

EPAM Systems, RD Dep.

Работа с данными на базе MS SQL Server и T-SQL

REVISION HISTORY					
Ver.	Description of Change	Author	Date	Approved	
				Name	Effective Date
<1.0>	Первая версия	Беляков Павел	<21.08.2012>		

Legal Notice

This document contains privileged and/or confidential information and may not be disclosed, distributed or reproduced without the prior written permission of EPAM Systems.

СОДЕРЖАНИЕ

1. ОБЗОР MS SQL SERVER 2008	3
2. МАНИПУЛИРОВАНИЕ ДАННЫМИ И ОБЪЕКТАМИ	8
3. ЗАПРОС И ФИЛЬТРАЦИЯ ДАННЫХ	11
4. ГРУППИРОВАНИЕ И АГРЕГИРОВАНИЕ	16
5. ПРИСОЕДИНЕНИЕ ТАБЛИЦ И ВЛОЖЕННЫЕ ЗАПРОСЫ	20
6. ИЗМЕНЕНИЕ ДАННЫХ В ТАБЛИЦАХ	31
7. ОБЪЕКТЫ ДЛЯ РАБОТЫ С ДАННЫМИ	33

1. Обзор MS SQL Server 2008

Microsoft® SQL Server™ — это система анализа и управления реляционными базами данных в решениях электронной коммерции, производственных отраслей и хранилищ данных. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

SQL Server предоставляет набор функций и средств, которые можно использовать для разработки баз данных и решений, а также для управления ими:

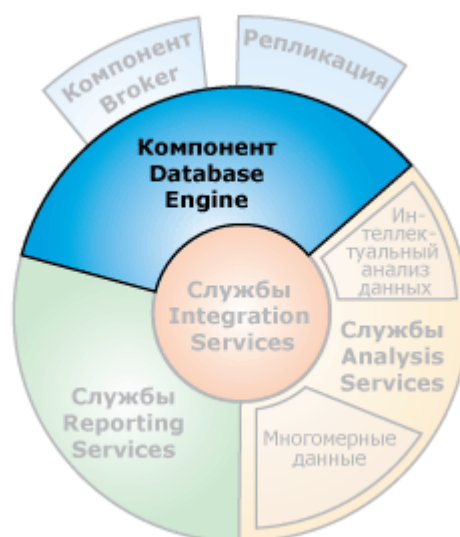


Рисунок 1 – Службы MS SQL Server

Компонент **Database Engine** представляет собой основную службу для хранения, обработки и обеспечения безопасности данных. Этот компонент обеспечивает управляемый доступ к ресурсам и быструю обработку транзакций, что позволяет использовать его даже в самых требовательных корпоративных приложениях обработки данных. Кроме того, компонент Database Engine предоставляет разносторонние средства поддержания высокого уровня доступности.

Службы **Analysis Services** поддерживают обработку OLAP, позволяя проектировать и создавать многомерные структуры, которые содержат данные, собранные из других источников данных (например, реляционных баз данных), и управлять этими многомерными структурами.

Благодаря службам **Analysis Services** можно проектировать, создавать и визуализировать модели интеллектуального анализа данных. Разнообразие стандартных алгоритмов интеллектуального анализа данных позволяет создавать такие модели на основе других источников данных.

Службы **Integration Services** представляют собой платформу для создания высокопроизводительных решений по интеграции данных, в том числе пакетов для хранения данных, обеспечивающих извлечение, преобразование и загрузку данных.

Службы **Master Data Services** являются источником основных данных организации. Интеграция разрозненных операционных и аналитических систем со

службами Master Data Services обеспечивает централизованный, точный источник данных для всех приложений организации. С помощью служб Master Data Services создается единый источник основных данных, а также поддерживается доступная для аудита запись изменений данных со временем.

Репликация представляет собой набор технологий, с помощью которых данные или объекты базы данных можно скопировать и перенести из одной базы данных в другую, а затем синхронизировать эти базы данных для обеспечения согласованности. Благодаря репликации данные можно размещать в различных местах, обеспечивая возможность доступа к ним удаленных и мобильных пользователей по локальным или глобальным сетям, посредством коммутируемых и беспроводных соединений, а также через Интернет.

Службы **Reporting Services** предлагают средства создания корпоративных отчетов с поддержкой веб-интерфейса, которые позволяют включать в отчеты данные из различных источников, публиковать отчеты в разнообразных форматах, а также централизованно управлять безопасностью и подписками.

SQL Server 2008 R2 представляет новые возможности встроенной бизнес-аналитики и средства интеграции с продуктами и технологиями **SharePoint**. В этом выпуске и службы Analysis Services, и службы Reporting Services поддерживают развертывание на ферме SharePoint.

Компонент **Service Broker** призван помочь разработчикам в создании безопасных масштабируемых приложений баз данных. Это новая технология компонента Database Engine предоставляет платформу для взаимодействия на основе обмена сообщениями, благодаря которой независимые компоненты приложений могут действовать как единое целое. В компонент Service Broker включена инфраструктура асинхронного программирования, которая может использоваться как приложениями в пределах одной базы данных или экземпляра, так и распределенными приложениями.

Безопасность MS SQL Server

SQL Server включает различные методы и средства настройки безопасности пользователей, служб и других учетных записей для доступа к системе. Сосредоточим внимание на работе с участниками (пользователями и учетными записями входа), ролями (группами участников), защищаемыми объектами и разрешениями.

Участники (Principals)

Участники — сущности, которые могут запрашивать ресурсы SQL Server. Как и другие компоненты модели авторизации SQL Server, участников можно иерархически упорядочить. Область влияния участника зависит от области его определения: Windows, сервер, база данных, — а также от того, коллективный это участник или индивидуальный. Имя входа Windows является примером индивидуального (неделимого) участника, а группа Windows — коллективного. Каждый из участников имеет идентификатор безопасности (SID).

Участники уровня Windows:

- Имя входа домена Windows
- Локальное имя входа Windows

Участники уровня SQL Server:

- Имя входа SQL Server

Участники уровня базы данных

- Пользователь базы данных
- Роль базы данных
- Роль приложения

Роли

Для удобства управления разрешениями на сервере SQL Server предоставляет несколько *ролей*, которые являются субъектами безопасности, группирующими других участников. *Роли* похожи на *группы* в операционной системе Microsoft Windows.

Предопределенные роли сервера предоставляются для удобства пользователей и обеспечения обратной совместимости. При возможности назначайте более специализированные разрешения.

Роли уровня сервера также называются *предопределенными ролями сервера*, поскольку пользователь не может создать новые роли уровня сервера. Разрешения ролей уровня сервера распространяются на весь сервер.

В роли уровня сервера можно добавлять имена входа SQL Server, учетные записи Windows и группы Windows. Каждый член предопределенной роли сервера может добавлять другие имена входа к той же роли.

В следующей таблице представлены роли уровня сервера и их возможности.

