Индексы
и
все, все, все

Часть 1

Давайте познакомимся

- 12+ лет опыта работы с Microsoft SQL Server
- Microsoft SQL Server MVP
- Microsoft Certified Master (SQL Server 2008)
- MCPD
  - Enterprise Application Developer

- Blog: http://aboutsqlserver.com
  - Презентация будет доступна для скачивания
- Email: dmitri@aboutsqlserver.com

Содержание

- Сегодня (часть 1):
  - Структура индексов
  - Как и когда SQL Server использует индексы
  - Дополнительные возможности
    - Индексы с дополнительными полями
    - Фильтрованные индексы
    - Секционированные индексы
- Через месяц (часть 2):
  - Фрагментация и поддержка индексов
  - Стратегии индексации
  - Стратегии оптимизации

Типы Систем

OLTP

- (Обычно) поддержка операционной активности
- Большое количество небольших и оптимизированных транзакций
- Статичные запросы
- Данные постоянно меняются

Data Warehouse

- (Обычно) системы анализа и отчетности
- Небольшое количество сложных и длительных транзакций
- Сложные, редко повторяющиеся запросы
- Редко меняющиеся данные

Страницы, участки, итд

- SQL Server хранит данные на 8K страницах
  - 8060 байт доступно для юзера
- 8 страниц сгруппированы в участок (Extent)
  - Смешанные участки (mixed extents) хранят данные из разных объектов
  - Унифицированные участки (uniform extents) хранят данные для одного объекта
- Первые 8 страниц объекта хранятся в смешанных участках. После этого используются только унифицированные участки
- Специальные страницы
  - GAM – используется ли участок
  - SGAM – есть ли в смешанном участке свободные страницы
  - **PFS – оценка свободного места на странице**
  - **IAM "Карта использования индекса" (Index Allocation Map) – какие участки принадлежат объекту**

Heap таблицы

- Таблицы без кластерных индексов
- SQL Server использует PFS страницы для оценки количества свободного места на странице
  - PFS «оценивает» занятое место следующим образом:
    - Пусто
    - 1-50%
    - 51-80%
    - 81-95%
    - 96%-100%

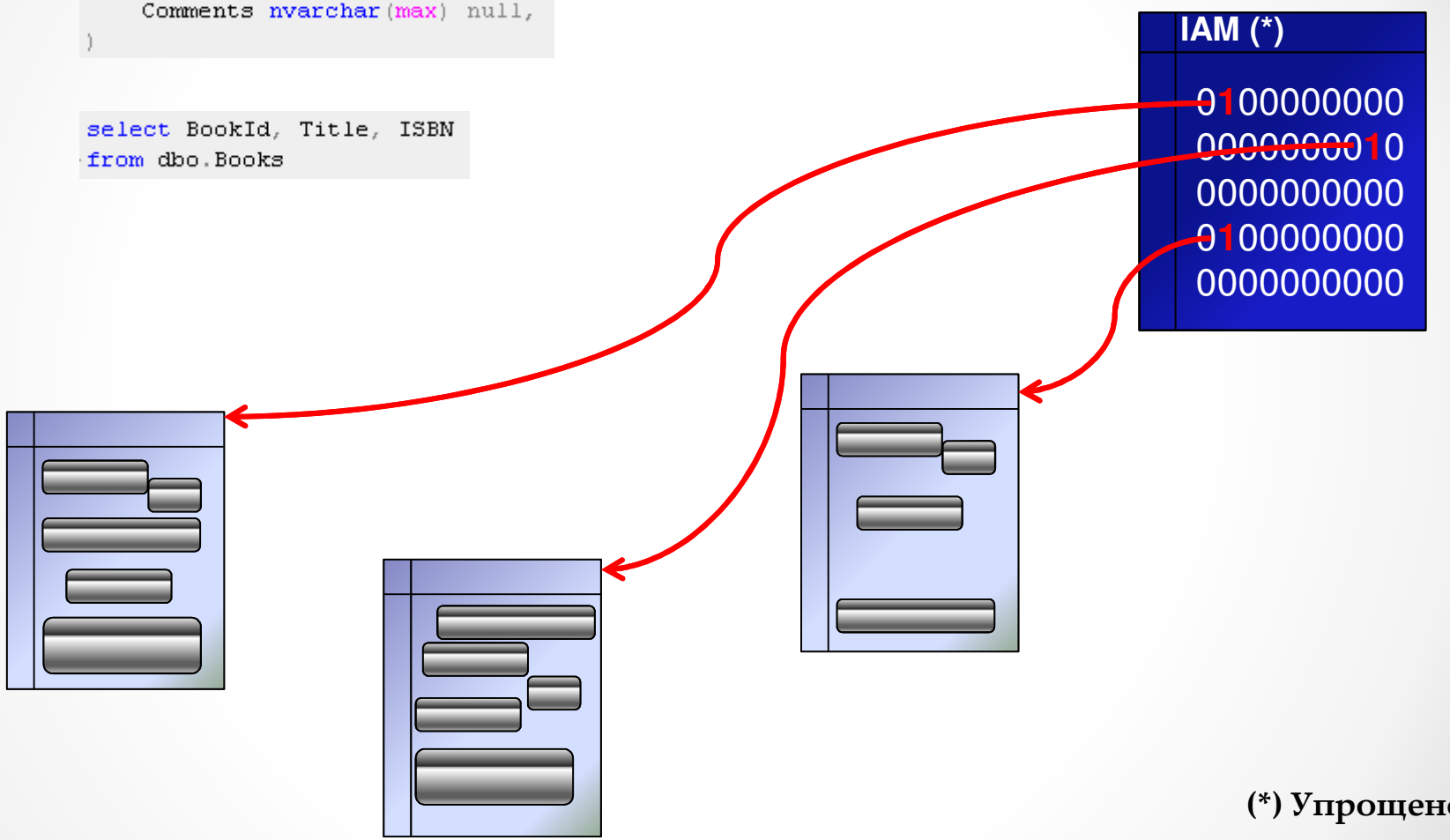
Heap таблицы: PFS

Демонстрация

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
```

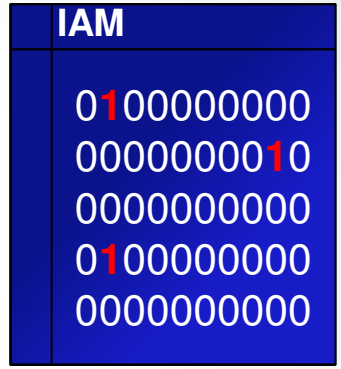


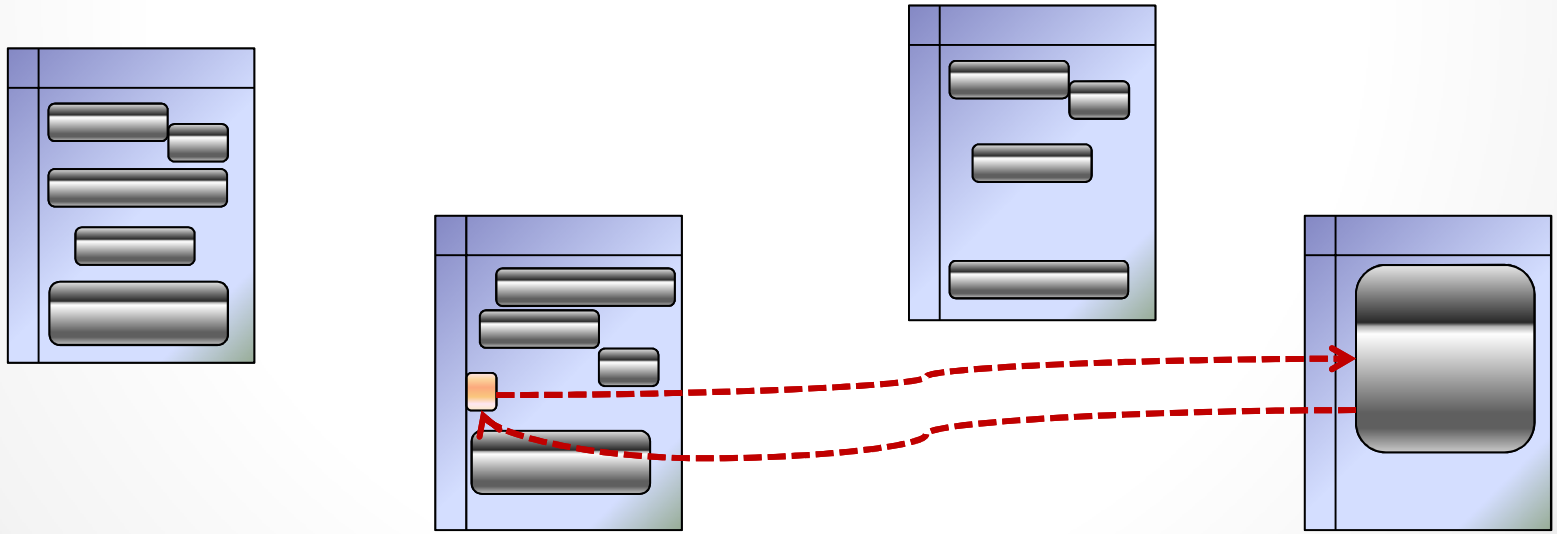
(*) Упрощено

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
```

