**Индексы. B\*-tree индексы. Роль индексов в оптимизации. Невидимые индексы. Виртуальные индексы. Измерение времени исполнения запроса. Понятие о нагрузочном тестировании.**

Ссылка на основную статью Oracle:

<https://docs.oracle.com/en/database/oracle/oracle-database/18/cncpt/indexes-and-index-organized-tables.html#GUID-797E49E6-2DCE-4FD4-8E4A-6E761F1383D1>

**Индексы Oracle** обеспечивают быстрый доступ к строкам таблиц в базе данных, сохраняя отсортированные значения указанных столбцов и используя эти отсортированные значения для быстрого нахождения ассоциированных строк таблицы — во многом подобно тому, как применяется предметный указатель в конце книги для быстрого нахождения определенного места. Индексы позволяют находить строку с определенным значением столбца, просматривая при этом лишь небольшую часть общего объема строк таблицы. Таким образом, правильное использование индексов сокращает до минимума количество дорогостоящих операций ввода-вывода. Индексы — необязательные структуры базы данных, поддерживаемые полностью самой системой Oracle Database.

Применение индексов представляет собой компромисс между ускорением получения результатов запросов и замедлением обновлений и вставок данных. Первая часть этого компромисса — ускорение запросов — довольно очевидна: если поиск выполняется по отсортированному индексу вместо полного сканирования всей таблицы, то запрос проходит намного быстрее. Но всякий раз, когда вы обновляете, вставляете или удаляете строку таблицы с индексами, индексы также должны быть обновлены соответствующим образом. То есть такие операции на таблицах с индексами обходятся дороже. Вдобавок стоит помнить, что огромные таблицы будут иметь огромные индексы, и для их хранения потребуется диск большего объема.

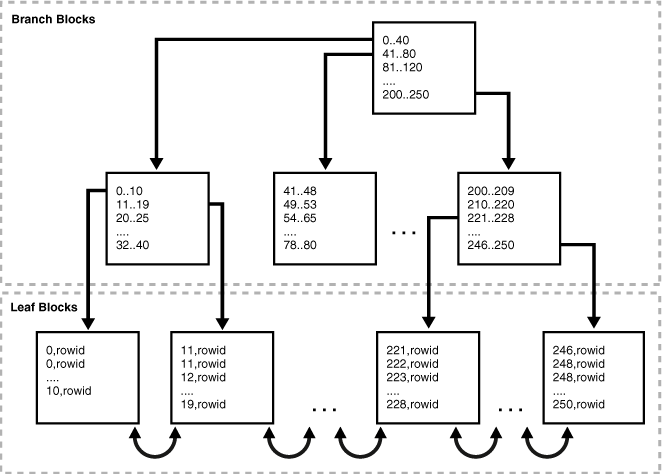
Вообще говоря, если таблицы в основном используются для чтения (выборки) информации, как в хранилищах данных, то лучше иметь много индексов. Если же база данных относится к типу OLTP, с большим количеством вставок, обновлений и удалений, то лучше обойтись меньшим числом индексов.

Если только вам не нужно обращаться к большинству строк таблицы, индексированные запросы обеспечивают более быстрое получение результатов, чем запросы, не использующие индексы. Не существует ограничений на количество индексов, которые могут относиться к одной таблице Oracle, но, как упоминалось ранее, от их количества зависит производительность. Индекс полностью прозрачен для пользователя — т.е. оператор SQL пользователя не должен изменяться в результате создания индексов. Однако разработчикам приложений для построения эффективных запросов следует хорошо представлять себе, что такое индексы и как они работают.

**Виды индексов Oracle Database**

Индексы Oracle могут относиться к нескольким видам, наиболее важные из которых перечислены ниже.

* **Уникальные и неуникальные индексы**. Уникальные индексы основаны на уникальном столбце — обычно вроде номера карточки социального страхования сотрудника. Хотя уникальные индексы можно создавать явно, Oracle не рекомендует это делать. Вместо этого следует использовать уникальные ограничения. Когда накладывается ограничение уникальности на столбец таблицы, Oracle автоматически создает уникальные индексы по этим столбцам.
* **Первичные и вторичные индексы**. Первичные индексы — это уникальные индексы в таблице, которые всегда должны иметь какое-то значение и не могут быть равны null. Вторичные индексы — это прочие индексы таблицы, которые могут и не быть уникальными.
* **Составные индексы**. Составные индексы — это индексы, содержащие два или более столбца из одной и той же таблицы. Они также известны как сцепленные индексы (concatenated index). Составные индексы особенно полезны для обеспечения уникальности сочетания столбцов таблицы в тех случаях, когда нет уникального столбца, однозначно идентифицирующего строку.



