Оптимизация запросов в реляционных БД

Николай Александрович Шестаков

Томский Политехнический Университет/Rubius

Физическая организация данных

- Файлы данных
  - Можно распределять по разным дискам (в зависимости от сценария работы)
- Файл журнала транзакций
  - Последовательная (sequential) запись
- Файлы резервных копий
  - Full backup
  - Incremental backup
  - Log backup (для point-in-time restore)

Физическая организация данных в SQL Server

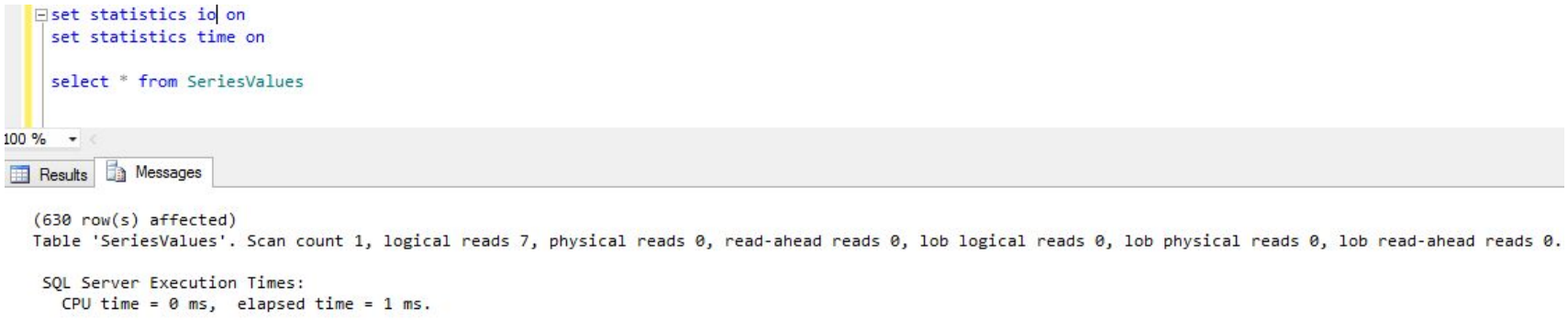
- БД SQL Server хранится в одном или нескольких файлах (.mdf, .ndf)
- Кроме файлов данных есть файл журнала транзакций (.ldf)
- Файл данных разбит на страницы по 8К
- Страница – минимальная единица чтения/записи данных
- Страницы сгруппированы в 64К экстенты
- Постраничная (поблочная) организация данных характерна для большинства реляционных СУБД

Организация данных на странице

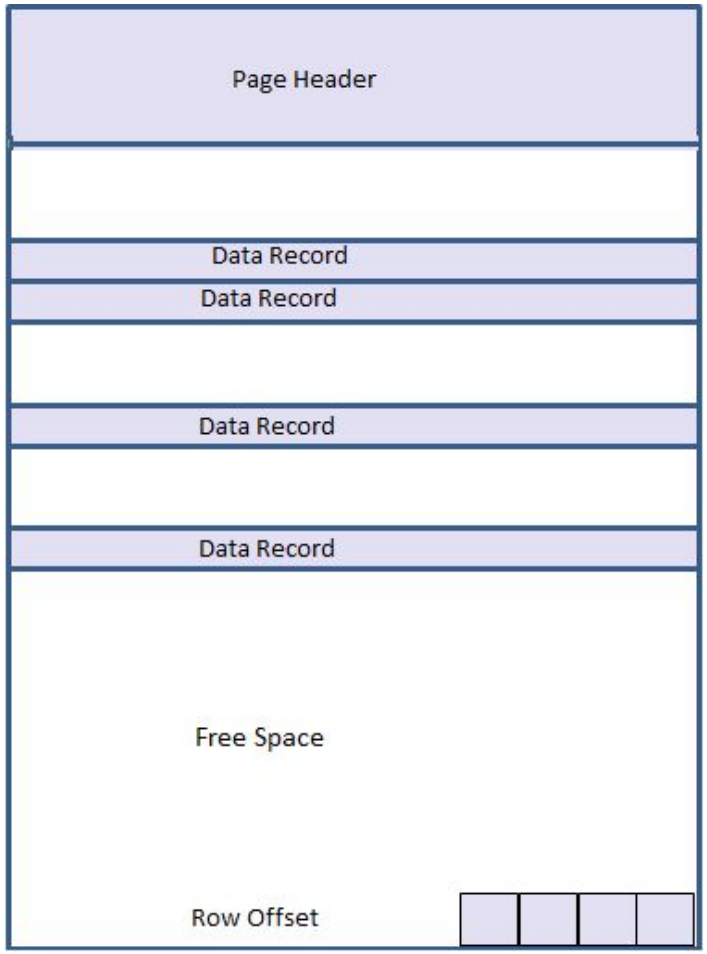
- Страница с данными хранит записи таблицы
- Одна запись хранится на странице целиком (кроме LOB и var-типов, которые не влезли на страницу)
- Таким образом, при чтении всего одного атрибута, с диска считывается запись целиком (не считая LOB)
- Постраничная организация данных характерна для большинства реляционных СУБД

Дисковые операции

- Количество чтений/записей – это количество дисковых страниц
- Операции могут быть логические и физические



Организация данных на странице



- Размер страницы – 8К
- Страница хранит записи только одной таблицы
- Все атрибуты записи, кроме LOB-полей, хранятся на одной странице (исключение – длинные значения типов переменной длины)
- Таким образом, при чтении всего-лишь одного атрибута, с диска считывается запись целиком
- Можно посмотреть дамп страницы DBCC PAGE

Дисковые операции

```
create table demo1
(
  a int,
  b int
)

create table demo2
(
  a int,
  b int,
  c char(100)
)
```

```
declare @n int

set @n = 0

while @n < 1000
begin
  insert into demo1(a, b) values (@n, 10*@n)
  insert into demo2(a, b) values (@n, 10*@n)
  set @n = @n + 1
end
```

```
set statistics io on
select a from demo1 where b = 500
select a from demo2 where b = 500
```

Results | Messages

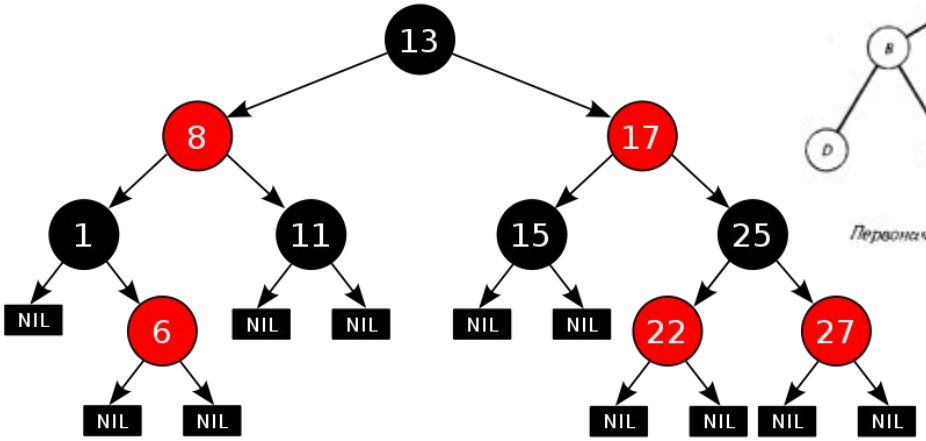
```
(1 row(s) affected)
Table 'demo1'. Scan count 1, logical reads 3, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

(1 row(s) affected)
Table 'demo2'. Scan count 1, logical reads 16, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
```

Логарифмический поиск

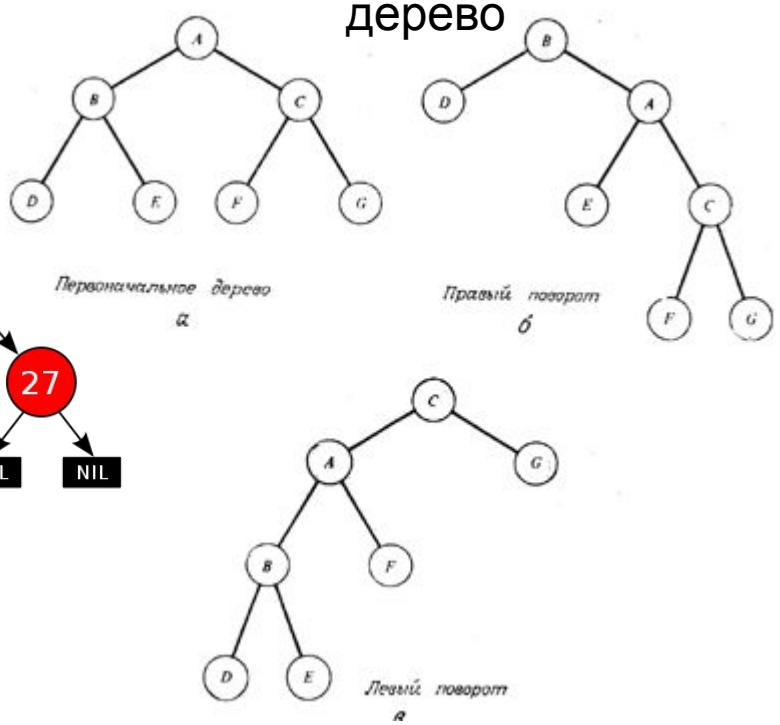
- Чтобы найти данные в таблице по условию, необходимо выполнить сканирование таблицы – это *O(N)*
- Если данные отсортированы, то это можно сделать поиском за *O(log(N))*

Сбалансированные деревья

Красно-чёрное дерево



AVL-дерево



Не подходят для хранения во внешней памяти!

