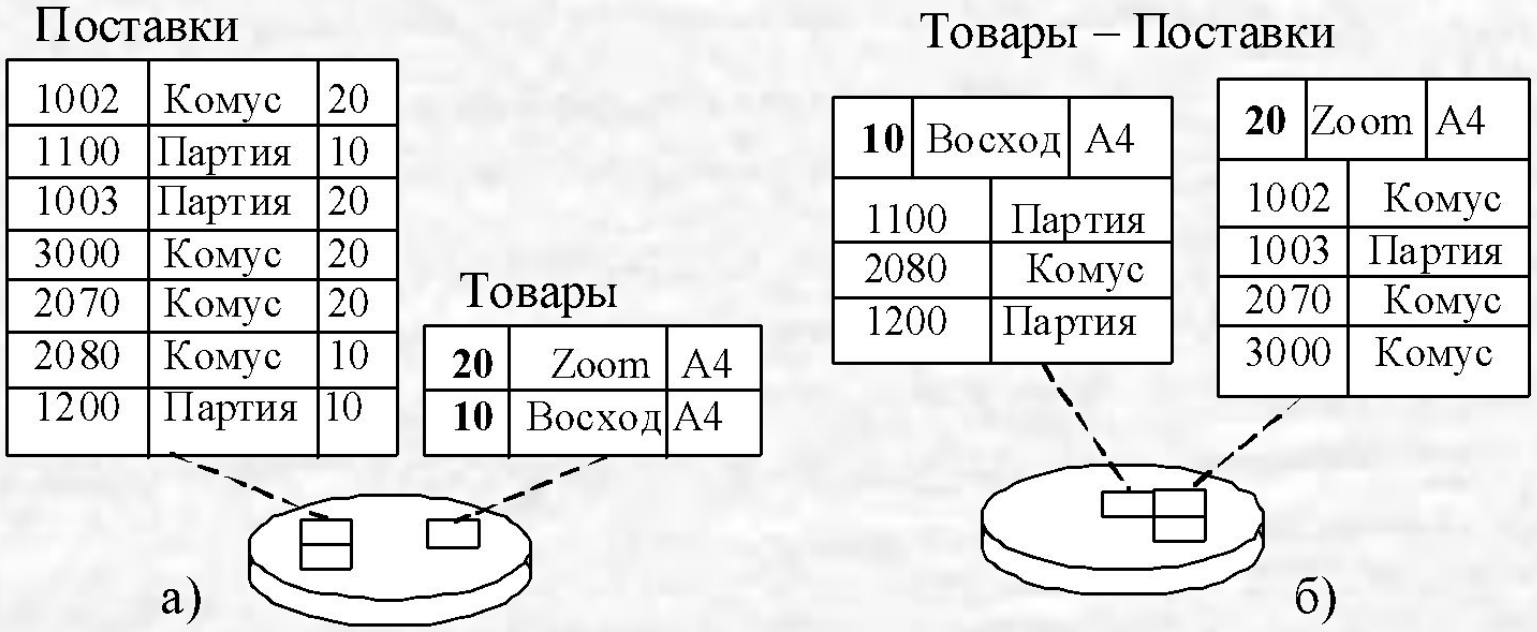
Организация размещения данных и доступа к данным

Хеширование и кластеризация.
Особенности СУБД Oracle

Кластеризация

Кластеризация является методом совместного хранения родственных данных (таблиц).

Кластер – это структура памяти, в которой хранится набор таблиц (в одних и тех же блоках памяти). Таблицы, помещаемые в кластер, должны иметь общие столбцы, используемые для соединения.



Кластеризация

Кластерный ключ (КК) – это поле или набор полей, общих для всех таблиц кластера. Каждая таблица, хранимая в кластере, должна иметь поля, соответствующие типам и размерам полей кластерного ключа. Количество полей в кластерном ключе ограничено (например, для СУБД Oracle8 это ограничение равно 16).

