Доступ к данным в БД

Основные концепции.
Особенности СУБД Oracle

Способы доступа к данным в БД

Основные способы доступа к данным:

- **Последовательная обработка области БД**. Областью БД может быть файл или другое множество страниц (блоков) памяти. Последовательная обработка предполагает, что система последовательно просматривает страницы, пропускает пустые участки и выдаёт записи в физической последовательности их хранения.
- **Доступ по ключу базы данных** (КБД). КБД определяет местоположение записи в памяти ЭВМ. Зная его, система может извлечь нужную запись за одно обращение к памяти.
- **Доступ по ключу (в частности, первичному).** Если система обеспечивает доступ по ключу, то этот ключ также может использоваться при запоминании записи (для определения места размещения записи в памяти). В базах данных применяются такие способы доступа по ключу, как индексирование, хеширование и кластеризация.
- **Доступ по структуре:** используется в иерархических, сетевых и объектно-реляционных БД.

Индексирование данных

Определим индексирование как способ доступа к данным в реляционной таблице с помощью специальной структуры – индекса.

- **Индекс** – это структура, которая определяет соответствие значения ключа записи (атрибута или группы атрибутов) и местоположения этой записи – КБД. Каждый индекс связан с определённой таблицей, но является внешним по отношению к таблице и обычно хранится отдельно от неё.

Индекс		Пространство памяти		
Значение атрибута	*КБД*	F6:00	Волкова	…
Белова	FA:00	F6:1E	Волков	…
Волков	F6:1E	F6:31	Поспелов	…
Волкова	F6:00		…	
Осипов	FA:2B	FA:00	Белова	…
Поспелов	F6:31	FA:1D	Фридман	…
Фридман	FA:1D	FA:2B	Осипов	…

Индексирование данных

Особенности организации индексов:

1. Индекс обычно хранится в отдельном файле или отдельной области памяти.
2. Пустые значения атрибутов (NULL) не индексируются.
3. Индексирование используется для ускорения доступа к записям по значению ключа и не влияет на размещение данных этой таблицы.
   Ускорение поиска данных через индекс обеспечивается за счёт:
   - упорядочивания значений индексируемого атрибута. Это позволяет просматривать в среднем половину индекса при линейном поиске;
   - индекс занимает меньше страниц памяти, чем сама таблица, поэтому система тратит меньше времени на чтение индекса, чем на чтение таблицы.
4. Индексы поддерживаются динамически.
5. Каждый индекс относится к одной таблице, на одну таблицу можно создать несколько индексов.

Индексирование данных

Индексы бывают:

- ✓ **Первичные** (уникальные) и **вторичные** (неуникальные). Большинство СУБД автоматически строят индекс по первичному ключу и по уникальным столбцам.
- ✓ **Плотные и неплотные.** В плотных для каждого значения ключа имеется отдельная запись индекса, указывающая место размещения конкретной записи. **Неплотные** (разреженные) индексы строятся в предположении, что на каждой странице памяти хранятся записи, отсортированные по значениям индексируемого атрибута. Тогда для каждой страницы в индексе задаётся диапазон значений ключей хранимых в ней записей, и поиск записи осуществляется среди записей на указанной странице.
- ✓ **Сжатые** и **несжатые**.
- ✓ **Одиночные** и **составные**.
- ✓ **Линейные** и **многоуровневые**.

Индексирование данных: составные индексы

Таблица

ID	EDATE	CODE	FIRM	PRICE
100	01.12.95	A4	Комус	312.0
200	01.12.95	A4	Партия	321.5
100	02.12.95	A2	ОАО "Заря"	110.6
110	10.12.95	A4	Фирма "Б+"	314.0
200	01.12.95	A2	Партия	114.0
200	02.12.95	A1	Amos ltd.	52.8

Индекс

ID	EDATE	CODE
100	01.12.95	A4
100	02.12.95	A2
110	10.12.95	A4
200	01.12.95	A2
200	01.12.95	A4
200	02.12.95	A1

Составной индекс включает два или более столбца одной таблицы. Последовательность вхождения столбцов в индекс определяется при его создании.

Многоуровневые индексы: B-дерево

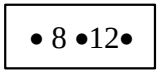
B-дерево строится динамически по мере заполнения базы данными. Оно растёт вверх, и корневая вершина может меняться. Параметрами B-дерева являются порядок ***n*** и количество уровней. Порядок – это количество ссылок из вершины i-го уровня на вершины (i+1)-го уровня.