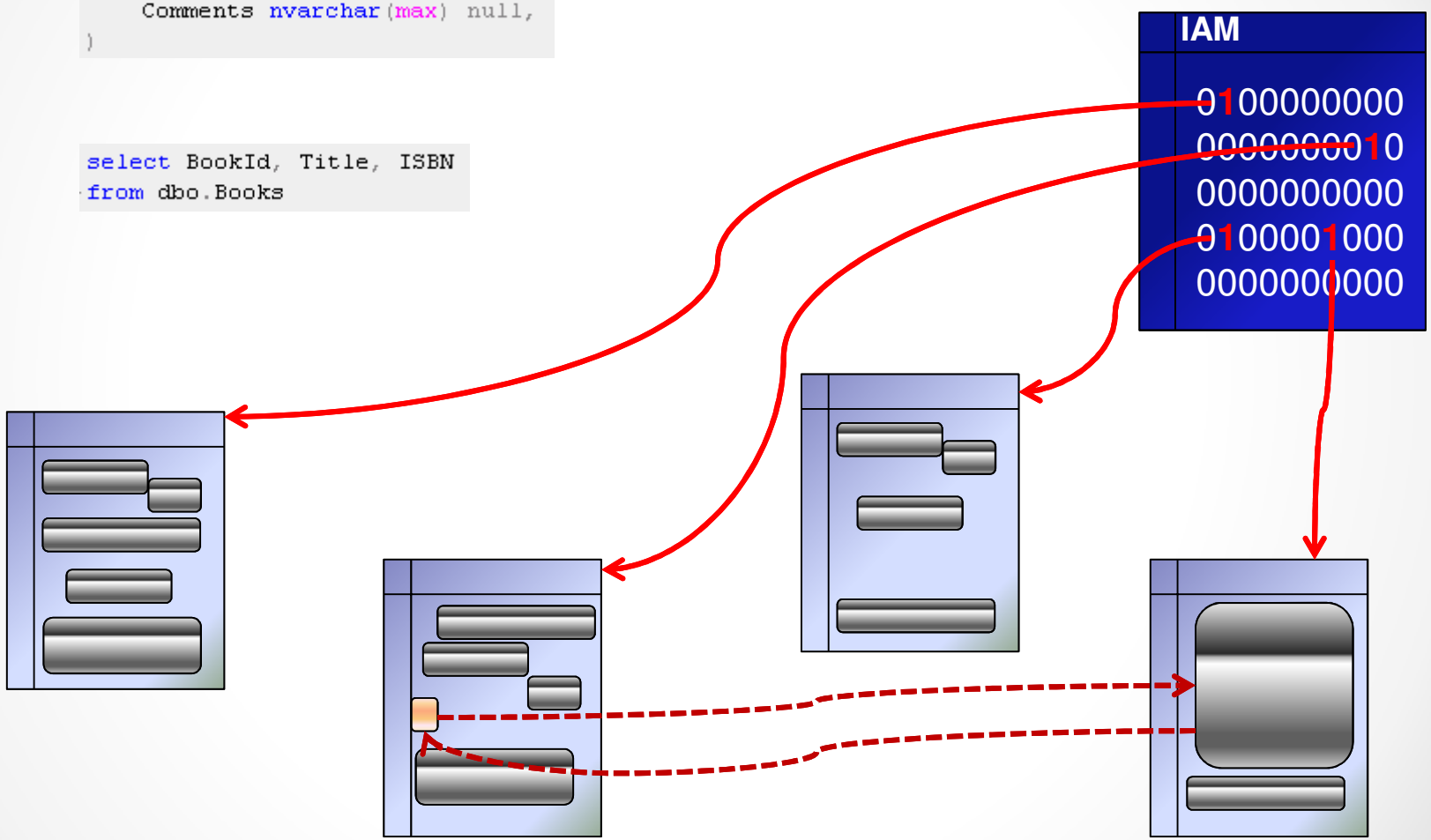




Heap Tables

```
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
```



Heap таблицы: Forwarding Pointers

Демонстрация

Heap таблицы

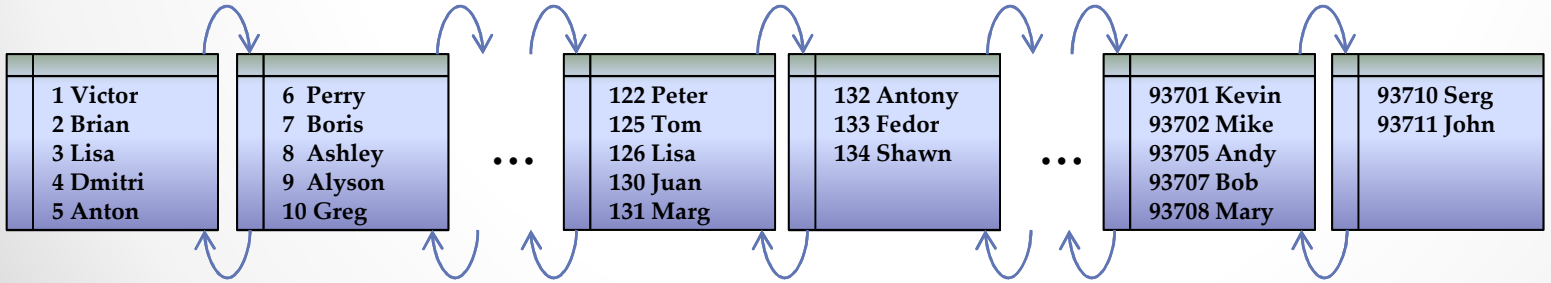
- Потенциальные проблемы
  - Дополнительные I/O операции из-за forwarding pointers
  - Неоптимальный контроль свободного места на страницах
- Использование:
  - Быстрая загрузка данных