B-Tree

- Оптимизировано под страничную организацию данных во внешней памяти (один узел – одна страница)
- Сбалансированное (длина пути от корня до любого листа одинакова для всех листов)
- Логарифмический поиск (по количеству дисковых чтений)
- Логарифмическая запись
- Сильная ветвистость (сотни потомков у узла)

B-Tree

Id	A	B
23	Cat	A
85	Dog	A
67	Frog	B
37	Hen	C
94	Cow	D
17	Pig	D
49	Wolf	E
44	Rat	B
72	Duck	C
82	Ant	F
80	Bat	B

B-Tree

Для примера допустим: на одну страницу помещается 3 записи данных или 4 ключа индекса
(на самом деле – намного больше)

Id	A	B
23	Cat	A
85	Dog	A
67	Frog	B
37	Hen	C
94	Cow	D
17	Pig	D
49	Wolf	E
44	Rat	B
72	Duck	C
82	Ant	F
80	Bat	B

Id	A	B
23	Cat	A

B-Tree

Id	A	B
23	Cat	A
85	Dog	A
67	Frog	B
37	Hen	C
94	Cow	D
17	Pig	D
49	Wolf	E
44	Rat	B
72	Duck	C
82	Ant	F
80	Bat	B

Id	A	B
23	Cat	A
85	Dog	A

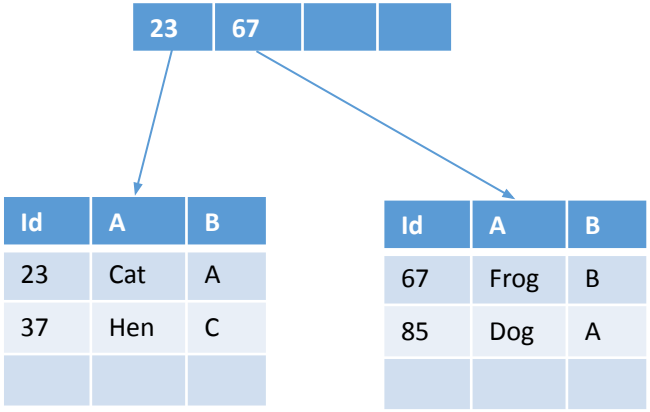
B-Tree

Id	A	B
23	Cat	A
85	Dog	A
67	Frog	B
37	Hen	C
94	Cow	D
17	Pig	D
49	Wolf	E
44	Rat	B
72	Duck	C
82	Ant	F
80	Bat	B

Id	A	B
23	Cat	A
67	Frog	B
85	Dog	A

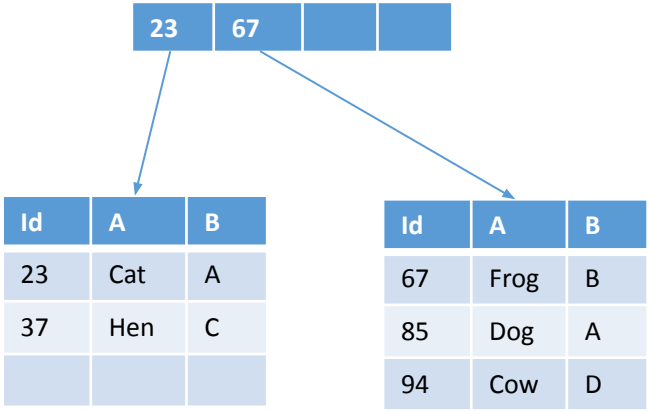
B-Tree

Id	A	B
23	Cat	A
85	Dog	A
67	Frog	B
37	Hen	C
94	Cow	D
17	Pig	D
49	Wolf	E
44	Rat	B
72	Duck	C
82	Ant	F
80	Bat	B



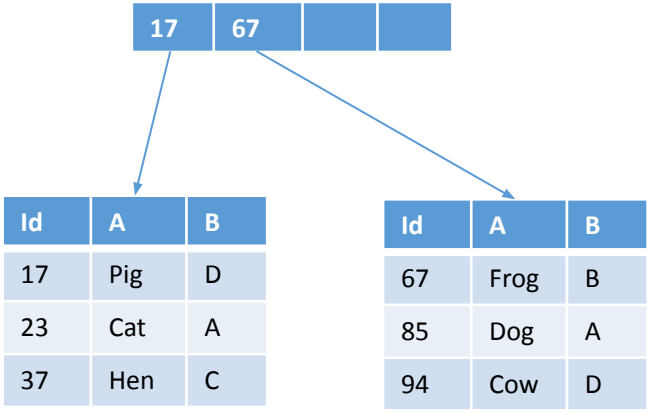
B-Tree

Id	A	B
23	Cat	A
85	Dog	A
67	Frog	B
37	Hen	C
94	Cow	D
17	Pig	D
49	Wolf	E
44	Rat	B
72	Duck	C
82	Ant	F
80	Bat	B



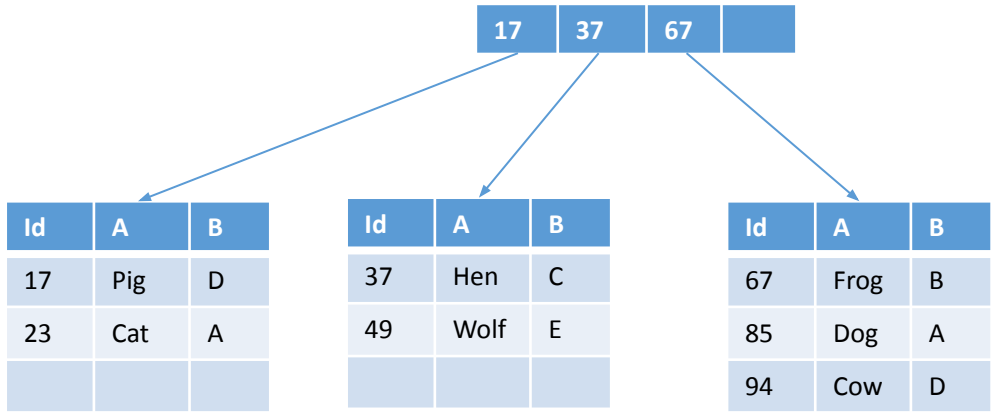
B-Tree

Id	A	B
23	Cat	A
85	Dog	A
67	Frog	B
37	Hen	C
94	Cow	D
17	Pig	D
49	Wolf	E
44	Rat	B
72	Duck	C
82	Ant	F
80	Bat	B



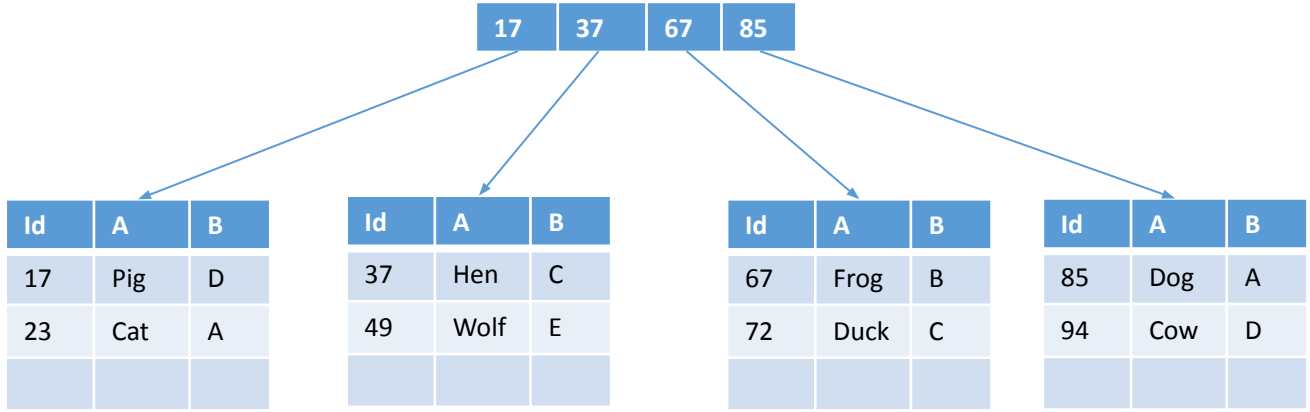
B-Tree

Id	A	B
23	Cat	A
85	Dog	A
67	Frog	B
37	Hen	C
94	Cow	D
17	Pig	D
49	Wolf	E
44	Rat	B
72	Duck	C
82	Ant	F
80	Bat	B



