

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

КУРСОВОЙ ПРОЕКТ

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	3
Описание окружения	4
Глава 1. Задача «Процент мужчин/женщин в заданную дату (OE)»	5
1.1. Описание задачи	5
1.2. Составленный запрос	5
1.3. Первая оптимизация (function-based индекс)	6
1.4. Вторая оптимизация (bitmap индекс)	7
Глава 2. Задача «Имена сотрудников, встречающиеся более 2-ух раз (HR)»	8
2.1. Описание задачи	8
2.2. Составленный запрос	8
2.3. Первая оптимизация (переписывание запроса)	9
2.4. Вторая оптимизация (материализованное представление)	10
Глава 3. Задача «Вывод групп с отличниками и должниками (STUDENT)»	12
3.1. Описание задачи	12
3.2. Составленный запрос	12
Глава 4. Задача «Клиент, сделавший покупку на максимальную сумму (SH)»	15
4.1. Описание задачи	15
4.2. Составленный запрос	15
Глава 5. Задача «Список сотрудников по должностям и зарплатам (HR)» ...	17
5.1. Описание задачи	17
5.2. Составленный запрос	17
Заключение	19
Список использованных источников	20

ВВЕДЕНИЕ

SQL (Structured Query Language, язык структурированных запросов) — это специальный язык, используемый для определения данных, доступа к данным и их обработки. Язык SQL относится к непроцедурным (nonprocedural) языкам — он лишь описывает нужные компоненты (например, таблицы) и желаемые результаты, не указывая, как именно эти результаты должны быть получены. Каждая реализация SQL является надстройкой над процессором базы данных (database engine), который интерпретирует операторы SQL и определяет порядок обращения к структурам БД для корректного и эффективного формирования желаемого результата.

Стандарт SQL определяется ANSI — American National Standards Institute (Американским Национальным Институтом Стандартов) и в настоящее время принят ISO — International Standards Organization (Международной Организацией по Стандартизации).

SQL — непроцедурный язык: серверу базы данных сообщается, что нужно сделать и каким образом. Для обработки запроса сервер базы данных транслирует команды SQL во внутренние процедуры. Благодаря тому, что SQL скрывает детали обработки данных, его легко использовать.

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] хепdb1, поэтому строки подключения имели вид: CONNECT hr/PASSWORD@хепdb1.

Стандартные схемы Oracle (HR, OE и т. д.) были взяты из официального репозитория Oracle [1].

Схема STUDENT была взята с учебного курса «Программирование и оптимизация баз данных» [3].

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

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

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

1.1. Описание задачи

Используются таблицы схемы ОЕ. 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

```
SELECT
    '29-06-2007' 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('29-06-2007', '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		250	5000	56 (8)	00:00:01
3	VIEW		250	3250	29 (14)	00:00:01
4	HASH UNIQUE		250	4000	29 (14)	00:00:01
* 5	TABLE ACCESS FULL	ORDERS	253	4048	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"),'fmd')=TO_DATE('
      2007-06-29 00:00:00', 'yyyy-mm-dd hh24:mi:ss') AND "CUSTOMER_ID">0)

```

Statistics

```

1184 recursive calls
0 db block gets
1653 consistent gets
297 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
114 sorts (memory)
0 sorts (disk)
1 rows processed

```

Рис.1.3. План выполнения запроса

1.3. Первая оптимизация (function-based индекс)

```
CREATE INDEX orders_order_date_fnidx ON 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"),'fmd')=TO_DATE(' 2007-06-29 00:00:00', 'yyyy-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

1.4. Вторая оптимизация (bitmap индекс)

```
CREATE BITMAP INDEX customers_gender_btmdix 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		250	5000	51 (8)	00:00:01
3	VIEW		250	3250	29 (14)	00:00:01
4	HASH UNIQUE		250	4000	29 (14)	00:00:01
* 5	TABLE ACCESS FULL	ORDERS	253	4048	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_BTMDIX				
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"),'fmd')=TO_DATE(' 2007-06-29 00:00:00',
    'syyy-mm-dd hh24:mi:ss') AND "CUSTOMER_ID">0)
7 - access(ROWID=ROWID)

```

Statistics

```

1252 recursive calls
13 db block gets
1637 consistent gets
252 physical reads
2032 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
116 sorts (memory)
0 sorts (disk)
1 rows processed

```

Рис.1.7. План выполнения оптимизированного запроса

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		1	10	1.00	0.01
1	GROUP BY		1	10	1.00	0.01
2	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
3	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
4	TABLE ACCESS	DEPARTMENTS	10	100	1.00	0.01
5	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
6	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
7	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
8	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
9	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
10	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
11	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
12	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
13	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
14	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
15	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
16	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
17	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
18	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
19	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
20	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
21	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
22	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
23	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
24	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
25	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
26	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
27	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
28	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
29	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
30	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
31	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
32	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
33	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
34	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
35	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
36	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
37	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
38	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
39	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
40	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
41	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
42	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
43	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
44	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
45	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
46	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
47	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
48	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
49	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
50	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
51	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
52	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
53	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
54	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
55	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
56	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
57	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
58	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
59	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
60	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
61	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
62	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
63	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
64	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
65	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
66	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
67	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
68	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
69	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
70	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
71	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
72	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
73	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
74	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
75	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
76	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
77	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
78	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
79	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
80	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
81	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
82	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
83	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
84	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
85	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
86	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
87	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
88	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
89	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
90	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
91	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
92	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
93	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
94	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
95	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
96	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
97	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
98	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01
99	TABLE ACCESS	EMPLOYEES	100	1000	1.00	0.01

	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)