Кластерный индекс

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

Heap таблица

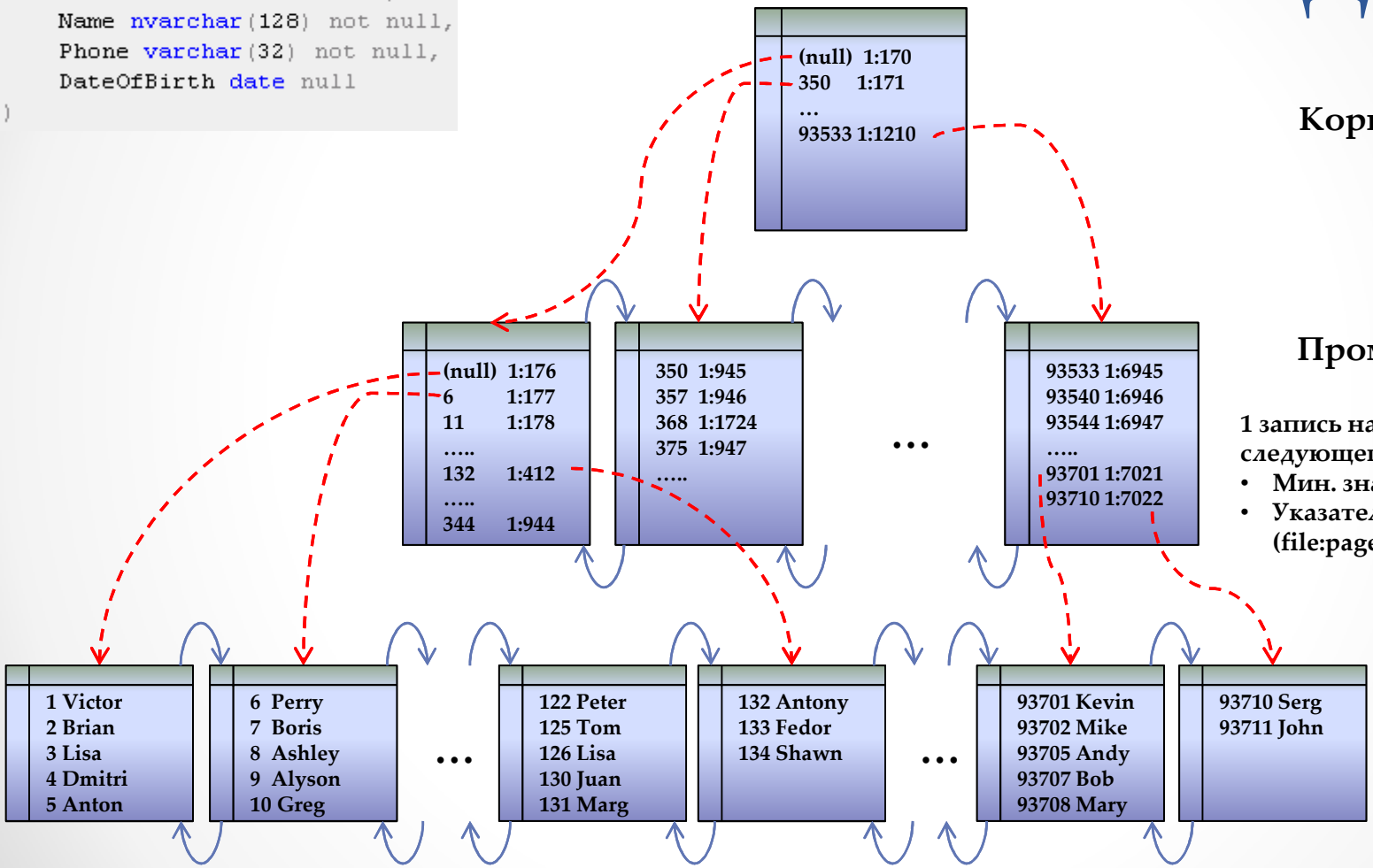
```
create unique clustered index
IDX_Customers_CustomerId
on dbo.Customers(CustomerId)
```



Уровень листьев (данные)

Кластерный индекс

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



Корневой уровень

Промежуточный уровень

1 запись на страницу со следующего уровня

- **Мин. значение ключа**
- **Указатель на страницу (file:page)**

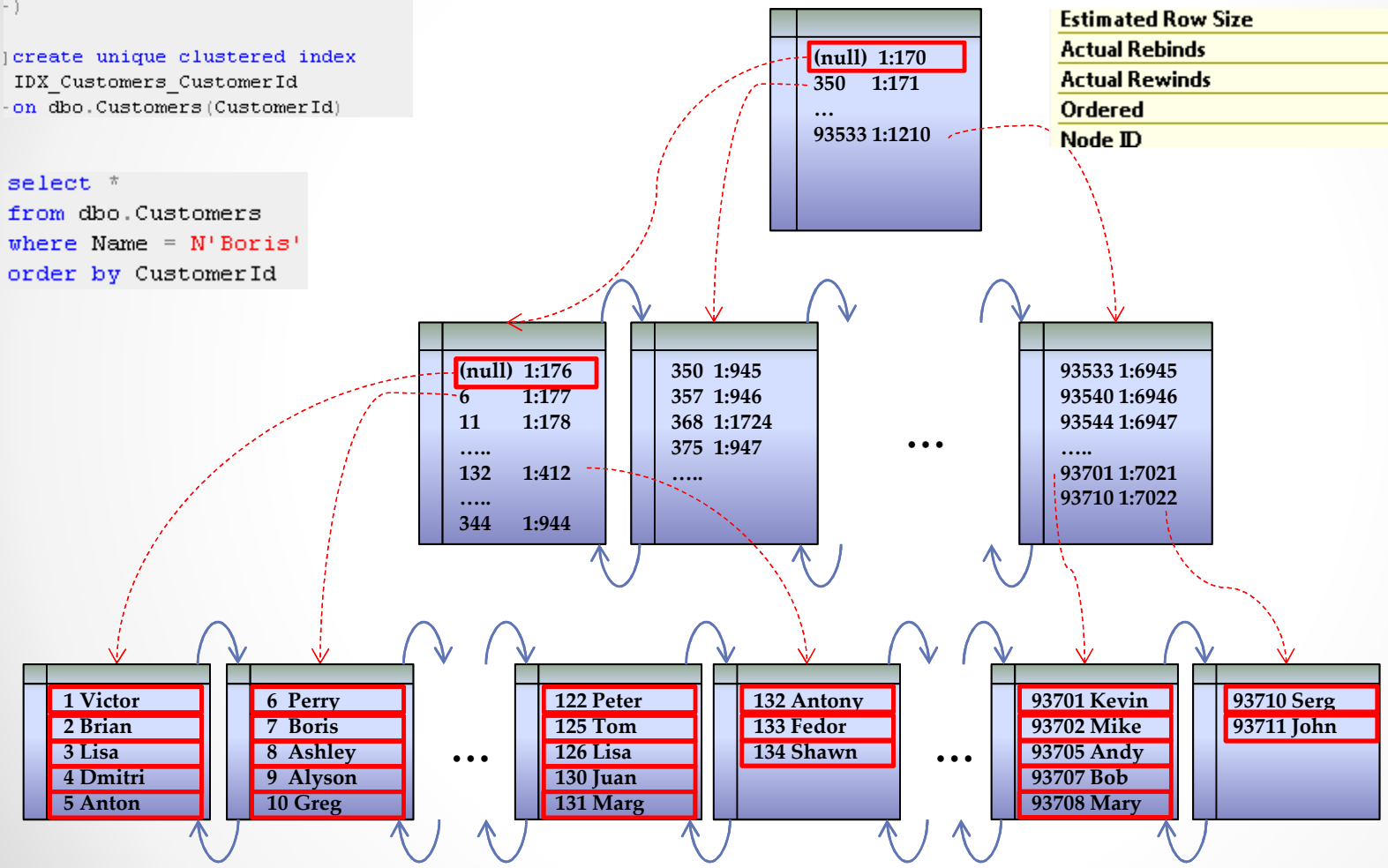
Уровень листьев (данные)

