Индексы

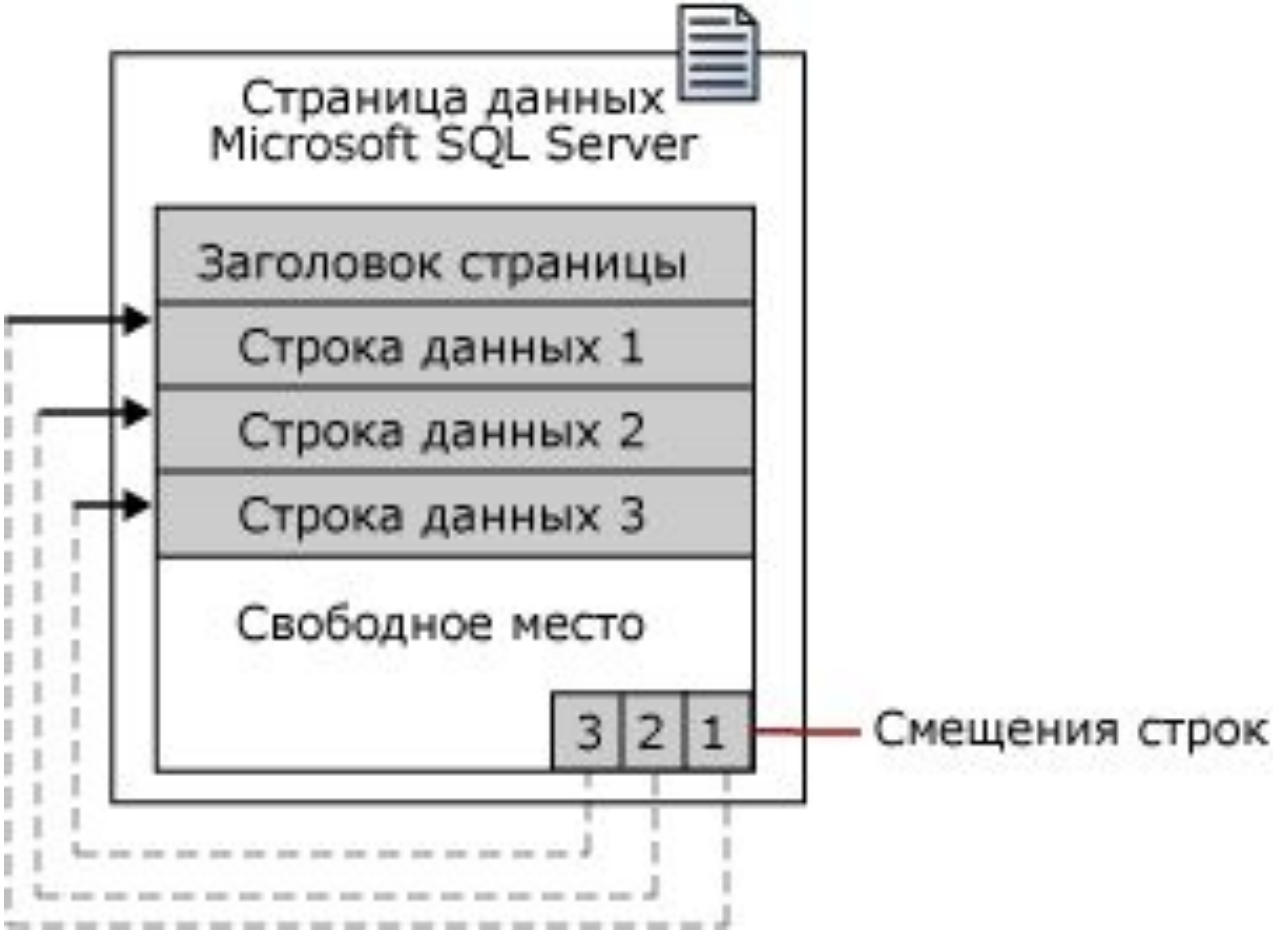
Физическое хранение данных

- Основной единицей хранилища данных в SQL Server является страница. Место на диске для размещения файла данных в базе данных, логически разделяется на страницы с непрерывным перечислением от 0 до n.
- Дисковые операции ввода-вывода выполняются на уровне страницы. SQL Server считывает или записывает целые страницы данных.

Страницы и экстенты

- В SQL Server размер страницы составляет 8 КБ. 1 МБ = 128 страниц.
- Заголовок 96 Б для хранения системных данных о странице (номер страницы, тип страницы, объем свободного места на странице и идентификатор объекта, которому принадлежит страница).
- Экстент — это коллекция, состоящая из восьми физически непрерывных страниц; они используются для эффективного управления страницами. Все страницы хранятся в экстентах.

Схема блока с таблицей смещения записей



Поддержка больших строк

- Желательно, чтобы строка целиком хранилась в одной странице (IN_ROW_DATA)
- Часть очень большой строки может быть перемещена на другую страницу.
- Длина строки на странице < =8 060 байт (без учета данных «Текст/изображение»).
- Может быть больше для таблиц, содержащих столбцы varchar, nvarchar, varbinary и пр. (varchar (max) до 2 ГБ)

