

3.2.3. Секционирование

Таблицы фактов обычно содержат очень большой объем данных, поэтому управление ими может вызывать затруднения. Вставка больших объемов данных и возможное перестроение индексов может занимать значительное время, к тому же может потребоваться перезагрузить ранее загруженные данные, а для этого нужно удалить часть данных из хранилища, что тоже требует временных затрат.

Секционирование позволяет упростить управление большой таблицей за счет горизонтального разбиения её на секции, каждая из которых хранится в отдельной файловой группе (рис. 50).

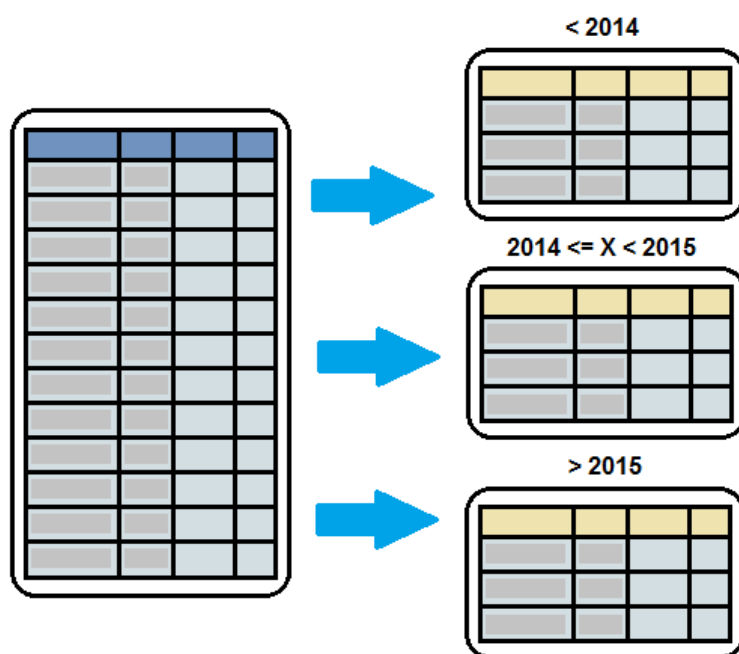


Рис. 50. Секционирование таблицы на основе столбца дат

Секционирование имеет следующие преимущества:

- ✓ увеличение производительности запросов за счет распараллеливания операций считывания с разных дисков, на которых хранятся секции;
- ✓ ускорение загрузки и удаления данных в хранилище, при использовании метода скользящего окна;
- ✓ использование более гибких вариантов резервного копирования и восстановления по секциям;
- ✓ уменьшение диапазона блокировок;

- ✓ возможность управлять индексами на уровне секции, например, восстанавливать индекс для одной секции, где данные были изменены.

Этот механизм доступен только в выпусках SQL Server Enterprise Edition, Developer Edition и Evaluation Edition.

Начиная с 2005 версии, SQL Server поддерживает автоматическое секционирование. Для пользователей таблица фактов представляется единой и неделимой, а сервер сам поддерживает заданную схему секционирования и распределяет данные по секциям, определяя необходимость объединения секций для обслуживания пользовательских запросов.

Для реализации секционирования нужно выполнить следующие шаги.

- ✓ Выбрать ключ секционирования, т.е. поле, по которому будут определяться границы секций.
- ✓ Создать функцию секционирования (Partition function), указывающую границы секций.
- ✓ Затем создать схему секционирования, управляющую размещением секций в файловой системе с использованием файловых групп.

В качестве ключа секционирования можно выбрать одно любое поле таблицы, в том числе и вычисляемое, за исключением полей следующих типов: timestamp, ntext, text, image, xml, varchar(max), nvarchar(max) и varbinary(max).

Если секционирование задается на кластеризованной таблице, то ключ секционирования должен быть частью первичного ключа или кластеризованного индекса.

Как правило, секционирование проводят на основе ключевого поля даты. Интервал (день, месяц, год и т.д.) выбирается исходя из потребностей пользователей. В SQL Server 2012 таблица может иметь до 15 000 секций.

Ниже приведен пример разделения таблицы фактов «FactSales» на три секции: продажи за текущий год, продажи за предыдущий год и продажи за все остальные годы.

Сначала создается функция секционирования:

```
CREATE PARTITION FUNCTION PartFunctionFactSales_Date (bigint)
```

AS RANGE RIGHT FOR VALUES (20140101, 20150101)

, где **PartFunctionFactSales_Date** – название функции секционирования, **bigint** — тип данных ключа секционирования, в скобках указаны границы для секций. **RANGE LEFT | RIGHT** – необязательный параметр, который указывает какая граница интервала будет открытой, а какая замкнутой. **LEFT** – открытая граница слева, **RIGHT** – открытая граница справа. Если значение не задано, то по умолчанию используется значение **LEFT**.

Область действия функции секционирования ограничена базой данных, в которой она была создана. Посмотреть список всех функций секционирования для конкретной базы данных можно в обозревателе объектов, пройдя по узлам дерева: **База данных -> Хранилище (Storage)- > Функции секционирования (Partition Functions)**.

Приведенная выше функция распределит строки данных таблицы фактов по секциям следующим образом:

Секция	1	2	3
Интервал значений	Ключ даты <20140101	20140101<= Ключ даты <20150101	20150101<= Ключ даты

Если бы не был указан параметр **RIGHT**, то интервалы значений выглядели бы так:

Секция	1	2	3
Интервал значений	Ключ даты <=20140101	20140101< Ключ даты <=20150101	20150101< Ключ даты

Затем необходимо задать схему секционирования. Это позволит привязать каждую секцию к определенной файловой группе и разместить их на разных дисках.

```
CREATE PARTITION SCHEME PartSchFactSales_Date
AS PARTITION PartFunctionFactSales_Date TO
([Fast_Growing],
[Frequently_Requested],
[Frequently_Requested])
```

Схема секционирования обязательно опирается на определенную функцию секционирования и в скобках перечисляются те файловые группы, к которым следует привязать каждую секцию. В данном примере продажи за

текущий и предыдущий годы помещаются в файловую группу для наиболее часто запрашиваемых данных, а продажи за все остальные годы в файловую группу для быстро растущих таблиц.

Посмотреть список всех схем секционирования можно в обозревателе объектов, пройдя по узлам дерева: **База данных -> Хранилище (Storage)- > Схема секционирования (Partition Schemes)**.

Очевидно, что файловые группы, функция и схема секционирования должны существовать в БД до того, как будет создана таблица.

Ниже приведен пример кода создания секционированной таблицы фактов «FactSales», которая логически представляется в базе данных как одна таблица.

При создании такой таблицы вместо указания файловой группы указывается схема секционирования и столбец, по которому будет проведено разбиение. Если в таблице задан первичный ключ, то в его составе должно быть поле ключа секционирования.

```
CREATE TABLE FactSales
(NumberSale INT NOT NULL,
NumberLineInSale INT NOT NULL,
CustomerKey INT NOT NULL,
CustomerDemographyKey INT,
ProductKey INT NOT NULL,
DateKey BIGINT NOT NULL,
ManagerKey INT,
KindOfPayKey INT,
PromotionKey INT,
Quantity INT NOT NULL,
UnitPrice DECIMAL(15,2) NOT NULL,
Subtotal DECIMAL(15,2)NOT NULL,
DiscountCustAmt DECIMAL(15,2),
PromoAmt DECIMAL(15,2),
TaxAmt DECIMAL(15,2)NOT NULL,
Freight DECIMAL(15,2),
Total DECIMAL(15,2)NOT NULL,
CONSTRAINT PKDW_10 PRIMARY KEY (NumberSale,NumberLineInSale,DateKey)
) ON PartSchFactSales_Date(DateKey)
```

Информация о секциях хранится в специальной системной таблице Sys.Partitions, посмотреть которую можно выполнив запрос, представленный на рис. 51.

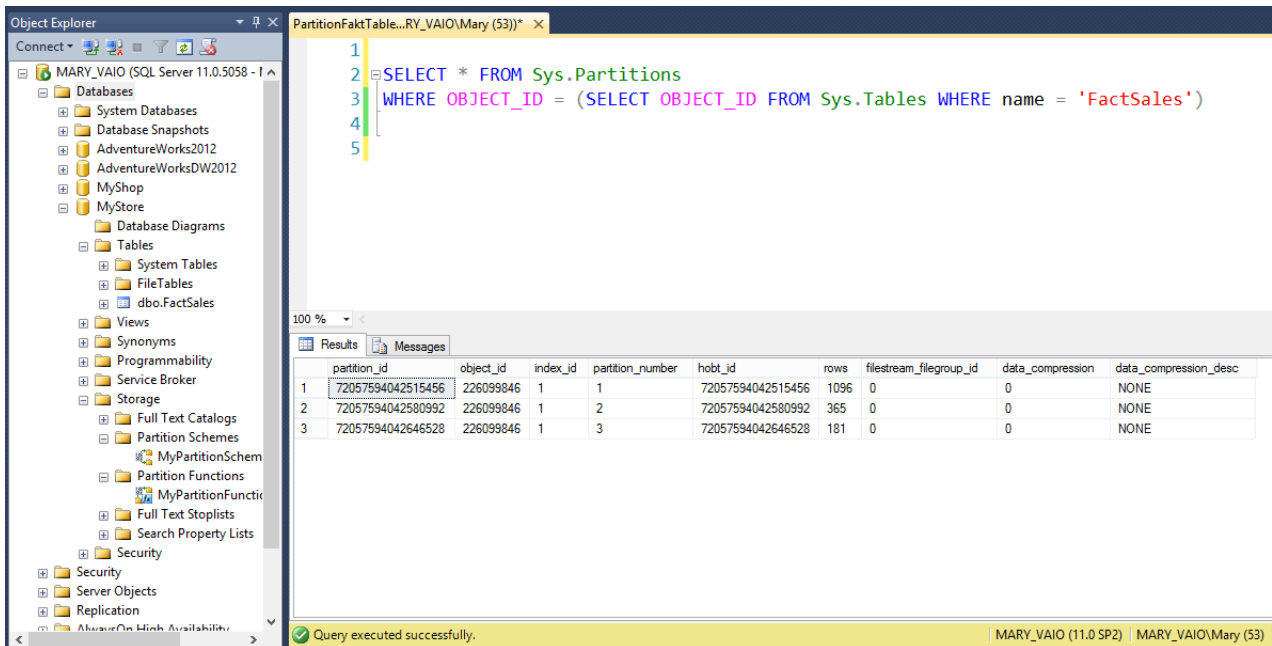


Рис. 51. Просмотр информации о секциях таблицы «FactSales»

Секции нумеруются автоматически, начиная с единицы (столбец `partition_number`). Первая секция содержит самые ранние данные. В столбце `rows` показано количество строк данных в каждой секции, а столбец `index_id` показывает, задан ли кластеризованный индекс на таблице, 0 – нет, 1 – да.

