Professional Association for SQL Server

Все, что Вы всегда хотели знать об индексах, но боялись спросить

Дмитрий Короткевич
Tampa, FL
Actsoft, Inc
MVP, MCITP, MCPD

О Докладчике

- Начальник отдела разработки в Actsoft, Inc
  - Архитектура, дизайн и разработка OLTP системы обрабатывающей ~3,000 транзакций в секунду
- 15 лет в индустрии
- Более 10 лет работы с Microsoft SQL Server
- SQL Server MVP
- MCITP - DB Developer, DBA
- MCPD – Enterprise Application Developer
- Blog: http://aboutsqlserver.com
- Email: dmitri@aboutsqlserver.com
  - Примеры доступны для скачивания
    - http://aboutsqlserver.com/presentations

Microsoft® CERTIFIED IT Professional

Microsoft® CERTIFIED Professional Developer

Содержание

- Первая часть
  - Физическая структура индексов
  - Как и когда SQL Server использует индексы
- Вторая часть
  - Дополнительные возможности
    - Фильтруемые (Filtered) индексы
    - Индексы с дополнительными (INCLUDE) полями
    - Секционированные индексы
  - Фрагментация
- Третья часть
  - Стратегии индексации
  - Стратегии оптимизации

Перед тем как мы начнем...

- На практике все всегда работает не так, как было запланировано
  - Тестируем, тестируем, тестируем, тестируем, ...
- Каждая система – уникальна. В некоторых случаях необходимо отступать от общепринятых правил
  - Тестируем, тестируем, тестируем, тестируем, ...
- В каждом правиле есть исключения
  - Тестируем, тестируем, тестируем, тестируем, ...
- Ответом на любой вопрос является фраза:
  - «Это зависит от других факторов»
    - Ну и, конечно, тестируем, тестируем, тестируем, ...

Физическая Структура Индексов

• • •

Как SQL Server хранит данные

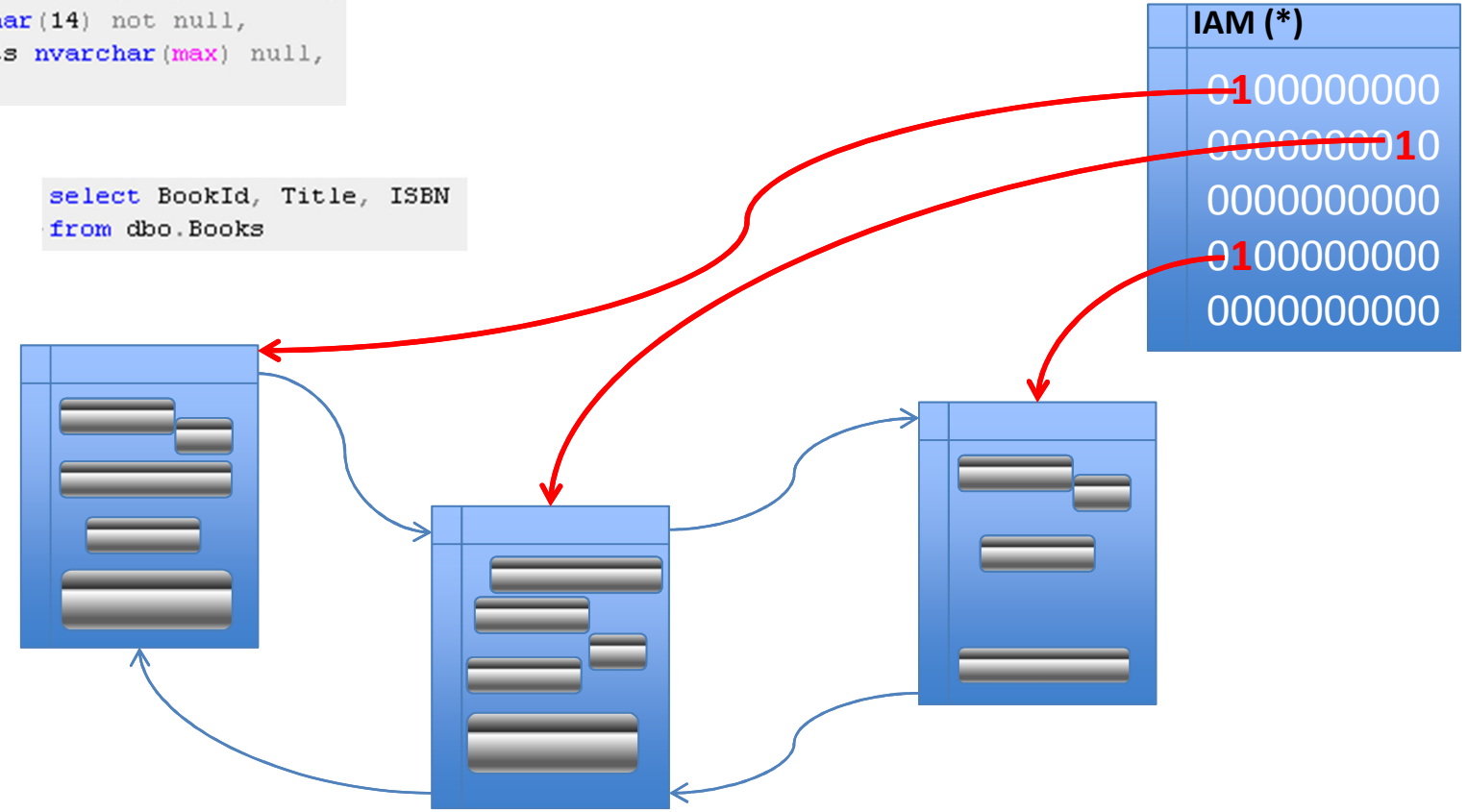
В начале была.. Страница

- SQL Server хранит данные на страницах (pages) размером 8К
  - 8,060 байт доступны пользователю
- Страницы принадлежащие объекту связаны в двунаправленный список
- 8 страниц сгруппированы в «участки» (extents)
  - Смешаные участки (mixed extents) – хранят данные из разных «объектов»
  - Унифицированные участки (unified extents) – хранят данные одного «объекта»
- Первые 8 страниц «объекта» хранятся в смешанных участках. После этого данные хранятся только в унифицированных участках.
- SQL Server использует специальные страницы - «карты размещения индексов» (Index Allocation Map) – IAM - для определения страниц принадлежащих «объекту»
  - На следующих слайдах структура IAM упрощена

Heap таблицы

```
create table dbo.Books
(
    BookId int not null,
    Title nvarchar(256) not null,
    ISBN char(14) not null,
    Comments nvarchar(max) null,
)
```

```
select BookId, Title, ISBN
from dbo.Books
```

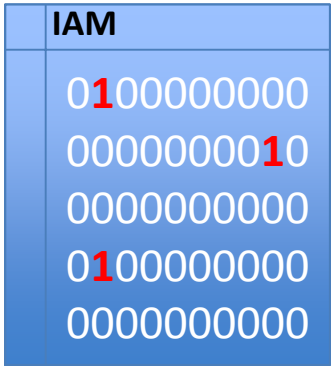


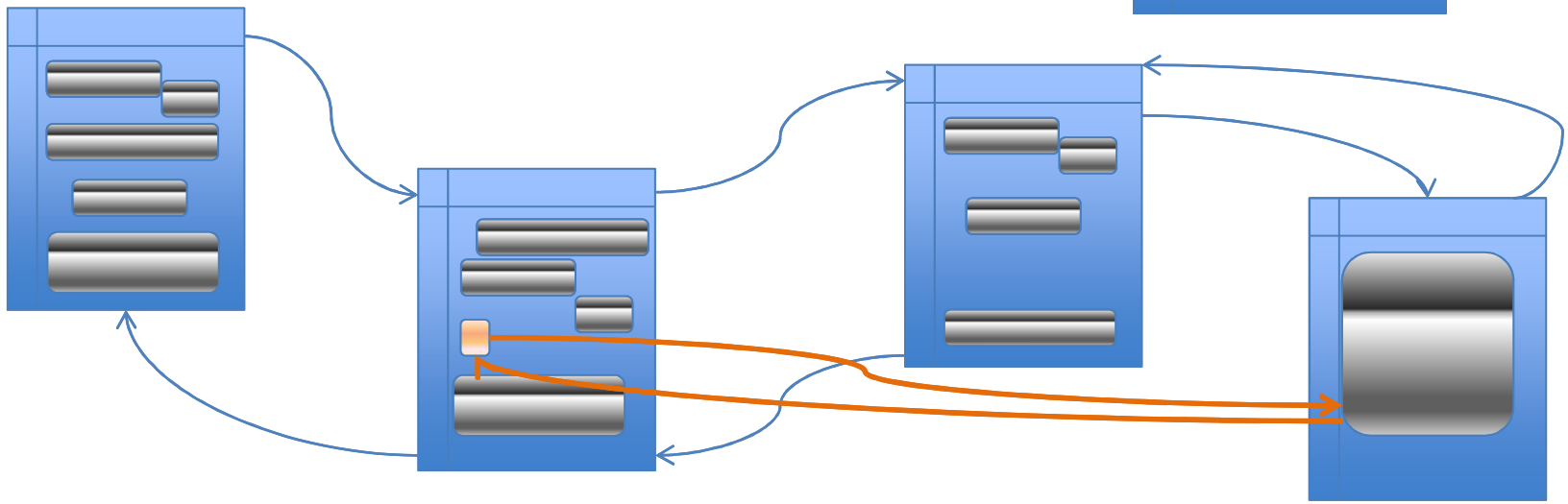
(*) Упрощенно

Heap таблицы

```
create table dbo.Books
(
    BookId int not null,
    Title nvarchar(256) not null,
    ISBN char(14) not null,
    Comments nvarchar(max) null,
)
```

```
update dbo.Books
set
    Comments = N'Very Long Text'
where
    BookId = 123
```