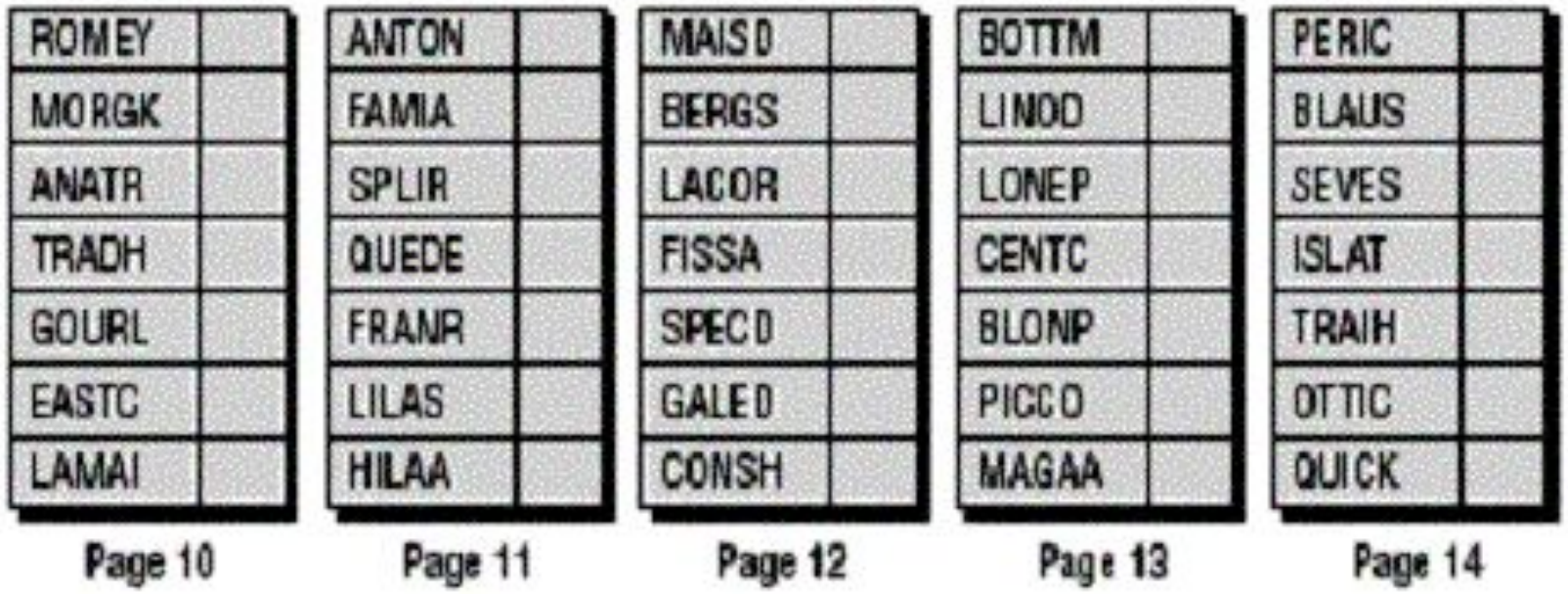
Длина строки > 8 060 байт

- SQL Server динамически перемещает один или более столбцов переменной длины на страницы в единице распределения (ROW_OVERFLOW_DATA), начиная со столбца наибольшей длины. Если потом размер строки уменьшается, SQL Server динамически перемещает столбцы обратно на исходную страницу данных.
- Поиск в неупорядоченном файле – в среднем половина файла m/2.

Типы запросов

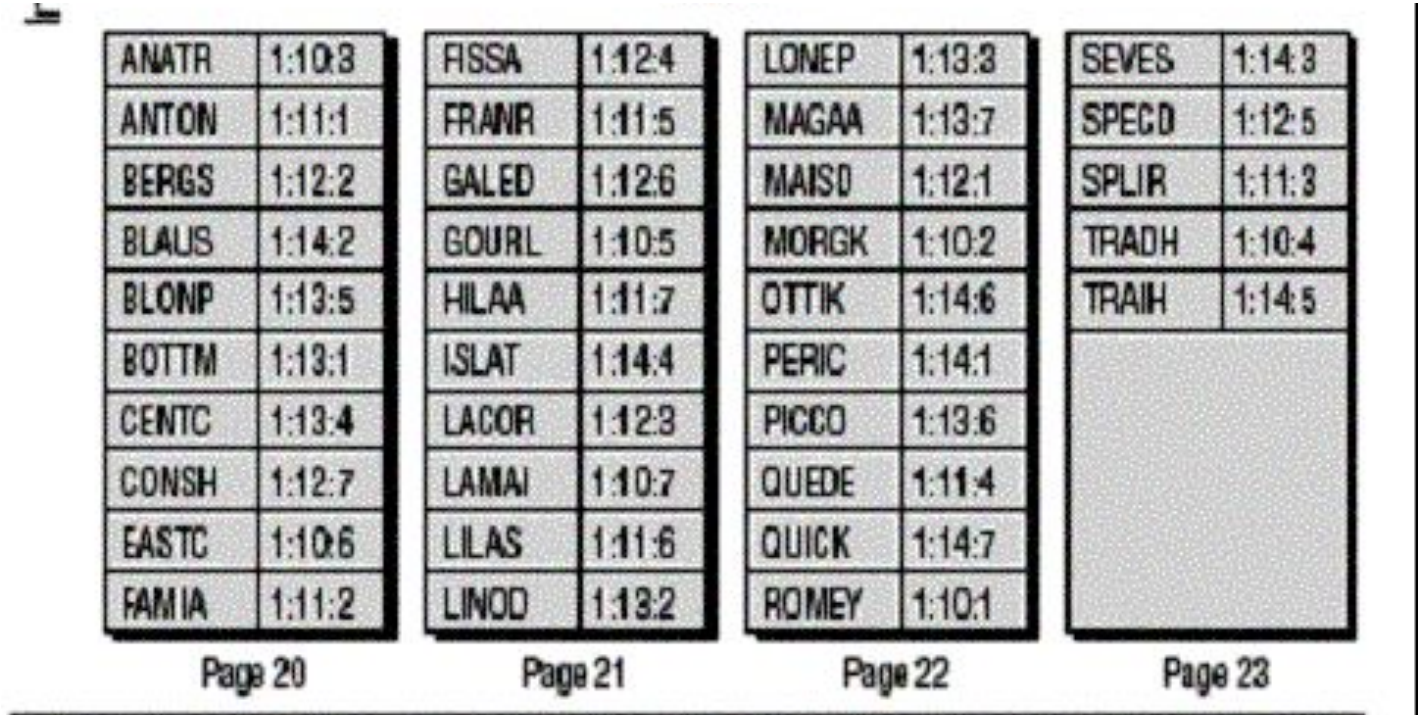
- Точечный запрос - результат 1 запись.
- Набор из нескольких записей, относительно небольшое их количество.
- Ранговые запросы, где в качестве критериев обычно указывается диапазон неких значений.
- Минимумы-максимумы, группировки, сортировки.

