

Лабораторна робота 4

Індекси

Завдання роботи

60%

0. Підготовчий етап. Разом із текстом роботи наведено сценарій init.sql. Він створює таблиці new_dept і new_emp – аналоги emp та dept, але більші за розміром.

Перед початком роботи завантажимо сценарій в базу даних за допомогою sqlplus. Зробити це можна наступною командою в Linux:

```
sqlplus dmytro/dmytro@192.168.1.230:1521/XEPDB1 < Downloads/init.sql
```

Перевіряємо чи створені та заповнені таблиці:

The screenshot displays the SQL Developer interface with the 'NEW_EMP' table selected in the Database Explorer. The table contains 501 rows of employee data. The 'Scripts' panel shows the SQL script used to create the table and insert data.

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
9999	RICHARD ANDERSON	HR	8113	2017-11-18	3080.00	1000.00	30
9998	JOSE MARTIN	TRAINER	2904	2017-02-21	5830.00	1000.00	75
9997	EDWARD THOMPSON	QA	2781	2018-01-22	5280.00	1100.00	83
9996	RICHARD WILSON	QA	961	2016-11-30	3300.00	1600.00	73
9995	CHARLES BROWN	CLERK	207	2016-06-19	770.00	1000.00	83
9994	JOSE TAYLOR	ANALYTIC	9635	2020-02-04	7600.00	1000.00	84
9993	DAVID WILSON	DEVELOPER	3221	2019-04-10	5280.00	1400.00	84
9992	DANIEL MILLER	SENIOR DEVELOPER	226	2018-03-16	1870.00	1400.00	19
9991	JOHN WHITE	CLERK	5795	2019-10-03	550.00	2300.00	35
9990	DANIEL THOMPSON	ANALYTIC	928	2019-03-25	5500.00	800.00	96
9989	JOHN ROBINSON	SECURITY	781	2020-05-03	1430.00	1000.00	45
9988	DAVID WILLIAMS	CLERK	1885	2019-12-23	880.00	1600.00	10
9987	DONALD JONES	SALESMAN	2407	2019-12-24	1320.00	900.00	3
9986	GARY GARCIA	JUNIOR DEVELOPER	8703	2016-05-07	4510.00	1500.00	16
9985	GARY WILLIAMS	SALES ASSISTANT	5284	2020-03-31	3300.00	1200.00	38
9984	DONALD SMITH	ANALYTIC	1875	2020-04-25	5940.00	1600.00	76

```
SQL> CREATE TABLE NEW_EMP (
  EMPNO NUMBER(4) NOT NULL,
  ENAME VARCHAR2(30) NOT NULL,
  JOB VARCHAR2(30) NOT NULL,
  MGR NUMBER(4) NOT NULL,
  HIREDATE DATE NOT NULL,
  SAL NUMBER(8,2) NOT NULL,
  COMM NUMBER(8,2) NOT NULL,
  DEPTNO NUMBER(2) NOT NULL,
  CONSTRAINT PK_NEW_EMP PRIMARY KEY (EMPNO)
);

SQL> INSERT INTO NEW_EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)
SELECT t.EMPNO, t.ENAME, t.JOB, t.MGR, t.HIREDATE, t.SAL, t.COMM, t.DEPTNO
FROM DMYTRO.NEW_EMP t
ORDER BY EMPNO DESC
WHERE ROWNUM <= 501;
```