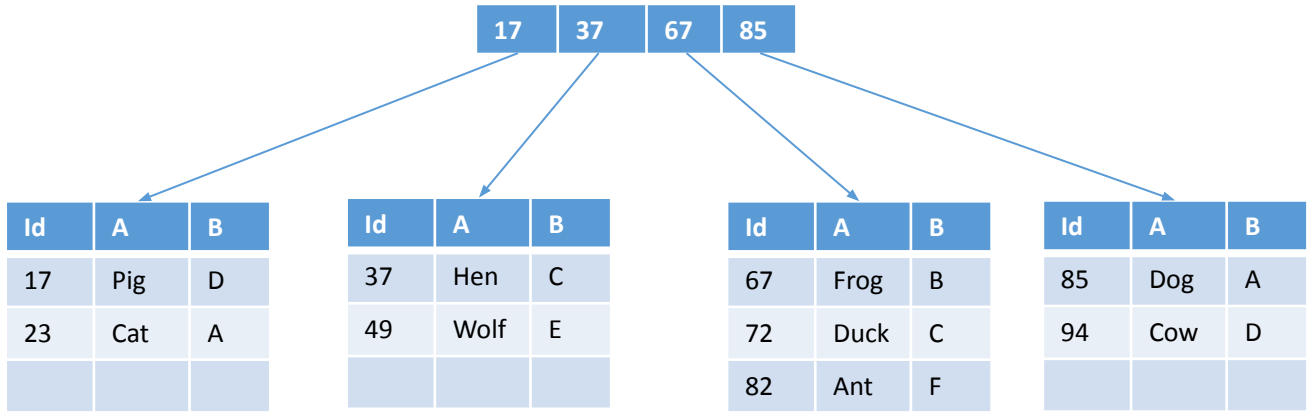
B-Tree

Id	A	B
23	Cat	A
85	Dog	A
67	Frog	B
37	Hen	C
94	Cow	D
17	Pig	D
49	Wolf	E
44	Rat	B
72	Duck	C
82	Ant	F
80	Bat	B



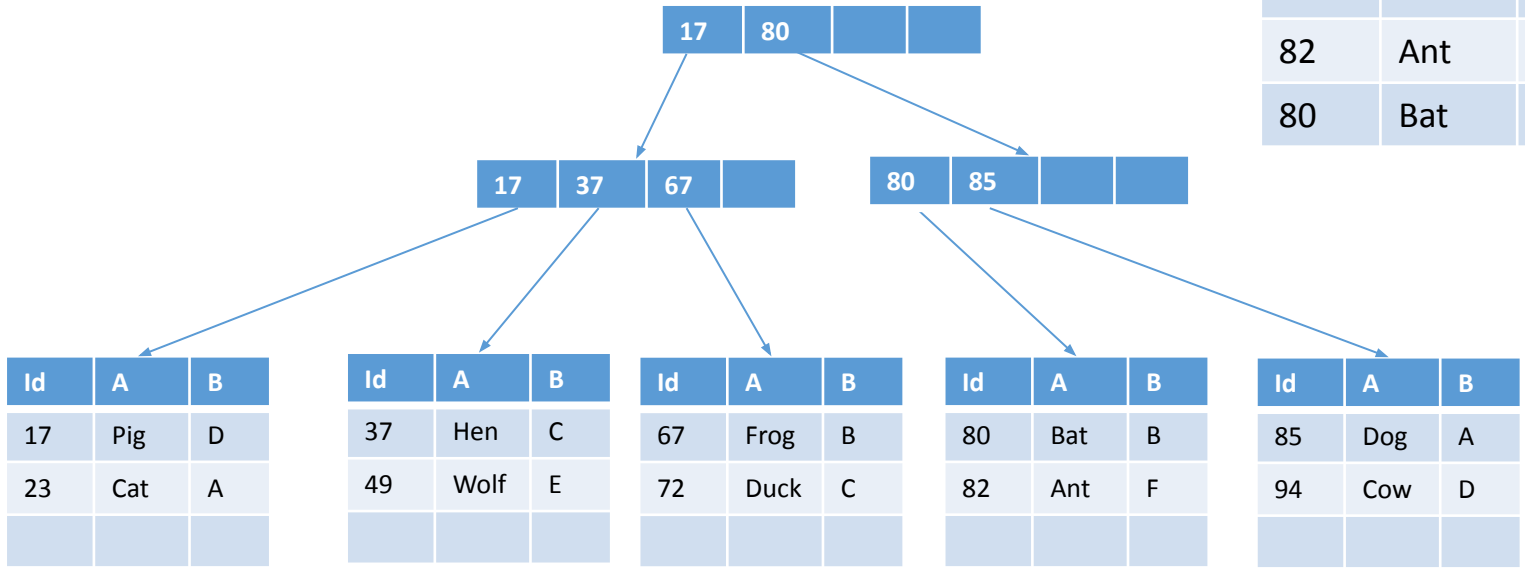
B-Tree

Id	A	B
23	Cat	A
85	Dog	A
67	Frog	B
37	Hen	C
94	Cow	D
17	Pig	D
49	Wolf	E
44	Rat	B
72	Duck	C
82	Ant	F
80	Bat	B



B-Tree

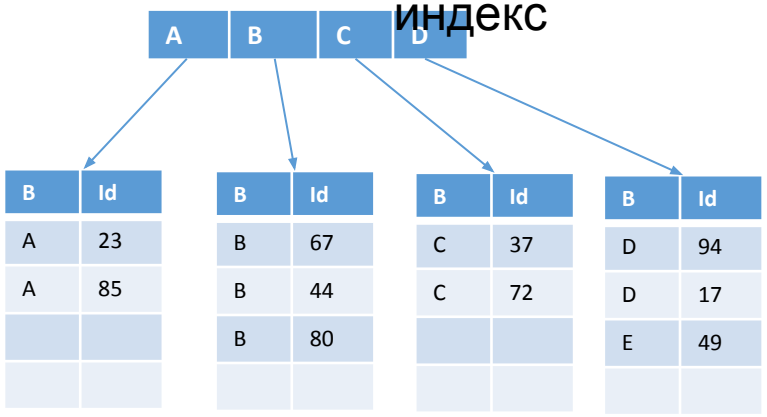
Id	A	B
23	Cat	A
85	Dog	A
67	Frog	B
37	Hen	C
94	Cow	D
17	Pig	D
49	Wolf	E
44	Rat	B
72	Duck	C
82	Ant	F
80	Bat	B



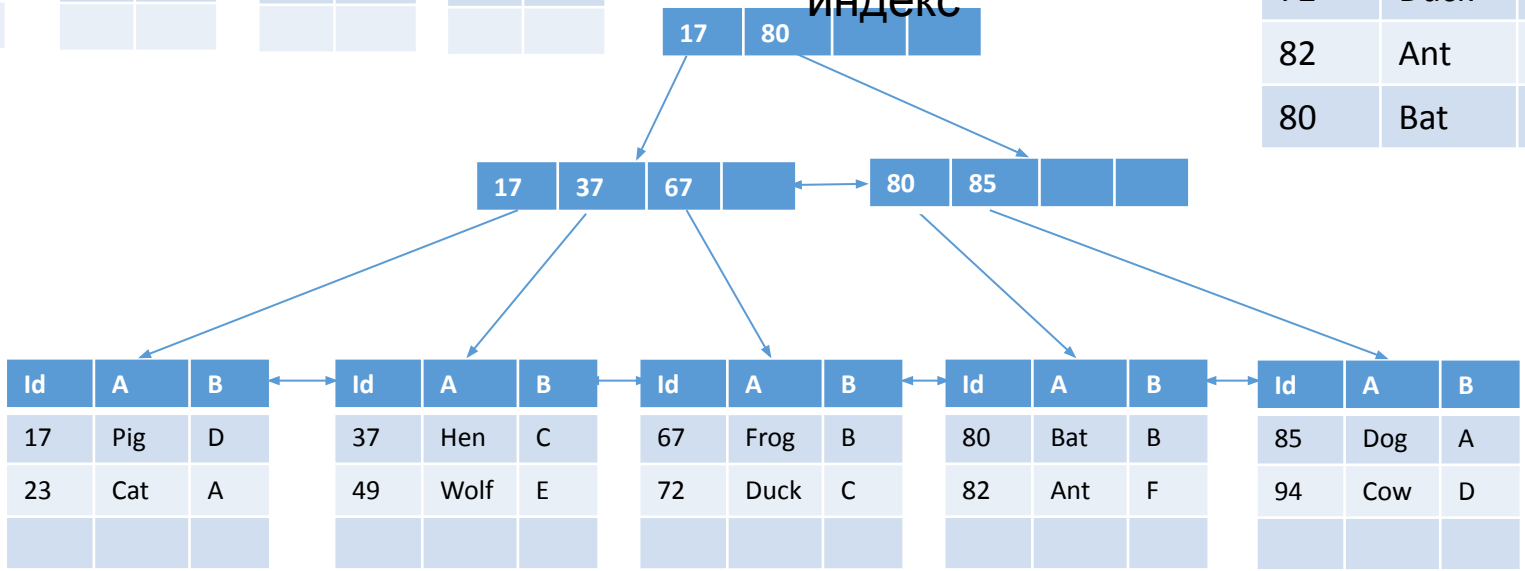
B-Tree

Id	A	B
23	Cat	A
85	Dog	A
67	Frog	B
37	Hen	C
94	Cow	D
17	Pig	D
49	Wolf	E
44	Rat	B
72	Duck	C
82	Ant	F
80	Bat	B

Некластерный индекс



Кластерный индекс



B-Tree

- Кластерный индекс
  - Листовые узлы – страницы с данными
  - Может быть только один для таблицы
- Некластерный индекс
  - Листовые узлы – страницы с ключами, плюс ссылка на запись с данными
  - Ссылка:
    - значение ключа кластерного индекса (если он есть)
    - адрес страницы с данными + идентификатор внутри страницы (если кластерный индекс отсутствует)

Когда использовать?