Сканирование

```
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
```



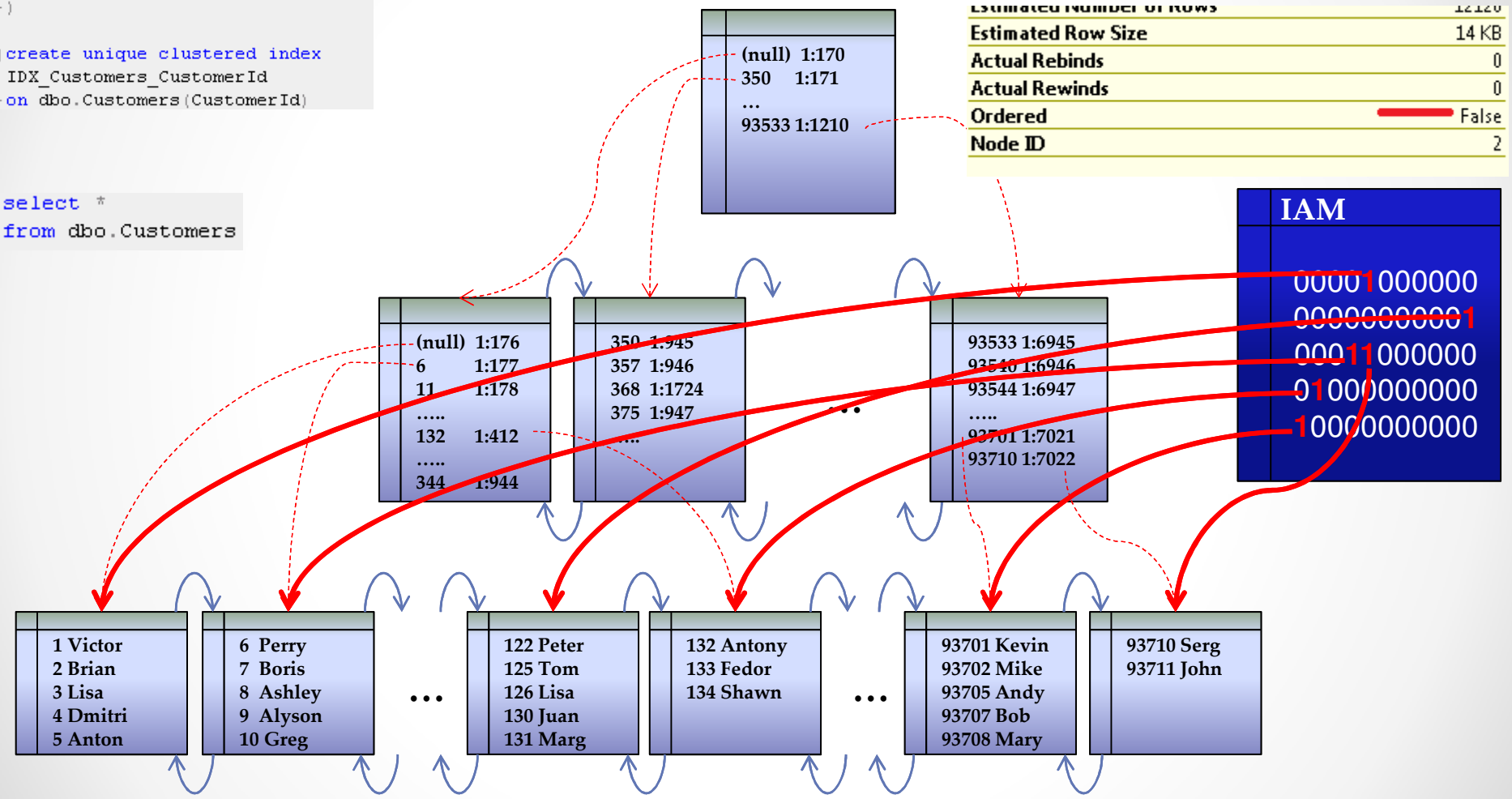
Estimated Row Size	14 KB
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	2

IAM сканирование

```
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
```

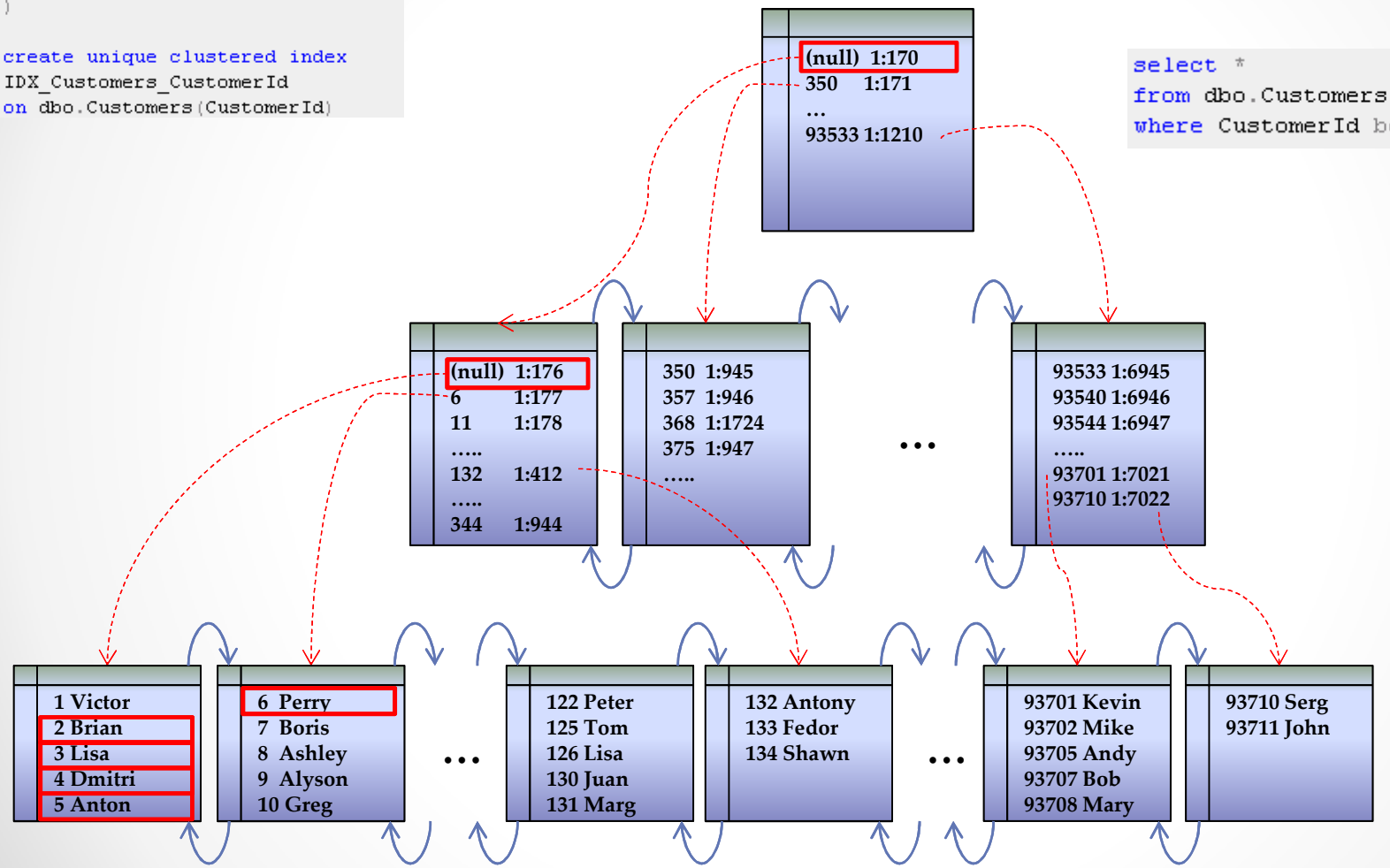


```
create table dbo.Customers
(
    CustomerId int not null,
    Name nvarchar(128) not null,
    Phone varchar(32) not null,
    DateOfBirth date null
)

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

Поиск

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



SARG (Seekable/Searchable Arguments)

- SARG предикаты, позволяющие сделать поиск

```
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'
```

- Non-SARG предикаты – сканирование