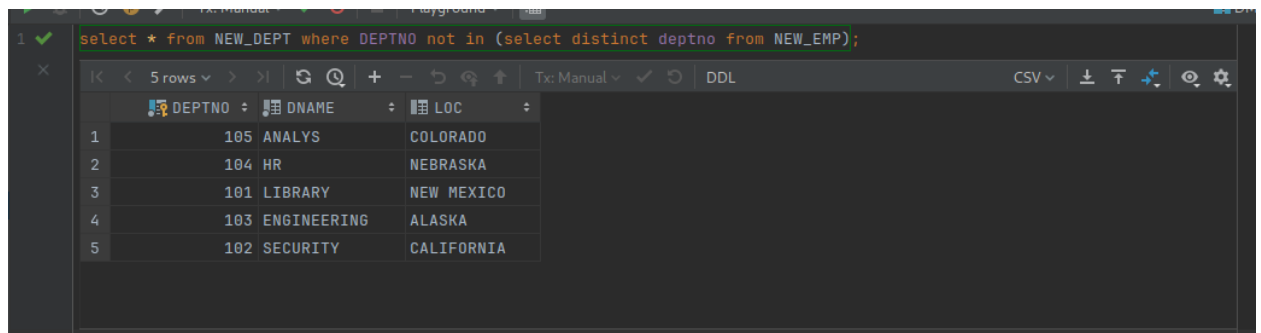
1. Плани запитів

Варіант 1 — Назви відділів, у яких НЕМАЄ співробітників

Виконайте запит різними варіантами (використовуючи Join, підзапит з in, підзапит з exist, операції над множинами, агрегатні функції) Порівняйте плани виконання запитів. Який з варіантів більш оптимальний?

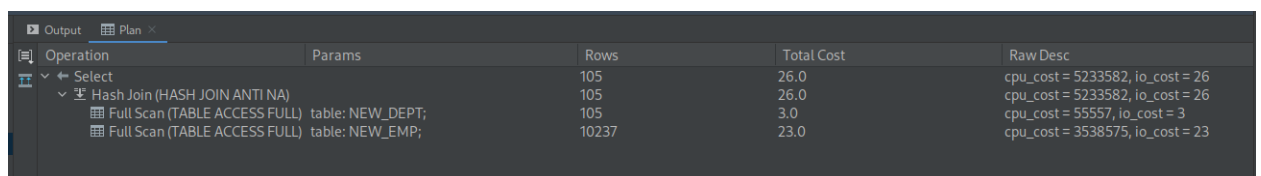
Спосіб 1

```
select * from NEW_DEPT where DEPTNO not in (select distinct deptno from NEW_EMP);
```



	DEPTNO	DNAME	LOC
1	105	ANALYS	COLORADO
2	104	HR	NEBRASKA
3	101	LIBRARY	NEW MEXICO
4	103	ENGINEERING	ALASKA
5	102	SECURITY	CALIFORNIA

План виконання запиту:

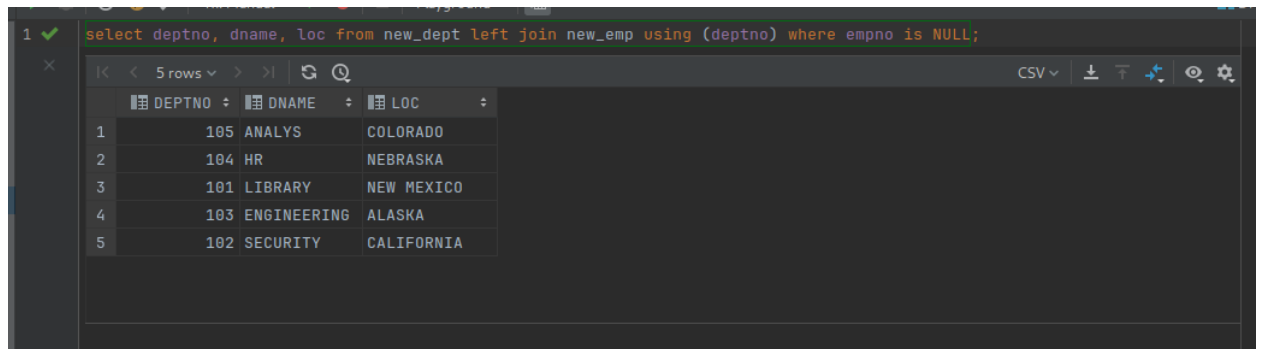


Operation	Params	Rows	Total Cost	Raw Desc
Select		105	26.0	cpu_cost = 5233582, io_cost = 26
Hash Join (HASH JOIN ANTI NA)		105	26.0	cpu_cost = 5233582, io_cost = 26
Full Scan (TABLE ACCESS FULL)	table: NEW_DEPT;	105	3.0	cpu_cost = 55557, io_cost = 3
Full Scan (TABLE ACCESS FULL)	table: NEW_EMP;	10237	23.0	cpu_cost = 3538575, io_cost = 23

