# Лекция «Таблицы»

## Типы таблиц

Прежде чем перейти к деталям, определим каждый тип таблиц. В Oracle — семь основных типов таблиц.

• Таблицы, организованные в виде кучи. Это "обычные", стандартные таблицы базы данных. Данные управляются по принципу "кучи". При добавлении данных используется первое же свободное место в сегменте, достаточное для их размещения. При удалении данных из таблицы освободившееся место может повторно использоваться следующими операторами INSERT и UPDATE. Вот откуда название "организованные в виде кучи" для таких таблиц. "Куча" — это пространство памяти, используемое достаточно случайным образом.

• Таблицы, организованные по индексу. Такая таблица хранится в структуре индекса. Это предполагает физическое упорядочение ее строк. Если в обычной таблице данные вставляются в любое свободное место, в таблице, организованной по индексу, хранимые данные отсортированы по первичному ключу.

• Таблицы в кластере. Хранение таблицы в кластере дает два преимущества. Во первых, несколько таблиц можно хранить вместе. Обычно в блоке хранятся данные только одной таблицы. В кластере же в одном блоке могут храниться данные нескольких таблиц. Во-вторых, все данные, содержащие одно и то же значение ключа кластера, также хранятся вместе. Данные "кластеризованы" вокруг значения ключа кластера. Ключ кластера строится с помощью индекса на основе В\*-дерева.

• Таблицы в хеш-кластере. Аналогичны представленным ранее таблицам в кластере, но, вместо индекса на основе В\*-дерева, для поиска блока данных по ключу кластера используется хеширование ключа. В хеш-кластере сами данные (образно говоря) и есть индекс. Это хорошо подходит для чтения данных, соответствующих определенному значению ключа.

• Вложенные таблицы. Часть объектно-реляционных расширений сервера Oracle. Это просто генерируемые и поддерживаемые системой дочерние таблицы связанные как предок-потомок. Они устроены аналогично таблицам ЕМР и DEPT в схеме SCOTT. Таблицу ЕМР можно считать дочерней по отношению к таблице DEPT, поскольку в таблице ЕМР есть внешний ключ, DEPTNO, ссылающийся на таблицу DEPT. Главное различие в том, что это — не "отдельные" таблицы, как ЕМР.

• Временные таблицы. В этих таблицах сохраняются черновые данные на время транзакции или сеанса. При необходимости для этих таблиц выделяются временные экстенты из временного табличного пространства пользователя. Каждый сеанс будет "видеть" только выделенные им самим экстенты и никогда не "увидит" данные, созданные в других сеансах.

• Объектные таблицы. Это таблицы, создаваемые на основе объектного типа. Они имеют специальные атрибуты, отсутствующие у необъектных таблиц, например генерируемый системой псевдостолбец REF (идентификатор объекта) для каждой строки. Объектные таблицы фактически являются отдельным случаем обычных, организованных по индексу и временных таблиц, и могут включать вложенные таблицы.

Имеется несколько общих свойств таблиц, не зависящих от их типа.

• Таблица может иметь до 1000 столбцов. Таблицы наиболее эффективно работают при количестве столбцов, намного меньшем, чем 1000.

• Таблица может иметь практически неограниченное количество строк. Хотя при этом придется столкнуться с другими ограничениями. Например, табличное пространство может обычно состоять не более чем из 1022 файлов. Пусть используются файлы размером 32 Гбайт, тогда получаем 32704 Гбайт в каждом табличном пространстве. Это означает — 2143289344 блока по 16 Кбайт каждый. В один такой блок можно вместить 160 строк размером от 80 до 100 байт. Это дает в итоге 342926295040 строк. Однако если фрагментировать таблицу, это количество можно увеличить в десять раз. Теоретически ограничения, конечно, есть, но прежде чем они будут достигнуты, придется столкнуться с другими, практическими ограничениями.

• Таблица может иметь столько индексов, сколько имеется перестановок столбцов (и перестановок функций от этих столбцов), но не более 32 столбцов, хотя и в этом случае будут практические ограничения на количество реально создаваемых и сопровождаемых индексов.

• Нет ограничения на количество таблиц. И в этом случае практические ограничения будут держать количество таблиц в разумных границах. Миллионов таблиц у вас не будет (такое количество сложно создать и поддерживать), но тысячи таблиц поддерживаются элементарно.

### Терминология

Цель этого раздела — представить термины и дать их определения. При необходимости, более детальная информация об использовании конкретных параметров будет представлена в следующих разделах. Ниже приведен скрипт на создание простой таблицы **t**

CREATE TABLE "TEST\_1"."T"

("X" NUMBER(\*,0)

) SEGMENT CREATION IMMEDIATE

PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING

STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645

PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER\_POOL DEFAULT FLASH\_CACHE DEFAULT CELL\_FLASH\_CACHE DEFAULT)

TABLESPACE "USERS";

В этой SQL-команде, кроме стандартных фраз CREATE TABLE TEST\_1.T (X NUMBER(\*,0)), приведены дополнительные. Рассмотрим назначение дополнительных параметров.

#### Отметка максимального уровня

Этот термин (high water mark) используется для объектов, хранящихся в базе данных. Если представить себе таблицу как "линейную" структуру в виде последовательности блоков слева направо, то отметкой максимального уровня будет крайний правый блок, когда-либо содержавший данные. Например:



Этот рисунок показывает, что отметка максимального уровня сначала указывает на первый блок только что созданной таблицы. По мере добавления данных в таблицу и использования новых блоков отметка максимального уровня повышается. Если удалить некоторые (или даже все) строки таблицы, может появиться много блоков, уже не содержащих данные, но они все равно находятся до отметки максимального уровня и останутся там, пока объект не будет пересоздан или усечен.