```

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

2.4. Вторая оптимизация (материализованное представление)

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.2.6. План выполнения материализованного представления

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.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.group_num
  GROUP BY
    student.study_group.group_num,
    student.speciality.spec_title
),
student_courses AS (
  SELECT
    student.study_group.group_num,

```

```

        student.stud_num,
        student.curriculum.course_num
FROM student.student
    JOIN student.study_group
        ON student.study_group.group_num = 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	Cost (%CPU)	Time
0	SELECT STATEMENT		15	1350	96 (10)	00:00:01
1	HASH GROUP BY		15	1350	96 (10)	00:00:01
* 2	HASH JOIN OUTER		24486	2152K	94 (8)	00:00:01
3	VIEW		15	1095	40 (8)	00:00:01
4	HASH GROUP BY		15	1125	40 (8)	00:00:01
* 5	HASH JOIN OUTER		10009	733K	38 (3)	00:00:01
6	MERGE JOIN		5	330	6 (17)	00:00:01
7	TABLE ACCESS BY INDEX ROWID	SPECIALITY	4	236	2 (0)	00:00:01
8	INDEX FULL SCAN	SYS_C009182	4		1 (0)	00:00:01
* 9	SORT JOIN		5	35	4 (25)	00:00:01
10	TABLE ACCESS FULL	STUDY_GROUP	5	35	3 (0)	00:00:01
11	TABLE ACCESS FULL	STUDENT	10009	90081	32 (0)	00:00:01
12	VIEW		8162	135K	54 (8)	00:00:01
13	HASH GROUP BY		8162	57134	54 (8)	00:00:01
14	VIEW		8162	57134	53 (6)	00:00:01
15	WINDOW SORT		8162	342K	53 (6)	00:00:01
16	NESTED LOOPS		8162	342K	52 (4)	00:00:01
* 17	HASH JOIN		11280	396K	51 (2)	00:00:01
* 18	HASH JOIN		10009	156K	35 (0)	00:00:01
19	TABLE ACCESS FULL	STUDY_GROUP	5	35	3 (0)	00:00:01
20	TABLE ACCESS FULL	STUDENT	10009	90081	32 (0)	00:00:01
21	TABLE ACCESS FULL	GRADES	11280	220K	15 (0)	00:00:01
* 22	INDEX UNIQUE SCAN	PK_UCHEB	1	7	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"."GROUP_NUM" (+))
9 - access("STUDY_GROUP"."SPEC_NUM"= "SPECIALITY"."SPEC_NUM")
   filter("STUDY_GROUP"."SPEC_NUM"= "SPECIALITY"."SPEC_NUM")
17 - access("GRADES"."STUD_NUM"= "STUDENT"."STUD_NUM")
18 - access("STUDY_GROUP"."GROUP_NUM"= "STUDENT"."GROUP_NUM")
22 - access("STUDY_GROUP"."SPEC_NUM"= "CURRICULUM"."SPEC_NUM" AND
   "GRADES"."COURSE_NUM"= "CURRICULUM"."COURSE_NUM")

```

Statistics

```

2462 recursive calls
22 db block gets
3941 consistent gets
431 physical reads
3860 redo size
1541 bytes sent via SQL*Net to client
52 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from client
244 sorts (memory)
0 sorts (disk)
5 rows processed

```

Рис.3.3. План выполнения запроса

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
  FIRST_VALUE("Surname") AS "Surname",
  FIRST_VALUE("Name") AS "Name",
  FIRST_VALUE("Spent") AS "Spent"
FROM (
  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
);
```

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

```

4224 recursive calls
4 db block gets
9984 consistent gets
3571 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
372 sorts (memory)
0 sorts (disk)
1 rows processed

```

Рис.4.3. План выполнения запроса

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/*+ OPT_PARAM('_GBY_HASH_AGGREGATION_ENABLED', 'false') */
  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

ЗАКЛЮЧЕНИЕ

Вода. Земля. Огонь. Воздух. Моя бабушка любила рассказывать мне истории о прошлом, когда Аватар сохранял равновесие между Племенами Воды, Царством Земли, Народом Огня и Воздушными кочевниками. Всё изменилось, когда Народ Огня развязал войну. Только Аватар, властелин всех четырёх стихий, мог остановить безжалостные атаки магов огня. Но, когда мир нуждался в нём больше всего, он исчез. Прошло сто лет, и Народ Огня уже был близок к победе в войне. Два года назад мой отец и мои соплеменники отправились в Царство Земли, чтобы помочь им сражаться против Народа Огня, оставив меня и моего брата управлять племенем. Некоторые люди полагали, что Аватар никогда больше не возродится, и связь времён утеряна навсегда, но я не утратила надежду. Я верила, что Аватар всё же вернётся и спасёт мир.

Земля. Огонь. Воздух. Вода. Когда я был мальчиком, мой отец, Аватар Аанг, рассказывал мне историю о том, как он и его друзья героически завершили Столетнюю войну. Аватар Аанг и Хозяин Огня Зуко преобразовали колонии Народа Огня в Объединенную Республику наций, общество, в котором маги и не-маги со всего мира могли бы жить и процветать вместе, в мире и гармонии. Они назвали столицу этой великой страны — город Республика. Аватар Аанг сделал много замечательных вещей в своей жизни но, к сожалению, его время в этом мире подошло к концу. И, как и цикл времён года, цикл Аватара начался заново.

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