Heap (куча)
сканирование таблицы (full scan)



Таблица, в которой записи упорядочены по значению ключа

- Полезны при частых интервальных запросах
- $\log_2 m$



ANATR	1:10:3
ANTON	1:11:1
BERGS	1:12:2
BLAUS	1:14:2
BLONP	1:13:5
BOTTM	1:13:1
CENTC	1:13:4
CONSH	1:12:7
EASTC	1:10:6
FAMIA	1:11:2

Page 20

FISSA	1:12:4
FRANR	1:11:5
GALED	1:12:6
GOURL	1:10:5
HILAA	1:11:7
ISLAT	1:14:4
LACOR	1:12:3
LAMAI	1:10:7
LILAS	1:11:6
LINOD	1:13:2

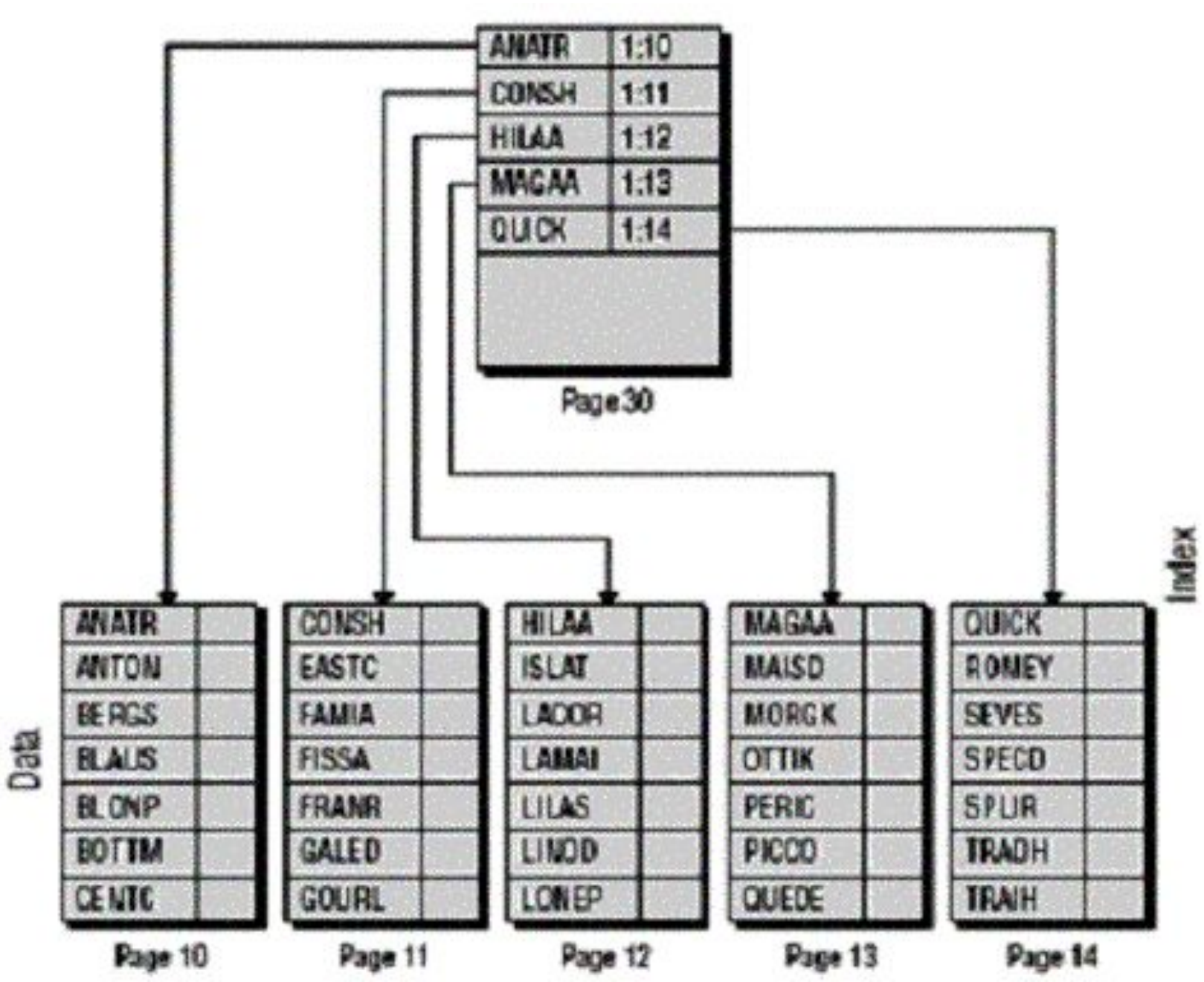
Page 21

LONEP	1:13:3
MAGAA	1:13:7
MAISD	1:12:1
MORGK	1:10:2
OTTIK	1:14:6
PERIC	1:14:1
PICCO	1:13:6
QUEDE	1:11:4
QUICK	1:14:7
ROMEY	1:10:1

Page 22

SEVES	1:14:3
SPECD	1:12:5
SPLIR	1:11:3
TRADH	1:10:4
TRAIH	1:14:5

Page 23


ANATR 1:10
CONSH 1:11
HILAA 1:12
MAGAA 1:13
QUICK 1:14
Page 30
Index
Data
ANATR
ANTON
BERGS
BLAUS
BLONP
BOTTM
CENTC
Page 10
CONSH
EASTC
FAMIA
FISSA
FRANR
GALED
GOURL
Page 11
HILAA
ISLAT
LACOR
LAMAI
LILAS
LINOD
LONEP
Page 12
MAGAA
MAISD
MORGK
OTTIK
PERIC
PICCO
QUEDE
Page 13
QUICK
ROMEY
SEVES
SPECD
SPLIR
TRADH
TRAIH
Page 14

B-дерево

- Таблица упорядочена по значению ключа
- Для каждого блока данных определяем пару: минимальное значение ключа и адрес блока.
- Эти пары также размещаем в блоках.
- С новыми блоками поступаем так же, наращивая уровни, пока не появится уровень из одного блока.