Отметка максимального уровня важна, поскольку сервер Oracle при полном просмотре будет сканировать все блоки до этой отметки, даже если они не содержат данных. Это скажется на скорости выполнения полного просмотра, особенно если большая часть блоков до отметки максимального уровня — пустые. Чтобы убедиться в этом, просто создайте таблицу из 1000000 (или другого большого количества) строк. Выполните запрос SELECT COUNT(\*) к этой таблице. Теперь удалите все строки таблицы и убедитесь, что запрос SELECT COUNT(\*), не возвращающий строк, работает так же долго, как и в случае возвращения 1000000 строк. Так происходит потому, что сервер Oracle просматривает все блоки до отметки максимального уровня в поисках данных. Сравните это с результатом применения оператора TRUNCATE. При выполнении TRUNCATE отметка максимального уровня будет сброшена "в ноль". Если предполагается удаление всех строк таблицы, именно по этой причине надо использовать оператор TRUNCATE.

#### Списки свободных мест

В списке свободных мест (FREELIST) сервер Oracle отслеживает блоки объекта до отметки максимального уровня, в которых есть свободное пространство. С каждым объектом будет связан хотя бы один список свободных мест, и по мере использования блоков они будут при необходимости добавляться или удаляться из этого списка. Важно отметить, что в списке свободных мест будут только блоки, находящиеся до отметки максимального уровня. Блоки за отметкой максимального уровня будут использоваться, только когда списки свободных мест пусты, — в этот момент сервер Oracle переносит отметку максимального уровня дальше и добавляет соответствующие блоки в список свободных мест. Таким образом, сервер Oracle откладывает увеличение отметки максимального уровня для объекта, пока это не станет действительно необходимым. У объекта может быть несколько списков свободных мест. Если предполагается выполнение значительного количества операций INSERT или UPDATE для объекта множеством одновременно работающих пользователей, создание нескольких списков свободных мест может существенно повысить производительность (за счет возможного использования дополнительной памяти). Как будет показано далее, наличие достаточного количества списков свободных мест принципиально важно.

В среде с множеством одновременных вставок или изменений данных списки свободных мест могут как снижать так и повышать производительность. Предельно простой тест может показать преимущества установки подходящего количества списков свободных мест. Берем простейшую таблицу:

create table t (x int) ;

и с помощью двух сеансов начинаем интенсивно вставлять в нее данные. Если измерить общесистемное количество ожиданий событий, связанных с блоками, до и после этой операции, можно выявить продолжительные ожидания доступности блоков данных (при вставке данных). Это зачастую вызвано недостаточным количеством списков свободных мест в таблицах.

Итак, создадим временную таблицу:

create global temporary table waitstat\_before

on commit preserve rows as select \* from v$waitstat

where 1=0

Table created.

для хранения исходного количества ожиданий блоков. Затем в двух сеансах одновременно выполним:

truncate table waitstat\_before;

Table truncated.

insert into waitstat\_before select \* from v$waitstat

14 rows created.

begin

for i in 1. . 100000

loop

insert into t values (i) ;

commit;

end loop;

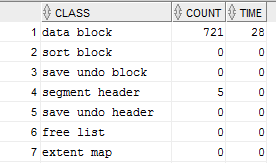
end;

PL/SQL procedure successfully completed.

Это — очень простой блок кода, и мы — единственные пользователи базы данных в этот момент. Производительность при этом должна быть максимальной. У меня создан большой буферный кэш, журналы повторного выполнения имеют нужный размер, индексы не замедляют работу; все должно работать быстро. В результате, однако, я получаю следующее:

select a.class, b.count-a.count count, b.time-a.time time

from waitstat\_before a, v$waitstat b where a.class = b.class



Время ожидания составило порядка 20-22 сек.

Это целиком следствие недостаточного количества списков свободных мест для одновременного выполнения подобного рода действий. Можно легко вообще избавиться от этого ожидания, просто создав таблицу с несколькими списками свободных мест:

create table t (x int) storage (FREELISTS 2);

Table created,

или изменив уже существующий объект:

alter table t storage (FREELISTS 2);

Table altered.

Время ожидания сократилось. Для таблицы необходимо попытаться определить максимальное количество одновременных вставок или изменений, для выполнения которых потребуется выделение дополнительного пространства. Под одновременностью я имею в виду ситуации, когда несколько сеансов в один и тот же момент будут запрашивать свободный блок для соответствующей таблицы. Речь не идет о количестве одновременно выполняющихся транзакций — нас интересует, сколько сеансов будет одновременно выполнять вставку, независимо от границ транзакций. Для достижения максимального параллелизма необходимо как минимум столько списков свободных мест, сколько вставок одновременно выполняется в таблицу.