Имя роли на уровне сервера	Описание
sysadmin	Члены предопределенной роли сервера sysadmin могут выполнять любые действия на сервере.
serveradmin	Члены предопределенной роли сервера serveradmin могут изменять параметры конфигурации на уровне сервера, а также выключать сервер.
securityadmin	<p>Члены предопределенной роли сервера securityadmin управляют именами входа и их свойствами. Они могут предоставлять, запрещать и отменять разрешения на уровне сервера (инструкции GRANT, DENY и REVOKE). Они также могут предоставлять, запрещать и отменять разрешения на уровне базы данных (инструкции GRANT, DENY и REVOKE) при наличии доступа к базе данных. Кроме того, они могут сбрасывать пароли для имен входа SQL Server.</p> <p>Примечание по безопасности</p> <p>Возможность предоставления доступа к компоненту Database Engine и настройки разрешений пользователя позволяет администраторам безопасности назначать большинство разрешений сервера. Роль securityadmin должна рассматриваться как эквивалент роли sysadmin.</p>
processadmin	Члены предопределенной роли сервера processadmin могут завершать

	процессы, выполняемые на экземпляре SQL Server.
setupadmin	Члены предопределенной роли сервера setupadmin могут добавлять или удалять связанные серверы.
bulkadmin	Члены предопределенной роли сервера bulkadmin могут выполнять инструкцию BULK INSERT.
diskadmin	Предопределенная роль сервера diskadmin используется для управления файлами на диске.
dbcreator	Члены предопределенной роли сервера dbcreator могут создавать, изменять, удалять и восстанавливать любые базы данных.
public	Каждое имя входа SQL Server принадлежит к роли сервера public. Если для участника на уровне сервера не были предоставлены или запрещены конкретные разрешения на защищаемый объект, он наследует разрешения роли public на этот объект. Разрешения роли public следует назначать только тому объекту, который будет доступен всем пользователям.

Пользователь базы данных является участником уровня базы данных. Каждый пользователь базы данных является членом роли **public**.

По умолчанию при создании база данных включает в себя пользователя **guest**. Разрешения, предоставленные пользователю **guest**, наследуются пользователями, которые не имеют учетной записи пользователя в базе данных.

Пользователя **guest** нельзя удалить, но его можно отключить, если отменить его разрешение CONNECT. Разрешение CONNECT можно отменить, выполнив инструкцию REVOKE CONNECT FROM GUEST в любой базе данных, кроме **master** или **tempdb**.

Роли уровня базы данных

Для удобства управления разрешениями в базах данных SQL Server предоставляет несколько *ролей*, которые являются субъектами безопасности, группирующими других участников. Разрешения ролей уровня базы данных распространяются на всю базу данных.

В SQL Server существует два типа ролей уровня базы данных: *предопределенные роли базы данных*, стандартные для базы данных, и *гибкие роли базы данных*, которые может создать пользователь.

Предопределенные роли базы данных задаются на уровне базы данных и предусмотрены в каждой базе данных. Члены ролей базы данных **db_owner** и **db_securityadmin** могут управлять членством в предопределенных ролях базы данных. Но только члены роли базы данных **db_owner** могут добавлять членов в предопределенную роль базы данных **db_owner**. Кроме того, в базе данных msdb имеются специальные предопределенные роли базы данных.

В роли уровня базы данных можно добавить любую учетную запись базы данных и другие роли SQL Server. Каждый член предопределенной роли базы данных может добавлять другие имена входа к той же роли.

В следующей таблице представлены предопределенные роли уровня базы данных и их возможности. Эти роли существуют во всех базах данных.

Имя роли уровня базы данных	Описание
db_owner	Члены предопределенной роли базы данных db_owner могут выполнять все действия по настройке и обслуживанию базы данных, а также удалять базу данных.
db_securityadmin	Элементы предопределенной роли базы данных db_securityadmin могут изменять членство в роли и управлять разрешениями. Добавление участников к этой роли может привести к непреднамеренному повышению прав доступа.
db_accessadmin	Члены предопределенной роли базы данных db_accessadmin могут добавлять или удалять права удаленного доступа к базе данных для имен входа и групп Windows, а также имен входа SQL Server.
db_backupoperator	Члены предопределенной роли базы данных db_backupoperator могут создавать резервные копии базы данных.
db_ddladmin	Члены предопределенной роли базы данных db_ddladmin могут выполнять любые команды языка определения данных (DDL) в базе данных.
db_datawriter	Члены предопределенной роли базы данных db_datawriter могут добавлять, удалять или изменять данные во всех пользовательских таблицах.
db_datareader	Элементы предопределенной роли базы данных db_datareader могут считывать все данные из всех пользовательских таблиц.
db_denydatawriter	Члены предопределенной роли базы данных db_denydatawriter не могут добавлять, изменять или удалять данные в пользовательских таблицах базы данных.
db_denydatareader	Члены предопределенной роли базы данных db_denydatareader не могут считывать данные из пользовательских таблиц базы данных.

2. Манипулирование данными и объектами

В SQL Server 2008 R2 существует ряд объектов БД:

Объект	Описание
Table	Вся информация в SQL-сервере хранится в виде таблиц
View	Можно представить как «виртуальную» таблицу, т.е. некоторую выборку из реальных таблиц при помощи заранее определенных запросов
Indexes	Ускоряют операции доступа к данным, генерируются на основе одного или нескольких полей
Triggers	Позволяют выполнять SQL-код при выполнении операций добавления/удаления/обновления данных в определенные таблицы
Stored procedures	Выполнение заранее откомпилированного SQL-кода на сервере, возможна передача входных и выходных параметров, возвращение набора данных
Constraints	Препятствуют вставке противоречивых данных
Rules	Задают значения, которые можно вставлять в базу

Язык описания данных DDL — это словарь, используемый для определения структур данных в SQL Server 2008 R2. Эти инструкции используются для создания, изменения и удаления структур данных в экземпляре SQL Server.

CREATE

Инструкции CREATE используются для определения новых сущностей. Например, при помощи инструкции CREATE TABLE можно добавить новую таблицу в базу данных.

Пример создания базы данных:

```
USE master;
GO
CREATE DATABASE Sales
ON
( NAME = Sales_dat,
  FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL10_50\MSSQLSERVER\MSSQL\DATA\saledat.mdf',
```



```
    SIZE = 10,  
    MAXSIZE = 50,  
    FILEGROWTH = 5 )  
LOG ON  
( NAME = Sales_log,  
  FILENAME = 'C:\Program Files\Microsoft SQL  
Server\MSSQL10_50\MSSQLSERVER\MSSQL\DATA\salelog.ldf',  
  SIZE = 5MB,  
  MAXSIZE = 25MB,  
  FILEGROWTH = 5MB ) ;  
GO
```

Пример создания логина:

```
CREATE LOGIN [<domainName>\<loginName>] FROM WINDOWS;  
GO
```

ALTER

Инструкции ALTER используются для изменения определений существующих сущностей. Например, с помощью инструкции ALTER TABLE можно добавить новый столбец в таблицу, а инструкция ALTER DATABASE позволяет задавать параметры базы данных.

Пример изменения имени пользователя:

```
USE AdventureWorks2008R2;  
ALTER USER Mary5 WITH NAME = Mary51;  
GO
```

Пример удаления колонки из таблицы:

```
CREATE TABLE dbo.doc_exb (column_a INT, column_b VARCHAR(20) NULL) ;  
GO  
ALTER TABLE dbo.doc_exb DROP COLUMN column_b ;  
GO  
EXEC sp_help doc_exb ;  
GO  
DROP TABLE dbo.doc_exb ;  
GO
```

DROP

Инструкции DROP используются для удаления существующих сущностей. Например, при помощи инструкции DROP TABLE можно удалить таблицу из базы данных.

Пример удаления триггера:

```
USE AdventureWorks2008R2;  
GO  
IF OBJECT_ID ('employee_insupd', 'TR') IS NOT NULL  
  DROP TRIGGER employee_insupd;
```

GO

Так же есть ряд других DML операторов:

ENABLE TRIGGER - Включает триггер DML, DDL или logon.

DISABLE TRIGGER - Отключает триггер.

TRUNCATE - Удаляет все строки в таблице, не записывая в журнал удаление отдельных строк. Инструкция TRUNCATE TABLE похожа на инструкцию DELETE без предложения WHERE, однако TRUNCATE TABLE выполняется быстрее и требует меньших ресурсов системы и журналов транзакций.