Для каждой секции строится отдельное индексное дерево и, следовательно, ускорение запросов на секционированной таблице происходит, если указать поле ключа секционирования в условии `WHERE`. В этом случае просматривается только определенная секция, а не вся таблица целиком.

На рис. 52 показан пример выполнения запроса, затрагивающего только одну секцию таблицы (`Actual partition count = 1`), хранящую данные за 2011 год.

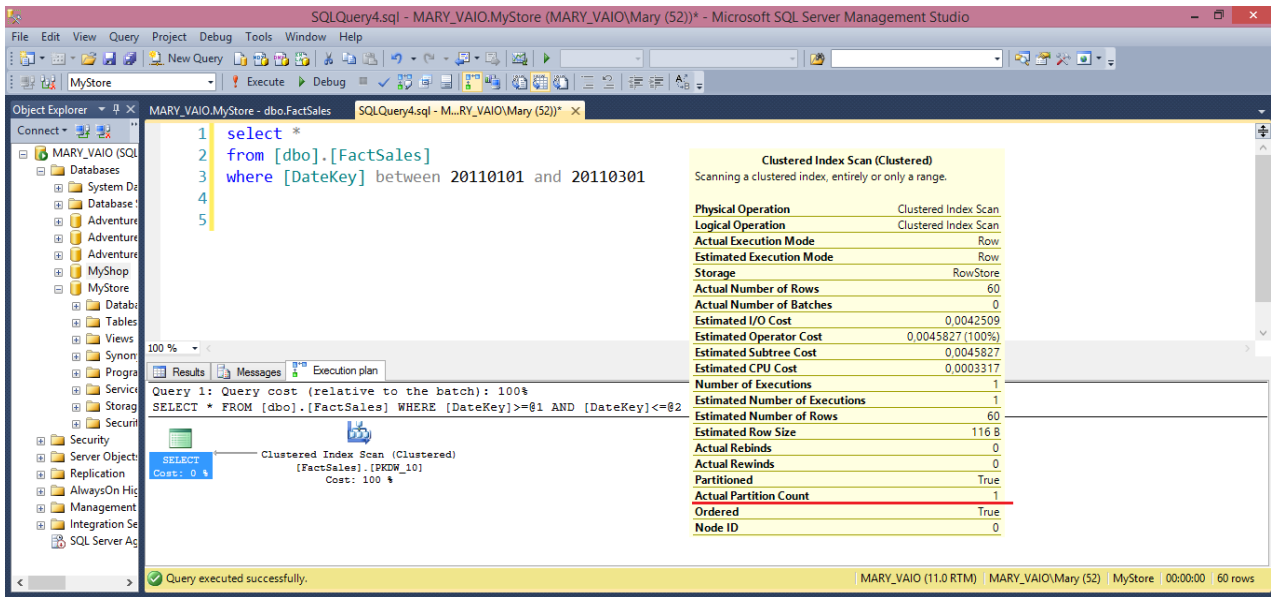


Рис. 52. Запрос, затрагивающий только одну секцию таблицы «FactSales»

Помимо секционирования таблиц (кучи или кластеризованного индекса) можно также секционировать индексы, которые в этом случае могут быть двух видов.

Выровненный индекс – индекс, созданный на основе той же схемы секционирования, что и соответствующая таблица. Если таблица и ее индексы выровнены, SQL Server может быстро переключаться с секции на секцию, сохраняя при этом структуру секций как таблицы, так и ее индексов. Для выравнивания с базовой таблицей индексу необязательно использовать функцию секционирования с тем же именем. Однако функции секционирования индекса и базовой таблицы не должны существенно различаться, то есть:

- ✓ аргументы функции секционирования должны иметь один и тот же тип данных;
- ✓ функции должны определять одинаковое количество секций;
- ✓ функции должны определять для секций одинаковые граничные значения.

Невыровненный индекс – индекс, секционированный независимо от соответствующей таблицы, т. е. имеющий другую схему секционирования или находящийся в другой файловой группе, нежели базовая таблица.

В любом случае, при секционировании индексов следует придерживаться следующих правил:

- ✓ для секционирования уникальных индексов (кластеризованного или некластеризованного) столбец секционирования должен содержаться в ключе индекса;
- ✓ для секционирования неуникального кластеризованного индекса, если столбец секционирования не указан явно в ключе кластеризации, SQL Server по умолчанию добавляет столбец секционирования в список ключей кластеризованного индекса;
- ✓ для секционирования неуникального некластеризованного индекса SQL Server по умолчанию добавляет столбец секционирования как неключевой (включенный) столбец индекса, чтобы обеспечить выравнивание индекса с базовой таблицей. Если столбец секционирования уже присутствует в индексе, SQL Server его не добавляет.

Ниже приведен пример создания секционированного выровненного неуникального некластеризованного индекса для таблицы «FactSales» при этом столбец секционирования «DateKey» явно в индекс не входит, а добавляется SQL Server автоматически.

```
CREATE INDEX NonCL_Index_ProductKey  
ON [dbo].[FactSales] ([ProductKey]) INCLUDE ([UnitPrice]) ON  
[PartSchFactSales_Date]([DateKey])
```

Теперь посмотрим на вновь образованные секции для таблицы «FactSales» и её индексов (рис. 53). В столбце «Index_id» значение 1 – задан кластеризованный индекс, 3 – некластеризованный индекс.

The screenshot shows a SQL Server Enterprise Manager interface. The top pane displays a query window with the following SQL code:

```

1 CREATE INDEX NonCL_Index_ProductKey
2 ON [dbo].[FactSales] ([ProductKey]) INCLUDE ([UnitPrice]) ON [PartSchFactSales_Date]([DateKey])
3
4 SELECT * FROM Sys.Partitions
5 WHERE OBJECT_ID = (SELECT OBJECT_ID FROM Sys.Tables WHERE name = 'FactSales')

```

The bottom pane shows the 'Results' tab with a grid of 6 rows and 9 columns. The columns are: partition_id, object_id, index_id, partition_number, hobt_id, rows, filestream_filegroup_id, data_compression, and data_compression_desc. The data is as follows:

partition_id	object_id	index_id	partition_number	hobt_id	rows	filestream_filegroup_id	data_compression	data_compression_desc
1	72057594039959552	949578421	1	72057594039959552	1096	0	0	NONE
2	72057594040025088	949578421	1	72057594040025088	365	0	0	NONE
3	72057594040090624	949578421	1	72057594040090624	304	0	0	NONE
4	72057594040156160	949578421	3	72057594040156160	1096	0	0	NONE
5	72057594040221696	949578421	3	72057594040221696	365	0	0	NONE
6	72057594040287232	949578421	3	72057594040287232	304	0	0	NONE

The status bar at the bottom indicates: 'Query executed successfully. MARY_VAIO (11.0 RTM) MARY_VAIO\Mary (52) MyStore 00:00:01 6 rows'.

Рис. 53. Секции таблицы «FactSales»

При работе с секциями нужно уметь выполнять следующие операции.

- ✓ Переключение секции (switch partition), т.е. переприсваивание блоков данных из одной таблицы или секции другой таблице или секции. Эта процедура выполняется с помощью инструкции T-SQL - ALTER TABLE.

Можно переключать:

- все данные из несекционированной таблицы на имеющуюся пустую секцию секционированной таблицы;
- секции одной секционированной таблицы на секции другой секционированной таблицы;
- все данные из секции секционированной таблицы на существующую пустую несекционированную таблицу.

При этом все таблицы, секции, соответствующие индексы или секции индексов должны быть расположены в одной файловой группе, для того чтобы избежать физического перемещения данных между дисками. В этом случае это операция только над метаданными.

Операцию переключения между секциями часто применяют при загрузке хранилища, используя две вспомогательные неиндексированные таблицы с той же структурой, что и секционированная таблица для

вставки или удаления больших порций данных с минимальным протоколированием. При вставке данные добавляются во вспомогательную таблицу, а затем переключаются на пустую секцию основной таблицы. Это позволяет не блокировать основную таблицу на время загрузки данных, переключение между секциями занимает минимальное время. При удалении секция переключается на пустую таблицу, а затем данные из неё удаляются.