Значит, задав большое значение параметра FREELIST, можно забыть про все эти проблемы, верно? Нет, — это было бы слишком просто. Каждый процесс будет использовать один список свободных мест. Он не будет при поиске свободного места переходить от одного списка к другому. Это означает, что если в списке, используемом процессом, свободного места не осталось, он не будет искать свободное место в другом списке. И хотя в таблице имеется десять списков свободных мест, будет передвинута отметка максимального уровня для таблицы. Если же передвигать отметку некуда (все пространство использовано), будет выделен новый экстент. Затем процесс продолжит использовать пространство только в своем списке свободных мест (который сейчас пуст). При использовании нескольких списков свободных мест приходится идти на компромисс. С одной стороны, несколько списков свободных мест позволяют существенно повысить производительность. С другой стороны, таблица будет занимать больше пространства на диске, чем необходимо. Придется решать, что важнее. Не стоит недооценивать полезность этого параметра, особенно с учетом возможности изменять его при желании в версиях, начиная с Oracle 8.1.Можно увеличить его значение для параллельной загрузки данных в обычном режиме с помощью утилиты SQLLDR. При этом будет достигнута высокая степень параллелизма загрузки с минимальными ожиданиями. После загрузки можно вернуть прежнее, более подходящее для обычной работы, значение параметра FREELIST. При этом блоки из нескольких существующих списков свободных мест будут объединены в один основной список свободных мест.

#### Параметры PCTFREE и PCTUSED

Эти два параметра управляют добавлением и удалением блоков из списков свободных мест. При задании для таблицы (но не для таблицы, организованной по индексу, как будет показано далее) параметр PCTFREE сообщает серверу Oracle, сколько места должно быть зарезервировано в блоке для будущих изменений. Стандартное значение — 10 процентов. Предположим, используются блоки размером 8 Кбайт. Когда при добавлении новой строки в блок свободного места в блоке останется меньше 800 байт, сервер Oracle будет использовать новый блок вместо существующего. Эти 10 процентов пространства данных в блоке оставляются для изменений строк блока. Если строку придется изменять, в блоке будет место для размещения измененной строки.

Итак, если параметр PCTFREE определяет, когда сервер Oracle убирает блок из списка свободных мест, чтобы в него больше не вставлялись строки, то параметр PCTUSED определяет, когда сервер Oracle снова вернет блок в список свободных мест. Если параметр PCTUSED установлен равным 40 процентам (стандартное значение) и в блоке достигнут уровень заполнения PCTFREE (блок уже не находится в списке свободных мест), то сервер Oracle вернет его в список, только когда в блоке станет свободным **61** процент пространства. При использовании стандартных значений параметров PCTFREE (10) и PCTUSED (40) блок будет оставаться в списке свободных мест, пока не заполнится на 90 процентов (в нем останется 10 процентов свободного пространства). Как только он заполнится на 90 процентов, то будет удален из списка свободных мест и не попадет в него, пока свободное пространство не составит 60 процентов блока.

Параметры PCTFREE и PCTUSED реализуются по-разному для разных типов таблиц. Для некоторых типов таблиц используются оба параметра, для других — только PCTFREE, да и то лишь при создании объекта. Есть три значения параметра PCTFREE: слишком большое, слишком маленькое и подходящее. Если установить слишком большое значение параметра PCTFREE для блоков, будет напрасно расходоваться пространство на диске. Если установить значение PCTFREE равным 50 процентам, а данные никогда не изменяются, то 50 процентов каждого блока просто пустуют. В другой таблице, однако, значение 50 процентов может быть вполне уместным. Если первоначально маленькие строки со временем увеличиваются примерно вдвое, большое значение параметра PCTFREE позволит избежать переноса строк.

#### Перенос строки

Итак, когда переносится строка? Строка переносится из блока, в котором она была создана, когда она выросла настолько, что уже не помещается в блоке с остальными строками. Перенос строки проиллюстрирован ниже. Все начинается с блока, который выглядит примерно так:



Примерно одна седьмая блока — свободное место. Однако необходимо более чем в два раза увеличить занимаемое строкой 4 место с помощью оператора UPDATE (сейчас эта строка занимает одну седьмую часть блока). В данном случае, даже если сервер Oracle объединит все свободное пространство в блоке в один фрагмент следующим образом:



все равно для увеличения строки 4 более чем в два раза места не хватит, потому что общее свободное пространство меньше, чем текущий размер строки 4. Если бы строка могла поместиться в свободное пространство после его объединения, то это объединение было бы выполнено. В нашем случае, однако, сервер Oracle не будет выполнять объединение, и блок останется неизменным. Поскольку строка 4 должна выйти за границы блока, сервер Oracle перенесет ее. Но она, однако, не будет просто перемещена; ее не обходимо оставить как "*адрес для пересылки*". *Могут существовать индексы, физически ссылающиеся на текущий адрес строки 4*. При простом изменении эти индексы не будут обновляться. Поэтому когда сервер Oracle переносит строку, он оставляет по старому адресу указатель на ее реальное местонахождение. После изменения блоки могут выглядеть примерно так:



Итак, вот что такое перенесенная строка: это строка, которую пришлось перенести из блока, в который она была вставлена, в какой-то другой блок. Почему это надо учитывать? Приложение никогда о переносе строки не "узнает", SQL-операторы менять не придется. Это влияет только на производительность. Если придется читать эту строку через индекс, он будет ссылаться на исходный блок. Этот блок, в свою очередь, будет ссылаться на новый блок. Вместо выполнения, скажем, двух операций ввода/вывода для чтения индекса и одной для чтения таблицы, для получения фактических данных строки потребуется еще одна дополнительная операция ввода/вывода блока. Сама по себе это "небольшая" проблема; вы эту дополнительную операцию даже не заметите. Однако, если в этом состоянии окажется значительная часть строк и обращаться к таблице будет множество пользователей, этот побочный эффект будет заметен. Доступ к данным начнет замедляться (дополнительная операция ввода/вывода увеличивает время доступа), эффективность буферного кэша — снижаться (придется буферизовать в два раза больше блоков, чем при отсутствии переноса строк), а размер и сложность таблицы — расти. Именно по этим причинам переноса строк следует избегать. Интересно разобраться, что сервер Oracle будет делать, если перенесенную с левого блока (на представленной ранее схеме) в правый блок строку снова придется переносить. Это может понадобиться, поскольку в блок, куда она была перенесена, добавлены строки, а наша строка опять увеличивается при изменении. Сервер Oracle перенесет строку назад в исходный блок и, если в нем достаточно места, оставит ее там (строка становится "неперенесенной"). Если места в исходном блоке недостаточно, сервер Oracle перенесет всю строку в другой блок и изменит адрес перенаправления в исходном блоке. Таким образом, *перенос строк всегда создает один уровень косвенной ссылки*. Итак, теперь мы можем вернуться к параметру PCTFREE и его использованию; при правильной установке значения этого параметра можно свести перенос строк к минимуму.

#### Параметры INITIAL, NEXT и PCTINCREASE

Эти параметры хранения определяют размер начального (INITIAL) и последующих экстентов, выделяемых таблице, а также процент увеличения для следующих экстентов. Например, если используется начальный экстент размером 1 Мбайт, следующий — размером 2 Мбайта, а параметр PCTINCREASE имеет значение 50, в таблице будут следующие экстенты:

1. 1 Мбайт.

2. 2 Мбайт.

3. 3 Мбайт (150 процентов от 2).

4. 4,5 Мбайт (150 процентов от 3).

и т.д. В базе данных необходимо использовать только локально управляемые табличные пространства с одинаковыми экстентами. При этом размер первого экстента всегда совпадает с размером следующего, а параметр PCTINCREASE вообще не имеет смысла, — он лишь вызывает фрагментацию табличного пространства. Если локально управляемые табличные пространства не используются, рекомендуется всегда устанавливать INITIAL = NEXT и PCTINCREASE = 0. Это позволяет эмулировать процесс выделения пространства, происходящий в локально управляемых табличных пространствах. Для всех объектов в табличном пространстве должна использоваться одинаковая стратегия выделения экстентов во избежание фрагментации.

#### Параметры MINEXTENTS и MAXEXTENTS

Эти параметры управляют количеством экстентов, которые могут быть выделены объекту. Значение MINEXTENTS указывает серверу Oracle, сколько экстентов выделять таблице первоначально. Например, в локально управляемом табличном пространстве с одинаковыми экстентами размером 1 Мбайт при установке параметру MINEXTENTS значения 10 таблице будет выделено 10 Мбайт дискового пространства. Параметр MAXEXTENTS задает верхний предел для количества экстентов, которые могут быть выделены объекту. Если в том же табличном пространстве задать параметру MAXEXTENTS значение 255, таблица сможет иметь размер не более 255 Мбайт. Конечно, если в табличном пространстве не хватит места для обеспечения такого роста, выделить соответствующие экстенты таблице тоже не удастся.

#### Параметры LOGGING и NOLOGGING

Обычно объекты создаются с журнализацией (LOGGING), тем самым все действия с объектом, которые могут генерировать информацию повторного выполнения, ее генерируют. Параметр NOLOGGING позволяет выполнять с объектом ряд действий, не генерируя данных повторного выполнения. Параметр NOLOGGING затрагивает лишь несколько специфических действий: первоначальное создание объекта, непосредственная загрузка данных с помощью утилиты SQLLDR или пересоздание (подробнее о том, какие из этих действий применимы к тому или иному объекту базы данных, см. в справочном руководстве по языку SQL — "SQL Language Reference Manual"). Этот параметр не отключает генерирование данных повторного выполнения для объекта вообще, а только для некоторых специфических действий. Например, если таблица создается как SELECT NOLOGGING, а затем выполняется оператор INSERT INTO THAT\_TABLE VALUES (1), этот оператор будет зарегистрирован в журнале, а создание таблицы — нет.

#### Параметры INITRANS and MAXTRANS

Каждый блок данных объекта включает заголовок. Часть этого заголовка — таблица транзакций. В таблицу транзакций вносятся записи о том, какие строки/элементы блока заблокированы какими транзакциями. Первоначальный размер таблицы транзакций определяется параметром INITRANS объекта. Для таблиц стандартное значение — 1 (для индексов — 2). При необходимости таблица транзакций может динамически расти вплоть до MAXTRANS записей (естественно, при наличии достаточного свободного пространства в блоке). **Каждая запись транзакции занимает 23 байта в заголовке блока.**

### Таблицы, организованные в виде кучи

Таблицы, организованные в виде кучи, используются приложениями в 99 (если не более) процентах случаев, хотя со временем это может измениться за счет более интенсивного использования таблиц, организованных по индексу, — ведь по таким таблицам теперь тоже можно создавать дополнительные индексы. Таблица, организованная в виде кучи, создается по умолчанию при выполнении оператора CREATE TABLE. Если необходимо создать таблицу другого типа, это надо явно указать в операторе CREATE.