Каждое B-дерево должно удовлетворять следующим условиям:

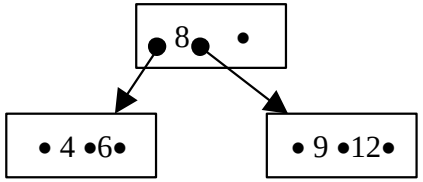
- Все конечные вершины расположены на одном уровне, т.е. длина пути от корня к любой конечной вершине одинакова.
- Каждая вершина может содержать *n* адресных ссылок и (*n-1*) ключей. Ссылка влево от ключа обеспечивает переход к вершине дерева с меньшими значениями ключей, а вправо – к вершине с б***о***льшими значениями.
- Любая неконечная вершина имеет не менее *n/2* подчинённых вершин. (Для деревьев нечётного порядка значение n/2 округляется в б***о***льшую сторону).
- Если неконечная вершина содержит *k* (*k*<*n*) ключей, то ей подчинена (*k+1*) вершина на следующем уровне иерархии.

Многоуровневые индексы: B-дерево

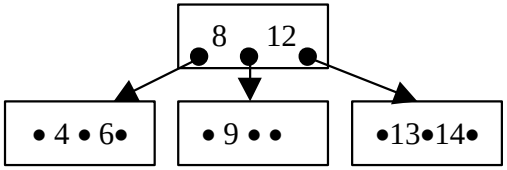
Шаги 1,2



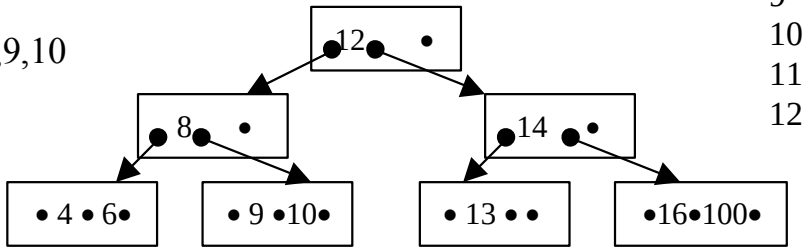
Шаги 3,4,5



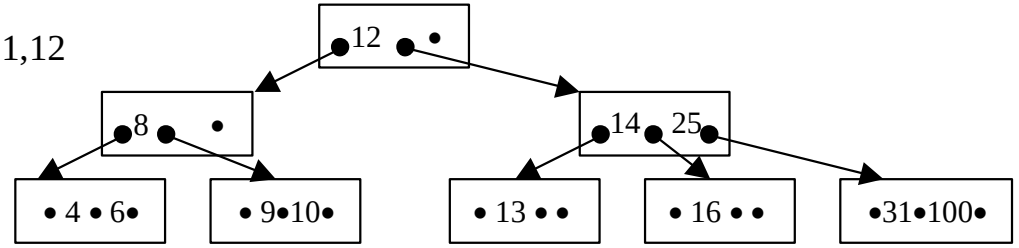
Шаги 6,7



Шаги 8,9,10



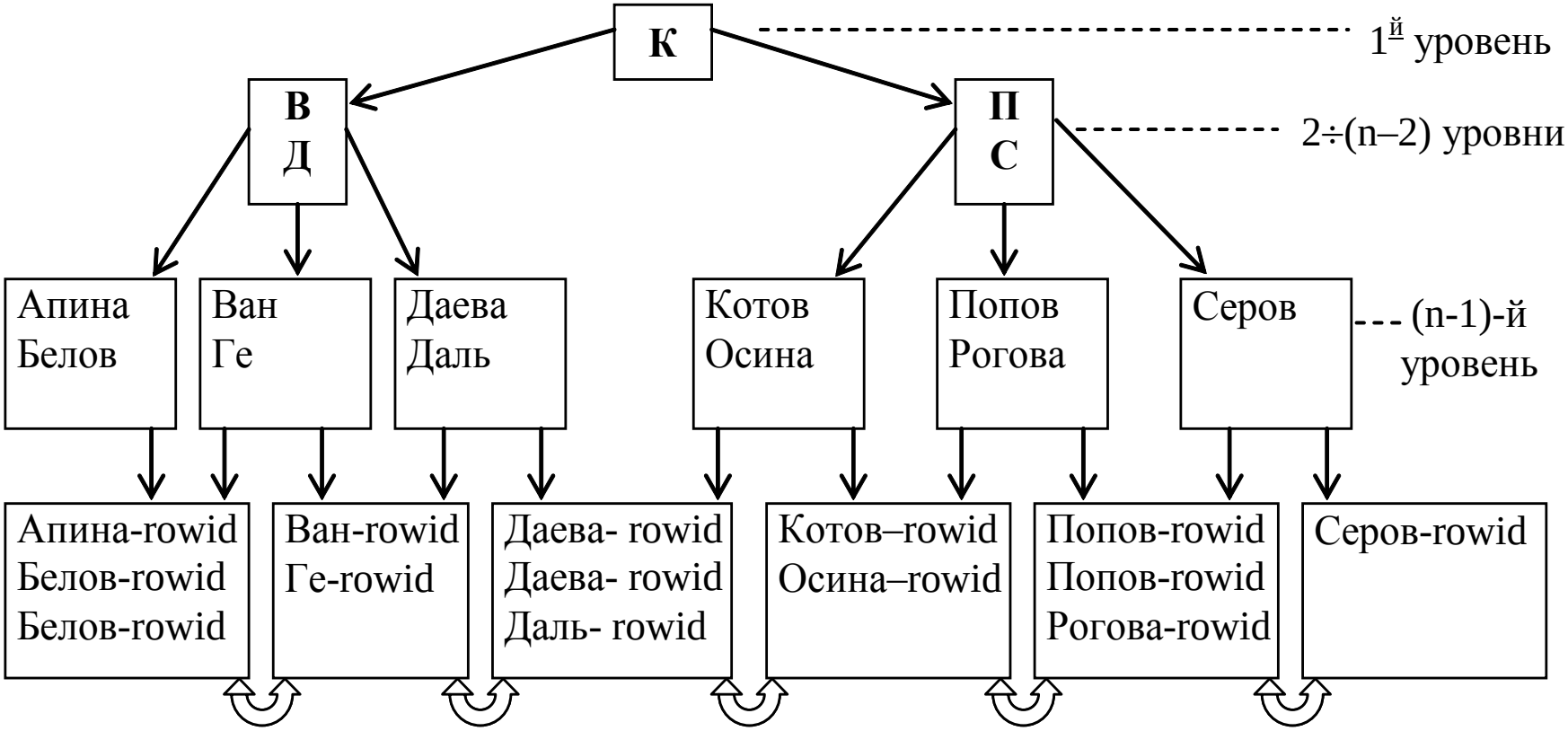
Шаги 11,12



Записи поступают в таком порядке:

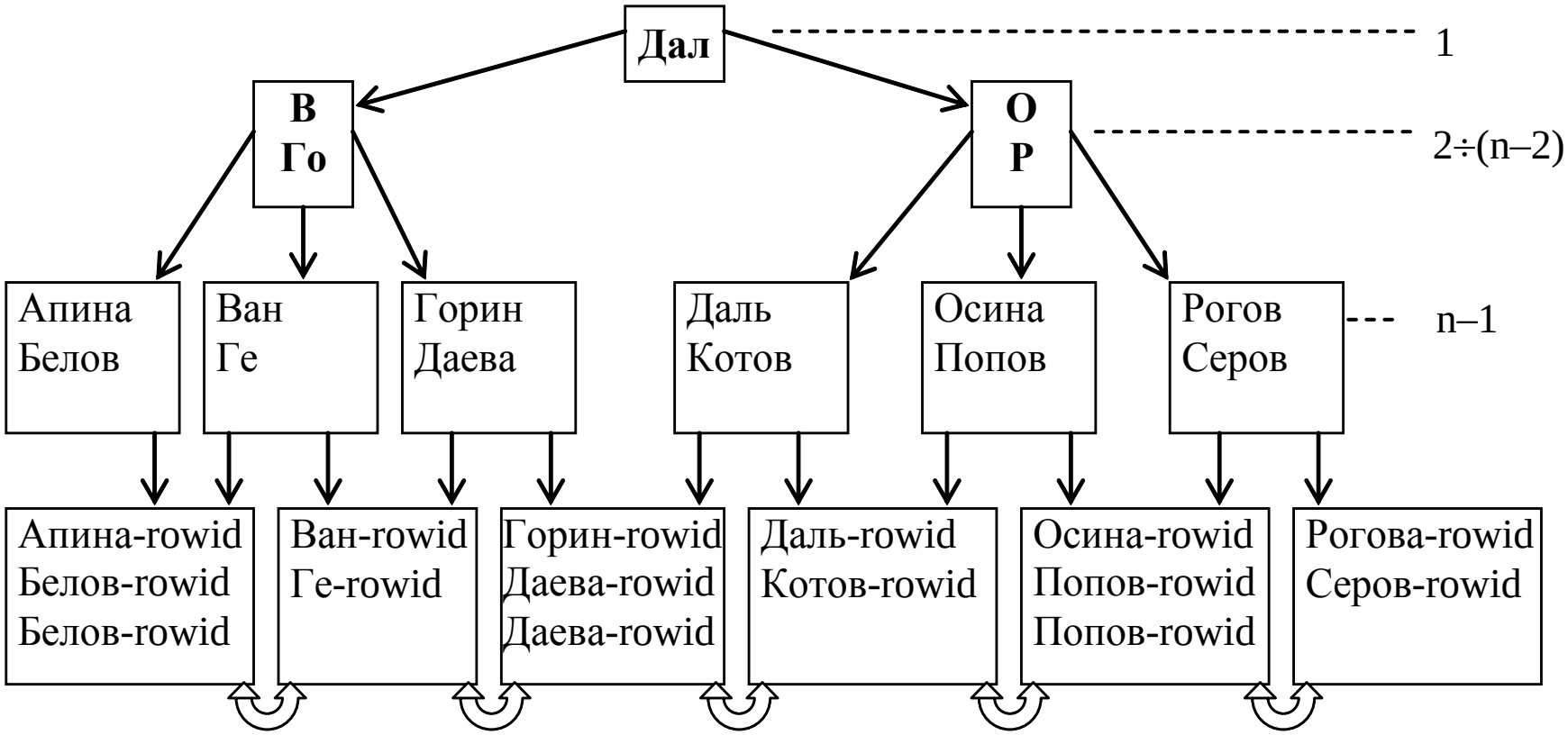
шаг	ключ
1	12
2	8
3	4
4	9
5	6
6	13
7	14
8	16
9	100
10	10
11	25
12	31