3. Запрос и фильтрация данных

Язык Transact-SQL является ключом к использованию SQL Server. Все приложения, взаимодействующие с экземпляром SQL Server, независимо от их пользовательского интерфейса отправляют серверу инструкции Transact-SQL.

SELECT

Возвращает строки из базы данных и позволяет делать выборку одной или нескольких строк или столбцов из одной или нескольких таблиц в SQL Server 2008 R2. Полный синтаксис инструкции SELECT сложен, однако основные предложения можно вкратце описать следующим образом:

```
[ WITH <common_table_expression> ]  
SELECT select_list [ INTO new_table ]  
[ FROM table_source ] [ WHERE search_condition ]  
[ GROUP BY group_by_expression ]  
[ HAVING search_condition ]  
[ ORDER BY order_expression [ ASC | DESC ] ]
```

Пример выборки всех записей:

```
SELECT ProductID, ProductName, Price, ProductDescription  
FROM dbo.Products  
GO
```

WHERE

В большинстве случаев требуется выбирать записи, удовлетворяющие некоторому условию. Для этого используется предложение WHERE - указывает условие поиска для строк, возвращаемых инструкцией DELETE, MERGE, SELECT или UPDATE в SQL Server 2008 R2. Это предложение используется для ограничения числа строк, возвращаемых или обрабатываемых инструкцией.

```
[ WHERE <search_condition> ]
```

где <search_condition> - условие выборки, сочетание одного или нескольких предикатов, в котором используются логические операторы AND, OR и NOT. Параметр <условие_поиска> в инструкции DELETE, MERGE, SELECT или UPDATE указывает, сколько строк возвращается или обрабатывается инструкцией.

```
<search_condition> ::=  
    { [ NOT ] <predicate> | ( <search_condition> ) }  
    [ { AND | OR } [ NOT ] { <predicate> | ( <search_condition> ) } ]  
    [ ,...n ]  
  
<predicate> ::=  
    { expression { = | < > | != | > | > = | ! > | < | < = | ! < } expression  
    | string_expression [ NOT ] LIKE string_expression  
    [ ESCAPE 'escape_character' ]  
    | expression [ NOT ] BETWEEN expression AND expression  
    | expression IS [ NOT ] NULL  
    | CONTAINS  
    ( { column | * } , '<contains_search_condition>'  
    | FREETEXT ( { column | * } , 'freetext_string')
```

```
| expression [ NOT ] IN (subquery | expression [ ,...n ] )  
| expression { = | < > | ! = | > | > = | ! > | < | < = | ! < }  
{ ALL | SOME | ANY } (subquery)  
| EXISTS (subquery) }
```

NOT

Инвертирует логическое выражение, задаваемое предикатом.

AND

Объединяет два условия и выдает значение TRUE, если оба условия имеют значение TRUE.

OR

Объединяет два условия и выдает значение TRUE, если хотя бы одно условие имеет значение TRUE.

< predicate >

Выражение, возвращающее значения TRUE, FALSE или UNKNOWN.

expression

Может являться именем столбца, константой, функцией, переменной, скалярным вложенным запросом или любым сочетанием имен столбцов, констант и функций, связанных операторами или вложенным запросом. Также может содержать выражение CASE.

=

Оператор, используемый для проверки равенства двух выражений.

<>

Оператор, используемый для проверки условий неравенства условий двух выражений.

!=

Оператор, используемый для проверки условий неравенства условий двух выражений.

>

Оператор, используемый для проверки превышения одного выражения над условием другого.

>=

Оператор, используемый для проверки превышения либо равенства двух выражений.

!>

Оператор, используемый для проверки того, что одно выражение не превышает другое выражение.

<

Оператор, используемый для проверки того, что одно выражение меньше другого.

<=

Оператор, используемый для проверки того, что одно выражение меньше другого или равно ему.

!<

Оператор, используемый для проверки того, что одно выражение не меньше другого.

string_expression

Строка обычных символов и символов-шаблонов.

[NOT] LIKE

Показывает, что последующая строка символов будет использоваться при

совпадении с шаблоном.

ESCAPE 'escape_character'

Позволяет найти сам символ-шаблон в строке (вместо того, чтобы использовать его как шаблон). Аргумент *escape_character* — это символ, который нужно поместить перед символом-шаблоном, чтобы указать такое специальное использование.

[NOT] BETWEEN

Задаёт включающий диапазон значений. Используйте оператор AND для разделения начальных и конечных значений.

IS [NOT] NULL

Задаёт поиск значений NULL или значений, не являющихся значениями NULL, в зависимости от используемых ключевых слов. При обращении одного из операндов выражения с битовыми или арифметическими операторами в значение NULL указанное выражение также обращается в значение NULL.

CONTAINS

Осуществляет поиск столбцов, содержащих символьные данные с заданной точностью (*fuzzy*), соответствующие заданным отдельным словам и фразам на основе схожести словам и точному расстоянию между словами, взвешенному совпадению. Этот параметр может быть использован только в инструкции SELECT.

FREETEXT

Предоставляет простую форму естественного языка ввода запросов на осуществление поиска столбцов, содержащих символьные данные, совпадающие с содержанием предиката не точно, а по смыслу. Этот параметр может быть использован только в инструкции SELECT.

[NOT] IN

Задаёт поиск выражения, основанного на выражении, включённого или исключённого из списка. Выражение поиска может быть константой или именем столбца, а списком может быть набор констант или, что чаще, вложенный запрос. Список значений необходимо заключать в скобки.

subquery

Может рассматриваться как ограниченная инструкция SELECT и являющаяся подобной на *<query_expression>* в инструкции SELECT. Использование предложений ORDER BY, COMPUTE и ключевого слова INTO не допускается.

ALL

Используется с оператором сравнения и вложенным запросом. Возвращает для *<предиката>* значение TRUE, если все получаемые для вложенного запроса значения удовлетворяют условию, и значение FALSE, если не все значения удовлетворяют условию или в случае, когда в результате выполнения вложенного запроса внешней инструкции не выдается ни одной строки.

{ SOME | ANY }

Используется с оператором сравнения и вложенным запросом. Возвращает для *<предиката>* значение TRUE, если хотя бы одно получаемое для вложенного запроса значение удовлетворяет условию, и значение FALSE, если ни одно из значений не удовлетворяет условию или в случае, когда в результате выполнения вложенного запроса внешней инструкции не выдается ни одной строки. В противном случае результатом выражения является значение UNKNOWN.

EXISTS

Используется во вложенном запросе для проверки существования строк, возвращенных вложенным запросом.

CASE

Оценка списка условий и возвращение одного из нескольких возможных выражений результатов.

Выражение CASE имеет два формата:

- простое выражение CASE для определения результата сравнивает выражение с набором простых выражений;
- поисковое выражение CASE для определения результата вычисляет набор логических выражений.

Оба формата поддерживают дополнительный аргумент ELSE.

Выражение CASE может использоваться в любой инструкции или предложении, которые допускают допустимые выражения. Например, выражение CASE можно использовать в таких инструкциях, как SELECT, UPDATE, DELETE и SET, а также в таких предложениях, как select_list, IN, WHERE, ORDER BY и HAVING.