- ✓ Разделение секции (split partition). Позволяет разделить существующую секцию на две. Эта процедура выполняется с помощью инструкции T-SQL - `ALTER PARTITION FUNCTION {SPLIT RANGE (boundary_value)}`. Аргумент `boundary_value` определяет диапазон новой секции, который должен отличаться от существующих пограничных значений функции секционирования. Для того чтобы назначить файловую группу для вновь создаваемых секций, нужно изменить схему секционирования, установив для файловой группы свойство `NEXT USED` с помощью инструкции `ALTER PARTITION SCHEME partition_scheme_name NEXT USED [filegroup_name]`.
- ✓ Слияние двух секций (merge partition). Удаляет секцию и объединяет все значения, существующие в секции, в одну из оставшихся. Эта процедура также выполняется с помощью инструкции T-SQL - `ALTER PARTITION FUNCTION {MERGE RANGE (boundary_value)}`. Аргумент `boundary_value` должен быть существующим пограничным значением для двух объединяемых секций.

Хорошей практикой является резервирование пустых секций в начале и в конце диапазона секций, для того чтобы избежать перемещения данных при разбиении секций (до загрузки новых данных) и их объединении (после выгрузки старых данных).

Это особенно важно при реализации метода *скользящего окна* (sliding window), позволяющего динамически секционировать таблицы.

С помощью этого метода можно не только физически, но и логически

(явно для пользователя) разделить данные из таблицы фактов, перенося устаревшие в архивную таблицу или удаляя, в зависимости от потребностей бизнеса.

Дело в том, что при секционировании логически таблица представляется единой и это накладывает свои ограничения, такие как необходимость поддерживать одинаковые индексы на всех секциях или наложение блокировок на всю таблицу при определенных действиях, поэтому иногда бывает полезным явно разделить данные.

Для реализации метода необходимо создать архивную таблицу с такой же структурой, как и секционированная таблица и выполнить следующие шаги.

- ✓ При создании новой секции, куда будут загружаться свежие данные, выполнить операцию SPLIT на последней пустой секции таблицы фактов. Выполнить ту же операцию для подготовки новой пустой секции в архивной таблице.
- ✓ Для выгрузки устаревших данных нужно переключить секцию, их содержащую, в пустую секцию архивной таблицы, и выполнить операцию MERGE на этой секции, ту же операцию выполнить в архивной таблице, объединив пустую и только что загруженную секции. При этом физического перемещения данных не происходит, если секции расположены в одной файловой группе.
- ✓ создать вспомогательную таблицу без индексов для загрузки новых данных, затем загрузить в неё данные и построить индексы для подготовки к переключению в новую секцию основной таблицы.

На рис. 54 подробно показан процесс перестройки секций при реализации «скользящего окна».

Таблица фактов

пустая секция < 2013	2013	2014	2015	пустая секция 2016
-------------------------	------	------	------	-----------------------

Архивная таблица

пустая секция < 2013	пустая секция 2013
-------------------------	-----------------------

1) Добавление новой секции (SPLIT)**Таблица фактов**

пустая секция < 2013	2013	2014	2015	пустая секция 2016	пустая секция 2017
-------------------------	------	------	------	-----------------------	-----------------------

Архивная таблица

пустая секция < 2013	пустая секция 2013	пустая секция 2014
-------------------------	-----------------------	-----------------------

2) Перенос данных за 2013 год в архивную таблицу (SWITCH)**Таблица фактов**

пустая секция < 2013	2013	2014	2015	пустая секция 2016	пустая секция 2017
-------------------------	------	------	------	-----------------------	-----------------------

**Архивная таблица**

пустая секция < 2013	пустая секция 2013	пустая секция 2014
-------------------------	-----------------------	-----------------------

3) Слияние первых двух секций (MERGE)**Таблица фактов**

пустая секция 2013	2014	2015	пустая секция 2016	пустая секция 2017
-----------------------	------	------	-----------------------	-----------------------

Архивная таблица

2013	пустая секция 2014
------	-----------------------

Рис. 54. Реализация метода «скользящего окна»

Можно создать хранимую процедуру без параметров, которая будет запускаться по расписанию через SQL Agent и позволит автоматически переключать секции.

Ниже приведен пример процедуры для создания «скользящего окна» с шагом в один год и пятью секциями для таблицы «FactSales». В таблице должны храниться данные только за текущий и два предыдущих года, а данные за более ранние года перемещаются в архивную таблицу «ArchivalFactSales»,

имеющую ту же структуру, что и таблица «FactSales». В начале и в конце диапазона зарезервированы пустые секции, для быстрого разделения и объединения секций.

1. Создание функций и схем секционирования для таблиц «FactSales» и «ArchivalFactSales».

```
CREATE PARTITION FUNCTION PartFunctionFactSales_Date (bigint)
AS RANGE RIGHT FOR VALUES (20130101,20140101,20150101,20160101)
```

```
CREATE PARTITION SCHEME PartSchFactSales_Date
AS PARTITION PartFunctionFactSales_Date TO
([Fast_Growing],
[Fast_Growing],
[Frequently_Requested],
[Frequently_Requested],
[Frequently_Requested])
```

```
CREATE PARTITION FUNCTION PartFunctionForArchivalTable (bigint)
AS RANGE RIGHT FOR VALUES (20130101)
```

```
CREATE PARTITION SCHEME PartSchForArchivalTable
AS PARTITION PartFunctionForArchivalTable TO
([Fast_Growing],
[Fast_Growing])
```

2. Создание таблиц «FactSales» и «ArchivalFactSales».

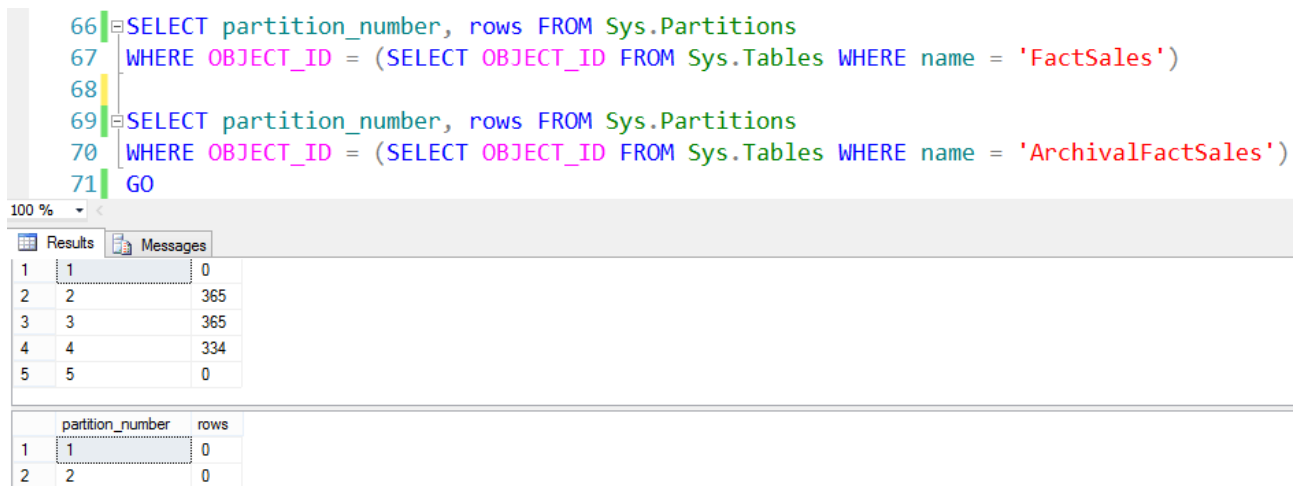
```
CREATE TABLE FactSales
(NumberSale INT NOT NULL,
NumberLineInSale INT NOT NULL,
CustomerKey INT NOT NULL,
CustomerDemographyKey INT,
ProductKey INT NOT NULL,
DateKey BIGINT NOT NULL,
ManagerKey INT,
KindOfPayKey INT,
PromotionKey INT,
Quantity INT NOT NULL,
UnitPrice DECIMAL(15,2) NOT NULL,
Subtotal DECIMAL(15,2) NOT NULL,
DiscountCustAmt DECIMAL(15,2),
PromoAmt DECIMAL(15,2),
TaxAmt DECIMAL(15,2) NOT NULL,
Freight DECIMAL(15,2),
Total DECIMAL(15,2) NOT NULL,
CONSTRAINT PKDW_10 PRIMARY KEY (NumberSale,NumberLineInSale,DateKey)
) ON PartSchFactSales_Date(DateKey)
GO
CREATE TABLE ArchivalFactSales
```

```

(NumberSale INT NOT NULL,
NumberLineInSale INT NOT NULL,
CustomerKey INT NOT NULL,
CustomerDemographyKey INT,
ProductKey INT NOT NULL,
DateKey BIGINT NOT NULL,
KindOfPayKey INT,
PromotionKey INT,
Quantity INT NOT NULL,
UnitPrice DECIMAL(15,2) NOT NULL,
Subtotal DECIMAL(15,2) NOT NULL,
DiscountCustAmt DECIMAL(15,2),
PromoAmt DECIMAL(15,2),
TaxAmt DECIMAL(15,2) NOT NULL,
Freight DECIMAL(15,2),
Total DECIMAL(15,2) NOT NULL,
CONSTRAINT PK_Arch_1 PRIMARY KEY (NumberSale,NumberLineInSale,DateKey)
) ON PartSchForArchivalTable(DateKey)

```

3. После заполнения таблицы «FactSales» данными выполним запросы к системной таблице Sys.Partitions и посмотрим на содержимое секций для таблиц «FactSales» и «ArchivalFactSales» (рис. 55).



```

66 SELECT partition_number, rows FROM Sys.Partitions
67 WHERE OBJECT_ID = (SELECT OBJECT_ID FROM Sys.Tables WHERE name = 'FactSales')
68
69 SELECT partition_number, rows FROM Sys.Partitions
70 WHERE OBJECT_ID = (SELECT OBJECT_ID FROM Sys.Tables WHERE name = 'ArchivalFactSales')
71 GO

```

partition_number	rows
1	0
2	365
3	365
4	334
5	0

partition_number	rows
1	0
2	0

Рис. 55. Секции, изначально созданные для таблиц «FactSales» и «ArchivalFactSales»

Для таблицы «FactSales» образовано 5 секций. Данные содержатся в секциях за 2013, 2014 и 2015 год. Секция для более ранних данных и за 2016 год данных не содержат. Для таблицы «ArchivalFactSales» образованы две пустые секции.