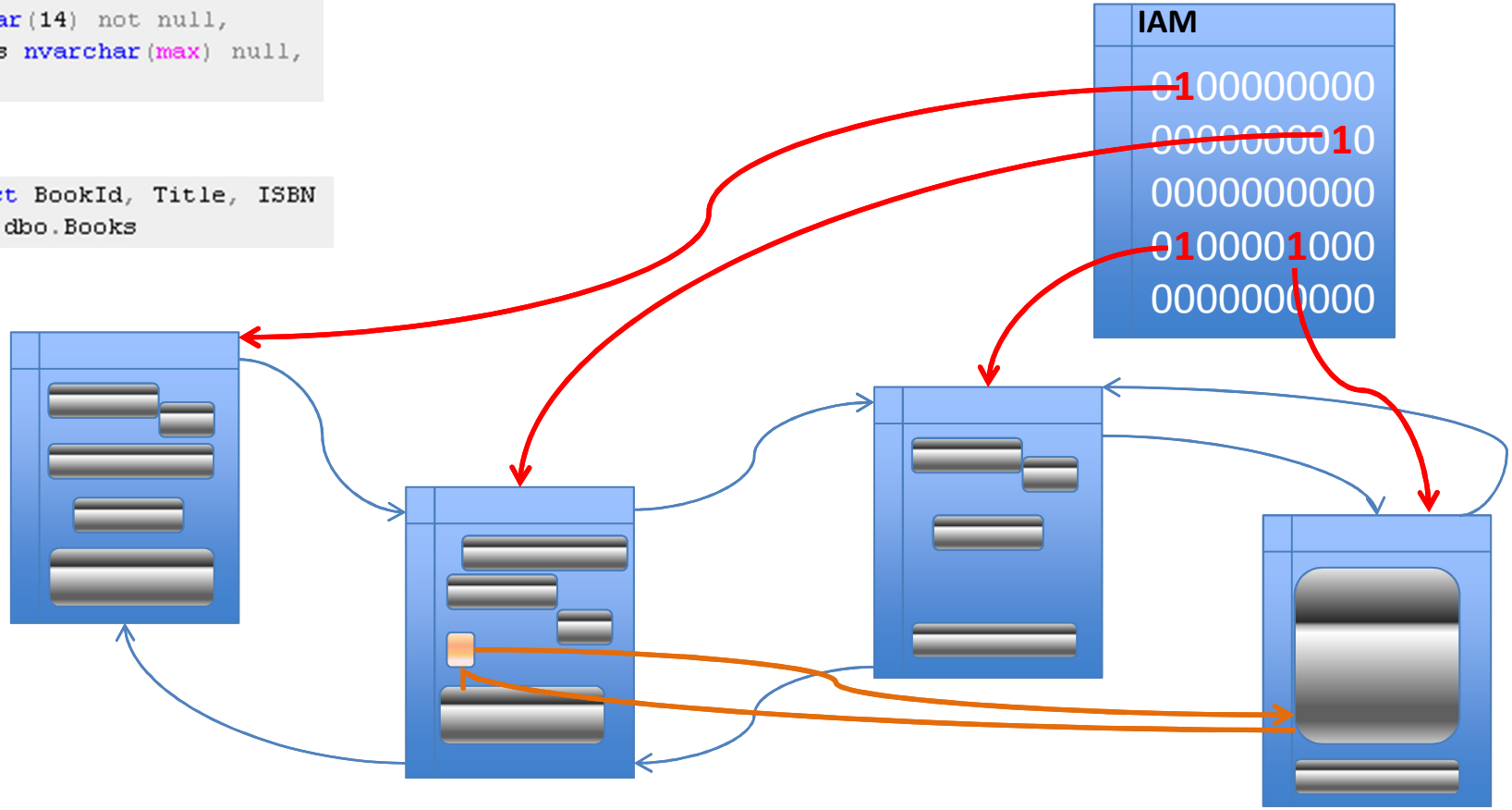




Heap таблицы

```
create table dbo.Books
(
    BookId int not null,
    Title nvarchar(256) not null,
    ISBN char(14) not null,
    Comments nvarchar(max) null,
)
```

```
select BookId, Title, ISBN
from dbo.Books
```



Professional Association for SQL Server

24 HOURS OF PASS

Демонстрация

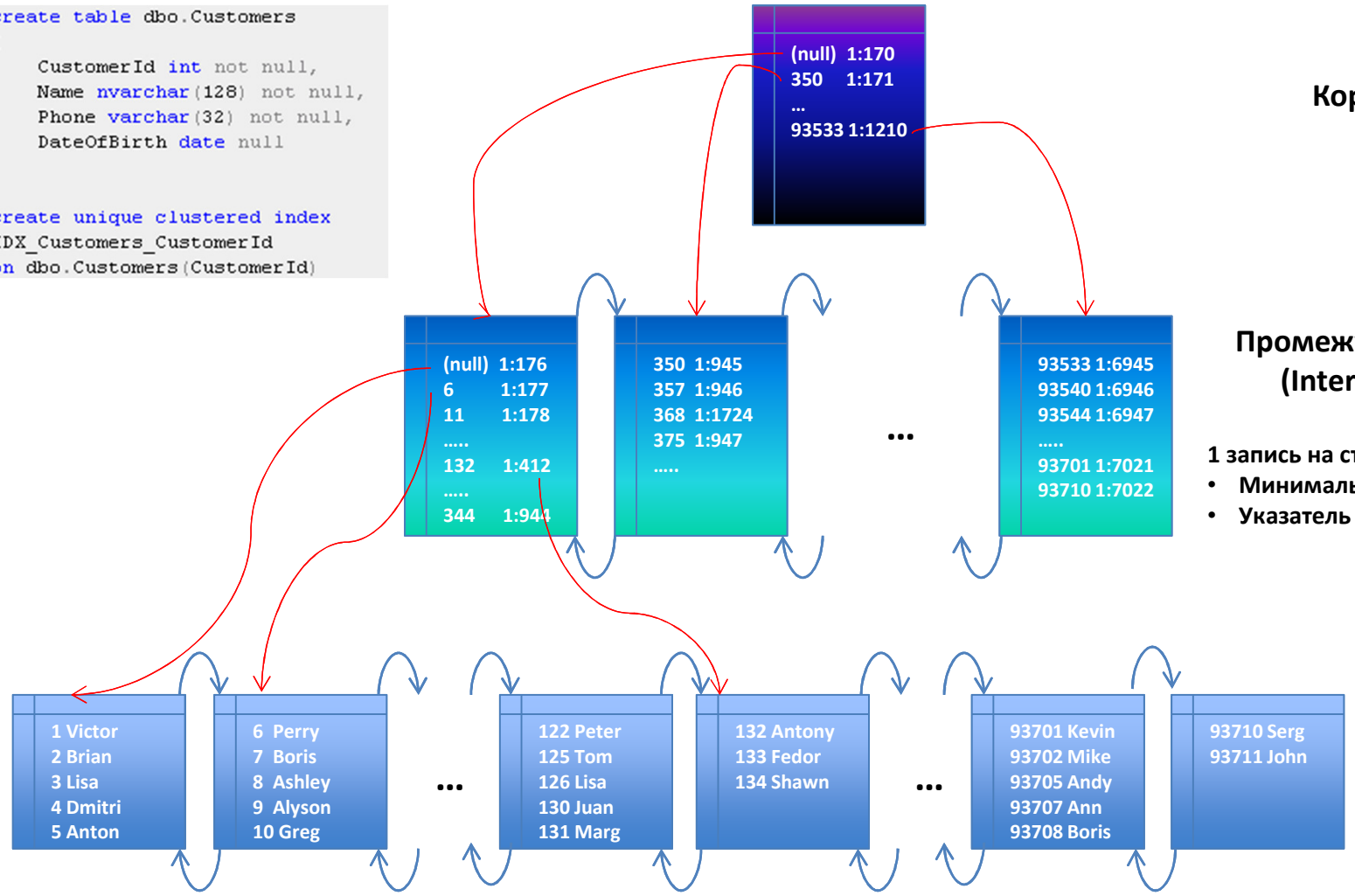
HEAP Таблицы

Heap таблицы

- Потенциальные проблемы:
  - Лишние операции чтения/записи из-за forwarding pointers
  - Сабоптимальный контроль свободного пространства
- Потенциальное использование
  - Staging Environment для быстрой загрузки

Кластерный Индекс

```
create table dbo.Customers
(
    CustomerId int not null,
    Name nvarchar(128) not null,
    Phone varchar(32) not null,
    DateOfBirth date null
)

create unique clustered index
IDX_Customers_CustomerId
on dbo.Customers(CustomerId)
```



Корневой уровень (Root Level)

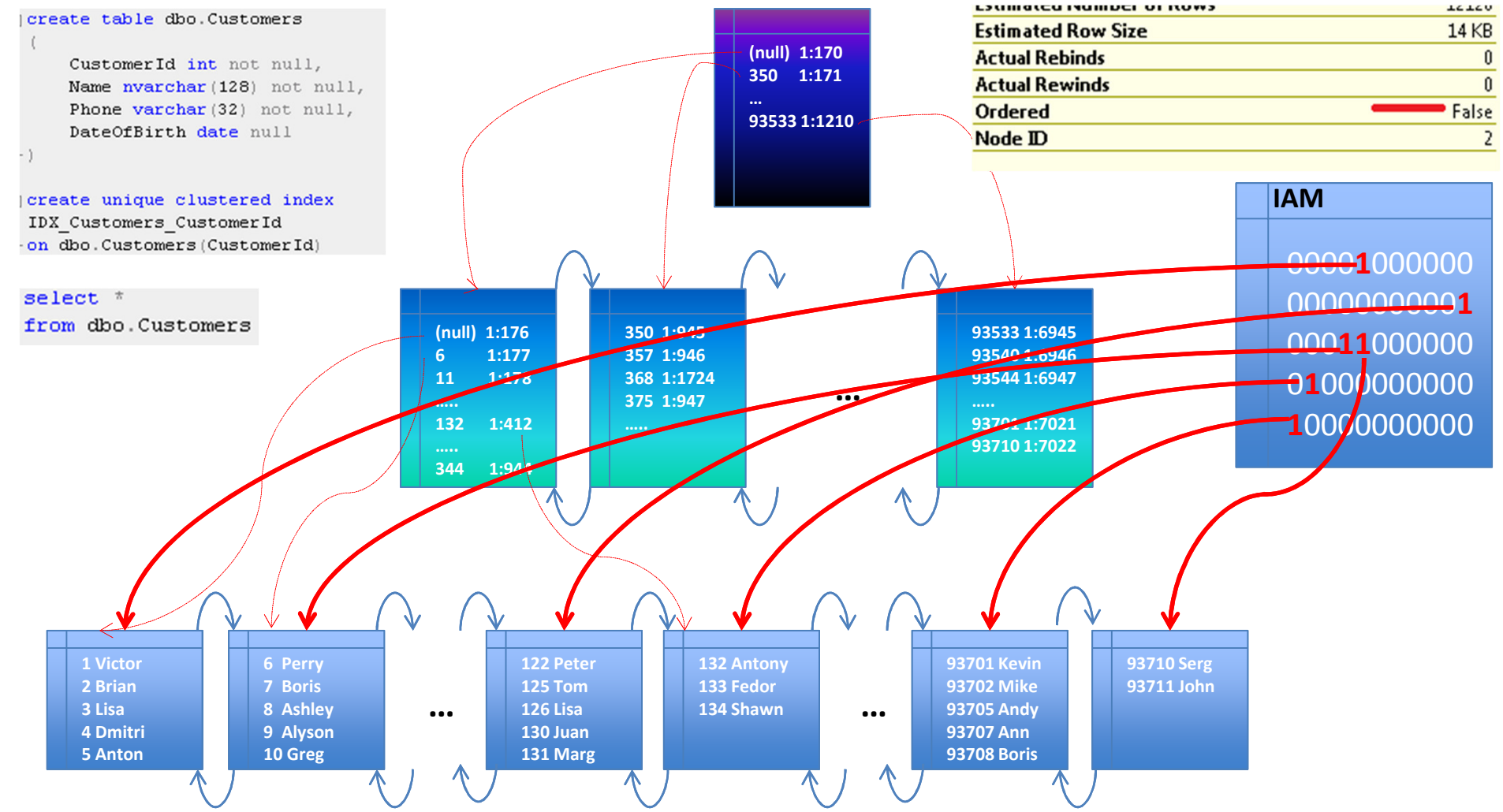
Промежуточный уровень (Intermediate Level)

