# Лекция: Индексы

## СУБД Oracle предлагает много различных типов индексов.

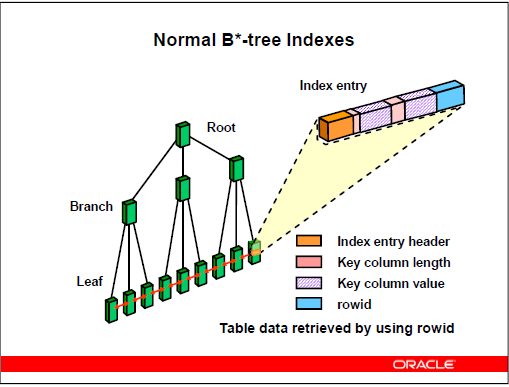
1. Индексы на основе В\*-дерева. Эти индексы называют "обычными". Они, несомненно, чаще всего используются в СУБД Oracle, да и в других СУБД. Аналогичные по конструкции двоичному дереву, они обеспечивают быстрый доступ по ключу к отдельной строке или диапазону строк, требуя обычно очень немного чтений для поиска соответствующей строки. Индекс на основе В\*-дерева имеет несколько подтипов:

* Таблицы, организованные по индексу. Это таблицы, хранящиеся в структуре В\*-дерева.
* Индексы кластера на основе В\*-дерева. Они немного отличаются от обычных, используются для индексации ключей. Они используются не для перехода от ключа к строке, а для перехода от ключа кластера к блоку, содержащему строки, связанные с этим ключом.
* Индексы с обращенным ключом. Это индексы на основе В\*-дерева, байты ключа в которых инвертированы. Это используется для более равномерного распределения записей по индексу при вводе возрастающих значений ключей. Предположим, при использовании последовательности для генерации первичного ключа генерируются значения 987500, 987501, 987502 и т.д. Поскольку это последовательные значения, они будут попадать в один и тот же блок индекса, конкурируя за него. В индексе с обращенным ключом сервер Oracle будет индексировать значения 205789, 105789, 005789. Эти значения обычно будут далеко отстоять друг от друга в индексе, и вставки в индекс будут распределены по нескольким блокам.

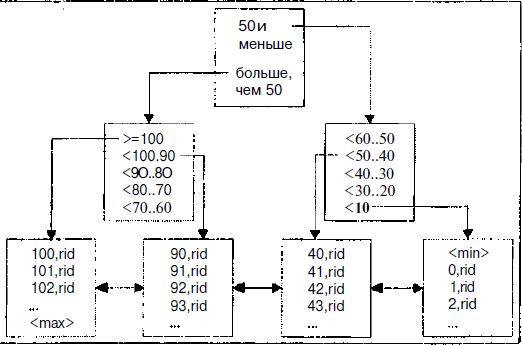
1. Индексы по убыванию. Индексы по убыванию позволяют отсортировать данные в структуре индекса от "больших" к "меньшим" (по убыванию), а не от меньших к большим (по возрастанию).
2. Индексы на основе битовых карт. Обычно в В\*-дереве имеется однозначное соответствие между записью индекса и строкой — запись индекса указывает на строку. В индексе на основе битовых карт запись использует битовую карту для ссылки на большое количество строк одновременно. Такие индексы подходят для данных с небольшим количеством различных значений, которые обычно только читаются. Столбец, имеющий всего три значения — Y, N и NULL, — в таблице с миллионом строк очень хорошо подходит для создания индекса на основе битовых карт. Индексы на основе битовых карт не нужно использовать в базе данных класса OLTP из-за возможных проблем с одновременным доступом.
3. Индексы по функции. Эти индексы на основе В\*-дерева или битовых карт хранят вычисленный результат применения функции к столбцу или столбцам строки, а не сами данные строки. Это можно использовать для ускорения выполнения запросов вида: ***SELECT \* FROM T WHERE ФУНКЦИЯ(СТОЛБЕЦ) = НЕКОТО РОЕ\_ЗНАЧЕНИЕ***, поскольку значение ***ФУНКЦИЯ(СТОЛБЕЦ)*** уже вычислено и хранится в индексе.
4. Прикладные (application domain) индексы. Это индексы, которые строит и хранит приложение, в базе данных Oracle или даже вне базы данных Oracle. Надо сообщить оптимизатору, насколько избирателен индекс, насколько "дорогостояще" его использование, а оптимизатор решает на основе этой информации, использовать этот индекс или нет. Текстовый индекс interMedia — пример прикладного индекса; он построен с помощью тех же средств, которые можно использовать для создания собственных прикладных индексов.
5. Текстовые индексы interMedia. Это встроенные в сервер Oracle специализированные индексы для обеспечения поиска ключевых слов в текстах большого объема