4. Процедура реализации «скользящего окна».

```

CREATE PROCEDURE Pr_SlidingWindow
AS
DECLARE @DayForPartFactSales VARCHAR(8)

```

```
DECLARE @DayForPartArchival VARCHAR(8)
```

Находим текущую верхнюю границу секций.

```
SET @DayForPartFactSales = CAST((SELECT TOP 1 [value] FROM
sys.partition_range_values
    WHERE function_id = (SELECT function_id
        FROM sys.partition_functions
        WHERE name = 'PartFunctionFactSales_Date')
    ORDER BY boundary_id DESC) AS VARCHAR(8))
```

Находим текущую нижнюю границу секций.

```
SET @DayForPartArchival = CAST((SELECT TOP 1 [value] FROM
sys.partition_range_values
    WHERE function_id = (SELECT function_id
        FROM sys.partition_functions
        WHERE name = 'PartFunctionFactSales_Date')
    ORDER BY boundary_id ASC) AS VARCHAR(8))
```

Определение граничных значений для новых секций. Значение года увеличивается на единицу.

```
DECLARE @Day_DT DATE
SET @Day_DT = DATEADD(YEAR, 1, CAST(@DayForPartFactSales AS DATE))

DECLARE @DayArchival_DT DATE
SET @DayArchival_DT = DATEADD(YEAR, 1, CAST(@DayForPartArchival AS DATE))
```

Назначение файловой группы для вновь создаваемых секций.

```
ALTER PARTITION SCHEME PartSchFactSales_Date
NEXT USED [Frequently_Requested];

ALTER PARTITION SCHEME PartSchForArchivalTable
NEXT USED [Fast_Growing]
```

Операция SPLIT на последней пустой секции таблицы фактов и архивной таблицы.

```
ALTER PARTITION FUNCTION PartFunctionFactSales_Date()
SPLIT RANGE (CAST(CONVERT (VARCHAR(8),@Day_DT, 112) AS BIGINT))

ALTER PARTITION FUNCTION PartFunctionForArchivalTable()
SPLIT RANGE (CAST(CONVERT (VARCHAR(8),@DayArchival_DT, 112) AS BIGINT))
```

После выполнения этих операций для таблицы «FactSales» будет добавлена еще одна секция для данных за 2017 год, а для таблицы «ArchivalFactSales» секция для данных за 2014 год (рис. 56).

```

130 SELECT partition_number, rows FROM Sys.Partitions
131 WHERE OBJECT_ID = (SELECT OBJECT_ID FROM Sys.Tables WHERE name = 'FactSales')
132
133 SELECT partition_number, rows FROM Sys.Partitions
134 WHERE OBJECT_ID = (SELECT OBJECT_ID FROM Sys.Tables WHERE name = 'ArchivalFactSales')

```

100 %

Results Messages

	partition_number	rows
1	1	0
2	2	365
3	3	365
4	4	334
5	5	0
6	6	0

	partition_number	rows
1	1	0
2	2	0
3	3	0

Рис. 56. Секции для таблиц «FactSales» и «ArchivalFactSales», образованные после выполнения операции SPLIT

Перемещение данных из секции таблицы «FactSales» в секцию таблицы «ArchivalFactSales» (рис. 57).

```

ALTER TABLE FactSales
SWITCH PARTITION 2
TO ArchivalFactSales PARTITION 2

```

```

153 SELECT partition_number, rows FROM Sys.Partitions
154 WHERE OBJECT_ID = (SELECT OBJECT_ID FROM Sys.Tables WHERE name = 'FactSales') ORDER BY partition_number
155
156 SELECT partition_number, rows FROM Sys.Partitions
157 WHERE OBJECT_ID = (SELECT OBJECT_ID FROM Sys.Tables WHERE name = 'ArchivalFactSales') ORDER BY
partition_number

```

100 %

Results Messages

	partition_number	rows
1	1	0
2	2	0
3	3	365
4	4	334
5	5	0
6	6	0

	partition_number	rows
1	1	0
2	2	365
3	3	0

Рис. 57. Перемещение данных между секциями таблиц «FactSales» и «ArchivalFactSales»

Объединение секции, из которой данные были перенесены в архивную таблицу, с ближайшей справа секцией таблицы «FactSales» и объединение пустой и только что загруженной секции архивной таблицы.

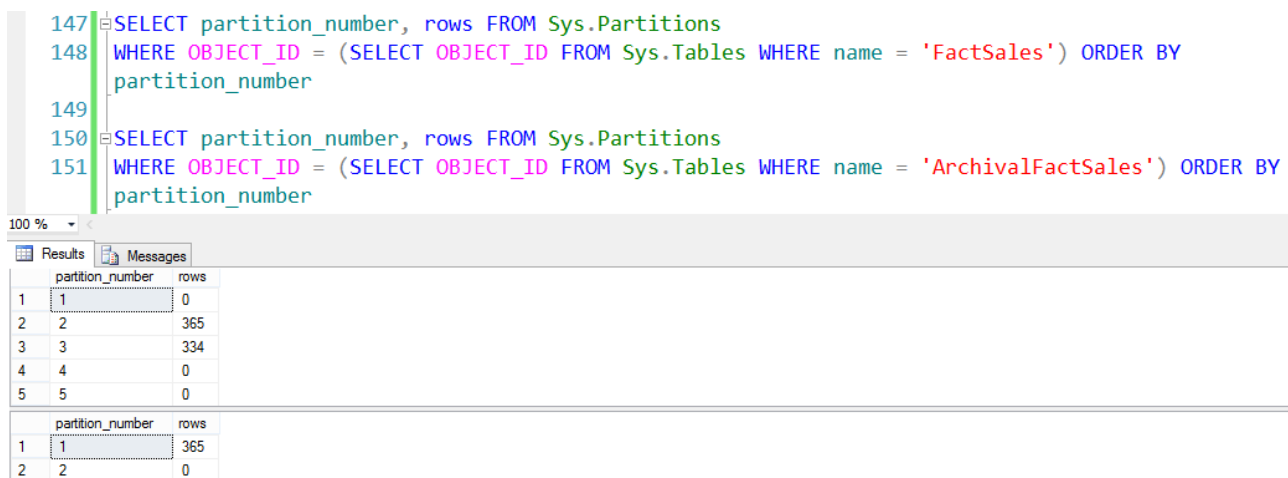
```

ALTER PARTITION FUNCTION PartFunctionFactSales_Date()
MERGE RANGE (CAST(@DayForPartArchival AS BIGINT));

ALTER PARTITION FUNCTION PartFunctionForArchivalTable()
MERGE RANGE (CAST(@DayForPartArchival AS BIGINT));

```

Содержимое секций для таблиц «FactSales» и «ArchivalFactSales» после выполнения процедуры показано на рис. 58.



```

147 SELECT partition_number, rows FROM Sys.Partitions
148 WHERE OBJECT_ID = (SELECT OBJECT_ID FROM Sys.Tables WHERE name = 'FactSales') ORDER BY
    partition_number
149
150 SELECT partition_number, rows FROM Sys.Partitions
151 WHERE OBJECT_ID = (SELECT OBJECT_ID FROM Sys.Tables WHERE name = 'ArchivalFactSales') ORDER BY
    partition_number
  
```

	partition_number	rows
1	1	0
2	2	365
3	3	334
4	4	0
5	5	0

	partition_number	rows
1	1	365
2	2	0

Рис. 58. Результат работы процедуры «Pr_SlidingWindow»

На секционированных таблицах также может быть задан колоночный индекс, если столбец секционирования является одним из столбцов этого индекса, но поскольку в таблицу, где задан колоночный индекс, нельзя загружать данные, то алгоритм загрузки данных в секционированную таблицу с колоночным индексом будет выглядеть следующим образом:

- ✓ Создать временную таблицу с той же структурой, что и таблица в которую необходимо загрузить данные.
- ✓ Загрузить данные во временную таблицу.
- ✓ Построить на временной таблице колоночный индекс.
- ✓ Переключить временную таблицу (SWITCH) на секцию основной таблицы.

3.2.4. Сжатие таблиц

Начиная с 2008 версии, SQL Server позволяет выполнять компрессию строк и страниц таблиц и индексов для экономии дискового пространства, но только в редакции Enterprise. Однако для сжатия и распаковки данных при обмене данными с приложениями требуются дополнительные ресурсы ЦП на сервере баз данных (см. [2]). Для пользователя работа с такой таблицей ничем не отличается от работы с обычной несжатой таблицей.

Сжатие можно применять к следующим объектам базы данных: таблице, хранящейся в виде кучи или кластеризованного индекса; некластеризованному индексу; некластеризованному представлению; секционированной таблице и индексу, причем параметры сжатия можно задать для каждой секции, и разные секции объекта могут иметь различные параметры.

Параметры сжатия таблицы не применяются автоматически к ее некластеризованным индексам. Для системных таблиц сжатие недоступно.

Возможно два типа сжатия.

ROW-сжатие строк является имитацией компрессии за счет уменьшения объема дополнительных метаданных и сохранения столбцов с фиксированным типом данных в формате переменной длины (строки и числовые данные). Например, символы Unicode занимают два байта. Сжатие Unicode (Unicode compression) отводит один байт на хранение тех символов, которым на самом деле не нужны два байта.

