 |    73 |   634  (17)| 00:00:01 |       |       |
|*  3 |    WINDOW SORT PUSHED RANK |                  |  1149 |   131K|   634  (17)| 00:00:01 |       |       |
|   4 |     HASH GROUP BY          |                  |  1149 |   131K|   634  (17)| 00:00:01 |       |       |
|   5 |      NESTED LOOPS          |                  |  1149 |   131K|   632  (17)| 00:00:01 |       |       |
|*  6 |       HASH JOIN            |                  |  1149 | 80430 |   632  (17)| 00:00:01 |       |       |
|   7 |        MERGE JOIN CARTESIAN|                  |     2 |    84 |    20   (0)| 00:00:01 |       |       |
|*  8 |         TABLE ACCESS FULL  | PROMOTIONS       |     1 |    29 |    17   (0)| 00:00:01 |       |       |
|   9 |         BUFFER SORT        |                  |     2 |    26 |     3   (0)| 00:00:01 |       |       |
|* 10 |          TABLE ACCESS FULL | CHANNELS         |     2 |    26 |     3   (0)| 00:00:01 |       |       |
|  11 |        PARTITION RANGE ALL |                  |  9188 |   251K|   612  (17)| 00:00:01 |     1 |    28 |
|* 12 |         TABLE ACCESS FULL  | SALES            |  9188 |   251K|   612  (17)| 00:00:01 |     1 |    28 |
|* 13 |       INDEX UNIQUE SCAN    | CUSTOMERS_IOT_PK |     1 |    47 |     0   (0)| 00:00:01 |       |       |
-----------------------------------------------------------------------------------------------------

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

   2 - filter("from$_subquery$_008"."rowlimit_$$_rownumber"<=1)
   3 - filter(ROW_NUMBER() OVER ( ORDER BY SUM("SALES"."AMOUNT_SOLD"*"SALES"."QUANTITY_SOLD") DESC
              )<=1)
   6 - access("SALES"."PROMO_ID"="PROMOTIONS"."PROMO_ID" AND
              "SALES"."CHANNEL_ID"="CHANNELS"."CHANNEL_ID")
   8 - filter("PROMOTIONS"."PROMO_NAME"='NO PROMOTION #')
  10 - filter("CHANNELS"."CHANNEL_DESC"='Internet' OR "CHANNELS"."CHANNEL_DESC"='Partners')
  12 - filter(TRUNC(INTERNAL_FUNCTION("SALES"."TIME_ID"),'fmyear')=TRUNC(TO_DATE('1998','yyyy'),'fmyear
              '))
  13 - access("SALES"."CUST_ID"="CUSTOMER_LOOKUP"."CUST_ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)


Statistics
----------------------------------------------------------
       2227  recursive calls
          0  db block gets
      52689  consistent gets
       2054  physical reads
          0  redo size
        735  bytes sent via SQL*Net to client
         52  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
        197  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

Рис.4.5. План выполнения оптимизированного запроса

Как можно заметить на плане, стоимость запроса существенно снизилась — на ~ 40% (1059 → 635). Посмотрим на изменение статистик (многие статистики улучшились, причём некоторые — довольно ощутимо, — хотя некоторые, всё же ухудшились, особенно `consistent gets`):

- `recursive calls` — 2471 → 2227,

- `db block gets` — 23 → 0,
- `consistent gets` — 4264 → 52689,
- `physical reads` — 610 → 2054,
- `redo size` — 4052 → 0,
- `bytes sent via SQL*Net to client` — 38650 → 735,
- `bytes received via SQL*Net from client` — 536 → 52,
- `SQL*Net roundtrips to/from client` — 46 → 2,
- `sorts (memory)` — 246 → 197,
- `rows processed` — 671 → 1.

4.4. Вторая оптимизация (компизитный B-tree индекс)

Есть также другой путь оптимизации 3-х столбцов из предыдущей оптимизации — заметим, что все эти столбцы находятся внутри `GROUP BY`, что как бы намекает нам на создание композитного B-tree индекса (B-tree потому, что значения не имеют тенденции иметь много одинаковых значений). Этим мы и займёмся — создадим такой индекс (рис.4.6).

```
CREATE INDEX customers_cmpidx ON sh.customers (cust_id, cust_first_name, cust_last_name);
```