### Индексы и ключи

Часто можно встретить взаимозаменяемое употребление терминов “индекс” и “ключ”. Тем не менее, эти две сущности на самом деле отличаются друг от друга. Индекс — это физическая структура, хранящаяся в базе данных. Индекс можно создавать, изменять и уничтожать; в основном он служит для ускорения доступа к данным таблицы. С другой стороны, ключи — полностью логическая концепция. Ключи, с другой стороны, являются чисто логическим концепциями. Они представляют ограничения целостности, создаваемые для реализации бизнес-правил. Путаница между индексами и ключами обычно возникает потому, что база данных часто использует индекс для обеспечения ограничения целостности. Просто помните, что эти две вещи — не одно и то же.

## Руководство по созданию индексов

Хотя хорошо известно, что индексы повышают производительность базы данных, следует знать, как их заставить работать должным образом. Добавление ненужных или неподходящих индексов к таблице может даже привести к снижению производительности. Ниже предоставлены некоторые рекомендации по созданию эффективных индексов в базе данных Oracle.

* Индексация имеет смысл, если нужно обеспечить доступ одновременно не более чем к 4–5% данных таблицы. Альтернативой использованию индекса для доступа к данным строки является полное последовательное чтение таблицы от начала до конца, что называется полным сканированием таблицы. Полное сканирование таблицы больше подходит для запросов, которые требуют извлечения большего процента данных таблицы. Помните, что применение индексов для извлечения строк требует двух операций чтения: индекса и затем таблицы.
* Избегайте создания индексов для сравнительно небольших таблиц. Для таких таблиц больше подходит полное сканирование. В случае маленьких таблиц нет необходимости в хранении данных и таблиц, и индексов.
* Создавайте первичные ключи для всех таблиц. При назначении столбца в качестве первичного ключа Oracle автоматически создает индекс по этому столбцу.
* Индексируйте столбцы, участвующие в многотабличных операциях соединения.
* Индексируйте столбцы, которые часто используются в конструкциях WHERE.
* Индексируйте столбцы, участвующие в операциях ORDER BY и GROUP BY или других операциях, таких как UNION и DISTINCT, включающих сортировку. Поскольку индексы уже отсортированы, объем работы по выполнению необходимой сортировки данных для упомянутых операций будет существенно сокращен.
* Столбцы, состоящие из длинно-символьных строк, обычно плохие кандидаты на индексацию.
* Столбцы, которые часто обновляются, в идеале не должны быть индексированы из-за связанных с этим накладных расходов.
* Индексируйте таблицы только с высокой селективностью. То есть индексируйте таблицы, в которых мало строк имеют одинаковые значения.
* Сохраняйте количество индексов небольшим.
* Составные индексы могут понадобиться там, где одностолбцовые значения сами по себе не уникальны. В составных индексах первым столбцом ключа должен быть столбец с максимальной селективностью.

Всегда помните золотое правило индексации таблиц: индекс таблицы должен быть основан на типах запросов, которые будут выполняться над столбцами этой таблицы. На таблице можно создавать более одного индекса; например, можно создать индекс на столбце X, или столбце Y, или обоих сразу, а также один составной индекс на обоих столбцах. Принимая правильное решение относительно того, какие индексы следует создавать, подумайте о наиболее часто используемых типах запросов данных таблицы.

**Схемы индексации Oracle**

Oracle предлагает несколько схем индексации, соответствующих требованиям различных типов приложений. На фазе проектирования после тщательного анализа конкретных требований приложения необходимо выбрать правильный тип индекса.

В реализации индексов на основе B-деревьев используется концепция сбалансированного (на что указывает буква “B” (balanced)) дерева поиска в качестве основы структуры индекса. В Oracle имеется собственный вариант B-дерева, именуемый “B\*tree”. Это обычные индексы, создаваемые по умолчанию, когда вы применяете оператор CREATE INDEX. Термин “индекс B\*tree” обычно не используется, когда речь идет об обычных индексах Oracle — они называются просто “индексами”.