Пример:

```
SELECT ProductNumber, Category =  
CASE ProductLine  
WHEN 'R' THEN 'Road'  
WHEN 'M' THEN 'Mountain'  
WHEN 'T' THEN 'Touring'  
WHEN 'S' THEN 'Other sale items'  
ELSE 'Not for sale'  
END,  
Name  
FROM Production.Product
```

ORDER BY

Сортирует данные, возвращаемые запросом в SQL Server 2012. Используйте это предложение для выполнения следующих задач:

- Упорядочение результирующего набора запроса по заданному списку столбцов и (дополнительно) ограничение числа возвращаемых строк указанным диапазоном. Порядок, в котором строки возвращаются в результирующем наборе, не гарантируется, если не указано предложение ORDER BY.
- Определение порядка, в котором значения ранжирующей функции применяются к результирующему набору.

```
ORDER BY order_by_expression  
[ COLLATE collation_name ]  
[ ASC | DESC ]  
[ ,...n ]  
[ <offset_fetch> ]  
  
<offset_fetch> ::=  
{  
    OFFSET { integer_constant | offset_row_count_expression } { ROW | ROWS }  
    [  
        FETCH { FIRST | NEXT } {integer_constant | fetch_row_count_expression } {  
ROW | ROWS } ONLY  
    ]  
}
```

Пример:

```
SELECT ProductID, Name FROM Production.Product
WHERE Name LIKE 'Lock Washer%'
ORDER BY ProductID;
```

Сортирование с условием:

```
SELECT BusinessEntityID, LastName, TerritoryName, CountryRegionName
FROM Sales.vSalesPerson
WHERE TerritoryName IS NOT NULL
ORDER BY CASE CountryRegionName WHEN 'United States' THEN TerritoryName
        ELSE CountryRegionName END;
```

4. Группирование и агрегирование

SQL Server содержит множество встроенных функций, а также поддерживает создание определяемых пользователем функций:

Функция	Описание
Функции, возвращающие наборы строк.	Возвращают объект, который можно использовать так же, как табличные ссылки в SQL-инструкции.
Агрегатные функции	Обрабатывают коллекцию значений и возвращают одно результирующее значение.
Ранжирующие функции	Возвращают ранжирующее значение для каждой строки в секции.
Скалярные функции	Обрабатывают и возвращают одиночное значение. Скалярные функции можно применять везде, где выражение допустимо.

Агрегатные функции - выполняют вычисление на наборе значений и возвращают одиночное значение. Агрегатные функции часто используются в выражении GROUP BY инструкции SELECT.

Все агрегатные функции являются детерминированными. Это означает, что агрегатные функции возвращают одну и ту же величину при каждом их вызове на одном и том же наборе входных значений.

Агрегатные функции могут быть использованы в качестве выражений только в следующих случаях.

- Список выбора инструкции SELECT (вложенный или внешний запрос).
- Предложение HAVING.

Transact-SQL предоставляет следующие агрегатные функции:

SUM

Возвращает сумму всех, либо только уникальных, значений в выражении. Функция SUM может быть использована только для числовых столбцов. Значения NULL не учитываются.

```
SELECT Color, SUM(ListPrice), SUM(StandardCost)
FROM Production.Product
WHERE Color IS NOT NULL
      AND ListPrice != 0.00
      AND Name LIKE 'Mountain%'
```

AVG

Возвращает среднее арифметическое группы значений. Значения NULL не учитываются.

```
SELECT AVG(VacationHours) AS 'Average vacation hours',
```



```
SUM(SickLeaveHours) AS 'Total sick leave hours'  
FROM HumanResources.Employee
```

MIN, MAX

Возвращают минимальное и максимальное значение выражения, соответственно.

```
SELECT MAX(TaxRate)  
FROM Sales.SalesTaxRate;
```

COUNT

Возвращает количество элементов в группе. Функция COUNT работает аналогично функции COUNT_BIG. Единственное различие между двумя функциями — возвращаемые значения. Функция COUNT всегда возвращает значение типа **int**. Функция COUNT_BIG всегда возвращает значение типа данных **bigint**.

```
SELECT COUNT(DISTINCT Title)  
FROM HumanResources.Employee;
```

UNION

Объединяет результаты двух или более запросов в один результирующий набор, в который входят все строки, принадлежащие всем запросам в объединении. Операция UNION отличается от соединений столбцов из двух таблиц.

Ниже приведены основные правила объединения результирующих наборов двух запросов с помощью операции UNION:

- Количество и порядок столбцов должны быть одинаковыми во всех запросах.
- Типы данных должны быть совместимыми.

Простой UNION:

```
SELECT ProductModelID, Name  
FROM Production.ProductModel  
WHERE ProductModelID NOT IN (3, 4)  
UNION  
SELECT ProductModelID, Name  
FROM dbo.Gloves  
ORDER BY Name;
```

Объединять можно данные из разных таблиц:

```
SELECT CityID, CityName FROM Cities WHERE CountryID = 2  
UNION  
SELECT DepartmentID, DepartmentName FROM Departments  
ORDER BY 2 DESC
```

В результате увидим:

CityID	CityName
8	Volgograd
3	Store
18	Store
22	Store
6	St. Piterburg
8	RD

Рисунок 2 – Результат работы некорректного UNION

GROUP BY

Группирует выбранный набор строк для получения набора сводных строк по значениям одного или нескольких столбцов или выражений в SQL Server. Возвращается одна строка для каждой группы. Агрегатные функции в списке <select> предложения SELECT предоставляют информацию о каждой группе, а не об отдельных строках.

Пример группирования:

```
SELECT dp.DepartmentName, ct.CityName, COUNT(dp.DepartmentName)
FROM Departments as dp
JOIN Cities AS ct ON ct.CityID = dp.CityID
GROUP BY dp.DepartmentName, ct.CityName
```

В результате сможем увидеть сколько и каких офисов в каждом городе:

5	HR	Grodno	1
6	Office	Grodno	1
7	Office	Kiev	2
8	B Store	Lvov	1
9	Product Store	Lvov	1
10	HR	Minsk	1

Рисунок 3 – результат группирования

Совместно с GROUP BY может использоваться HAVING, чтобы указать, какая из групп, сформированная в предложении GROUP BY, должна быть включена в результирующий набор:

```
SELECT DATEPART(yyyy,OrderDate) AS N'Year',SUM(TotalDue) AS N'Total Order Amount'  
FROM Sales.SalesOrderHeader  
GROUP BY DATEPART(yyyy,OrderDate)  
HAVING DATEPART(yyyy,OrderDate) >= N'2003'  
ORDER BY DATEPART(yyyy,OrderDate);
```

5. Присоединение таблиц и вложенные запросы

JOINS

С помощью соединения можно получать данные из двух или нескольких таблиц на основе логических связей между ними. Соединения указывают, как Microsoft SQL Server должен использовать данные из одной таблицы для выбора строк из другой таблицы.

Соединение определяет способ связывания двух таблиц в запросе следующим образом:

- для каждой таблицы указываются столбцы, используемые в соединении. В типичном условии соединения указывается внешний ключ из одной таблицы и связанный с ним ключ из другой таблицы;
- указывается логический оператор (например, = или <>,) для сравнения значений столбцов.

Внутреннее соединение можно задавать в предложениях FROM и WHERE. Внешнее соединение можно указывать только в предложении FROM. Условия соединения сочетаются с условиями поиска WHERE и HAVING для управления строками, выбранными из базовых таблиц, на которые ссылается предложение FROM.

То, что условия соединения задаются в предложении FROM, помогает отделить их от условий поиска, которые могут быть заданы в предложении WHERE. Объединение рекомендуется задавать именно таким способом. Ниже приведен упрощенный синтаксис соединения с использованием предложения FROM стандарта ISO:

```
FROM first_table join_type second_table [ON (join_condition)]
```

Аргумент *join_type* задает тип соединения: внутренний, внешний или перекрестный. Аргумент *join_condition* определяет предикат, который будет вычисляться для каждой пары соединяемых строк. Ниже приведен пример предложения FROM с заданным соединением:

```
SELECT ProductID, Purchasing.Vendor.BusinessEntityID, Name
FROM Purchasing.ProductVendor JOIN Purchasing.Vendor
    ON (Purchasing.ProductVendor.BusinessEntityID =
Purchasing.Vendor.BusinessEntityID)
WHERE StandardPrice > $10
```

Если один запрос содержит ссылки на несколько таблиц, то все ссылки столбцов должны быть однозначными. В предыдущем примере как таблица ProductVendor, так и таблица Vendor содержат столбец с именем BusinessEntityID. Имена столбцов, совпадающие в двух или более таблицах, на которые ссылается запрос, должны уточняться именем таблицы. Все ссылки на столбец Vendor в этом примере являются уточненными.

