**Лекция 15:**

**Создание физической модели базы данных: проектирование производительности**

**Аннотация:**В настоящей лекции рассматриваются вопросы проектирования для обеспечения требуемого уровня производительности физической структуры хранилища данных на основе СУБД-ориентированных средств: индексов, секций, кластеров.

**Цель лекции**

Изучив материал настоящей лекции, вы будете знать:

* основные типы *индексирования* объектов базы данных;
* основные приемы повышения производительности с помощью подключения *индекса* ;
* основные типы *индексов* реляционных баз данных;
* достоинства и недостатки *индексирования* ;
* что такое *секционирование* таблиц;
* что такое *секционирование индексов* ;
* что такое *секционирование представлений* ;
* достоинства и недостатки *секционирования* ;
* что такое *кластеризация* ;
* что такое *индексный кластер* и *хеш-кластер* ;
* какие достоинства и недостатки имеет *кластеризация* ;

и научитесь:

* создавать различные типы *индексов* ;
* создавать *секционированные таблицы*;
* создавать секционированные *индексы* ;
* создавать *секционированные представления*;
* создавать *кластеры*.

**Литература**: [[2](https://intuit.ru/studies/courses/599/455/literature#literature.2)], [[3](https://intuit.ru/studies/courses/599/455/literature#literature.3)], [[37](https://intuit.ru/studies/courses/599/455/literature#literature.37)], [[67](https://intuit.ru/studies/courses/599/455/literature#literature.67)].

**Введение**

Как уже отмечалось в предыдущей лекции, одной из важнейших задач разработки *физической модели данных* ХД является обеспечение гарантии того, что ХД обеспечивает требуемый уровень производительности. В той же лекции мы рассмотрели основной подход к решению этой задачи — *денормализацию* таблиц *физической модели данных* ХД в рамках подхода реляционной модели. В настоящей лекции мы продолжим обсуждать методы борьбы за *производительность* ХД и сконцентрируем свое внимание на *СУБД*-ориентированных средствах для решения этой задачи: *индексах*, секциях, *кластерах*.

**Повышение производительности запросов: индексы**

**Индексирование**

***Индексирование*** (indexing) — **это способ обеспечения быстрого доступа к значениям колонки или комбинации колонок**. Физически новые строки добавляются в конец таблицы, результатом чего становится неупорядоченное размещение значений в колонках. Без использования каких-либо методов упорядочения данных единственным способом просмотра значения колонки со стороны СУБД является последовательный просмотр каждой строки от начала таблицы к ее концу — так называемое **сканирование таблицы**. Производительность такого сканирования пропорциональна размеру таблицы, размеру физической страницы БД и длине строки таблицы.

Одним из способов внесения отношения порядка в значения колонок без нарушения физического расположения строк таблицы является создание объекта реляционной СУБД — *индекса* (index). ***Индекс*** — это объект в реляционной БД, который предназначен для организации быстрого доступа к строкам таблицы по значениям одной или более колонок этих строк.

Концептуально действие *индекса* состоит в следующем. В *индексе* содержатся упорядоченный список значений колонки или комбинации колонок, а также сведения о местонахождении на жестком диске соответствующих этим значениям строк таблицы. Значения колонки в *индексе* упорядочены. Несмотря на то, что порядок строк в таблице случаен, *индекс* можно быстро просмотреть, чтобы найти конкретное значение. Упорядоченный *индекс* можно просмотреть во много раз быстрее, чем неупорядоченную таблицу. Чем выше степень различия значений ключа в колонке, тем быстрее будет выполняться доступ к строкам этой таблицы.

Так, при вставке новой записи в таблицу проверка уникальности первичного ключа реализуется не реальным просмотром таблицы БД, а тем, что требование уникальности предъявляется к значениям колонки первичного ключа в *индексе*. Таким образом, *индекс* — это объект базы данных, который может существенно сократить время поиска нужных строк в таблице.

*Индексы* занимают место в БД. При вводе новых данных или удалении данных СУБД приходится обновлять и таблицы, и *индексы*. Это может замедлить выполнение операций модификации данных, особенно для таблиц с большим числом строк, как в ХД. Таким образом, может появиться проблема, суть которой состоит в возникновении конфликта между скоростью обновления данных в таблице и скоростью ее считываний. При разрешении этой проблемы следует придерживаться следующего эмпирического правила: создавать *индексы* для колонок первичных ключей и других колонок, часто используемых в тех запросах, в которых для выборки данных применяются логические критерии. Если в результате скорость обновления данных ухудшается, то можно рассмотреть вопрос об удалении некоторых *индексов*.

Каждая таблица БД может иметь один или несколько *индексов*. *Индексы* создаются по одной колонке или нескольким колонкам таблицы. Колонки, входящие в *индекс*, принято называть ***ключевыми полями (key fields), или ключами***. *Индексы* могут быть **уникальными** и **неуникальными**. Уникальность *индекса* означает, что не существует двух строк с одним и тем же значением ключа *индекса*. Неуникальный *индекс* может иметь несколько ключей с одинаковыми значениями.

Основными целями создания *индексов* в БД являются:

* ускорение поиска строк в таблицах;
* обеспечение уникального значения в колонках;
* извлечение строк в заданном порядке на основании значений индексированных колонок (что оправдано только для очень больших таблиц, когда использование предложения ORDER ухудшает производительность).

На этапе создания *физической модели данных* ХД необходимо принять ряд важных решений о том, что и как индексировать; при этом важно четко сформулировать правила *индексирования*. Для каждого ИТ-проекта с ХД необходимо создать и оформить в письменном виде правила *индексирования* как часть общих правил обеспечения производительности. Поддержка и сопровождение *индексов* в процессе эксплуатации ХД является в основном задачей *администратора БД*. Решая задачи обеспечения производительности, *администратор БД* будет ставить вопрос о перепроектировании ее физической структуры (обратные задачи проектирования), в том числе и вопрос об удалении и создании новых *индексов*. Он может решать эти задачи самостоятельно. Тем более важно знать, по каким правилам и из каких соображений создавались *индексы* того или иного типа. Разработка таких правил значительно повысит качество эксплуатации ХД с точки зрения обеспечения ее производительности.

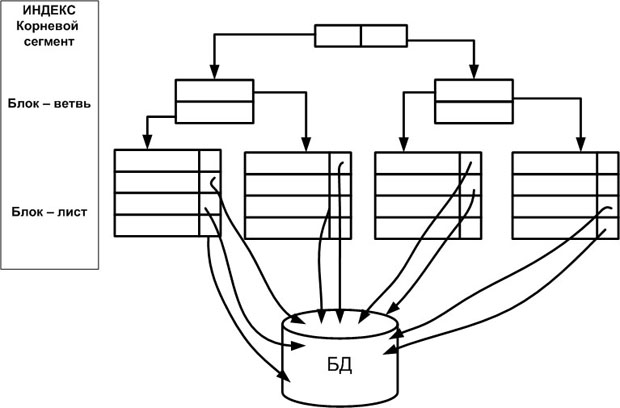
Чтобы решать эти задачи, проектировщик должен знать, как работает *индекс*, какие типы *индексов* поддерживает СУБД, и понимать смысл методов *индексирования*.

Сначала мы опишем типы *индексов* вместе с методами *индексирования* для каждого типа, затем разберем вопрос о том, как работает *индекс*, и в заключение дадим некоторые рекомендации по созданию и использованию *индексов*.

**Индекс со структурой B-Tree**

*Индекс* на основе сбалансированной иерархической структуры (или *индекс* B-Tree, balanced *tree structured* object) напоминает дерево, если смотреть на него снизу вверх. При работе СУБД с этой структурой сначала считывается самый верхний блок — корневой узел (root), затем блок на следующем уровне — блок-ветвь (branch), и так далее, до тех пор, пока не будет извлечен блок-лист (*leaf*) с индексируемыми колонками (колонкой) строки. Обратим внимание, что значения индексируемых колонок сохраняется в *индексе* ([рис. 20.1](https://intuit.ru/studies/courses/599/455/lecture/10182?page=1#image.20.1)).

Такая структура позволяет сократить до минимума число операций ввода-вывода. Для получения индексируемых колонок строки обычно требуется одно посещение блока-листа, т.е. физической страницы файловой структуры БД, отведенной под *индекс*.

[](https://intuit.ru/EDI/20_07_20_2/1595197216-9970/tutorial/632/objects/20/files/14_01.jpg)

[увеличить изображение](https://intuit.ru/EDI/20_07_20_2/1595197216-9970/tutorial/632/objects/20/files/14_01.jpg)  
**Рис. 20.1.**Концептуальная организация B-Tree индекса

Следует отметить два случая, когда после выборки индексируемых колонок строки из *индекса* может понадобиться несколько посещений физической страницы *индекса*:

1. когда строка имеет длину более одной физической страницы файловой структуры БД — так называемая **расщепленная строка** ;
2. когда строка за время своего существования в БД увеличилась и была перемещена из исходной страницы в другую страницу — так называемая **мигрировавшая строка**.

*Индекс* B-Tree характеризуется количеством уровней в *индексе* – высотой (height). Чем меньше уровней, тем выше производительность.

**Индекс B-Tree** — это физический объект реляционной БД, организованный по принципу сбалансированной иерархической структуры и обладающий набором свойств. Сформулируем некоторые свойства *индексов со структурой B-Tree*.

* Количество операций ввода-вывода, необходимых для получения идентификатора строки, зависит от числа уровней ветвления дерева. По мере увеличения *индекса* в результате добавления новых данных СУБД добавляет в него новые уровни, чтобы обеспечить сбалансированность дерева. Однако в действительности таких уровней редко бывает более четырех.
* Корневой узел и узлы — ветви *индекса* сжимаются и поэтому содержат ровно столько начальных байтов значения ключа, сколько нужно для того, чтобы отличить его от других значений. Узлы-листья содержат полное значение ключа.
* Значения в *индексе* упорядочиваются по ключевому значению, а физические страницы *индекса* организуются в двунаправленный список. Это обеспечивает последовательный доступ к *индексу* и позволяет использовать *индекс* для выполнения операции ORDER BY в запросе.
* *Индекс* можно использовать для поиска как точного соответствия, так и диапазона значений.
* *Индексы* могут быть построены для нескольких колонок таблицы — так называемый составной *индекс*. СУБД использует составные *индексы* для выполнения тех запросов, в которых задана лидирующая часть *составного ключа*. Например, составной *индекс* **{"Фамилия" (Ename), "Должность" (Job)}** для обработки запроса SELECT \* FROM EMPLOYEE WHERE Job='Инженер'; применяться не будет.
* СУБД обычно сама принимает решение, использовать *индекс* или нет.
* Значения колонок **NULL** не индексируются. Если для таких колонок строится *индекс*, то СУБД будет отказываться применять его в некоторых операциях, например, ORDER BY.

В СУБД семейства MS SQL Server все *индексы* организованы на основе сбалансированной иерархической структуры. Помимо того, что *индексы* обладают свойствами уникальности ( UNIQUE ) и неуникальности, *индексы* в СУБД семейства MS SQL Server могут быть *кластеризованными* ( CLUSTERED ) или *некластеризованными* ( NONCLUSTERED ), являющимися *индексами* по умолчанию.

***Кластеризованный индекс*** – **это индекс, в котором логический порядок значений ключа определяет физический порядок соответствующих строк в таблице**. На нижнем (конечном) уровне такого *индекса* хранятся действительные строки данных таблицы. Для таблицы или представления в каждый момент времени может существовать только один *кластеризованный индекс*.

***Некластеризованный индекс*** – **это индекс, в котором задается логическое упорядочение для таблицы**. Логический порядок строк в *некластеризованном индексе* не влияет на их физический порядок. Для каждой таблицы можно создать до 999 *некластеризованных индексов*, независимо от того, каким образом они создаются: неявно, с помощью ограничений PRIMARY KEY и UNIQUE, или явно, с помощью команды CREATE INDEX, которая была рассмотрена в ["Знакомство с CASE инструментом"](https://intuit.ru/studies/courses/599/455/lecture/10166).

В *кластеризованном индексе* конечные узлы содержат страницы данных базовой таблицы. На страницах *индекса* корневого и промежуточного узлов находятся строки *индекса*. Каждая строка *индекса* содержит ключевое значение и указатель либо на страницу промежуточного уровня сбалансированного дерева, либо на строку данных на конечном уровне *индекса*. Страницы на каждом уровне связаны в двунаправленный список.