Алгоритм сжатия строк немного уменьшает объем данных и слабо влияет на загрузку процессора.

PAGE-сжатие страниц включает сжатие строк, но добавляет сжатие префиксов и словарей. При сжатии префикса (prefix compression) повторяющиеся префиксы значений из одного столбца сохраняются в специальной структуре сжатой информации (compression information, CI), которая располагается сразу за заголовком страницы и повторяющиеся префиксные значения заменяются ссылкой на соответствующий префикс. При сжатии словаря (dictionary compression) повторяющиеся значения, встретившиеся в любом месте страницы, сохраняются в структуре CI. Сжатие словаря не ограничивается значениями в одном столбце (см.[4]).

Этот способ сжатия более эффективен, чем сжатие строк, но вызывает большие нагрузки на ЦП, поэтому не всегда может быть применен.

Применение механизмов сжатия данных в хранилище является полезной практикой, так как данные в денормализованных таблицах очень часто повторяются, в том числе и внешние ключи в таблице фактов. Таблицы

измерения со строками Unicode могут выиграть от сжатия Unicode.

Помимо экономии места, сжатие данных позволяет повысить производительность при рабочих нагрузках с интенсивным вводом-выводом, поскольку данные хранятся в меньшем количестве страниц и в запросах требуется считывать меньше страниц с диска.

Ниже приведен пример применения механизма сжатия к секционированной таблице фактов «FactSales».

Сравним показатели загруженности ЦП и операций чтения с диска и из КЭШ на запросе к несжатой и сжатой таблице. Для просмотра статистики нужно включить соответствующие опции. На рис. 59 показаны значения для несжатой таблицы. Если параметр `physical reads = 0`, значит все данные таблицы загружены в КЭШ.

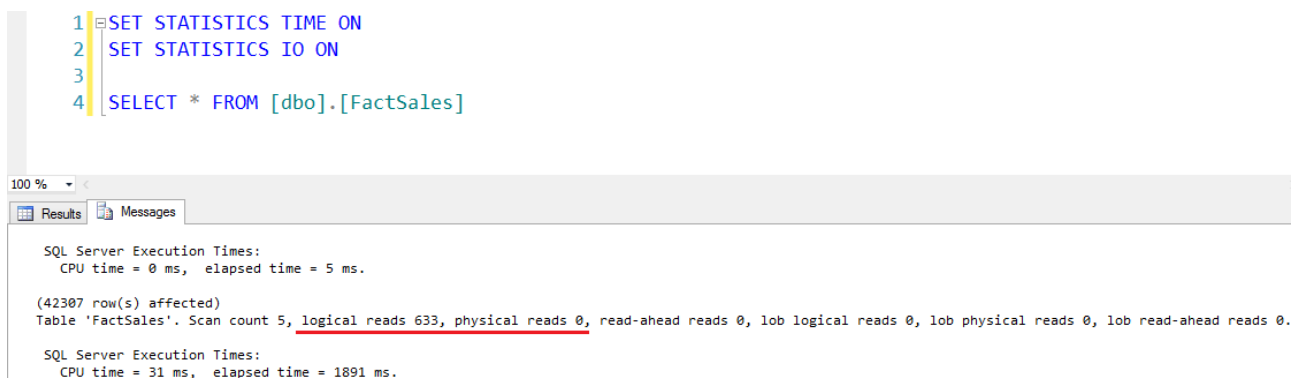


Рис. 59. Статистика запроса к несжатой таблице «FactSales»

Сжатие таблицы можно задать с помощью встроенного мастера SQL Server Management Studio, выбрав из контекстного меню таблицы пункт **Хранилище (Storage) -> Управление сжатием (Manage Compression)**. Так как таблица «FactSales» секционирована, то можно задать режим сжатия для каждой отдельной секции (рис. 60).

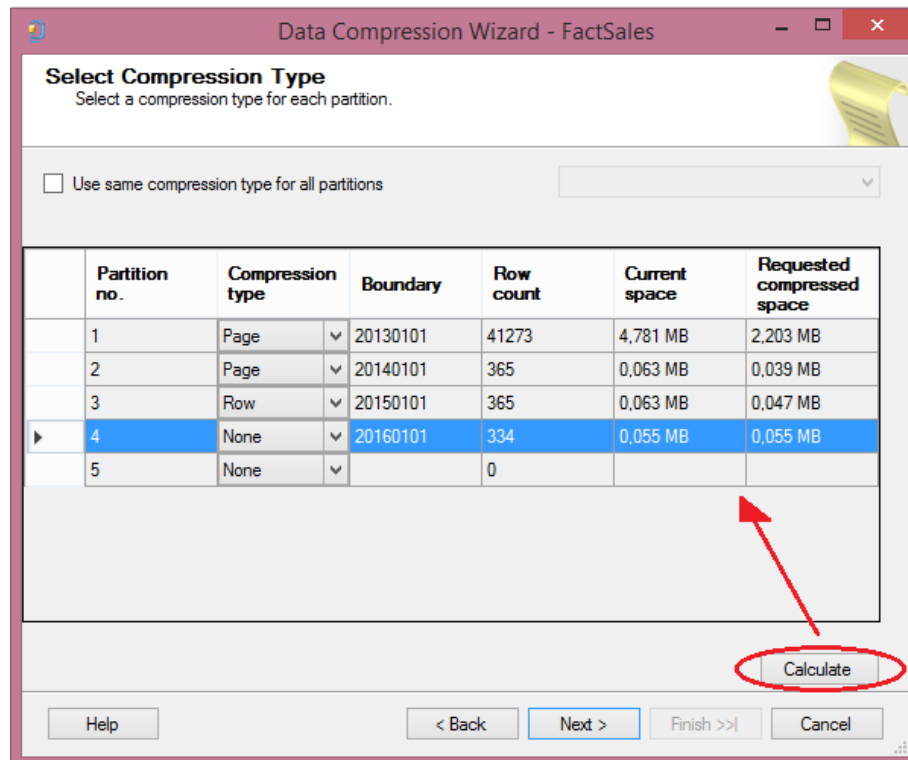


Рис 60. Мастер сжатия данных для таблицы «FactSales»

Для первых двух секций, где хранятся данные за прошедшие года, применяется режим сжатия PAGE, для секции данных за текущий год режим ROW. При нажатии на кнопку **Рассчитать** (Calculate) в последних двух столбцах показываются текущий объем дискового пространства, занимаемый секциями таблицы, и предполагаемый объем после сжатия. Как видно из рисунка, сжатие PAGE существенно уменьшает объем данных.

Выполним тот же запрос на сжатой таблице и посмотрим статистику (рис. 61).

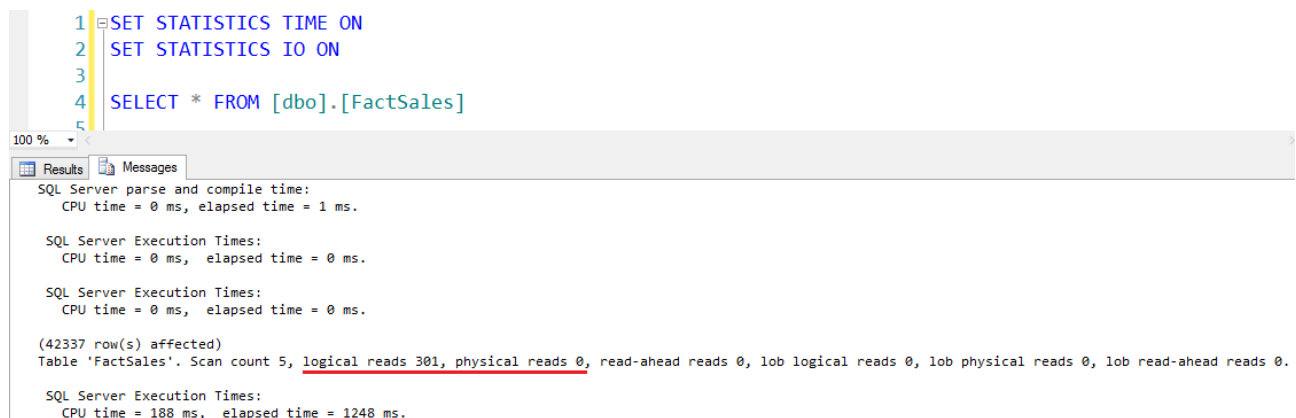


Рис. 61. Статистика запроса к сжатой таблице «FactSales»

Как видно из рисунка, количество операций чтения сократилось вдвое, но и загрузка ЦП увеличилась.

Также режимы сжатия таблицы можно задать с помощью инструкций T-SQL.

```
USE [MyStore]
ALTER TABLE [dbo].[FactSales] REBUILD PARTITION = 1 WITH(DATA_COMPRESSION
= PAGE )
ALTER TABLE [dbo].[FactSales] REBUILD PARTITION = 2 WITH(DATA_COMPRESSION
= PAGE )
ALTER TABLE [dbo].[FactSales] REBUILD PARTITION = 3 WITH(DATA_COMPRESSION
= ROW )
ALTER TABLE [dbo].[FactSales] REBUILD PARTITION = 4 WITH(DATA_COMPRESSION
= NONE )
ALTER TABLE [dbo].[FactSales] REBUILD PARTITION = 5 WITH(DATA_COMPRESSION
= NONE )
```

Отключить сжатие можно с помощью следующей команды:

```
USE [MyStore]
ALTER TABLE [dbo].[FactSales] REBUILD PARTITION = 3 WITH(DATA_COMPRESSION
= NONE )
```

Следует отметить, что колоночные индексы поддерживают собственный патентованный алгоритм сжатия Microsoft **VertiPaq**, для них нельзя использовать сжатие строк или страниц. Это сжатие является более эффективным, чем сжатие типа PAGE и ROW. Колоночный индекс автоматически сжимается при создании и пользователи не могут влиять или контролировать сжатие колоночного индекса.