Индексы на основе B-деревьев структурированы в форме обратного дерева, где блоки верхнего уровня называются блоками ветвей (branch blocks), а блоки нижнего уровня — листовыми блоками (leaf blocks). В иерархии узлов все узлы кроме вершины, или корневого узла, имеют родительский узел и могут иметь ноль или более дочерних узлов. Если глубина древовидной структуры, т.е. количество уровней, одинакова от каждого листового блока до корневого узла, то такое дерево называется сбалансированным, или B-деревом.

B-деревья автоматически поддерживают необходимый уровень индекса по размеру таблицы. B-деревья также гарантируют, что индексные блоки всегда будут заполнены не меньше, чем наполовину, и менее, чем на 100%. B-деревья допускают операции выборки, вставки и удаления с очень небольшим количеством операций ввода-вывода на один оператор. Большинство B-деревьев имеет всего три и менее уровней. При использовании B-дерева нужно читать только блоки B-дерева, так что количество операций ввода-вывода будет ограничено числом уровней B-дерева (скажем, тремя) плюс две операции ввода-вывода на выполнение обновления или удаления (одна для чтения и одна для записи). Для выполнения поиска по B-дереву понадобится всего три или менее обращений к диску.

Реализация B-дерева от Oracle — B\*tree — всегда сохраняет дерево сбалансированным. Листовые блоки содержат по два элемента: индексированные значения столбца и соответствующий идентификатор ROWID для строки, которая содержит это значение столбца. ROWID — уникальный указатель Oracle, идентифицирующий физическое местоположение строки и обеспечивающий самый быстрый способ доступа к строке в базе данных Oracle. Сканирование индекса быстро дает ROWID строки, и отсюда можно быстро получить к ней доступ непосредственно. Если запрос нуждается лишь в значении индексированного столбца, то конечно, последний шаг исключается, поскольку извлекать дополнительные данные, кроме прочитанных из индекса, не потребуется.

**Оценка размера индекса**

Как и в случае таблиц, для оценки размера нового индекса можно использовать пакет DBMS\_SPACE. Процедуре CREATE\_INDEX\_COST этого пакета потребуется передать оператор DDL, создающий индекс, в качестве атрибута, как показано в листинге ниже.

SET SERVEROUTPUT ON

declare

l\_index\_ddl VARCHAR2(1000);

l\_used\_bytes NUMBER;

l\_allocated\_bytes NUMBER;

BEGIN

DBMS\_SPACE.create\_index\_cost (

ddl => 'create index persons\_idx on persons(person\_id)',

used\_bytes => l\_used\_bytes,

alloc\_bytes => l\_allocated\_bytes);

DBMS\_OUTPUT.PUT\_LINE ('используется = ' || l\_used\_bytes || 'байт'

|| ' выделено = ' || l\_allocated\_bytes || 'байт');

END;

SQL> /

используется = 154414918 байт выделено = 427720704 байт

PL/SQL procedure successfully completed

Обратите внимание на отличие между атрибутами, касающимися размера, в процедуре CREATE\_INDEX\_COST:

* used\_bytes показывает количество байт, которыми представлены данные индекса;
* alloc\_bytes показывает количество байт, которое займет индекс в табличном пространстве после его создания.

**Совет.** Для того чтобы пакет DBMS\_SPACE мог адекватно оценить размеры индексов, таблица, на которой планируется создание нового индекса, должна существовать, а база данных должна иметь актуальную статистику по этой таблице.

## Создание индекса

Индекс создается с помощью оператора CREATE INDEX, как показано ниже:

CREATE INDEX employee\_id ON employee(employee\_id)

TABLESPACE emp\_index\_01;

При создании индекса для большой таблицы, уже заполненной данными, статистику оптимизатора можно собрать во время создания таблицы, специфицировав опцию COMPUTE STATISTICS, как показано в следующем примере:

CREATE INDEX employee\_id ON employee(employee\_id)

TABLESPACE emp\_index\_01

COMPUTE STATISTICS;

Если не специфицировать настройки хранения, база данных использует опции хранения по умолчанию табличного пространства, которое было указано при создании индекса.