По умолчанию *кластеризованный индекс* занимает одну секцию на дисковом пространстве. Если *кластеризованный индекс* занимает несколько секций, каждая секция включает сбалансированное дерево, содержащее данные этой секции. Например, если *кластеризованный индекс* занимает четыре секции, существует четыре сбалансированных дерева: по одному в каждой секции.

SQL Server движется вниз по *индексу*, чтобы найти строку, соответствующую ключу *кластеризованного индекса*. Чтобы найти диапазон ключей, SQL Server сначала находит начальное значение ключа в диапазоне, а затем сканирует страницы данных, используя указатели на следующую и предыдущую страницу. Чтобы найти первую страницу в цепочке страниц данных, SQL Server движется по самым левым указателям от корня *индекса*.

В зависимости от типов данных каждая структура *кластеризованного индекса* состоит из одной или более единиц распределения, которые применяются для хранения и управления данными секциями. Для каждой секции *кластеризованный индекс* содержит как минимум одну единицу распределения IN\_ROW\_DATA. Для хранения столбцов больших объектов ( LOB ) *кластеризованному индексу* требуется одна единица распределения LOB\_DATA для каждой секции. Кроме того, для хранения строк переменной длины, которые превышают ограничение на размер строки, равное 8 060 байтам, для каждой секции требуется одна единица распределения ROW\_OVERFLOW\_DATA.

По своей организации *некластеризованные индексы* отличаются от *кластеризованных* следующим: строки данных в базовой таблице не сортируются и хранятся в порядке, который основан на их некластеризованных ключах; конечный уровень *некластеризованного индекса* состоит из страниц *индекса* вместо страниц данных.

*Некластеризованные индексы* могут определяться на таблице или представлении с *кластеризованным индексом* либо на куче. Каждая строка *некластеризованного индекса* содержит некластеризованное ключевое значение и указатель на строку. Этот указатель определяет строку данных *кластеризованного индекса* или кучи, содержащую ключевое значение.

Указатели строк на строках *некластеризованных индексов* являются либо указателем на строку, либо ключом *кластеризованного индекса* для строки, как описано ниже.

Если таблица является кучей (то есть не содержит *кластеризованный индекс* ), то указатель строки является указателем на строку. Указатель строится на основе идентификатора файла ( **ID** ), номера страницы и номера строки на странице. Весь указатель целиком называется идентификатором строки ( **RID** ).

Если для таблицы имеется *кластеризованный индекс* или *индекс* построен на индексированном представлении, то указатель строки — это ключ *кластеризованного индекса* для строки. Если *кластеризованный индекс* не является уникальным *индексом*, то SQL Server создает все имеющиеся повторяющиеся ключи уникальными путем добавления внутри созданного значения, называемого uniqueifier. Это четырехбайтовое значение невидимо для пользователей. Оно применяется тогда, когда необходимо сделать *кластеризованный ключ* уникальным, чтобы использовать в *некластеризованных индексах*. SQL Server получает строку данных путем поиска по *кластеризованному индексу*, задействуя ключ *кластеризованного индекса*, который хранится в конечной строке *некластеризованного индекса*.

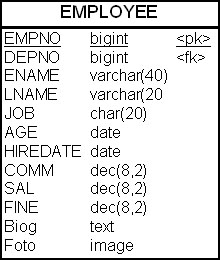
По умолчанию *некластеризованный индекс* включает одну секцию. Если он состоит из нескольких секций, то каждая секция имеет структуру сбалансированного дерева, в которой содержатся индексные строки для данной конкретной секции. Например, если *некластеризованный индекс* содержит четыре секции, то существуют четыре структуры сбалансированного дерева, по одной на каждую секцию.

В зависимости от типов данных в *некластеризованном индексе* каждая его структура будет содержать одну или более единиц распределения, в которых хранятся данные для определенной секции. Каждый *некластеризованный индекс* будет иметь по меньшей мере одну единицу распределения IN\_ROW\_DATA на секцию, в которой хранятся страницы сбалансированного дерева *индекса*. *Некластеризованный индекс* будет также содержать одну единицу распределения LOB\_DATA на секцию, если в *индексе* есть столбцы типа большого объекта (LOB). Кроме того, *некластеризованный индекс* будет включать одну единицу распределения ROW\_OVERFLOW\_DATA на секцию, если в *индексе* имеются столбцы переменной длины, в которых превышается максимальный размер строки, равный 8 060 байт.

**Пример 20.1**.

Далее в примерах, если не оговорено особо, мы будем использовать таблицу "Служащий" (EMPLOYEE) ([рис. 20.2](https://intuit.ru/studies/courses/599/455/lecture/10182?page=1#image.20.2)) из ["Проектирования модели ХД по логической модели"](https://intuit.ru/studies/courses/599/455/lecture/10172). Повторим ее здесь.

|  |  |  |
| --- | --- | --- |
| Таблица 20.1. Описание полей таблицы "Служащий" (EMPLOYEE) | | |
| **№** | **Наименование атрибута** | **Наименование колонки** |
| 1 | Номер личной карточки | EMPNO (PK) |
| 2 | Фамилия | ENAME |
| 3 | Имя | LNAME |
| 4 | Номер подразделения | DEPNO |
| 5 | Должность | JOB |
| 6 | Дата рождения | AGE |
| 7 | Стаж | HIREDATE |
| 8 | Доплаты | COMM |
| 9 | Зарплата | SAL |
| 10 | Штрафы | FINE |
| 11 | Автобиография | Biog |
| 12 | Фотография | Foto |



**Рис. 20.2.**Таблицы "Служащий" (EMPLOYEE)

Создадим для таблицы "Служащий" (EMPLOYEE) составной *индекс* по колонкам "Фамилия" (Ename) и "Должность" (Job).

CREATE INDEX emp\_ndx2 ON EMPLOYEE (Ename, Job)

GO

**Исключительно индексные таблицы и другие типы индексов на основе B-Tree**

*Исключительно индексная таблица* (index-organized table) может создаваться на основе значений одной или нескольких колонок. Если требования к данным в запросе удовлетворяются на основе информации из связанного с этими данными *индекса*, то доступ к базовой таблице не осуществляется.

В некоторых СУБД, в частности в СУБД Oracle, *исключительно индексные таблицы* поддерживаются. *Исключительно индексная таблица* является *индексом* типа B-Tree БД, который одновременно исполняет роль таблицы. Все данные такой таблицы хранятся в *индексе*. Преимущество создания полностью *индексированных таблиц* состоит в экономии места хранения на диске и сокращении объема ввода-вывода, поскольку ключевые колонки нет необходимости сохранять еще раз в таблице. Результат выполнения запроса будет получен на основе данных, сохраненных в индексной таблице.

*Исключительно индексная таблица* в СУБД Oracle создается с помощью команды SQL CREATE TABLE, как показано в примере 20.2.

**Пример 20.2**.

Предположим, что требуется в отдельной таблице БД сохранять и отслеживать проблемы, возникающие по выполнении всех проектов, а так же запоминать, кто обращался и как часто возникали проблемы. Создадим *исключительно индексную таблицу* "Проект индексированный" (Proj\_Index) для решения этой задачи, как показано ниже:

CREATE TABLE Proj\_Index

( projno char(8) NOT NULL,

t\_person char(32) NOT NULL,

t\_frequency integer,

t\_problem varchar2(512),

CONSTRAINT pk\_ndx PRIMARY KEY( projno, t\_person) )

ORGANIZATION INDEX

TABLESPACE ts\_ndx1

PCTTHRESHOLD 20

INCLUDING t\_frequency

OVERFLOW TABLESPACE ts\_\_of\_ndx1;

Команда CREATE TABLE не отличается ничем от других команд создания таблиц — до тех пор, пока не встретится предложение ORGANIZATION INDEX, которое указывает СУБД на создание *исключительно индексной таблицы*. Для размещения *индекса* на диске указывается *табличное пространство*. Параметр PCTTHRESHOLD говорит, что оставшуюся часть строки нужно сохранять в заданном *табличном пространстве* — сегменте переполнения, если данная строка превышает размер физической страницы базы данных на указанное число процентов. Параметр INCLUDING определяет имя колонки, с которой строка индексной таблицы делится на две части: индексную и переполнения. Эта колонка может быть частью первичного ключа таблицы или неключевой колонкой. Все неключевые колонки, которые следуют за указанной колонкой, размещаются в сегменте переполнения, который определяется ключевым словом OVERFLOW.

В СУБД семейства MS SQL Server нет предопределенной возможности создавать *исключительно индексные таблицы*. Однако в ряде практических случаев в СУБД этого семейства можно создавать структуры, аналогичные *исключительно индексным таблицам*. Для этого может быть использован *некластеризованный индекс* с включенными колонками (опция INCLUDE команды CREATE INDEX ).

Опция INCLUDE указывает неключевые колонки, добавляемые на конечный уровень *некластеризованного индекса*. *Некластеризованный индекс* может быть уникальным или неуникальным. Имена колонок в списке INCLUDE не могут повторяться и не могут использоваться одновременно как ключевые и неключевые.

Для примера 20.2 команды создания структуры, подобной *исключительно индексной таблице* по своему назначению, на диалекте SQL MS SQL Server будут иметь вид, как в примере ниже.

**Пример. 20.3**.

Сначала создаем таблицу "Проект индексированный" (Proj\_Index), как на [рис. 20.3](https://intuit.ru/studies/courses/599/455/lecture/10182?page=2#image.20.3).

CREATE TABLE Proj\_Index (

projno char(8) NOT NULL,

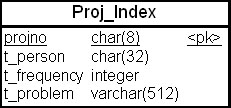
t\_person char(32) NOT NULL,

t\_frequency integer,

t\_problem varchar(512),

CONSTRAINT pk\_ndx PRIMARY KEY( projno, t\_person) )

GO



**Рис. 20.3.**Таблица "Проект индексированный" (Proj\_Index)

Затем создаем *некластеризованный индекс* с ключом "Номер проекта" (projno) и с тремя неключевыми колонками "Фамилия сотрудника" (t\_person), "Частота проблемы" (t\_frequency) и "Описание проблемы" (t\_problem).

CREATE NONCLUSTERED INDEX IX\_Proj

ON Proj\_Index (projno)

INCLUDE (t\_person, t\_frequency, t\_problem);

GO

В семействе СУБД Oracle предусмотрено еще несколько типов *индексов*, которые позволяют улучшить традиционные для всех СУБД *индексы со структурой B-Tree*. К таким модификациям, помимо *исключительно индексных таблиц*, относятся *битовые индексы*, *индексы с обращением ключа*, *индексы на основе значения функций*.

Каждый бит так называемого **битового** (bitmap) *индекса* относится к идентификатору строки ROWID (который в Oracle создается и хранится для каждой строки и используется во внутренней организации *индексов* ) в табличном объекте. Если некоторая строка содержит данное ключевое значение, то в *индексе* для этого значения сохраняется единица. Такая организация *индекса* может в некоторых случаях значительно повысить производительность выборки данных, т. к. для извлечения строк с определенными значениями *индекса* СУБД нужно лишь найти все единицы, отвечающие ключу. Физически такой *индекс* организован на основе структуры B-Tree, но задача сводится к поиску данной строки за счет одной операции чтения битовой индексной структуры. Этот тип *индекса* очень эффективен для *индексирования* колонок с небольшим кардинальным числом — пол, цвет и т.д. Если значений у колонки будет много, то объем ввода-вывода будет возрастать.

**Пример 20.4**.

Для нашего учебного примера можно построить *битовый индекс* для таблицы "Служащий" (EMPLOYEE) по колонке "Номер отдела" (DEPNO):

CREATE BITMAP INDEX emp\_ndx ON EMPLOYEE (DEPNO);

В СУБД семейства MS SQL Server возможность создания *битовых индексов* средствами диалекта SQL отсутствует.

В *индексе с обращением ключа* (reverse-key index) применяется обращение байтов индексируемой колонки числового типа. Этот прием позволяет получать равномерное распределение значений колонок среди блоков-листков *индекса со структурой B-Tree*, который хорошо подходит для *индексирования* колонок с последовательной нумерацией или нумерацией с заданным шагом. Заметим, что такие *индексы* применяются только для возвращения отдельных строк, и с их помощью нельзя выполнить поиск значений в некотором диапазоне. В СУБД Oracle нельзя применить опцию **REVERSE** к *битовым индексам* и к *исключительно индексным таблицам*.