СЕЛЕКТИВНОСТЬ!

СЕЛЕКТИВНОСТЬ!

B*-Tree

- Индексы повышают эффективность
  - Операции поиска записей с хорошей селективностью
  - Поддержка уникальности значений атрибутов
  - Операции, требующие упорядочивания по ключу (JOIN, DISTINCT)
  - Проекция по небольшому количеству атрибутов

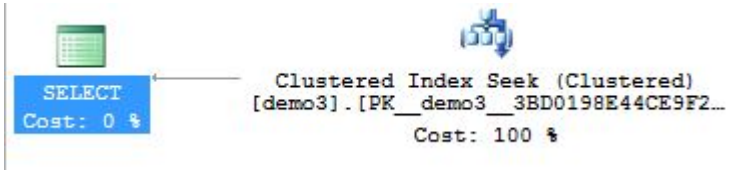
B-Tree

- Кластерный индекс лучше выбирать для атрибутов:
  - Короткое значение ключа (т.к. ключи к.и. используются как ссылки в некластерных)
  - Данные часто выбираются диапазонами значений ключа (т.к. по ключу к.и. сгруппированы записи в страницах данных)
  - Чаще используются для поиска (т.к. нет дополнительного обращения к страницам данных, к.и. быстрее)
- По умолчанию индекс первичного ключа делается кластерным, но это не всегда оптимальный выбор

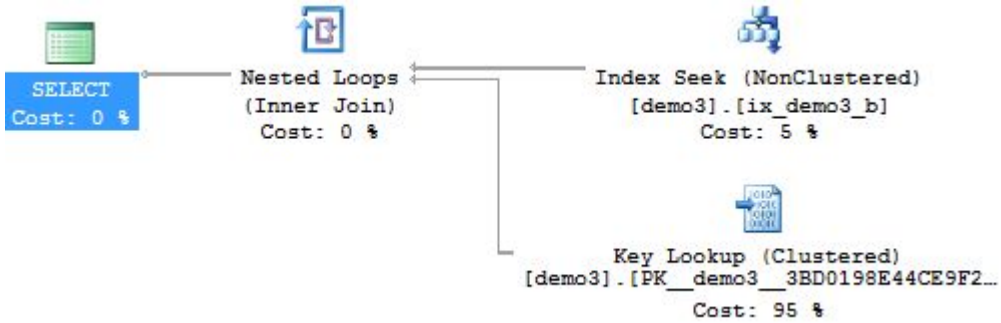
Пример

```
create table demo3
(
  a int primary key,
  b int,
  c char(100)
)
create nonclustered index ix_demo3_b on demo3 (b)

declare @n int

set @n = 0

while @n < 10000
begin
  insert into demo3(a, b) values (@n, @n % 500)
  set @n = @n + 1
end
```

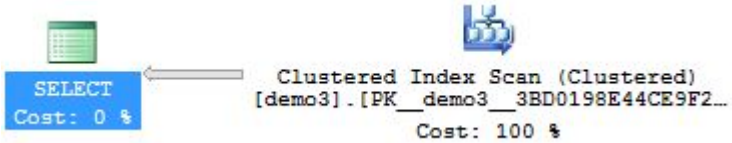
```
select * from demo3
where a = 1234
```




```
select * from demo3
where b = 123
```




```
select * from demo3
where b > 123
```



B-Tree

Why “B”?

The origin of "B-tree" has never been explained by the authors. ... "balanced," "broad," or "bushy" might apply. Others suggest that the "B" stands for Boeing. [Bayer and McCreight were at Boeing Scientific Research Labs in 1972.] Because of his contributions, however, it seems appropriate to think of B-trees as "Bayer"-trees.

- **Douglas Comer**, *The Ubiquitous B-Tree*, Computing Surveys, 11(2):123, June 1979.

B*-Tree

- Included columns
  - Можно построить индекс по нескольким атрибутам,
  - а можно включить атрибуты посредством INCLUDE
  - (и это не одно и то же)
  - INCLUDED атрибуты не входят в состав ключа, но сохраняются в листьях
  - Если запрос требует только атрибуты, входящие в индекс, либо в INCLUDED атрибуты индекса, то обращаться к странице с данными не придётся – нужные данные уже есть в индексе

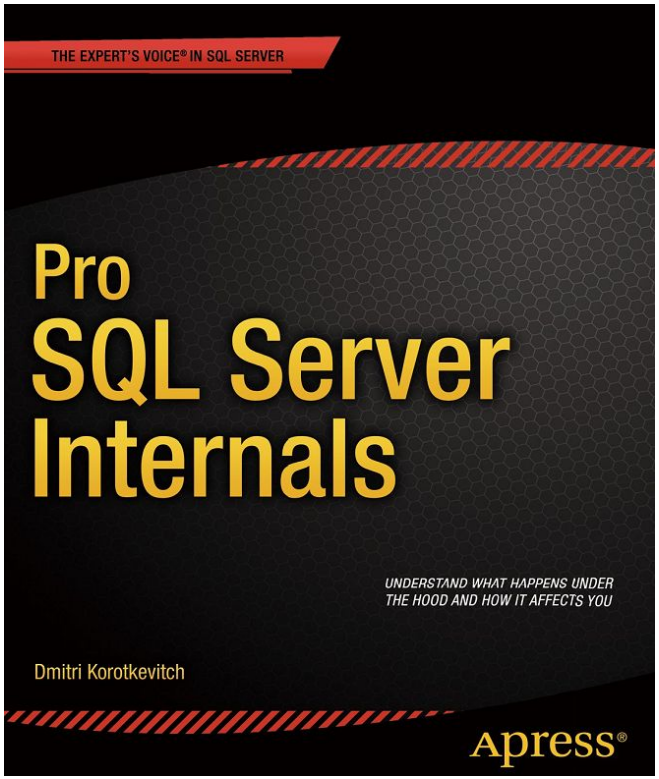
B*-Tree

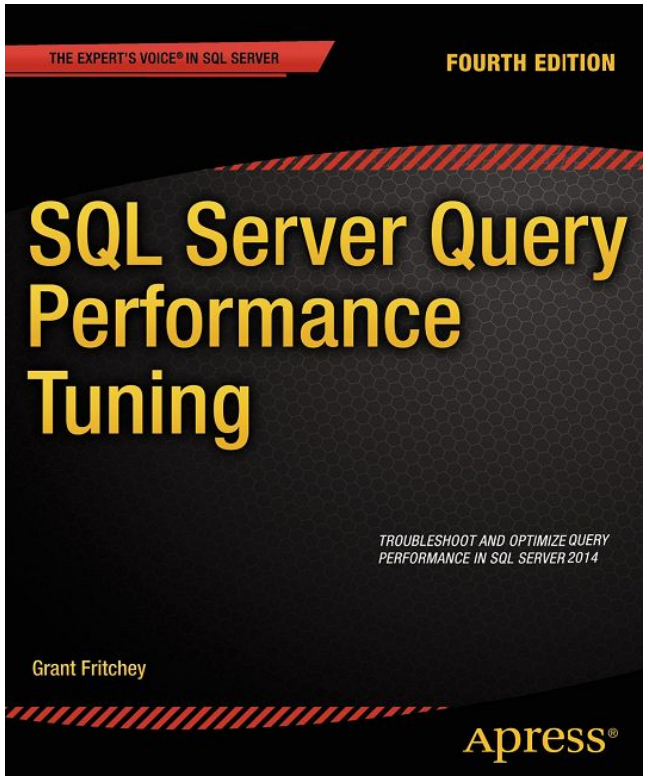
- FILLFACTOR
  - Задаёт объём занятого пространства на листовых страницах, которое выделяет СУБД при создании/перестройке индекса
  - По умолчанию – 100%
  - Необходим для сокращения времени вставок/обновлений
  - Несколько снижает скорость чтения

B*-Tree

- Фрагментация данных
  - External fragmentation
  - Internal fragmentation
- Способы дефрагментации:
  - INDEX REORGANIZE
  - INDEX REBUILD
  - CREATE WITH DROP EXISTING
- Фрагментация – последнее, о чём стоит беспокоиться

Что почитать





http://www.brentozar.com/
https://www.youtube.com/user/BrentOzar
…

nikolay.shestakov@rubius.com

Оптимизация запросов в реляционных БД

Часть II

Николай Александрович Шестаков

Томский Политехнический Университет

Rubius

Оптимизация и выполнение запроса

- Алгоритмы выполнения соединений
- Этапы жизненного цикла запроса
- План выполнения и выполнение запроса
- Кеширование планов выполнения