```
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'
```

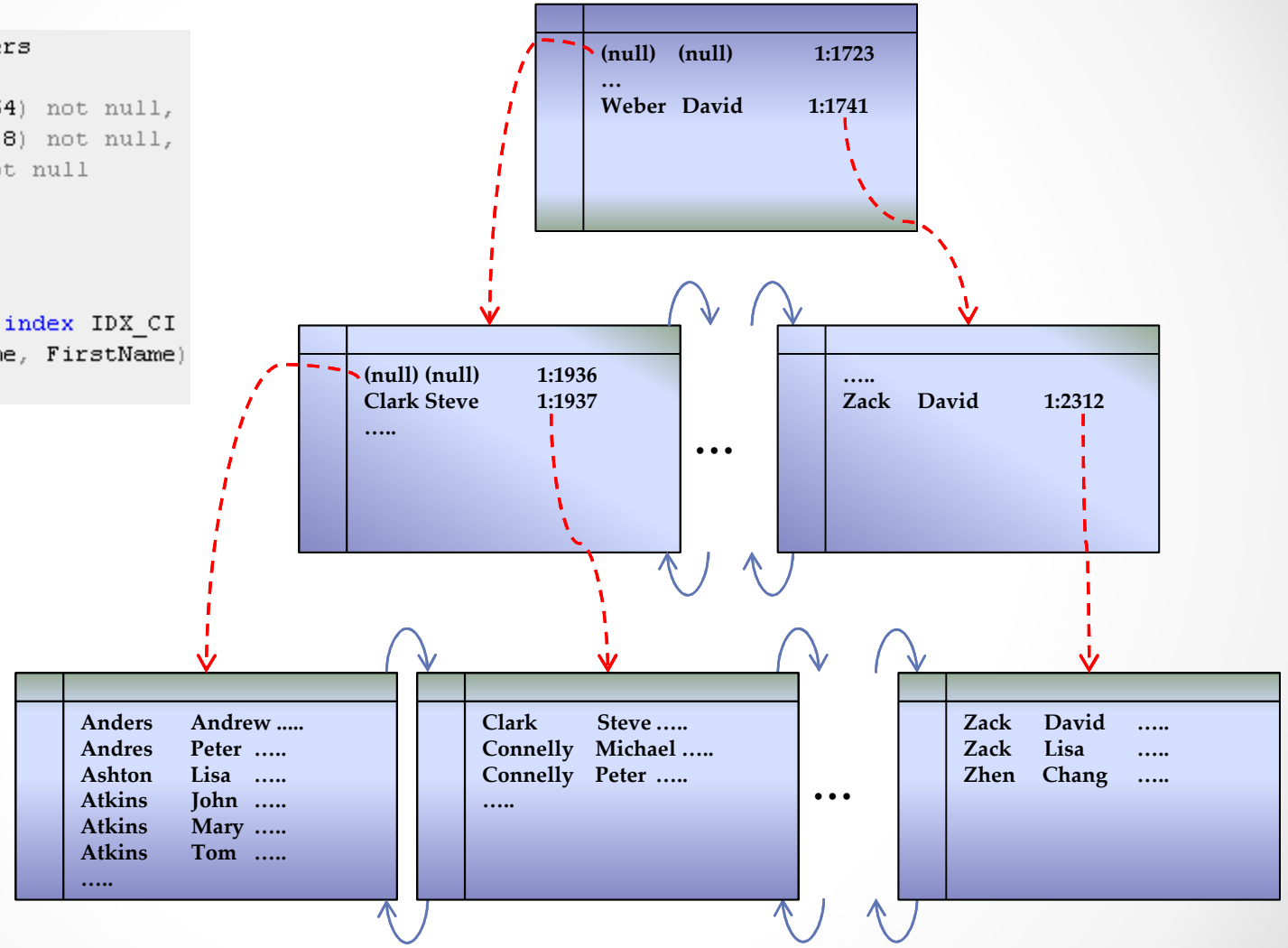
SARG предикаты и типы данных

Демонстрация

Композитные индексы

```
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
```



SARG для Композитных индексов

```
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
```

- SARG предикаты:

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

where LastName = N'Sanders'
```

SARG на «левых» столбцах индекса

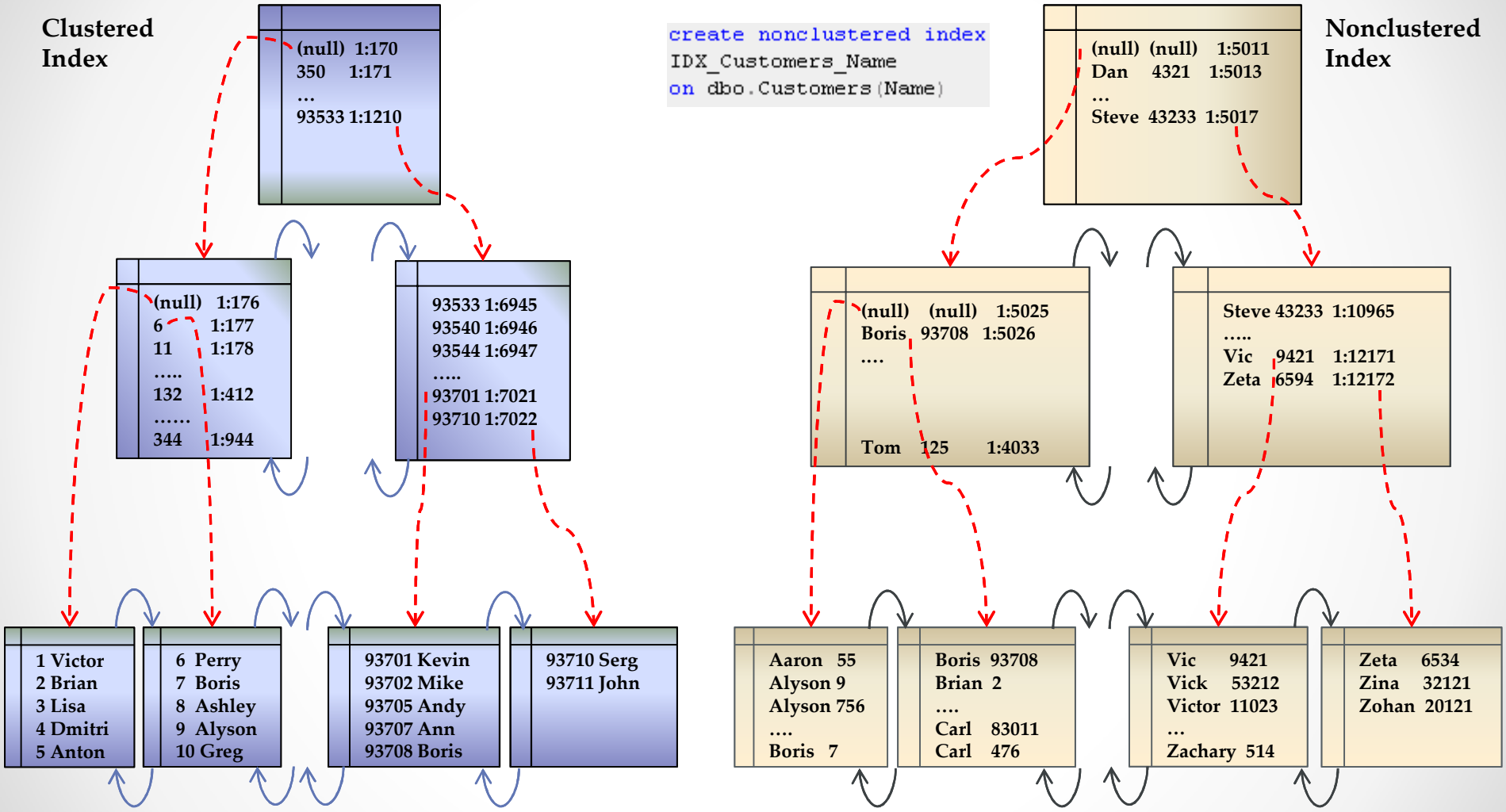
- Non-SARG предикаты:

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

where FirstName = N'Tom'
```

Остальные случаи

Некластерные индексы

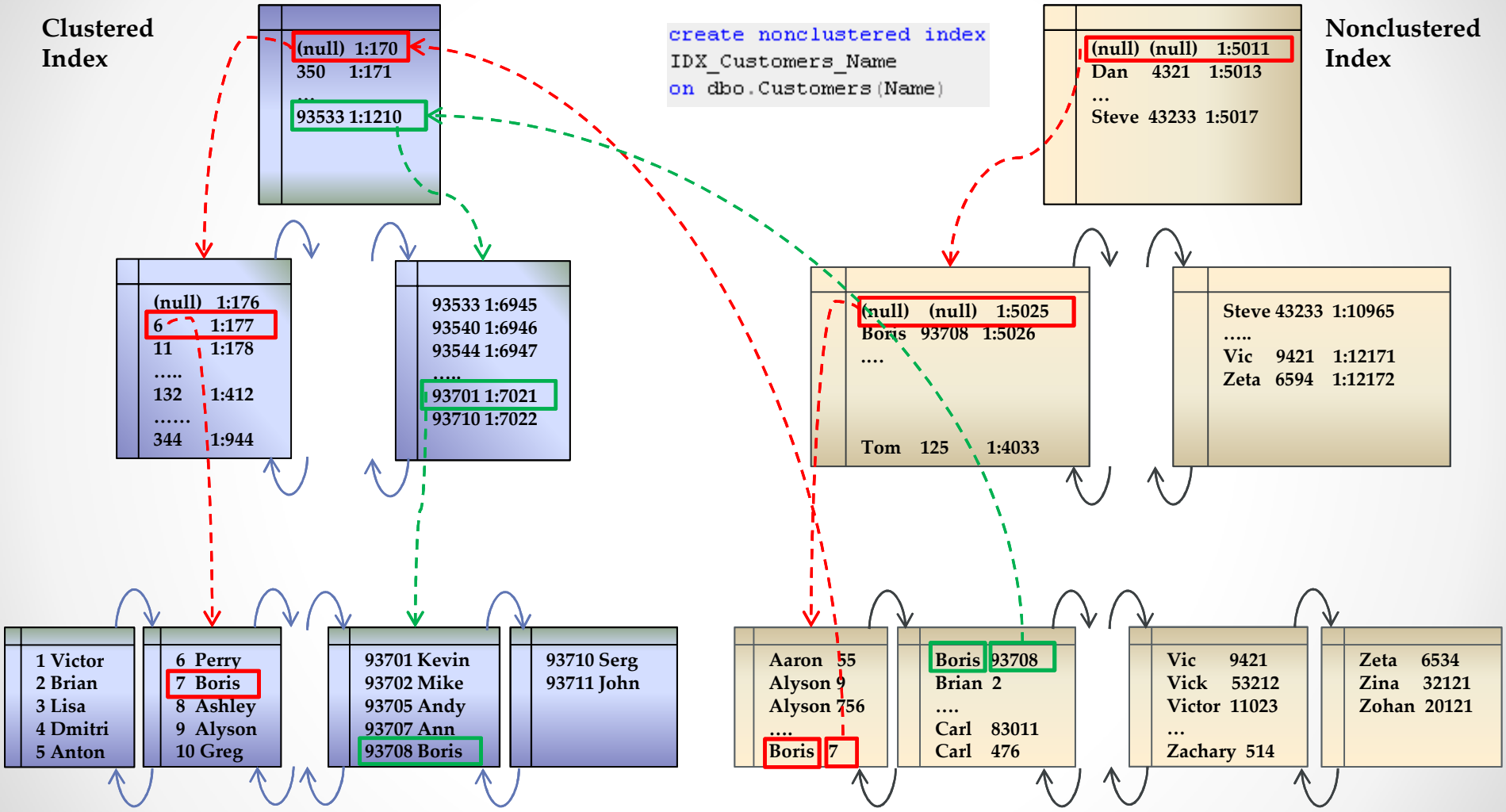


Структура некластерного индекса

- Уровень данных: Значение ключа + row id
- Row id:
  - Heap таблицы: row id = file:page:slot
  - **Таблицы с кластерным индексом: row id = значение ключа CI**
- Промежуточные и корневой уровни
  - Уникальные индексы: Указатель на страницу + минимальное значение ключа на странице
  - Неуникальные индексы: Указатель на страницу + минимальное значение ключа на странице + row id для этой записи
- Рекомендация: Определять индекс уникальным когда возможно (меньший размер ключа на промежуточных и корневом уровнях)

Использование NCI

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



Использование некластерных индексов

Демонстрация

Оптимизатор запросов

- **Оптимизатор не пытается найти «наилучший план выполнения запроса»**
- **Цель оптимизации: «Быстрый поиск приемлемого плана»**
- В общем случае:
  - Один из вариантов: сканирование таблицы / CI
  - Остальные варианты требуют оценку «стоимости» использования некластерного индекса
    - Один из основных критериев: сколько записей будет найдено → сколько требуется операций Key Lookup
    - SQL Server использует статистику для этой оценки

Статистика

Демонстрация

Статистика

- Гистограмма хранит не более 200 шагов
  - Больше записей в таблице → менее аккуратная статистика
- Автоматическое обновление статистики базируется от количества изменений первого (левого) столбца индекса
  - Добавление записей в пустую таблицу
  - <= 500 записей → 500 изменений
  - **> 500 записей → 500 + 20% от числа записей в таблице**
- TF 2371 (SQL Server 2008 R2 SP1+) изменяет условия пересчета
  - **Тестируйте перед применением в рабочих системах**
- Гистограмма хранится только для первого (левого) столбца индекса

Статистика и композитные индексы

```
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
```

- Варианты:
  - Сканирование кластерного индекса
  - Сканирование некластерного индекса + Key lookup
- Зависит от числа найденных записей (количество Key lookup операций)
- Решение: статистика на уровне столбцов
  - В некоторых случаях создается автоматически

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

Статистика на уровне столбцов

Демонстрация

Статистика и композитные индексы

```
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)
```

- Гистограмма хранится только для первого (левого) столбца
- DeviceId уникален в пределах компании
- Какой индекс выбрать?

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

Ключевые моменты:

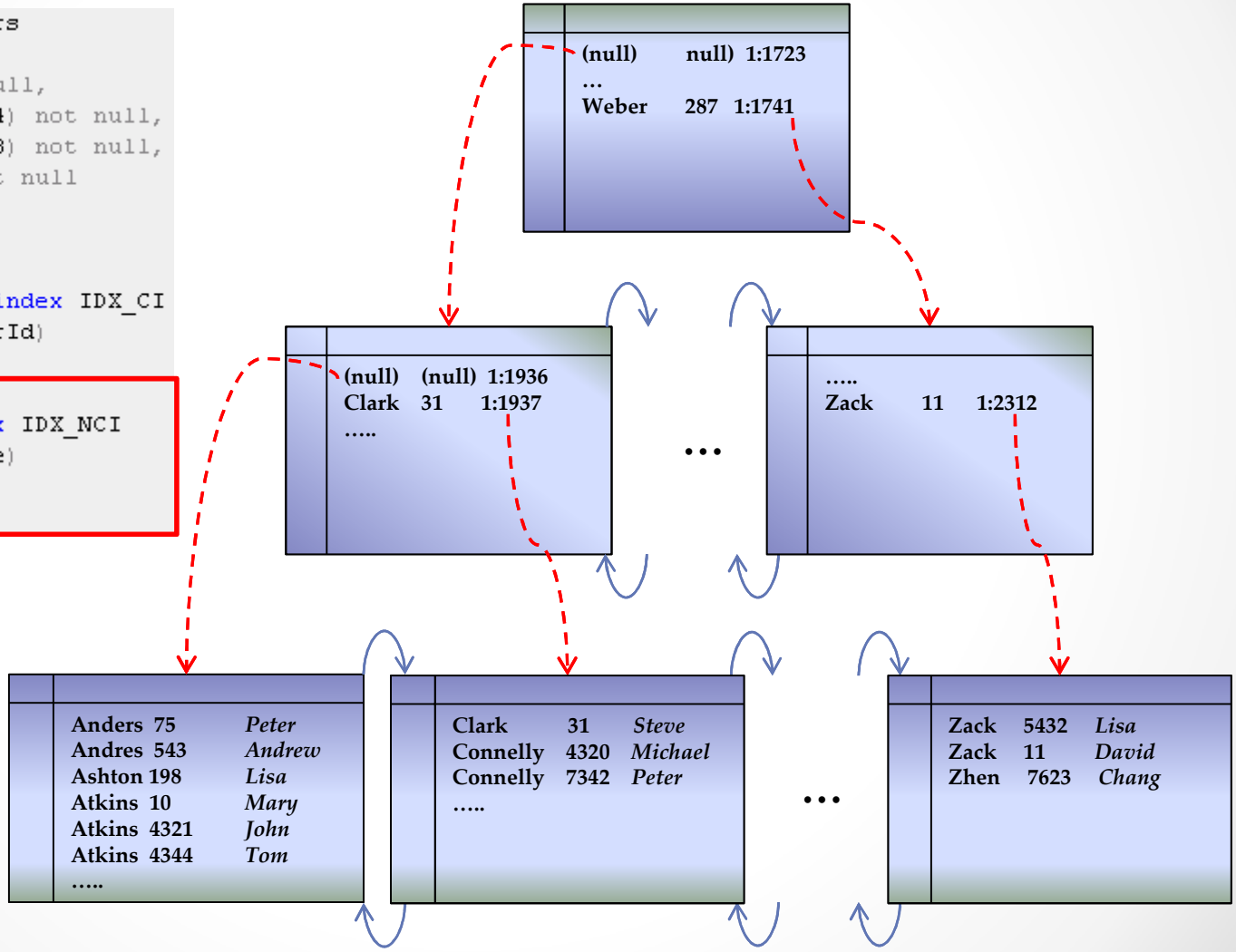
- Не используйте HEAP таблицы
- Определяйте индексы уникальными когда возможно
- Key lookup очень дорогая операция
  - SQL Server не использует некластерные индексы, в случае если он предполагает, что требуется большое количество key lookup операций
- SQL Server использует статистику для оценки числа найденных записей
  - Гистограмма хранится только для первого (левого) столбка индексов и содержит не более 200 шагов (значений ключа)
  - Статистика неаккуратна при большом количестве данных
  - Автоматическое обновление статистики требует изменения ~20% записей (значений первого (левого) поля) в таблице
    - TF 2371 (SQL Server 2008 R2 SP1+) изменяет алгоритм

Индексы с включенными столбцами