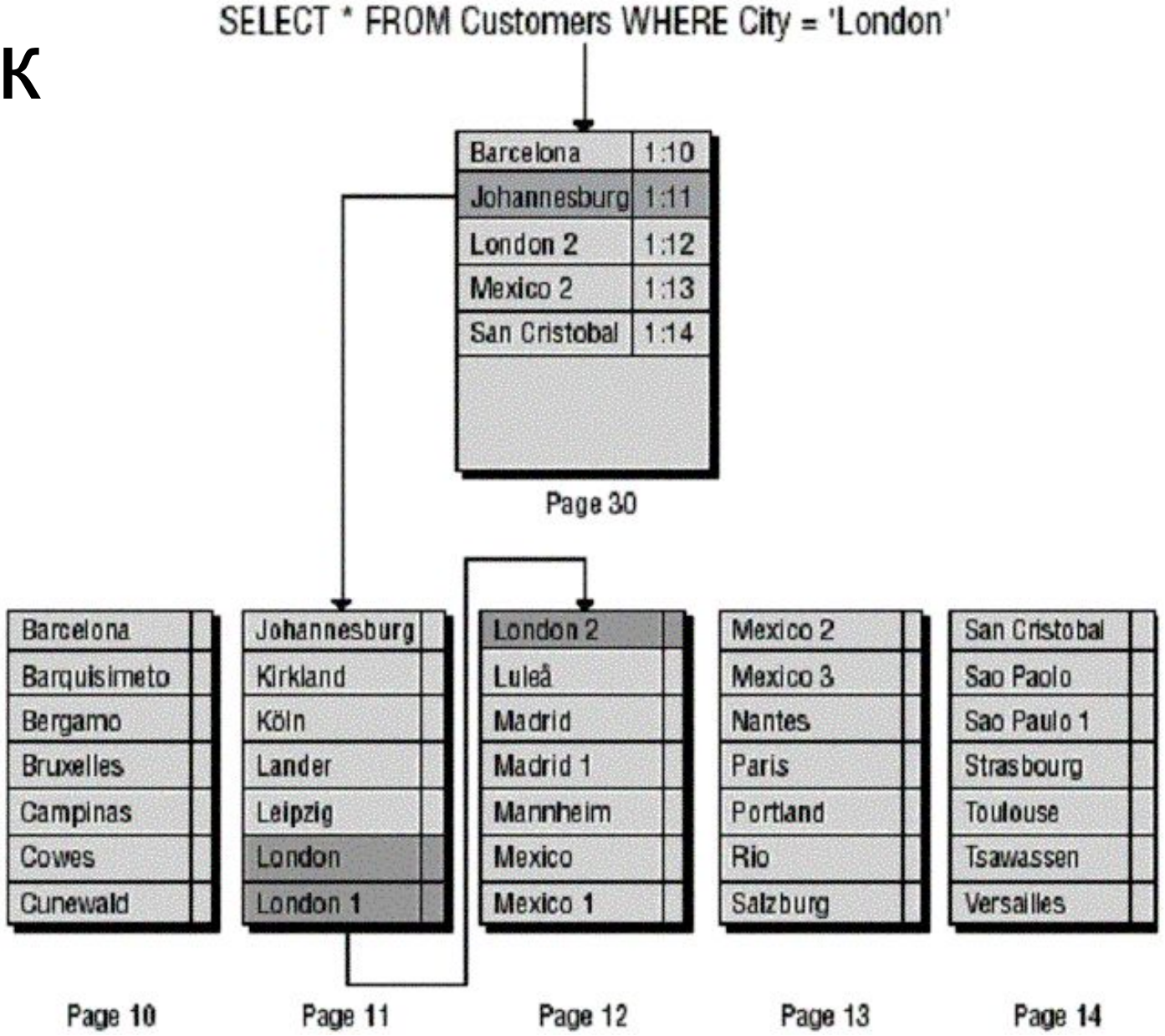
B-дерево

- Имеет внутренние (индексные) и листовые страницы
- Листовые вершины находятся на самом нижнем уровне дерева, все остальные – внутренние (индексные)
- Индексные вершины содержат пары (key, adr) , где key – минимальное значение ключа в блоке adr.

B-дерево

- B-дерево – сбалансированная структура, т.е. от корня до любой листовой страницы одинаковое число шагов
- Высота B-дерева - $\log_m N$
- Листовые страницы могут быть связаны одно- или двунаправленным списком.

Поиск


SELECT * FROM Customers WHERE City = 'London'
Barcelona 1:10
Johannesburg 1:11
London 2 1:12
Mexico 2 1:13
San Cristobal 1:14
Page 30
Barcelona
Barquisimeto
Bergamo
Bruxelles
Campinas
Cowes
Cunewald
Page 10
Johannesburg
Kirkland
Köln
Lander
Leipzig
London
London 1
Page 11
London 2
Luleå
Madrid
Madrid 1
Mannheim
Mexico
Mexico 1
Page 12
Mexico 2
Mexico 3
Nantes
Paris
Portland
Rio
Salzburg
Page 13
San Cristobal
Sao Paolo
Sao Paulo 1
Strasbourg
Toulouse
Tsawassen
Versailles
Page 14

Вставка в B-дерево

- Производим поиск по значению вставляемого ключа.
- Если в блоке есть место, то добавляем. Иначе создаем новый блок, а записи старого распределяем поровну в два блока.
- Так же поступаем со всеми уровнями.

Индекс

- Избыточная структура, предназначенная для ускорения поиска.

Основное назначение:

- увеличение скорости доступа к данным
- поддержка уникальности данных

Поиск с помощью индекса:

- На точное значение
- На интервал
- На значение нескольких атрибутов

Примеры предикатов без использования индекса

- WHERE IdNum + 1 = 101
- WHERE ABS(IdNum) = 100
- WHERE datepart(year,Date_beg)=2014
- WHERE Name LIKE ‘%ва%’
- WHERE DATEADD(DAY,7,Date_beg)>GETDATE()

Исправленные примеры предикатов с использованием индекса

- WHERE IdNum = 100
- WHERE IdNum IN (-100, 100)
- WHERE Date_beg > ‘2013-12-31’ and Date_beg < ‘2015-01-01’
- WHERE Name =N‘Иванов’
- WHERE Date_beg<DATEADD(DAY,-7,GETDATE())

Способы определения индекса:

- автоматическое создание индекса при создании первичного ключа;
- автоматическое создание индекса при определении ограничения целостности UNIQUE;
- создание индекса с помощью команды CREATE INDEX.

Создание индекса

```
CREATE [ UNIQUE ]
[ CLUSTERED | NONCLUSTERED ]
INDEX index_name
    ON <object> ( column [ ASC | DESC ] [ ,...n ] )
    [ INCLUDE ( column_name [ ,...n ] ) ]
    [ WHERE <filter_predicate> ]
    [ WITH ( <relational_index_option> [ ,...n ] ) ]
```

Характеристики индекса

- кластеризованный или некластеризованный;
- уникальный или неуникальный;
- с одним или несколькими столбцами;
- порядок по возрастанию или по убыванию в столбцах индекса;
- может содержать включенные столбцы;
- полнотабличные или фильтруемые некластеризованные индексы.

CLUSTERED

- Использует возможность физического индексирования данных
- В результате будут отсортированы данные в самой таблице согласно порядку этого индекса.
- Добавление информации в таблицу приводит к изменению физического порядка данных.
- Кластерным может быть только один индекс в таблице.

Кластерный индекс

- Кластерный индекс обеспечивает самый быстрый поиск по заданному ключу
- Столбцы типа ntext, text, image, varchar(max), nvarchar(max) и varbinary(max) нельзя указывать в качестве ключевых столбцов индекса.
- Длина полей, составляющих ключ, обратно пропорциональна скорости поиска.