Многоуровневые индексы: Oracle



Многоуровневые индексы: Oracle

Перераспределение данных:



Многоуровневые индексы

Структура B-дерева (balance tree) имеет следующие преимущества:

- B-дерево автоматически поддерживается в сбалансированном виде.
- Все блоки-листья в дереве расположены на одном уровне, следовательно, поиск любой записи в индексе занимает примерно одно и то же время.
- B-деревья обеспечивают хорошую производительность для широкого спектра запросов, включая поиск по конкретному значению и поиск в открытом и закрытом интервалах (благодаря ссылкам между блоками-листьями).
- Модификация данных таблицы выполняется достаточно эффективно, т.к. в блоках индекса обычно есть свободное место для размещения новых значений, а полная перестройка дерева выполняется достаточно редко.
- Производительность B-дерева одинаково хороша для маленьких и больших таблиц, и не меняется существенно при росте таблицы.

Использование индексов

Команды управления индексами не входят в стандарт SQL, но с небольшими отличиями поддерживается практически всеми РСУБД.
Общий синтаксис команды **create index** следующий:

```
create [<тип>] index <имя_индекса>
    on <имя_таблицы>(<поле1> [, <поле2>,...])
    [<параметры>];
```

Имя индекса должно быть уникальным среди имён объектов БД.
Если индекс составной, то входящие в него поля перечисляются через запятую. Необязательные <тип> (например, UNIQUE, BITMAP, HASH или FULLTEXT) и <параметры> зависят от используемой СУБД.
Например, в Oracle с помощью следующей команды можно создать составной индекс для таблицы *СОТРУДНИКИ* (EMP) по полям *Фамилия* (fam) и *Имя* (name):

```
create index ind_emp_name on emp (fam, name)
tablespace my_indexes;
```

А в MySQL можно создать индекс по первым символам текстового поля:

```
create index part_of_name ON customer (name(10));
```

Индексы и таблицы желательно создавать на разных дисках.

Использование индексов

Выбор столбцов для индекса определяется следующими соображениями:

- В первую очередь выбираются столбцы, которые часто встречаются в условиях поиска.
- Стоит индексировать столбцы, которые используются для соединения таблиц или являются внешними ключами. В последнем случае наличие индекса позволяет обновлять строки подчинённой таблицы без блокировки основной таблицы, когда происходит интенсивное конкурентное обновление связанных между собою таблиц.
- Нецелесообразно индексировать столбцы с низкой селективностью. Исключения для низкой селективности составляют случаи, при которых выборка чаще производится по редко встречающимся значениям.
- Не индексируются столбцы, которые часто обновляются, т.к. команды обновления ведут к потере времени на обновление индекса.
- Не индексируются столбцы, которые часто используются как аргументы выражений или функций: как правило, это не позволяет использовать индекс.

Использование индексов

В некоторых случаях использование составного индекса предпочтительнее, чем одиночного, а именно:

- Если в запросах часто используются только столбцы, участвующие в индексе, система может вообще не обращаться к таблице для поиска данных.
- Несколько столбцов с низкой селективностью в комбинации друг с другом могут дать гораздо более высокую селективность.