```
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
```



Индексы с включенными столбцами

Демонстрация

Индексы с включенными столбцами

- Включенные столбцы:
  - Хранятся только на уровне листьев
  - Неотсортированы
  - Не включаются в 900 байт (максимальный размер ключа NCI)

- **Помогают избавится от операции Key Lookup**
- Помогают консолидировать индексы

- Недостатки
  - Больший размер записи на уровне листьев
  - Дополнительная нагрузка при изменении данных
  - Дополнительная нагрузка при поддержке индексов

Фильтрованные индексы

```
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
        constraint DEF_Pos_Processed
        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)
include(Processed)
where Processed = 0
go

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

- Индексация части значений
  - Меньший размер индекса и стоимость поддержки
- Потенциальные проблемы
  - Ограничения в фильтрах
  - Оптимизатор консервативен в использовании фильтрованных индексов
    - **Обязательно включайте столбцы из фильтра в индекс (ключ или «включенный столбец»)**
- Оптимальное использование:
  - Индексация для части значений ключа
  - Sparse столбцы
  - Уникальность части (not null) значений

Фильтрованные индексы

Демонстрация

Секционированные таблицы

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

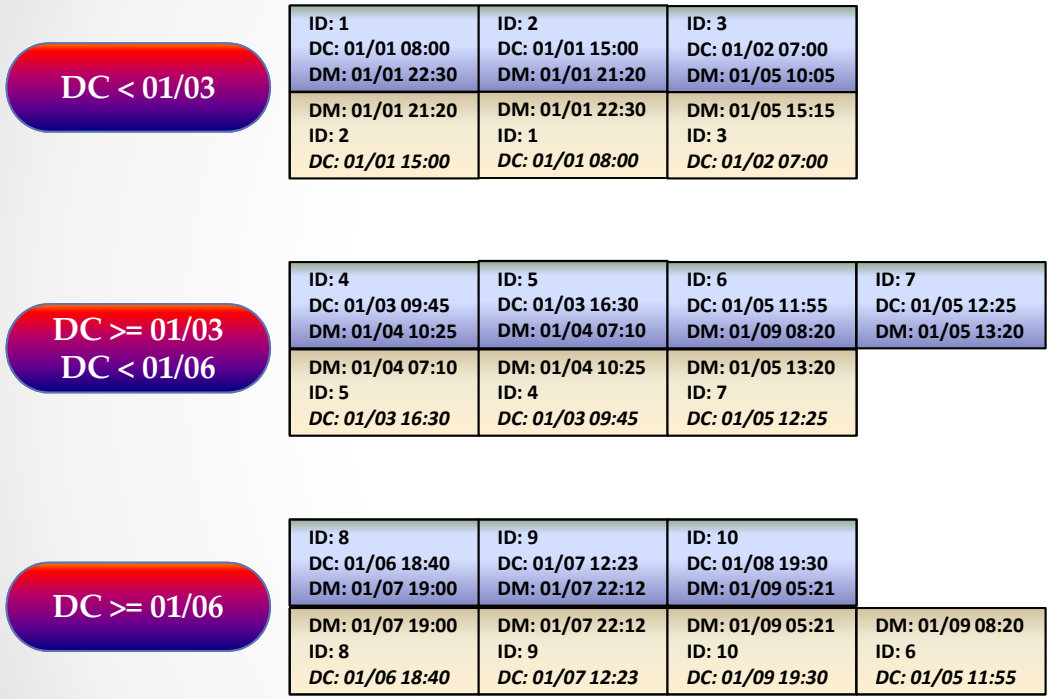
```
create unique clustered index IDX_CI
on dbo.Data(ID)
```

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: 01/02 07:00 DM: 01/05 15:15	ID: 4 DC: 01/03 09:45 DM: 01/04 10:25	ID: 5 DC: 01/03 16:30 DM: 01/04 07:10	ID: 6 DC: 01/05 11:55 DM: 01/09 08:20	ID: 7 DC: 01/05 12:25 DM: 01/05 13:20	ID: 8 DC: 01/06 18:40 DM: 01/07 19:00	ID: 9 DC: 01/07 12:23 DM: 01/07 22:12	ID: 10 DC: 01/08 19:30 DM: 01/09 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: 01/04 07:10 ID: 5 *DC: 01/03 16:30*	DM: 01/04 10:25 ID: 4 *DC: 01/03 09:45*	DM: 01/05 13:20 ID: 7 *DC: 01/05 12:25*	DM: 01/05 15:15 ID: 3 *DC: 01/02 07:00*	DM: 01/07 19:00 ID: 8 *DC: 01/06 18:40*	DM: 01/07 22:12 ID: 9 *DC: 01/07 12:23*	DM: 01/09 05:21 ID: 10 *DC: 01/08 19:30*	DM: 01/09 08:20 ID: 6 *DC: 01/05 11:55*

Секционированные таблицы




```
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

```
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: 01/02 07:00 DM: 01/05 15:15	ID: 4 DC: 01/03 09:45 DM: 01/04 10:25	ID: 5 DC: 01/03 16:30 DM: 01/04 07:10	ID: 6 DC: 01/05 11:55 DM: 01/09 08:20	ID: 7 DC: 01/05 12:25 DM: 01/05 13:20	ID: 8 DC: 01/06 18:40 DM: 01/07 19:00	ID: 9 DC: 01/07 12:23 DM: 01/07 22:12	ID: 10 DC: 01/08 19:30 DM: 01/09 05:21