Более подробную информацию о механизмах сжатия можно найти в электронной документации по ссылке [https://msdn.microsoft.com/ru-ru/library/cc280449\(v=sql.110\).aspx](https://msdn.microsoft.com/ru-ru/library/cc280449(v=sql.110).aspx).

В прил. 2 представлен скрипт создания всех таблиц хранилища данных с привязкой к определенным файловым группам и схемам секционирования, а также с ограничениями в виде первичных и внешних ключей.

В прил. 3 представлен скрипт создания индексов для хранилища данных.

В прил. 4 приведен скрипт заполнения таблицы «dimDate».

Лабораторная работа 2

1. Создание базы данных хранилища

Запустите среду SSMS и подключите ваш экземпляр SQL Server. С помощью графического интерфейса или кода T-SQL, используя контекст базы

данных master, создайте новую базу данных. У базы данных должны быть следующие свойства: один файл данных и один журнал транзакций (по возможности на разных дисках); начальный размер файла данных 50 Мбайт и должен быть включен режим автоувеличения с приращением 10 Мбайт; файл журнала объемом 10 Мбайт с 10-процентным приращением; простая (simple) модель восстановления; автоматическое сжатие отключено.

2. Создание файловых групп

Продумайте количество и назначение файловых групп. С помощью графического интерфейса или кода T-SQL создайте файловые группы (не менее 3-х) и файлы в них (обязательно наличие хотя бы одного файла).

3. Создание таблиц измерений

Создайте таблицы измерений с привязкой к соответствующим файловым группам. Везде, где это только возможно, используйте целочисленные суррогатные ключи с минимальным значением. Установите для них ограничение PRIMARY KEY и автоинкрементное приращение. В хранилище обязательно должна быть хотя бы одна таблица измерения, поддерживающая SCD (медленно меняющееся измерение) по типу 2 и одна таблица мини измерения для поддержки RCD (быстро меняющееся измерение).

Создайте таблицу измерения, основанную на датах, продумайте соответствующий уровень детализации данных и иерархию вдоль временной шкалы. Используйте для ключа даты осмысленное целочисленное значение, получаемое на основе типа данных DATETIME (например, 20150515). Заполнить эту таблицу данными можно с помощью скрипта T-SQL, пример приведен в прил. 4, или с помощью Microsoft® Excel®.

Продумайте, какие из таблиц измерений будут содержать естественные иерархии атрибутов.

4. Создание таблиц фактов

В хранилище должна быть хотя бы одна секционированная таблица фактов. Поэтому перед созданием таблицы, необходимо создать

соответствующую секционирующую функцию с набором граничных значений для секций и схему секционирования.

Как правило, секционирование проводят на основе ключевого поля даты. Помните, что в этом случае поле даты должно быть частью кластеризованного индекса! Создайте таблицы фактов с привязкой к соответствующим схемам секционирования и задайте ограничение PRIMARY KEY.

Затем добавьте ограничения FOREIGN KEY для организации связей между таблицами.

Создайте диаграмму хранилища данных.

5. *Создание некластеризованных индексов*

Постройте некластеризованный индекс на бизнес-ключе для поддержки поиска по суррогатному ключу во время загрузки.

Отчеты ХД могут быть параметризованы. Например, отчет ХД может отображать продажи по определенным регионам, заказчикам и т.д. Пользователь в этом случае использует названия. В подобных отчетах наличие некластеризованного индекса на основе столбца или столбцов содержащих названия, по которым ведется поиск данных, может привести к повышению производительности. Создайте некластеризованные индексы по таким столбцам *в таблицах измерений*. Создайте хотя бы один фильтруемый некластеризованный индекс, для наиболее часто запрашиваемых данных.

Постройте некластеризованные индексы *в таблицах фактов* по наиболее часто используемым внешним ключам и мерам.

6. *Реализация метода «скользящего окна»*

Реализуйте метод «скользящего окна» для таблицы фактов. Для этого нужно создать временную таблицу с той же структурой, что и таблица фактов. Для тестирования заполните таблицу фактов случайными данными. После тестирования удалите данные из таблицы.

7. *Применение сжатия данных и создание индекса колоночного индекса (columnstore)*

С помощью графического интерфейса или кода T-SQL примените сжатие данных в фактической таблице, для каждой секции выберите свой тип сжатия.

Создайте колоночный индекс в таблице фактов, продумав какие поля следует в него включить (помимо мер не забудьте включить внешние ключи, связанные с этими мерами).

Помните, что таблица, на которой задан колоночный индекс становится доступной только для чтения, следовательно, при любых операциях обновления необходимо отключать этот индекс!

Список литературы

1. Барсегян А.А., Куприянов М.С., Степаненко В.В., Холод И.И. Технологии анализа данных: Data Mining, Visual Mining, Text Mining, OLAP - 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2007.
2. Библиотека Microsoft SQL Server. <https://msdn.microsoft.com/ru-ru/library/bb545450.aspx> (дата обращения 5.12.2015)
3. Kimball R. The Data Warehouse Toolkit: the complete guide to dimensional modeling. – 2nd Edition. - Wiley Computer Publishin, 2002.
4. Сарка Д., Лах М., Йеркич Г. Microsoft SQL Server 2012. Реализация хранилищ данных. - М.: Издательство «Русская редакция», 2014.
5. Fritchey G. SQL Server Execution Plans. – 2nd Edition. - Simple Talk Publishing, 2012.

Приложение 1

Полный код создание хранилища данных «MyStore». После создания базы хранилища модель восстановления изменяется на простую (simple) и отключается автоматическое сжатие.

```
USE [master]
GO
IF DB_ID(N'MyStore') IS NOT NULL
DROP DATABASE [MyStore];
GO

CREATE DATABASE [MyStore]
    CONTAINMENT = NONE
    ON PRIMARY
    ( NAME = N'MyStore', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\MyStore.mdf' , SIZE = 51200KB ,
MAXSIZE = UNLIMITED, FILEGROWTH = 10240KB )
    LOG ON
    ( NAME = N'MyStore_log', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\MyStore_log.ldf' , SIZE = 10240KB ,
MAXSIZE = 2048GB , FILEGROWTH = 10%)
GO

ALTER DATABASE [MyStore] SET COMPATIBILITY_LEVEL = 110
GO
ALTER DATABASE [MyStore] SET AUTO_SHRINK OFF
GO
ALTER DATABASE [MyStore] SET RECOVERY SIMPLE WITH NO_WAIT
GO
```

Создание файловых групп и необходимого набора файлов.

```
USE [master]
GO
ALTER DATABASE [MyStore] ADD FILEGROUP [Fast_Growing]
GO
ALTER DATABASE [MyStore] ADD FILE ( NAME = N'MyStore_FastGrowing',
FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\MyStore_FastGrowing.ndf' , SIZE =
358400KB , FILEGROWTH = 51200KB ) TO FILEGROUP [Fast_Growing]
GO
ALTER DATABASE [MyStore] ADD FILEGROUP [Frequently_Requested]
GO
ALTER DATABASE [MyStore] ADD FILE ( NAME =
N'MyStore_FrequentlyRequested', FILENAME = N'C:\Program Files\Microsoft
SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\MyStore_FrequentlyRequested.ndf' ,
SIZE = 204800KB , FILEGROWTH = 10240KB ) TO FILEGROUP
[Frequently_Requested]
```

```

GO
ALTER DATABASE [MyStore] ADD FILEGROUP [Indices]
GO
ALTER DATABASE [MyStore] ADD FILE ( NAME = N'MyStore_Indices', FILENAME =
N'C:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\MyStore_Indices.ndf' , SIZE =
30720KB , FILEGROWTH = 5120KB ) TO FILEGROUP [Indices]
GO
ALTER DATABASE [MyStore] ADD FILEGROUP [MyDefault]
GO
ALTER DATABASE [MyStore] ADD FILE ( NAME = N'MyStore_MyDefault', FILENAME
= N'C:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\MyStore_MyDefault.ndf' , SIZE =
5120KB , FILEGROWTH = 5120KB ) TO FILEGROUP [MyDefault]
GO
ALTER DATABASE [MyStore] ADD FILEGROUP [Read_Only]
GO
ALTER DATABASE [MyStore] ADD FILE ( NAME = N'MyStore_ReadOnly', FILENAME
= N'C:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\MyStore_ReadOnly.ndf' , SIZE =
5120KB , FILEGROWTH = 1024KB ) TO FILEGROUP [Read_Only]
GO
ALTER DATABASE [MyStore] ADD FILEGROUP [Slow_Growing]
GO
ALTER DATABASE [MyStore] ADD FILE ( NAME = N'MyStore_SlowGrowing',
FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\MyStore_SlowGrowing.ndf' , SIZE =
5120KB , FILEGROWTH = 1024KB ) TO FILEGROUP [Slow_Growing]
GO
USE [MyStore]
GO
IF NOT EXISTS (SELECT name FROM sys.filegroups WHERE is_default=1 AND
name = N'MyDefault') ALTER DATABASE [MyStore] MODIFY FILEGROUP
[MyDefault] DEFAULT
GO