Join algorithms

Nested loop join

- Inner join
  - for each row R1 in outer table
    for each row R2 in inner table //or index lookup!
      if R1 joins with R2
        return join (R1, R2)
- Outer join
  - for each row R1 in outer table
    for each row R2 in inner table //or index lookup!
      if R1 joins with R2
        return join (R1, R2)
      else
        return join (R1, NULL)

Join algorithms

Merge join

```
/* Prerequirements: Inputs I1 and I2 are sorted */
get first row R1 from input I1
get first row R2 from input I2
while not end of either input
begin
        if R1 joins with R2
        begin
                return join (R1, R2)
                get next row R2 from I2
        end
        else if R1 < R2
                get next row R1 from I1
        else /* R1 > R2 */
                get next row R2 from I2
end
```

Join algorithms

Hash join

```
/* Build Phase */
for each row R1 in input I1
begin
        calculate hash value on R1 join key
        insert hash value to appropriate bucket in hash table
end
/* Probe Phase */
for each row R2 in input I2
begin
        calculate hash value on R2 join key
        for each row R1 in hash table bucket
                if R1 joins with R2
                        return join (R1, R2)
end
```

Join algorithms

	Nested Loop Join	Merge Join	Hash Join
Best use-case	Small inputs. Preferable with index on join key in inner table.	Medium to large inputs sorted on index key.	Medium to large inputs.
Requires sorted input	No	Yes	No
Requires equality predicate	No	Yes	Yes
Blocking operator	No	No	Yes (Build phase only)
Uses memory	No	No	Yes
Uses tempdb	No	No (with exception of many-to-many joins)	Yes in case of spills
Preserves order	Yes (outer input)	Yes	No

Query optimization and execution

- Query Life Cycle

Step	Description	Result
Parse	Syntax validation and initial transformation	Logical query tree
Bind	Binding to objects. Loading metadata properties	Bound tree
Optimize	Execution plan generation	Execution plan
Execute	Query execution	Result

Query optimization and execution

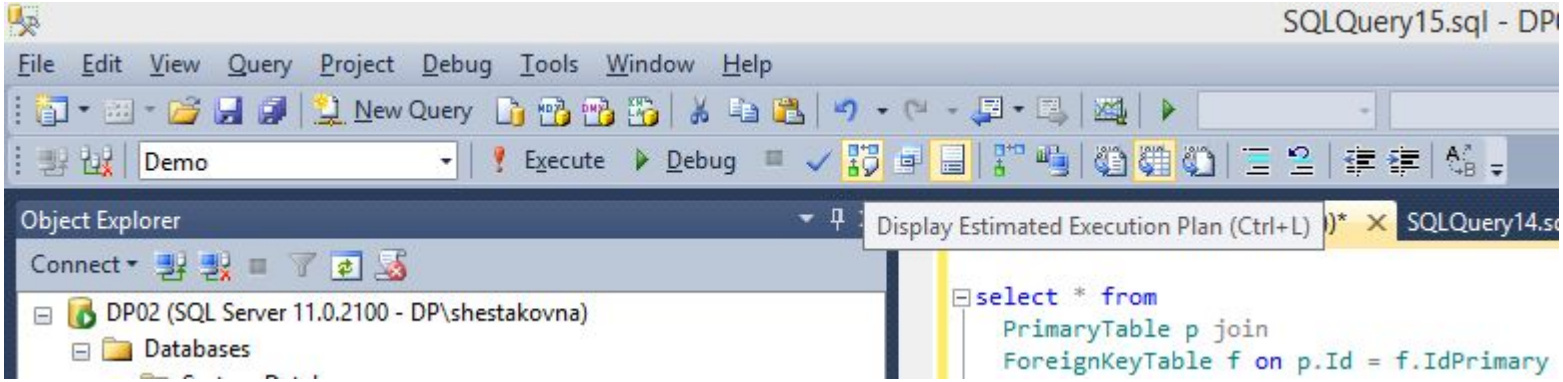
- Optimization
  - Goal – to find a **good enough** execution plan, **quickly enough**
- Phases
  - Simplification
  - Trivial plan search
  - Statistics update
  - Cost-based optimization (several stages here)
  - Execution plan generation

Query optimization and execution

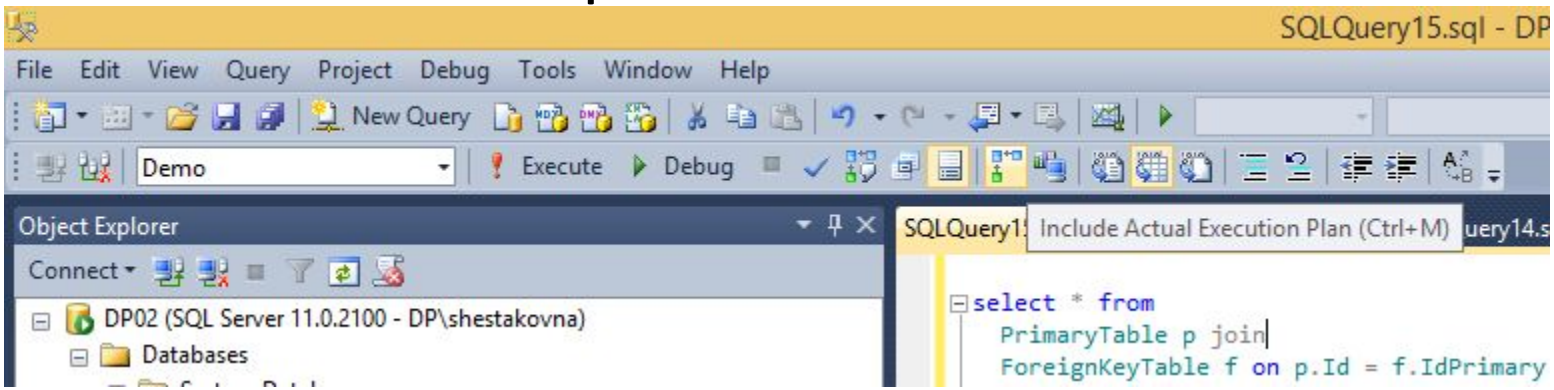
- Посмотреть план выполнения:
- Можно в Management Studio, включив опцию “Show execution plan”
- В текстовом/табличном/xml виде, предварительно выполнив запрос SET SHOWPLAN_TEXT ON или SET SHOWPLAN_ALL ON или SET SHOWPLAN_XML ON
- Использовать системную функцию sys.dm_exec_query_stats. В этом случае не нужно отдельно запускать запрос, план будет показан из кеша