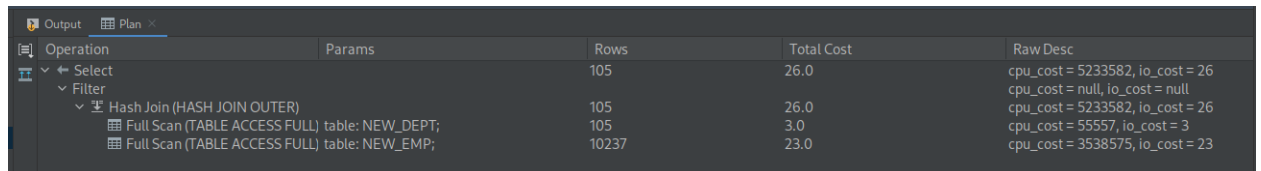
Бачимо, що ціна по io буде 26 папуг.

Спосіб 2

```
select deptno, dname, loc from new_dept left join new_emp using (deptno) where empno is NULL;
```



	DEPTNO	DNAME	LOC
1	105	ANALYS	COLORADO
2	104	HR	NEBRASKA
3	101	LIBRARY	NEW MEXICO
4	103	ENGINEERING	ALASKA
5	102	SECURITY	CALIFORNIA

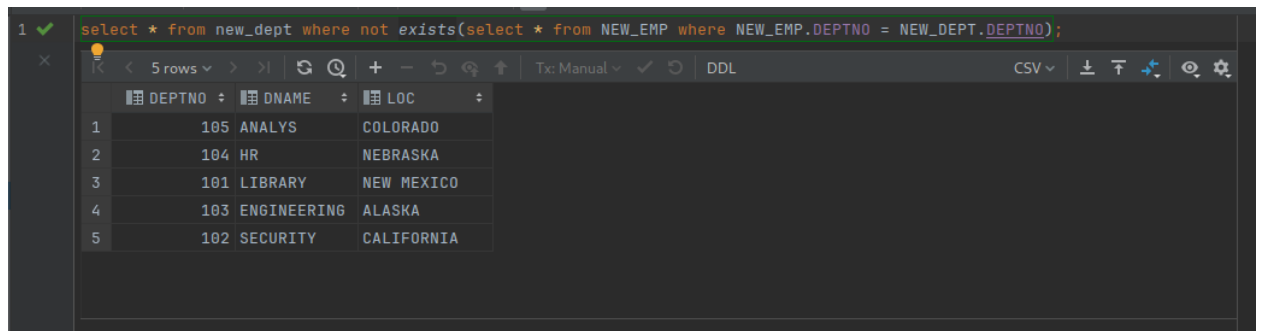
План виконання:

Operation	Params	Rows	Total Cost	Raw Desc
Select		105	26.0	cpu_cost = 5233582, io_cost = 26
Filter				cpu_cost = null, io_cost = null
Hash Join (HASH JOIN OUTER)		105	26.0	cpu_cost = 5233582, io_cost = 26
Full Scan (TABLE ACCESS FULL) table: NEW_DEPT;		105	3.0	cpu_cost = 55557, io_cost = 3
Full Scan (TABLE ACCESS FULL) table: NEW_EMP;		10237	23.0	cpu_cost = 3538575, io_cost = 23

У цього запиту ціна по іо буде також 26 папуг.

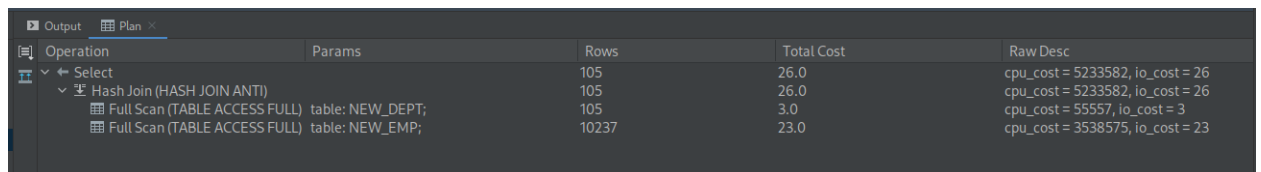
Спосіб 3

```
select * from new_dept where not exists(select * from NEW_EMP where NEW_EMP.DEPTNO = NEW_DEPT.DEPTNO);
```