1 запись на страницу на след. уровне

- **Минимальное значение ключа**
- **Указатель на страницу**

Листья (Leaf Level) Данные

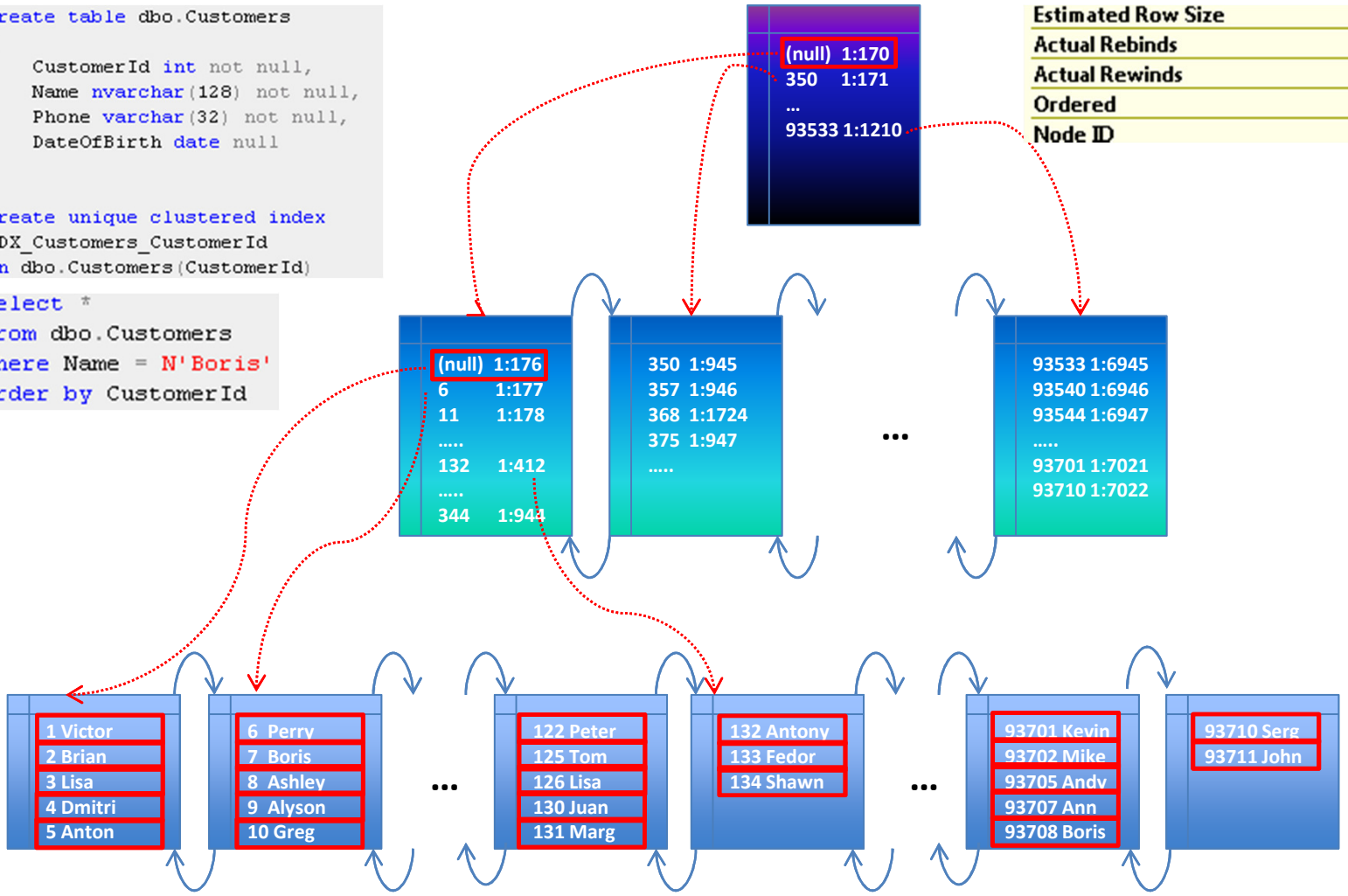
Использование Индексов
IAM сканирование



Использование Индексов
Отсортированное сканирование (ordered scan)

```
create table dbo.Customers
(
    CustomerId int not null,
    Name nvarchar(128) not null,
    Phone varchar(32) not null,
    DateOfBirth date null
)

create unique clustered index
IDX_Customers_CustomerId
on dbo.Customers(CustomerId)
```

```
select *
from dbo.Customers
where Name = N'Boris'
order by CustomerId
```

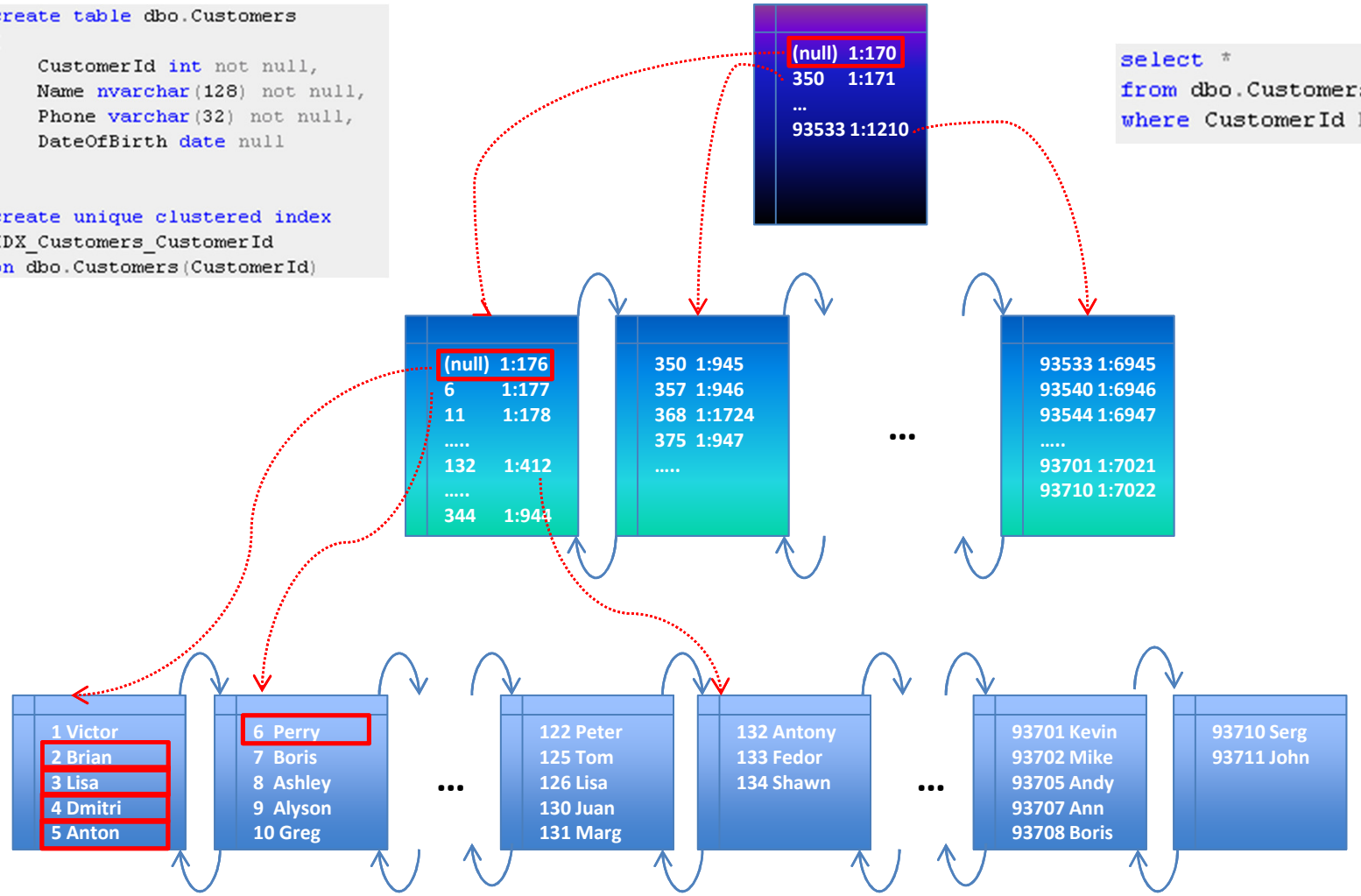


Estimated Row Size	14 KB
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	2

Использование Индексов
Поиск (Index Seek)

```
create table dbo.Customers
(
    CustomerId int not null,
    Name nvarchar(128) not null,
    Phone varchar(32) not null,
    DateOfBirth date null
)

create unique clustered index
IDX_Customers_CustomerId
on dbo.Customers(CustomerId)
```

```
select *
from dbo.Customers
where CustomerId between 2 and 6
```



Предикаты приводящие к INDEX SEEK (SARG – Seekable/Searchable Argument)

- Предикаты приводящие к INDEX SEEK:

```
where CustomerId = 1

where CustomerId > 1000

where CustomerId in (1, 2)

where VarCharCol like 'A%'

where DateCol between '2012-01-01' and '2012-02-01'
```