План выполнения

- Estimated execution plan



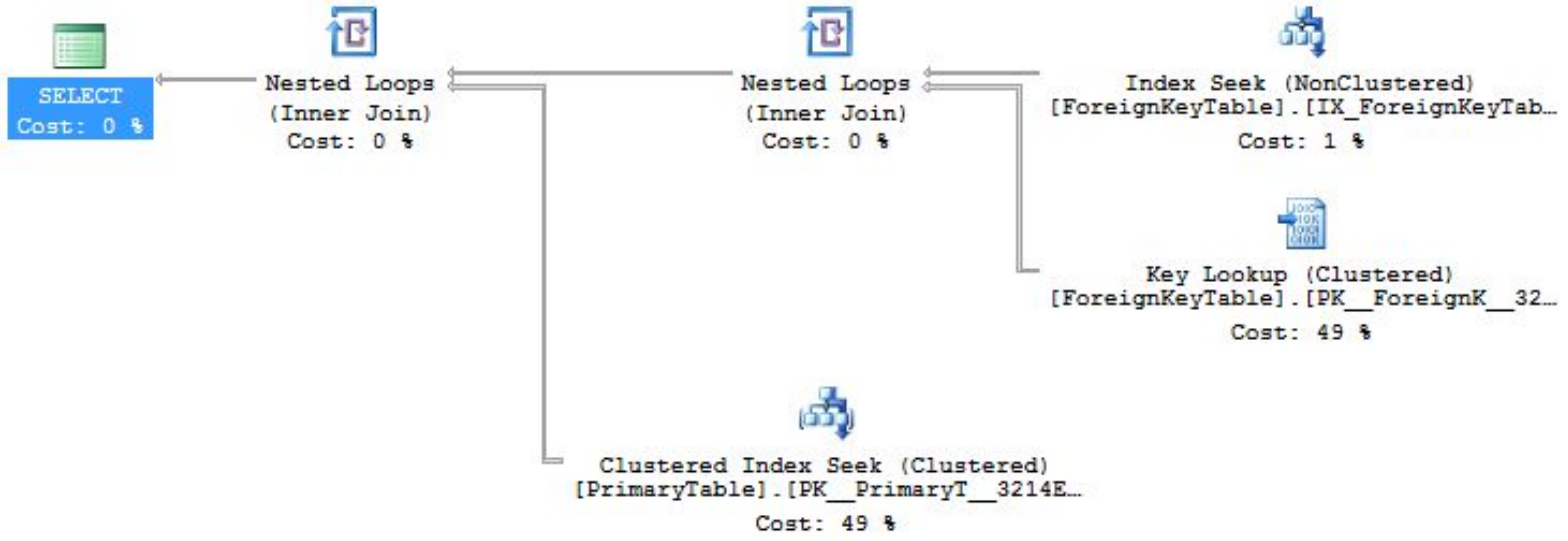
- Actual execution plan



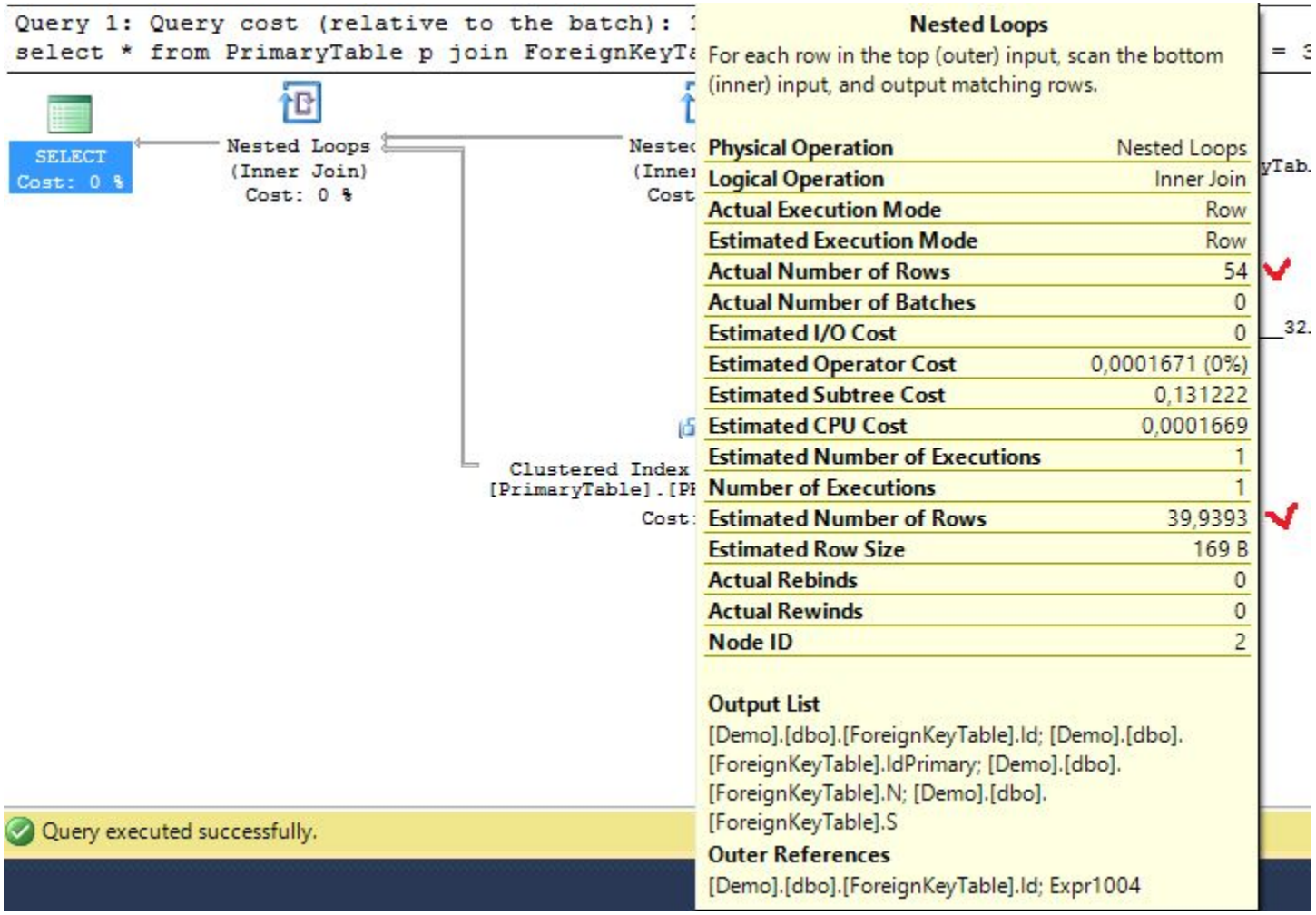
План выполнения

- Actual execution plan показывает не только оцениваемые показатели, но и реальные

```
select * from
  PrimaryTable p join
  ForeignKeyTable f on p.Id = f.IdPrimary
where
  f.N = 3456  -- change selectivity here
```



План выполнения



План выполнения

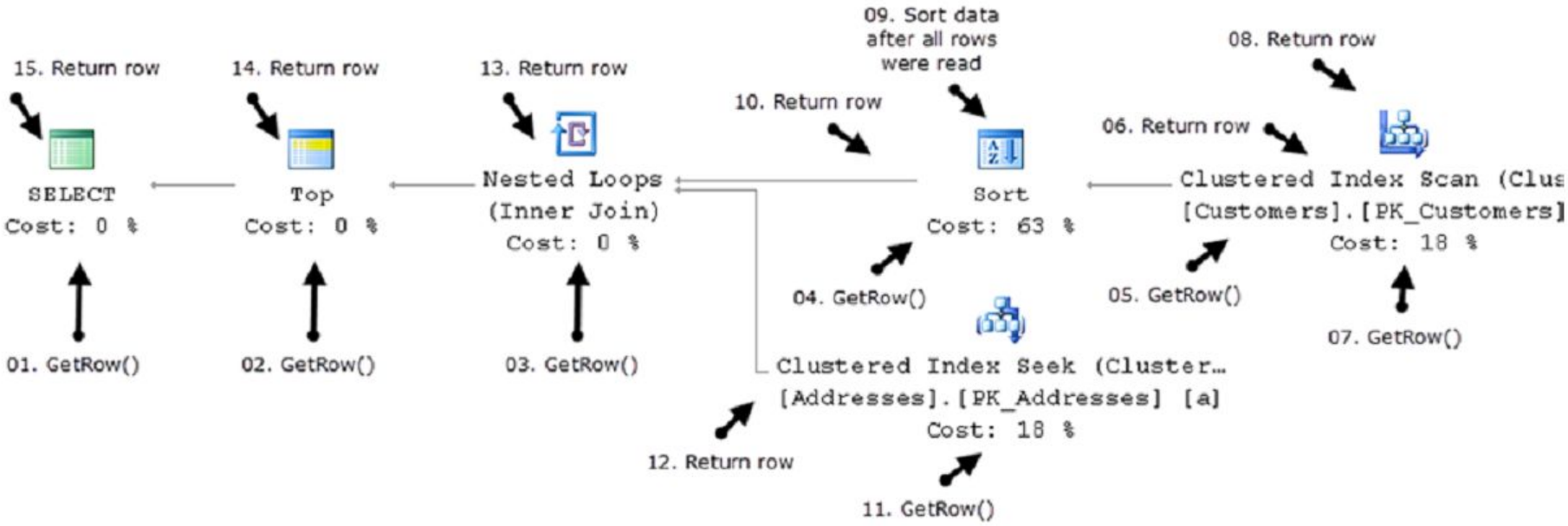
Планы можно выводить в виде:

- Графический
- XML
- Табличный (через SET STATISTICS PROFILE ON)
- Текстовый

Query optimization and execution

- Query execution

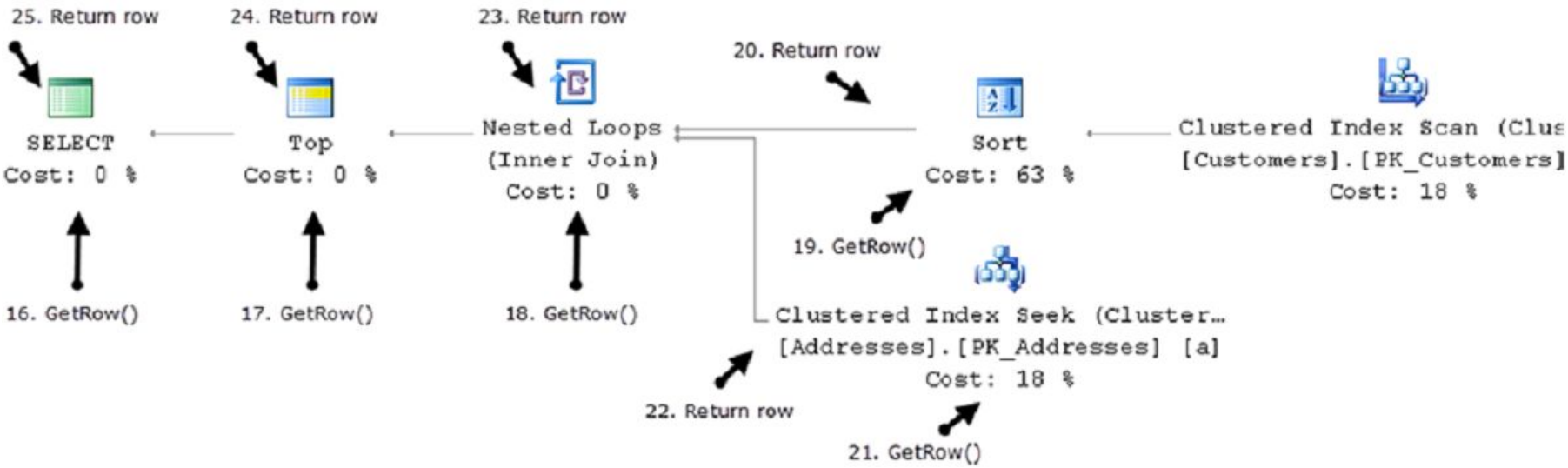
```
select top 10 c.CustomerId, c.Name, a.Street, a.City, a.State, a.ZipCode
from
  Customers c join
  Addresses a on c.PrimaryAddressId = a.AddressId
order by c.Name
```