Фактически, данные хранятся в виде соединения таблиц по значениям кластерного ключа. Поэтому соединение кластеризованных таблиц по сравнению с раздельно хранимыми таблицами выполняется в 3-6 раз быстрее.

Если все данные, относящиеся к одному значению кластерного ключа, не помещаются в одном блоке, то выделяется новый блок памяти и предыдущий блок хранит ссылку на него. Но если система позволяет изменять размер блока, при создании кластера желательно установить размер блока исходя из оценки среднего объёма записей с одинаковыми значениями кластерного ключа. Если же записи с одинаковым значением КК занимают только часть блока (например, в среднем 1К при размере блока 4К), то при создании таблицы кластера можно указать количество значений КК на один блок.

Кластеризация: достоинства и недостатки

Два основных **преимущества** кластеров:

- Уменьшается время соединения таблиц по значению кластерного ключа.
- Каждое значение кластерного ключа хранится только один раз, за счёт чего достигается экономия памяти.

Недостатки:

- Наличие кластеров обычно увеличивает время выполнения операции добавления записи (INSERT): система тратит дополнительное время на поиск блока, в который нужно поместить новую запись.
- Чтение отдельной таблицы из кластера может занимать гораздо больше времени, чем чтение некластеризованной таблицы.

Значения кластерного ключа таблицы могут обновляться. Но это обновление может вызвать физическое перемещение записи, т.к. расположение записи зависит от значения кластерного ключа. Поэтому часто обновляющиеся атрибуты не являются хорошими кандидатами на вхождение в кластерный ключ.

Использование кластеров

Кластер создаётся с помощью команды CREATE CLUSTER:

create cluster <имя_кластера> (<имя_поля1> <тип_поля1>
[,<имя_поля2> <тип_поля2> ,…]);

Здесь в скобках перечисляются поля кластерного ключа.

Затем создаются таблицы в кластере:

create table <имя_таблицы> (<список полей таблицы>)
cluster <имя_кластера> (<список полей КК>);

Количество и типы полей кластерного ключа таблицы должны совпадать с количеством и типами полей КК в определении кластера, а имена полей могут быть другими. Типы данных в <списке полей КК> для таблицы не указываются.

Перед занесением данных в таблицы кластера необходимо создать *кластерный индекс* – индекс по кластерному ключу:

create index <имя_индекса> on cluster <имя_кластера>;

Поля для индексирования не указываются, потому что кластерный индекс создаётся по полям кластерного ключа. В отличие от обычного индекса в кластерном индексе null-значения индексируются.

Использование кластеров

Кластеры обычно строятся для таблиц, часто используемых в соединении друг с другом, например, связанных отношением "один-ко-многим".

Не стоит создавать кластер в следующих случаях:

- Если данные в кластерном ключе этих таблиц часто обновляются.
- Если часто требуется полный просмотр отдельной таблицы. Полный просмотр индивидуальных таблиц кластера требует больше времени, чем просмотр раздельно хранящихся таблиц, т.к. физически требуется обратиться к большему числу блоков. Если по отдельности некластеризованные таблицы занимают n1 и n2 блока соответственно, то вместе они будут занимать (n1+n2) блоков, и для полного просмотра каждой из них придётся обращаться к диску (n1+n2) раз.
- Если суммарные данные таблиц с одним и тем же значением кластерного ключа занимают больше одного блока данных. Второй и последующие блоки для одного и того же значения кластерного ключа выделяются не подряд, что вызывает частые перемещения считывающей головки диска и увеличение времени доступа к данным.

Создание кластеров

Последовательность создания кластера:

1) Создать кластер (cluster).
2) Создать таблицы в кластере (table).
3) Создать кластерный индекс (index ... on cluster).