"Куча" — классическая структура данных, изучаемая в курсах программирования. Это, по сути, большая область пространства на диске или в памяти (в случае таблицы базы данных, конечно же, на диске), используемая произвольным образом. Данные размещаются там, где для них найдется место, а не в определенном порядке. Многие полагают, что данные будут получены из таблицы в том же порядке, в каком туда записывались, но при организации в виде кучи это не гарантировано. Фактически гарантировано как раз обратное: строки будут возвращаться в абсолютно непредсказуемом порядке. Это очень легко продемонстрировать. Создадим такую таблицу, чтобы в моей базе данных в блоке помещалась одна полная строка (я использую блоки размером 8 Кбайт). Совсем не обязательно создавать пример с одной строкой в блоке. Я просто хочу продемонстрировать предсказуемую последовательность событий. Такое поведение будет наблюдаться для таблиц любых размеров и в базах данных с любым размером блока:

Create table t (a int, b varchar2(4000) default rpad('\*',4000,'\*'), с varchar2(3000) default rpad('\*',3000,'\*') );

insert into t (a) values (1);

insert into t (a) values (2);

insert into t (a) values (3);

delete from t where a = 2 ;

insert into t (a) values (4);

select a from t;

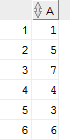


При полном просмотре данные выдаются в том порядке, в котором обнаруживаются, а не в порядке вставки. Это принципиально важное свойство обычных таблиц базы данных; в общем случае они представляют собой неупорядоченные наборы данных. Учтите, что для получения подобного результата необязательно использовать оператор DELETE — тех же результатов можно достичь с помощью только операторов INSERT. Если вставить маленькую строку, потом — очень большую, которая не поместится в один блок с маленькой строкой, а затем — опять маленькую строку, при выборке эти строки вполне могут быть получены в порядке "маленькая, маленькая, большая". Они не будут извлекаться в порядке вставки. Сервер Oracle размещает данные там, где они помещаются, а не в порядке выполнения транзакций.

insert into t (a,b,с) values (5,'b','c');

insert into t (a) values (6);

insert into t (a,b,с) values (7,'b','c');



Если в результате запроса данные должны выдаваться в порядке вставки, придется добавить столбец, который будет использоваться для упорядочения данных при извлечении. Этот столбец может быть числовым, например, с последовательно увеличивающимися значениями (которые генерируются с помощью объекта SEQUENCE базы данных Oracle). Затем можно будет примерно воспроизвести последовательность вставки, упорядочивая данные по этому столбцу. Это будет лишь приближение, поскольку строка с последовательным номером 55 вполне могла быть зафиксирована в базе данных до строки с последовательным номером 54, поэтому официально она была в базе данных "первой". Поэтому таблицу, организованную в виде кучи, можно рассматривать просто как большой неупорядоченный набор строк. Эти строки будут выдаваться во внешне случайном порядке, зависящем от используемых опций (параллельные запросы, различные режимы оптимизации и т.д.), причем могут выдаваться в разном порядке даже для одного и того же запроса. Никогда не полагайтесь на порядок строк в результатах запроса, если только запрос не включает конструкцию ORDER BY!

• FREELISTS. Для каждой таблицы, выделенные из кучи, блоки отслеживаются в списке свободных, FREELIST. Таблица может иметь несколько списков FREELIST. Если предполагается интенсивная вставка данных в таблицу большим количеством пользователей, создание нескольких списков FREELIST может существенно повысить производительность (за счет использования дополнительного пространства). Влияние значения этого параметра на производительность рассматривалось ранее (в разделе "Списки свободных мест").

• PCTFREE. Степень заполнения блока в процессе вставки строк. Как только в блоке осталось менее чем PCTFREE процентов свободного пространства, он уже не просматривается при вставке новых строк. Этот параметр позволяет контролировать перенос строк при последующих изменениях и устанавливается в зависимости от предполагаемого использования таблицы.

• PCTUSED. Указывает, насколько пустым должен стать блок, чтобы в него можно было вставлять строки. Новые строки будут вставляться в блок, только если он занят менее чем на PCTUSED процентов. Как и в случае параметра PCTFREE, для правильной установки значения этого параметра необходимо учитывать предполагаемое использование таблицы.

• INITRANS. Количество записей в таблице транзакций, первоначально выделяемое в блоке. При слишком низком значении (стандартное значение — 1) могут возникнуть проблемы при одновременном доступе большого количества пользователей. Если блок базы данных почти заполнен и таблицу транзакций нельзя динамически увеличить, сеансы будут ожидать в очереди доступа к блоку, поскольку для каждой из одновременно выполняемых транзакций необходима запись в таблице транзакций. Если предполагается одновременное выполнение большого количества изменений одних и тех же блоков, имеет смысл увеличить значение этого параметра.

**Примечание:** для данных больших объектов, хранящихся в отдельном сегменте вне строки, не используются значения параметров PCTFREE/PCTUSED, установленные для таблицы. Эти блоки больших объектов управляются по-другому. Они всегда заполняются полностью и возвращаются в список свободных, когда полностью пусты. Именно на эти параметры надо обращать особое внимание. Я обнаружил, что остальные параметры хранения теперь просто не слишком важны. Как я уже упоминал ранее в этой главе, необходимо использовать локально управляемые табличные пространства, а для них параметры PCTINCREASE, NEXT и т.п. не используются.

### Таблицы, организованные по индексу

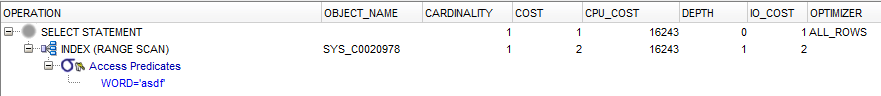
Таблицы, организованные по индексу (index organized tables — IOTs), — это таблицы, хранящиеся в структуре индекса. Таблица, хранящаяся в куче, организована случайным образом (данные попадают в любое свободное место). В таблице же, организованной по индексу, хранимые данные отсортированы по первичному ключу. С точки зрения приложений, таблицы, организованные по индексу, ничем не отличаются: к ним применяются такие же SQL-операторы, как и для доступа к обычной таблице. Они особенно полезны для информационно-поисковых (information retrieval — IR) систем, хранения пространственных данных и приложений оперативного анализа информации (OLAP).

