Хранение данных и доступ к ним

Бессарабов Н.В.
bes@fpm.kubsu.ru
2014 г.

Цели лекции

Будут рассмотрены структуры хранения данных, доступ к данным, их буферирование, индексы, представления таблиц в базах данных табличного типа. Из-за обширности изучаемого предмета и недостатка времени изложение будет отрывочным.

Бегло рассмотрим планы исполнения и оптимизацию запросов SQL. Уже говорилось о том, что языки баз данных как правило декларативны. Например, запрос в минмальном варианте SQL указывает какими свойствами должны обладать данные, образующие ответ, но ничего не говорит о том, как этот ответ будет получен. Иначе говоря, отсутствует процедурная семантика. План исполнения запроса может быть выбран не единственным способом и от выбранного варианта существенно зависит производительность.

Изучение планов исполнения позволит получить первые представления о настройке приложения и немного приблизиться к пониманию SQL-тюнинга – одного из аспектов обширной области, которую принято называть администрированием баз данных.

Возможности проверить всё своими руками у вас не будет, за исключением планов исполнения.

Часть 1. Структуры хранения (1/4)

Замечание: терминология, применяемая в различных базах данных, различается существенно. Наша терминосистема ближе всего к применяемой в СУБД Oracle.

В Oracle база данных состоит из одного или нескольких табличных пространств. Каждое такое пространство строится на одном или нескольких файлах данных. В одно табличное пространство стараются помещать объекты с одинаковым поведением. Например, для словаря базы можно выделить отдельное табличное пространство, обычно называемое системным.

Пользовательские данные желательно помещать отдельно от словаря. Это уменьшит вероятность сбоя. Для, индексов следует иметь свои табличные пространства.

В некоторых СУБД можно отключать отдельные табличные пространства и делать их доступными только по чтению. Для больших сортировок можно создавать временные табличные пространства.

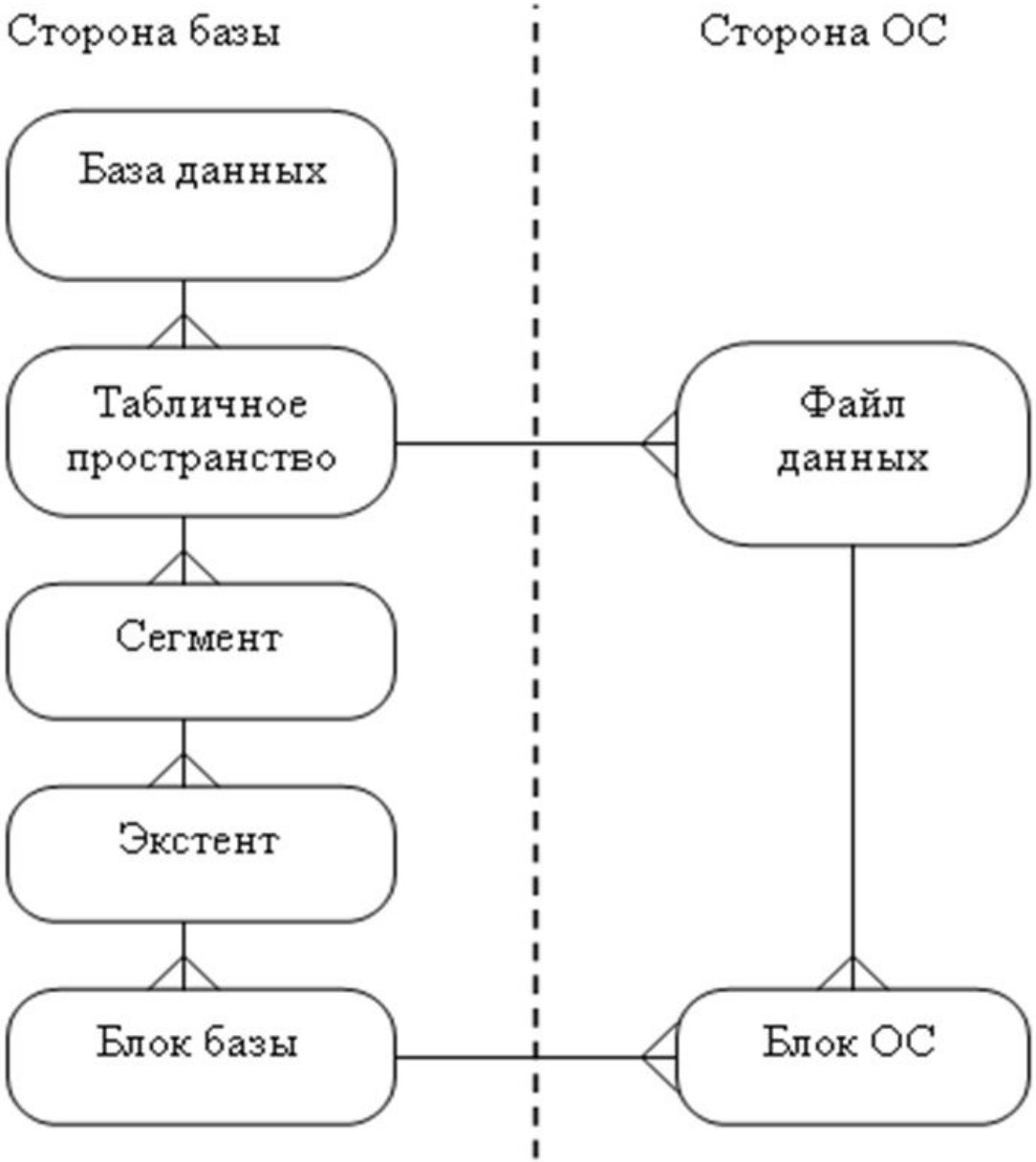
Администратор должен выбрать состав, размеры табличных пространств и определить, могут ли они расширяться, и какими порциями им будет предоставляться свободное пространство дисковой памяти.

Структуры хранения (2/4)

Табличные пространства состоят из сегментов, содержащих хранимые объекты базы, например, таблицы, индексы. Каждому такому объекту положено иметь свой сегмент, куда нет доступа данным других объектов.