Пример создания кластера для таблиц DEPART и EMP^

```
CREATE CLUSTER emp_dept (depno NUMBER(3))
        PCTUSED 80    PCTFREE 5    SIZE 600   TABLESPACE users
        STORAGE (INITIAL 100K  NEXT 200K  MINEXTENTS 2);
CREATE TABLE depart (
        depno NUMBER(3)  PRIMARY KEY, . . . );
        CLUSTER emp_dept (depno);
CREATE TABLE emp (
        tabno NUMBER(5)  PRIMARY  KEY,
        depno NUMBER(3)  REFERENCES  depart, . . .)
        CLUSTER emp_dept (depno);
CREATE INDEX emp_dept_index  ON CLUSTER emp_dept
        INITRANS 2   MAXTRANS 5   TABLESPACE   inds);
```

Хеширование

Принцип хеширования: для определения адреса записи в области хранения к значению ключевого поля этой записи применяется *хеш-функция* ***h(K)***, которая преобразует значение ключа ***K*** в адрес участка памяти (это называется *свёрткой ключа*). Новая запись будет размещаться по тому адресу, который выдаст хеш-функция для ключа этой записи.

Хеш-функция *h(K)* должна обладать следующими свойствами:

- выдавать такие значения адресов, чтобы обеспечить равномерное распределение записей в памяти, в частности, для близких значений ключа значения адресов должны сильно отличаться, чтобы избегать перекосов в размещении данных:

$$K1 \approx K2 \Rightarrow h(K1) >> h(K2) \ V \ h(K2) >> h(K1),$$

- для разных значений ключа выдавать разные адреса:

$$K1 \neq K2 \Rightarrow h(K1) \neq h(K2).$$

Для реальных функций хеширования допускается совпадение значений функции *h(K)* для различных ключей. Для разрешения неопределённости при совпадении адресов после вычисления *h(K)* используются специальные методы.

Хеширование: достоинства и недостатки

Достоинства:

- обращение к данным по значению ключа происходит за одну операцию ввода/вывода;
- не нужно создавать никаких дополнительных структур (типа индекса) и тратить память на их хранение.

Недостатки:

- количество данных и распределение значений ключа должны быть известны заранее;
- записи обычно неупорядочены по значению ключа, что приводит к дополнительным затратам, например, при выполнении сортировки;
- сложности подбора хеш-функции.

Хеширование: сложности подбора хеш-функции

Пример: парадокс дней рождения.

23 человека (случайным образом) собрались в одной комнате. С вероятностью 0.507 у двух из них дни рождения (число и месяц) совпадут.

Т.е. при случайном выборе функции, отображающей 23 элемента в 365 ячеек, эта функция с вероятностью 0.507 поместит два элемента в одну ячейку.

n	$p(n)$
10	12 %
20	41 %
30	70 %
50	97 %
100	99,99996 %

Методы хеширования

Все рассуждения ведутся в предположении, что хеш-функция *h(K): 0≤h(K)≤N* для всех ключей *K*, где *N* – размер памяти (количество ячеек).

1) Метод деления использует остаток от деления на M:

$$h(K) = K \bmod M$$

Если *M* – чётное число, то при чётных *K* значение *h(K)* будет чётным, и наоборот, что даёт значительные смещения значений функции для близких значений *K*. Нельзя брать *M* кратным основанию системы счисления машины или кратным 3. *M* должно удовлетворять условию:

$$M \neq rk \pm a ,$$

где *k* и *a* – небольшие числа, а *r* – "основание системы счисления" для большинства используемых литер (как правило, 128 или 256), т.к. остаток от деления на такое число оказывается обычно простой суперпозицией цифр ключа. Чаще всего в качестве *M* берут простое число, например, вполне удовлетворительные результаты даёт ***M* = 1009**.

Методы хеширования

2) Мультипликативный метод. В соответствии с ним хеш-функция определяется так:

$$h(K) = M\left(\left(\frac{A}{w}K\right) \bmod 1\right)$$

где ***w*** – размер машинного слова (обычно, 2^{31}); ***A*** – целое число простое по отношению к ***w***; а ***M*** – некоторая степень основания системы счисления ЭВМ (2^m). Таким образом, в качестве значения функции берутся ***M*** правых значащих цифр дробной части произведения значения ключа и константы ***A/w***.