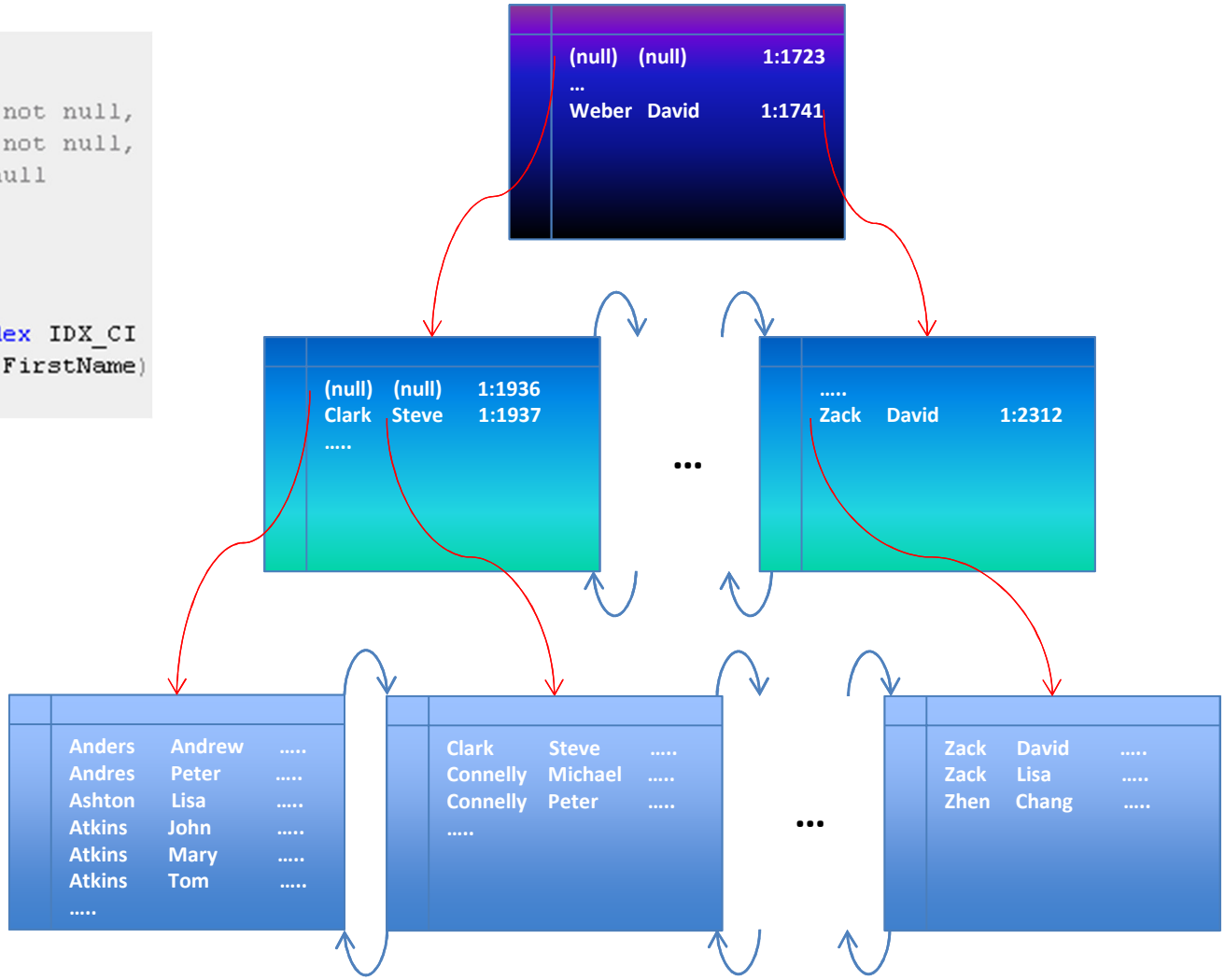
- Предикаты не использующие INDEX SEEK:

```
where IntCol + 1 = 10          → where IntCol = 9

where ABS(IntCol) = 1          → where IntCol in (-1, 1)

where DATEPART(YEAR,DateTimeCol) = 2012   → where DateTimeCol >= '2012-01-01' and DateTimeCol < '2013-01-01'

where DATEADD(DAY,7,DateTimeCol) > GETDATE()   → where DateTimeCol > DATEADD(DAY,-7,GETDATE())

where VarCharCol like '%A%'
```

```
where VarCharCol = N'nvarchar parameter'
```

Структура Композитного Индекса

```
create table dbo.Customers
(
    FirstName nvarchar(64) not null,
    LastName nvarchar(128) not null,
    Phone varchar(32) not null
    -- ...
)
go

create unique clustered index IDX_CI
on dbo.Customers(LastName, FirstName)
go
```



Предикаты приводящие к INDEX SEEK (SARG) для композитного индекса

```
create table dbo.Customers
(
    FirstName nvarchar(64) not null,
    LastName nvarchar(128) not null,
    Phone varchar(32) not null
    -- ...
)
go

create unique clustered index IDX_CI
on dbo.Customers(LastName, FirstName)
go
```

- Предикаты приводящие к INDEX SEEK:

```
where LastName = N'Sanders' and FirstName = N'Tom'

where LastName = N'Sanders'
```

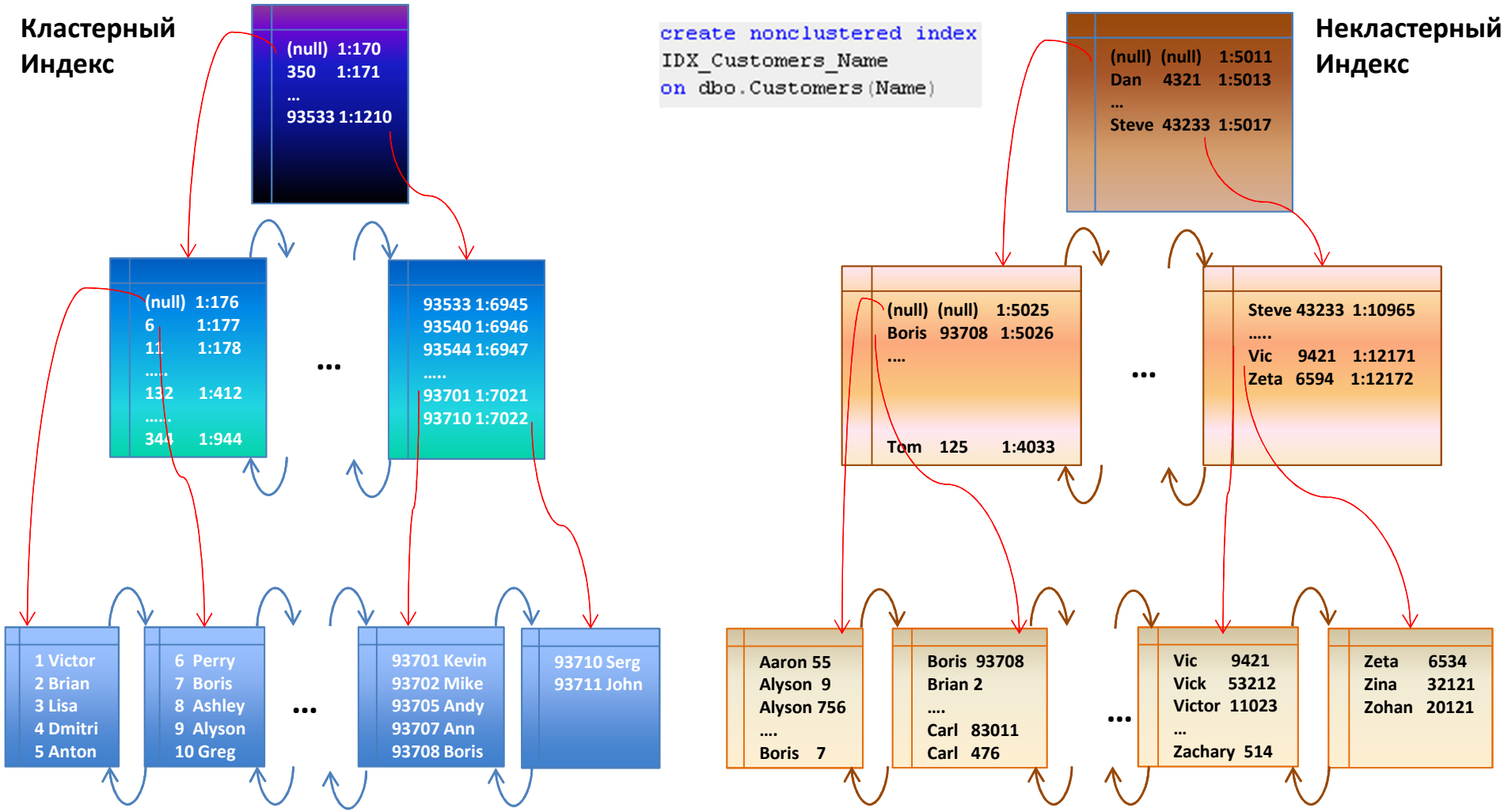
“Левые” поля должны быть SARG

- Предикаты не использующие INDEX SEEK:

```
where LastName like N'%A%' and FirstName = N'Tom'

where FirstName = N'Tom'
```