## Индексы на основе В\*-дерева

Индексы на основе В\*-дерева, или "обычные" индексы, — наиболее широко используемый тип индексной структуры в базе данных. По реализации они подобны двоичному дереву поиска. Цель их создания — минимизировать время поиска данных сервером Oracle.



При наличии индекса по числовому столбцу, структура индекса может выглядеть так:



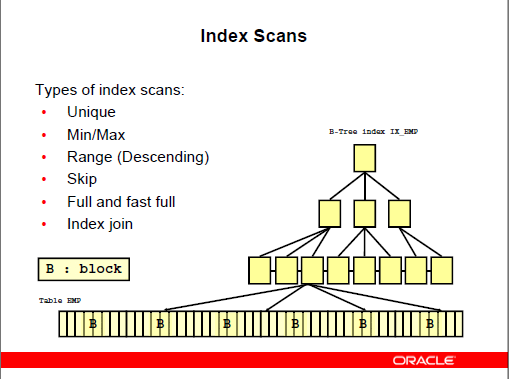
Блоки самого нижнего уровня в индексе, которые называют листовыми вершинами, содержат все проиндексированные ключи и идентификаторы строк (rid на схеме), ссылающиеся на соответствующие строки. Промежуточные блоки над листовыми вершинами называют блоками ветвления. Они используются для переходов по структуре. Например, если необходимо найти в индексе значение 42, надо начать с вершины дерева и двигаться вправо. При проверке этого блока оказывается, что необходимо перейти к блоку в диапазоне "от 40 до 50". Этот блок оказывается листовым и ссылается на строки, содержащие число 42. Интересно отметить, что листовые блоки фактически образуют двухсвязный список. Как только найдено "начало" среди листовых вершин, т.е. первое значение, очень легко просматривать значения по порядку (это называют также просмотром диапазона по индексу, index range scan). Проходить по структуре индекса больше не нужно; мы просто переходим по листовым вершинам. Это существенно упрощает поиск строк но условиям следующего вида:

where x between 20 and 30

Сервер Oracle находит первый блок индекса, содержащий значение 20, а затем проходит по двухсвязному списку листовых вершин, пока не обнаружит значение больше 30. На самом деле такой структуры, как неуникальный индекс на основе В\*-дерева, нет. В неуникальном индексе сервер Oracle просто добавляет идентификаторы строк к ключу индекса, что и делает его неуникальным. В неуникальном индексе данные хранятся отсортированными сначала по значению ключа индекса (в порядке, задаваемом ключом индекса), а потом — по идентификатору строки. В уникальном индексе данные отсортированы только по значению ключа индекса.

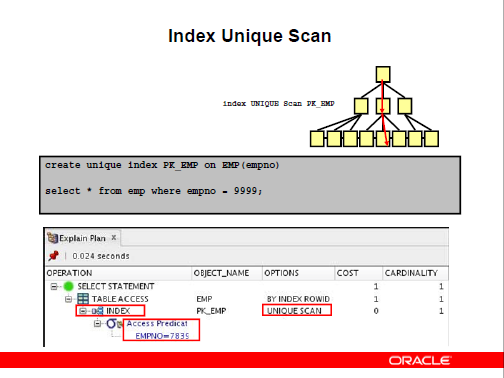
Одно из свойств В\*-дерева состоит в том, что все листовые блоки должны быть на одном уровне, если точнее, разница по высоте между ветвями дерева не может быть больше 1. Уровень листовых блоков называют также высотой дерева. Все вершины выше листовых могут указывать только на содержащие более детальную информацию вершины следующего уровня, а листовые вершины указывают на конкретные идентификаторы строк или диапазоны идентификаторов строк. Большинство индексов на основе В\*-дерева будут иметь высоту 2 или 3, даже для миллионов записей. Это означает, что для поиска ключа в индексе потребуется 2 или 3 чтения, что неплохо. Еще одно свойство — автоматическая балансировка листовых вершин: они почти всегда располагаются на одном уровне. Есть причины, по которым индекс может оказаться не идеально сбалансированным при изменении и удалении записей. Сервер Oracle будет пытаться заполнять блоки индекса не более чем на три четверти, но и это свойство может временно нарушаться при выполнении операторов DELETE и UPDATE. В общем случае В\*-дерево — отличный универсальный механизм индексирования, хорошо работающий как в случае больших, так и маленьких таблиц, и лишь немного хуже работающий при росте базовой таблицы, если только дерево остается сбалансированным. Интересно, что индексы на основе В\*-дерева можно "сжимать". Это не такое сжатие, как в zip-файлах, при сжатии удаляется избыточность в составных индексах по аналогии сжатия данных в таблицах организованных по индексу. В основе сжатия ключа индекса лежит разбиение записи на две части: префикс и суффикс. Префикс строится по начальным столбцам составного индекса, и его значения часто повторяются. Суффикс — это завершающие столбцы ключа индекса, и эта часть в записях индекса с одинаковым префиксом — уникальна.