Если имя столбца не дублируется в двух или более таблицах, указанных в запросе, то ссылки на него уточнять именем таблицы не обязательно. Это показано в предыдущем примере. Подобную инструкцию SELECT иногда трудно понять, поскольку в ней нет ничего, что указывало бы на таблицы, из которых берутся столбцы. Запрос гораздо легче читать, если все столбцы указаны с именами

соответствующих таблиц. Запрос будет читаться еще легче, если используются псевдонимы таблиц, особенно когда имена таблиц сами должны уточняться именами базы данных и владельца. Ниже приведен тот же пример, но чтобы упростить чтение, используются псевдонимы таблиц, уточняющие названия столбцов.

```
SELECT pv.ProductID, v.BusinessEntityID, v.Name
FROM Purchasing.ProductVendor AS pv
JOIN Purchasing.Vendor AS v
    ON (pv.BusinessEntityID = v.BusinessEntityID)
WHERE StandardPrice > $10
```

В предыдущем примере условие соединения задается в предложении FROM, что является рекомендуемым способом. В следующем запросе это же условие соединения указывается в предложении WHERE:

```
SELECT pv.ProductID, v.BusinessEntityID, v.Name
FROM Purchasing.ProductVendor AS pv, Purchasing.Vendor AS v
WHERE pv.VendorID = v.VendorID
    AND StandardPrice > $10
```

Список выборки для соединения может ссылаться на все столбцы в соединяемых таблицах или на любое подмножество этих столбцов. Список выборки не обязательно должен содержать столбцы из каждой таблицы в соединении. Например, в соединении из трех таблиц связующим звеном между одной из таблиц и третьей таблицей может быть только одна таблица, при этом список выборки не обязательно должен ссылаться на столбцы средней таблицы.

Хотя обычно в условиях соединения для сравнения используется оператор равенства (=), можно указать другие операторы сравнения или реляционные операторы, равно как другие предикаты.

При обработке соединений в SQL Server механизм запросов выбирает наиболее эффективный метод обработки из нескольких возможных. При физическом выполнении различных соединений можно использовать много разных оптимизаций, поэтому их нельзя надежно прогнозировать.

Столбцы, используемые в условии соединения, не обязательно должны иметь одинаковые имена или одинаковый тип данных. Однако если типы данных не совпадают, то они должны быть совместимыми или допускать в SQL Server неявное преобразование. Если типы данных не допускают неявное преобразование, то условия соединения должны явно преобразовывать эти типы данных с помощью функции CAST.

Условия соединения можно задать в предложении FROM или WHERE. Рекомендуется указывать их в предложении FROM. Предложения WHERE и HAVING могут также содержать условия поиска для дальнейшей фильтрации строк, выбранных этими условиями соединения.

Соединения можно разделить на следующие категории.

- Внутренние соединения (типичные операции соединения, использующие такие операторы сравнения, как = или <>). Они включают эквивалентные соединения и естественные соединения.

Внутренние соединения

Внутренние соединения используют оператор сравнения для установки соответствия строк из двух таблиц на основе значений общих столбцов в каждой таблице. Примером может быть получение всех строк, в которых идентификационный номер студента одинаковый как в таблице students, так и в таблице courses.

Внутреннее соединение — это такое соединение, в котором значения в соединяемых столбцах подвергаются сравнению с использованием оператора сравнения.

В стандарте SQL внутренние соединения задаются предложением FROM или предложением WHERE. Это единственный тип соединения, допустимый в стандарте SQL для предложения WHERE. Внутренние соединения, задаваемые предложением WHERE, известны как внутренние соединения старого типа.

Следующий запрос на языке Transact-SQL является примером внутреннего соединения:

```
SELECT Empl.Name, Depts.Name from Empl
JOIN Depts
ON Empl.DepartmentID = Depts.ID
```

	Name	Name
1	Mike	Production
2	John	TC
3	Paul	Maintanence
4	Jessie	Production

Рисунок 4 – Результат inner join

Внешние соединения.

Внешние соединения бывают левыми, правыми и полными.

Если внешние соединения задаются в предложении FROM, они указываются с одним из следующих наборов ключевых слов.

LEFT JOIN или LEFT OUTER JOIN

Результирующий набор левого внешнего соединения включает все строки из левой таблицы, заданной в предложении LEFT OUTER, а не только те, в которых соединяемые столбцы соответствуют друг другу. Если строка в левой таблице не имеет совпадающей строки в правой таблице, результирующий набор строк содержит значения NULL для всех столбцов списка выбора из правой таблицы.

```
SELECT Empl.Name, Depts.Name from Empl
LEFT JOIN Depts
ON Empl.DepartmentID = Depts.ID
```

	Name	Name
1	Mike	Production
2	John	TC
3	Eric	NULL
4	Paul	Maintenance
5	Jessie	Production
6	Jonny	NULL

Рисунок 5 – Результат LEFT OUTER JOIN

RIGHT JOIN или RIGHT OUTER JOIN

Правое внешнее соединение является обратным для левого внешнего соединения. Возвращаются все строки правой таблицы. Для левой таблицы возвращаются значения NULL каждый раз, когда строка правой таблицы не имеет совпадающей строки в левой таблице.

```
SELECT Empl.Name, Depts.Name from Empl  
RIGHT JOIN Depts  
ON Empl.DepartmentID = Depts.ID
```

	Name	Name
1	Mike	Production
2	Jessie	Production
3	John	TC
4	Paul	Maintenance
5	NULL	Secret

Рисунок 6 – Результат RIGHT OUTER JOIN

FULL JOIN или FULL OUTER JOIN

Полное внешнее соединение возвращает все строки из правой и левой таблицы. Каждый раз, когда строка не имеет соответствия в другой таблице, столбцы списка выбора другой таблицы содержат значения NULL. Если между таблицами имеется соответствие, вся строка результирующего набора содержит значения данных из базовых таблиц.

```
SELECT Empl.Name, Depts.Name from Empl  
FULL JOIN Depts  
ON Empl.DepartmentID = Depts.ID
```

	Name	Name
1	Mike	Production
2	John	TC
3	Eric	NULL
4	Paul	Maintenance
5	Jessie	Production
6	Jonny	NULL
7	NULL	Secret

Рисунок 7 – Результат FULL JOIN

Перекрестные соединения (CROSS JOINS)

Перекрестное соединение возвращает все строки из левой таблицы. Каждая строка из левой таблицы соединяется со всеми строками из правой таблицы. Перекрестные соединения называются также декартовым произведением.

```
SELECT Empl.Name, Depts.Name from Empl
CROSS JOIN Depts
```

