МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

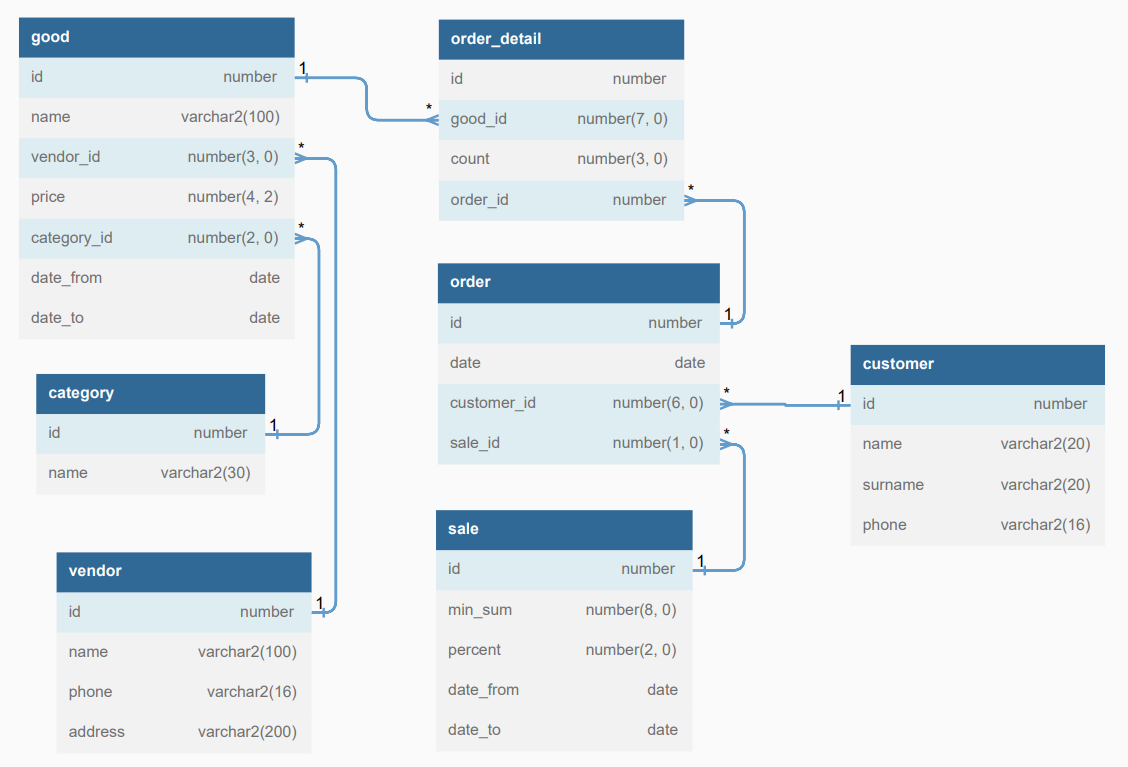
Федеральное государственное автономное образовательное учреждение

высшего образования

«Национальный исследовательский ядерный университет «МИФИ»»

ЛАБОРАТОРНАЯ РАБОТА №2-3:

«Индексирование и кластеризация данных.»



Выполнила студентка группы Б19-515

Щербакова Александра

Москва, 2023 г.

**1. План выполнения запросов.**

Рассмотрим 5 типичных для данной БД запросов – на чтение/добавление/изменение таблиц. Для каждого проанализируем план запроса, будем двигаться от простых запросов к более сложным. Код всех запросов в приложении A.

Примечание: вручную индексы не создавались, но Oracle автоматические создает индекс при назначении столбца в качестве первичного ключа. По умолчанию применяется схема индексации B-дерево.

1. Добавление нового покупателя (client\_manager).

Изображение выглядит как текст

Автоматически созданное описание

Для сравнения выполним тот же запрос, но id добавляемого покупателя захардкодим:

Изображение выглядит как текст

Автоматически созданное описание

Видим, что в первом случае выполняется полное сканирование (сортировка) для поиска наибольшего значение id что увеличивает стоимость операции на 2.

Индекс используется только в первом случае (shop\_main\_admin.customer\_pk – это первичный ключ таблицы, собственно, рассматриваемый индекс).

1. Изменить информацию о товаре, например, продлить срок действия цены (prch\_manager)

Изображение выглядит как стол

Автоматически созданное описание

Сначала выполняется уникальное сканирование по индексу, затем доступ к таблице через операцию index rowid с использованием полученного rowed. Далее доступ через полное сканирование, так как столбец name не проиндексирован.