Индекс может быть просмотрен по следующим схемам:



### Уникальное сканирование

Такой индекс возвращает самое точно одно значение ROWID. Система выполняет уникальное сканирование, если в операторе выбора содержится выражение, которое гарантирует, что будет обрабатываться только одна строка UNIQUE PRIMARY KEY. Этот путь доступа используется, когда все столбцы уникального (B \* -дерева) индекса заданы с условиями равенства. Значения ключей и ROWID получены из индекса, а строки таблицы извлекаются с использованием ROWID.



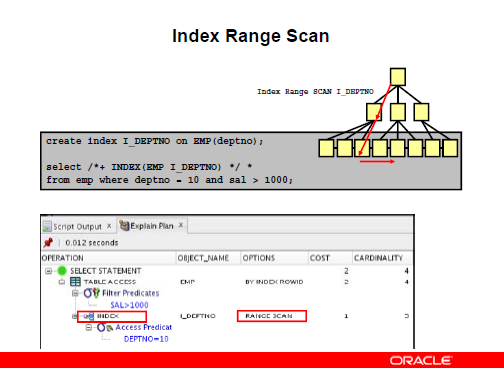
Примечание. Условия фильтрации фильтруют строки после операции выборки и выводят отфильтрованные строки.

### Сканирование диапазона

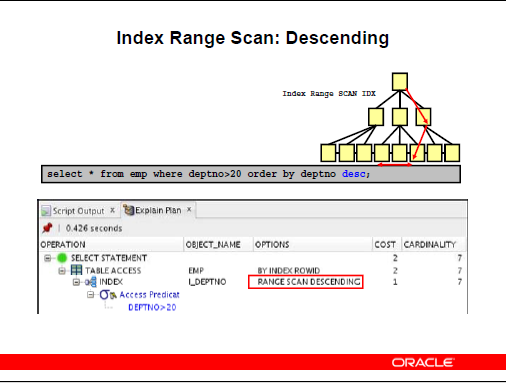
Сканирование диапазона в индексе является распространенной операцией для доступа к выборочным данным. Он может быть ограниченным (с обеих сторон) или неограниченным (с одной или обеих сторон). Данные возвращаются в порядке возрастания столбцов индекса. Несколько строк с одинаковыми значениями ключа сортируются в порядке возрастания по ROWID.

Оптимизатор использует сканирование диапазона, когда находит один или несколько ведущих столбцов индекса, указанного в условиях, таких как: b1= col1, b1 <col1или b1> col1, или любая комбинация этих условий.

На пример поиск (col1 like '%SD ') не приводит к сканированию диапазона. При сканировании диапазона могут использоваться уникальные или неуникальные индексы. Сканирования диапазона не применяется, если столбцы индекса входят в предложения ORDER BY / GROUP BY, и индексированные столбцы NOT NULL.



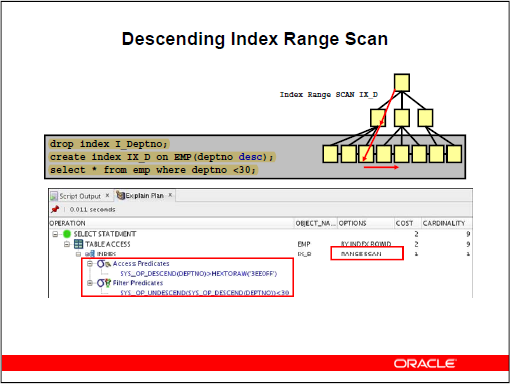
Сканирование диапазона индекса по убыванию идентично диапазону индекса, за исключением того, что данные возвращаются в порядке убывания. Оптимизатор использует сканирование диапазона индекса по убыванию, когда порядок поиска может быть выполнен эффективно по убыванию, т. е. когда правый диапазон неограничен.