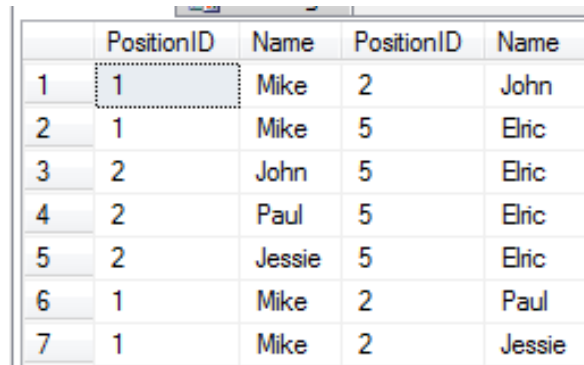
	Name	Name
1	Mike	Production
2	John	Production
3	Eric	Production
4	Paul	Production
5	Jessie	Production
6	Jonny	Production
7	Mike	TC
8	John	TC
9	Eric	TC
10	Paul	TC
11	Mike	NULL
12	John	NULL
13	Eric	NULL
14	John	Maintenance
15	Eric	Maintenance
16	Paul	Maintenance
17	Jessie	Maintenance
18	Jonny	Maintenance
19	Mike	NULL
20	John	NULL
21	Eric	NULL
22	Paul	NULL
23	Jessie	NULL
24	Jonny	NULL

Рисунок 8 – Результат CROSS JOIN

SELF JOINS

Таблицу можно соединить с собой в самосоединение. Используйте самосоединение, когда требуется создать результирующий набор, который соединяет записи в таблице с другими записями в той же таблице. Чтобы указать таблицу два раза в одном и том же запросе, необходимо задать псевдоним таблицы хотя бы для одного экземпляра имени таблицы. Этот псевдоним таблицы помогает обработчику запросов определить, какие данные должны быть представлены в столбцах (из правой или из левой версии таблицы)


```
SELECT e1.PositionID, e1.Name, e2.PositionID, e2.Name from Empl e1
JOIN Empl e2
ON e1.PositionID < e2.PositionID
```



	PositionID	Name	PositionID	Name
1	1	Mike	2	John
2	1	Mike	5	Eric
3	2	John	5	Eric
4	2	Paul	5	Eric
5	2	Jessie	5	Eric
6	1	Mike	2	Paul
7	1	Mike	2	Jessie

Рисунок 9 – Подobie иерархии сотрудников, полученной при помощи самосодинения

Вложенные запросы

Вложенным запросом называется запрос, помещаемый в инструкцию SELECT, INSERT, UPDATE или DELETE или в другой вложенный запрос. Подзапрос может быть использован везде, где разрешены выражения. В данном примере вложенный запрос используется в качестве выражения для столбца с именем MaxUnitPrice в инструкции SELECT.

```
SELECT Ord.SalesOrderID, Ord.OrderDate,
       (SELECT MAX(OrdDet.UnitPrice)
        FROM AdventureWorks.Sales.SalesOrderDetail AS OrdDet
        WHERE Ord.SalesOrderID = OrdDet.SalesOrderID) AS MaxUnitPrice
FROM AdventureWorks2008R2.Sales.SalesOrderHeader AS Ord
```

Вложенный запрос по-другому называют внутренним запросом или внутренней операцией выбора, в то время как инструкцию, содержащую вложенный запрос, называют внешним запросом или внешней операцией выбора.

Многие инструкции языка Transact-SQL, включающие подзапросы, можно записать в виде соединений. Другие запросы могут быть осуществлены только с помощью подзапросов. В языке Transact-SQL обычно не бывает разницы в производительности между инструкцией, включающей вложенный запрос, и семантически эквивалентной версией без вложенного запроса. Однако в некоторых случаях, когда проверяется существование, соединения показывают лучшую производительность. В противном случае для устранения дубликатов вложенный запрос должен обрабатываться для получения каждого результата внешнего запроса. В таких случаях метод работы соединений дает лучшие результаты. Следующий пример содержит запросы SELECT с вложенным запросом и с соединением, возвращающие одинаковый результирующий набор:

```
/* SELECT statement built using a subquery. */
SELECT Name
FROM AdventureWorks2008R2.Production.Product
WHERE ListPrice =
    (SELECT ListPrice
     FROM AdventureWorks2008R2.Production.Product
     WHERE Name = 'Chainring Bolts' );

/* SELECT statement built using a join that returns
the same result set. */
SELECT Prd1. Name
FROM AdventureWorks2008R2.Production.Product AS Prd1
JOIN AdventureWorks2008R2.Production.Product AS Prd2
    ON (Prd1.ListPrice = Prd2.ListPrice)
WHERE Prd2. Name = 'Chainring Bolts';
```

На вложенный запрос распространяются следующие ограничения:

- Список выбора вложенного запроса, начинающийся с оператора сравнения, может включать только одно выражение или имя столбца (за исключением операторов EXISTS и IN, работающих в инструкции SELECT * или в списке соответственно).
- Если предложение WHERE внешнего запроса включает имя столбца, оно должно быть совместимо для соединения со столбцом в списке выбора вложенного запроса.
- Типы данных **ntext**, **text** и **image** не могут быть использованы в списке выбора вложенных запросов.
- Вложенные запросы, представленные оператором немодифицированного сравнения (после которого нет ключевого слова ANY или ALL), не могут включать предложения типа GROUP BY и HAVING, поскольку они должны возвращать одиночное значение.
- Ключевое слово DISTINCT не может быть использовано во вложенном запросе, включающем предложение GROUP BY.
- Нельзя указывать предложения COMPUTE и INTO.
- Предложение ORDER BY может быть указано только вместе с предложением TOP.
- Представление, созданное с помощью вложенного запроса, не может быть обновлено.
- Список выбора вложенного запроса, начинающегося с предложения EXISTS, по соглашению содержит звездочку (*) вместо отдельного имени столбца. Правила для вложенного запроса, начинающегося с предложения EXISTS, являются такими же, как для стандартного списка выбора, поскольку вложенный запрос, начинающийся с предложения EXISTS, проводит проверку существования и возвращает TRUE или FALSE вместо данных.

Вложенные запросы могут быть указаны во многих местах:

Вложенные запросы с псевдонимами

Многие инструкции, где вложенный и внешний запросы ссылаются на одну и ту же таблицу, могут быть переформулированы как самосоединения (соединения таблицы с самой собой). Например, можно найти адреса сотрудников из конкретного региона с помощью вложенного запроса:

```
SELECT StateProvinceID, AddressID
FROM Person.Address
WHERE AddressID IN
    (SELECT AddressID
     FROM Person.Address
     WHERE StateProvinceID = 39)
```

Вложенные запросы с ключевыми словами NOT и IN

Результат вложенного запроса, в котором присутствует ключевое слово IN (или NOT IN) — это список из нуля или более значений. После того как вложенный запрос вернул результат, он используется внешним запросом.

Следующий запрос ищет названия всех колесных изделий, которые производит компания Adventure Works Cycles:

```
SELECT Name
FROM Production.Product
WHERE ProductSubcategoryID IN
    (SELECT ProductSubcategoryID
     FROM Production.ProductSubcategory
     WHERE Name = 'Wheels');
```

Вложенные запросы с ключевым словом NOT IN также возвращают список из нуля или более значений.

В следующем запросе выполняется поиск названий продуктов, не являющихся готовыми велосипедами.

```
SELECT Name
FROM Production.Product
WHERE ProductSubcategoryID NOT IN
    (SELECT ProductSubcategoryID
     FROM Production.ProductSubcategory
     WHERE Name = 'Mountain Bikes'
        OR Name = 'Road Bikes'
        OR Name = 'Touring Bikes')
```

Эту инструкцию нельзя преобразовать в соединение. Аналогичное соединение по неравенству имеет другой смысл: оно находит названия продуктов, которые принадлежат какой-либо подкатегории, отличной от готового велосипеда.

Подзапросы в инструкциях UPDATE, DELETE и INSERT

Подзапросы вкладываются в инструкции языка DML UPDATE, DELETE, INSERT и SELECT.