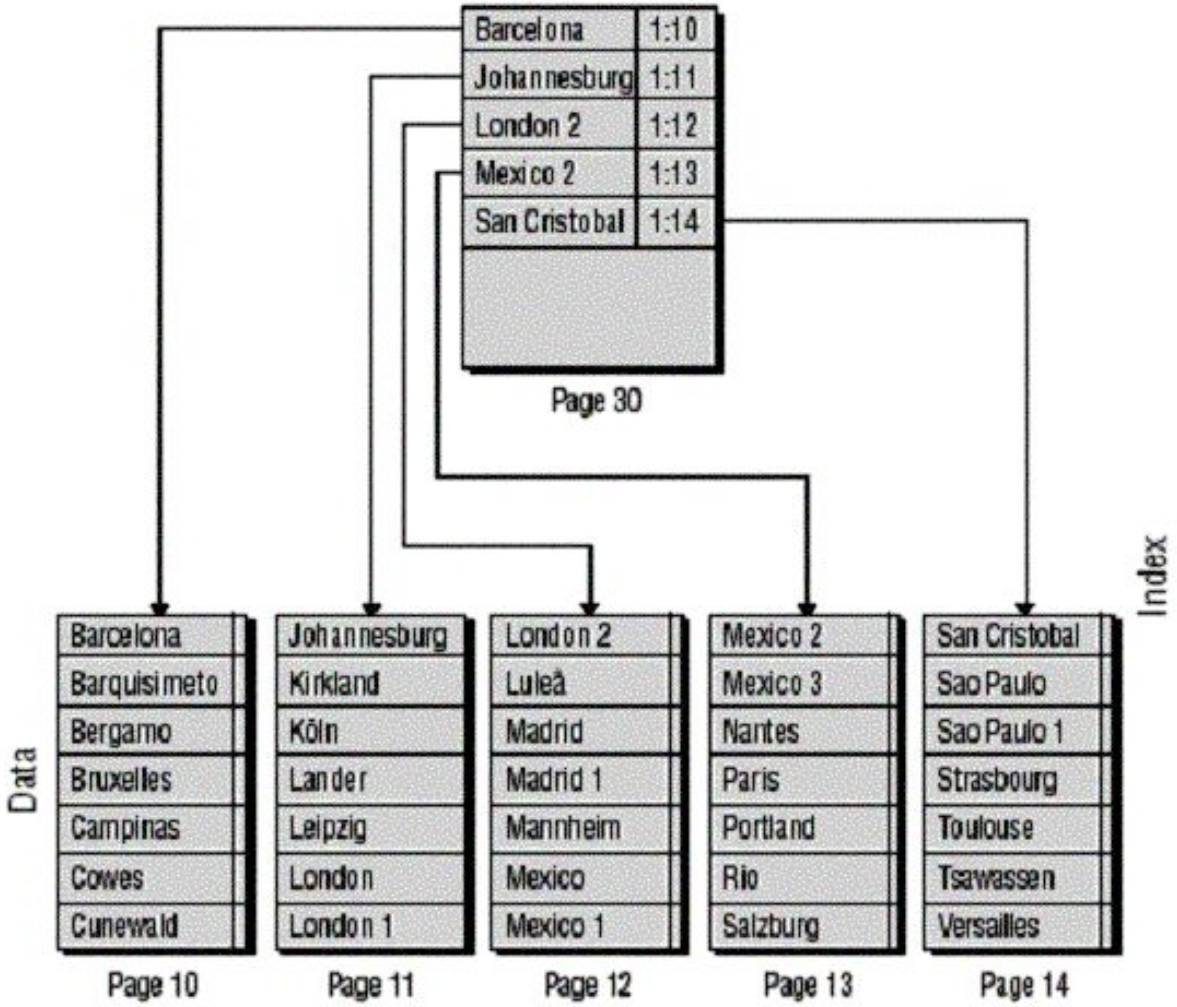
UNIQUE

- Используется при необходимости ввода в определенное поле только уникальных значений.
- В индексируемом столбце желательно запретить хранение значений NULL.