По умолчанию Oracle допускает дублированные значения в столбцах индекса, которые также называются ключевыми столбцами. Однако можно специфицировать уникальный индекс, что исключит дублирование значений столбца в нескольких строках. Для создания уникального индекса служит оператор CREATE UNIQUE INDEX:

CREATE UNIQUE INDEX employee\_id ON employee(employee\_id)

TABLESPACE emp\_index\_01;

Приведенные до сих пор примеры демонстрировали создание индексов на одиночных столбцах. Также можно создать составной индекс на таблице, специфицируя несколько столбцов в операторе CREATE INDEX, как показано в следующем примере:

CREATE INDEX employee\_id ON employee(employee\_id,location\_id)

TABLESPACE emp\_index\_01;

Все примеры создания индекса до сих пор демонстрировали явное создание индекса на столбце таблицы. Тем не менее, есть и другой способ создания индекса на таблице, который заключается в простой спецификации ограничений целостности UNIQUE или PRIMARY KEY на этой таблице. Если поступить так, Oracle автоматически создает уникальный индекс по уникальному или первичному ключу. База данных создаст индекс автоматически, когда будет включено ограничение, и по умолчанию он получит имя соответствующего ограничения. Ниже приведено два примера, демонстрирующие ситуации, когда база данных создает автоматический индекс на столбцах таблицы.

В первом случае задается уникальное ограничение на двух столбцах: dept\_name и location.

SQL> CREATE TABLE dept(

dept\_no NUMBER(3),

dept\_name VARCHAR2(15),

location VARCHAR2(25),

CONSTRAINT dept\_name\_ukey UNIQUE(dept\_Name,location);

База данных автоматически создает уникальный индекс по этим двум столбцам,чтобы обеспечить соблюдение ограничения уникальности по имени dept\_name\_ukey.

Во втором примере показано, как при создании таблицы специфицировать ограничение первичного ключа на столбце.

CREATE TABLE employee (

empno NUMBER (5) PRIMARY KEY, age INTEGER)

ENABLE PRIMARY KEY USING INDEX

TABLESPACE users;

Приведенный выше оператор CREATE TABLE включает ограничение первичного ключа, которое автоматически создает уникальный индекс на столбце empno.

Можно также указать, что база данных должна использовать существующий индекс для обеспечения нового ограничения, как показано в следующем примере:

SQL> ALTER TABLE employee ADD CONSTRAINT test\_const1

PRIMARY KEY (pkey1) USING INDEX ind1;

В этом примере новый первичный ключ использует существующий индекс ind1, без создания нового индекса. Интересно то, что оператор CREATE INDEX можно специфицировать при создании ограничения уникальности или первичного ключа. В следующем примере создается первичный ключ на столбце emp\_id:

CREATE TABLE employee (

emp\_id INT PRIMARY KEY USING INDEX (create index ind1

ON employee (emp\_id)))

Применение оператора CREATE INDEX в этом примере обеспечивает более тонкий контроль над созданием индекса для указанного ограничения первичного ключа.

**Специальные типы индексов**

Нормальный или типовой индекс, который создается в базе данных, называется индексом кучи (heap index), или неупорядоченным индексом. Oracle также предоставляет несколько специальных типов индексов для специфических нужд. Сейчас рассмотрим основные типы индексов.

**Битовые индексы**

Битовые индексы (bitmap indexes) используют битовые карты для указания значения индексированного столбца. Это идеальный индекс для столбца с низкой кардинальностью при большом размере таблицы. Эти индексы обычно не годятся для таблиц с интенсивным обновлением, но хорошо подходят для приложений хранилищ данных.

Битовые индексы состоят из битового потока (единиц и нулей) для каждого столбца индекса. Битовые индексы очень компактны по сравнению с нормальными индексами на основе B-деревьев. В табл. 7.2 дано сравнение индексов B-деревьев с битовыми индексами.

|  |  |
| --- | --- |
| **Индексы B-деревьев** | **Битовые индексы** |
| Хороши для данных с высокой кардинальностью | Хороши для данных с низкой кардинальностью |
| Хороши для баз данных OLTP | Хороши для приложений хранилищ данных |
| Занимают много места | Используют относительно мало места |
| Легко обновляются | Трудно обновляются |

Для создания битового индекса используется оператор CREATE INDEX с добавочным ключевым словом BITMAP:

SQL> CREATE BITMAP INDEX gender\_idx ON employee(gender)

TABLESPACE emp\_index\_05;

Иногда можно наблюдать значительное повышение производительности при замене обычных индексов B\*tree на битовые в некоторых очень крупных таблицах. Однако каждый элемент битового индекса покрывает огромное количество строк в таблице, так что когда данные обновляются, вставляются или удаляются из таблицы, то необходимые обновления битового индекса очень велики, и сам индекс может существенно увеличиться в размере. Единственный способ обойти это увеличение размера индекса с последующим падением производительности заключается в регулярной его перестройке. Вы можете сделать вывод, что битовый индекс — не слишком разумная альтернатива для таблиц, подвергающихся большому количеству вставок, удалений и обновлений.