Сегменты состоят из экстентов, представляющих наборы блоков данных базы, расположенных на диске непрерывно. Это ускоряет операции с блоками данных, входящими в состав экстента. Можно, например, при работе с любым элементом данных, читать сразу весь экстент, в надежде, что эти данные скоро понадобятся. Нетрудно догадаться, что сегмент увеличивается или уменьшается на целое число экстентов.

Структуры хранения (3/4)



Замечание: Это структура данных для СУБД Oracle.

Структуры хранения (4/4)

Блок базы, в другой терминологии страница памяти, это минимальная единица хранения, которой база данных обменивается с диском. Блок базы образуется из нескольких блоков операционной системы.

Существует несколько списков блоков пригодных для записи.

Ниже приведен возможный формат строки.

Часть 2. Индексы

Индексы **могут ускорить доступ к данным**, **но не всегда это делают, зато могут замедлить манипулирование данными**.

СУБД, как правило, поддерживают следующие типы индексов:

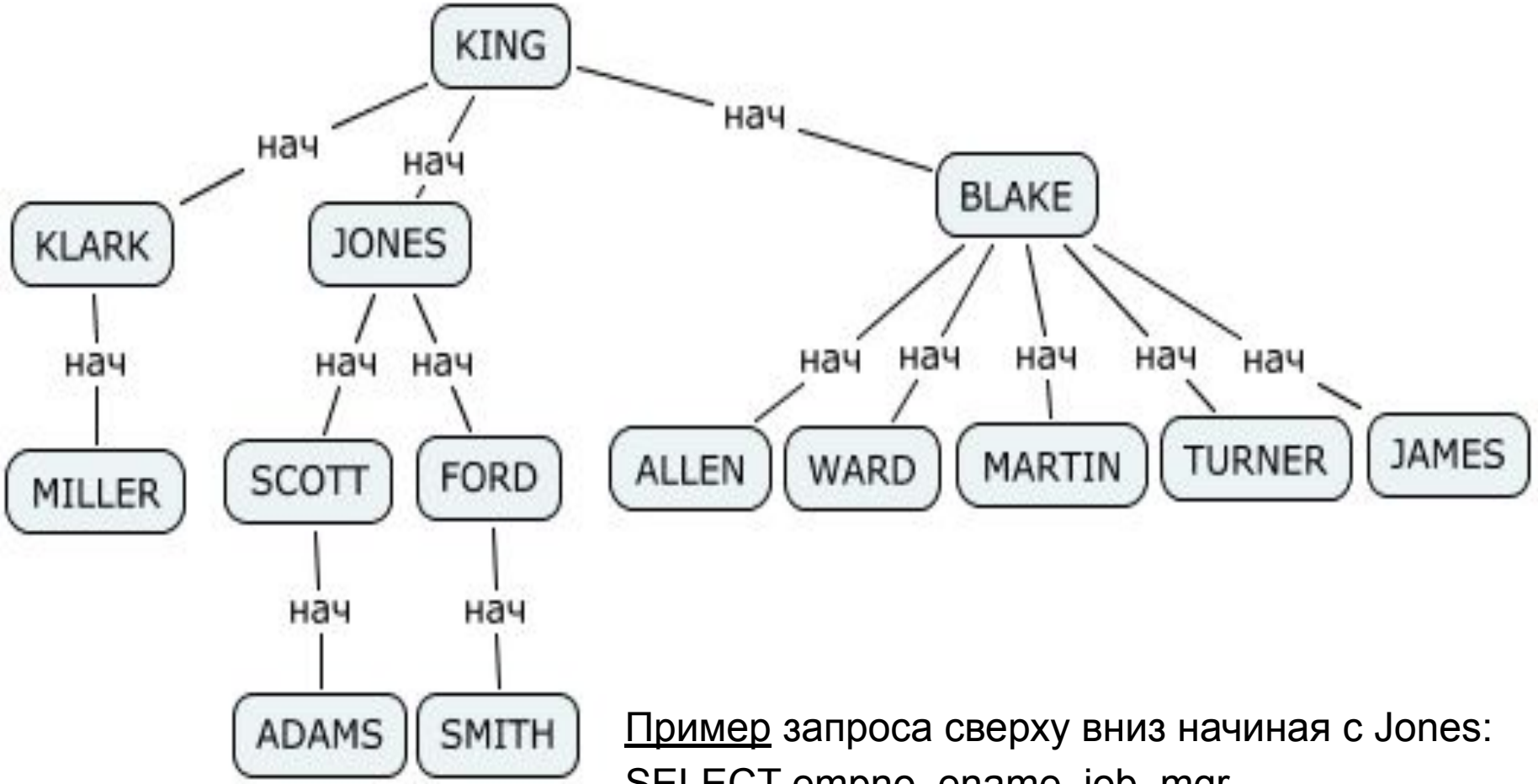
- Древесный индекс на основе B*-деревьев.
- Побитовые индексы -- bitmap index.

Альтернатива индексированию -- хеширование.

В реальной СУБД могут существовать другие типы индексов и структур, связанных с ними, например, индексы с обратными ключами, которые могут быть эффективными в параллельных серверах. Часто используют индексные таблицы (index-organized table). Это разновидность B*-индекса, в которой листовые блоки индекса содержат не значения ROWID, адресующие данные, а сами данные.

Важное замечание в связи с текстом, выделенным на второй строке:
Структуры и процессы, которые рассматриваются в администрировании таковы, что почти на любое утверждение, можно найти контрпример. Поэтому важно всегда понимать контекст, в котором делается высказывание. Забегая вперёд, заметим, что индекс на таблицу занимающую один блок всегда замедляет запрос, хотя и не значительно.