Зачем организовывать таблицу по индексу? Можно задать и прямо противоположный вопрос; зачем организовывать таблицу в виде кучи? Поскольку все таблицы в реляционной базе данных в любом случае должны иметь первичный ключ, не будет ли организация таблицы в виде кучи непозволительной тратой пространства? При использовании таблицы, организованной в виде кучи, необходимо управлять дисковым пространством как для таблицы, так и индекса по первичному ключу. При использовании таблицы, организованной по индексу, дополнительное пространство для поддержки индекса по первичному ключу не требуется, поскольку индекс — это данные, а данные — это индекс. Проблема в том, что индекс — сложная структура данных, поддержка и управление которой требуют выполнения множества действий. Кучей же управлять сравнительно легко. В некоторых аспектах таблица, организованная в виде кучи, эффективнее таблицы, организованной по индексу. Тем не менее, у таблиц, организованных по индексу, есть ряд преимуществ. Например, есть таблица документов. Необходимо анализировать документы и находить в них слова. Для этого была создана таблица следующего вида:

create table keywords (word varchar2(50) , position int, doc\_id int, primary key(word,position,doс\_id));

Эта таблица состояла исключительно из столбцов первичного ключа. При этом расходовалось двойное количество пространства; размеры таблицы и индекса по первичному ключу были сопоставимы (на самом деле индекс по первичному ключу был больше, поскольку в нем хранился идентификатор строки, на которую указывал ключ, а в таблице идентификатор строки отдельно не хранится — он определяется). Применяются к этой таблице SQL-операторы только с конструкцией WHERE, задающей условие для столбца WORD или столбцов WORD и POSITION. В этом случае никогда не используется таблицу, а только индекс таблицы. На таблицу только попусту расходовалось дисковое пространство. Такие таблицы необходимо организовывать по индексу.

select \* from keywords where word='asdf';



Отличие таблицы индекс — организованной от индекса.



Наконец, если необходимо обеспечить совместное размещение данных или физически хранить данные в определенном порядке, индексная организация таблицы тоже подойдет. Пользователи СУБД Sybase и SQL Server в этом случае использовали бы кластерный индекс, но организация таблицы по индексу намного лучше. В случае кластерного индекса в этих СУБД может дополнительно расходоваться до 110 процентов пространства (аналогично моей таблице KEYWORDS в представленном ранее примере). Здесь же дополнительных расходов вообще нет, поскольку данные хранятся только в одном месте. Классический пример желательности совместного размещения взаимосвязанных данных представляет собой отношение главный/подчиненный. Пусть у таблицы ЕМР имеется подчиненная таблица:

Create table addresses (empno number(4) references emp(empno) on delete cascade, addr\_type varchar2(10) , street varchar2(20) , city varchar2(20) , state varchar2(2), zip number, primary key (empno,addr\_type)) ORGANIZATION INDEX

Физически близкое размещение всех адресов сотрудника (домашнего адреса, адреса места работы, адреса школы, прежнего адреса и т.д.) уменьшит объем ввода/вывода при соединении таблиц ЕМР и ADDRESSES. Объем логического ввода/вывода будет тем же, но физического ввода/вывода потребуется существенно меньше. В обычной таблице каждый адрес сотрудника может физически находиться в другом блоке по отношению к другим адресам того же сотрудника. Сортируя адреса по столбцам EMPNO и ADDR\_TYPE, мы гарантируем, что все адреса данного сотрудника хранятся "рядом".

Это применимо и в случае частого использования запросов с конструкцией BETWEEN по первичному или уникальному ключу. Хранение отсортированных данных повысит производительность и этих запросов.

Например, в базе данных имеется таблица котировок акций. Каждый день в нее собирается информация о символах акций, дате, конечной цене, колебаниях курса в течение дня, количестве проданных акций и т.д. Это делается для акций сотен компаний. Соответствующая таблица имеет вид:

Create table stocks (ticker varchar2(10), day date, value number, change number, high number, low number, vol number, primary key(ticker,day) ) organization index;

Предположим, что часто просматриваются котировки акций одной компании, за некоторый период (вычисляя скользящее среднее, например). При использовании таблицы, организованной в виде кучи, вероятность того, что две строки для акций ORCL окажутся в одном блоке базы данных, практически равна нулю. Дело в том, что каждую ночь вставляются записи за день для всех акций. При этом заполняется как минимум один блок базы данных (обычно — много блоков). Поэтому каждый день добавляется новая запись для акций ORCL, но она попадает в блок, не совпадающий с другими, содержащими записи для ORCL, блоками таблицы. Если выполнить запрос:

Select \* from stocks where ticker = 'ORCL' and day between sysdate and sysdate - 100;

сервер Oracle прочитает индекс, а затем обратится к таблице за остальными данными строки по идентификатору строки. Каждая из 100 выбираемых строк окажется в другом блоке базы данных из-за принятого способа загрузки данных в таблицу, и каждая, вероятно, потребует выполнения операции физического ввода/вывода. Теперь предположим, что такая таблица организована по индексу. При выполнении этого же запроса придется прочитать только соответствующие блоки индекса, и все необходимые данные будут получены. Не только нет необходимости обращаться к таблице, но и все строки для акций ORCL в заданном диапазоне дат физически хранятся "рядом" друг с другом. Требуется меньше логического и физического ввода/вывода.