“Левые” поля не являются SARG

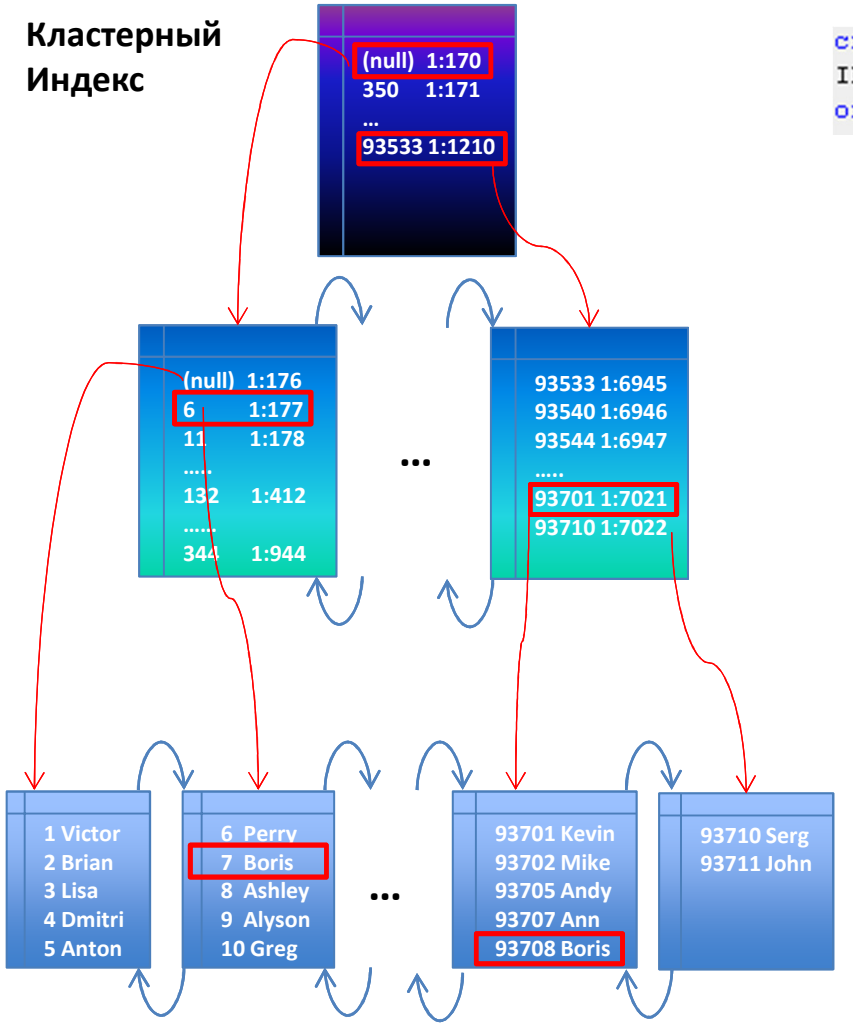
Некластерный индекс



Некластерный индекс

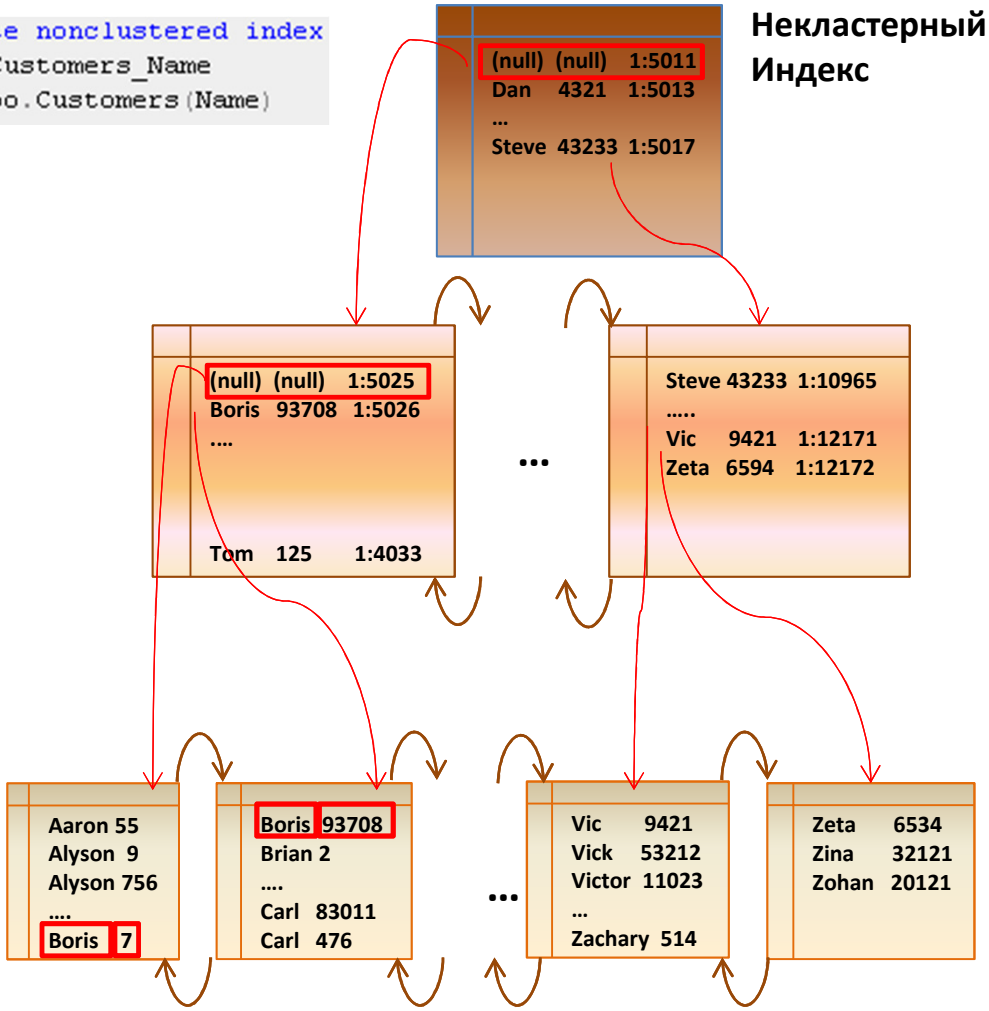
```
select *
from dbo.Customers
where Name = N'Boris'
```

Кластерный Индекс




```
create nonclustered index
IDX_Customers_Name
on dbo.Customers(Name)
```

Некластерный Индекс



Структура некластерного индекса

- Листья (Leaf level): Ключ + идентификатор записи
  - HEAP таблицы: файл:страница:слот
  - Таблицы с кластерным индексом: значение кластерного ключа
    - Приблизительная стоимость использования (чтение):
      (Количество уровней в индексе) +
      (Количество страниц прочитанных на leaf уровне) +
      *(Количество найденых записей) * (Количество уровней кластерного индекса)*
    - Упреждающее чтение (read-ahead) для кластерного индекса работает неэффективно
- Промежуточные и корневой уровни
  - Уникальные индексы - Значения ключа
  - Неуникальные индексы – Значения ключа + идентификатор записи

Professional Association for SQL Server

24 HOURS OF PASS

Демонстрация

Использование Некластерного Индекса

Оптимизатор

- Оптимизатор не пытается найти *наилучший* план
- Оптимизатор ищет *приемлимый* план
- В общем случае:
  - Опция 1 – сканирование таблицы / кластерного индекса
  - Для остальных опций оптимизатору необходимо оценить *стоимость* использования некластерного индекса
    - Одним из критериев является количество записей для обработки на каждом шагу и как результат количество key lookup операций
    - SQL Server использует статистику для оценки этого количества
    - При создании индекса статистика на индекс создается автоматически

Professional Association for SQL Server

Демонстрация

Статистика

Статистика

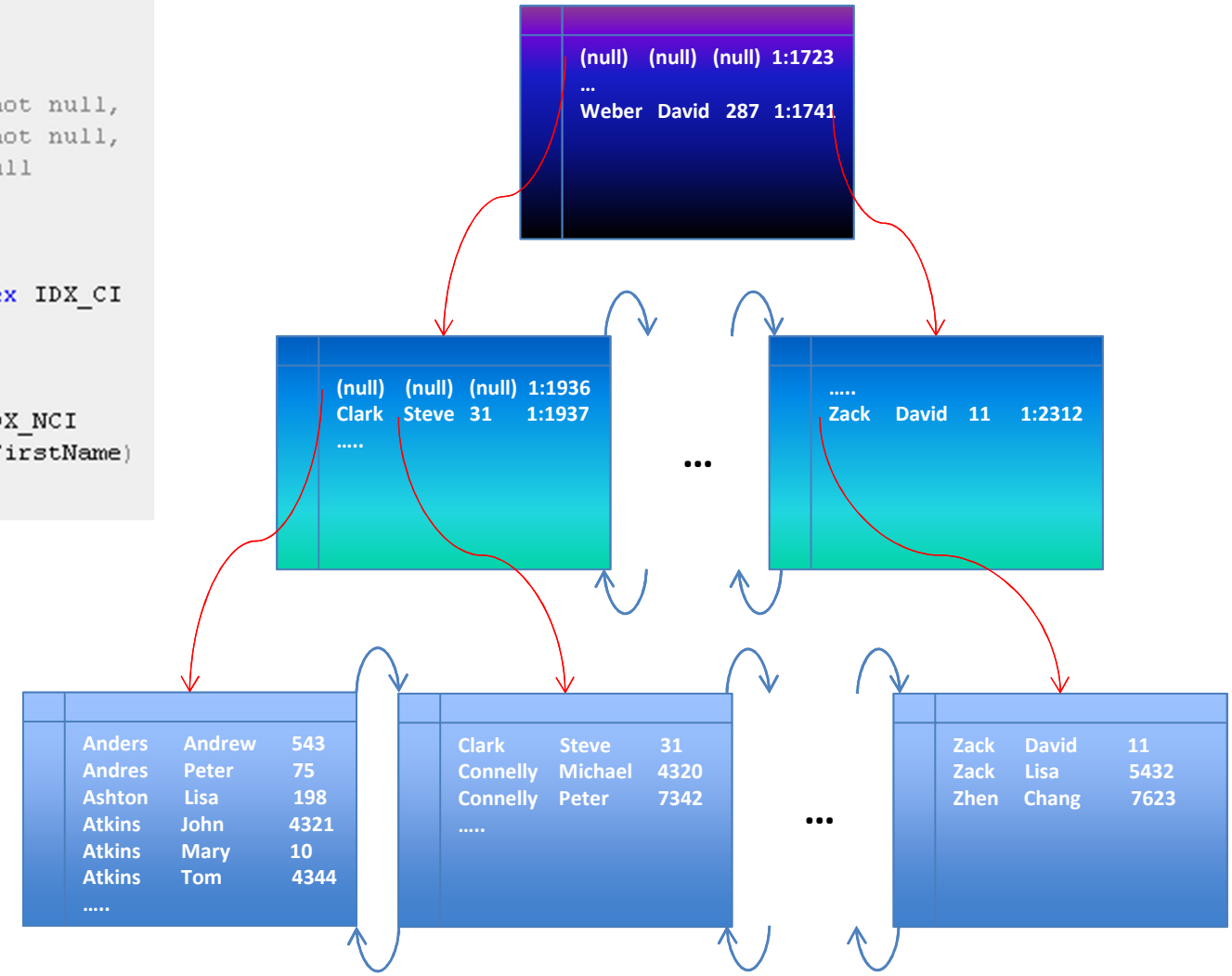
- В гистограмме хранится не более 200 значений
  - Чем больше записей в таблице, тем менее аккуратна статистика
- Ошибки в оценке числа записей на начальном этапе выполнения (правая часть плана) прогрессируют в левую часть плана в геометрической прогрессии
  - Как результат, генерируется сабоптимальный план
    - Типичный пример – Nested Loop Join вместо Hash Join
- Гистограмма хранится только по первому (левому) полю индекса

Статистика

- Обновление статистики
  - Автоматически / Вручную
    - Автоматическое обновление базируется на количестве изменений первого (левого) поля индекса
      - Вставка новых записей в пустую таблицу
      - <= 500 записей – 500 изменений
      - > 500 записей – 500 + 20% от числа записей
  - Синхронно / Асинхронно
- Чем больше записей в таблице, тем менее аккуратна статистика

Структура Композитного Индекса

```
create table dbo.Customers
(
    CustomerId int not null,
    FirstName nvarchar(64) not null,
    LastName nvarchar(128) not null,
    Phone varchar(32) not null
)
go

create unique clustered index IDX_CI
on dbo.Customers(CustomerId)
go

create nonclustered index IDX_NCI
on dbo.Customers(LastName, FirstName)
go
```



Статистика и Композитные Индексы

- Проблема:

```
create table dbo.Customers
(
    CustomerId int not null,
    FirstName nvarchar(64) not null,
    LastName nvarchar(128) not null,
    Phone varchar(32) not null
)
go

create unique clustered index IDX_CI
on dbo.Customers(CustomerId)
go

create nonclustered index IDX_NCI
on dbo.Customers(LastName, FirstName)
go
```

```
select *
from dbo.Customers
where FirstName = @FirstName
```

- Альтернативы:
  - Сканирование кластерного индекса
  - Сканирование некластерного индекса + key lookup
- Стоимость плана зависит от числа записей

- Решение: Статистика на уровне поля

```
create statistics CLS_FirstName
on dbo.Customers(FirstName)
go
```

Professional Association for SQL Server

24 HOURS OF PASS

Демонстрация

Статистика на уровне поля

Статистика и Композитные Индексы

- Гистограмма хранится только по первому (левому) полю
- Проблема:

```
create table dbo.Positions
(
    CompanyId int not null,
    UTCTimeTag datetime2(0) not null,
    RecId bigint not null,
    DeviceId int not null,
    Latitude decimal(9,6) not null,
    Longitude decimal(9,6) not null
)
go

create unique clustered index IDX_CI
on dbo.Positions
(CompanyId, UTCTimeTag, RecId)
go

create nonclustered index IDX_NCI
on dbo.Positions
(CompanyId, DeviceId, UTCTimeTag)
```

- DeviceId уникален в рамках компании
- Какой индекс выбрать?

```
select *
from dbo.Positions
where
    CompanyId = @CompanyId and
    UTCTimeTag between @StartTime and @StopTime and
    DeviceId in
    (
        select DeviceId
        from #Devices
    )
```

Предикаты приводящие к INDEX SEEK (SARG) для композитного индекса

```
create table dbo.Customers
(
    CustomerId int not null,
    FirstName nvarchar(64) not null,
    LastName nvarchar(128) not null,
    Phone varchar(32) not null
)
go

create unique clustered index IDX_CI
on dbo.Customers(CustomerId)
go

create nonclustered index IDX_NCI
on dbo.Customers(LastName, FirstName)
go
```

- Предикаты приводящие к INDEX SEEK:

```
where LastName = N'Sanders' and FirstName = N'Tom'

where LastName = N'Sanders'
```

“Левые” поля должны быть SARG

~~create nonclustered index IDX_NCI2 on dbo.Customers(LastName)~~

- Предикаты не использующие INDEX SEEK:

```
where LastName like N'%A%' and FirstName = N'Tom'

where FirstName = N'Tom'
```