**Индексы с реверсированным ключом**

Индексы с реверсированным ключом — это, по сути, то же самое, что и индексы B-деревьев, за исключением того, что байты данных ключевого столбца при индексации меняют порядок на противоположный. Порядок столбцов остается нетронутым; меняется только порядок байтов. Самое большое преимущество применения индексов с реверсированным ключом состоит в том, что они исключают неприятные последствия упорядоченной вставки значений в индекс. Вот как создается индекс с реверсированным ключом:

CREATE INDEX reverse\_idx ON employee(emp\_id) REVERSE;

При использовании индекса с реверсированным ключом база данных не сохраняет ключи индекса друг за другом в лексикографическом порядке. Таким образом, когда в запросе присутствует предикат неравенства, ответ получается медленнее, поскольку база данных вынуждена выполнять полное сканирование таблицы. При индексе с реверсированным ключом база данных не может запустить запрос по диапазону ключа индекса.

**Индексы со сжатым ключом**

Сэкономить пространство хранения индекса вместе с повышением производительности можно за счет создания индекса со сжатым ключом. Всякий раз, когда индексируемый ключ имеет повторяющийся компонент, или же создается уникальный многостолбцовый индекс, получается выигрыш от использования сжатия ключа. Вот пример:

SQL> CREATE INDEX emp\_indx1 ON employees(ename)

TABLESPACE users

COMPRESS 1;

Приведенный выше оператор сжимает все дублированные вхождения индексированного ключа в листовом блоке индекса (на уровне 1).

**Индексы на основе функций (виртуальные столбцы)**

Индексы на основе функций предварительно вычисляют значения функций по заданному столбцу и сохраняют результат в индексе. Когда конструкция where содержит вызовы функций, то основанные на функциях индексы являются идеальным способом индексирования столбца.

Ниже показано, как создать индекс на основе функции LOWER:

CREATE INDEX lastname\_idx ON employees(LOWER(l\_name));

Этот оператор CREATE INDEX создаст индекс по столбцу l\_name, хранящему фамилии сотрудников в верхнем регистре. Однако этот индекс будет основан на функции, поскольку база данных создает его по столбцу l\_name, применив к нему предварительно функцию LOWER для преобразования его значения в нижний регистр.

Виртуальные столбцы (Virtual columns). Виртуальные колонки представляют из себя функции. Индексы по таким столбцам — это индексы над функциями. В выражениях могут использоваться: столбцы этой таблицы, константы, SQL-функции или функции, созданные пользователем. Поддерживается также секционирование (не в XE 11). Виртуальные колонки не могут использоваться в индексно-организованных (index organized) или внешних (external) таблицах. Прямая запись в такие колонки невозможна.

Пример:



**Секционированные индексы**

Секционированные индексы используются для индексации секционированных таблиц. Oracle предлагает два типа индексов для таких таблиц: локальные и глобальные.

Существенное различие между ними заключается в том, что локальные индексы основаны на [разделах таблицы](https://oracle-patches.com/oracle/begin/324-%D1%80%D0%B0%D0%B7%D0%B4%D0%B5%D0%BB%D1%8B-oracle), по которой они созданы. Если таблица секционирована на 12 разделов по диапазонам дат, то индексы также будут распределены по тем же 12 разделам. Другими словами, между разделами индексов и разделами таблиц существует соответствие “один к одному”. Такого соответствия нет между глобальными индексами и разделами таблицы, потому что глобальные индексы секционируются независимо от базовых таблиц.

В следующих разделах этой статьи будут раскрыты важные различия между управлением глобально секционированными индексами и локально секционированными индексами.

**Глобальные индексы**

Глобальные индексы на секционированных таблицах могут быть как секционированными, так и несекционированными. Глобальные несекционированные индексы подобны обычным индексам Oracle для несекционированных таблиц. Для создания таких индексов применяется обычный синтаксис CREATE INDEX.