Иерархическая структура в таблице EMP

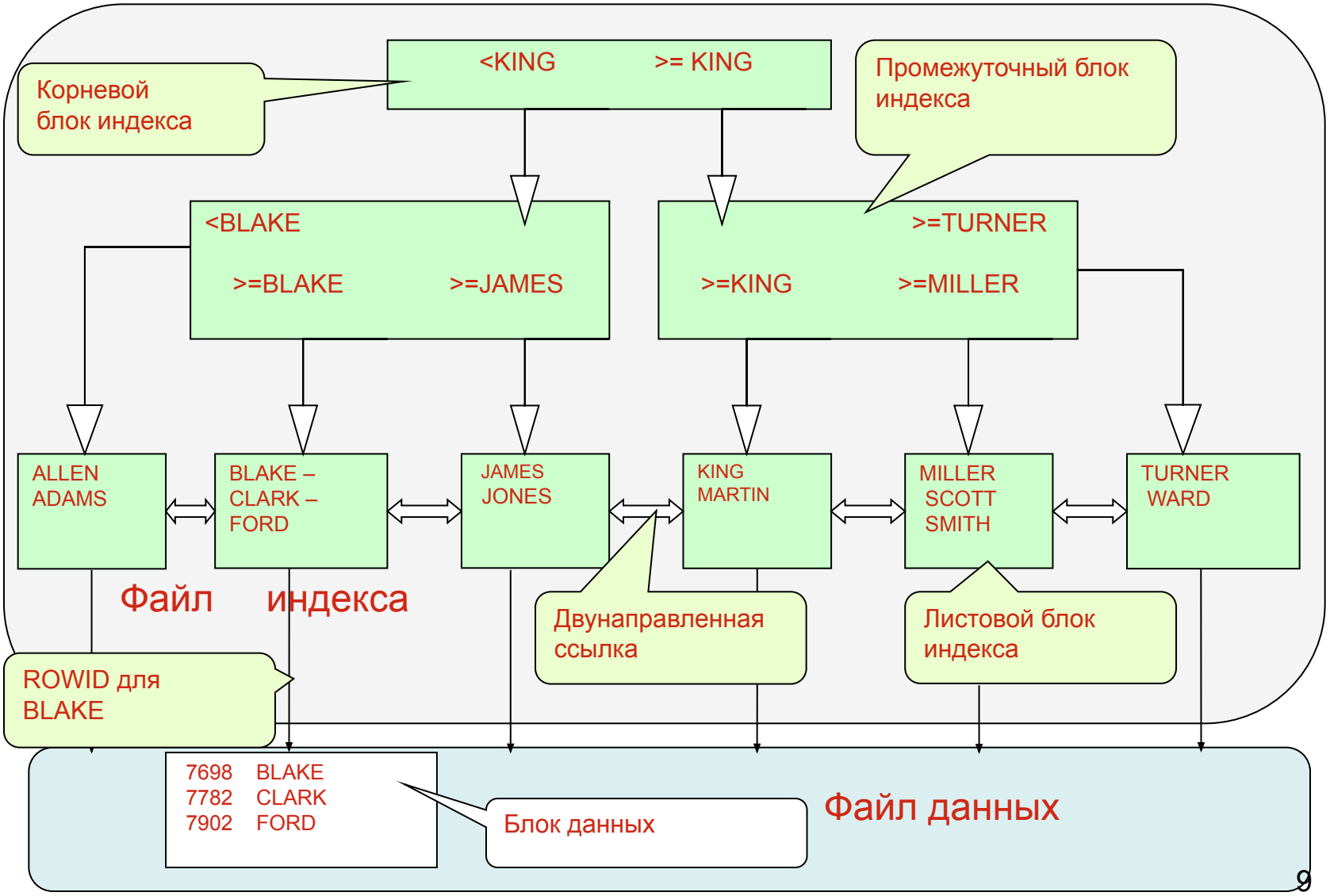


Пример запроса сверху вниз начиная с Jones:

```
SELECT empno, ename, job, mgr
FROM emp
START WITH empno = 7566
CONNECT BY PRIOR mgr = empno
```

Пример работы B*-индекса (1/2)

Выборка из EMP по условию ENAME=BLAKE. Работает индекс на ENAME



Пример работы B*-индекса (2/2)

Как работает индекс в примере выше?

Сначала просматривается корневой блок и по условиям <KING и >=KING определяется на который из двух дочерних блоков перейти. Идём по левой ветви. В блоке первого уровня проверяются условия <BLAKE, >=BLAKE и >=JAMES. Затем из листового блока второго уровня, содержащего три значения ROWID выбираем идентификатор для BLAKE и используя его как адрес выбираем значение из файла данных.

Известно, что глубина файла индекса для современных данных вряд ли превысит 5.

Заметим, что для поиска нужного значения пришлось просмотреть и, может быть, извлечь три блока индекса и один блок данных. Если бы индекс не работал, пришлось бы просматривать все блоки таблицы. Теперь понятно, что для таблицы, занимающей один блок индекс вреден.

Следует также помнить, что хотя индекс может улучшить скорость выполнения запросов, но он обязательно уменьшает скорость при добавлении и обновлении записей в таблице.

О работе B*-индекса (1/2)

В общем случае при поиске по значению ключа или по диапазону значений проходят по дереву индекса от корня через блоки ветвей до крайнего левого листового блока, отвечающего условию поиска.

Если индекс не уникальный и при выборе диапазона, дальнейшее движение может происходить вправо по цепочке ссылок между листьями, пока не выберутся все значения ссылок.

Вставка нового значения ключа производится в соответствующий листовой блок. Значения ключа обновляются путём удаления старой и вставки новой записей индекса. Переполняющийся блок дробится. При этом добавляется пустой блок и значения ключей поровну распределяются между старым и новым блоками, за исключением случая, когда дробится самый правый листовой блок.

Процесс дробления листовых блоков может рекурсивно перейти на промежуточные узлы.

Заметим, что блокирование на уровне строк удобно реализуется при описанной структуре индекса задающего адреса именно строк.

О работе B*-индекса (2/2)

Если строки индекса постоянно удаляются и добавляются, то через какое-то время индекс может “разредиться” или фрагментироваться. Обычно индекс расширяется вправо, и разрежается слева. Со временем объём индекса может существенно превысить объём данных. Дерево индекса может при этом стать глубже, чем должно быть для такого числа значений. Это уменьшает скорость индексного доступа. Поскольку типовой механизм организации B*индекса не поддерживает динамического уплотнения и перебалансирования дерева, то лучшее решение в этом случае – пересоздание индекса (уничтожение старого индекса, физическая сортировка таблицы и последующее создание нового индекса).

Замечание: Oracle научился ремонтировать индексы “на ходу” не лишая пользователей возможности работать с индексами.