Рис.4.6. Создание композитного индекса

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name             | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                  |     1 |    73 |   700  (16)| 00:00:01 |       |       |
|   1 |  SORT ORDER BY             |                  |     1 |    73 |   700  (16)| 00:00:01 |       |       |
|*  2 |   VIEW                     |                  |     1 |    73 |   699  (16)| 00:00:01 |       |       |
|*  3 |    WINDOW SORT PUSHED RANK |                  |  7059 |   620K|   699  (16)| 00:00:01 |       |       |
|   4 |     HASH GROUP BY          |                  |  7059 |   620K|   699  (16)| 00:00:01 |       |       |
|*  5 |      HASH JOIN             |                  |  1149 |   100K|   697  (16)| 00:00:01 |       |       |
|*  6 |       HASH JOIN            |                  |  1149 | 80430 |   632  (17)| 00:00:01 |       |       |
|   7 |        MERGE JOIN CARTESIAN|                  |     2 |    84 |    20   (0)| 00:00:01 |       |       |
|*  8 |         TABLE ACCESS FULL  | PROMOTIONS       |     1 |    29 |    17   (0)| 00:00:01 |       |       |
|   9 |         BUFFER SORT        |                  |     2 |    26 |     3   (0)| 00:00:01 |       |       |
|* 10 |          TABLE ACCESS FULL | CHANNELS         |     2 |    26 |     3   (0)| 00:00:01 |       |       |
|  11 |        PARTITION RANGE ALL |                  |  9188 |   251K|   612  (17)| 00:00:01 |     1 |    28 |
|* 12 |         TABLE ACCESS FULL  | SALES            |  9188 |   251K|   612  (17)| 00:00:01 |     1 |    28 |
|  13 |       INDEX FAST FULL SCAN | CUSTOMERS_CMPIDX | 55500 |  1083K|    65   (2)| 00:00:01 |       |       |
-------------------------------------------------------------------------------------------------------------

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

   2 - filter("from$_subquery$_008"."rowlimit_$$_rownumber"<=1)
   3 - filter(ROW_NUMBER() OVER ( ORDER BY SUM("SALES"."AMOUNT_SOLD"*"SALES"."QUANTITY_SOLD") DESC
              )<=1)
   5 - access("SALES"."CUST_ID"="CUSTOMERS"."CUST_ID")
   6 - access("SALES"."PROMO_ID"="PROMOTIONS"."PROMO_ID" AND
              "SALES"."CHANNEL_ID"="CHANNELS"."CHANNEL_ID")
   8 - filter("PROMOTIONS"."PROMO_NAME"='NO PROMOTION #')
  10 - filter("CHANNELS"."CHANNEL_DESC"='Internet' OR "CHANNELS"."CHANNEL_DESC"='Partners')
  12 - filter(TRUNC(INTERNAL_FUNCTION("SALES"."TIME_ID"),'fmyear')=TRUNC(TO_DATE('1998','yyyy'),'fmyear
              '))


Statistics
```

```
-------------------------------------------------------
       4217  recursive calls
          4  db block gets
       8701  consistent gets
       2286  physical reads
        792  redo size
        735  bytes sent via SQL*Net to client
         52  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
        370  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

Рис.4.7. План выполнения оптимизированного запроса

Теперь посмотрим, как изменился cost. Улучшение поменьше, чем при первой оптимизации — на ~ 34% (1059 → 700), — однако плюсом этой оптимизации является то, что нам не нужно поддерживать отдельную таблицу `sh.customer_lookup`, хотя и нужно поддерживать индекс, при этом прошлая оптимизация работает лишь для `sh.customer_lookup`, текущая — для изначальной таблицы. Посмотрим на статистики (здесь, по сравнению с прошлой оптимизацией, не произошло ухудшения статистик, однако и улучшения не столь значительны):

- `recursive calls` — 4230 → 4217,
- `consistent gets` — 10016 → 8701,
- `physical reads` — 3582 → 2286,
- `sorts (memory)` — 374 → 370.

ГЛАВА 5. ЗАДАЧА «СПИСОК СОТРУДНИКОВ ПО ДОЛЖНОСТЯМ И ЗАРПЛАТАМ (HR)»

5.1. Описание задачи

Используются таблицы схемы HR. Одной командой SELECT вывести список сотрудников компании, имеющих коллег с таким же идентификатором должности и окладом. Если некоторый идентификатор должности и размер оклада имеет один единственный сотрудник, то сведения о нём в результат попадать не должны.

В результат вывести:

1. идентификатор должности;
2. размер оклада;
3. список фамилий сотрудников, имеющих данный идентификатор должности и данный оклад.

Фамилии в списке должны быть:

a. упорядочены по алфавиту (по возрастанию),
b. разделены символами ', ' («запятая» и «пробел»),
c. перед первой фамилией не должно быть символов-разделителей,
d. после последней фамилии символов-разделителей быть не должно.