При использовании любых методов хеширования для размещения записей должен быть выделен участок памяти размером *N*. Для того чтобы полученное в результате значение *h(K)* не вышло за границы отведённого участка памяти, окончательно адрес записи вычисляется так:

$$A(K) = h(K) \bmod N.$$

Разрешение коллизий

Коллизия – случай, когда для двух и более ключей выдаётся одинаковый адрес.

Разрешение коллизий – **рехеширование** – специальный алгоритм, который используется каждый раз при размещении новой записи или при поиске существующей, если возникла коллизия.

В системах БД рехеширование выполняется одним из следующих способов:

1. **Открытая адресация**: новая запись размещается вслед за последней записью на данной странице или на следующей, если страница заполнена.
2. Использование **коллизионных страниц**: новая запись размещается на одной из коллизионных страниц, относящихся к таблице (в *области переполнения*). Для ускорения поиска рехешированных записей может использоваться связанная область переполнения, для которой на странице хранится ссылка на коллизионную страницу. Нулевое значение такой ссылки говорит об отсутствии коллизий для данных, размещённых на этой странице.
3. **Многократное хеширование**. Заключается в том, что при возникновении коллизии для поиска другого адреса (возможно, на коллизионных страницах) применяется другая функция хеширования.

Использование хеширования

Хеширование таблицы полезно в следующих случаях:

- В таблице есть уникальный ключ, и большинство запросов обращается к записям по значению этого ключа, например:

```
SELECT <список выбора>
FROM <таблица>
WHERE unique_key = <значение>;
```

Значение, указанное в условии, хешируется; по этому хеш-значению происходит прямой доступ к соответствующему блоку данных (обычно, одно физическое чтение, если нет коллизий и запись помещается в одном блоке).

- Для неуникального хеш-ключа все записи с таким значением ключа помещаются в одном блоке, который также можно прочитать за один раз.
- Таблица практически статична (редко обновляется). Число записей и их средний размер можно определить заранее и сразу выделить под таблицу требуемое физическое пространство.

Использование хеширования

Хеширование **не рекомендуется** в следующих случаях:

- Нельзя сразу выделить столько памяти, сколько требуется таблице. Если потребуется выделять таблице дополнительную память, эта память будет отведена под коллизионные страницы, что сильно ухудшит производительность (это следует из формулы, по которой рассчитывается адрес записи).
- Большинство запросов выбирает записи в некотором интервале значений ключа. Хеширование не даёт здесь преимуществ, т.к. записи обычно не упорядочены, и система использует последовательное чтение.

Эффективность использования хеширования не в последней степени определяется качеством хеш-функции. Системы, поддерживающие возможность хеширования данных, обычно имеют встроенную хеш-функцию, но и позволяют пользователю задавать свою. Это может понадобиться тогда, когда встроенная хеш-функция не даёт хороших результатов, а пользовательская хеш-функция может учесть особенности распределения значений конкретного ключа. Если же ключ является уникальным и распределение его значений равномерно, то сами значения могут быть использованы в качестве хеш-значений (тогда данные будут размещаться в порядке увеличения значений хеш-ключа).

Создание хеш-кластеров в Oracle

Хэш-кластер создается с помощью команды SQL CREATE CLUSTER.
Пример создания кластер с именем TRIAL_CLUSTER, который используется для хранения таблицы TRIAL, кластеризуемой по столбцу TRIALNO:

```
CREATE CLUSTER trial_cluster (trialno NUMBER(5,0))
   PCTUSED 80
   PCTFREE 5
   TABLESPACE users
   STORAGE (INITIAL   250K    NEXT   50K
         MINEXTENTS  1    MAXEXTENTS  3
         PCTINCREASE 0)
   SIZE 2K
   HASH IS  trialno  HASHKEYS  150;

CREATE TABLE trial (
   trialno  NUMBER(5,0)  PRIMARY KEY,
   ...)
   CLUSTER trial_cluster (trialno);
```

Хэш-функция

В ORACLE можно воспользоваться внутренней хэш-функцией или обойти её.