Ниже приведен пример глобального индекса на таблице ticket\_sales:

CREATE INDEX ticketsales\_idx ON ticket\_sales(month)

GLOBAL PARTITION BY range(month)

(PARTITION ticketsales1\_idx VALUES LESS THAN (3)

PARTITION ticketsales1\_idx VALUES LESS THAN (6)

PARTITION ticketsales2\_idx VALUES LESS THAN (9)

PARTITION ticketsales3\_idx VALUES LESS THAN (MAXVALUE);

Обратите внимание, что управление глобально секционированными индексами требует серьезных усилий. Всякий раз, когда происходит какое-то действие DDL над секционированной таблицей, ее глобальные индексы требуют перестройки. Действия DDL над лежащей в основе таблицей помечают глобальные индексы как недействительные. По умолчанию любая операция обслуживания секционированной таблицы делает недействительными глобальные индексы.

Давайте в качестве примера воспользуемся таблицей ticket\_sales, чтобы разобраться, почему это так. Предположим, что вы ежеквартально уничтожаете самый старый раздел, чтобы освободить место для нового раздела, в который поступят данные за новый квартал. Когда уничтожается раздел, относящийся к таблице ticket\_sales, глобальные индексы могут стать недействительными, потому что часть данных, на которые они указывают, перестают существовать. Чтобы предотвратить такое объявление недействительным индекса из-за уничтожения раздела, необходимо использовать опцию UPDATE GLOBAL INDEXES вместе с оператором DROP PARTITION:

SQL> ALTER TABLE ticket\_sales

DROP PARTITION sales\_quarter01

UPDATE GLOBAL INDEXES;

**На заметку!** Если не включить оператор UPDATE GLOBAL INDEXES, то все глобальные индексы станут недействительными. Опцию UPDATA GLOBAL INDEXES можно также использовать при добавлении, объединении, обмене, слиянии, перемещении, разделении или усечении секционированных таблиц. Разумеется, с помощью ALTER INDEX...REBUILD можно перестраивать любого индекса, который становится недействительным, но эта опция также требует дополнительных затрат времени и обслуживания.

При небольшом количестве листовых блоков индекса, что приводит к высокой конкуренции, Oracle рекомендует использовать глобальные индексы с хеш-секционированием. Синтаксис для создания хеш-секционированного глобального индекса подобен тому, что применяется для хеш-секционированной таблицы. Например, следующий оператор создает хеш-секционированный глобальный индекс:

CREATE INDEX hgidx ON tab (c1,c2,c3) GLOBAL

PARTITION BY HASH (c1,c2)

(PARTITION p1 TABLESPACE tbs\_1,

PARTITION p2 TABLESPACE tbs\_2,

PARTITION p3 TABLESPACE tbs\_3,

PARTITION p4 TABLESPACE tbs\_4);

### Локальные индексы

Локально секционированные индексы, в отличие от глобально секционированных индексов, имеют отношение “один к одному” с разделами таблицы. Локально секционированные индексы можно создавать в соответствии с разделами и даже подразделами. База данных конструирует индекс таким образом, чтобы он был секционирован так же, как и его таблица. При каждой модификации раздела таблицы база автоматически сопровождает это соответствующей модификацией раздела индекса. Это, наверное, самое большое преимущество использования локально секционированных индексов — Oracle автоматически перестраивает их всегда, когда уничтожается раздел или над ним выполняется какая-то другая операция DDL.

Ниже приведен простой пример создания локально секционированного индекса на секционированной таблице:

CREATE INDEX ticket\_no\_idx ON

ticket\_sales(ticket\_\_no) LOCAL

TABLESPACE localidx\_01;

**Совет.** С помощью нового инструмента SQL Access Advisor можно получать рекомендации относительно того, какие индексы нужно создать. SQL Access Advisor также сообщит о том, какие индексы не используются и потому являются кандидатами на удаление.

### Невидимые индексы

По умолчанию оптимизатор “видит” все индексы. Тем не менее, можно создать невидимый индекс, который оптимизатор не обнаруживает и не принимает во внимание при создании плана выполнения оператора. Невидимый индекс можно применять в качестве временного индекса для определенных операций или его тестирования перед тем, как сделать его “официальным”. Вдобавок, иногда объявление индекса невидимым можно использовать в качестве альтернативы уничтожению индекса или объявлению его недоступным. Сделать индекс невидимым можно временно, чтобы протестировать эффект от его уничтожения.