Query optimization and execution

- Query execution

```
select top 10 c.CustomerId, c.Name, a.Street, a.City, a.State, a.ZipCode
from
  Customers c join
  Addresses a on c.PrimaryAddressId = a.AddressId
order by c.Name
```



Статистики

```
/*
drop table ForeignKeyTable
drop table PrimaryTable
*/

create table PrimaryTable (Id int identity primary key, N int, S char(150))
go
create table ForeignKeyTable (Id int identity primary key, IdPrimary int references PrimaryTable, N int, S char(150))
go

insert into PrimaryTable (N, S) values (CAST(RAND() * 10000 as int), CAST(NEWID() as char(150)))
go 100000

insert into ForeignKeyTable (IdPrimary, N, S) values (CAST(RAND() * 100000 as int), CAST(RAND() * 1000 as int),
CAST(NEWID() as char(150)))
go 200000
insert into ForeignKeyTable (IdPrimary, N, S) values (CAST(RAND() * 100000 as int), CAST(RAND() * 1000 + 1000 as int),
CAST(NEWID() as char(150)))
go 100000
insert into ForeignKeyTable (IdPrimary, N, S) values (CAST(RAND() * 100000 as int), CAST(RAND() * 1000 + 2000 as int),
CAST(NEWID() as char(150)))
go 150000

create nonclustered index IX_ForeignKeyTable ON dbo.ForeignKeyTable(N)
go

select * from
  PrimaryTable p join
  ForeignKeyTable f on p.Id = f.IdPrimary
where
  f.N = 1234  -- change selectivity here and see how execution plan is changing

DBCC SHOW_STATISTICS ("dbo.ForeignKeyTable", "IX_ForeignKeyTable")
```

Статистики

	RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
1	0	0	200	0	1
2	43	8492	176	42	202,1905
3	52	1632	230	8	204
4	61	1630	221	8	203,75
5	71	1820	225	9	202,2222
6	91	3847	217	19	202,4737
7	115	4548	232	23	197,7391
8	137	4095	210	21	195
9	143	939	169	5	187,8
10	157	2642	223	13	203,2308
11	177	3806	177	19	200,3158
12	195	3344	185	17	196,7059
13	204	1621	176	8	202,625
14	215	1948	205	10	194,8
15	225	1734	168	9	192,6667
16	234	1647	184	8	205,875
17	273	7629	214	38	200,7632
18	297	4580	179	23	199,1304
19	304	1276	180	6	212,6667
20	314	1868	224	9	207,5556
21	327	2420	187	12	201,6667
22	345	3439	211	17	202,2941
23	356	2003	176	10	200,3
24	366	1794	216	9	199,3333
25	378	2169	219	11	197,1818

Статистики

52	915	5080	216	25	203,2
53	939	4576	192	23	198,9565
54	957	3423	231	17	201,3529
55	995	7207	210	37	194,7838
56	1002	1005	99	6	167,5
57	1075	7162	105	72	99,47222
58	1313	23905	119	237	100,865
59	1488	17381	90	174	99,89081
60	1510	2025	103	21	96,42857
61	1521	1026	78	10	102,6
62	1747	22416	82	225	99,62666
63	1795	4634	121	47	98,59574
64	1850	5358	108	54	99,22222
65	1889	3976	72	38	104,6316
66	1951	6164	83	61	101,0492
67	1993	4046	82	41	98,68293
68	2016	2904	171	22	132
69	2053	5374	162	36	149,2778
70	2081	4005	160	27	148,3333
71	2109	3938	155	27	145,8519
72	2137	4162	143	27	154,1481
73	2179	6163	145	41	150,3171
74	2222	6221	155	42	148,119
75	2264	6209	165	41	151,439
76	2307	6333	135	42	150,7857
77	2366	8870	169	58	152,931
78	2415	7141	137	48	148,7708

Статистики

RANGE_HI_KEY	This is the top value of the step represented by this row within the histogram.
RANGE_ROWS	This number shows the number of rows within the step that are greater than the previous top value and the current top value, but not equal to either.
EQ_ROWS	This number shows the number of rows within the step that are greater than the previous top value and the current top value, but not equal to either.
DISTINCT_RANGE_ROWS	These are the distinct count of rows within a step. If all the rows are unique, then the RANGE_ROWS and the DISTINCT_RANGE_ROWS will be equal.
AVG_RANGE_ROWS	This represents the average number of rows equal to a key value within the step.

Статистики

- Статистики обновляются автоматически
- Иногда эвристики автообновления не срабатывают
- Тогда нужно обновить вручную
- Если Actual Rows = Estimated Rows – статистики ни при чём

Кеширование планов выполнения

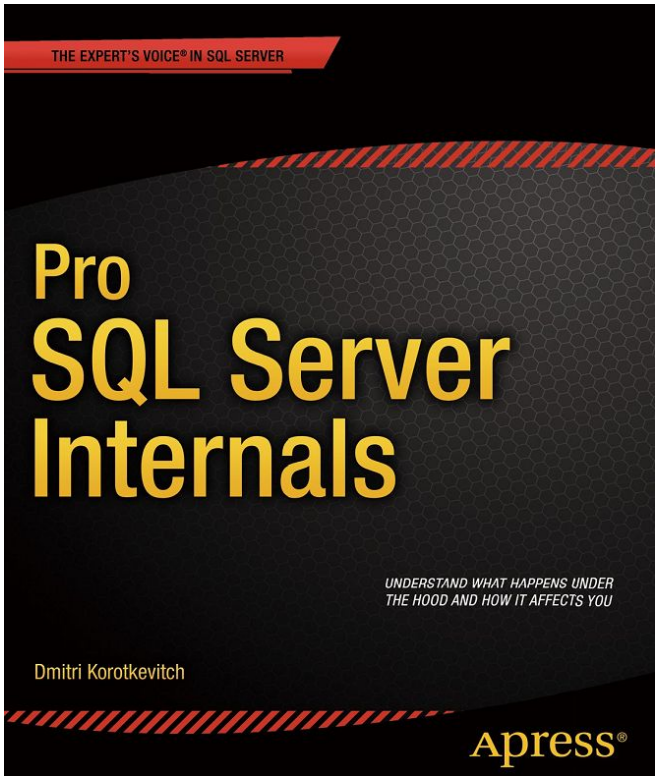
- Планы кешируются
- Разные значения параметров –> один кеш
- Parameter Sniffing
- OPTIMIZE FOR UNKNOWN hint

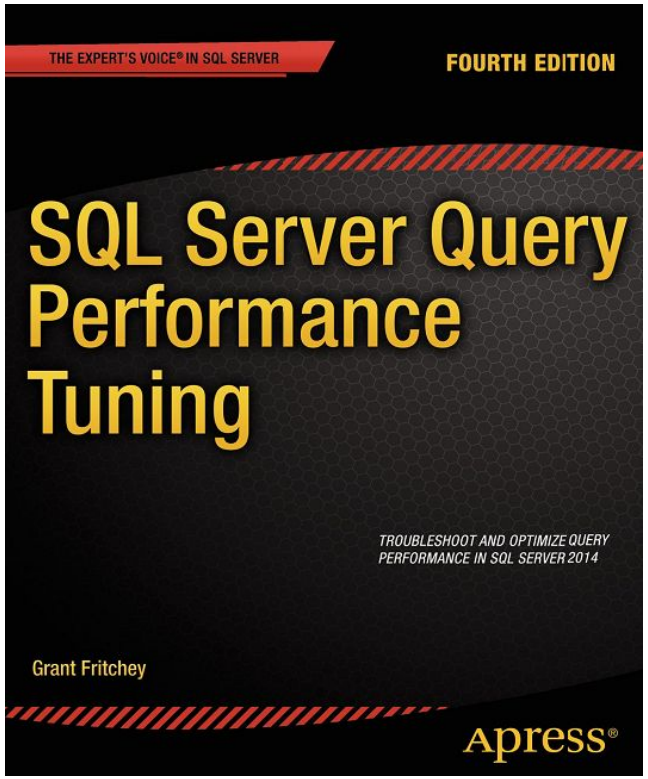
Query optimization and execution

- Просмотр кешированных планов выполнения

```
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 as [Exec Cnt]
,(qs.total_logical_reads + qs.total_logical_writes)
/ qs.execution_count as [Avg IO]
,qp.query_plan as [Plan]
,qs.total_logical_reads as [Total Reads]
,qs.last_logical_reads as [Last Reads]
,qs.total_logical_writes as [Total Writes]
,qs.last_logical_writes as [Last Writes]
,qs.total_worker_time as [Total Worker Time]
,qs.last_worker_time as [Last Worker Time]
,qs.total_elapsed_time/1000 as [Total Elps Time]
,qs.last_elapsed_time/1000 as [Last Elps Time]
,qs.creation_time as [Compile Time]
,qs.last_execution_time as [Last Exec Time]
from
sys.dm_exec_query_stats qs with (nolock)
cross apply sys.dm_exec_sql_text(qs.sql_handle) qt
cross apply sys.dm_exec_query_plan(qs.plan_handle) qp
order by
qs.last_execution_time desc
--[Avg IO] desc
option (recompile)
```