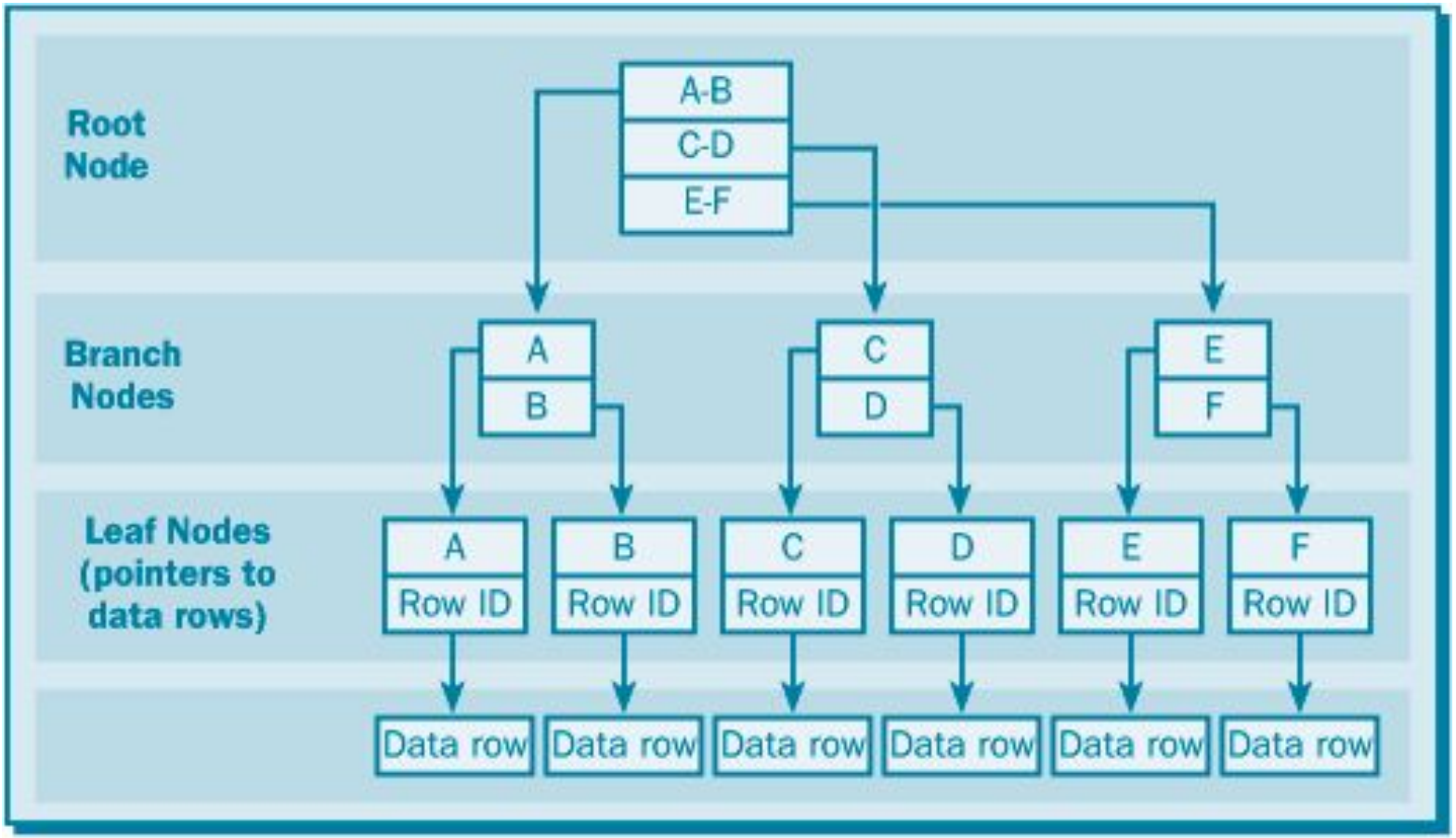
Ключ кластерного индекса: желательно

- Уникальный
- Узкий (как можно меньше байт)
- Статичный (редко меняется)

Кластерные индекс для неуникальных значений



Кластерный индекс



Некластерный индекс

- Строим по тем полям, которые часто используются при поиске.
- Таблицы могут содержать до 249 некластерных индексов.
- Не создавайте индексы по столбцам с низкой избирательностью (selectivity) (пол, дни недели и т.д.)

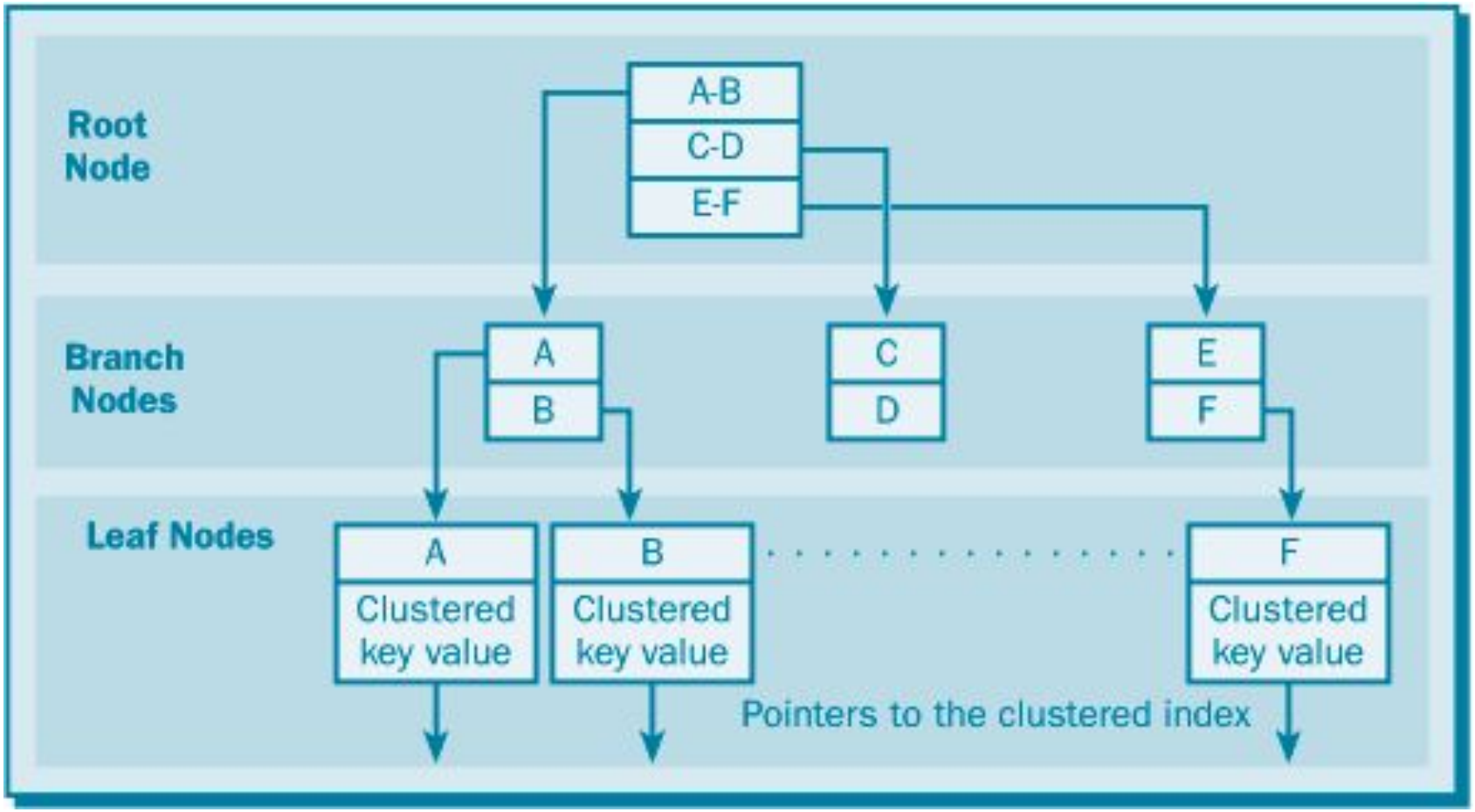
Некластерный индекс

- В индексный файл для каждой записи помещаем пару: значение ключа +
  - адрес записи, если нет кластерного индекса
  - указатель на значение записи из кластерного индекса
- В новый индексный уровень помещаем минимальное значение ключа и адрес индексного блока.
- Наращивая уровни, пока не появится уровень из одного блока.