Когда B*-индекс ускоряет запрос?

В материалах Oracle 80-х -90-х годов можно было найти “золотое правило”, в соответствии с которым неуникальный индекс ускоряет работу, если запрос возвращает меньше, чем 10-15% строк таблицы. Позже эти цифры заменили на 3-5%. Покажем, что при некоторых условиях индекс может быть неэффективен при любом числе возвращаемых строк.

Пусть таблица содержит 10 000 строк по 100 строк в каждом из 100 блоков. Ключевой столбец содержит значения от 0 до 99. Строки равномерно распределены по блокам, так что в любом блоке с большой вероятностью содержатся строки с любым значением ключа.

При выполнении запроса по одному значению ключа скорее всего будут прочитаны все блоки, хотя нужно выбрать всего 1% строк. Кроме того, будут прочитаны еще и все блоки индекса, что увеличит время выполнения запроса. Если индекс не используется, то выбираются все 100 блоков таблицы. В рассмотренной ситуации индекс не эффективен.

Отсортируем строки таблицы по ключу. Предположим, что распределение строк по значениям ключа равномерное. При использовании индекса будет выбран всего один или небольшое число блоков данных и блоки индекса (а не 1000 блоков таблицы). Индекс ускоряет запрос.

Индексы битовой карты (1/2)

Побитовые (BitMapped) индексы это разновидность не уникальных индексов. Побитовые индексы эффективны при малой разрешающей способности ключа. Функционально битовый индекс идентичен обычному древесному индексу, хотя внутреннее их устройство различно. Сами битовые индексы хранятся в виде B*-структуры.

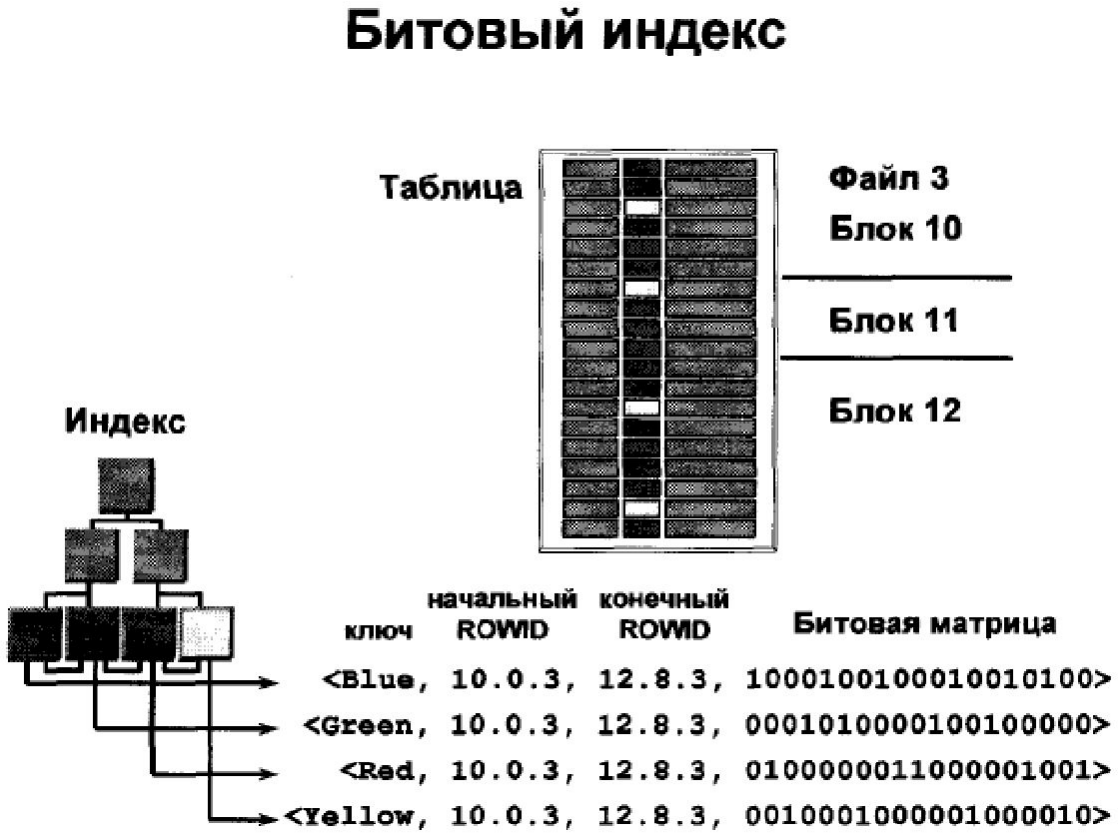
Структура строки битового индекса (в Oracle):
(<значение_ключа, начальное_значение_rowid, конечное_значение_rowid, сегмент_битовой_карты>).

где:

- начальное и конечное значения rowid указывает диапазон строк в таблице с конкретным значением ключа
- сегмент битовой карты это длинное битовое поле; установка бита в 1 означает наличие значения, а в 0 — на отсутствие значения ключа.

Сравним индексы. Пара <значение ключа, rowid> в B*-индексе заменена парой <значение ключа, сегмент двоичной карты>, где “значение_ключа” состоит из колонок “значение_ключа”, “начало_rowid” и “конец_rowid”. Битовые индексы, как и древесные, могут быть конкатенированными.