### Сканирование диапазона по убыванию индекса

Сканирование диапазона индекса по убыванию идентично сканированию диапазона индекса, за исключением того, что данные возвращаются в порядке убывания. Нисходящие индексы позволяют сортировать данные от «большого к маленькому» (по убыванию) вместо «от маленького к большому» (по возрастанию) в структуре индекса. Обычно это сканирование используется при упорядочении данных в порядке убывания, чтобы сначала получить самые последние данные, или при поиске значения, которое меньше указанного значения, как в примере на слайде.

Оптимизатор использует сканирование диапазона по убыванию индексов, когда порядок по убыванию может быть удовлетворен по убыванию индекса. Подсказка может быть использована для форсирования этого пути доступа INDEX\_DESC (table\_alias index\_name), если это возможно.

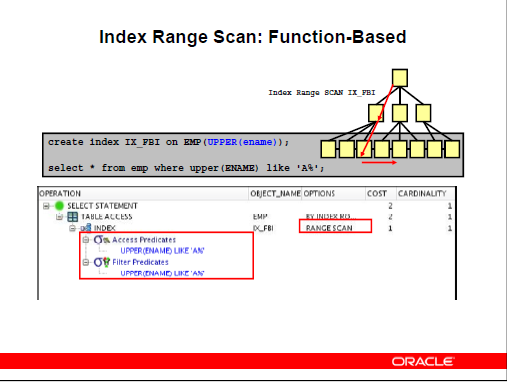


Примечание. Система обрабатывает нисходящие индексы как индексы на основе функций. Отмеченные столбцы сохраняются в специальном порядке убывания в структуре индекса, которая снова переворачивается DESC с помощью функции. SYS\_OP\_UNDESCEND

### Сканирование диапазона индексов: на основе функций

Индекс на основе функций может быть сохранен как B \* -дерево или растровая структура. Эти индексы включают столбцы, которые преобразуются функцией. С помощью индекса на основе функций вы можете хранить в индексе выражения с использованием вычислений типа col1 + col2. Определение индекса на основе функции для столбца или выражения позволяет возвращать данные с использованием индекса, когда соответствующая функция или выражение используется в предложении WHERE. Это позволяет системе обходить вычисление значения выражения ORDER BY при обработке и утверждениях. Следовательно, индекс на основе функций может быть полезным, если часто выполняемые операторы SQL — SELECT, DELETE включают преобразованные столбцы или столбцы в выражениях в выражении WHERE или ORDER BY.

Например, индексы на основе функций, определенные с помощью ключевых слов или UPPER (column\_name), позволяют выполнять поиск без учета регистра, например, показанный на слайде LOWER (column\_name).



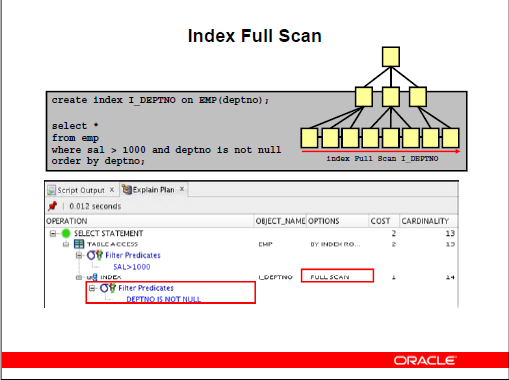
### Индекс полного сканирования

Полная проверка доступна, если предикат ссылается на один из столбцов в индексе. Предикат не обязательно должен быть драйвером индекса (начальный столбец). Полное сканирование также доступно при отсутствии предиката, если выполняются условия:

• Все столбцы в таблице, на которые есть ссылка в запросе, включены в индекс.

• По крайней мере, один из столбцов индекса не является нулевым.

Полное сканирование можно использовать для исключения операции сортировки, поскольку данные упорядочены по ключу индекса.