1. Вычислить сумму проданных товаров из каждой категории (analytic)

Изображение выглядит как стол

Автоматически созданное описание

Видим трижды полное сканирование таблиц. Доступов по индексу нет, что логично исходя из текста запроса.

1. Вычислить сумму продаж за определенное время (accounting)

Изображение выглядит как текст

Автоматически созданное описание

Посмотреть план запроса не получилось из-за недостатка привилегий. Проблему можно решить, либо выдав пользователю привилегии на всю таблицу sale (вместо представления saleV), либо открыв план запроса от имени админа.

Пойдем первым путем:

Изображение выглядит как текст

Автоматически созданное описание

Итак, сам запрос:

Изображение выглядит как текст

Автоматически созданное описание

И его план:

Изображение выглядит как стол

Автоматически созданное описание

Видим дважды использование индексов в операциях index \*table\_name\* unique scan, соответствующие строчкам 6 и 7 запроса.

Странно, что не показаны значения cardinality и cost для строк index unique scan. С чем это связано – не смогла разобраться.

1. Вычислить сумму покупок клиента (client\_manager)

Запрос:

Изображение выглядит как текст

Автоматически созданное описание

План запроса:

Изображение выглядит как стол

Автоматически созданное описание

Как и в прошлом запросе, видим два использования индексов в операциях index \*table\_name\* unique scan. Значения cardinality и cost для этой операции 1 и 0 соответственно, что делает ее самой дешевой из всех операций данного запроса.

**2. Попытка обоснования полезности индексов.**

1. Тест 1 – с индексом / без индекса для первичного ключа.

В примерах выше таблицы заполнены данными максимум на 20 строк, так что разница в производительности между поиском по индексу и полным сканированием таблицы не видна. Попробуем заполнить таблицу good до 250 строк, сгенерировав однотипные значения на питоне.

Изображение выглядит как текст

Автоматически созданное описание

Теперь выполним запрос #5 с индексированным атрибутом id:

Изображение выглядит как стол

Автоматически созданное описание

В самое нижней строке видим новое значение cardinality = 250, однако cost =3 не изменилось. Видимо, нужен бОльший размер таблиц, чтобы увидеть здесь разницу.

Теперь удалим индекс и еще раз посмотрим на план запроса.

Изображение выглядит как стол

Автоматически созданное описание

Две нижние строки с таблицей good просто пропали.

1. Тест 2 – с индексом / без индекса для не первичного ключа

Попробуем обнаружить разницу другим путем - добавим индекс в атрибут order\_detail.order\_id, так как этот столбец очень часто участвует в соединениях.

Изображение выглядит как текст

Автоматически созданное описание

Выполним все тот же запрос #5:

Изображение выглядит как стол

Автоматически созданное описание

Наконец видим разницу: в строке shop\_main\_admin.order\_detail без индекса было полное сканирование таблицы (cardinality = 11, cost = 3), а с индексом эта операция разбилась на две – index range scan (cardinality = 3, cost = 0) и table access by index rowid (cardinality = 2, cost = 1).

1. Тест 3 – с индексом в столбце, выборка большого/маленького количества данных.

Добавим индекс столбцу good.category\_id.

Таблица good на данный момент выглядит так: 245 видов яблок (category 1) и один огурец (category 2).

При выборке большого процента данных oracle автоматически применяет table access full.

Изображение выглядит как текст

Автоматически созданное описание

А при выборке малого процента данных – поиск по индексу.

Изображение выглядит как текст

Автоматически созданное описание

**Выводы**:

+ выгодно индексировать столбцы, участвующие в многотабличных операциях соединения;

+ выгодно индексировать столбцы, из которых типичные запросы извлекают маленький процент данных;

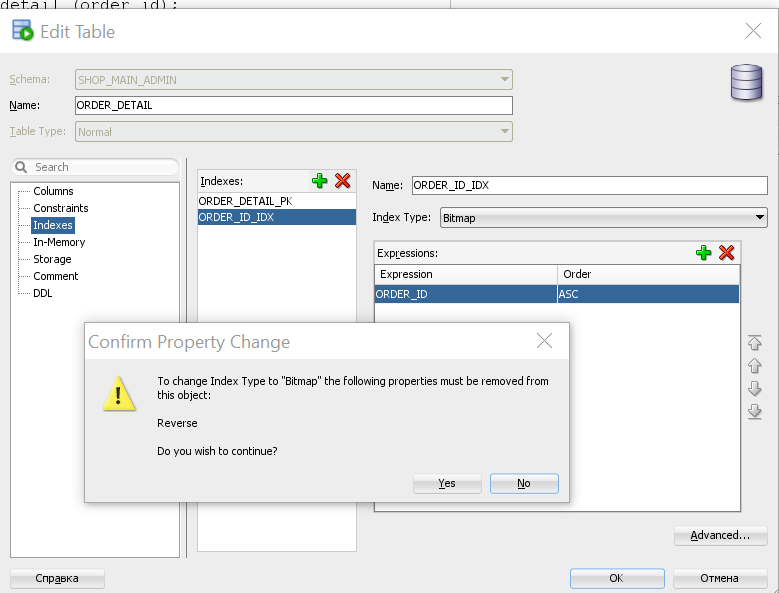
- злоупотреблять индексированием для очень маленьких таблиц не стоит, так как это будет невыгодно по памяти, и скорости работы