Индексы битовой карты (2/2)

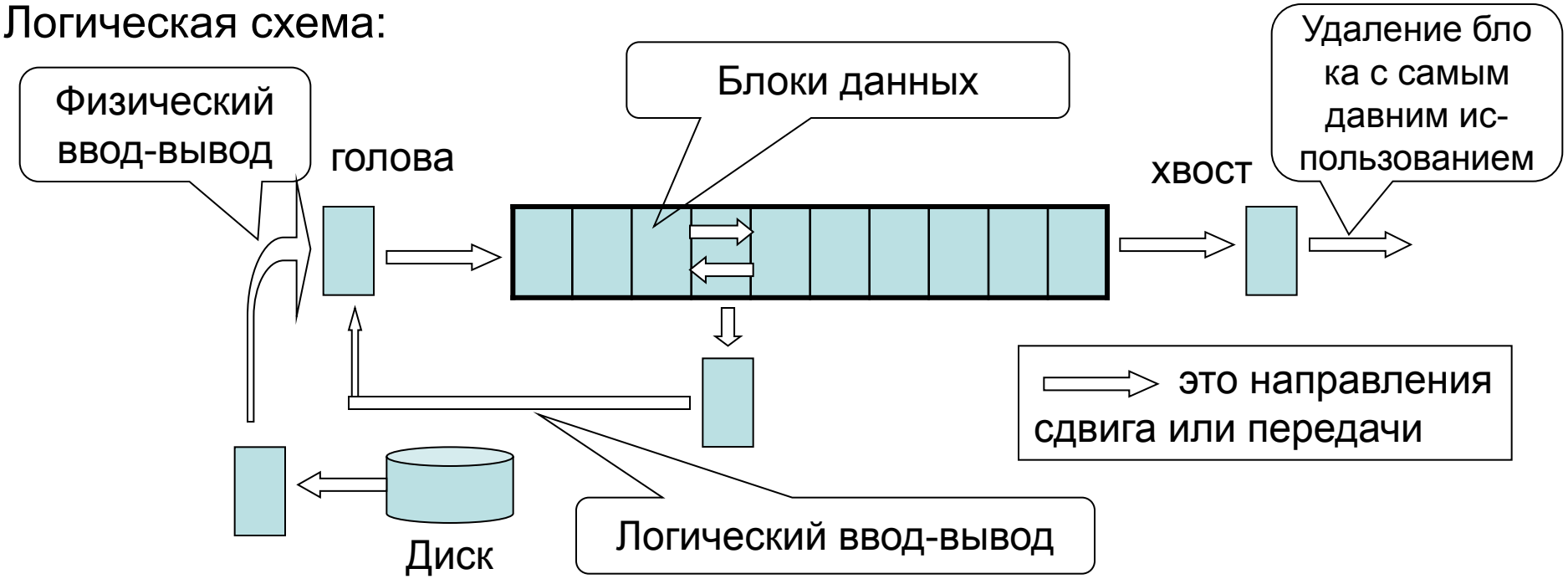


Операции с такими индексами могут выполняться очень быстро, так как логические операции над битовыми матрицами транслируются непосредственно в команды центрального процессора, выполняющие побитовые операции над словами длиной 32 или 64 бита.

Часть 3. Доступ к данным

Кэш буферов данных, реализующий алгоритм LRU

Логическая схема:



1. Физическое чтение. Считываемый блок помещается в голову. Содержимое кэша сдвигается на один блок. Хвостовой блок выталкивается.
2. Логическое чтение. Блок из кэша перемещается в голову. Часть кэша левее выбранного блока смещается вправо на один блок.

Важное замечание: При числе блоков в сотни тысяч и более алгоритм не реализуем по скорости. Выполняют эквивалентные действия с указателями.

Доступ к единственной таблице

Существует два варианта доступа к одной таблице:

1. Полное сканирование таблицы.
2. Индексный доступ к таблице.

Полное сканирование таблицы это чтение всех блоков таблицы без использования индексов. Одна из возможных неприятностей в том, что при использовании стратегии LRU сканирование большой таблицы может удалить из кэша многие блоки данных и индексов других таблиц. Это вызвало бы снижение производительности запросов к другим таблицам. Чтобы этого не случилось, алгоритм изменяют так, чтобы блоки, полученные при полном сканировании большой таблицы, отправлялись в хвост кэша. По миновании необходимости их заменяют на следующую группу блоков.

Соединения

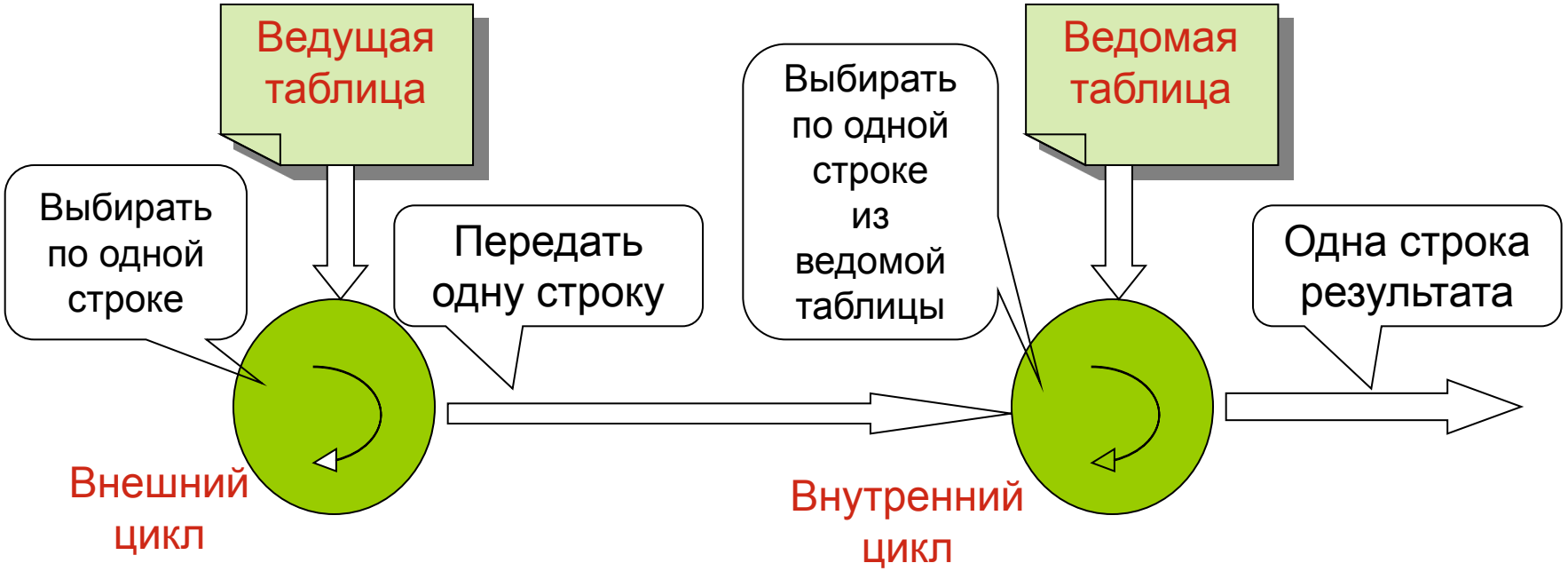
Рассмотренные ранее (воображаемые!) алгоритмы выполнения запросов SQL к нескольким таблицам, основаны на создании декартового произведения, которое даже для не очень больших таблиц вычисляется неприемлемо медленно.