Рассмотрим с три простейшие разновидности таблицы, организованной по индексу:

Create table t1 (x int primary key, у varchar2 (25), z date ) organization index;

Create table t2 (x int primary key, у varchar2 (25), z date ) organization index OVERFLOW;

Create table t3 (x int primary key, у varchar2(25), z date ) organization index overflow INCLUDING y;

Рассмотрим полный текст SQL-оператора, необходимого для создания первой из представленных выше таблиц:

CREATE TABLE "SCOTT"."T1"

( "X" NUMBER(\*,0),

"У" VARCHAR2(25 BYTE),

"Z" DATE,

PRIMARY KEY ("X") ENABLE

) ORGANIZATION INDEX **NOCOMPRESS PCTFREE** 10 INITRANS 2 MAXTRANS 255 LOGGING

STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645

PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER\_POOL DEFAULT FLASH\_CACHE DEFAULT CELL\_FLASH\_CACHE DEFAULT)

TABLESPACE "SYSTEM"

**PCTTHRESHOLD** 50;

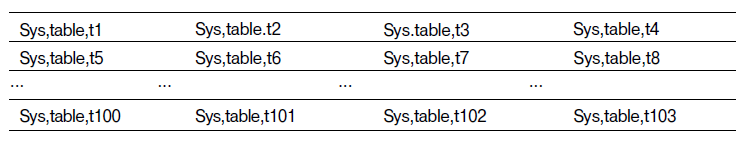
В код создания таблицы добавились две новые опции — NOCOMPRESS и PCTTHRESHOLD. В тексте оператора CREATE TABLE чего-то не хватает; конструкции PCTUSED нет, a PCTFREE — есть. Это потому, что индекс — сложная структура данных, организованная не случайным образом, как куча, а так, чтобы данные попали в определенное место. В отличие от кучи, где блоки доступны для вставки время от времени, блоки индекса всегда доступны для вставки новых записей. Если данные принадлежат определенному блоку в соответствии со значением ключа, они попадут в него независимо от степени его заполнения. Кроме того, параметр PCTFREE используется только при создании объекта и наполнении данными индексной структуры. В таблицах, организованных в виде кучи, он используется иначе.

Параметр PCTFREE резервирует пространство во вновь созданном индексе, но не для последующих операций с ним (во многом, по той же причине, почему вообще не используется параметр PCTUSED). Все соображения относительно списков свободных блоков FREELIST, высказанные для таблиц, организованных в виде кучи, относятся и к таблицам, организованным по индексу.

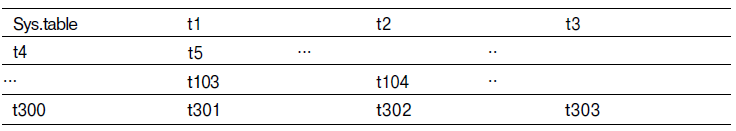
Опция NOCOMPRESS. Эта опция используется для индексов. Она требует, чтобы сервер Oracle хранил все значения в записи индекса (не сжимал ее). Если первичный ключ объекта создан по столбцам А, В и С, будут храниться все комбинации А, В и С. Противоположностью NOCOMPRESS является опция COMPRESS N, где N — целое число, задающее количество сжимаемых столбцов. В результате удаляются повторяющиеся значения; они факторизуются на уровне блоков, так что повторяющиеся значения столбца А и, возможно, В, больше не хранятся. Рассмотрим пример таблицы:

Create table iot (owner, object\_type, object\_name, primary key(owner,object\_type,object\_name)) organization index NOCOMPRESS as select owner, object\_type, object\_name from all\_objects;

Значение в столбце OWNER повторяется много сотен раз. Каждой схеме (OWNER) обычно принадлежит множество объектов. Даже пара значений OWNER, OBJECT\_TYPE повторяется многократно; в схеме имеются десятки таблиц, десятки пакетов и т.д. Не повторяются только все три столбца вместе. Можно попросить сервер Oracle убрать повторяющиеся значения. Вместо создания индексного блока со значениями:



можно использовать конструкцию COMPRESS 2 (факторизовать первых два столбца) и получить блок с такими данными:



Т.е. значения SYS и TABLE сохраняются лишь один раз, а затем — только значения третьего столбца. При этом в блок индекса может поместиться намного больше записей. При этом ни степень параллелизма, ни функциональные возможности никак не ограничиваются. Требуется чуть больше процессорного времени, поскольку сервер Oracle выполняет дополнительные действия, чтобы собрать значения ключей. Однако при этом существенно сокращается объем данных при вводе/выводе, а в буферном кэше помещается больше данных, поскольку их больше помещается в блоке. Это очень хороший компромисс. Продемонстрируем экономию с помощью простого тестового примера, в котором создадим представленную ранее таблицу (CREATE TABLE AS SELECT) с параметрами NOCOMPRESS, COMPRESS 1 и COMPRESS 2.