Результат упорядочить:

1. по размеру оклада (по убыванию),
2. по идентификатору должности (по возрастанию).

5.2. Составленный запрос

```
SELECT
  job_id AS "Job ID",
  salary AS "Salary",
  -- CAST нужен для того, чтобы список помещался в одну строчку.
  CAST(
    LISTAGG(last_name, ', ')
      WITHIN GROUP (ORDER BY last_name)
    AS VARCHAR2(64)
  ) AS "Surnames"
FROM hr.employees
GROUP BY job_id, salary
HAVING COUNT(*) > 1
ORDER BY "Salary" DESC, "Job ID";
```

Рис.5.1. Запрос для задачи №5

```
Job ID         Salary Surnames
---------- ---------- ------------------------------
AD_VP           17000 De Haan, Kochhar
SA_REP          10000 Bloom, King, Tucker
SA_REP           9500 Bernstein, Greene, Sully
SA_REP           9000 Hall, McEwen
SA_REP           8000 Olsen, Smith
SA_REP           7500 Cambrault, Doran
SA_REP           7000 Grant, Sewall, Tuvault
SA_REP           6200 Banda, Johnson
IT_PROG          4800 Austin, Pataballa
ST_CLERK         3300 Bissot, Mallin
SH_CLERK         3200 McCain, Taylor

Job ID         Salary Surnames
---------- ---------- ------------------------------
ST_CLERK         3200 Nayer, Stiles
SH_CLERK         3100 Fleaur, Walsh
SH_CLERK         3000 Cabrio, Feeney
SH_CLERK         2800 Geoni, Jones
ST_CLERK         2700 Mikkilineni, Seo
SH_CLERK         2600 Grant, OConnell
SH_CLERK         2500 Perkins, Sullivan
ST_CLERK         2500 Marlow, Patel, Vargas
ST_CLERK         2400 Gee, Landry
ST_CLERK         2200 Markle, Philtanker
```

Рис.5.2. Результат запроса

```
-------------------------------------------------------------------------------
| Id  | Operation           | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |           |    39 |   975 |    70   (3)| 00:00:01 |
|*  1 |  FILTER             |           |       |       |            |          |
|   2 |   SORT GROUP BY     |           |    39 |   975 |    70   (3)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| EMPLOYEES | 10107 |   246K|    68   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

   1 - filter(COUNT(*)>1)


Statistics
----------------------------------------------------------
        469  recursive calls
          0  db block gets
        918  consistent gets
        295  physical reads
          0  redo size
     100333  bytes sent via SQL*Net to client
        855  bytes received via SQL*Net from client
         75  SQL*Net roundtrips to/from client
         49  sorts (memory)
          0  sorts (disk)
       1101  rows processed
```

Рис.5.3. План выполнения запроса

5.3. Первая оптимизация (материализованное представление)

Как мы уже обсуждали ранее, материализованное представление — довольно хороший выбор для таблицы `employees`, текущий запрос — не исключение (он возвращает не так много строк, поэтому представление будет небольшим). Создадим материализованное представление с помощью скрипта на рис.5.4.

```
CREATE MATERIALIZED VIEW employees_job_salary_mview
  BUILD IMMEDIATE
  REFRESH COMPLETE
  AS /* ИСХОДНЫЙ ЗАПРОС */
```

Рис.5.4. Создание материализованного представления `employees_job_salary_mview`

```
-------------------------------------------------------------------------------------------------
| Id  | Operation            | Name                       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |                            |  1101 | 82575 |     6   (0)| 00:00:01 |
|   1 |  MAT_VIEW ACCESS FULL| EMPLOYEES_JOB_SALARY_MVIEW |  1101 | 82575 |     6   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------


Statistics
----------------------------------------------------------
        954  recursive calls
         18  db block gets
       1576  consistent gets
        113  physical reads
       2940  redo size
     103883  bytes sent via SQL*Net to client
        855  bytes received via SQL*Net from client
         75  SQL*Net roundtrips to/from client
         84  sorts (memory)
          0  sorts (disk)
       1101  rows processed
```

Рис.5.5. План выполнения материализованного представления

Посмотрим на план выполнения. Видим, что очень заметно снизился cost — на ~ 91% (70 → 6). Однако большинство статистик ухудшилось:

- `recursive calls` — 469 → 954,
- `db block gets` — 0 → 18,
- `consistent gets` — 918 → 1576,
- `physical reads` — 295 → 113,
- `redo size` — 0 → 2940,
- `bytes sent via SQL*Net to client` — 100333 → 103883,
- `sorts (memory)` — 49 → 84.