```

Приложение 2

Создание таблиц измерений с ограничением PRIMARY KEY.

```
CREATE TABLE dimDate
(KeyDate BIGINT NOT NULL,
[Date] DATE NOT NULL,
[Year] INT NOT NULL,
[Quarter] INT NOT NULL,
[Month] INT NOT NULL,
[Week] INT NOT NULL,
[Day] INT NOT NULL,
[MonthName] NVARCHAR(20) NOT NULL,
[DayName] NVARCHAR(20) NOT NULL
,CONSTRAINT PKDW_1 PRIMARY KEY (KeyDate)
) ON [Frequently_Requested]
```

```
CREATE TABLE dimGeography
(keyGeography INT NOT NULL IDENTITY(1,1),
NameCountry NVARCHAR(50) NOT NULL,
AlphaCode CHAR(3) NOT NULL,
CodeRegion INT NOT NULL,
NameRegion NVARCHAR(45) NOT NULL,
PostalCode NVARCHAR(15) NOT NULL,
NameCity NVARCHAR(30) NOT NULL,
CONSTRAINT PKDW_2 PRIMARY KEY (keyGeography)
) ON [Slow_Growing]
```

```
CREATE TABLE MiniDimCustomerDemography
(keyCustomerDemography INT NOT NULL IDENTITY(1,1),
Age NVARCHAR(10) NOT NULL,
MaritalStatus BIT NOT NULL,
NumberOfChildren INT NOT NULL,
CONSTRAINT PKDW_3 PRIMARY KEY (keyCustomerDemography)
) ON [Slow_Growing]
```

```
CREATE TABLE dimCustomer
(keyCustomer INT NOT NULL IDENTITY(1,1),
CustomerAlternateKey NVARCHAR(12) NOT NULL,
[TaxNumber] CHAR(15),
[Date_of_entry] INT,
Name NVARCHAR(50) NOT NULL,
Gender CHAR(1),
BirthDate DATE,
GeographyKey INT,
Address NVARCHAR(50),
CustomerDemographykey INT,
PhoneNumber VARCHAR(12),
Email VARCHAR(30),
DiscountPct FLOAT,
Rate INT,
```

```

startDate DATE,
endDate DATE,
flag BIT NOT NULL,
CONSTRAINT PKDW_4 PRIMARY KEY (keyCustomer)
) ON [Fast_Growing]

```

```

CREATE TABLE dimPromotion
(keyPromotion INT NOT NULL IDENTITY(1,1),
PromotionAlternateKey NVARCHAR(15) NOT NULL,
Name NVARCHAR(50) NULL,
DiscountPct FLOAT NOT NULL,
StartDate BIGINT NOT NULL,
EndDate BIGINT NOT NULL,
CONSTRAINT PKDW_5 PRIMARY KEY (keyPromotion)
) ON [Fast_Growing]

```

```

CREATE TABLE dimProduct
(keyProduct INT NOT NULL IDENTITY(1,1),
ProductAlternateKey NVARCHAR(25),
CategoryProduct NVARCHAR(50),
NameProduct NVARCHAR(50) NOT NULL,
Manufacturer NVARCHAR(50),
ManufacturingCountry INT,
Color NVARCHAR(30),
Weight DECIMAL(8,2),
Size NVARCHAR(10),
[Material] NVARCHAR(30),
[Type] NVARCHAR(15),
[Kind] NCHAR(1),
CONSTRAINT PKDW_6 PRIMARY KEY (keyProduct)
) ON [Frequently_Requested]

```

```

CREATE TABLE dimKindOfPay
(keyKindOfPay INT NOT NULL IDENTITY(1,1),
Name NVARCHAR(30) NOT NULL,
CONSTRAINT PKDW_7 PRIMARY KEY (keyKindOfPay)
) ON [Slow_Growing]

```

```

CREATE TABLE dimOffices
(keyOffices INT NOT NULL IDENTITY(1,1),
OfficesAlternateKey NVARCHAR(25),
[Name] NVARCHAR(50),
[Address] NVARCHAR(50) NOT NULL,
Chief NVARCHAR(50),
GeographyKey INT,
CONSTRAINT PKDW_8 PRIMARY KEY (keyOffices)
) ON [Slow_Growing]

```

```

CREATE TABLE dimManagers
(keyManager INT NOT NULL IDENTITY(1,1),
ManagerAlternateKey NVARCHAR(25),

```

```

FIO NVARCHAR(50) NOT NULL,
[BirthDate] DATE,
OfficeKey INT,
CONSTRAINT PKDW_9 PRIMARY KEY (keyManager)
) ON [Slow_Growing]

CREATE TABLE dimMethodOfDelivery
(keyMethodOfDelivery INT NOT NULL IDENTITY(1,1),
Name NVARCHAR(30) NOT NULL,
doer NVARCHAR(30) NOT NULL
CONSTRAINT PKDW_10 PRIMARY KEY (keyMethodOfDelivery)
) ON [Slow_Growing]

```

Таблица фактов «FactSales» будут секционированной, поэтому перед её созданием необходимо создать соответствующие функции и схемы секционирования.

```

CREATE PARTITION FUNCTION PartFunctionFactSales_Date (bigint)
AS RANGE RIGHT FOR VALUES (20130101,20140101,20150101,20160101)

CREATE PARTITION SCHEME PartSchFactSales_Date
AS PARTITION PartFunctionFactSales_Date TO
([Fast_Growing],
[Fast_Growing],
[Frequently_Requested],
[Frequently_Requested],
[Frequently_Requested])

CREATE PARTITION FUNCTION PartFunctionForArchivalTable (bigint)
AS RANGE RIGHT FOR VALUES (20130101)

CREATE PARTITION SCHEME PartSchForArchivalTable
AS PARTITION PartFunctionForArchivalTable TO
([Fast_Growing],
[Fast_Growing])
GO

```

Создание таблиц фактов.

```

CREATE TABLE FactSales
(NumberSale INT NOT NULL,
NumberLineInSale INT NOT NULL,
CustomerKey INT NOT NULL,
CustomerDemographyKey INT,
ProductKey INT NOT NULL,
DateKey BIGINT NOT NULL,
ManagerKey INT,
KindOfPayKey INT,
PromotionKey INT,
Quantity INT NOT NULL,

```

```

UnitPrice MONEY NOT NULL,
Subtotal MONEY NOT NULL,
DiscountCustAmt MONEY,
PromoAmt MONEY,
TaxAmt MONEY NOT NULL,
Freight MONEY,
Total MONEY NOT NULL,
CONSTRAINT PKDW_11 PRIMARY KEY (NumberSale,NumberLineInSale,DateKey)
WITH (ALLOW_ROW_LOCKS = OFF, ALLOW_PAGE_LOCKS = OFF)
) ON PartSchFactSales_Date(DateKey)

```

```

CREATE TABLE FactDelivery
(keyDelivery INT NOT NULL,
CustomerKey INT NOT NULL,
GeographyKey INT NOT NULL,
MethodOfDeliveryKey INT NOT NULL,
ExpectedDateOfDelivery BIGINT,
FactualDateOfDelivery BIGINT,
Cost MONEY,
KindOfPaykey INT,
CONSTRAINT PKDW_12 PRIMARY KEY (keyDelivery) WITH (ALLOW_ROW_LOCKS =
OFF, ALLOW_PAGE_LOCKS = OFF)
) ON [Fast_Growing]

```

Добавление ограничения внешнего ключа для связей между таблицами.

```

ALTER TABLE dimPromotion
ADD CONSTRAINT FK1 FOREIGN KEY (StartDate) REFERENCES dimDate(KeyDate)

```

```

ALTER TABLE dimPromotion
ADD CONSTRAINT FK2 FOREIGN KEY (EndDate) REFERENCES dimDate(KeyDate)

```

```

ALTER TABLE dimCustomer
ADD CONSTRAINT FK3 FOREIGN KEY (GeographyKey) REFERENCES
dimGeography(KeyGeography)

```

```

ALTER TABLE dimCustomer
ADD CONSTRAINT FK4 FOREIGN KEY (CustomerDemographyKey) REFERENCES
MinidimCustomerDemography(keyCustomerDemography)

```

```

ALTER TABLE dimProduct
ADD CONSTRAINT FK5 FOREIGN KEY (ManufacturingCountry) REFERENCES
dimGeography(KeyGeography)

```

```

ALTER TABLE dimOffices
ADD CONSTRAINT FK6 FOREIGN KEY (GeographyKey) REFERENCES
dimGeography(KeyGeography)

```

```

ALTER TABLE dimManagers

```

```
ADD CONSTRAINT FK7 FOREIGN KEY (OfficeKey) REFERENCES
dimOffices(keyOffices)
```

```
ALTER TABLE FactSales
ADD CONSTRAINT FK8 FOREIGN KEY (ProductKey) REFERENCES
dimProduct(keyProduct)
```

```
ALTER TABLE FactSales
ADD CONSTRAINT FK9 FOREIGN KEY (CustomerKey) REFERENCES
dimCustomer(keyCustomer)
```

```
ALTER TABLE FactSales
ADD CONSTRAINT FK10 FOREIGN KEY (CustomerDemographyKey) REFERENCES
MinidimCustomerDemography(keyCustomerDemography)
```

```
ALTER TABLE FactSales
ADD CONSTRAINT FK11 FOREIGN KEY (DateKey) REFERENCES DimDate(KeyDate)
```

```
ALTER TABLE FactSales
ADD CONSTRAINT FK12 FOREIGN KEY (KindOfPayKey) REFERENCES
dimKindOfPay(KeyKindOfPay)
```

```
ALTER TABLE FactSales
ADD CONSTRAINT FK13 FOREIGN KEY (PromotionKey) REFERENCES
dimPromotion(KeyPromotion)
```

```
ALTER TABLE FactSales
ADD CONSTRAINT FK14 FOREIGN KEY (ManagerKey) REFERENCES
dimManagers(KeyManager)
```

```
ALTER TABLE FactDelivery
ADD CONSTRAINT FK15 FOREIGN KEY (GeographyKey) REFERENCES
dimGeography(KeyGeography)
```

```
ALTER TABLE FactDelivery
ADD CONSTRAINT FK16 FOREIGN KEY (ExpectedDateOfDelivery) REFERENCES
DimDate(KeyDate)
```

```
ALTER TABLE FactDelivery
ADD CONSTRAINT FK17 FOREIGN KEY (FactualDateOfDelivery) REFERENCES
DimDate(KeyDate)
```

```
ALTER TABLE FactDelivery
ADD CONSTRAINT FK18 FOREIGN KEY (KindOfPayKey) REFERENCES
dimKindOfPay(keyKindOfPay)
```

```
ALTER TABLE FactDelivery
ADD CONSTRAINT FK19 FOREIGN KEY (MethodOfDeliveryKey) REFERENCES
DimMethodOfDelivery(KeyMethodOfDelivery)
```

```
ALTER TABLE FactDelivery
```

ADD CONSTRAINT FK20 FOREIGN KEY (CustomerKey) REFERENCES
dimCustomer(keyCustomer)

На рис. 62 представлена схема созданного хранилища данных.