Create table iot (owner, object\_type, object\_name, constraint iot\_pk primary key(owner,object\_type,object\_name)) organization index NOCOMPRESS as select distinct owner, object\_type, object\_name from all\_objects;

select \* from iot;

Выполним анализ структуры индекс-таблицы

analyze index iot\_pk validate structure;

и получим результат из системной таблицы index\_stats.

select lf\_blks, br\_blks, used\_space,

opt\_cmpr\_count, opt\_cmpr\_pctsave

from index\_stats;



В этом индексе используется 72 листовых блока и 1 блок ветвления.

analyze index iot\_pk validate structure;

Используемое пространство порядка 500 Кбай. Столбец OPT\_CMPR\_COUNT содержит

оптимальное значение сжатия.

Добавим опцию **compress 1** и выполним анализ полученной индекс—таблицы.

Create table iot1 (owner, object\_type, object\_name, constraint iot\_pk1 primary key(owner,object\_type,object\_name)) organization index COMPRESS 1 as select distinct owner, object\_type, object\_name from all\_objects;

analyze index iot\_pk1 validate structure;

select lf\_blks, br\_blks, used\_space,

opt\_cmpr\_count, opt\_cmpr\_pctsave

from index\_stats



Create table iot2 (owner, object\_type, object\_name, constraint iot\_pk2 primary key(owner,object\_type,object\_name)) organization index COMPRESS 2 as select distinct owner, object\_type, object\_name from all\_objects;

analyze index iot\_pk2 validate structure;

select lf\_blks, br\_blks, used\_space,

opt\_cmpr\_count, opt\_cmpr\_pctsave

from index\_stats



На этом шаге количество листовых блоков сократилось до 47, а объем используемой памяти до 327 Кбайт.

Параметры **PCTTHRESHOLD, OVERFLOW и INCLUDING**. Они взаимосвязаны и позволяют обеспечить более эффективное хранение данных в листовых блоках индекса (именно в этих блоках и хранятся фактические данные индекса). Индекс обычно создается по подмножеству столбцов таблицы. Обычно в блоке индекса помещается во много раз больше строк, чем в блоке таблицы, организованной в виде кучи. Эффективность индекса зависит от возможности хранить в блоке много строк; в противном случае серверу Oracle придется тратить много времени на поддержку структуры индекса, ведь каждый оператор INSERT или UPDATE при этом может приводить к делению индексного блока, чтобы он мог вместить новые данные.

Конструкция OVERFLOW позволяет задать другой сегмент, дополнительный сегмент, который помещаются данные строк организованной по индексу таблицы, когда они становятся слишком большими. Обратите внимание, что для сегмента OVERFLOW опять можно задавать параметр PCTUSED. Параметры PCTFREE и PCTUSED для сегмента OVERFLOW имеют такое же значение, что и для таблицы, организованной в виде кучи. Условия использования дополнительного сегмента можно задавать двумя способами.

• С помощью конструкции PCTTHRESHOLD. Когда объем данных в строке превзойдет этот процент от размера блока, хвостовые столбцы такой строки будут храниться в дополнительном сегменте. Итак, если параметр PCTTHRESHOLD имеет значение 10 процентов, а размер блока — 8 Кбайт, любая строка размером более 800 байт будет частично храниться в другом месте, вне блока индекса.

• С помощью конструкции INCLUDING. Все столбцы строки, вплоть до столбца, порядковый номер которого указан в конструкции INCLUDING, хранятся в блоке индекса, а остальные столбцы — в дополнительном сегменте.



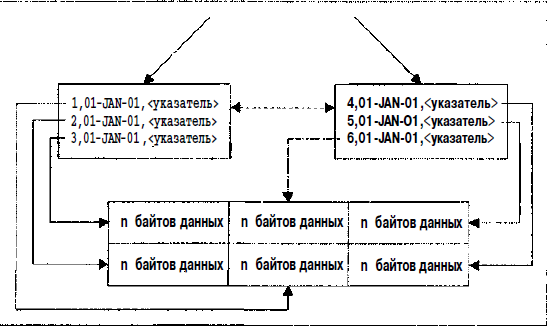
Верхние прямоугольники — это записи индекса, часть большей индексной структуры. Если коротко, структура индекса — дерево, а листовые блоки (в которых хранятся данные) фактически образуют двухсвязный список, упрощающий последовательный просмотр блоков. Нижний прямоугольник представляет дополнительный сегмент, в нем будут храниться данные, превосходящие устанавливаемый параметром PCTTHRESHOLD предел. Сервер Oracle будет просматривать столбцы в обратном порядке, начиная с последнего столбца строки и заканчивая последним столбцом первичного ключа (но не включая его), чтобы определить, какие столбцы надо хранить в дополнительном сегменте. В нашем примере числовой столбец X и столбец типа даты Y всегда будут помещаться в блоке индекса. Последний столбец, Z, имеет переменную длину. Когда его длина составляет менее 190 байт (10 процентов от блока размером 2 Кбайт — это около 200 байт, но надо вычесть 7 байт для даты и от 3 до 5 — для числа), он будет храниться в блоке индекса. Когда же его длина превысит 190 байт, сервер Oracle будет хранить данные столбца Z в дополнительном сегменте, и добавит указатель на них в блок индекса.

Еще можно использовать конструкцию INCLUDING. Она позволяет явно указать, какие столбцы должны храниться в блоке индекса, а какие — в дополнительном сегменте.

Для следующей таблицы:

Create table lot (x int, у date, z varchar2(2000), constraint iot\_pk primary key (x) ) organization index **including у overflow**;

можно ожидать такую структуру:



В этом случае столбец Z, независимо от объема содержащихся в нем данных, будет храниться "вне строки", в сегменте остатка.