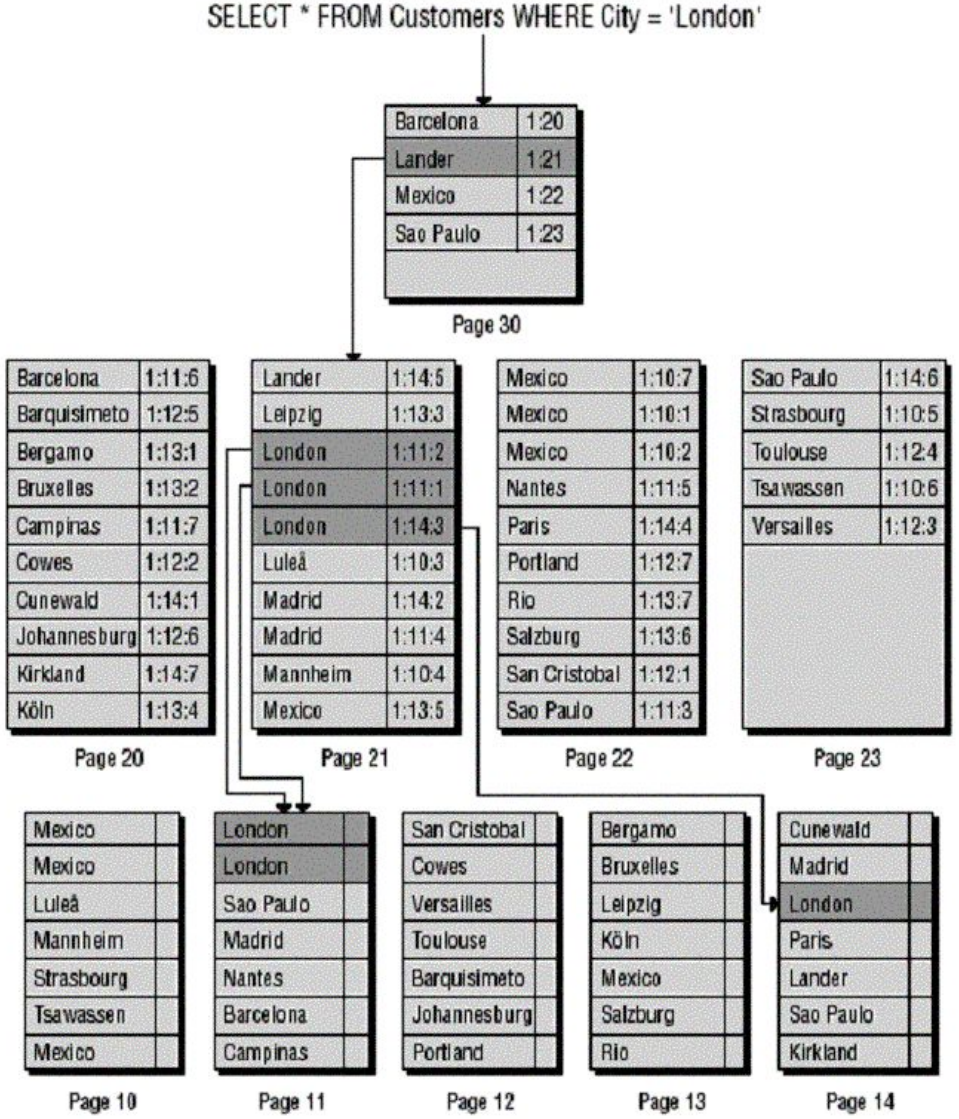
Некластерный индекс ссылается на значения кластерного ключа

- Уникальный КК- если кластерный ключ не уникален, sql server его «уникализирует» добавлением к информации 4 байтового целого => возникают дополнительные накладные расходы на создание индекса, расходуется место на диске, дополнительно возрастает стоимость операций вставки и обновления.
- Узкий КК - его значения дублируются во всех некластерных индексах. «Узкий» — это значит нужно постараться использовать как можно меньше байт, чтобы уникально определить ваши строки. «Узкое» число, если возможно.
- Статичный КК - используется для поиска из всех некластерных индексов, тогда он дублируется во всех некластерных индексах. Изменяется => требуется обновить как значения в базовой таблице, так и значения в каждом некластерном индексе. И если ключ изменяется, это заставляет запись перемещаться. Когда запись перемещается — это создает фрагментацию.

Некластерный индекс



Поиск с помощью B-дерева



Составной ключ

- Индекс может быть создан на основании нескольких полей.
- В один ключ составного индекса могут входить до 16 столбцов. Все столбцы ключа составного индекса должны находиться в одной таблице или одном и том же представлении.
- Составной индекс для (Column1, Column2) является совершенно отличным от (Column2, Column1), а так же от индексов созданных по двум этим полям в отдельности.

Составной ключ

- Располагайте в начале ключи индекса, которые часто используются в WHERE выражениях.
- Старайтесь располагать ключи индекса в порядке уменьшения избирательности (selectivity), т.е. ключ с наибольшей избирательностью должен быть самым левым.

Ограничения по длине

- Суммарная длина ключа индекса не должна превышать 900 байтов.
- Если индекс построен по полям с фиксированным размером, сумма длин этих полей должна не превышать эти 900 байт.
- Если индекс построен по полям с переменной длинной, сумма максимальных размеров полей может превышать 900 байт, но само значение сумм по каждой записи не может быть больше 900 байт.

Выбор столбцов

- Следует создавать некластеризованные индексы для всех столбцов, которые часто используются в предикатах и условиях соединения в запросах.
- Нужно избегать добавления столбцов без необходимости – снижается производительность поддержания индекса.
- Определите тип запроса и то, как в нем используются столбцы - столбец, который используется в запросе с точным соответствием, может оказаться подходящим кандидатом для создания индекса.

Примеры предикатов с использованием составного индекса

- CREATE CLUSTERED INDEX Ind1 ON Table1 (Name, IdNum)
- Where Name like ‘Ива%’ and IdNum=200
- Where Name like ‘Ива%’

Примеры предикатов без использования индекса

- CREATE CLUSTERED INDEX Ind1 ON Table1 (Name, IdNum)
- Where IdNum > 100
- Where Name like ‘%ва%’ and IdNum=200

Покрывающий индекс

- Если все столбцы запросы входят в состав ключа в индексе, то такой индекс называется покрывающим.
- Покрывающие индексы могут повысить производительность запросов, так как данные, необходимые для удовлетворения требований запроса, присутствуют в самом индексе – не требует считывания страниц данных.

Включенные столбцы для некластеризованных индексов

- Можно добавлять неключевые столбцы к конечному уровню некластеризованного индекса. Это позволяет покрывать больше запросов.
- Они могут содержать типы данных, не разрешенные для ключевых столбцов индекса.
- Они не учитываются при расчете числа ключевых столбцов индекса и размера ключа индекса.
- Индекс с включенными неключевыми столбцами может значительно повысить производительность запроса, когда все столбцы запроса включены в индекс как ключевые или неключевые.

Отфильтрованные индексы

- Отфильтрованный индекс - некластеризованный индекс, построенный по некоторому подмножеству значений ключа.
- Может повысить производительность запросов, снизить затраты на обслуживание и хранение индексов по сравнению с полнотабличными индексами.
- Фильтрованные индексы полезны на больших таблицах.