Обращение к составному индексу возможно только в том случае, если в условиях выбора участвуют столбцы, представляющие собой лидирующую часть составного индекса. Если индекс, например, включает поля (X, Y, Z), то обращение к индексу будет происходить в тех случаях, когда в условии запроса участвуют поля XYZ, XY или X, причём именно в таком порядке.

Использование индексов

Вопрос. Будет ли система пользоваться индексом при выполнении следующих запросов:

1) SELECT * FROM emp;
2) SELECT * FROM emp
   WHERE name = 'Даль';
3) SELECT * FROM emp
   WHERE sex = 'ж';
4) SELECT depno, count(*)
   FROM emp
   GROUP BY depno;
5) SELECT * FROM emp e, child c
   WHERE e.tabno=c.tabno;

Необходимое условие использования индекса: в запросе есть условие на значение индексируемого поля (или СУБД может его вывести).
Достаточное условие использования индекса: запрос по индексу выполняется быстрее, чем без индекса (повышение эффективности).

Использование индексов в Oracle

Рекомендации по созданию эффективных индексов в СУБД Oracle:

- Индекс имеет смысл, если нужно обеспечить доступ одновременно не более чем к 4-5% 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 колюча Oracle автоматически создаст индекс по этому столбцу.
- Индексируйте столбцы, участвующие в многотабличных операциях соединения.
- Индексируйте столбцы, которые часто используются в конструкциях WHERE.

Использование индексов в Oracle

Рекомендации по созданию эффективных индексов в СУБД Oracle:

- 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, стоящие 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 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 в котором количество строк с одинаковым значением минимально.

Битовые индексы Oracle

Битовые индексы (BITMAP) используют битовые карты для указания значения индексированного столбца. Это идеальный индекс для столбца с низкой кардинальностью (число уникальных записей в таблице мало) при большом размере таблицы.

Пример битового индекса для поля "День недели"

Адрес (КБД)	пн	вт	ср	чт	пт	сб	вс
A0:03	0	0	1	0	0	0	0
A0:04	0	0	0	0	1	0	0
A0:05	0	0	1	0	0	0	0
A0:06	0	0	1	0	0	0	0
A0:07	0	0	0	0	0	0	1
A0:08	1	0	0	0	0	0	0
…							
FC:10	0	0	0	0	0	1	0

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.

Битовые индексы Oracle

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 OLAP
Занимают много места	Используют относительно мало места
Легко обновляются	Трудно обновляются

Для создания битового индекса используется оператор^

```
CREATE BITMAP INDEX day_ind ON tab(day)
TABLESPACE MY_INDEXES;
```

Индексы Oracle 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 реверсивным 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 базы 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 Oracle 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:

```
CREATE  INDEX  emp_indx1  ON  employees(ename)
        TABLESPACE  MY_INDEXES
        COMPRESS  1;
```

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

Индексы Oracle на основе функций

Индексы на основе функций предварительно вычисляют значения функций по заданному столбцы 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  emp(LOWER(name));
```

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

Невидимые индексы Oracle

Невидимый индекс оптимизатором не обнаруживается и не принимается 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». 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_index 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:

1) CRETE INDEX имя_индекса имя_таблицы(поля) INVISIBLE;
2) ALTER INDEX имя_индекса INVISIBLE / VISIBLE;

Обслуживание индексов Oracle

Данные индекса постоянно изменяются из-за DML-действий, связанных с его таблицей. Индексы часто становятся слишком большими, если происходит много удалений строк: пространство, занятое удаленными значениями, автоматически повторно индексом не используется.

Команда REBUILD позволяет реорганизовать индексы и сделать их более компактными, а потому и более эффективными, а также изменить параметры хранения, которые устанавливаются во время начального создания индекса.

```
ALTER INDEX sales_idx REBUILD;
```

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.

Команду REBUILD можно использовать в оперативном (online) режиме. Во время оперативной перестройки индекса разрешено применение всех операций DML, но не операций DDL.

```
ALTER INDEX sales_idx REBUILD ONLINE [NOLOGGING];
```

NOLOGGING: база данных не будет генерировать данные повторного выполнения для операции перестройки индекса.

Дополнительные виды индексов

Индексы соединения.
Идея происходит от классической экспериментальной системы IBM System R и заключается в том, чтобы индексировать совместно таблицы, которые требуется соединять. Обычно используют обычные B+-деревья. В поле записи этих деревьев содержится список идентификаторов записей соединяемых таблиц.

Полнотекстовые индексы.