Мы рассмотрим три способа реализации соединений, используемые в практике – соединения при помощи вложенных циклов (nested loops), соединения хешированием (hash join) и соединения с сортировкой слиянием (merge join).

В соединениях хэшированием и с сортировкой слиянием сначала обращаются к каждой таблице отдельно, а затем соединяют соответствующие строки и отбрасывают ненужные.

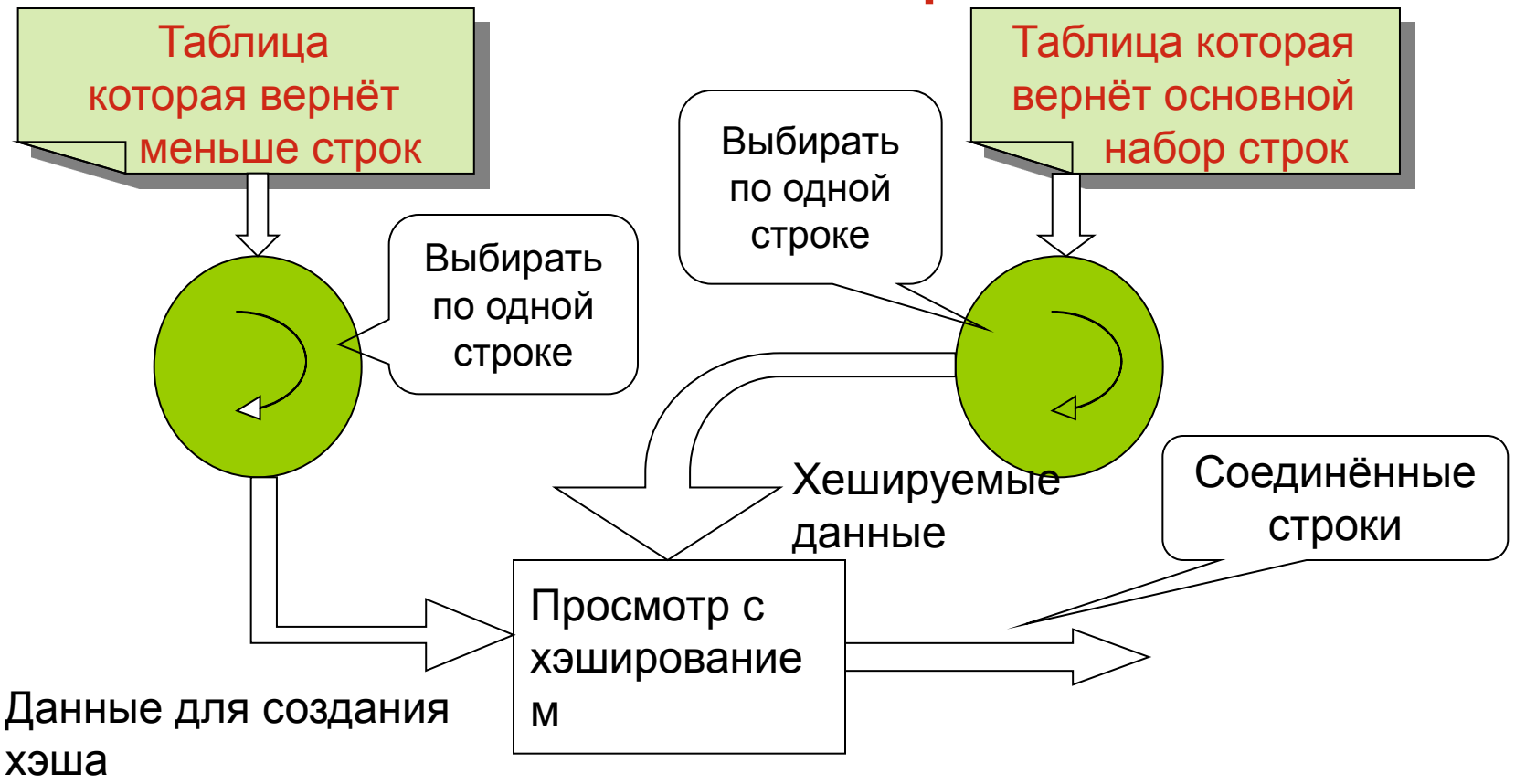
В дальнейшем это позволит нам понимать конструкции планов исполнения запросов SQL. А овладение знаниями и навыками управления планами исполнения – это ещё один слой знаний SQL, совершенно необходимый для написания запросов на профессиональном уровне.