- для столбцов, которые часто обновляются, также не стоит применять индексирование из-за накладных расходов на изменение b-дерева

**3. Индексы на основе битовых карт.**

1. Тест 1 - сложный запрос с соединениями.

В теории говорится, что индексы на основе битовых карт подходят для столбцов, содержащих сравнительно мало различных значений. Столбец order\_detail.order\_id должен содержать примерно на порядок меньше значений, чем всего строк в таблице order\_detail. Изменим тип индекса на битовую карту:



Все тот же запрос #5:

Изображение выглядит как стол

Автоматически созданное описание

План запроса совпадает с планом, когда на атрибуте order\_detail.order\_id не было индекса вообще. Почему так – непонятно.

1. Тест 2 – простой запрос.

Изменим индекс good.category\_id на bitmap и сравним с результатами теста 3 из части 3.

Изображение выглядит как текст

Автоматически созданное описание

Операция access table full заменилась на три операции: вычисление одного значения индекса, перевод в rowids и вывод найденных строк. Заметим, что стоимость уменьшилась на 1.

**4. Индекс-таблицы**

Индекс-таблицы целесообразно применять, если выполняются следующие условия:

- таблица редко обновляется;

- в таблице очень много строк;

- часто выполняются запросы на основе первичного ключа;

- таблица не содержит данные long и lob.

Таблица vendor подходит по критериям. Создадим ее копию, но организованную по индексу (для этого также надо создать два tablespace). Код в приложении B.

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Вставим в таблицу 250 строк при помощи генерации строк на питоне:

Изображение выглядит как текст

Автоматически созданное описание

Выполним запрос: найти кол-во позиций товаров, поставляемых каждым вендором.

Запрос для таблиц vendor и vendors\_new (обе таблицы заполнены до 250 строк):

Изображение выглядит как текст

Автоматически созданное описание Изображение выглядит как текст

Автоматически созданное описание

План запроса для vendor:

Изображение выглядит как стол

Автоматически созданное описание

План запроса для vendors\_new:

Изображение выглядит как стол

Автоматически созданное описание

Видим, что таблица без индексов отработала быстрее. В теории на бОльшем объеме данных индекс-таблица должна выигрывать: на скрине видим операции unique scan и fast full scan, которые должны быть предпочтительнее.

**5. Кластеризация**

Для кластеризации выгодно выбирать таблицы с совпадающими столбцами, которые часто приходится соединять. Скорость доступа связанных таблиц возрастет, но при этом уменьшится производительность оператора insert.

В рассматриваемой модели базы данных нет таблиц с совпадающими столбцами, а в задании лабораторной работы не сказано создавать новые таблицы, если существующие не подходят для кластеризации. Что ж, придется пропустить 6 пункт.

**6. Хеш-кластер.**

В хеш-кластер целесообразно помещать таблицу(ы), если выборки в основном производятся по одному и тому же столбцу. Простейший случай – одна таблица. Среди имеющихся таблиц для этого подойдет, например, vendor.

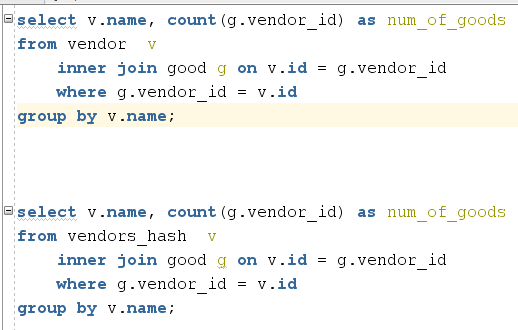
Создадим новое табличное пространство, хеш-кластер и новую таблицу vendors\_hash. Скопируем в нее все строки из vendor. Код в приложении C.