5.4. Вторая оптимизация (композитный bitmap индекс)

Заметим, что в `GROUP BY` находятся 2 столбца — `salary` и `job_id`, что говорит нам о том, что неплохо было бы иметь на них композитный индекс. Создадим же композитный bitmap индекс (bitmap — потому что значения часто повторяются — иначе вряд ли есть смысл по ним группировать) — рис.5.6. Обратим внимание на `DESC` у `salary` — он используется потому, что в сортировке в исходном запросе мы сортируем по `salary DESC` (и, вообще говоря, довольно часто в запросах `employees` сортировка по `salary` идёт именно в обратном порядке).

```
CREATE BITMAP INDEX employees_salary_job_idx ON hr.employees (salary DESC, job_id);
```

Рис.5.6. Создание композитного bitmap индекса

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name                      | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                           |    39 |   975 |    57   (4)| 00:00:01 |
|*  1 |  FILTER                        |                           |       |       |            |          |
|   2 |   SORT GROUP BY                |                           |    39 |   975 |    57   (4)| 00:00:01 |
|   3 |    VIEW                        | index$_join$_001          | 10107 |   246K|    55   (0)| 00:00:01 |
|*  4 |     HASH JOIN                  |                           |       |       |            |          |
|   5 |      BITMAP CONVERSION TO ROWIDS|                          | 10107 |   246K|    10   (0)| 00:00:01 |
|   6 |       BITMAP INDEX FULL SCAN   | EMPLOYEES_SALARY_JOB_IDX  |       |       |            |          |
|   7 |      INDEX FAST FULL SCAN      | EMP_NAME_IX               | 10107 |   246K|    56   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

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

   1 - filter(COUNT(*)>1)
   4 - access(ROWID=ROWID)


Statistics
----------------------------------------------------------
        506  recursive calls
         12  db block gets
        786  consistent gets
        113  physical reads
       2024  redo size
     100333  bytes sent via SQL*Net to client
        855  bytes received via SQL*Net from client
         75  SQL*Net roundtrips to/from client
         54  sorts (memory)
          0  sorts (disk)
       1101  rows processed
```

Рис.5.7. План выполнения оптимизированного запроса

Посмотрим, как эта оптимизация повлияла на план выполнения. В целом, неплохо уменьшилась стоимость (cost) — на ~ 19% (70 → 57). Теперь посмотрим на изменение статистик (в целом, они ухудшились):

- `recursive calls` — 469 → 506,
- `db block gets` — 0 → 12,
- `consistent gets` — 918 → 786,
- `physical reads` — 295 → 113,
- `redo size` — 0 → 2024,
- `sorts (memory)` — 49 → 54.

ЗАКЛЮЧЕНИЕ

В данной работе было выполнено исследование различных способов оптимизации SQL-запросов на примере СУБД Oracle для 5-ти задач. Были написаны SQL-запросы для каждой задачи, были проанализированы планы выполнения, статистики и предложены по два варианта оптимизаций в каждой задаче. В ходе решения были использованы следующие способы оптимизации:

- function-based индекс,
- bitmap индекс,
- переписывание запроса,
- материализованное представление,
- индексно-организованная таблица,
- композитный B-tree индекс,
- композитный bitmap индекс.

В результате для каждого запроса удалось достигнуть снижения стоимости (cost). На протяжении всей работы была видна тенденция к ухудшению статистик (хотя и не везде) после оптимизаций. Объяснить это можно, вероятно, тем, что перед каждым запросом мы чистили shared pool и buffer cache, что могло негативно повлиять на запросы с индексами (по сравнению с тем, как если бы мы не чистили shared pool и buffer cache).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Oracle DB sample schemas. — URL: https://github.com/oracle/db-sample-schemas (дата обращения: 13.11.2021).

2. Oracle portable database. — URL: https://docs.oracle.com/database/121/CNCPT/cdbovrvw.htm#CNCPT89234 (дата обращения: 13.11.2021).

3. Using EXPLAIN PLAN. — URL: https://docs.oracle.com/cd/B10501_01/server.920/a96533/ex_plan.htm#19598 (дата обращения: 05.12.2021).

4. Репозиторий с данной работой. — URL: https://github.com/Ruminat/coursework-optimizing-sql-queries (дата обращения: 05.12.2021).

5. Схема STUDENT. — URL: https://dl.spbstu.ru/mod/folder/view.php?id=145766 (дата обращения: 13.11.2021).