Отфильтрованные индексы

- Индекс обслуживается только в случае, если инструкции языка обработки данных (DML) затрагивают данные в индексе. Возможно наличие большого числа отфильтрованных индексов, особенно если они содержат редко изменяющиеся данные. Аналогично, если отфильтрованный индекс содержит только часто изменяемые данные, меньший размер индекса уменьшает затраты на обновление статистики.
- Снижение затрат на хранение индекса
- Создание отфильтрованного индекса может уменьшить место на диске для некластеризованных индексов, если нет необходимости в полнотабличном индексе. Полнотабличный некластеризованный индекс можно заменить несколькими отфильтрованными индексами без значительного увеличения требований к хранилищу.

Отфильтрованные индексы

- Разреженные столбцы, содержащие небольшое количество не NULL значений.
- Разнородные столбцы, содержащие категории данных.
- Столбцы, содержащие диапазоны значений, таких как количество долларов, время и даты.
- Секции таблицы, определенные логикой простого сравнения для значений столбцов.

Отфильтрованные индексы

```
CREATE NONCLUSTERED INDEX Ind2
   ON Production.Materials (CmpD, StartDate)
   WHERE EndDate IS NOT NULL ;
```

Индексы - недостатки

- Индексы занимают дополнительное место на диске и в оперативной памяти. Чем больше/длиннее ключ, тем больше размер индекса.
- Замедляются операции вставки, обновления и удаления записей.
- Однако алгоритмы построения индексов разработаны так, чтобы иметь как можно меньший негативный эффект для указанных операций и даже позволяет выполнять их быстрее.

Не строить лишние индексы:

- Большое количество индексов в таблице снижает производительность инструкций INSERT, UPDATE, DELETE и MERGE, потому что при изменении данных в таблице все индексы должны быть изменены соответствующим образом.
- Для интенсивно обновляемых таблиц – меньше индексов.
- Для таблиц с редкими обновлениями, но большими объемами данных - больше индексов.

FILLFACTOR

- Коэффициент заполнения - при создании или перестроении индекса позволяет зарезервировать место на каждой странице конечного уровня для будущего расширения.
- Коэффициент заполнения — это значение в процентах от 1 до 100; значение по умолчанию на сервере — 0 (полное заполнение страниц конечного уровня).
- Например, FillFactor = 80 => 20 % free. Пустое место резервируется между строками индекса.

Индексы для маленьких таблиц

- Индексирование маленьких таблиц может оказаться не лучшим выбором, так как поиск данных в индексе может потребовать у оптимизатора запросов больше времени, чем простой просмотр таблицы.
- Для маленьких таблиц индексы могут вообще не использоваться, но тем не менее их необходимо поддерживать при изменении данных в таблице.

Представления WITH SCHEMABINDING

- **SCHEMABINDING**
  Привязывает представление к схеме базовой таблицы или таблиц
- => базовая таблица или таблицы не могут быть изменен таким образом, чтобы повлиять на определение представления.

Индексированное представление

- WITH SCHEMABINDING – можно построить индексы, первым – кластерный.
- Тогда представление будет храниться в базе данных подобно таблице с кластеризованным индексом.
- Поддерживать индексированное представление труднее, чем индекс таблицы. Если базовые данные обновляются часто, расходы на поддержание данных индексированного представления могут перевесить преимущества от его использования.

Индексное представление

- Может дать значительное улучшение производительности, если представление содержит агрегаты, объединения таблиц или сочетание того и другого.
- Повышают эффективность запросов:
  - Соединения и статистические вычисления, в ходе которых обрабатывается большое число строк.
  - Соединения и статистические вычисления, которые часто выполняются несколькими запросами.

Советы по использованию индексов

- Перед построением нового индекса убедитесь, что такого еще нет.
- Уникальность столбцов лучше указывать.
- Мало уникальных значений - плохо.
- Отфильтрованные индексы для столбцов, имеющих точно определенные подмножества
- Индексировать вычисляемые столбцы.

Статистика

Дата и время последнего обновления статистики.

Общее число строк в таблице или индексированном представлении

Распределение значений ключа.

Плотность ключа - 1/distinct values для всех значений в первом ключевом столбце объекта статистики.

Среднее число байтов на значение для всех ключевых столбцов в объекте статистики.

Является ли индекс строковым

И прочая полезная информация

Статистика

Посмотреть статистику

DBCC SHOW_STATISTICS
(table_or_indexed_view_name , index_name)

Обновить статистику

UPDATE STATISTICS
table_or_indexed_view_name index_name

Фрагментация

- При INSERT блок делится на два
- При UPDATE может увеличиться длина записи
- При DELETE остаются пустые места
- Фрагментация уменьшает производительность групповых операций модификации данных
- Низкий % заполнения страниц увеличивает количество страниц в индексе и использует лишнюю память

Выявление фрагментации

- select *
  from sys.dm_db_index_physical_stats
  (DB_ID(‘DB_name')
  ,OBJect_ID('table')
  , NULL
  , NULL
  , 'DETAILED')

Параметры

- avg_fragmentation_in_percent
  Процентная доля логической фрагментации (неупорядоченные страницы в индексе).
- fragment_count
  Число фрагментов (физически последовательные конечные страницы) в индексе.
- avg_fragment_size_in_pages
  Среднее число страниц в одном фрагменте индекса.

Перестройка индекса

Значение **avg_fragmentation_in_percent**	Корректирующая инструкция
> 5 % и <= 30 %	ALTER INDEX REORGANIZE
> 30%	ALTER INDEX REBUILD [WITH (ONLINE = ON|OFF)]

Рекомендации – 1

- Создавайте кластерный индекс для каждой таблицы.
- Удаляйте лишние индексы.
- Постарайтесь создать индексы по столбцам, которые имеют целые, а не символьные значения.
- Ограничьте количество индексов, если ваше приложение часто обновляет данные.

Рекомендации – 2

- Создавайте индекс для столбцов, которые часто используются в JOIN’ах.
- Создавайте кластерный индекс для увеличения производительности запросов, которые возвращают диапазон значений, и для запросов, содержащих GROUP BY или ORDER BY выражения и возвращающих отсортированные результаты.

Рекомендации – 3

- Не используйте Identity в качестве первичного ключа (запись всегда в конец – много запросов пытаются прочитать и записать данные в одну область в одно время).
- Желателен покрывающий индекс, (включающий все столбцы) для частых запросов.
- Периодически перестраивайте все индексы во всех таблицах для уменьшения фрагментации.

Виды индексов:

- B-деревья
- Hash-индексы
- Индексы на основе битовых карт
- R-деревья
- Многомерные индексы