Изображение выглядит как текст

Автоматически созданное описание

Выполним запрос из части 4 отчета для обеих таблиц и сравним планы запросов.

Сам запрос:



План для vendor:

Изображение выглядит как стол

Автоматически созданное описание

План для vendors\_hash:

Изображение выглядит как стол

Автоматически созданное описание

Запрос в хеш-кластере работает быстрее за счет операции index uqinue scan.

**7. Заключение**

В данной работе рассмотрены методы оптимизации работы с таблицами в БД: индексы на основе B-деревьев и битовых карт, индекс-таблицы, хеш-кластеры. Изучены теоретические основы методов, а также экспериментальным путем получены рекомендации по их применению:

1. Все перечисленные методы оптимизации целесообразно применять только на больших таблицах, которые редко обновляются. Эти методы ускоряют выборки select, но замедляют операции вставки/удаления из-за накладных расходов.
2. Индексы может быть выгодно применять для столбцов, которые часто участвуют в многотабличных операциях соединения. Причем B-дерево используется, если извлекается малый процент данных, а BitMap, если большой (столбец содержит мало различных значений).
3. Индекс-таблица и хеш-кластер применяются, чтобы ускорить работу с таблицей, которая используется в основном для выборок по одному столбцу.

**Приложение A. Типичные запросы.**

-- 1) Добавить нового клиента (client\_manager)

insert into customer values

((select max(id) + 1 from customer),

'Egor', 'Egorov', '80001112233');

-- 2) Изменить информацию о товаре, например, продлить срок действия цены (prch\_manager)

update good

set date\_to = (select date\_to from good where id = 1) + 30

where name = 'apple';

-- 3) Вычислить сумму проданных товаров из каждой категории (analytic)

select c.name, sum(count\*price) as sum

from order\_detail od

inner join good g on od.good\_id = g.id

inner join category c on g.category\_id = c.id

group by c.name;

-- 4) Вычислить сумму продаж за определенное время (accounting)

select

sum(g.price\*od.count\*(1-s.percent/100)) as result

from

orders o

inner join order\_detail od on o.id = od.order\_id

inner join saleV s on o.sale\_id = s.id

inner join good g on od.good\_id = g.id

where o."date" between '01.01.23' and '31.01.23';

-- 5) Вычислить сумму покупок клиента (client\_manager)

select

sum(g.price\*od.count\*(1-s.percent/100)) as result

from customer c

inner join orders o on c.id = o.customer\_id

inner join order\_detail od on o.id = od.order\_id

inner join saleV s on o.sale\_id = s.id

inner join goodV g on od.good\_id = g.id

where c.surname = 'Ivanov';

**Приложение B. Создание индекс-таблицы.**

-- create iot

CREATE TABLESPACE vendors\_tbs

DATAFILE 'C:\APP\ALEXM\PRODUCT\18.0.0\ORADATA\XE\SHOP\_PDB\VENDORS\_TBS.dbf'

SIZE 500M;

CREATE TABLESPACE overflow\_tables

DATAFILE 'C:\APP\ALEXM\PRODUCT\18.0.0\ORADATA\XE\SHOP\_PDB\OVERFLOW\_TABLES.dbf'

SIZE 500M;

CREATE TABLE vendors\_new(

id number(4, 0),

name varchar2(100) NOT NULL,

phone varchar2(16) NOT NULL,

address varchar2(200),

CONSTRAINT vendors\_new\_pk PRIMARY KEY (id))

ORGANIZATION INDEX TABLESPACE vendors\_tbs

PCTTHRESHOLD 25

INCLUDING name

OVERFLOW TABLESPACE overflow\_tables;

**Приложение С. Создание хеш-кластера.**

-- create hash-cluster

CREATE TABLESPACE hash\_cluster

DATAFILE 'C:\APP\ALEXM\PRODUCT\18.0.0\ORADATA\XE\SHOP\_PDB\HASH\_CLUSTER.dbf'

SIZE 500M;

CREATE CLUSTER vendor\_hash\_cluster (

id number(4, 0),

name varchar2(100),

phone varchar2(16),

address varchar2(200)

)

hash is id HASHKEYS 1024

size 1024 single table

tablespace hash\_cluster;

CREATE TABLE vendors\_hash (

id number(4, 0),

name varchar2(100) NOT NULL,

phone varchar2(16) NOT NULL,

address varchar2(200),

CONSTRAINT vendors\_hash\_pk PRIMARY KEY (id)

)

CLUSTER vendor\_hash\_cluster (id, name, phone, address);

insert into vendors\_hash select \* from vendor;