Примечание. Полное сканирование индекса считывает индекс, используя одноблочный ввод / вывод (в отличие от быстрого полного сканирования индекса).

### Полное быстрое сканирование

Полное быстрое сканирование индекса является альтернативой полному сканированию таблицы, когда индекс содержит все столбцы, необходимые для запроса, и хотя бы один столбец в ключе индекса имеет ограничение NOT NULL. Быстрое полное сканирование обращается к данным в самом индексе, не обращаясь к таблице.

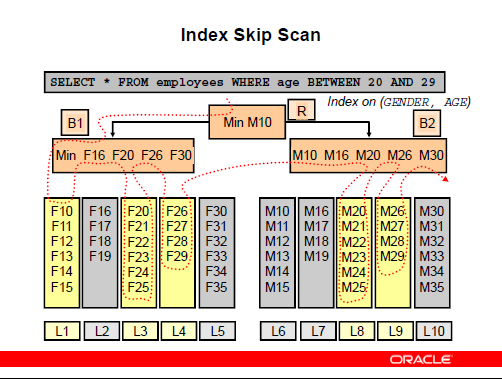
Его нельзя использовать для исключения операции сортировки, поскольку данные не упорядочены по ключу индекса. Может использоваться для функций min / avg / aggregate. В этом случае оптимизатор должен в сумме знать, что все строки таблицы представлены в индексе.

Эта операция читает весь индекс, используя многоблочные чтения (в отличие от полного сканирования индекса). Быстрое полное сканирование индекса не может быть выполнено для растровых индексов. Быстрое полное сканирование быстрее, чем обычное полное индексное сканирование, потому что оно может использовать многоблочный ввод-вывод просто как сканирование таблицы. Вы можете указать быстрое полное сканирование индекса с помощью параметра инициализации OPTIMIZER\_FEATURES\_ENABLE или подсказки, как показано в примере слайда. INDEX\_FFS

Примечание. Индекс быстрого полного сканирования используется для индекса, когда он перестраивается в автономном режиме.

### Сканирование с пропуском

Сканирование с пропуском индекса улучшает сканирование индекса, пропуская блоки, которые не содержат ключи, соответствующие значениям столбца фильтра. Сканирование индексных блоков часто происходит быстрее, чем сканирование блоков данных таблицы. Пропуск сканирования может происходить, если начальный столбец составного индекса не указан в запросе. Предположим, что есть объединенный индекс столбца GENDER и столбца ВОЗРАСТ в таблице. Этот пример иллюстрирует, как пропускается сканирование ВОЗРАСТНЫХ СОТРУДНИКОВ(AGE EMPLOYEES), чтобы ответить на запрос.

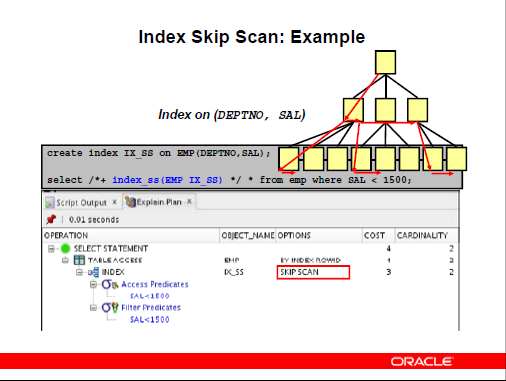


Система начинается с корня индекса [R] и переходит к левому ответвлению [B1]. Оттуда система идентифицирует первую запись, которая будет F16, переходит на левый лист [L1] и начинает сканировать его, потому что он может содержать A25 (то есть, где «пол» находится перед «F» в алфавите) ,

Сервер определяет, что это невозможно, потому что первая запись - F10. Таким образом, невозможно найти запись типа A25 в этом листе, поэтому ее можно пропустить. Возвращаясь к первому переходному блоку [B1], сервер определяет, что следующее поддерево (F16) не нужно сканировать, потому что следующая запись в [B1] - это F20. Поскольку сервер уверен, что невозможно найти 25 между F16 и F20, второй листовой блок [L2] можно пропустить.

Возвращаясь к [B1], сервер обнаруживает, что следующие две записи имеют общий префикс F2. Это определяет возможные поддеревья для сканирования. Система знает, что эти поддеревья упорядочены по возрасту.