**Пример 20.5**.

Числовые ключи, содержащие последовательные числа, есть, в частности, в таблице "Служащий" (EMPLOYEE) – "Номер служащего" (EMPNO). Можно создать для этой таблицы дополнительный *индекс с обращением ключа* для извлечения записи о сотруднике. Заметим, что для этой колонки уже есть *индекс* первичного ключа.

CREATE INDEX dep\_ndx ON EMPLOYEE (EMPNO) REVERSE;

В процессе эксплуатации администратор базы данных может перестроить этот *индекс* с помощью команды ALTER INDEX:

ALTER INDEX EMPLOYEE REBUILD NOREVERSE;

В СУБД семейства MS SQL Server возможность создания *индексов* с обращением значения ключа средствами диалекта SQL отсутствует.

Если в предложении WHERE применяется функция по индексированной колонке, то обычно СУБД не используют этот *индекс* при организации доступа к строкам таблицы. Но при создании *индекса на основе значения функции* (function-based index), которая является той же функцией, что и в предложении WHERE, СУБД как семейства Oracle, так и семейства MS SQL Server использует такой *индекс* для считывания строк, удовлетворяющих критерию отбора.

В СУБД семейства MS SQL Server вычисляемые колонки могут иметь свойство PERSISTED. Это означает, что компонент *Database Engine* хранит вычисленные значения в таблице и обновляет их при обновлении любых колонок, от которых зависит вычисляемый столбец. Компонент *Database Engine* использует эти материализованные значения, когда создает *индекс* по колонке и когда запрос обращается к *индексу*.

Для индексации вычисляемой колонки она должна быть детерминированной и точной. Если используется свойство PERSISTED, список типов индексируемых вычисляемых колонок расширяется и включает следующие типы:

1. вычисляемые столбцы, основанные на выражениях языка Transact-SQL, функциях CLR и методах пользовательских типов данных CLR, отмеченных пользователем как детерминированные;
2. вычисляемые столбцы, основанные на выражениях, которые определены компонентом *Database Engine* как детерминированные, но не являются точными.

**Пример 20.6**.

В качестве примера использования *индексов* по вычисляемой колонке приведем пример моделирования хеш-индекса в MS SQL Server.

Предположим, что никакого *индекса* для колонки "Фамилия" (ENAME) в таблице "Служащий" (EMPLOYEE) создано не было. Тогда, если в предложении WHERE команды SELECT задан предикат поиска по колонке "Фамилия" (ENAME), то СУБД будет сканировать таблицу. Если таблица содержит около 1000000 строк, то операция сканирования будет увеличивать время обработки запроса.

Создадим хеш-индекс для колонки "Фамилия" (ENAME) с помощью функции *checksum*(), для этого добавим в таблицу вычисляемую колонку, как показано ниже.

alter table EMPLOYEE

add ENameHash as checksum(EName);

go

Далее создадим *индекс* по этой колонке, как показано ниже.

create index IDX\_ENames on EMPLOYEE (ENameHash);

go

При наличии в БД такого *индекса* СУБД будет его использовать при обработке запроса описанного в данном примере.

В СУБД семейства MS SQL Server можно создавать и использовать фильтрованные *индексы*, для создания таких *индексов* в команде CREATE INDEX предусмотрена возможность использования предложения WHERE.

Применение предложения WHERE <filter\_predicate> создает отфильтрованный *индекс* путем указания строк для включения в *индекс*. Отфильтрованный *индекс* должен быть *некластеризованным индексом* для таблицы. Также создается статистика фильтрации для строк данных отфильтрованного *индекса*.

Предикат фильтра использует простую логику сравнения и не может ссылаться на вычисляемый столбец, столбец определяемого пользователем типа, столбец пространственного типа данных или столбец типа hierarchyID. Сравнения с помощью литералов NULL с операторами сравнения недопустимы. Вместо этого используются операторы IS NULL и IS NOT NULL.

Фильтруемый *индекс* подходит для столбцов, имеющих точно определенные подмножества данных.

**Пример 20.7**.

Создадим фильтрованный *индекс* в таблице "Служащий" (EMPLOYEE) для колонки "Штрафы" (FINE), чтобы быстро получать списки оштрафованных сотрудников.

CREATE NONCLUSTERED INDEX IDX\_FINE

ON EMPLOYEE (EName, FINE)

WHERE FINE > 0;

GO

В настоящем разделе мы рассмотрели различные типы *индексов* и примеры их создания. В следующем разделе остановимся подробнее на параметрах проектирования *индексов*.

**О некоторых параметрах проектирования индексов**

При проектировании *индексов* необходимо иметь некоторый механизм для оценки качества предполагаемого *индекса*. Введем несколько понятий, с помощью которых можно грубо оценить качество потенциального *индекса*.

***Кардинальностью колонки*** (*cardinality*) **таблицы называется число дискретных различных значений колонки, которые встречаются в строках таблицы**. Например, если в таблице "Служащий" (EMPLOYEE) мы заводим колонку для указания пола – "Пол" (SEX), то *кардинальность* этой колонки есть 2, так в природе у людей существует только два пола — мужской и женский. Для колонки первичного ключа *кардинальность* будет равна числу строк в таблице.

Причина, по которой *кардинальность колонки* важна для проектирования *индексов*, состоит в том, что *кардинальность* индексируемой колонки определяет число уникальных входов, которые должны сохраняться в *индексе*, т.е. число записей в *индексе*. Так, для индексируемой колонки "Пол" (SEX) будут существовать два уникальных входа, которые будут повторяться много раз в *индексе*. При предположении равновероятного распределения пола сотрудников на 100000 строк в таблице "Служащий" (EMPLOYEE) каждый вход *индекса* будет повторяться 50000 раз. СУБД вряд ли будут принимать решение об использовании такого *индекса* при построении плана запроса.

Определить *кардинальность* потенциальной колонки *индексирования* в существующей таблице БД достаточно просто.

SELECT COUNT (DISTINCT колонка) FROM таблица;

При проектировании новой БД для ХД следует оценить *кардинальность* всех потенциальных индексируемых колонок во всех таблицах *физической модели данных* ХД, исходя из имеющейся документации.

Способ, с помощью которого СУБД оценивает действие кардинальности, состоит в использовании *фактора селективности выборки* (selectivity factor). ***Фактор селективности выборки*** *индекса* определяется как величина, обратная *кардинальности* индексной колонки:

\mbox{selectivity\_factor}=\frac{1}{\mbox{cardinality}}

*Фактор селективности* оценивает потенциальный объем операций ввода-вывода. Чем меньше *фактор селективности*, тем меньше требуется операций ввода-вывода для получения результирующего множества строк таблицы. СУБД оценивает эту величину, чтобы решить, применять *индекс* для доступа к строкам таблицы или нет.

В заключение раздела приведем список правил для определения колонок, которые являются хорошими и плохими кандидатами для *индексирования*. Эти правила могут быть использованы при принятии решения о построении *индексов* реляционной базы данных.

Хорошими кандидатами для *индексирования* обычно являются:

* *колонки первичного ключа*. По определению, колонки первичного ключа должны иметь уникальный *индекс* ;
* *колонки внешнего ключа*. Они дают хороший *индекс* по двум причинам. Во-первых, они часто применяются для выполнения соединений с родительскими таблицами. Во-вторых, они могут быть использованы СУБД при *поддержке ссылочной целостности* в операциях удаления строк родительской и *дочерних таблиц*;
* любые колонки, которые содержат уникальные значения;
* колонки, запросы или соединения, которые используют от 5 до 10% строк таблицы;
* колонки, которые часто входят как аргументы в функции агрегирования;
* колонки, которые часто используются для проверки правильности ввода данных в программах ввода-редактирования.

Факторы, влияющие на низкую эффективность *индексов*:

* *таблицы маленького размера*. Не следует создавать *индексы* для таблиц размером менее пяти физических страниц. Для таких страниц стоимость поддержки *индекса* больше, чем стоимость сканирования всей таблицы. Конечно, уникальный *индекс* требуется для первичного ключа и *поддержки ссылочной целостности*;
* *интенсивные обновления таблиц в пакетном режиме*. Такие таблицы обычно имеют проблемы с переполнением *индекса* при интенсивной модификации таблицы. Если *индекс* необходим для такой таблицы, то целесообразнее его удалять перед обновлением и создавать после него;
* *асимметрия значений ключей (Skewness of keys)*. Если распределение значений ключа имеет значительную асимметрию, то кардинальность *индекса* может оказаться достаточно высокой и СУБД из-за низкого *фактора селективности* будет часто использовать этот *индекс*. Но результат применения *индекса* будет неудовлетворительным из-за того, что значительная часть строк таблицы имеет одно и то же значение ключа, что приведет к нивелированию стоимости использования *индекса* по сравнению со сканирование всей таблицы.

Плохими кандидатами для *индексирования* обычно являются:

* *колонки с низкой кардинальностью*. Они дают высокий *фактор селективности*, и СУБД обычно избегает их использования при обработке запросов. Стоимость поддержки *индекса* для колонки с низкой кардинальностью сопоставима со стоимостью сканирования всей таблицы, поскольку при доступе через *индекс* многие страницы базой таблицы посещаются много раз;
* *колонка имеет много неопределенных значений* (NULL-значения). В этом случае неопределенные значения могут дать значительную асимметрию распределения значений колонки, несмотря на то, что *кардинальность колонки* будет подходящей для использования *индекса* ;
* *колонки с часто изменяемыми значениями*. *Индекс* для таких колонок часто обновляется, что приводит к его переполнению, поскольку в большинстве алгоритмов обработки B-Tree-индексов физическая страница *индекса* становится доступной для распределения данных только после того, как из нее будет удалена последняя запись. В частности, это обстоятельство приводит к созданию дополнительных страниц *индекса* и уровней *индекса* ;
* *значительная длина индексных колонок*. Составной *индекс* или *индекс* для одной колонки с длиной более чем 50 байт будет приводить к росту числа уровней *индекса*, несмотря на то, что строк в таблице может быть немного. Большое число уровней снижает производительность операций выборки строк через *индекс*, т.к. каждый уровень требует по крайней мере одной операции ввода-вывода.

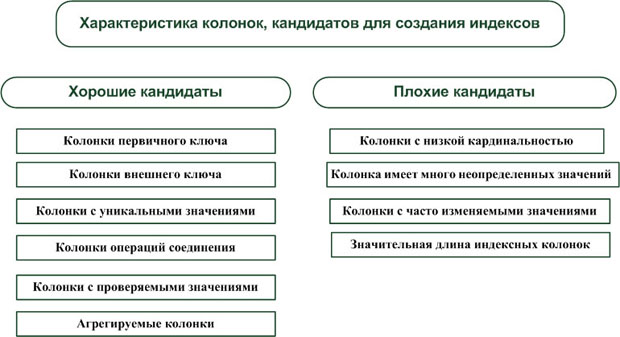
Следует соблюдать следующие общие правила при создании *индексов*.

* Там, где это возможно, использовать *ограничение первичного ключа*.
* Не следует создавать для таблицы несколько составных *индексов*, содержащих одни и те же колонки, но в другом порядке.
* Ни в коем случае нельзя допускать, чтобы у нескольких *индексов* для одной таблицы была одинаковая лидирующая часть.

К сожалению, часто проектировщики принимают крайне неудачные решения об *индексировании*. Это приводит к тому, что в ХД появляется слишком много *индексов*. В результате тратится много времени на поддержку этих *индексов*, дисковое пространство расходуется неэффективно, СУБД "путается" в выборе подходящего *индекса* или не использует их вовсе. Поэтому необходимо помнить о двух основных принципах построения *индекса*:

1. гарантировать уникальность значений колонки, которая будет индексироваться;
2. увеличить производительность обработки запросов в ХД. Это, кстати, единственная разумная причина для создания неуникальных *индексов*.

На [рис. 20.4](https://intuit.ru/studies/courses/599/455/lecture/10182?page=3#image.20.4) приведены рекомендации по выбору колонок для создания *индексов*.

[](https://intuit.ru/EDI/20_07_20_2/1595197216-9970/tutorial/632/objects/20/files/14_04.jpg)

[увеличить изображение](https://intuit.ru/EDI/20_07_20_2/1595197216-9970/tutorial/632/objects/20/files/14_04.jpg)  
**Рис. 20.4.**Характеристика колонок для создания индексов

**Повышение производительности запросов: секционирование**

**Секционирование**

Как правило, в таблицах ХД содержится огромное количество данных. Чем больше размер таблицы, тем больше времени потребуется для некоторых операций по выборке строк таблицы и для выполнения некоторых функций администратора базы данных — например, для резервного копирования и восстановления. Большие по размеру *индексированные таблицы* имеют также большие *индексы*, которые требуют много времени СУБД для их обслуживания.