База данных поддерживает невидимый индекс точно так же, как и нормальный (видимый) индекс. После объявления индекса невидимым, его и все прочие невидимые индексы можно сделать вновь видимым для оптимизатора, установив значение параметра optimizer\_use\_invisible\_indexes равным TRUE на уровне сеанса или всей системы. Значением этого параметра по умолчанию является FALSE, а это означает, что оптимизатор по умолчанию не может использовать невидимые индексы.

**Создание невидимого индекса**

Чтобы сделать индекс невидимым, к оператору CREATE INDEX нужно добавить конструкцию INVISIBLE, как показано ниже:

CREATE INDEX test\_idx ON test(tname)

TABLESPACE testdata

STORAGE (INITIAL 20K

NEXT 20k

PCTINCREASE 75)

INVISIBLE;

Приведенный выше оператор создает невидимый индекс test\_idx по столбцу tname таблицы test.

**Превращение индекса в невидимый**

В дополнение к созданию невидимого индекса, с помощью команды ALTER INDEX можно превратить существующий индекс в невидимый:

ALTER INDEX test\_idx INVISIBLE;

Чтобы сделать невидимый индекс вновь видимым, используйте следующий оператор:

ALTER INDEX test\_idx VISIBLE;

Приведенные ниже запрос к представлению DBA\_INDEXES показывает состояние видимости индекса:

SELECT INDEX\_NAME, VISIBILITY FROM USER\_INDEXES

WHERE INDEX\_NAME = 'INDX1';

INDEX\_NAME VISIBILITY

---------- ----------

INDX1 VISIBLE

## Мониторинг использования индекса

Oracle предлагает инструменты EXPLAIN PLAN и SQL Trace, которые помогают увидеть путь, проходимый запросом перед его выполнением. Вывод команды EXPLAIN PLAN и результаты SQL Trace позволяют увидеть путь выполнения запроса и определить, использует ли он индексы.

Oracle также предлагает более простой способ слежения за индексами в базе данных. Если вы сомневаетесь в использовании определенного индекса, можете попросить Oracle выполнить мониторинг его применения. Таким образом, если индекс окажется избыточным, его можно уничтожить и сэкономить место в хранилище, а также снизить накладные расходы на операции DML.

Опишем, что потребуется сделать для отслеживания индекса в базе данных. Предположим, что вы пытаетесь узнать, используется ли индекс p\_key\_sales в определенных запросах к таблице sales. Обеспечьте репрезентативный промежуток времени для оценки использования индекса. Для базы данных OLTP этот промежуток может быть относительно коротким. Для хранилища данных может понадобиться запустить тестовый мониторинг на несколько дней, чтобы точно проверить, как используется индекс.

Чтобы запустить мониторинг использования индекса, войдите в базу как владелец индекса p\_key\_sales и запустите следующую команду:

ALTER INDEX p\_key\_sales MONITORING USAGE;

Index altered.

Теперь запустите какие-нибудь запросы к таблице sales. Завершите мониторинг, применив следующую команду:

ALTER INDEX p\_key\_sales NOMONITORING USAGE;

Index altered.

После этого можно запросить представление словаря данных V$OBJECT\_USAGE для определения того, использовался ли индекс p\_key\_sales. Следующий результат подтверждает использование индекса:

SELECT \* FROM v$object\_usage

WHERE index\_name='P\_KEY\_SALES';

INDEX\_NM TAB\_NM MON USED START\_MON END\_MONITORING

---------- ------ ---- ----- ------------------ --------------------

P\_KEY\_SALES SALE NO YES 01/23/2008 06:20:45 01/23/2008 06:40:22

В приведенном выводе Oracle выводит значение YES в столбце USED, указывая на то, что интересующий индекс использовался базой данных. Если индекс был проигнорирован во время мониторинга, столбец содержал бы значение NO. Причина, по которой нельзя узнать количество случаев использования индекса, связана с тем, что база данных выполняет мониторинг его использования только на фазе разбора (parsing); если бы разбор производился при каждом выполнении, пострадала бы производительность.

**Обслуживание индексов**

Данные индекса постоянно изменяются из-за DML-действий, связанных с его таблицей. Индексы часто становятся слишком большими, если происходит много удалений строк, потому что пространство, занятое удаленными значениями, автоматически повторно индексом не используется. За счет периодического применения команды REBUILD можно реорганизовать индексы и сделать их более компактными, а потому и более эффективными. Команда REBUILD также служит для изменения параметров хранения, которые устанавливаются во время начального создания индекса. Вот пример:

ALTER INDEX sales\_idx REBUILD;

Index altered.