“Левые” поля не являются SARG

Суммируя

- Key Lookup очень дорогая операция
  - SQL Server не использует некластерный индекс в случае если предполагаемое количество key lookup операций > *нескольких* процентов
- SQL Server использует статистику для оценки числа записей
  - Гистограмма хранится только по первому (левому) полю индекса
  - Гистограмма содержит не более 200 значений
- В случае больших таблиц
  - Статистика / Гистограмма не аккуратны
  - Обновление статистики происходит при обновлении примерно 20% записей

Дополнительные Возможности

• • •

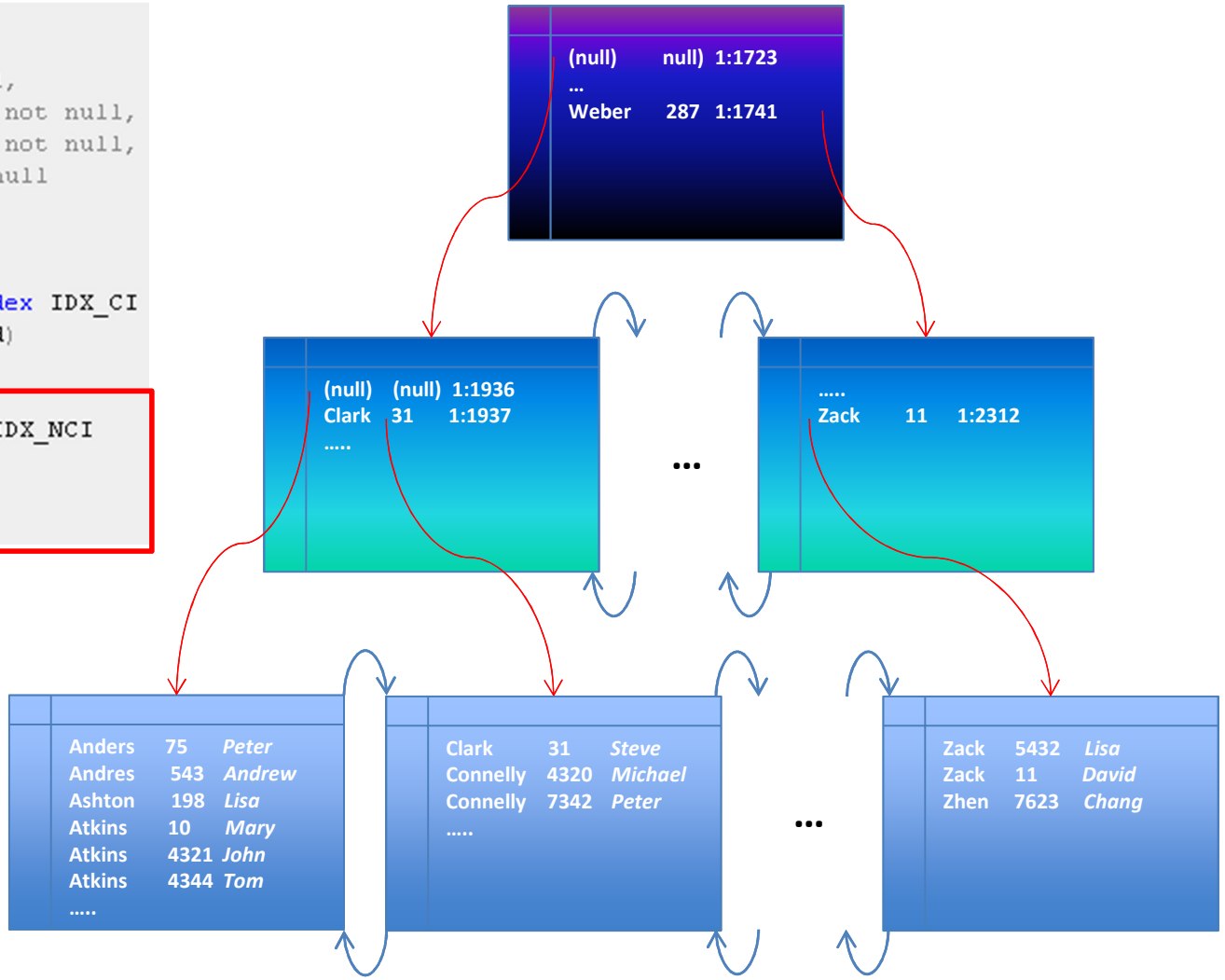
Индексы с дополнительными полями

Фильтрованные индексы

Секционированные индексы

Индекс с дополнительными полями (INCLUDE)

```
create table dbo.Customers
(
    CustomerId int not null,
    FirstName nvarchar(64) not null,
    LastName nvarchar(128) not null,
    Phone varchar(32) not null
)
go

create unique clustered index IDX_CI
on dbo.Customers(CustomerId)
go

create nonclustered index IDX_NCI
on dbo.Customers(LastName)
include(FirstName)
go
```



Индекс с дополнительными полями (INCLUDE)

- Данные по дополнительным полям хранятся только на уровне листьев (leaf level)
- Дополнительные поля не сортируются
- Позволяет обойти ограничение максимального размера ключа в 900 байт
- Помогает при консолидации индексов
- Позволяет избавиться от key lookup в запросах (covering indexes)
  - **Один из основных способов оптимизации**

Professional Association for SQL Server

24 HOURS OF PASS

Демонстрация

Индекс с дополнительными полями

Индекс с дополнительными полями (INCLUDE)

- Недостатки:
  - Больший размер записи на уровне листьев
  - Затраты на обновление дополнительных полей

Фильтрованные Индексы (SQL Server 2008+)

```
create table dbo.Positions
(
    CompanyId int not null,
    UTCTimeTag datetime2(0) not null,
    RecId bigint not null,
    DeviceId int not null,
    Latitude decimal(9,6) not null,
    Longitude decimal(9,6) not null,
    Processed bit not null default 0
)
go

create unique clustered index IDX_CI
on dbo.Positions
(CompanyId, UTCTimeTag, RecId)
go

create unique nonclustered index
IDX_Positions_Processed_Filtered
on dbo.Positions(RecId)
where Processed = 0
go

select top 5000 *
from dbo.Positions
where Processed = 0
order by RecId
go
```

- Индексирование части данных
  - Меньший размер индекса
  - Меньшая стоимость поддержки
- Проблемы
  - Ограничения в фильтрах
  - Оптимизатор очень консервативен при выборе индекса
- Сценарии использования
  - Индексирование для специфических значений ключа
  - Таблицы со SPARSE полями
  - Уникальность на части значений
    - Например уникальность (not null) значений

Professional Association for SQL Server

24 HOURS OF PASS

Демонстрация

Фильтрованные индексы

Секционированные Таблицы (Partitioned Tables)

```
create table dbo.Data
(
    ID int not null,
    DateCreated datetime not null,
    DateModified datetime not null,
    -- ...
)
```

```
create unique clustered index IDX_CI
on dbo.Data(ID)
```

ID: 1 DC: 01.01 08:00 DM: 01.01 22:30	ID: 2 DC: 01.01 15:00 DM: 01.01 21:20	ID: 3 DC: 02.01 07:00 DM: 05.01 15:15	ID: 4 DC: 03.01 09:45 DM: 04.01 10:25	ID: 5 DC: 03.01 16:30 DM: 04.01 07:10	ID: 6 DC: 05.01 11:55 DM: 09.01 08:20	ID: 7 DC: 05.01 12:25 DM: 05.01 13:20	ID: 8 DC: 06.01 18:40 DM: 07.01 19:00	ID: 9 DC: 07.01 12:23 DM: 07.01 22:12	ID: 10 DC: 08.01 19:30 DM: 09.01 05:21

```
create nonclustered index IDX_NCI
on dbo.Data(DateModified)
include(DateCreated)
```

DM: 01.01 21:20 ID: 2 *DC: 01.01 15:00*	DM: 01.01 22:30 ID: 1 *DC: 01.01 08:00*	DM: 04.01 07:10 ID: 5 *DC: 03.01 16:30*	DM: 04.01 10:25 ID: 4 *DC: 03.01 09:45*	DM: 05.01 13:20 ID: 7 *DC: 05.01 12:25*	DM: 05.01 15:15 ID: 3 *DC: 02.01 07:00*	DM: 07.01 19:00 ID: 8 *DC: 06.01 18:40*	DM: 07.01 22:12 ID: 9 *DC: 07.01 12:23*	DM: 09.01 05:21 ID: 10 *DC: 08.01 19:30*	DM: 09.01 08:20 ID: 6 *DC: 05.01 11:55*

Секционированные Таблицы (Partitioned Tables)


```
create partition function pfData(datetime)
as range right
for values('2012-01-03','2012-01-06')
go

create partition scheme psData
as partition pfData
all to ([primary])
go

create table dbo.Data
(
    ID int not null,
    DateCreated datetime not null,
    DateModified datetime not null,
    -- ...
)
go

create unique clustered index IDX_CI
on dbo.Data(ID,DateCreated)
on pfData(DateCreated)
go

create nonclustered index IDX_NCI
on dbo.Data(DataModified(DateCreated)
on pfData(DateCreated)
```

Секционированные Таблицы (Partitioned Tables)

- Таблица (Кластерный и выровненные (aligned) некластерные индексы) состоят из множества «физических» индекс-секций
- Индекс отсортирован в пределах секции
  - И это создает очень интересные проблемы

Секционированные Таблицы (Partitioned Tables)

```
create table dbo.Data
(
    ID int not null,
    DateCreated datetime not null,
    DateModified datetime not null,
    -- ...
)
```

```
create unique clustered index IDX_CI
on dbo.Data(ID)
```

ID: 1 DC: 01.01 08:00 DM: 01.01 22:30	ID: 2 DC: 01.01 15:00 DM: 01.01 21:20	ID: 3 DC: 02.01 07:00 DM: 05.01 15:15	ID: 4 DC: 03.01 09:45 DM: 04.01 10:25	ID: 5 DC: 03.01 16:30 DM: 04.01 07:10	ID: 6 DC: 05.01 11:55 DM: 09.01 08:20	ID: 7 DC: 05.01 12:25 DM: 05.01 13:20	ID: 8 DC: 06.01 18:40 DM: 07.01 19:00	ID: 9 DC: 07.01 12:23 DM: 07.01 22:12	ID: 10 DC: 08.01 19:30 DM: 09.01 05:21

```
create nonclustered index IDX_NCI
on dbo.Data(DateModified)
include(DateCreated)
```

```
select top 3 *
from dbo.Data
where DateModified > '2012-01-05'
order by DateModified, ID
```

DM: 01.01 21:20 ID: 2 *DC: 01.01 15:00*	DM: 01.01 22:30 ID: 1 *DC: 01.01 08:00*	DM: 04.01 07:10 ID: 5 *DC: 03.01 16:30*	DM: 04.01 10:25 ID: 4 *DC: 03.01 09:45*	DM: 05.01 13:20 ID: 7 *DC: 05.01 12:25*	DM: 05.01 15:15 ID: 3 *DC: 02.01 07:00*	DM: 07.01 19:00 ID: 8 *DC: 06.01 18:40*	DM: 07.01 22:12 ID: 9 *DC: 07.01 12:23*	DM: 09.01 05:21 ID: 10 *DC: 08.01 19:30*	DM: 09.01 08:20 ID: 6 *DC: 05.01 11:55*