***Секционирование*** (partitioning) — это способ физического распределения таблиц и *индексов* среди двух или более *табличных пространств* (СУБД Oracle) или в одной или более файловых группах (СУБД MS SQL Server) в зависимости от значений ключевых колонок таблиц с целью повышения производительности операций ввода-вывода. Таким образом, *секционирование* – это *разбиение таблицы* на группы, с сохранением для всех групп общего определения структуры. *Табличное пространство* (СУБД Oracle) или файловая группа (СУБД MS SQL Server) – это физическое месторасположение таблиц БД в файловой структуре операционной системы.

Фрагмент таблицы, расположенный в отдельной файловой группе, будем называть ***секцией таблицы***. *Секционирование* также повышает эффективность резервного копирования и восстановления за счет выполнения этих задач с участием меньшего объема данных. *Секцию таблицы* можно трактовать как предопределенный поименованный фрагмент памяти на одном или нескольких дисках, к которому можно обращаться в предложениях SQL по имени.

В реализации *секционирования* одним из важных понятий является колонка таблицы, относительно значений которой СУБД будет делать физическое разнесение таблицы по различным файловым группам на жестких дисках. Эти колонка называется ***ключом секционирования*** (partition key).

*Секционирование* таблиц и *индексов* задается жестко на уровне строк ( *секционирование* по столбцам не допускается) и позволяет осуществлять доступ через единую точку входа (имя таблицы или имя *индекса* ) таким образом, что в коде приложения не требуется знать число секций, находящихся за точкой входа. *Секционирование* может осуществляться на базовой таблице, а также на связанных с ней *индексах*.

Мы отдельно рассмотрим *секционирование* таблиц для семейств СУБД Oracle и MS SQL Server.

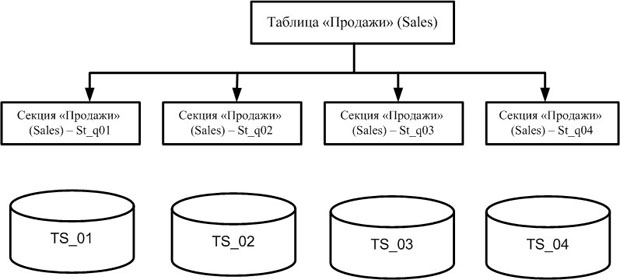
**Секционирование таблиц в СУБД семейства Oracle**

В СУБД семейства Oracle поддерживается несколько видов *секционирования*: *секционирование по диапазону*, *хеш-секционирование*, *составное секционирование*, а также различные виды *секционирования индексов*.

**Секционирование по диапазону**

***Секционирование по диапазону*** (range partitioning) означает распределение строк таблицы на различные предопределенные *табличные пространства* в зависимости от значения *ключа секционирования*. Доступ к такой таблице, как и к любой другой, осуществляется по ее имени, причем доступ к секциям, расположенным в каждом *табличном пространстве*, можно получить отдельно. Например, таблицу, содержащую финансовые квартальные отчеты организации, можно секционировать по дате таким образом, что отчеты по каждому кварталу будут храниться в отдельном *табличном пространстве*. При такой организации секций данные только по одному кварталу будут выбираться из одного табличного пространства, что повысит эффективность работы базы данных в целом.

*Секционирование по диапазону* базируется на упорядочении строк таблицы в секциях на основе значения колонок *ключа секционирования*. Концептуально таблица, секционированная по диапазону, устроена, как на [рис. 20.5](https://intuit.ru/studies/courses/599/455/lecture/10182?page=3#image.20.5).

[](https://intuit.ru/EDI/20_07_20_2/1595197216-9970/tutorial/632/objects/20/files/14_05.jpg)

[увеличить изображение](https://intuit.ru/EDI/20_07_20_2/1595197216-9970/tutorial/632/objects/20/files/14_05.jpg)  
**Рис. 20.5.**Секционирование таблицы "Продажи" (Sales)

Для создания *секционированных таблиц* используется команда SQL CREATE TABLE с предложением PARTITION. В СУБД Oracle *ключ секционирования* не может иметь тип LONG.

**Пример 20.8**.

Рассмотрим систему обработки заказов. Предположим, что в ней есть таблица "Продажи" (Sales), в которой сохраняются данные о количестве, времени и цене продаж для каждого покупателя. Можно использовать *секционирование по диапазону*, а именно по кварталу, для представления этой таблицы в базе данных. Предположим, что мы имеем четыре определенных ранее табличных пространства c именами ts\_01, ts\_02, ts\_03, ts\_04, распределенные по четырем дискам, как показано на рисунке ниже.

Фрагмент скрипта определяет таблицу "Продажи" (Sales) с физическим размещением секций, как на [рис. 20.5](https://intuit.ru/studies/courses/599/455/lecture/10182?page=3#image.20.5):

CREATE TABLE Sales

(

s\_customer\_id number(6),

s\_amt number(9,2),

s\_date date)

PARTITION BY RANGE (s\_date)

(PARTITION st\_q01 VALUES LESS THAN ('01-apr-2002')

TABLESPACE ts\_01,

PARTITION st\_q02 VALUES LESS THAN ('01-jul-2002')

TABLESPACE ts\_02,

PARTITION st\_q03 VALUES LESS THAN ('01-oct-2002')

TABLESPACE ts\_03,

PARTITION st\_q04 VALUES LESS THAN (MAXVALUE)

TABLESPACE ts\_04

);

Предложение PARTITION BY RANGE (s\_date) указывает СУБД Oracle выполнить *секционирование* таблицы по *ключу секционирования* s\_date. Предложения вида (PARTITION st\_q01 VALUES LESS THAN ('01-apr-2002') *TABLESPACE* ts\_01 определяют имя секции st\_q01 и ее размещение в соответствующем *табличном пространстве* ts\_01.

Чтобы получить доступ к строкам таблицы, расположенным в определенной секции, и узнать о продажах в третьем квартале, можно использовать команду SELECT, как показано ниже:

SELECT s\_customer\_id, s\_amt FROM Sales PARTITION (st\_q03);

Мы видим, что для того, чтобы получить доступ к строкам *секции таблицы*, нужно указать опцию PARTITION (имя секции) после имени таблицы в предложении FROM.

Администратор базы данных может легко удалять, добавлять, перемещать, расщеплять, усекать и изменять секции с помощью команды ALTER TABLE. Удалить отдельную секцию можно также, удалив соответствующее ей *табличное пространство*.

**Хеш-секционирование**

***Хеш-секционирование*** (hash partitioning) означает равномерное распределение строк таблицы по назначенным *табличным пространствам* в зависимости от значения *ключа секционирования*, который в данном случае хешируется. Этот вид *секционирования* удобно применять для строк, у которых распределение значений *ключа секционирования* неравномерно или плохо группируется. Для принятия решения о создании хеш-*секционированной таблицы* необходимо достаточно точно представлять размер этой таблицы, поскольку встроенные в СУБД Oracle алгоритмы хеширования используют этот размер для вычисления позиции строки на физической странице базы данных. Определение размера таблицы может привести к большому числу коллизий, т.е. к попаданию строк с различными значениями ключа на одну и ту же страницу, что приводит к поддержке цепочек переполнения и дополнительному вводу-выводу.

**Пример 20.9**.

Рассмотрим ту же, что и в примере 20.8, таблицу "Продажи" (Sales) и ту же схему ([рис. 20.5](https://intuit.ru/studies/courses/599/455/lecture/10182?page=3#image.20.5)) *табличных пространств*. Однако используем в качестве *ключа секционирования* "Идентификатор покупателя" ( s\_customer\_id ). Отметим, что распределение значений этой колонки может быть очень неравномерно. Фрагмент кода SQL для создания хеш-*секционированной таблицы* "Продажи" (Sales) можно написать так:

CREATE TABLE Sales

(

s\_customer\_id number(6),

s\_amt number(9,2),

s\_date date)

PARTITION BY HASH (s\_customer\_id)

(PARTITION q01 TABLESPACE ts\_01,

PARTITION q02 TABLESPACE ts\_02,

PARTITION q03 TABLESPACE ts\_03,

PARTITION q04 TABLESPACE ts\_04

);

Предложение PARTITION BY HASH (s\_customer\_id) указывает СУБД Oracle выполнить *секционирование* таблицы по *ключу секционирования* s\_customer\_id. Предложения вида (PARTITION q01 *TABLESPACE* ts\_01 определяют имя секции st\_q01 и ее размещение в соответствующем *табличном пространстве* ts\_01.

**Составное секционирование**

***Составное секционирование*** (composite partittioning) является комбинацией *секционирования по диапазону* и *хеш-секционирования*. Это означает, что таблица сначала распределяется среди *табличных пространств* на основе диапазона значений *ключа секционирования*, далее каждая из полученных секций диапазонов делится на *подчиненные секции*, или подсекции, и затем строки равномерно распределяются среди подчиненных секций по значению хеш-ключа.

**Пример 20.10**.

Рассмотрим ту же, что и в предыдущем примере, таблицу "Продажи" (Sales) и ту же схему *табличных пространств*. В качестве *ключа* *секционирования по диапазону* используем колонку "Дата продажи" (s\_date). В качестве *ключа* *хеш-секционирования* — "Идентификатор покупателя" (s\_customer\_id). Однако теперь каждая секция по диапазону будет разделена на предопределенное число подсекций. Фрагмент кода SQL для создания таблицы "Продажи" (Sales) с *составным секционированием* можно написать так:

CREATE TABLE Sales

(

s\_customer\_id number(6),

s\_amt number(9,2),

s\_date date)

PARTITION BY RANGE (s\_date)

SUB PARTITION BY HASH (s\_customer\_id)

SUB PARTITION 4

STORE IN (ts\_01, ts\_02, ts\_03, ts\_04)

(PARTITION q01 VALUES LESS THAN ('01-apr-2002'),

PARTITION q02 VALUES LESS THAN ('01-jul-2002'),

PARTITION q03 VALUES LESS THAN ('01-oct-2002'),

PARTITION q04 VALUES LESS THAN (MAXVALUE)

);

Секции q01, q02, q03, q04 будут содержать строки с диапазоном дат, которые определены в предложениях типа PARTITION q02 VALUES LESS THAN ('01-jul-2002') и будут распределены в табличных пространствах ts\_01, ts\_02, ts\_03, ts\_04. Предложение SUB PARTITION 4 предписывает СУБД Oracle разбиение каждой секции на четыре логические единицы, а предложение SUB PARTITION BY HASH (s\_customer\_id) распределяет строки заданного диапазона среди этих четырех подчиненных секций.

**Секционирование индексов в СУБД семейства Oracle**

В СУБД Oracle предусмотрено ***секционирование индексов*** (index partitioning), которое означает преднамеренное распределение *индексов* таблиц по назначенным *табличным пространствам* в соответствии с *ключом секционирования*. *Секционирование индексов* может быть **глобальным** и **локальным**.

*Локально секционированный индекс* имеет такой же *ключ секционирования*, то же количество *табличных пространств* и те же правила *секционирования*, что и отвечающая ему базовая таблица. *Глобально секционированный индекс* содержит предложение PARTITION BY RANGE, в котором задаются параметры *секционирования*, отличные от параметров *секционирования* соответствующей базовой таблицы.

Секционированные *индексы* могут быть префиксными или непрефиксными. В случае префиксного секционированного *индекса* *секционирование* производится по *ключу секционирования*, который содержит основную часть индексного ключа. В случае непрефиксных секционированных *индексов* *секционирование* выполняется по значениям, отличным от значений колонки *индексирования*.

*Индексы* могут быть секционированы и в случае, когда индексируемая таблица не секционируется. В этом случае по умолчанию предполагается, что *индекс* является глобальным *секционированным* *индексом*. В СУБД Oracle не предусмотрена поддержка глобальных непрефиксных секционированных *индексов*.

В локально секционированном *индексе* ключевые значения одной секции *индекса* соответствуют строкам таблицы из одной ее секции.