Что почитать





http://www.brentozar.com/
https://www.youtube.com/user/BrentOzar
…

nikolay.shestakov@rubius.com

Транзакции

- ACID
  - Atomicity
  - Consistency
  - Isolation
  - Durability

Транзакции

Atomicity – всё или ничего

Транзакции

Durability – если транзакция выполнена, она *выполнена* (результат устойчив к системным сбоям)

Транзакции

Consistency

- Данные согласованы в начале транзакции и после окончания транзакции (но не обязательно внутри транзакции)
- Согласованные данные – удовлетворяющие ограничениям целостности и бизнес-правилам

- Иногда свойство понимают в том смысле, что результаты завершённых транзакций должны быть видны последующим транзакциям

Transactions

Isolation

- Выполняющиеся параллельно транзакции логически не влияют друг на друга
- Принцип сериализации: транзакции, выполняющиеся параллельно, должны логически выполняться так, как будто они запущены по очереди
- На практике сериализация обходится дорого, поэтому поддержка изоляции сводится к поддержке выполнения условий заданного *уровня изоляции транзакции*

Уровни изоляции транзакций

- Read Uncommitted
  - Разрешены грязные чтения
- Read Committed
  - Чтения только зафиксированных данных, но повторное чтение может вернуть изменённые данные
- Repeatable Read
  - Повторное чтение внутри транзакции всегда возвращает одинаковые данные для прочитанных ранее записей. Но могут появиться новые записи, которых раньше не было
- Serializable
  - Полная сериализация!

- Snapshot
  - Используется мультиверсионность записей, каждая транзакция видит состояние БД, которое было на момент её начала

Уровни изоляции транзакций

- Read Uncommitted
  - Быстр
  - Никого не ждёт
  - Наибольшая вероятность получить несогласованные данные
  - При чтении ничего не блокирует

Уровни изоляции транзакций

- Read Committed
  - Ждёт освобождения exclusive lock
  - Есть вероятность получить изменённые данные при повторных чтениях
  - При чтении ничего не блокирует

Уровни изоляции транзакций

- Repeatable Read
  - Ждёт освобождения exclusive lock
  - При чтении ставит Shared lock
  - При повторных чтениях могут появляться фантомные записи

Уровни изоляции транзакций

- Serializable
  - Обеспечивает 100% изоляцию
  - Ждёт освобождения exclusive lock
  - При чтении ставит Shared lock на диапазон значений атрибутов (диапазон берётся из условия запроса)

Locks

- Shared Lock
  - Блокирует изменение (попытки Update и Exclusive), позволяет чтение. Несколько транзакций могут установить одновременно на один ресурс.
- Exclusive Lock
  - Блокирует изменение (попытки Update и Exclusive) и чтение. Только одна транзакция может установить одновременно на один ресурс. Запрашивается для изменения данных.
- Key-Range
  - Запрещает INSERT по значениям диапазона ключей
- Update Lock
  - Блокирует изменение (попытки Update и Exclusive), позволяет чтение. Только одна транзакция может установить одновременно на один ресурс. Потом для обновления записи запрашивается Exclusive.
- Intent Lock, Schema Lock, Bulk Update Lock

Deadlocks

- Вероятность взаимных блокировок повышается, если:
  - Большое количество параллельных транзакций, которые меняют данные
  - Используются долгие сложные транзакции, состоящие из нескольких операций
  - Используются операции, блокирующие большое количество записей (агрегатные, типа SUM)
  - Используется высокий уровень изоляции
  - Не используется MVCC (Snapshot Isolation)

Snapshot Isolation

- Multiversion Concurrency Control
  - Используется timestamp для обозначения версии записи
  - Транзакции читают версии записей, соответствующие моменту начала транзакции
  - При изменении данных, старые записи перемещаются в хранилище Version Store (в tempdb)
  - В SQL Server включается для БД целиком:
    - SET READ_COMMITTED_SNAPSHOT ON или
    - SET ALLOW_SNAPSHOT_ISOLATION ON
- Snapshot Isolation исключает аномалии грязных, повторных и фантомных чтений
- Отсутствуют блокировки при чтении
- Snapshot Isolation не обеспечивает Serializable
- «Serializable» в Oracle – на самом деле Snapshot!

Snapshot Isolation

- Snapshot Isolation исключает аномалии грязных, повторных и фантомных чтений
- Отсутствуют блокировки при чтении
- Snapshot Isolation не обеспечивает Serializable
- «Serializable» в Oracle – на самом деле Snapshot!
- (аналогично в PostgreSQL < 9.1)
- MVCC режим выгоден при большом количестве параллельных транзакций, но требует ресурсов на управление версиями данных

Snapshot Isolation write-skew anomaly

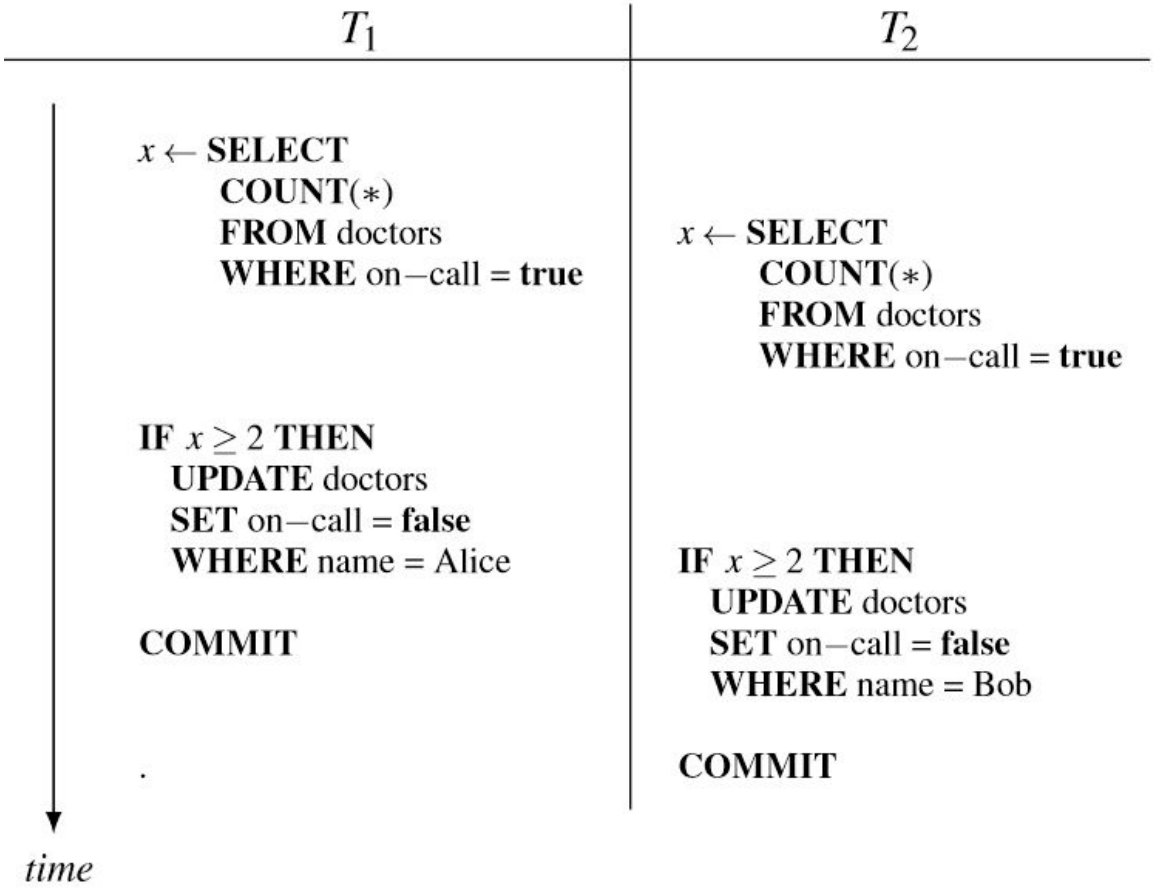


Figure 1: A simple write-skew anomaly

Serializable Snapshot Isolation in PostgreSQL (2012)
http://drkp.net/papers/ssi-vldb12.pdf

Monitoring locks and deadlocks

- SSMS Activity Monitor
- Performance Monitor SQL Server Lock counters
- Dynamic Management Views
  - sys.dm_exec_requests
  - sys.dm_tran_locks
  - sys.dm_os_waiting_tasks
- SQL Server Profiler
- SQL Server Extended Events (since 2012) http://www.brentozar.com/archive/2014/04/introduction-extended-events/

http://www.mssqltips.com/sqlservertip/2732/different-techniques-to-identify-blocking-in-sql-server/

http://www.brentozar.com/archive/2014/06/capturing-deadlock-information/