Секционированные Таблицы (Partitioned Tables)

```
select top 3 *
from dbo.Data
where DateModified > '2012-01-05'
order by DateModified, ID
```

DC < 03.01

ID: 1 DC: 01.01 08:00 DM: 01.01 22:30	ID: 2 DC: 01.01 15:00 DM: 01.01 21:20	ID: 3 DC: 02.01 07:00 DM: 05.01 10:05
DM: 01.01 21:20 ID: 2 *DC: 01.01 15:00*	DM: 01.01 22:30 ID: 1 *DC: 01.01 08:00*	DM: 05.01 15:15 ID: 3 *DC: 02.01 07:00*

DC >= 03.01
DC < 06.01

ID: 4 DC: 03.01 09:45 DM: 04.01 10:25	ID: 5 DC: 03.01 16:30 DM: 04.01 07:10	ID: 6 DC: 05.01 11:55 DM: 09.01 08:20	ID: 7 DC: 05.01 12:25 DM: 05.01 13:20
DM: 04.01 07:10 ID: 5 *DC: 03.01 16:30*	DM: 04.01 10:25 ID: 4 *DC: 03.01 09:45*	DM: 05.01 13:20 ID: 7 *DC: 05.01 12:25*	

DC >= 06.01

ID: 8 DC: 06.01 18:40 DM: 07.01 19:00	ID: 9 DC: 07.01 12:23 DM: 07.01 22:12	ID: 10 DC: 08.01 19:30 DM: 09.01 05:21	
DM: 07.01 19:00 ID: 8 *DC: 06.01 18:40*	DM: 07.01 22:12 ID: 9 *DC: 07.01 12:23*	DM: 09.01 05:21 ID: 10 *DC: 08.01 19:30*	DM: 09.01 08:20 ID: 6 *DC: 05.01 11:55*

DM: 05.01 13:20 ID: 7 *DC: 05.01 12:25*	DM: 05.01 15:15 ID: 3 *DC: 02.01 07:00*	DM: 07.01 19:00 ID: 8 *DC: 06.01 18:40*	DM: 07.01 22:12 ID: 9 *DC: 07.01 12:23*	DM: 09.01 05:21 ID: 10 *DC: 08.01 19:30*

Professional Association for SQL Server

24 HOURS OF PASS

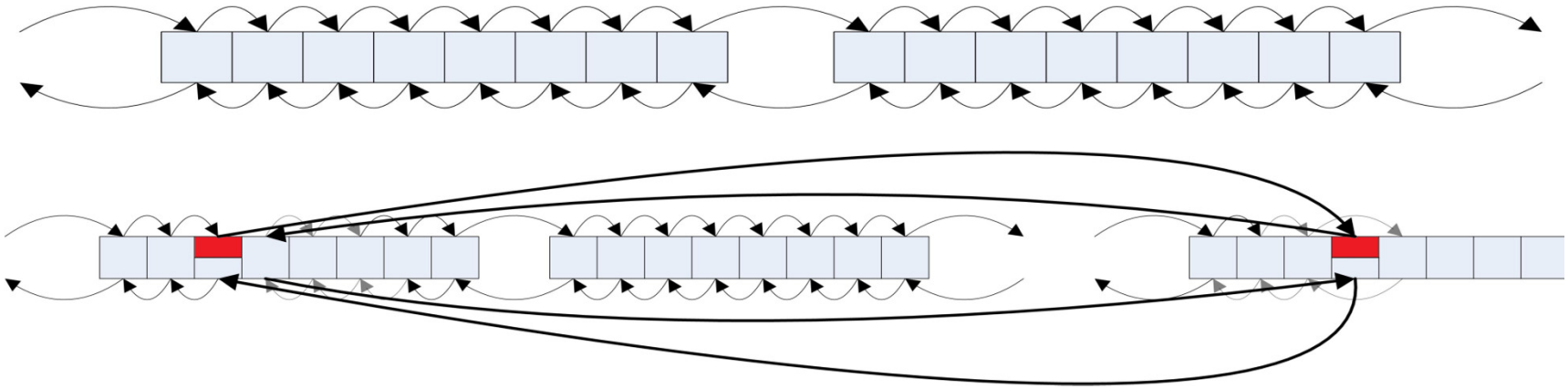
Демонстрация

Секционированные индексы

Фрагментация

• • •

Разделение страниц (Page Split)



- SQL Server переносит *примерно* половину данных со страницы в новое место
- FILLFACTOR (% **начального** заполнения страницы) позволяет зарезервировать место на страницах при создании индекса
  - FILLFACTOR используется только при CREATE / ALTER REBUILD
  - Уменьшение FILLFACTOR уменьшает количество разделений страниц, но увеличивает размер (число страниц) индекса

Professional Association for SQL Server

24 HOURS OF PASS

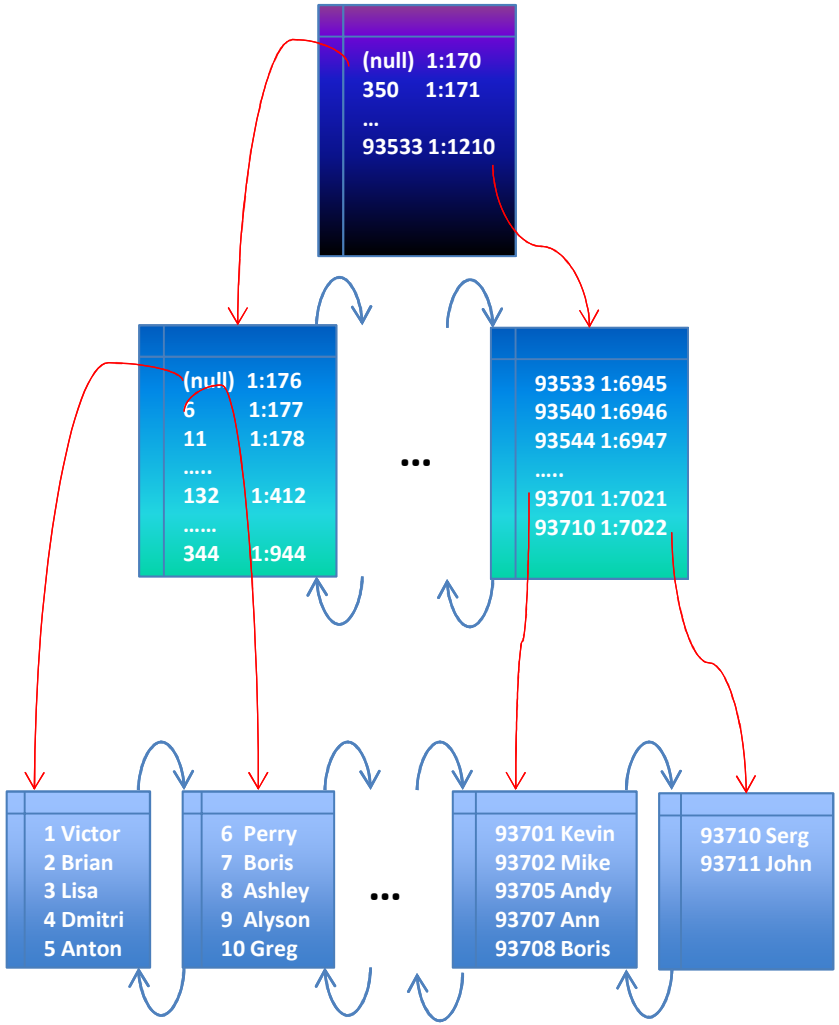
Демонстрация

Паттерны приводящие к фрагментации

Паттерны приводящие к фрагментации

- Insert/Update - увеличение размера записи после вставки
- Read Committed Snapshot / Snapshot
- Индексы на «Случайные» значения
  - Uniqueidentifier
  - HashBytes

Фрагментация и система



- Фрагментация плоха при сканировании большого числа страниц
  - Data Warehouse / DSS
  - Неоптимизированные запросы
- Фрагментация менее страшна при поиске одной (или нескольких) записей
  - OLTP
- Фрагментация уменьшает производительность групповых операций модификации данных
- Низкий % заполнения страниц увеличивает количество страниц в индексе и использует лишнюю память в кэше

Поддержка индексов (Index Maintenance)

- Sys.dm_db_index_physical_stats
  - **avg_fragmentation_in_percent**
  - **avg_page_space_used_in_percent**
- REBUILD
  - Пересоздает индекс
  - Online операция с Enterprise Edition
  - Рекомендуется при фрагментации >30% (*)
- REORGANIZE
  - Логическая дефрагментация
  - Online операция
  - Рекомендуется при фрагментации от 5% до 30% (*)
- (*) – Books Online как «средняя температура по больнице»

Поддержка индексов (Index Maintenance)

- Фрагментация – это плохо, но..
- Факторы которые следует принять во внимание:
  - Каков % заполнения страниц?
  - Тип системы: OLTP, DSS/Data Warehouse, Смешанные
  - Работает ли система 24x7x365?
  - SQL Server edition – Enterprise edition позволяет поддерживать индексы online

Стратегии Индексации

• • •

Кластерный Индекс

- Первичный Ключ (primary key) != Кластерный Индекс
  - Первичный ключ – элемент логического дизайна
  - Кластерный индекс – элемент физического дизайна
- В идеале:
  - Кластерный Индекс «оптимизирует» самые важные запросы
  - Минимизирует разделение страниц (page split) / фрагментацию
- Физические требования (не существует правил без исключений):
  - Статичный (не меняется)
  - Имеет небольшой размер ключа
  - Уникальный
    - В случае если кластерный индекс не определен как уникальный, SQL Server добавляет внутреннее поле uniquifier int nullable
      - От 0 до 8 байт дополнительной информации

Professional Association for SQL Server

24 HOURS OF PASS

Демонстрация

Неуникальный Кластерный Индекс

Кластерный Индекс (Identity / Uniqueidentifiers)

- Уникальны, статичны, имеют небольшой размер ключа
- Потенциальные проблемы
  - «Горячие точки» (hotspots) – задержки (сериализация) при создании новых страниц и участков (extents)
  - Редко помогают «оптимизировать» запросы
- Потенциальные сценарии использования
  - Справочные таблицы
    - Ограниченный объем данных
    - Ссылочная целостность (foreign keys)
  - Отсутствие других кандидатов для кластерного индекса
- Identity - 4 / 8 байт. Uniqueidentifier – 16 байт
  - NEWID() – проблемы с фрагментацией и производительностью

Кластерный Индекс («Логическое Секционирование»)

```
create table dbo.Positions
(
    CompanyId int not null,
    UTCTimeTag datetime2(0) not null,
    RecId bigint not null,
    DeviceId int not null,
    Latitude decimal(9,6) not null,
    Longitude decimal(9,6) not null
)
go

create unique clustered index IDX_CI
on dbo.Positions
(CompanyId, UTCTimeTag, RecId)
go

create nonclustered index IDX_NCI
on dbo.Positions
(CompanyId, DeviceId, UTCTimeTag)
```