**Пример 14/11**.

Создадим локальный секционированный *индекс* для таблицы "Продажи" (Sales) из примера 20.8. *Ключом секционирования* этой таблицы является колонка "Дата продажи" (s\_date). Фрагмент кода создания *индекса* приведен ниже:

CREATE INDEX sales\_ndx ON Sales (s\_date)

LOCAL

(PARTITION st\_i\_q01 TABLESPACE ts\_01,

PARTITION st\_i\_q02 TABLESPACE ts\_02,

PARTITION st\_i\_q03 TABLESPACE ts\_03,

PARTITION st\_i\_q04 TABLESPACE ts\_04

);

Локально секционированный *индекс* называется **равносекционированным** (equi-partitioned), если он имеет то же число секций и те же правила *секционирования*, что и его базовая таблица. Обратите внимание, что в примере при создании *индекса* не использовалось предложение PARTITION BY RANGE. СУБД Oracle автоматически берет структуру *секционирования* для *индекса* из структуры *секционирования* базовой таблицы Sales. Также можно опустить и предложения типа PARTITION st\_i\_q02 *TABLESPACE* ts\_02. Если опущено PARTITION, то СУБД Oracle автоматически создаст имена секций. Если опущено *TABLESPACE*, то СУБД Oracle автоматически разместит секции в тех же табличных пространствах, в которых находятся соответствующие секции базовой таблицы.

Глобально секционированный *индекс* имеет структуру секций, отличную от структуры секций базовой таблицы данного *индекса*. В качестве примера создадим глобально секционированный *индекс* для таблицы "Продажи" (Sales) из примера 20.8.

**Пример 20.12**.

В качестве *ключа секционирования* для *индекса* возьмем колонку "Идентификатор покупателя" (s\_customer\_id). В фрагменте кода ниже для секций *индекса* используются другие индексные пространства с именами ts\_i\_01, ts\_i\_02, ts\_i\_03. Число секций *индекса* не совпадает с числом секций базовой таблицы для этого *индекса*.

CREATE INDEX sales\_ndx ON Sales (s\_customer\_id)

GLOBAL

PARTITION BY RANGE (s\_customer\_id)

(PARTITION st\_i\_q1 VALUES LESS THAN (10000)

TABLESPACE ts\_i\_01,

PARTITION st\_i\_q2 VALUES LESS THAN (20000)

TABLESPACE ts\_i\_02,

PARTITION st\_i\_q3 VALUES LESS THAN (MAXVALUE)

TABLESPACE ts\_i\_03,

);

Локально секционированный *индекс* может быть создан по колонке, отличной от *ключа секционирования* базовой таблицы *индекса*. В примере 20.13 создается такой непрефиксный *индекс* для таблицы "Продажи" (Sales) из примера 20.8.

**Пример 20.13**.

В качестве колонки *секционирования* для *индекса* взята колонка "Идентификатор покупателя" (s\_customer\_id), а для секций *индекса* выбраны другие *табличные пространства* ts\_i\_01, ts\_i\_02, ts\_i\_03, ts\_i\_04, отличающиеся от *табличных пространств* секций базовой таблицы *индекса*.

CREATE INDEX sales\_ndx\_1 ON Sales (s\_customer\_id)

LOCAL

(PARTITION st\_i\_q01 TABLESPACE ts\_i\_01,

PARTITION st\_i\_q02 TABLESPACE ts\_i\_02,

PARTITION st\_i\_q03 TABLESPACE ts\_i\_03,

PARTITION st\_i\_q04 TABLESPACE ts\_i\_04

);

При принятии решения о *секционировании индексов* следует иметь в виду следующее.

* Локальное префиксное *секционирование индекса* является наиболее эффективным методом *секционирования индекса*. Поскольку строки одной секции базовой таблицы будут индексироваться в одной секции *индекса*, СУБД не придется сканировать все секции при выборке данных по запросу.
* Локальное непрефиксное *секционирование индекса* требует от СУБД выполнения большего объема работы, так как для поиска данных требуется сканировать все секции *индекса*. Этот тип следует принимать во внимание при *параллельной обработке данных*.
* Глобальное префиксное *секционирование индекса* является наиболее эффективным методом *секционирования индекса* при обработке данных, когда необходимо сканирование диапазона. Этот тип *секционирования* группирует строки в одной секции, и СУБД знает, в какой секции искать значения из заданного диапазона.

**Секционирование представлений**

В СУБД Oracle есть возможность **секционировать представления**. Основная идея *секционирования представлений* проста. Пусть физическая таблица разбита на несколько таблиц (необязательно с помощью методов *секционирования* таблиц) в соответствии с критерием разбиения, который делает обработку запроса более производительной. Критерий разбиения будем называть **предикатом секционирования**. Тогда можно создать и настроить представления таким образом, чтобы с их помощью обращение к данным этих таблиц было проще для пользователя. Секция представления определяется в соответствии с диапазоном значений *ключа секционирования*. Запросы, которые используют диапазон значений для выборки данных из секций представления, будут получать доступ только к тем секциям, которые соответствуют диапазонам значений *ключа секционирования*.

Секции представления могут быть определены предикатами *секционирования*, заданными либо при помощи ограничения CHECK, либо с использованием предложения WHERE. Покажем, как могут быть применены оба приема, на примере несколько модифицированной таблицы "Продажи" (Sales), которую мы рассматривали в предыдущем разделе. Допустим, что данные о продажах для календарного года размещаются в четырех отдельных таблицах, каждая из которых соответствует кварталу года — Q1\_Sales, Q2\_Sales, Q3\_Sales и Q4\_Sales.

**Пример 20.14**.

*Секционирование представлений* с помощью ограничения CHECK. С помощью команды ALTER TABLE можно добавить ограничения на колонку "Дата продажи" (s\_date) каждой таблицы, чтобы ее строки соответствовали одному из кварталов года. Созданное затем представление sales дает возможность обращаться к этим таблицам, как к одной, так и ко всем вместе.

ALTER TABLE Q1\_Sales ADD CONSTRAINT C0 CHECK (s\_date BETWEEN 'jan-1-2002' AND 'mar-31-2002');

ALTER TABLE Q2\_Sales ADD CONSTRAINT C1 CHECK (s\_date BETWEEN 'apr 1-2002' AND 'jun-30-2002');

ALTER TABLE Q3\_Sales ADD CONSTRAINT C2 check (s\_date BETWEEN 'jul-1-2002' AND 'sep-30-2002');

ALTER TABLE Q4\_Sales ADD CONSTRAINT C3 check (s\_date BETWEEN 'oct-1-2002' AND 'dec-31-2002');

CREATE VIEW sales\_v AS

SELECT \* FROM Q1\_Sales UNION ALL

SELECT \* FROM Q2\_Sales UNION ALL

SELECT \* FROM Q3\_Sales UNION ALL

SELECT \* FROM Q4\_Sales;

Преимуществом такого *секционирования представлений* является то, что предикат ограничения CHECK не оценивается для каждой строки запроса. Такие предикаты исключают вставку в таблицы строк, не соответствующих критерию предиката. Строки, соответствующие предикату *секционирования*, извлекаются из базы данных быстрее.

**Пример 20.15**.

*Секционирование представлений* с помощью предложения WHERE. Создадим представление для тех же таблиц, что и в примере выше.

CREATE VIEW sales\_v AS

SELECT \* FROM Q1\_Sales WHERE s\_date BETWEEN 'jan-1-2002' AND 'mar-31-2002'

UNION ALL

SELECT \* FROM Q2\_Sales WHERE s\_date BETWEEN 'apr-1-2002' AND 'jun-30-2002'

UNION ALL

SELECT \* FROM Q3\_Sales WHERE s\_date BETWEEN 'jul-1-2002' AND 'sep-30-2002'

UNION ALL

SELECT \* FROM Q4\_Sales WHERE s\_date BETWEEN 'oct-1-2002' AND 'dec-31-2002';

Метод *секционирования представлений* с помощью предложения WHERE имеет некоторые недостатки. Во-первых, критерий *секционирования* проверяется во время выполнения для всех строк во всех секциях, которые охватываются запросом. Во-вторых, пользователи могут ошибочно вставить строку не в ту секцию, т.е. вставить строку, относящуюся к первому кварталу, в третий квартал, что приведет к неправильной выборке данных по этим кварталам.

У этого приема есть и достоинство по сравнению с использованием ограничения CHECK. Можно разместить секцию, соответствующую предикату WHERE, на удаленной базе данных. Фрагмент определения преставления приведен ниже.

SELECT \* FROM east\_sales@icp.ac.ru WHERE LOC = 'EAST'

UNION ALL

SELECT \* FROM west\_sales@ioc.ac.ru WHERE LOC = 'WEST';

Принимая решение о создании *секционированных представлений*, необходимо помнить о следующих факторах.

* *Секционирование представлений* позволяет операциям DML, таким как загрузка данных, создание *индексов* и удаление данных, работать на уровне секции, а не целой базовой таблицы.
* Доступ к одной из секций не оказывает никакого действия на данные в других секциях.
* СУБД Oracle обладает необходимыми встроенными возможностями для распознавания *секционированных представлений*.
* *Секционирование представлений* очень полезно при работе с таблицами, содержащими большое количество исторических данных.

**Секционирование таблиц в СУБД семейства MS SQL Server**

**Создание секционированных таблиц**

В СУБД семейства MS SQL Server также поддерживается *секционирование* таблиц, *индексов* и представлений. Однако, в отличие от СУБД семейства Oracle, *секционирование* в СУБД семейства MS SQL Server выполняется по унифицированной схеме.

В MS SQL Server все таблицы и *индексы* в БД считаются секционированными, даже если они состоят всего лишь из одной секции. Фактически, секции представляют собой базовую организационную единицу в физической архитектуре таблиц и *индексов*. Это означает, что логическая и физическая архитектура таблиц и *индексов*, включающая несколько секций, полностью отражает архитектуру таблиц и *индексов*, состоящих из одной секции.

*Секционирование* таблиц и *индексов* задается жестко на уровне строк ( *секционирование* по столбцам не допускается) и позволяет осуществлять доступ через единую точку входа (имя таблицы или имя *индекса* ) таким образом, что в коде приложения не требуется знать число секций. *Секционирование* может осуществляться на базовой таблице, а также на связанных с ней *индексах*.

Для создания *секционированной таблицы* в СУБД MS SQL Server используются следующие объекты БД: *функции секционирования* и *схемы секционирования*. Эти объекты позволяют разделять данные на конкретные сегменты и управлять их местоположением в БД или ХД. Например, можно распределить данные по нескольким дисковым массивам в зависимости от даты поступления данных или других отличительных признаков. Следует отметить, что таблицу можно секционировать по одному из ее столбцов, и каждая секция должна содержать данные, которые не могут храниться в других секциях.

**Функции секционирования**

При *секционировании* таблицы сначала нужно определить правило, по которому данные будут разделяться на сегменты. Для сопоставления отдельных строк данных с разными сегментами служит ***функция секционирования***.

Строки данных могут сегментироваться по колонке любого типа, кроме следующих: text, ntext, image, xml, timestamp, varchar(max), nvarchar(max), varbinary(max), псевдонимы типов данных и пользовательские типы данных среды CLR. Однако *функция секционирования* должна распределять каждую строку данных только в одну *секцию таблицы* ; иными словами, в результате применения функции одна и та же строка не может принадлежать нескольким секциям одновременно.

Чтобы секционировать таблицу, в ней необходимо создать или выбрать колонку *секционирования* ( *ключ секционирования* ). *Ключ секционирования* можно создать в схеме таблицы в момент создания таблицы либо добавить позднее путем модификации таблицы. Столбец может принимать значения NULL, но все строки, содержащие значения NULL, будут по умолчанию помещаться в самую левую *секцию таблицы*. Этого можно избежать, указав при создании *функции секционирования*, что значения NULL должны помещаться в самую правую *секцию таблицы*. Выбор левой или правой секций – важное решение проектирования, проявляющееся при изменении *схемы секционирования*, добавлении дополнительных секций или удалении существующих.

При создании *функции секционирования* можно выбрать функции LEFT или RIGHT. Разница между секциями LEFT и RIGHT состоит в размещении данных по секциям. Функция LEFT распределяет данные по принципу от самого низкого значения до самой высокой величины (то есть по возрастанию). Функция RIGHT распределяет данные по принципу от самого высокого значения до самого низкого (то есть по убыванию). Рассмотрим пример.