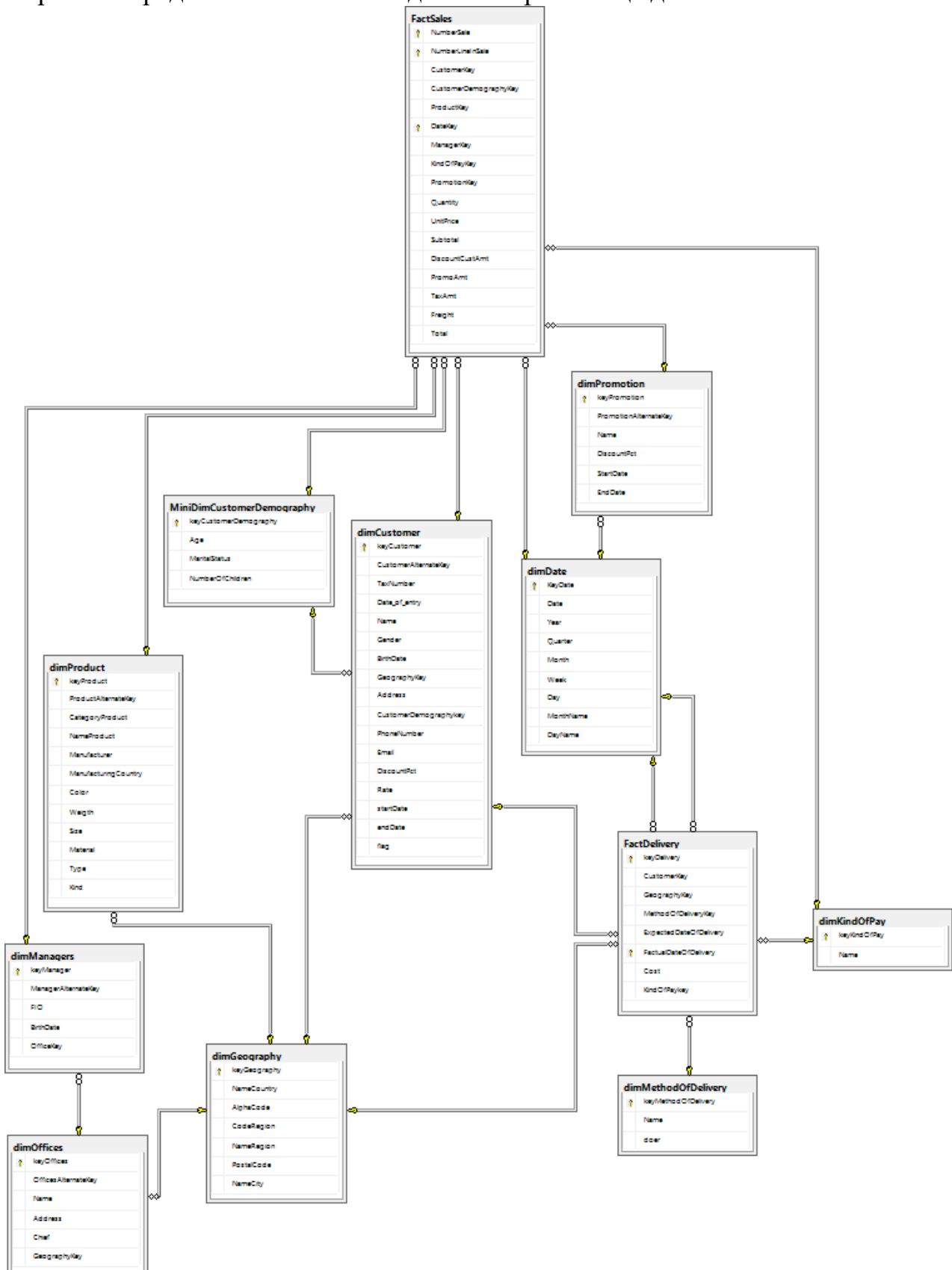


Рис. 62. Схема хранилища данных «MyStore»

Приложение 3

Создание индексов для хранилища данных.

```
CREATE INDEX NonCL_dimGeography_Country
ON [dbo].[dimGeography]([NameCountry]) INCLUDE ([NameRegion],[NameCity])
WITH (FILLFACTOR = 70);
```

```
CREATE INDEX NonCL_dimGeography_City
ON [dbo].[dimGeography]([NameCity])
WITH (FILLFACTOR = 70);
```

```
CREATE NONCLUSTERED INDEX NonCL_dimCustomer_AlternateKey
ON [dbo].[dimCustomer]([CustomerAlternateKey])
```

```
CREATE NONCLUSTERED INDEX NonCL_dimCustomer_GeographyKey
ON [dbo].[dimCustomer]([GeographyKey])
```

```
CREATE UNIQUE NONCLUSTERED INDEX NonCL_dimManagers_AlternateKey
ON [dbo].[dimManagers]([ManagerAlternateKey])
```

```
CREATE NONCLUSTERED INDEX NonCL_dimManagers_OfficeKey
ON [dbo].[dimManagers]([OfficeKey]) INCLUDE ([FIO],[BirthDate])
```

```
CREATE UNIQUE NONCLUSTERED INDEX NonCL_dimOffices_AlternateKey
ON [dbo].[dimOffices]([OfficesAlternateKey])
```

```
CREATE UNIQUE NONCLUSTERED INDEX NonCL_dimProduct_AlternateKey
ON [dbo].[dimProduct]([ProductAlternateKey])
```

```
CREATE NONCLUSTERED INDEX NonCL_dimProduct_CategoryProduct
ON [dbo].[dimProduct]([CategoryProduct])
```

```
CREATE UNIQUE NONCLUSTERED INDEX NonCL_dimPromotion_AlternateKey
ON [dbo].[dimPromotion]([PromotionAlternateKey])
```

```
CREATE NONCLUSTERED INDEX NonCL_FactSales_CustomerKey
ON [dbo].[FactSales]([CustomerKey])
```

```
CREATE NONCLUSTERED INDEX NonCL_FactSales_ManagerKey
ON [dbo].[FactSales]([ManagerKey])
```

```
CREATE NONCLUSTERED INDEX NonCL_FactSales_ProductKey
ON [dbo].[FactSales]([ProductKey])
```

```
CREATE NONCLUSTERED INDEX NonCL_FactSales_CustomerDemographyKey
ON [dbo].[FactSales]([CustomerDemographyKey])
```

```
CREATE NONCLUSTERED INDEX NonCL_FactSales_KindOfPayKey
ON [dbo].[FactSales]([KindOfPayKey])
```

```

CREATE NONCLUSTERED COLUMNSTORE INDEX ClSt_FactSales
ON
[dbo].[FactSales]([DateKey],[CustomerKey],[CustomerDemographyKey],[ProductKey],[ManagerKey],[KindOfPayKey],[PromotionKey],[Quantity],[UnitPrice],[Subtotal],[DiscountCustAmt],[PromoAmt],[TaxAmt],[Freight],[Total])

CREATE NONCLUSTERED INDEX NonCL_FactDelivery_CustomerKey
ON [dbo].[FactDelivery]([CustomerKey])

CREATE NONCLUSTERED INDEX NonCL_FactDelivery_GeographyKey
ON [dbo].[FactDelivery]([GeographyKey])

CREATE NONCLUSTERED INDEX NonCL_FactDelivery_MethodOfDeliveryKey
ON [dbo].[FactDelivery]([MethodOfDeliveryKey])

CREATE NONCLUSTERED INDEX NonCL_FactDelivery_Filter
ON [dbo].[FactDelivery]([FactualDateOfDelivery]) INCLUDE
([CustomerKey],[GeographyKey],[MethodOfDeliveryKey],[Cost])
WHERE [FactualDateOfDelivery] >='20130101' AND
[FactualDateOfDelivery]<='20160101'

```

Приложение 4

Код для заполнения данными таблицы измерения «dimDate».

```
SET LANGUAGE Russian
```

```
declare @startday date = '2005-01-01'  
declare @endday date = '2035-01-01'  
declare @inc date  
set @inc = @startday  
  
while @inc<= @endday  
begin  
insert into [MyStore].[dbo].[DimDate] values  
(cast (convert (varchar(8),@inc, 112) as bigint),  
convert(date, @inc, 104),  
year(@inc),  
datepart(qq, @inc),  
month(@inc),  
datepart(wk,@inc),  
day(@inc),  
datename(mm,@inc),  
datename(dw,@inc)  
)  
set @inc = dateadd(day,1,@inc)  
end
```