В следующем примере удваивается значение столбца ListPrice таблицы Production.Product. Вложенный запрос в предложении WHERE ссылается на таблицу Purchasing.ProductVendor для ограничения количества обновляемых строк таблицы Product только теми, у которых BusinessEntity 1540:

```
UPDATE Production.Product
SET ListPrice = ListPrice * 2
WHERE ProductID IN
```

```
(SELECT ProductID  
FROM Purchasing.ProductVendor  
WHERE BusinessEntityID = 1540);
```

Вложенные запросы с операторами сравнения

Вложенные запросы могут быть введены с помощью одного из операторов сравнения (=, < >, > =, < , ! >, ! < или < =).

Вложенный запрос, введенный с помощью немодифицированного оператора сравнения (оператора сравнения, за которым не следуют ключевые слова ANY или ALL), должен вернуть одиночное значение, а не список значений, как вложенные запросы, введенные с помощью IN. Если такой вложенный запрос возвращает более одного значения, то SQL Server отображает сообщение об ошибке.

Чтобы использовать подзапрос, начинающийся с немодифицированного оператора сравнения, необходимо достаточно хорошо знать свои данные и природу проблемы, чтобы быть уверенным, что вложенный запрос возвратит точно одно значение.

Например, если предполагается, что каждый менеджер по продажам отвечает только за одну территорию продаж, и нужно найти клиентов, расположенных на территории, за которую отвечает Линда Митчелл, можно написать инструкцию с вложенным запросом, начинающимся с простого оператора сравнения =.

```
SELECT CustomerID  
FROM Sales.Customer  
WHERE TerritoryID =  
    (SELECT TerritoryID  
     FROM Sales.SalesPerson  
     WHERE BusinessEntityID = 276)
```

Однако если Линда Митчелл работала более чем с одной территорией продаж, то результатом будет сообщение об ошибке. Вместо оператора сравнения = может использоваться формулировка IN (= ANY также работает).

Вложенные запросы, начинающиеся с немодифицированных операторов сравнения, часто включают агрегатные функции, потому что они возвращают одиночное значение. Например, следующая инструкция находит названия всех продуктов, у которых цена по прейскуранту больше, чем средняя цена по прейскуранту.

```
SELECT Name  
FROM Production.Product  
WHERE ListPrice >  
    (SELECT AVG (ListPrice)  
     FROM Production.Product)
```

Поскольку вложенные запросы, начинающиеся с немодифицированных операторов сравнения, должны возвращать одиночное значение, они не могут включать предложения GROUP BY или HAVING (за исключением случаев, когда достоверно известно, что предложение GROUP BY или HAVING возвратит одиночное значение). Например, следующий запрос находит продукты, оцененные выше, чем самый дешевый продукт, который находится в подкатегории 14.

```
SELECT Name
FROM Production.Product
WHERE ListPrice >
      (SELECT MIN (ListPrice)
       FROM Production.Product
       GROUP BY ProductSubcategoryID
       HAVING ProductSubcategoryID = 14)
```

Операторы сравнения с модификаторами ANY, SOME или ALL

Операторы сравнения с вложенными запросами могут быть уточнены с помощью ключевых слов ALL или ANY. SOME является эквивалентом ANY в стандарте ISO.

Вложенные запросы операторов сравнения с модификаторами возвращают список из нуля или более значений и могут включать предложения GROUP BY или HAVING. Эти вложенные запросы могут быть переформулированы с использованием ключевого слова EXISTS.

Рассмотрим, например оператор сравнения >: >ALL будет означать «больше любого значения». Другими словами, это сравнение с максимальным значением. Например, >ALL (1, 2, 3) означает «больше 3». >ANY означает «больше по крайней мере одного значения», т. е. «больше минимума». Поэтому >ANY (1, 2, 3) означает «больше 1».

Чтобы строка результата вложенного запроса с >ALL удовлетворяла условию, заданному внешним запросом, значение в столбце, для которого вводится вложенный запрос, должно быть больше каждого значения из списка, возвращаемого вложенным запросом.

Аналогичным образом, чтобы строка результата вложенного запроса с >ANY, удовлетворяла условию, заданному внешним запросом, значение в столбце, для которого вводится вложенный запрос, должно быть больше хотя бы одного значения из списка, возвращаемого вложенным запросом.

Следующий запрос содержит пример вложенного запроса, используемого оператором сравнения с модификатором ANY. Он вернет все продукты, цены на которые больше или равны максимальной цене в любой подкатегории продуктов.

```
SELECT Name
FROM Production.Product
WHERE ListPrice >= ANY
      (SELECT MAX (ListPrice)
       FROM Production.Product
       GROUP BY ProductSubcategoryID)
```

Подзапросы с ключевым словом EXISTS

Если вложенный запрос вводится с помощью ключевого слова EXISTS, то он выступает в роли проверки на существование. В предложении WHERE внешнего запроса проверяется факт существования строк, возвращенных вложенным запросом. Вложенный запрос не выдает никаких данных, а возвращает значение TRUE или FALSE.

Вложенный запрос, созданный с помощью ключевого слова EXISTS, имеет

следующий синтаксис.

WHERE [NOT] EXISTS (subquery)

По представленному ниже запросу осуществляется поиск всех наименований товаров, относящихся к подкатегории Wheels.

```
SELECT Name
FROM Production.Product
WHERE EXISTS
  (SELECT *
   FROM Production.ProductSubcategory
   WHERE ProductSubcategoryID =
         Production.Product.ProductSubcategoryID
    AND Name = 'Wheels')
```

Подзапросы с оператором NOT EXISTS

Оператор NOT EXISTS работает так же, как и оператор EXISTS, за исключением того, что предложение WHERE, в котором используется этот оператор, выполняется, если вложенный запрос не возвращает ни одной строки.

Например чтобы найти имена продуктов, не находящихся в подкатегории wheels:

```
SELECT Name
FROM Production.Product
WHERE NOT EXISTS
  (SELECT *
   FROM Production.ProductSubcategory
   WHERE ProductSubcategoryID =
         Production.Product.ProductSubcategoryID
    AND Name = 'Wheels')
```

Вложенные запросы, используемые вместо выражения

В языке Transact-SQL вложенный запрос может быть заменен в любом месте, где выражение может использоваться в инструкции SELECT, UPDATE, INSERT и DELETE, за исключением списка ORDER BY.

Следующий пример показывает, как можно использовать это улучшение. Запрос находит цены на все горные велосипеды, их среднюю цену и разницу между средней ценой и ценой каждого горного велосипеда.

```
SELECT Name, ListPrice,
  (SELECT AVG(ListPrice) FROM Production.Product) AS Average,
  ListPrice - (SELECT AVG(ListPrice) FROM Production.Product)
  AS Difference
FROM Production.Product
WHERE ProductSubcategoryID = 1;
```

6. Изменение данных в таблицах

Язык обработки данных DML представляет словарь, используемый для получения данных и работы с ними в SQL Server. Эти инструкции предназначены для добавления данных, изменения данных, запроса данных и удаления данных из базы данных SQL Server.

Вставка данных

Инструкция INSERT - Добавляет одну или несколько строк в таблицу или представление:

Вставка одной строки

```
INSERT INTO Production.UnitMeasure
VALUES (N'FT', N'Feet', '20080414');
```

Вставка нескольких строк

```
INSERT INTO Production.UnitMeasure
VALUES (N'FT2', N'Square Feet ', '20080923'),
(N'Y', N'Yards', '20080923'),
(N'Y3', N'Cubic Yards', '20080923');
```

Вставка при помощи SELECT

```
INSERT INTO dbo.EmployeeSales
SELECT 'SELECT', sp.BusinessEntityID, c.LastName, sp.SalesYTD
FROM Sales.SalesPerson AS sp
INNER JOIN Person.Person AS c
    ON sp.BusinessEntityID = c.BusinessEntityID
WHERE sp.BusinessEntityID LIKE '2%'
ORDER BY sp.BusinessEntityID, c.LastName;
```

Обновление данных

UPDATE (Transact-SQL) - Изменяет существующие данные в таблице или представлении в SQL Server.