**Пример 20.16**.

Возьмем следующие примеры определения *функций секционирования* с использованием LEFT и RIGHT:

CREATE PARTITION FUNCTION Left\_Partition (int) AS RANGE LEFT

FOR VALUES (1,10,100)

CREATE PARTITION FUNCTION Right\_Partition (int) AS RANGE RIGHT

FOR VALUES (1,10,100)

В первой функции ( Left\_Partition ) значения 1, 10 и 100 размещаются соответственно в первой, второй и третьей секциях. Во второй функции ( Right\_Partition ) эти значения размещаются во второй, третьей и четвертой секциях.

При создании *секционированной таблицы* важно, чтобы секции получились сбалансированными по кардинальности. Это позволяет оценить, сколько дискового пространства потребуется для каждой секции. Использование параметров LEFT и RIGHT определяет, куда будут размещаться данные, что, в свою очередь, задает размер секции и размеры *индексов*, созданных на ней.

Определить номер секции, в которую попадут те или иные данные, можно с помощью функции $PARTITION, как показано ниже:

SELECT $PARTITION.Left\_Partition (10)

SELECT $PARTITION.Right\_Partition (10)

Первая команда SELECT возвращает значение 2, вторая – значение 3.

**Схемы секционирования**

После создания *функции секционирования* и выбора способа разделения данных по секциям следует решить, где будет создаваться каждая секция в файловой системе. Для создания связи секций с их размещением в файловой системе используются *схемы секционирования*. ***Схемы секционирования*** **управляют тем, каким образом отдельные секции хранятся на диске, путем использования файловых групп для размещения каждой секции на дисковой подсистеме**. *Схемы секционирования* можно настроить таким образом, чтобы все секции располагались в единой файловой группе, чтобы каждая секция располагалась в своей файловой группе или чтобы несколько секций использовали общие файловые группы. Последний способ дает администратору базы данных широкие возможности рассредоточения операций ввода-вывода на диске.

В примере 20.17 показаны некоторые из способов, позволяющих присвоить *схеме секционирования* одну или несколько файловых групп. Следует помнить, что файловые группы, используемые *схемой секционирования*, должны существовать в базе данных перед созданием схемы.

**Пример 20.17**.

Выполним присвоение файловых групп *схеме секционирования*. Сначала приведем пример размещения всех *секций таблицы* в одной файловой группе с именем **PRIMARY**.

CREATE PARTITION SCHEME Primary\_Left\_Scheme

AS PARTITION Left\_Partition

ALL TO ([PRIMARY])

Чтобы разместить все секции в различных файловых группах, нужно выполнить следующую команду:

CREATE PARTITION SCHEME Different\_Left\_Scheme

AS PARTITION Left\_Partition

TO (Filegroup1, Filegroup2, Filegroup3, Filegroup4)

Чтобы разместить несколько секций в различных файловых группах, нужно выполнить следующую команду:

CREATE PARTITION SCHEME Multiple\_Left\_Scheme

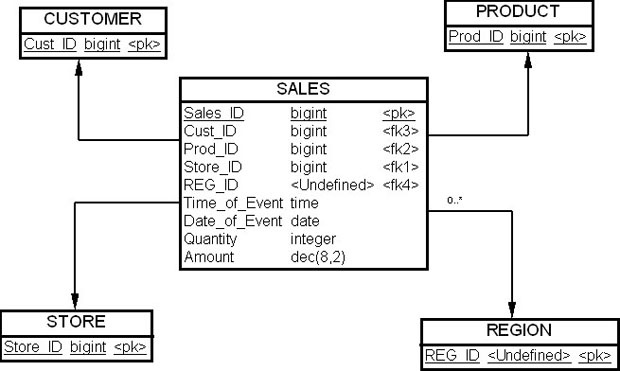
AS PARTITION Left\_Partition

TO (Filegroup1, Filegroup2, Filegroup1, Filegroup2)

Если создать указанные в примере *функции секционирования* и использовать приведенную *схему секционирования* для создания таблицы, то впоследствии можно будет определить, где будут размещаться отдельные строки данных в секционированных таблицах.

**Пример 20.18**.

Рассмотрим в качестве примера схему типа "звезда" с таблицей фактов "Продажи" (SALES), как показано на [рис. 20.6](https://intuit.ru/studies/courses/599/455/lecture/10182?page=4#image.20.6). Создадим *секционированную таблицу* "Продажи" (SALES).

[](https://intuit.ru/EDI/20_07_20_2/1595197216-9970/tutorial/632/objects/20/files/14_06.jpg)

[увеличить изображение](https://intuit.ru/EDI/20_07_20_2/1595197216-9970/tutorial/632/objects/20/files/14_06.jpg)  
**Рис. 20.6.**Таблица фактов "Продажи" (SALES) в схеме типа "звезда"

Сначала мы должны создать *функцию секционирования*:

CREATE PARTITION FUNCTION

MyPartitionFunctionLeft

(date)

AS RANGE LEFT

FOR VALUES ('1/01/2005', '1/01/2007', '1/01/2009)

MyPartitionFunctionLeft — это название функции разделения, datetime — тип данных *ключа секционирования*, а RANGE LEFT указывает, как делить значения данных, которые связаны с датами FOR VALUES.

*Ключ секционирования* имеет тип данных date, т.е. это колонка "Дата события" (Date\_of\_Event). В команде, приведенной выше, деление строк на непересекающиеся группы построено по принципу разбиения их на двухлетние группы. Разделение на секции RANGE LEFT делит данные в диапазонах значений, показанных на [рис. 20.7](https://intuit.ru/studies/courses/599/455/lecture/10182?page=4#image.20.7).

[Разделение на секции таблицы фактов "Продажи" (SALES) с использованием RANGE LEFT](https://intuit.ru/EDI/20_07_20_2/1595197216-9970/tutorial/632/objects/20/files/14_07.jpg)

[увеличить изображение](https://intuit.ru/EDI/20_07_20_2/1595197216-9970/tutorial/632/objects/20/files/14_07.jpg)  
**Рис. 20.7.**Разделение на секции таблицы фактов "Продажи" (SALES) с использованием RANGE LEFT

Каждая область значений в секции имеет границы, которые определены в операторе FOR VALUES. Если дата продажи была 23 июня 2006 года, то строка будет храниться в секции 2 (P2).

Теперь создадим *схему секционирования*. *Схема секционирования* отображает секции на различные файловые группы (с именами MyFilegroup1, MyFilegroup2, MyFilegroup3, MyFilegroup4 ) , как показано в следующей команде:

CREATE PARTITION SCHEME

MyPartitionScheme

AS MyPartitionFunction

TO (MyFilegroup1, MyFilegroup2, MyFilegroup3, MyFilegroup4)

MyPartitionScheme – это имя *схемы секционирования*, а имя MyPartitionFunction определяет *функцию секционирования*. Эта команда отображает данные в секции, которые связаны с одной или несколькими файловыми группами. Строки с данными со значениями колонки "Дата продажи" (Date\_of\_Event date) до 1/01/05 связаны с MyFilegroup1. Строки этой колонки со значениями, большими или равными 1/01/05 и до 1/01/07, назначены MyFilegroup2. Строки со значениями, большими или равными 1/01/07 и до момента 1/01/09, связаны с MyFilegroup3. Все остальные строки со значениями, большими или равными 1/01/09, связаны с MyFilegroup4.

Для каждого набора граничных значений (которые задаются условием FOR VALUES *функции секционирования* ) количество секций будет равно "Количество граничных значений" + 1 секция. Предыдущее предложение CREATE PARTITION SCHEME включает три ограничения и четыре секции. Независимо от того, созданы ли секции с RANGE RIGHT или RANGE LEFT, количество секций всегда будет равно "Количество граничных значений" + 1, вплоть до 1000 секций на таблицу.

Теперь мы можем создать *секционированную таблицу* фактов "Продажи" (SALES). Создание *секционированной таблицы* мало чем отличается от создания обычной таблицы, нужно только сослаться на имя *схемы секционирования* в условии ON, как показано в команде ниже.

CREATE TABLE SALES

(Sales\_ШВ bigint identity (1, 1) primary not clustered NOT NULL,

Cust\_ID bigint null,

Prod\_ID bigint null,

Store\_ID bigint null,

REG\_ID char(10) null,

Time\_of\_Event time null,

Quantity integer not null,

Amount dec(8,2) not null,

Date\_of\_Event date NOT NULL)

ON MyPartitionScheme (Date\_of\_Event)

Определяя имя *схемы секционирования*, проектировщик указывает, что эта таблица является *секционированной*. Очевидно, *схема секционирования*, *функция секционирования* и файловые группы должны существовать в БД до того, как будет создаваться таблица.

Можно объединять только две смежные секции. Чтобы слить две секции, выполните команду:

ALTER PARTITION FUNCTION

MyPartitionFunction()

MERGE RANGE ('1/01/2007')

Здесь секция 1 (P1) объединится с секцией P2. Это означает, что секция P2 будет содержать все строки со значением колонки "Дата продажи" (Date\_of\_Event) до значения даты 1/01/07. В системной таблице sys.partitions секции будут перенумерованы, начиная с единицы (не с нуля). Секции P1 и P2 станут P1, секция P3 станет P2 и P4 станет P3.

**Секционирование индексов в СУБД семейства MS SQL Server**

В СУБД семейства MS SQL Server предусмотрена возможность создавать секционированные *индексы*. Это позволяет проектировщику проектировать структуру *индекса* на основе разделенных данных, а не на основе всех данных таблицы. Создание секционированных *индексов* влечет за собой создание отдельных сбалансированных деревьев на секционированных *индексах*. В результате разделения *индексов* создаются *индексы* меньшего размера, и *администратору БД* или ХД становится проще их обслуживать во время изменения, добавления и удаления данных.

При создании секционированных *индексов* можно создавать **выровненные** или **невыровненные индексы**. Выровненные *индексы* подразумевают прямую связь с секционированными данными таблицы. В случае с невыровненными *индексами* выбирается другая *схема секционирования*.

Из этих двух методов предпочтителен выровненный *индекс*, который выбирается по умолчанию, если после создания *секционированной таблицы* *индексы* создаются без указания другой *схемы секционирования*. Использование выровненных *индексов* предоставляет необходимую гибкость для создания дополнительных секций в таблице, а также позволяет переводить принадлежность той или иной секции на другую таблицу. Для решения большинства задач, связанных с *секционированием*, достаточно применить для *индексов* *схему секционирования* таблицы.

**Пример. 20.19**.

Создадим секционированный *некластеризованный индекс* на *секционированной* таблице "Продажи" (SALES) из предыдущего примера 20.18. *Некластеризованный индекс* выравнивается с таблицей; в качестве ключа *некластеризованного индекса* используется *ключ секционирования* таблицы.

CREATE NONCLUSTERED INDEX cl\_multiple\_partition ON SALES

(Date\_of\_Event)

Для того чтобы создать невыровненный *некластеризованный индекс* на *секционированной* таблице "Продажи" (SALES) из примера 20.18, можно поступить следующим образом. Сначала создадим *функцию секционирования* для *индекса*.

CREATE PARTITION FUNCTION Index\_Left\_Partition (bigint) AS RANGE

LEFT

FOR VALUES (10, 50, 100)

Затем разместим все секции *индекса* в одной файловой группе с именем PRIMARY, выполним команду

CREATE PARTITION SCHEME Index\_primary\_Left\_Scheme

AS PARTITION Index\_Left\_Partition ALL TO ([PRIMARY])

Теперь выполним команду создания *индекса*, как показано ниже.

CREATE NONCLUSTERED INDEX cl\_multiple\_partition ON multiple\_partition(

Cust\_ID) ON Index\_primary\_Left\_Scheme (Cust\_ID)

В этом *некластеризованном индексе* в качестве ключа *индекса* используется колонка "Идентификатор покупателя" (Cust\_ID), которая не является *ключом секционирования* таблицы "Продажи" (SALES).

Решения о *секционировании индексов* принимаются проектировщиком ХД на стадии проектирования или администратором ХД на стадии эксплуатации ХД. Целью *секционирования индексов* является либо обеспечение производительности запросов, либо упрощение процедур сопровождения *индекса*.

#### Секционирование представлений в СУБД семейства MS SQL Server

Представление, или виртуальная таблица, которая предоставляет доступ к данным одной или более таблиц в СУБД MS SQL Server, создается командой CREATE VIEW, синтаксис которой приведен ниже.