	DEPTNO	DNAME	LOC
1	105	ANALYS	COLORADO
2	104	HR	NEBRASKA
3	101	LIBRARY	NEW MEXICO
4	103	ENGINEERING	ALASKA
5	102	SECURITY	CALIFORNIA

План виконання:



Operation	Params	Rows	Total Cost	Raw Desc
Select		105	26.0	cpu_cost = 5233582, io_cost = 26
Hash Join (HASH JOIN ANTI)		105	26.0	cpu_cost = 5233582, io_cost = 26
Full Scan (TABLE ACCESS FULL)	table: NEW_DEPT;	105	3.0	cpu_cost = 55557, io_cost = 3
Full Scan (TABLE ACCESS FULL)	table: NEW_EMP;	10237	23.0	cpu_cost = 3538575, io_cost = 23

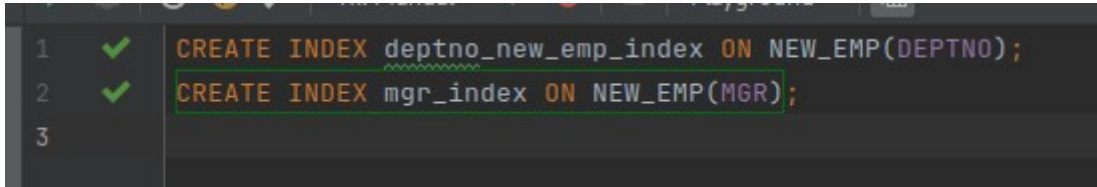
Отже, у всіх запитів однакова ціна по іо, бо в усіх випадках виконується повне сканування таблиць. А у порівняннях часу виконання, останній запит трішки виграє у перших двох, оскільки він використовує вбудовану функцію `exists()`.

64%

2. Використання індексів. Створіть 2 індекси: на колонках deptno та mgr. Порівняйте плани виконання запитів після створення індексів. Зробіть висновки.

Створюємо індекси:

```
CREATE INDEX deptno_new_emp_index ON NEW_EMP(DEPTNO);  
CREATE INDEX mgr_index ON NEW_EMP(MGR);
```



Порівнюємо результати:

Спосіб 1

Для того, щоб база даних використала наш індекс додамо поле deptno в конструкцію where, наприклад отак:

```
select * from NEW_DEPT where DEPTNO not in (select deptno from NEW_EMP where  
NEW_EMP.DEPTNO is not null);
```

Отримаємо такий план:

Operation	Params	Rows	Total Cost	Raw Desc
Select		105	12.0	cpu_cost = 3273043, io_cost = 12
Hash Join (HASH JOIN ANTI)		105	12.0	cpu_cost = 3273043, io_cost = 12
Nested Loops (NESTED LOOP)		105	12.0	cpu_cost = 3273043, io_cost = 12
Unknown (STATISTICS COLLE)				cpu_cost = null, io_cost = null
Full Scan (TABLE ACCESS)	table: NEW_DEPT;	105	3.0	cpu_cost = 55557, io_cost = 3
Index Scan (INDEX RANGE)	index: DEPTNO_NEW_EMP_INDEX;	1	9.0	cpu_cost = 1578036, io_cost = 9
Full Index Scan (INDEX FAST)	index: DEPTNO_NEW_EMP_INDEX;	10237	9.0	cpu_cost = 1578036, io_cost = 9

Тут зафіксовано зменшення вартості в іо більше ніж удвічі. Оскільки нам з таблиці NEW_EMP треба лише індексована колонка, база даних повертає її з індексу.

Запит без індексу:

```
SQL> select * from NEW_DEPT where DEPTNO not in (select distinct deptno from NEW_EMP)  
[2023-03-28 20:34:27] 5 rows retrieved starting from 1 in 65 ms (execution: 20 ms, fetching: 45 ms)  
SQL> select * from NEW_DEPT where DEPTNO not in (select distinct deptno from NEW_EMP)
```

Запит з індексом:

```
[2023-03-28 20:56:12] completed in 17 ms  
SQL> select * from NEW_DEPT where DEPTNO not in (select deptno from NEW_EMP where NEW_EMP.DEPTNO is not null)  
[2023-03-28 21:01:34] 5 rows retrieved starting from 1 in 18 ms (execution: 7 ms, fetching: 11 ms)
```

Бачимо, що запит виконався більш ніж удвічі швидше.