Соединение при помощи вложенных циклов



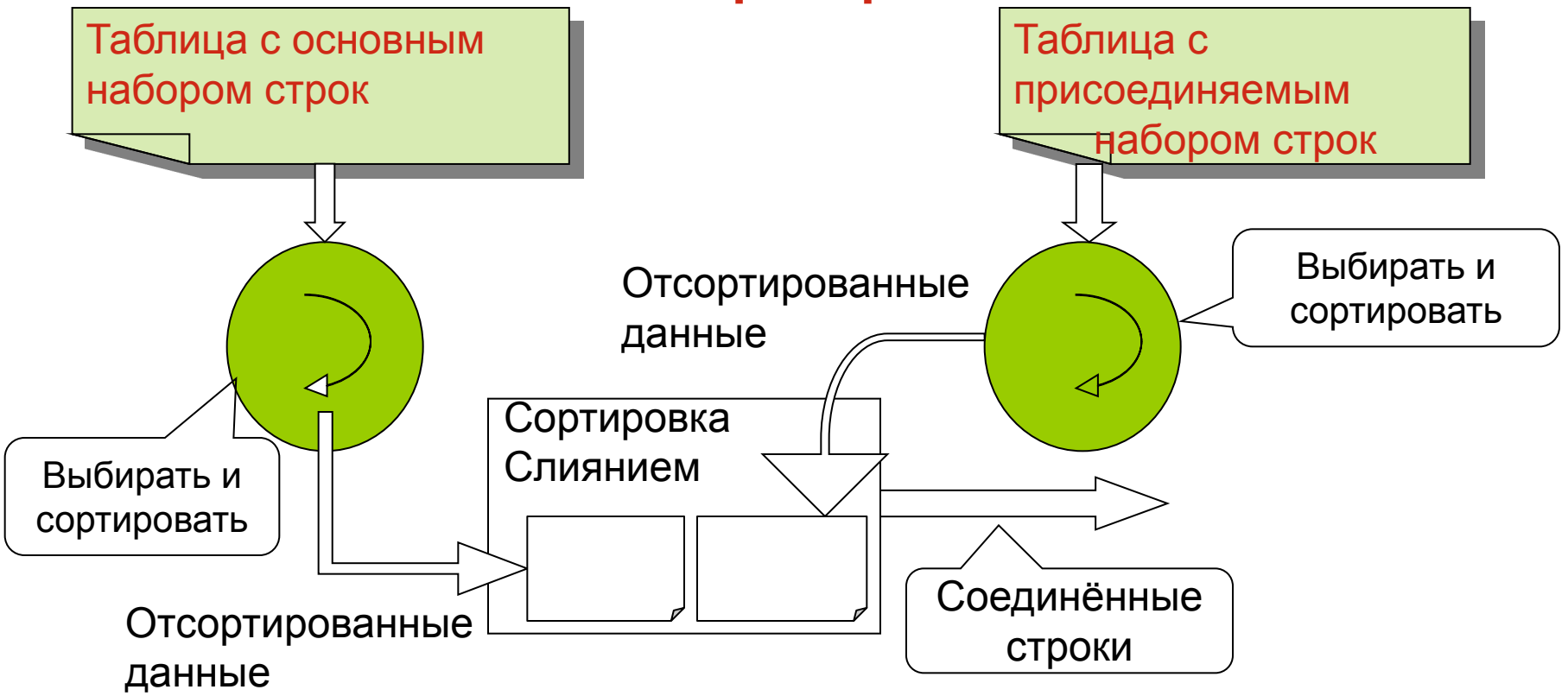
Внешний цикл выполняет фактически однотабличный запрос к ведущей таблице, используя только условия, относящиеся к этой таблице. Каждая найденная строка передаётся внутреннему циклу, который, перебирая строки ведомой таблицы, ищет по одной все подходящие строки. Первая такая строка формирует первую строку результата, передавая её в результирующую таблицу. После того, как будут перебраны все строки ведомой таблицы, внешний цикл, выберет следующую строку ведущей таблицы и т.д.

Соединение хешированием



Два цикла выполняют фактически независимые оптимизированные однотабличные запросы к исходным таблицам, используя для каждой свои условия. Оптимизатор выбирает таблицу, которая вернёт меньше строк и строит по ней хеш-функцию. Затем выполняется второй запрос с подборкой для результатов соответствующей области хеширования.

Соединение с сортировкой слиянием



Таблицы считываются независимо. Оба результирующих набора предварительно сортируются по ключу соединения и затем соединяются.

Можно представлять, что два отсортированных списка помещены рядом. Указатели смещаются с верхних записей только вниз. При временно фиксированном левом указателе правый идёт вниз до конца, задерживаясь только на соединяемых записях. Затем левый указатель опускается на шаг, а правый начинает движение вниз с позиции на строку ниже, чем в предыдущем цикле.

Сравнения соединений

1. Соединение при помощи вложенных циклов каждый раз формирует в оперативной памяти единственную строку результата. Требуется немного оперативной памяти. Место на диске не нужно. Можно создавать огромные результирующие таблицы при ограниченной оперативной памяти.
2. В соединении хэшированием меньший набор строк может оказаться неожиданно большим. Тогда потребуется дополнительное пространство на диске и процесс замедлится. Соединение хэшированием следует предпочесть соединению при помощи вложенных циклов только если есть уверенность в том, что меньший набор строк поместится в оперативную память.
3. В соединении с сортировкой слиянием предварительная сортировка данных может занять много времени и ресурсов. Если необходимо выбрать между соединениями с сортировкой слиянием и с хэшированием следует всегда выбирать соединения с хэшированием.

Часть 4. Планы исполнения

Создавая запрос SQL пользователь указывает какими свойствами обладают нужные ему данные, но ничего не говорит о том, как именно они получаются. Это облегчает жизнь программиста, но ровно до тех пор, пока производительность запроса остаётся удовлетворительной.

План исполнения (выполнения) описывает алгоритм используемый при выполнении запроса. В частности, определяются пути доступа (использование индексов, их объединение или игнорирование) и порядок соединений (в каком порядке обращаются к таблицам). Для оптимизации запроса важно определить селективность условий, то есть установить, какую долю записей определяет соответствующий предикат.

Выбирается план исполнения либо администратором вручную, либо одним из встроенных оптимизаторов автоматически.

Настройка (tuning) SQL это ещё один слой знаний, умений и навыков, которым должны владеть квалифицированный разработчик и администратор баз данных.

Оптимизация по правилам и по стоимости

Два основных способа:

- Оптимизация по правилам (RULE BASED). Учитываются только способы доступа к данным. Ранги эффективности доступа установлены заранее и не учитывают особенности действующей ситуации. Опытный администратор часто создаёт лучшие планы.
- Оптимизация по стоимости (COST BASED). Учитываются и способы доступа к данным, и статистика размещения данных и ресурсов.

Выбор режима работы оптимизатора определяет параметр `optimizer_mode`, который в Oracle может иметь значения:

- `optimizer_mode = rule - RBO;`
- `optimizer_mode = all_rows - CBO`, установлен по умолчанию.
- `optimizer_mode = first_rows - CBO`, пытается выбрать план, который наиболее быстро возвращает первые строки.

Существуют другие варианты.