Перестройка индексов лучше уничтожения и воссоздания неудачного индекса, потому что при этой операции пользователи продолжают иметь доступ к индексу в процессе его перестройки. Однако индексы в процессе перестройки накладывают много ограничений на действия пользователя. Еще более эффективный способ перестройки индексов состоит в том, чтобы сделать это в оперативном (online) режиме, как показано в следующем примере. Во время оперативной перестройки индекса разрешено применение всех операций DML, но не операций DDL.

ALTER INDEX p\_key\_sales REBUILD ONLINE;

Index altered.

Оперативную перестройку индекса можно ускорить за счет добавления к показанному выше оператору ALTER INDEX конструкции ONLINE NOLOGGING. После добавления этой конструкции база данных не будет генерировать данные повторного выполнения для операции перестройки индекса.

**Измерение времени выполнения запроса**

Измерить время выполнения запроса в sqlplus можно скриптом:

set timing on;

set linesize 1000;

set pagesize 1000;

<SQL statement>;

Результаты запроса выдаются в виде:

Elapsed: 00:00:02.71.

После точки сотые доли секунды.

В этом варианте результаты запроса выводятся не на экран, а в файл:

set timing on;

set linesize 1000;

set pagesize 1000;

set term off;

spool tmp.out;

<SQL statement>;

spool off;

**Измерение общего и CPU времени на запрос**

### v$timer — это простое представление, которое возвращает время в сотых долях секунды, мы будет фиксировать время до и после теста, и затем отнимать одно значение от другого. Состоит из двух полей:

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| hsecs | number | количество времени в сотых долях секунды |
| con\_id | number | идентификатор контейнера (для мультиарендной контейнерной базы данных CDB) |

Функция, которая возвращает время работы процессора (CPU) в сотых долях секунды.

Пример вызова:

select dbms\_utility.get\_cpu\_time from dual;

--результат 815

Логика простая - мы будем записывать значения времени в переменные до выполнения запроса и после, затем вычитать время завершения от времени начала и получать итоговый результат. Вывод результата будет осуществляться с помощью **dbms\_output.put\_line**, чтобы данные выводились в SQL Developer перед выполнением скриптов выполним команду, включающую вывод, а затем будет логика.

--включаем вывод данных в output

set serveroutput on;

declare

v\_t1 number; --время начала

v\_t2 number; --время завершения

v\_cpu1 number; --время ЦПУ до

v\_cpu2 number; --время ЦПУ после

v\_t\_res number; --общее время выполнения

v\_cpu\_res number; --общее время выполнения CPU

begin

--фиксируем начало

--помещаяем общее время и время ЦПУ в переменные

select t.hsecs

,dbms\_utility.get\_cpu\_time

into v\_t1

,v\_cpu1

from v$timer t;

--запустим 1000 раз цикл, который обновит дату у первых попавшихся строк

for rec in 1..1000

loop

update prod.operation

set op\_date = trunc(sysdate)

where rownum = 1

and op\_date != trunc(sysdate);

end loop;

--фиксируем завершение

--снова помещаяем общее время и время ЦПУ в переменные

select t.hsecs

,dbms\_utility.get\_cpu\_time

into v\_t2

,v\_cpu2

from v$timer t;

--считаем общее время выполнения в сотых долях секунды

v\_t\_res := v\_t2 - v\_t1;

--считаем общее время ЦПУ в сотых долях секунды

v\_cpu\_res := v\_cpu2 - v\_cpu1;

--выводим результат на экран

dbms\_output.put\_line('Общее время выполнения в секундах: '||to\_char(v\_t\_res/100,'0.00'));

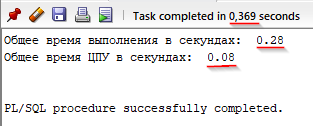
dbms\_output.put\_line('Общее время ЦПУ в секундах: '||to\_char(v\_cpu\_res/100,'0.00'));

--откатим изменения

rollback;

end;

Результат:



**Понятие о нагрузочном тестировании**

Основные статьи по теме:

<https://oracle-patches.com/oracle/tuning/технология-real-application-testing?showall=1>

https://translated.turbopages.org/proxy\_u/en-ru.ru.8680a3b8-627c96aa-70a24069-74722d776562/https/docs.oracle.com/cd/E59560\_01/ATSGS/chap4\_olt\_fm.htm