Таким образом, третий и четвертый листовые блоки [L3 – L4] сканируются и некоторые значения извлекаются. Посмотрев на четвертую запись в первом узловом блоке [B1], система определяет, что больше невозможно найти запись F2x. Таким образом, нет необходимости сканировать это поддерево [L5]. Тот же процесс продолжается с правой частью этого индекса. Обратите внимание, что из 10 листовых блоков сканируются только пять.



В примере показан поиск сотрудников с зарплатой менее 1500 используя сканирование с пропуском индекса.

Предполагается, что есть объединенный индекс в столбцах DEPTNO и SAL.

В запросе для поиска не используется ведущий столбец. Этот ведущий столбец DEPTNO имеет только некоторые дискретные значения, то есть 10, 20 и 30.

Пропуск сканирования позволяет логически разбить составной индекс на меньшие подиндексы. Количество логических подиндексов определяется количеством различных значений в исходном столбце.

Система делает вид, что индекс - это на самом деле три маленькие структуры индекса, спрятанные внутри одной большой. В примере это три структуры индекса:

• где DEPTNO = 10

• где DEPTNO = 20

• где DEPTNO = 30

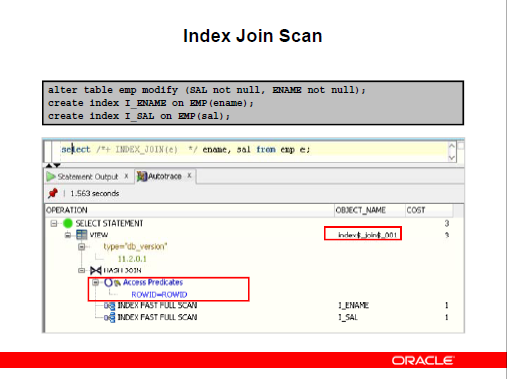
Выход упорядочен по DEPTNO.

Примечание. Пропускное сканирование выгодно, если в начальном столбце составного индекса имеется несколько отдельных значений и много значений в конечном ключевом поле индекса.

### Индекс Присоединение

Индексное соединение - это хеш-соединение нескольких индексов, которые вместе содержат все столбцы таблицы, на которые есть ссылки в запросе. Если используется соединение индекса, доступ к таблице не требуется, поскольку все соответствующие значения столбца могут быть получены из индексов. Индексное соединение не может использоваться для устранения операции сортировки.

Соединение с индексом не является реальной операцией соединения (обратите внимание, что пример является запросом к одной таблице), но оно строится с использованием доступа к индексу, за которым следует операция соединения с rowid. В примере на слайде предполагается, что у вас есть два отдельных индекса для столбцов ENAME и SAL в EMP.



Примечание. Вы можете указать соединение индекса с помощью подсказки INDEX\_JOIN, как показано в примере.

## Индексы с обращенным ключом

Еще одна особенность индексов на основе В\*-дерева — возможность "обратить" ключи. Сначала может показаться странным, зачем это вообще нужно? Они созданы для специфической среды и с конкретной целью. Они предназначены для уменьшения количества конфликтов при доступе к листовым блокам индекса в среде Oracle Parallel Server.

В индексе с обращенным ключом просто обращается порядок байтов в каждом столбце ключа индекса. Если взять числа 90101, 90102, 90103 и посмотреть на их внутреннее представление с помощью встроенной функции DUMP, окажется, что они представлены следующим образом:

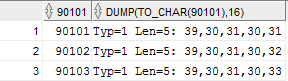
select 90101, dump(to\_char(90101),16) from dual

union all

select 90102, dump(to\_char(90102),16) from dual

union all

select 90103, dump(to\_char(90103),16) from dual



Каждое число представлено четырьмя байтами, и отличаются они только последним байтом. В структуре индекса эти числа окажутся рядом. Если же обратить порядок следования байтов, сервер Oracle вставит следующие значения:

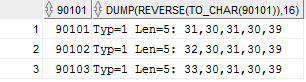
select 90101, dump(reverse(to\_char(90101)),16) from dual

union all

select 90102, dump(reverse(to\_char(90102)),16) from dual

union all

select 90103, dump(reverse(to\_char(90103)),16) from dual;