Спосіб 2

Другий спосіб матиме абсолютну таку ж ціну в іо папугах, бо ми в будь-якому випадку виконуємо повне сканування таблиці NEW_EMP для того, щоб співставити записи з таблиці NEW_DEPT під час операції join. Тому індекс не буде використано.

Спосіб 3

Третій спосіб також використовує індекс для перевірки існування запису в системі, якщо ми відредагуємо наш вкладений запит і зробимо не `select *`, а `select deptno` щоб уникнути звернення до таблиці, й обмежитися скануванням індексу.

План виконання:

Operation	Params	Rows	Total Cost	Raw Desc
Select		105	12.0	cpu_cost = 3073043, io_cost = 12
Hash Join (HASH JOIN ANTI)		105	12.0	cpu_cost = 3073043, io_cost = 12
Nested Loops (NESTED LOOPS A)		105	12.0	cpu_cost = 3073043, io_cost = 12
Unknown (STATISTICS COLLECTC				cpu_cost = null, io_cost = null
Full Scan (TABLE ACCESS F table: NEW_DEPT;		105	3.0	cpu_cost = 55557, io_cost = 3
Index Scan (INDEX RANGE SC index: DEPTNO_NEW_EMP_INDEX;		1	9.0	cpu_cost = 1378036, io_cost = 9
Full Index Scan (INDEX FAST FUL index: DEPTNO_NEW_EMP_INDEX;		10237	9.0	cpu_cost = 1378036, io_cost = 9

Таким чином ми так само зменшили ціну в іо папугах більш ніж удвічі, як і в першому способі.

Запит без індексу:

```
[2023-03-28 20:42:17] Completed in 42 ms  
SQL> select * from new_dept where not exists(select * from NEW_EMP where NEW_EMP.DEPTNO = NEW_DEPT.DEPTNO)  
[2023-03-28 20:43:41] 5 rows retrieved starting from 1 in 45 ms (execution: 10 ms, fetching: 35 ms)
```

Запит з індексом:

```
SQL> select * from new_dept where not exists(select deptno from NEW_EMP where NEW_EMP.DEPTNO = NEW_DEPT.DEPTNO)  
[2023-03-28 21:28:31] 5 rows retrieved starting from 1 in 15 ms (execution: 3 ms, fetching: 12 ms)
```

Бачимо пришвидшення запиту також як мінімум у двічі (якщо враховувати \pm середні значення кількох запусків)

74%

3. Функціональні та композитні індекси. За результатами аналізу запит виконується дуже часто його необхідно оптимізувати. Для цього можна використати композитні індекси (кілька колонок у індексі), функціональні індекси (індекс може зберігати не самі данні, а будь які функції на них)

Варіант 1 — Імена співробітників, що були найняті у такий самий день тижня, що і сьогодні

Для цього створимо комбінований функціональний індекс, що буде зберігати день прийняття на роботу для кожного співробітника та його номер. Зробити це можна так:

```
create index hiredate_day_index on NEW_EMP (TO_CHAR(HIREDATE, 'day'), EMPNO);
```

Якщо буде специфічна необхідність дізнатися конкретний параметр нашого EMP, то його необхідно буде додати до індексу, інакше не вдасться уникнути повного сканування таблиці (у випадку якщо це не поштучний select до конкретного empno)

Запит, що будемо робити:

```
select EMPNO from NEW_EMP where TO_CHAR(HIREDATE, 'day') like 'monday%';
```

План виконання запиту для пошуку працівників:

Operation	Params	Rows	Total Cost	Raw Desc
Select		1418	7.0	cpu_cost = 327050, io_cost = 7
Index Scan (INDEX RANGE SC)	index: HIREDATE_DAY_INDEX;	1418	7.0	cpu_cost = 327050, io_cost = 7

Порівняємо виконання вибірки з завдання без індексу та з (запит один і той самий, повертаємо лише empno).