- Левое поле индекса – определяет секцию
- Преимущества:
  - Оптимизация запросов работающих в пределах секций
  - Локализация IO
  - Возможное уменьшение чисда разделений страниц / фрагментации при условии монотонного увеличения правых полей индекса
- Недостаток:
  - Потенциальные проблемы со статистикой на композитном индексе

Некластерный Индекс (Уникальность)

- Unique Constraint != Unique Index
  - Unique Constraint – элемент логического дизайна
  - Unique Index – элемент физического дизайна
  - В обоих случаях SQL Server использует уникальный индекс
    - Unique Constraint не может включать доп. поля (INCLUDE)
- Уникальный индекс обладает более эффективной структурой
  - Промежуточные и корневой (Intermediate/Root) уровни не содержат значений идентификатора записи (кластерного ключ/RID)
- Уникальный индекс помогает оптимизатору

Некластерный Индекс (Использование)

- В общем случае, SQL Server редко использует более одного некластерного индекса (пересечение индексов – index intersection) на таблицу в запросе
- Композитные индексы с дополнительными полями гораздо более универсальны нежели множество узких индексов
  - Не забываем о затратах на обновление и поддержку индексов

Некластерный Индекс (Число индексов на таблицу)

- OLTP системы
  - Множество коротких транзакции
  - Постоянное обновление данных
  - Оптимизированные запросы (index seek)
    - Ограниченное число индексов
- Data Warehouse / Decision Support Systems
  - Ограниченное число сложных запросов
  - Данные обновляются исходя из расписания
  - Сложные запросы, частое сканирование таблиц / индексов
    - Количество индексов зависит от запросов и стратегии обновления данных
      - Удаление и пересоздание индексов во время загрузки
      - Columnstore индексы

Индексация (Нестандартные ситуации)

- Значение ключа > 900 байт
  - Создаем и индексируем persistent калькулируемое поле (CHECKSUM, HASHBYTES)
  - Обязательно добавляем оригинальное значение в селект
- Поиск по подстроке
  - Создание таблицы «подстрок» - паттернов

Professional Association for SQL Server

Демонстрация

Индексы на полях > 900 байт

Проиндексированные Представления (Indexed View)

- Обычные представления (views) – метадата
- Проиндексированные представления – физическая копия данных
- Проблемы
  - Множество ограничений при создании
  - Затраты на поддержку
  - Поведение зависит от редакции SQL Server
    - Standard Edition требует NOEXPAND hint

Проиндексированные Представления (Indexed View)

- Сценарии использования
  - Агрегаты (Data Warehouse / DSS)
  - Уменьшение числа join в редкомодифицируемых таблицах
  - Оптимизация стороннего кода (Enterprise Edition)

Professional Association for SQL Server

24 HOURS OF PASS

Демонстрация

Indexed Views

Стратегии Оптимизации

• • •

Стратегия Оптимизации Систем (Разработка системы с нуля)

- Шаг 1 - Создание минимально необходимого набора индексов
  - Первичные ключи и кластерные индексы
  - Поддержка уникальности
  - Индексы для поддержки ссылочной целостности (foreign keys)
  - Индексы для наиболее приоритетных запросов (если известны)

Стратегия Оптимизации Систем (Оптимизация сущестующих систем)

- Шаг 1
  - Удаление избыточных индексов
  - Удаление неиспользуемых индексов
  - Тривиальная консолидация
  - Анализ проблематичных индексов

Стратегия Оптимизации Систем (Оптимизация сущестующих систем)

- Шаг 1
  - **Удаление избыточных индексов**
  - Удаление неиспользуемых индексов
  - Тривиальная консолидация
  - Анализ проблематичных индексов

```
create nonclustered index IDX_1
on dbo.Customers(LastName, FirstName)
```

~~create nonclustered index IDX_2~~
~~on dbo.Customers(LastName)~~

- В любом правиле есть исключения
  - Размер ключа может иметь значение

Стратегия Оптимизации Систем (Оптимизация существующих систем)

- Шаг 1
  - Удаление избыточных индексов
  - **Удаление неиспользуемых индексов**
  - Тривиальная консолидация
  - Анализ проблематичных индексов

```
select *
from sys.dm_db_index_usage_stats us
where
    us.database_id = DB_ID() and
    us.object_id = OBJECT_ID(N'dbo.Orders')
order by
    us.index_id
```

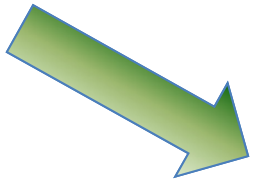
index_id	user_seeks	user_scans	user_lookups	user_updates	last_user_seek
1	27341	2578	28123	9921	2012-03-23 15:27:10.270
2	5368	0	0	299	2012-03-23 15:26:55.697
3	302	0	0	46	2012-03-23 15:21:06.557
5	1677	0	0	49	2012-03-23 15:27:05.317
8	21065	0	0	49	2012-03-23 15:27:08.737
50	0	0	0	28	NULL
51	0	2382	0	60	NULL

- Sys.dm_dm_index_operational_stats – более детализованая информация
- Данные очищаются при перезапуске сервера

Стратегия Оптимизации Систем (Оптимизация существующих систем)

- Шаг 1
  - Удаление избыточных индексов
  - Удаление неиспользуемых индексов
  - **Тривиальная консолидация**
  - Анализ проблематичных индексов

```
create nonclustered index IDX_1
on dbo.Customers(LastName, FirstName)
include(DateOfBirth)

create nonclustered index IDX_2
on dbo.Customers(LastName)
include(Phone)
```



```
create nonclustered index IDX_3
on dbo.Customers(LastName, FirstName)
include(Phone, DateOfBirth)
```

Стратегия Оптимизации Систем (Оптимизация существующих систем)

- Шаг 1
  - Удаление избыточных индексов
  - Удаление неиспользуемых индексов
  - **Тривиальная консолидация**
  - Анализ проблематичных индексов

```
create nonclustered index IDX_4
on dbo.Orders(OrderDate, Total)

create nonclustered index IDX_5
on dbo.Orders(OrderDate, WarehouseId)
```

```
create nonclustered index IDX_6
on dbo.Orders(OrderDate, Total)
include(WarehouseId)
```

```
create nonclustered index IDX_7
on dbo.Orders(OrderDate, WarehouseId)
include(Total)
```

Стратегия Оптимизации Систем (Оптимизация существующих систем)

- Шаг 1
  - Удаление избыточных индексов
  - Удаление неиспользуемых индексов
  - Тривиальная консолидация
  - **Анализ проблематичных индексов**

```
select *
from sys.dm_db_index_physical_stats(DB_ID(),Object_Id(N'dbo.Orders'),null, null, 'DETAILED')
```

index_id	avg_fragmentation_in_percent	fragment_count	avg_fragment_size_in_pages	page_count	avg_page_space_used_in_percent
3	98.5034013605442	732	1.00409836065574	735	64.8512972572276
51	3.53982300884956	19	23.7894736842105	452	97.8963800345935

Стратегия Оптимизации Систем (Оптимизация существующих систем)

- Шаг 1
  - Удаление избыточных индексов
  - Удаление неиспользуемых индексов
  - Тривиальная консолидация
  - **Анализ проблематичных индексов**

```
select *
from sys.dm_db_index_operational_stats(DB_ID(),Object_Id(N'dbo.Users'),null, null)
```

index_id	leaf_insert_count	leaf_update_count	singleton_lookup_count	range_scan_count	row_lock_count	page_io_latch_wait_count	page_io_latch_wait_in_ms
1	18	27457	326612	0	88905	172	2782
2	307	0	0	58715693	1158	207	2075
3	18	0	304	0	36	17	673
5	23	0	0	1767	51	9	329
8	20	3	1471	257	148	11	516
50	0	0	0	0	0	0	0
51	125	0	0	0	357	4	62
100	307	0	0	42526	903	7	514

Стратегия Оптимизации Систем (Оптимизация)

- Поиск проблематичных запросов
- Анализ проблематичного кода
- Добавление необходимых индексов

Стратегия Оптимизации Систем (Оптимизация)

- **Поиск проблематичных запросов**
  - Data Management View (DMV)
  - SQL Profiler
  - Performance Data Collectors / Management Data Warehouse
  - Утилиты от сторонних разработчиков
  - ...
- Анализ проблематичного кода
- Добавление необходимых индексов

Стратегия Оптимизации Систем
(Оптимизация – sys.dm_exec_query_stats)

```
SELECT TOP 50
    SUBSTRING(qt.TEXT, (qs.statement_start_offset/2)+1,
        ((
            CASE qs.statement_end_offset
                WHEN -1 THEN DATALENGTH(qt.TEXT)
                ELSE qs.statement_end_offset
            END - qs.statement_start_offset)/2)+1) as [SQL],
    qs.execution_count,
    (qs.total_logical_reads + qs.total_logical_writes)
        / qs.execution_count as [Avg IO],
    qp.query_plan,
    qs.total_logical_reads, qs.last_logical_reads,
    qs.total_logical_writes, qs.last_logical_writes,
    qs.total_worker_time,
    qs.last_worker_time,
    qs.total_elapsed_time/1000 total_elapsed_time_in_ms,
    qs.last_elapsed_time/1000 last_elapsed_time_in_ms,
    qs.last_execution_time
FROM
    sys.dm_exec_query_stats qs
        OUTER APPLY sys.dm_exec_sql_text(qs.sql_handle) qt
        OUTER APPLY sys.dm_exec_query_plan(qs.plan_handle) qp
ORDER BY
    [Avg IO] DESC
```

	SQL	execution_c...	Avg IO	query_plan	total_logical_reads	last_logical_reads	total_logical_writes	last_logical_writes	total_worker_time	la
1	select top 1000 RecId, UtcTimeTag,...	1	1324346	<ShowPlanXML xmlns="http:/...	1324346	1324346	0	0	2254882	2
2	select NEWID() as ID, p.COMPA...	3	218543	<ShowPlanXML xmlns="http:/...	655631	218544	0	0	37335939	1
3	select count(*) from GPS.V$NEX...	1	191254	<ShowPlanXML xmlns="http:/...	191254	191254	0	0	2510740	2
4	with MinMax (MinRec, MaxRec) as ...	1	147437	<ShowPlanXML xmlns="http:/...	147435	147435	2	2	1310547	1
5	with MinMax (MinRec, MaxRec) as ...	1	118663	<ShowPlanXML xmlns="http:/...	118663	118663	0	0	772461	1

Стратегия Оптимизации Систем (Оптимизация)

- Поиск проблематичных запросов
- **Анализ проблематичного кода**
  - Насколько оптимален код?
  - Обновлена ли статистика?
  - Помогает ли рекопиляция (parameter sniffing)?
- Добавление необходимых индексов

Стратегия Оптимизации Систем (Инструментарий)

- Missing Indexes DMV
  - Рекоммендуют индексы которые помогают только текущему запросу и текущему плану
- Database Tuning Advisor (DTA)
  - Анализируется только та активность, которая включена в trace
- Можно использовать как вспомогательные инструменты, анализируя результаты их работы

Professional Association for SQL Server

24 HOURS OF PASS

Демонстрация

Оптимизация систем

Q&A

- Большое спасибо за внимание!

- Email: dmitri@aboutsqlserver.com
- Примеры доступны для скачивания
  - http://aboutsqlserver.com/presentations