CREATE VIEW [ schema\_name . ] view\_name [ (column [ ,...n ] ) ]

[ WITH <view\_attribute> [ ,...n ] ]

AS select\_statement

[ WITH CHECK OPTION ] [ ; ]

<view\_attribute> ::=

{

[ ENCRYPTION ]

[ SCHEMABINDING ]

[ VIEW\_METADATA ] }

schema\_name задает имя схемы, которой принадлежит представление, view\_name задает имя представления. Имена представлений должны соответствовать требованиям, предъявляемым к идентификаторам. Указывать имя владельца представления не обязательно.

column задает имя, которое будет иметь колонка в представлении. Имя колонки требуется только в тех случаях, когда колонка формируется на основе арифметического выражения, функции или константы, если две или более колонки могут по иной причине получить одинаковые имена (как правило, в результате соединения) или если колонке представления назначается имя, отличное от имени колонки, из которой она произведена. Назначать колонкам имена можно также в инструкции SELECT. Если column не указан, столбцам представления назначаются такие же имена, которые имеют столбцы в инструкции SELECT.

AS определяет действия, которые должны быть выполнены в представлении.

select\_statement задает команду SELECT, которая определяет представление. В этой команде можно указывать более одной таблицы и другие представления. Для выбора объектов, указанных в предложении SELECT создаваемого представления, необходимы соответствующие разрешения.

CHECK OPTION обеспечивает соответствие всех выполняемых для представления команд модификации данных критериям, заданным при помощи select\_statement. Если строка изменяется посредством представления, предложение WITH CHECK OPTION гарантирует, что после фиксации изменений доступ к данным из представления сохранится.

Атрибуты представления ENCRYPTION, SCHEMABINDING, VIEW\_METADATA обеспечивают дополнительные возможности управления представлением (см. техническую документацию по T-SQL).

Секционированное представление — это представление, определенное посредством объединения всех ( UNION ALL ) таблиц-элементов. Эти таблицы структурированы одинаковым образом, но хранимы отдельно в форме разных таблиц, либо в одном экземпляре SQL Server, либо в группе автономных экземпляров SQL Server, которые называются федеративными серверами баз данных.

При разработке *схемы секционирования* должно быть ясно, какие данные относятся к каждой секции. Например, данные таблицы "Покупатели" (Customers) распределены между тремя таблицами-элементами на трех серверах: таблицей "Покупатели 33" (Customers\_33) на сервере Server1, таблицей "Покупатели 66" (Customers\_66) на сервере Server2 и таблицей "Покупатели 99" (Customers\_99) на сервере Server3. Секционированное представление на сервере Server1 определяется, как в примере 20.20.

**Пример 20.20**.

Приведем определение секционированного представления на сервере Server1. Первая команда SELECT относится к таблице, размещенной на сервере Server1, вторая команда SELECT — к таблице, размещенной на сервере Server2, третья команда SELECT — к таблице, размещенной на сервере Server3.

CREATE VIEW Customers

AS

SELECT \* FROM CompanyData.dbo.Customers\_33

UNION ALL

SELECT \* FROM Server2.CompanyData.dbo.Customers\_66

UNION ALL

SELECT \* FROM Server3.CompanyData.dbo.Customers\_99

Как правило, представление считают *секционированным*, если оно соответствует следующему формату:

SELECT <select\_list1>

FROM T1

UNION ALL

SELECT <select\_list2>

FROM T2

UNION ALL

...

SELECT <select\_listn>

FROM Tn

*Секционированные представления* должны удовлетворять определенным требованиям.

1. Список выборки

В списке столбцов определения представления должны быть выбраны все колонки таблиц-элементов.

Колонки, занимающие одну и ту же порядковую позицию в каждом списке выбора, должны иметь одинаковый тип, включая параметры сортировки.

Ограничения должны быть такими, чтобы любое указанное значение <col> могло удовлетворять не более чем одному из ограничений C1, ..., Cn, т. е. они должны формировать совокупность неперекрывающихся интервалов. Колонка <col>, для которой определены неперекрывающиеся ограничения, называется **колонкой секционирования**. Обратите внимание, что колонка *секционирования* может иметь другие имена в базовых таблицах.

Одна колонка не может быть указана в списке выбора несколько раз.

1. Колонка *секционирования*

Колонка *секционирования* является частью первичного ключа ( PRIMARY KEY ) таблицы.

Колонка *секционирования* не может быть вычисляемой колонкой, колонкой-идентификатором, колонкой по умолчанию и колонкой временных меток (типа timestamp ).

Если для одной колонки таблицы-элемента определено более одного ограничения, ядро СУБД пропускает все ограничения и не учитывает их при определении того, является ли представление *секционированным*. Чтобы соответствовать требованиям к *секционированному представлению*, с колонкой *секционирования* должно быть связано только одно ограничение *секционирования*.

На возможность обновления колонки *секционирования* никакие ограничения не распространяются.

1. Таблицы-элементы или базовые таблицы T1, ..., Tn

Таблицы-элементы могут быть или локальными таблицами, или таблицами с других компьютеров, на которых выполняется SQL Server. Во втором случае для ссылки на таблицу должно быть использовано или четырехкомпонентное имя, или имя в формате функции OPENDATASOURCE или OPENROWSET (см. техническую документацию по T-SQL).

Если хотя бы одна таблица-элемент является удаленной, представление называется **распределенным секционированным представлением**, и тогда вступают в силу дополнительные требования. Они описаны ниже в данном разделе.

Одна таблица не может быть указана два раза в наборе таблиц, объединяемых при помощи инструкции UNION ALL.

Таблицы-элементы не могут иметь *индексы*, созданные для вычисляемых колонок в таблице.

Все *ограничения первичного ключа* ( PRIMARY KEY ), действующие в таблицах-элементах, должны быть связаны с одинаковым количеством колонок.

Всем таблицам-элементам в представлении должно быть назначено одинаковое значение заполнения ANSI. Его можно задать либо при помощи аргумента user options процедуры sp\_configure, либо при помощи инструкции SET.

При наличии таблиц-элементов и определения секционированного представления *оптимизатор запросов* MS SQL Server составляет планы эффективного выполнения запросов для доступа к данным из таблиц-элементов. При наличии определений ограничения CHECK обработчик запросов составляет карту распределения значений ключей по таблицам-элементам. Когда пользователь выполняет запрос, обработчик запросов сравнивает карту со значениями, указанными в предложении WHERE, и создает план выполнения, позволяющий свести к минимуму объем передачи данных между серверами-элементами. Следовательно, несмотря на то, что некоторые таблицы-элементы могут храниться на удаленных серверах, экземпляр MS SQL Server разрешает распределенные запросы таким образом, чтобы объем передаваемых распределенных данных оказался минимальным.

### Повышение производительности запросов: кластеры

#### Кластеризация таблиц

Самыми медленными операциями, выполняемыми СУБД, являются операции "чтение данных с диска" или "запись данных на диск". Если существует возможность уменьшить в несколько раз число таких операций, то общая производительность базы данных может заметно увеличиться.

Следует помнить, что СУБД считывает с диска или записывает на диск за один раз одну физическую страницу данных, размер которой колеблется, в зависимости от аппаратной платформы, от 512 б до 4 Кб. Таким образом, если можно физически хранить данные, к которым часто происходит совместное обращение, на одной и той же странице диска или на страницах, физически близко расположенных друг к другу, то скорость доступа к этим данных повышается.

***Кластеризация*** (Clustering) — **это способ физического размещения рядом, на одной физической странице данных, строк, доступ к которым осуществляется при помощи одинакового значения колонки (ключа) с целью увеличения производительности**. Такой ключ называется ***кластерным ключом***. Значением *кластерного ключа* являются значения одинаковых по смыслу колонок строк кластеризуемых таблиц. Ключ может быть либо хеш-ключом, либо индексным ключом.

Если ключ является хеш-ключом, то физическое размещение определяется функцией преобразования ключа (хеширования) и мы имеем дело с таблицей хеширования, или ***хеш-кластером***.

Если это индексный ключ, то для идентификации страницы данных в *кластере* используется *индекс со структурой B-Tree*, в котором строки, имеющие одинаковые значения ключа, размещаются либо в одной странице, либо в смежных страницах *индекса*. Такой *кластер* называется ***индексным кластером***. Строки, которые хранятся в *индексном кластере*, не обязательно должны принадлежать одной таблице.

Таким образом, *кластеры* являются одним из методов хранения таблиц данных, поддерживаемым СУБД. ***Кластер*** — **это группа таблиц, которая разделяет общие физические страницы данных при совместном использовании в запросах общих колонок этих таблиц**.

На практике *индексный кластер* создается для совместного хранения строк, связанных ограничением внешнего и первичного ключей. Совместное хранение строк родительской и *дочерней таблиц* может значительно ускорить выполнение соединения этих таблиц.

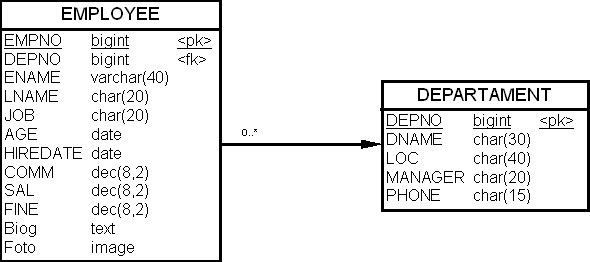
В СУБД семейства MS SQL Server *кластеры* не поддерживаются диалектом SQL. Все примеры, обсуждаемые ниже, ориентированы на использование СУБД семейства Oracle.

**Кластеризация таблиц в СУБД семейства Oracle**

В СУБД семейства Oracle поддерживаются различные типы *кластеризации* базовых таблиц. Рассмотрим использование *кластеризации* таблиц на примерах.

**Пример 20.21**.

Рассмотрим таблицы "Отдел" (DEPARTAMENT) и "Служащий" (EMPLOYEE), структура которых показана на [рис. 20.8](https://intuit.ru/studies/courses/599/455/lecture/10182?page=6#image.20.8).



**Рис. 20.8.**Таблицы "Отдел" (DEPARTAMENT) и "Служащий" (EMPLOYEE)

Таблица "Служащий" (EMPLOYEE) описана в примере 20.1 настоящей лекции, а описание таблицы "Отдел" (DEPARTAMENT) приведено в [табл. 20.2](https://intuit.ru/studies/courses/599/455/lecture/10182?page=6#table.20.2) ниже.

|  |  |  |
| --- | --- | --- |
| Таблица 20.2. Описание полей таблицы "Отдел" (DEPARTAMENT) | | |
| **№** | **Наименование атрибута** | **Наименование колонки** |
| 1 | Номер подразделения | DEPNO (PK) |
| 2 | Название отдела | DNAME |
| 3 | Месторасположение | *LOC* |
| 4 | Руководитель | MANAGER |
| 5 | Телефон | PHONE |

Обе таблицы некластеризованы и хранятся каждая на своих физических страницах. Предположим, что анализ запросов показывает, что для 80% запросов эти таблицы используются совместно, при этом соединение выполняется по колонке "Номер подразделения" (DEPNO). Можно построить *кластер* для этих двух таблиц.

*До кластеризации* таблицы сохраняются отдельно в своих физических областях на диске.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| DEPARTAMENT | | | | | | | |
| DEPNO | DNAME | | | *LOC* | | | …. |
| 10 | Торговля | | | Москва | | |  |
| 20 | Консалтинг | | | Черноголовка | | |  |
| EMPLOYEE | | | | | |
| EMPNO | | ENAME | LNAME | | DEPNO |
| 996 | | Козырев | Сергей | | 10 |
| 997 | | Сапегин | Алексей | | 20 |

*После кластеризации* по колонке "Номер отдела" (DEPNO) строки таблиц будут сохраняться совместно, разделяя одни и те же физические страницы базы данных.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CLUSTER | | | | |
| DEPNO |  | | | |
| 10 | DNAME | *LOC* | …. |  |
|  | Торговля | Москва | …. |  |
|  | …. | …. | …. |  |
|  | EMPNO | ENAME | LNAME | …. |
|  | 996 | Козырев | Сергей | …. |
|  | …. | …. | …. |  |
| 20 | DNAME | *LOC* | …. |  |
|  | Консалтинг | Черноголовка | …. |  |
|  | EMPNO | ENAME | LNAME | …. |
|  | 997 | Сапегин | Алексей | …. |
|  | … | …. | …. | …. |

Из примера видно, что при соединении таблиц число операций ввода-вывода при доступе к *кластеру* будет меньше. Также видно, что значение *кластерного ключа* сохраняется только один раз в *кластере* и/или кластерном *индексе*, независимо от того, сколько строк различных таблиц содержат это значение.