Внутренняя хеш-функция:

- позволяет кластерному ключу быть одиночным столбцом или составным ключом;
- ключ кластера может состоять из столбцов любого типа (за исключением LONG и LONG RAW).

Использование **уникального идентификатора в качестве хэш-функции**:

- ORACLE проверяет значение кластерного ключа. Если это значение меньше, чем HASHKEYS, то хэш-значение равно значению кластерного ключа; в противном случае за хэш-значение принимается остаток от деления значения кластерного ключа на HASHKEYS.
- указан параметр HASH IS, в котором специфицирован ключ кластера, являющийся одиночным столбцом, содержащим целочисленные значения.

Если внутренняя хэш-функция обходится, а значение кластерного ключа оказывается нецелым, то операция (INSERT или UPDATE) отменяется, и возвращается ошибка.

Управление использованием памяти в хэш-кластере

ВЫБОР КЛЮЧА. Выбор корректного ключа кластера зависит от типа запросов, которые наиболее часто выдаются по кластеризуемым таблицам.
В качестве ключа хеширования стоит выбирать то поле, по которому чаще всего происходят обращения при поиске данных.
Для хэш-кластеров, содержащих единственную таблицу, за ключ кластера обычно принимается полный первичный ключ этой таблицы.
Ключ хэш-кластера (как и ключ индексированного кластера) может быть одиночным столбцом или составным ключом (ключом из нескольких столбцов).
Хэш-кластер с составным ключом должен использовать внутреннюю хэш-функцию ORACLE.
УСТАНОВКА HASH IS. HASH IS специфицируется только тогда, когда ключ кластера является одиночным столбцом с типом данных NUMBER, и содержит равномерно распределенные целые значения. Если эти условия соблюдены, то строки в кластере распределяются так, что каждое уникальное значение ключа кластера хэшируется в уникальное хэш-значение (без коллизий). Если эти условия не соблюдены, эта опция не нужна, т.к. используется внутренняя хэш-функция ORACLE.

Управление использованием памяти в хэш-кластере

УСТАНОВКА SIZE. Параметр SIZE должен быть установлен как среднее количество памяти, требуемое для хранения всех строк с любым данным хэш-ключом. Поэтому для правильного задания SIZE необходимо знать характеристики данных.

* Если хэш-кластер должен содержать единственную таблицу, и значения хэш-ключа по строкам этой таблицы уникальны (одна строка на значение), то SIZE должен быть установлен как средний размер строки в кластере.

* Если хэш-кластер должен содержать несколько таблиц, то SIZE должен быть установлен как среднее количество памяти, необходимое для хранения всех строк, ассоциированных с репрезентативным хэш-значением.

* Переоценка значения SIZE увеличивает объем неиспользуемой памяти в кластере. Недооценка SIZE повышает эффективность использования памяти, но приводит к увеличению вероятности возникновения коллизий.

Управление использованием памяти в хэш-кластере

УСТАНОВКА HASHKEYS. Параметр HASHKEYS всегда следует устанавливать как число уникальных значений ключа кластера, округленное вверх до следующего простого числа (в предположении, что используются рекомендации предыдущих двух секций).
Для максимального распределения строк в хэш-кластере HASHKEYS всегда должно быть простым числом.
Например, для кластеризации таблицы EMP по номеру отдела DEPTNO: пусть существуют 100 номеров отделов, с значениями 10, 20, ..., 10000. Допустим, создается кластер при HASHKEYS = 100. Тогда отдел 10 хэшируется в 10, отдел 20 – в 20, ..., отдел 110 – в 10 (110 mod 100), отдел 120 – в 20, и так далее. Заметьте, что получится по десять строк для хэш-значений 10, 20, ..., но ни одной строки для хэш-значений 1, 2, ..., и т.д. Как следствие, будет много неиспользуемой памяти, и, возможно, много блоков переполнения из-за коллизий. Альтернативно, если установить HASHKEYS = 101, то каждый номер отдела хэшировался бы в уникальное значение хэш-ключа.