Без індексу:

```
[2023-03-28 22:46:10] 500 rows retrieved starting from 1 in 28 ms (execution: 4 ms, fetching: 21 ms)
DMYTR@> select EMPNO from NEW_EMP where TO_CHAR(HIREDATE, 'day') like 'monday%'
[2023-03-28 22:46:10] 500 rows retrieved starting from 1 in 28 ms (execution: 5 ms, fetching: 23 ms)
```

З індексом:

```
[2023-03-28 22:46:29] 500 rows retrieved starting from 1 in 21 ms (execution: 4 ms, fetching: 17 ms)
DMYTR@> select EMPNO from NEW_EMP where TO_CHAR(HIREDATE, 'day') like 'monday%'
[2023-03-28 22:45:29] 500 rows retrieved starting from 1 in 21 ms (execution: 3 ms, fetching: 18 ms)
```

Різниця здавалося б всього дві мілісекунди, але пропорційно, це майже вдвічі швидше. На великих вибірках даних це пришвидшення удвічі матиме дуже позитивний ефект.

80%**4. Розмір даних**

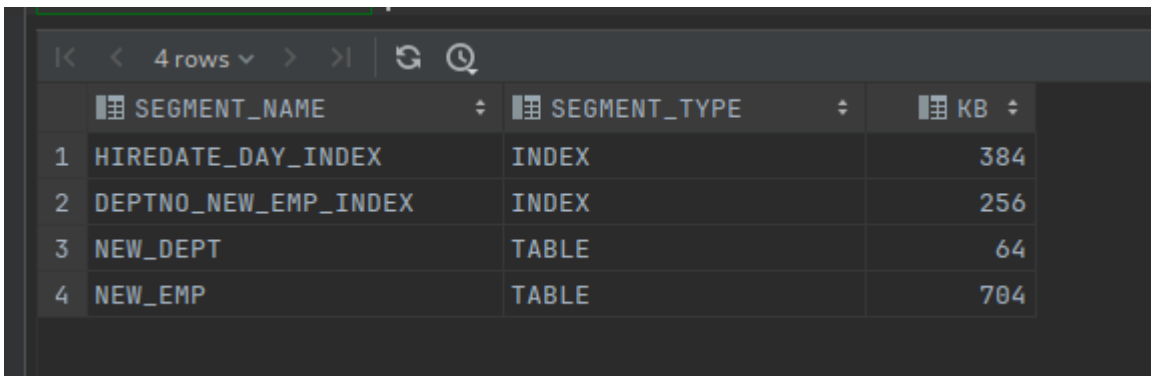
Напишіть запит, що виведе який розмір займають данні таблиць new_emp, new_dept на диску. Скільки місця займають індекси до цих таблиць.

Запит, що буде повертати розмір таблиць та індексів у кілобайтах буде мати наступний вигляд:

```
SELECT SEGMENT_NAME, SEGMENT_TYPE, BYTES/1024 KB FROM DBA_SEGMENTS
WHERE SEGMENT_NAME IN (
    'DEPTNO_NEW_EMP_INDEX', 'NEW_EMP', 'NEW_DEPT',
    'HIREDATE_DAY_INDEX'
)
ORDER BY SEGMENT_TYPE;
```

Ці дані зберігаються в таблиці DBA_SEGMENTS, достатньо лише фільтрувати їх по імені. Також треба зазначити, що доступ до неї є лише у DBA користувача, а не у звичайного.

Розміри таблиць та індексів(без вбудованих по PRIMARY KEY):



The screenshot shows a database query result with 4 rows. The columns are SEGMENT_NAME, SEGMENT_TYPE, and KB. The rows are: 1. HIREDATE_DAY_INDEX (INDEX, 384 KB), 2. DEPTNO_NEW_EMP_INDEX (INDEX, 256 KB), 3. NEW_DEPT (TABLE, 64 KB), 4. NEW_EMP (TABLE, 704 KB).

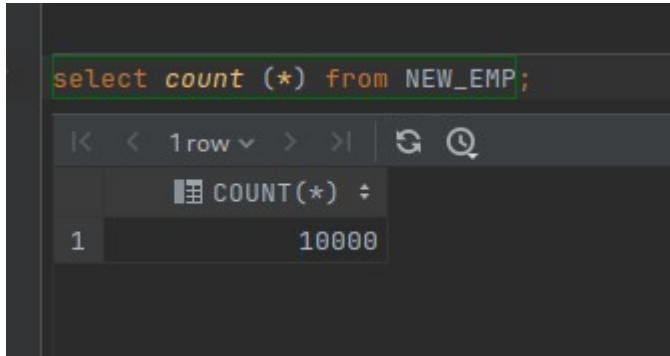
	SEGMENT_NAME	SEGMENT_TYPE	KB
1	HIREDATE_DAY_INDEX	INDEX	384
2	DEPTNO_NEW_EMP_INDEX	INDEX	256
3	NEW_DEPT	TABLE	64
4	NEW_EMP	TABLE	704

Видалить частину (~50% співробітників). Який розмір таблиць і індексів тепер?