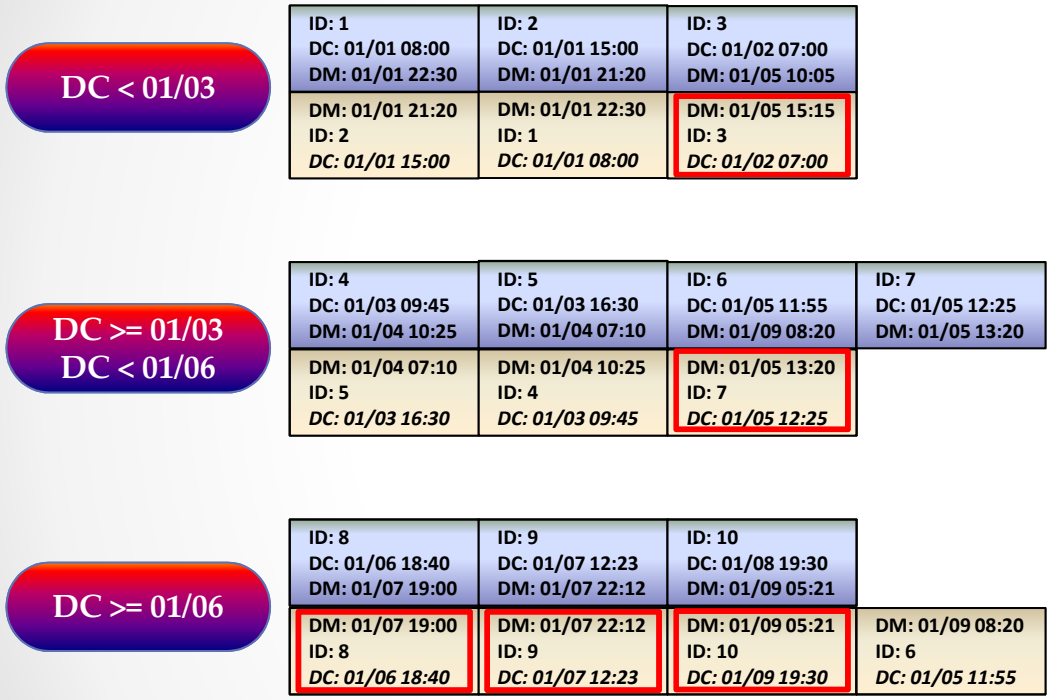
```
create nonclustered index IDX_NCI
 on dbo.Data(DateModified)
include(DateCreated)
```

```
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: 01/04 07:10 ID: 5 *DC: 01/03 16:30*	DM: 01/04 10:25 ID: 4 *DC: 01/03 09:45*	DM: 01/05 13:20 ID: 7 *DC: 01/05 12:25*	DM: 01/05 15:15 ID: 3 *DC: 01/02 07:00*	DM: 01/07 19:00 ID: 8 *DC: 01/06 18:40*	DM: 01/07 22:12 ID: 9 *DC: 01/07 12:23*	DM: 01/09 05:21 ID: 10 *DC: 01/08 19:30*	DM: 01/09 08:20 ID: 6 *DC: 01/05 11:55*

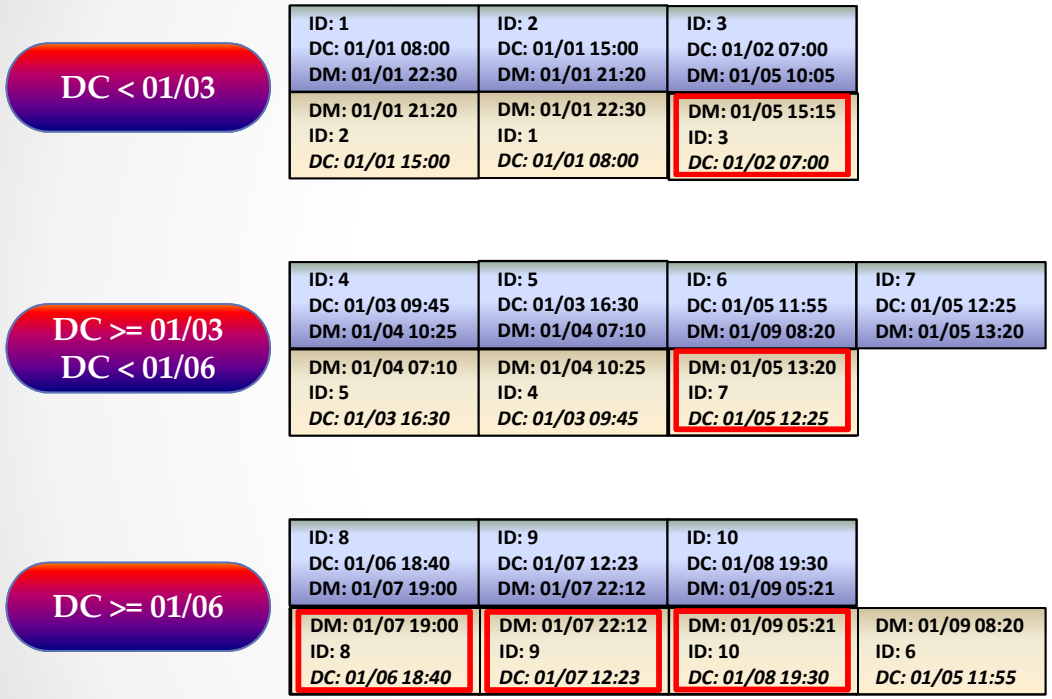
Секционированные таблицы

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



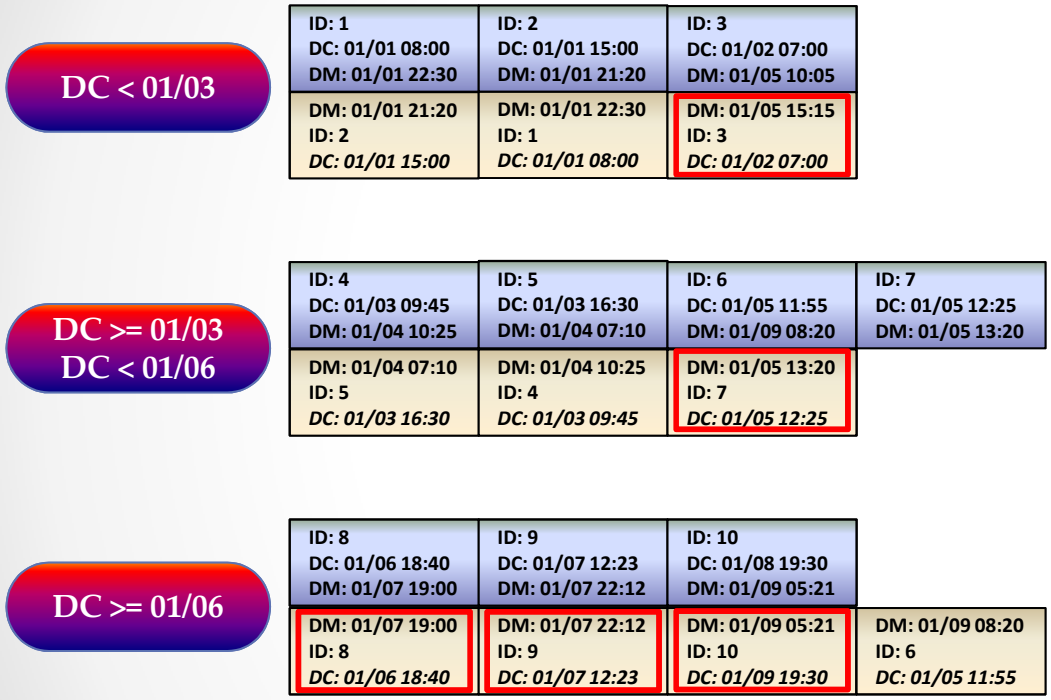
Секционированные таблицы

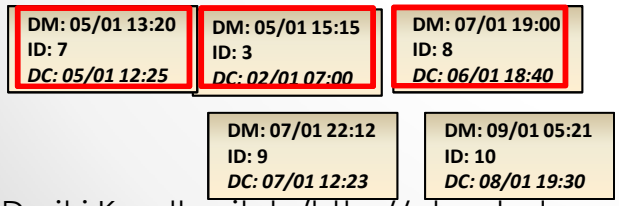
```
select top 3 *
from dbo.Data
where DateModified > '2012-01-05'
order by DateModified, ID
```

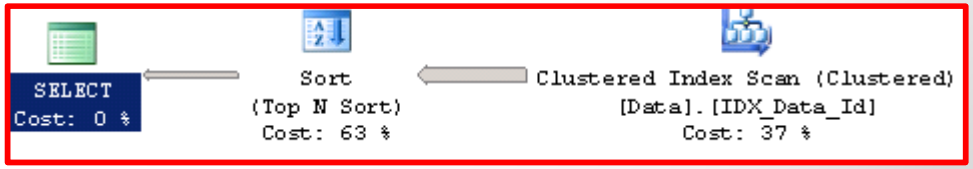


Секционированные таблицы

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







Оптимизация с использованием *$Partition* функции

Демонстрация

Ключевые моменты

- Используйте индексы с включенными столбцами для оптимизации (более подробно через месяц)
- Фильтрованные индексы: Всегда включайте столбцы из фильтра в индекс
- Используйте фильтрованную статистику в случае корреляции предикатов
- Будьте осторожны в случае секционированных индексов
  - http://aboutsqlserver.com/publications

Мы обсудили

- Структура индексов
- Как и когда SQL Server использует индексы
- Дополнительные возможности
  - Индексы с дополнительными полями
  - Фильтрованные индексы
  - Секционированные индексы

- Через месяц мы поговорим о:
  - Фрагментации и поддержке индексов
  - Стратегиях индексации
  - Стратегиях оптимизации

Вопросы?

- Огромное спасибо за внимание!

- Email: dmitri@aboutsqlserver.com
- Blog: http://aboutsqlserver.com