Распределение пространства для хэш-кластера

- Для хэш-кластера сразу выделяется столько памяти, чтобы отвести место для всех хэш-ключей кластера.
- Размер распределяемого пространства равен SIZE*HASHKEYS. (В действительности используется наибольшее из двух значений – SIZE*HASHKEYS и того, которое определяется фразой STORAGE).
- Последующие распределения памяти для хэш-кластера служат для размещения переполняющих строк из переполненных блоков данных.
- Частые коллизии могут привести к большому числу блоков переполнения в хэш-кластере, что уменьшает производительность извлечения данных.
- Если возникает коллизия, а в первоначальном блоке нет места для размещения строки, необходимо распределить блок переполнения для новой строки. Вероятность таких случаев большей частью зависит от средних размеров значений хэш-ключа и соответствующих данных, которые специфицируются при создании хэш-кластера.

Расчет памяти для хэш-кластеров

* Для хэш-кластеров необходимо определить размер памяти для каждого хэш-ключа, учитывая распределение ключей кластера среди хэш-ключей в кластере.
* В отличие от индексированного кластера, в хэш-кластере значение ключа кластера хранится в каждой строке данных.
* При вычислениях необходимо учитывать средний размер хэш-ключа, а не средний размер ключа кластера. Принимается во внимание, сколько ключей кластера попадают на каждое значение хэш-ключа.
* ORACLE гарантирует, что начальное распределение памяти достаточно для размещения хэш-таблицы, согласно значениям SIZE и HASHKEYS. Если значений параметров памяти (INITIAL, NEXT и MINEXTENTS) не хватает для размера хэш-таблицы, будут распределены дополнительные (инкрементальные) экстенты, пока не будет распределено как минимум SIZE*HASKEYS памяти.

Общая схема расчета памяти для кластеров

1. Вычислить общий размер заголовка блока.
2. Вычислить размер свободной памяти в блоке данных (зависит от параметра PCTFREE).
3. Вычислить сумму длин столбцов для средней строки на ключ кластера (для индексированного кластера каждое значение кластерного ключа хранится 1 раз, для хеш-кластера – на каждую строку своё).
4. Вычислить общий размер средней строки для всех кластеризуемых таблиц.
5. Вычислить средний размер блока кластера (зависит от среднего числа строк (по всем таблицам) на ключ кластера).
6. Вычислить общее число блоков, требуемое для кластера (определяется отношением количества разных ключей кластера к количеству ключей на блок).

Замечания к расчету памяти для кластеров

При расчете надо иметь в виду следующие факторы, которые могут повлиять на точность расчетов:

* Память, используемая для записей транзакций и для удаленных строк, не становится доступной немедленно, из-за отсроченной процедуры очистки.

* Хвостовые пустые значения и их байты длины не хранятся.

* Вставки, обновления и удаления строк, а также столбцы, превышающие размер блока данных, могут привести к фрагментации и цепочкам кусков строк. Поэтому последующие оценки имеют склонность к занижению по сравнению с реальными величинами, если фрагментация существенна.

Заключение

Рассмотренные способы размещения и доступа к данным прозрачны для пользователей и приложений. То есть кластеризация, хеширование и индексирование оказывают влияние на время обработки данных, но не требуют изменения программ и запросов.

Информация о методах размещения данных и методах доступа к данным хранится в словаре-справочнике данных и используется системой при выполнении запросов.

Для кластеризованных и хешированных таблиц можно строить дополнительные индексы по полям, не входящим в кластерный ключ и не являющимся ключом хеширования. Это также относится к преимуществам кластеризации и хеширования и позволяет устранить некоторые присущие им недостатки.