Ранжирование методов доступа

Ранг	Метод доступа
1	одна строка по ее идентификатору
2	одна строка по объединению кластеров
3	одна строка по хеш-ключу кластера с уникальным или первичным ключом
4	одна строка по уникальному или первичному ключу
5	объединение кластеров
6	хеш-ключ кластера
7	индекс кластера
8	составной индекс
9	индекс на основе одного столбца
10	ограниченный диапазон поиска по индексированным столбцам
11	неограниченный диапазон поиска по индексированным столбцам
12	объединение с сортировкой и слиянием
13	поиск минимального или максимального значения по индексированным столбцам
14	упорядочение по индексированным столбцам
15	полное сканирование таблицы

Статистики для оптимизатора по стоимости

CBO-оптимизатор использует для определения стоимости *пути доступа статистики:* число элементов таблицы, число возможных значений столбца и распределение данных.

Стоимость является мерой того, сколько памяти, ресурсов процессора и каналов ввода-вывода потребуется для выполнения запроса.

Сначала необходимо собрать статистику числа элементов (cardinality) и распределения данных для используемых в запросе таблиц, индексов и материализованных представлений.

В Oracle статистика собирается пакетом DBMS_STATS. В нём имеются процедуры для сбора статистики уровня базы данных, схемы или таблицы, а также раздела таблицы.

Пример команды сбора статистики:

```
ANALYSE TABLE employees COMPUTE STATISTICS
    FOR TABLE FOR ALL INDEXES FOR ALL INDEXED COLUMNS;
```

Статистики таблиц хранятся в представлении USER_TABLES.

Подсказки

Управлять планом исполнения можно размещая после слова SELECT подсказки в виде комментариев специального вида (hints).

Например, подсказка в запросе

```
SELECT /*+INDEX*/ empno FROM emp WHERE empno = 1739;
```

означает требование воспользоваться индексом. Правда оптимизатор может и не выполнить указание.

Некоторые подсказки:

Подсказка	Пояснение
FULL(таблица)	Выполнение полного просмотра таблицы
CASH	Разместить сканированную таблицу в кэше для сохранения ее блоков в памяти для последующего быстрого доступа
INDEX (индекс)	Использовать указанный индекс
USE_NL	Использовать вложенные циклы для объединения таблиц

Примеры планов исполнения (1/5)

Освоение SQL-настройки требует знания массы сведений об используемой СУБД, её физической организации и конфигурационных файлах.

Трудность ещё и в том, что в современных СУБД, таких как Oracle, этот аспект может за один – два года существенно измениться. Усовершенствуются оптимизаторы, в них вводятся системы искусственного интеллекта и т.д.

Из-за ограниченности времени в нашем курсе мы можем только показать несколько примеров планов, дав минимальные пояснения. Искусство управления планами, работа с оптимизаторами, сбор статистики остаются за кадром.

Примеры планов приведенные на последующих слайдах получены в СУБД Oracle. Их следует читать из глубины вверх. Помните, что выбор плана исполнения сильно зависит от настройки СУБД и её версии, так что при самостоятельной работе Вы можете получить совсем другие результаты. Как писал один из авторов хорошей книги по тюнингу, “не верь тому, что здесь написано” .

Примеры планов исполнения (2/5)

Примеры планов:

1. Простейший запрос

   SELECT * FROM emp;

   План исполнения:

   SELECT STATEMENT

      TABLE ACCESS full emp

2. Запрос с фразой WHERE и по-прежнему без индексов

   SELECT * FROM emp WHERE sal>1000;

   План исполнения тот же, хотя после извлечения данных работает фильтр, определённый фразой WHERE.

3. Запрос

   SELECT * FROM emp ORDER BY ename;

   План исполнения

   SELECT STATEMENT

      SORT order by

         TABLE ACCESS full emp

   Добавилась сортировка в памяти, а может быть и на диске.

Примеры планов исполнения (3/5)

4. Тот же запрос

```
SELECT * FROM emp ORDER BY ename;
```

но существует индекс на столбец ename. Имя индекса i_emp_ename.

План исполнения:

```
SELECT STATEMENT
    TABLE ACCESS full emp
   INDEX full scan i_emp_ename
```

Поскольку используется индекс, сортировка не нужна.

5. Запрос

```
SELECT job, sum(sal) FROM emp GROUP BY job
HAVING sum(sal)> 100000;
```

Индекс не существует.

План исполнения:

```
SELECT STATEMENT
FILTER
   SORT group by
      TABLE ACCESS full emp
```

Примеры планов исполнения (4/5)

6. Доступ по значению ROWID. Запрос:

SELECT * FROM emp WHERE rowid='00004F2A00A2000C';

Самый быстрый план исполнения:

SELECT STATEMENT

TABLE ACCESS by rowid emp

7. Соединение с вложенными циклами

SELECT * FROM emp, dept;

План исполнения:

SELECT STATEMENT

NESTED LOOPS

TABLE ACCESS full dept

TABLE ACCESS full emp

8. Запрет на использование индекса

SELECT ename FROM emp WHERE job='MANAGER'||' ';

Примеры планов исполнения (5/5)

9. Сортировка слиянием

   SELECT * FROM emp, dept WHERE emp.deptno=dept.deptno;

   План исполнения:

   SELECT STATEMENT

   MERGE JOIN

   SORT JOIN

   TABLE ACCESS full emp

   SORT JOIN

   TABLE ACCESS full emp

10. Тот же запрос, но существует индекс на столбец deptno играющий роль внешнего ключа в emp.

    План исполнения:

    SELECT STATEMENT

    NESTED LOOPS

    TABLE ACCESS full dept

    TABLE ACCESS by rowid emp

    INDEX range scan idx_fk_emp_deptno

Заключение

Полученных скудных сведений об архитектуре баз данных, индексах, способах доступа к данным и о SQL-тюнинге недостаточно для практической работы, однако мы теперь представляем сложность управления базами данных и начинаем догадываться о том громадном объёме труда и таланта, который вложен в создание любой хорошей СУБД, тем более в таких долгожителей как Oracle и Caché.

Есть основания полагать, что в таких задачах, как оптимизация планов исполнения, автоматического исправления индексов и устранения вредной фрагментации используются системы искусственного интеллекта, хотя сами вендоры об этом не упоминают.

Администрирование баз данных – это обширная область, требующая не только программистских знаний, но и большого объёма специфических знаний и навыков.

Хороших вам данных!!