*Кластеризация* может существенно ускорить работу с соединениями. Однако, планируя использование *кластеров*, следует учитывать следующие факторы.

* *Кластеризация* затрагивает физическое размещение данных в файлах БД. Поэтому рекомендуется кластеризовать таблицу только по одной колонке, реже по комбинации колонок.
* *Кластеризация* замедляет выполнение операций, в которых нужно просканировать всю таблицу, так как она может вызвать разброс строк одной таблицы по множеству физических страниц.
* *Кластеризация* может замедлить ввод данных.
* *Кластеризация* может замедлить модификацию данных в тех колонках, которые помещены в *кластер*.

В силу вышеперечисленных причин *кластеры* не рекомендуется создавать для таблиц с интенсивным обновлением данных. Для того чтобы таблица была хорошим кандидатом для *кластеризации*, должны выполняться, по крайней мере, следующие условия:

* значения колонок *кластерных ключей* распределены равномерно и плотно, а их размер почти всегда меньше размера физической страницы (иначе будут образовываться кластерные цепочки);
* в случае *индексного кластера* на каждый *кластерный ключ* приходится больше одной выбираемой строки, а для *хеш-кластера* — одна строка. Альтернативное решение — индексация таблицы;
* все данные для заданного *кластерного ключа* выбираются при каждом доступе по *кластерному ключу*. Альтернативное решение — индексация таблицы;
* интенсивность обращений операций вставки, обновления и удаления не очень велика, иначе общая производительность базы данных может уменьшиться (обратный отрицательный эффект).

Из этого следует, что существуют две основные причины использования *кластеров*: а) необходимость обеспечить прямой доступ к строке за одну операцию чтения, и б) необходимость сократить число операций ввода-вывода при доступе к часто используемым совместно данным путем размещения их в близко расположенных физических страницах БД.

С физической точки зрения, *кластер* находится отдельно от таблиц. Он создается с указанием параметров хранения, а затем в нем последовательно создаются кластеризованные таблицы. При описании *кластера* нужно указать колонки или колонку, для которых СУБД сформирует *кластер*, и таблицы, которые будут включены в его состав. При обработке данных СУБД будет размещать строки, содержащие одинаковые значения в колонках *кластера*, физически максимально близко. В результате строки таблицы могут быть распределены среди нескольких дисковых страниц, но первичные и внешние ключи обычно располагаются на одной странице.

**Пример 20.22**.

Создадим *кластер* для таблиц "Отдел" (DEPARTAMENT) и "Служащий" (EMPLOYEE). Для создания *кластеров* используется команда SQL CREATE CLUSTER, которая в нашем случае будет иметь вид:

CREATE CLUSTER emp\_dept\_c (DEPNO integer)

SIZE 512,

-- TABLESPACE …

-- STORAGE …

INDEX;

CREATE TABLE DEPARTAMENT

(

DEPNO integer NOT NULL,

DNAME char(20),

LOC char(30),

MANAGER char(20),

PHONE char(15),

CLUSTER emp\_dept\_c (DEPNO)

);

CREATE TABLE EMPLOYEE

(

EMPNO integer NOT NULL,

ENAME char(40),

LNAME char(20),

DEPNO int NOT NULL,,

JOB char(20),

AGE date,

HIREDATE date NOT NULL WITH DEFAULT,

SAL dec(8,2),

COMM dec(8,2),

FINE dec(8,2),

Biog text,

Foto image,

PRIMARY KEY (EMPNO),

CLUSTER emp\_dept\_c (DEPNO)

);

CREATE INDEX emp\_dept\_c\_id ON CLUSTER emp\_dept\_c;

Параметр SIZE определяет размер *кластера* — фактически, максимальное число *кластерных ключей*, которые могут сохраняться на одной физической странице данных. Ключевое слово INDEX означает, что создаваемый *кластер* является *индексным*.

Предложение CLUSTER emp\_dept\_c (DEPNO) указывает СУБД, что таблица должна быть добавлена в *кластер*. Обратите внимание, что в таблице "Отдел" (DEPARTAMENT) снято *ограничение первичного ключа* на колонку "Номер подразделения" (DEPNO). Это связано с тем, что Oracle автоматически создает *индекс* на первичный ключ, а этот *индекс* в данном случае не нужен. Последнее предложение создает кластерный *индекс* для *кластерного ключа* *кластера*.

В одном из предыдущих разделов мы уже обсуждали вопрос использования таблиц хеширования, которые представляют на самом деле один из способов реализации *кластера*. Рассмотрим теперь подробно, как работать с *хеш-кластером* СУБД Oracle.

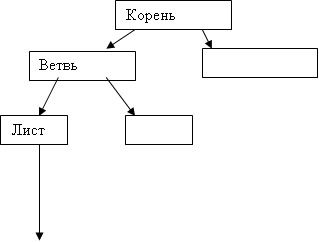
Напомним, что хеширование является способом хранения таблиц данных для увеличения производительности выборки. Физическим механизмом реализации хеширования в СУБД Oracle является *хеш-кластер*. Данные выбираются из *хеш-кластера* в соответствии с функцией хеширования, которая используется для генерации распределения значений ключа таблицы в числовые значения, определяющие физические страницы БД. *Хеш-кластер* является альтернативной техникой создания таблиц данных по отношению к *индексному кластеру* или некластеризованной таблице.

**Пример 20.23**.

Рассмотрим нашу учебную базу данных с целью создания *хеш-кластера* для таблицы "Служащий" (EMPLOYEE). Запрос на доступ к записям таблицы до и после *кластеризации* приведен ниже.

SELECT \* FROM EMPLOYEE WHERE EMPNO= 997;

*До кластеризации* по колонке "Номер служащего" (EPMNO) доступ будет выполняться через *индекс*, и, согласно [рис. 20.9](https://intuit.ru/studies/courses/599/455/lecture/10182?page=6#image.20.9), потребуется 4 операции ввода-вывода, чтобы получить результирующую строку.



**Рис. 20.9.**Доступ через индекс

*После кластеризации* по колонке "Номер служащего" (EPMNO) строки таблицы "Служащий" (EMPLOYEE) будут сохраняться в структуре, которая условно приведена на рисунке ниже. После хеширования ключа потребуется одна операция ввода-вывода, чтобы получить результирующую строку, если нет цепочек переполнения.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CLUSTER | | | | |
| Хеш-ключ | *Кластерный ключ* |  | | |
| 110 | EMPNO | ENAME | LNAME | …. |
|  | 996 | Козырев | Сергей | …. |
|  | …. | …. | …. |  |
| 120 | EMPNO | ENAME | LNAME | …. |
|  | 997 | Сапегин | Алексей | …. |

В *хеш-кластере* связанные строки (в данном случае имеющие одинаковое значение хеш-ключа) сохраняются также на одной физической странице базы данных на основе хешированных значений их общего ключа. В индексированных же таблицах и *индексном кластере* локализация результирующей строки выполняется с использованием значений ключа, которые хранятся в отдельном *индексе* (см. пример выше CREATE INDEX для кластерного *индекса* ).

**Пример 20.24**.

Создадим *хеш-кластер* для таблицы "Служащий" (EMPLOYEE) примера 20.21.

CREATE CLUSTER PERSONNEL (EMPNO integer)

SIZE 512

HASHKEYS 500

-- STORAGE (INITIAL 100K NEXT 50K PCTINCREASE 10)

;

Число уникальных значений хеш-ключа задается параметром HASHKEYS, после достижения этого значения в таблице будут возникать *коллизии* — ситуации, когда разные хешированные ключи должны будут размешаться в одном блоке. Это приводит к созданию при вставке строк так называемых *цепочек переполнения*, наличие которых приводит к увеличению числа доступов при выборке результирующей строки.

Параметр SIZE определяет максимальное число хеш-ключей, размещаемое на физической странице базы данных. Он равен оценке общего пространства в байтах, требуемого для сохранения среднего числа строк, которое связано с каждым значением хеш-ключа. Если доступного пространства на странице 1600 байт, а значение параметра –512 байт, то три значения хеш-ключа будут распределяться на физической странице.

С помощью предложения HASH IS можно переопределить хеш-функцию, которую СУБД Oracle использует по умолчанию.

**Пример 20.25**.

Если у нас есть *хеш-кластер* для таблицы **EMPLOYEE** и *кластерный ключ* определен как "Код домашнего адреса сотрудника" ( home\_area\_code number ), то вероятно, что будет случаться много коллизий в *хеш-кластере*, если город, где живут сотрудники, невелик. Для того чтобы избежать такой коллизии, можно переопределить встроенную хеш-функцию СУБД Oracle в команде CREATE CLUSTER, добавив предложение HASH IS:

CREATE CLUSTER personnel

(home\_area\_code number,

home\_prefix number )

HASHKEYS 20

HASH IS MOD(home\_area\_code + home\_suffix\_tel, 101);

В примере добавлено некоторое число к коду домашнего адреса, чтобы изменить распределение значений хеш-ключа с целью избежать коллизий. В качестве такого числа взяты две последние цифры домашнего телефона.

В заключение обсуждения *кластеризации* отметим следующее. Несмотря на то, что СУБД Oracle интенсивно применяет *кластеры* для доступа к системным таблицам БД, авторы рекомендуют проявлять осторожность при принятии решения о *кластеризации* таблиц при создании новой БД. Выигрыш в производительности может быть не слишком высок по сравнению с другими проектными решениями. Проектирование *кластеров* — "штучная" работа. Очень полезно знать статистику использования аналогичного *кластера* при эксплуатации аналогичного ХД, чтобы построить высокопроизводительный *кластер*. Следует придерживаться следующего эмпирического правила:

1. До 1000 записей – СУБД не имеет больших преимуществ перед последовательным файлом.
2. От 1000 до 10000 записей – это преимущество незначительно.
3. От 10000 до 100000 записей – между настольными и промышленными СУБД не ощущается разницы в производительности.
4. От 100000 до 1000000 записей – промышленные СУБД обеспечивают приемлемую производительность без специальных способов ее повышения.
5. От 1000000 записей – надо начинать думать о повышении производительности.

**Резюме**

В настоящей лекции мы рассмотрели три основных метода повышения производительности запросов, которые используют встроенные в *СУБД* *механизмы* и объекты: *индексы*, секции и *кластеры*.

Основная идея *индексирования* таблиц состоит в создании специальных объектов *СУБД* – *индексов* для повышения производительности выборки строк таблиц в запросах. Как правило, для первичных ключей таблиц *СУБД* создает *индексы* автоматически.

При создании дополнительных *индексов* следует помнить о том, что *индексы* требуют от *СУБД* дополнительных усилий для их поддержки и обслуживания, занимают дополнительное *место* в файловой системе и не всегда используются оптимизатором запросов при обработке команды SELECT.

Основная идея *секционирования* таблиц — с помощью встроенных команд *СУБД* разбить таблицы большого объема на ряд физических фрагментов в соответствии с некоторым критерием *секционирования*, чтобы сократить объем ввода-вывода при обработке фрагментов. *Секционирование* очень часто применяется при работе с таблицами большого объема. Если в проектируемой базе данных предполагается наличие объектов большого объема (более 1 Гб), обязательно следует рассмотреть возможность использования техники *секционирования*.

Решение вопроса, стоит ли применять *секционирование*, в основном зависит от того, насколько велика *таблица* или насколько она может увеличиться, как она используется и насколько эффективно отвечает на пользовательские запросы и *операции* обслуживания.

В целом, большую таблицу стоит секционировать, если выполняются следующие два условия:

* таблица содержит (или может в будущем накопить) множество данных, используемых различными способами;
* запросы или обновления таблицы выполняются не так, как ожидалось, либо затраты на обслуживание превышают прогнозируемые периоды технического обслуживания.

При разбиении таблиц на секции следует помнить о том, что секции требуют от *СУБД* дополнительных усилий для их поддержки и обслуживания.

Основная идея *кластеризации* таблиц состоит в том, чтобы хранить совместно используемые в запросах колонки различных таблиц на общих физических страницах дискового пространства файловой структуры с целью повышения производительности обработки запросов.

Однако нужно помнить, что проектирование *кластеров* – это "штучная" работа. Выигрыш в производительности может быть не слишком высок по сравнению с другими проектными решениями. Чтобы построить высокопроизводительный *кластер* для таблиц, полезно знать статистику использования аналогичного *кластера* при эксплуатации аналогичного ХД, а это не всегда бывает известно.