Эти числа окажутся "далеко" друг от друга. При этом сокращается количество экземпляров, обращающихся к одному и тому же блоку (крайнему слева) и, следовательно, количество выполняемых тестовых опросов. *Один из недостатков индекса* с обращенными ключами — то, что его нельзя использовать в некоторых случаях, когда обычный индекс вполне применим. Например, при поиске по следующему критерию индекс с обращенным ключом по столбцу Х не поможет:

where x > 5

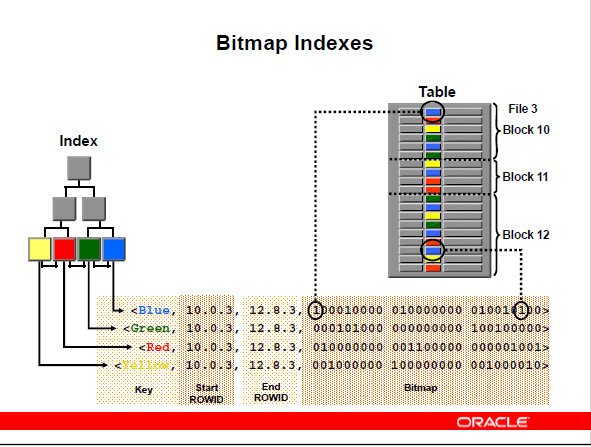
Данные в индексе не отсортированы, поэтому просмотреть диапазон нельзя. С другой стороны, некоторые просмотры диапазонов в индексе с обращенным ключом вполне выполнимы. Если имеется составной индекс по столбцам X, Y, при поиске по следующему условию можно будет использовать индекс с обращенным ключом и "просматривать диапазон" в нем:

where x = 5

Дело в том, что байты в столбце X обращены, и байты в столбце Y тоже обращены. Сервер Oracle не обращает байты значения X || Y, а сохраняет в записи индекса результат выполнения reverse(X) || reverse(Y). Это означает, что все значения X = 5 будут храниться вместе, так что сервер Oracle может просматривать последовательно листовые блоки индекса для поиска всех таких строк.

## Индексы на основе битовых карт

Индексы на основе битовых карт появились в версии 7.3 сервера Oracle. Индексы на основе битовых карт создавались для хранилищ данных или сред с произвольными запросами, где полный список возможных запросов к данным при реализации приложения не полностью известен. Они не походят для систем ОLTP или систем, где данные часто изменяются несколькими одновременно работающими сеансами.



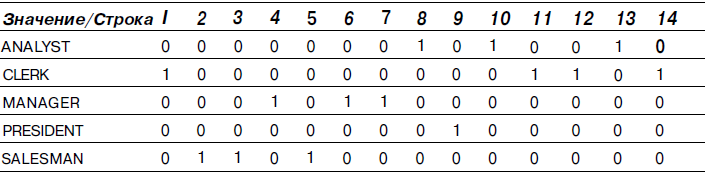
Индексы на основе битовых карт — это структуры, в которых хранятся указатели на множество строк, соответствующих одному значению ключа индекса, тогда как в структуре В\*-дерева количество ключей индекса обычно примерно соответствует количеству строк. В индексе на основе битовых карт записей очень мало, и каждая из них указывает на множество строк. В индексе на основе В\*-дерева обычно имеется однозначное соответствие — запись индекса ссылается на одну строку.

Предположим, создается индекс на основе битовых карт по столбцу JOB в таблице

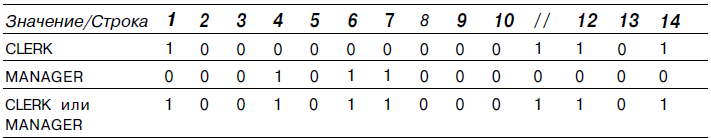
ЕМР:

create BITMAP index job\_idx on emp(job);

Сервер Oracle будет хранить в индексе примерно следующее:



Это показывает, что в строках 8, 10 и 13 находится значение ANALYST, тогда как в строках 4, 6 и 7 — значение MANAGER. Также понятно, что пустых строк нет (индексы на основе битовых карт содержат записи для пустых значений — отсутствие такой записи в индексе означает, что пустых строк нет). Если необходимо посчитать, в скольких строках хранится значение MANAGER, индекс на основе битовых карт позволит сделать это очень быстро. Если необходимо найти все строки, в которых в столбце JOB хранится значение CLERK или MANAGER, достаточно просто скомбинировать соответствующие битовые карты из индекса:



Это позволяет быстро понять, что критериям поиска удовлетворяют строки 1, 4, 6, 7, 11, 12 и 14. Битовая карта, которую сервер Oracle хранит для каждого значения ключа, устроена так, что каждая позиция представляет идентификатор строки базовой таблицы на случай, если понадобится выбрать для дальнейшей обработки соответствующую строку. На запросы вида:

Select count(\*) from emp where job = 'CLERK' or job = 'MANAGER'

можно ответить непосредственно по индексу на основе битовых карт. Для ответа на запрос вида:

Select \* from emp where job = 'CLERK' or job = 'MANAGER'

придется обратиться к таблице. Сервер Oracle применит функцию, преобразующую установленный бит i в битовой карте в идентификатор строки, по которому можно обратиться к таблице.

Индексы на основе битовых карт больше подходят для данных с небольшим количеством уникальных значений. Это данные, для которых при делении количества уникальных значений в строках на общее количество строк получается небольшое число (близкое к нулю). Например, столбец GENDER (пол) может иметь значения М, F и NULL.

При наличии таблицы с 20000 записей о сотрудниках, получаем 3/20000 = 0,00015. Этот столбец отлично подходит для создания индекса на основе битовых карт. Он, определенно, не подходит для создания индекса на основе В\*-дерева, поскольку для каждого значения ключа будет извлекаться существенная часть строк таблицы. В общем случае, как было показано выше, индексы на основе В\*-дерева должны быть избирательными.

Индексы на основе битовых карт не должны быть избирательными, наоборот, они должны быть очень "неуникальными".

Индексы на основе битовых карт особенно хорошо подходят для сред с множеством произвольных запросов, особенно, если запросы эти ссылаются произвольным образом на много столбцов или выбирают агрегированные значения типа COUNT. Предположим, имеется таблица с тремя столбцами: GENDER, LOCATION и AGE\_GROUP. В этой таблице столбец GENDER имеет значение M или F, столбец LOCATION может иметь значения от 1 до 50, а в столбце AGE\_GROUP находится код, представляющий возрастные группы не старше 18 лет, 19-25 лет, 26-30 лет, 31-40 лет, 41 год и старше. Необходимо обеспечить выполнение множества произвольных запросов вида:

Select count(\*) from T where gender = 'M' and location in (1, 10, 30) and age\_group = '41 год и старше';

Select \* from t where ((gender = 'M' and location = 20) or (gender = 'F' and location = 22)) and age\_group = 'не старше 18 лет';

select count(\*) from t where location in (11,20,30);

select count(\*) from t where age\_group = '41 год и старше' and gender = 'F';

Оказывается, обычная схема индексирования на основе В\*-дерева тут не поможет. Если хочется использовать для получения ответа индексы, придется использовать от трех до шести возможных комбинаций индексов на основе В\*-дерева для обеспечения доступа к данным. Поскольку в запросах может появиться любой из трех столбцов и любое подмножество трех столбцов, придется создать большие составные индексы на основе В\*-дерева по следующим столбцам.

• GENDER, LOCATION, AGE\_GROUP. Для запросов, использующих все три столбца, столбцы GENDER и LOCATION или только столбец GENDER.

• LOCATION, AGE\_GROUP. Для запросов, использующих столбцы LOCATION и AGE\_GROUP или только столбец LOCATION.

• AGE\_GROUP, GENDER. Для запросов, использующих столбцы AGE\_GROUP и GENDER или только столбец AGE\_GROUP.

Чтобы сократить объем просматриваемых данных, имеет смысл проиндексировать и другие перестановки — это позволит сократить размер просматриваемых индексных структур. Не говоря уже о том, что создание индекса на основе В\*-дерева по данным с таким небольшим количеством различных значений — вообще не лучшая идея.

Вот тут и пригодится индекс на основе битовых карт. С помощью трех небольших индексов на основе битовых карт, по одному для каждого отдельного столбца, можно эффективно находить строки, удовлетворяющие всем представленным выше условиям.

Сервер Oracle будет просто объединять битовые карты трех индексов с помощью функций AND, OR и XOR, чтобы найти результирующее множество для условия, ссылающегося на любое подмножество этих трех столбцов. Он возьмет затем полученную в результате битовую карту, при необходимости преобразует биты со значением 1 в соответствующие идентификаторы строк и получит соответствующие данные (если бы требовалось выдать количество строк, удовлетворяющих условию, серверу достаточно было бы посчитать биты со значением 1).