Видалимо всіх співробітників, що мають непарні EMPNO:

```
delete from NEW_EMP where MOD(EMPNO, 2) ≠ 0;
```

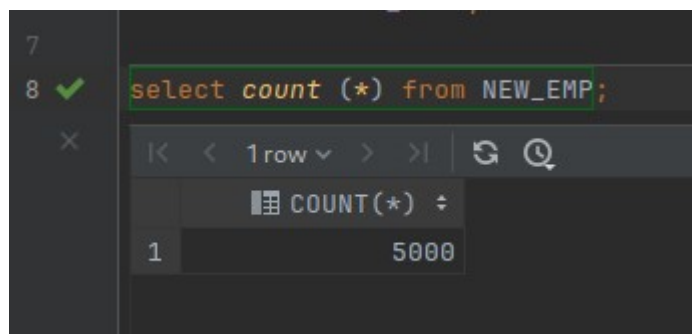
До видалення:



The screenshot shows a SQL query window with the query `select count (*) from NEW_EMP;` highlighted. Below the query, the results are displayed in a table with one row and one column, showing a count of 10000.

COUNT(*)
10000

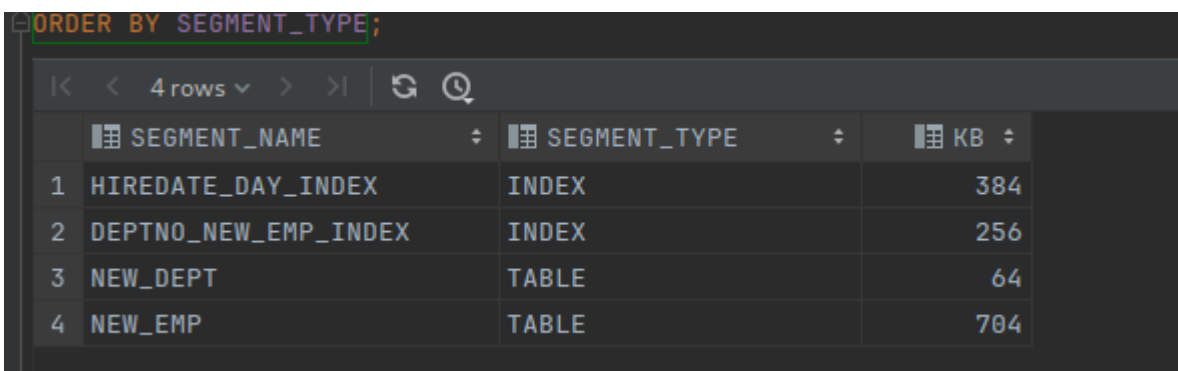
Після:



The screenshot shows the same SQL query window after execution. The results table now shows a count of 5000.

COUNT(*)
5000

Після видалення записів розмір сегментів не змінився:



The screenshot shows a query window with the query `ORDER BY SEGMENT_TYPE;` highlighted. The results are displayed in a table with three columns: SEGMENT_NAME, SEGMENT_TYPE, and KB. The table lists four segments: HIREDATE_DAY_INDEX (INDEX, 384 KB), DEPTNO_NEW_EMP_INDEX (INDEX, 256 KB), NEW_DEPT (TABLE, 64 KB), and NEW_EMP (TABLE, 704 KB).

	SEGMENT_NAME	SEGMENT_TYPE	KB
1	HIREDATE_DAY_INDEX	INDEX	384
2	DEPTNO_NEW_EMP_INDEX	INDEX	256
3	NEW_DEPT	TABLE	64
4	NEW_EMP	TABLE	704

Це пояснюється тим, що Oracle за замовчування не звільняє виділене місце на диску після delete запитів. Натомість цей алокований простір буде використаний у майбутньому для нових рядків.