Пример простого обновления

```
UPDATE Sales.SalesPerson
SET Bonus = 6000, CommissionPct = .10, SalesQuota = NULL;
```

Обновление определенных записей

```
UPDATE Production.Product
SET Color = N'Metallic Red'
WHERE Name LIKE N'Road-250%' AND Color = N'Red';
```

Обновление с вложенным запросом

```
UPDATE Sales.SalesPerson
SET SalesYTD = SalesYTD +
    (SELECT SUM(so.SubTotal)
     FROM Sales.SalesOrderHeader AS so
     WHERE so.OrderDate = (SELECT MAX(OrderDate)
                           FROM Sales.SalesOrderHeader AS so2
                           WHERE so2.SalesPersonID = so.SalesPersonID)
  AND Sales.SalesPerson.BusinessEntityID = so.SalesPersonID
GROUP BY so.SalesPersonID);
```

Удаление записей

DELETE - Удаляет одну или несколько строк из таблицы или представления в SQL Server 2012

Примеры:

Удаление всех записей из таблицы

```
DELETE FROM Sales.SalesPersonQuotaHistory;
```

Удаление определенных записей

```
DELETE FROM Production.ProductCostHistory
WHERE StandardCost > 1000.00;
```

Использование вложенного запроса при удалении

```
DELETE FROM Sales.SalesPersonQuotaHistory
WHERE BusinessEntityID IN
    (SELECT BusinessEntityID
     FROM Sales.SalesPerson
     WHERE SalesYTD > 2500000.00);
```


7. Объекты для работы с данными

MS SQL Server имеет ряд **встроенных объектов и механизмов**, предназначенных для работы с данными:

- Views или представления
- Пользовательские функции
- Триггеры
- Хранимые процедуры

Представления

Представление — это виртуальная таблица, содержимое которой определяется запросом. Как и таблица, представление состоит из ряда именованных столбцов и строк данных. Пока представление не будет проиндексировано, оно не существует в базе данных как хранимая совокупность значений. Строки и столбцы данных извлекаются из таблиц, указанных в определяющем представлении запросе и динамически создаваемых при обращениях к представлению.

Представление выполняет функцию фильтра базовых таблиц, на которые оно ссылается. Определяющий представление запрос может быть инициирован в одной или нескольких таблицах или в других представлениях текущей или других баз данных. Кроме того, для определения представлений с данными из нескольких разнородных источников можно использовать распределенные запросы. Это полезно, например, если нужно объединить структурированные подобным образом данные, относящиеся к разным серверам, каждый из которых хранит данные конкретного отдела организации.

Представления обычно используются для направления, упрощения и настройки восприятия каждым пользователем информации базы данных. Представления могут использоваться как механизмы безопасности, давая возможность пользователям обращаться к данным через представления, но не предоставляя им разрешений на непосредственный доступ к базовым таблицам, лежащим в основе представлений. Представления могут использоваться для обеспечения интерфейса обратной совместимости, моделирующего таблицу, которая существует, но схема которой изменилась. Представления могут также использоваться при прямом и обратном копировании данных в SQL Server для повышения производительности и секционирования данных.

Кроме основных определяемых пользователем представлений, выполняющих стандартные роли, в SQL Server предусмотрены следующие типы представлений, которые соответствуют специальным назначениям в базе данных.

Индексированные представления

Индексированным называется материализованное представление. Это означает, что определение представления вычисляется, а результирующие данные хранятся точно так же, как и таблица. Индексировать представление можно, создав для него уникальный кластеризованный индекс. Индексированные представления могут существенно повысить производительность некоторых типов запросов. Индексированные представления эффективнее всего использовать в

запросах, группирующих множество строк. Они не очень хорошо подходят для часто обновляющихся базовых наборов данных.

Секционированные представления

Секционированным называется представление, соединяющее горизонтально секционированные данные набора таблиц-элементов, находящихся на одном или нескольких серверах. При этом данные выглядят так, как будто находятся в одной таблице. Представление, соединяющее таблицы-элементы одного экземпляра SQL Server, называется локальным секционированным представлением.

Системные представления

Системные представления предоставляют доступ к метаданным каталога. Системные представления можно использовать для получения сведений об экземпляре SQL Server или объектах, определенных в экземпляре. Например, получить сведения об определяемых пользователем базах данных, доступных в экземпляре, можно через представление каталога sys.databases.

Представление можно использовать в следующих целях:

- Для направления, упрощения и настройки восприятия информации в базе данных каждым пользователем.
- В качестве механизма безопасности, позволяющего пользователям обращаться к данным через представления, но не предоставляя им разрешений на непосредственный доступ к базовым таблицам.
- Для предоставления интерфейса обратной совместимости, моделирующего таблицу, схема которой изменилась.

Пример представления, дающего информацию по нескольким таблицам:

```
CREATE VIEW EmployeeInfo AS
SELECT Empl.ID, Empl.Name, Depts.Name as Department, Position.Name as Position
FROM Empl
JOIN Depts on Empl.DepartmentID = Depts.ID
JOIN Position on Empl.PositionID = Position.ID
```

Пользовательские функции

Подобно функциям в языках программирования, определяемые пользователем функции Microsoft SQL Server являются подпрограммами, которые принимают параметры, выполняют действия, такие как сложные вычисления, а затем возвращают результат этих действий в виде значения. Возвращаемое значение может быть либо единичным скалярным значением, либо результирующим набором.

Преимущества:

Определяемые пользователем функции SQL Server предоставляют следующие преимущества.

- Делают возможным модульное программирование.

Можно, однажды создав функцию, сохранить ее в базе данных, а затем любое число раз вызывать из своей программы. Определяемые пользователем функции могут быть изменены независимо от исходного кода программы.

- Позволяют ускорить выполнение.

Как и хранимые процедуры, определяемые пользователем функции Transact-SQL снижают стоимость компиляции кода Transact-SQL, кэшируя и повторно используя планы выполнения. Это означает, что для определяемых пользователем функций нет необходимости выполнять повторный синтаксический анализ и оптимизацию при каждом вызове, что значительно ускоряет их выполнение.

Функции CLR предоставляют значительный выигрыш в производительности по сравнению с функциями Transact-SQL для вычислительных задач, работы со строками и бизнес-логикой. Функции Transact-SQL лучше приспособлены для логики доступа к данным.

- Позволяют уменьшить сетевой трафик.

Операция, которая фильтрует данные на основе какого-нибудь сложного ограничения и не может быть выражена одним скалярным выражением, может быть реализована в виде функции. Ее можно вызвать из предложения WHERE, чтобы уменьшить число строк, возвращаемых клиенту.

Определяемые пользователем функции могут быть написаны на языке Transact-SQL или на любом языке программирования .NET.

Любая определяемая пользователем функция состоит из двух частей: заголовка и текста. Функция принимает нуль и более параметров и возвращает либо скалярное значение, либо таблицу.

Заголовок определяет:

- имя функции с необязательным именем схемы или владельца;
- имя и тип данных входного аргумента;
- параметры, применимые к входному аргументу;
- тип данных возвращаемого значения и необязательное имя;
- параметры, применимые к возвращаемому значению.

Текст определяет действие или логику, которую выполняет функция. Он может содержать:

- одну или несколько инструкций Transact-SQL, реализующих логику функции;
- ссылку на сборку .NET.

В следующем примере показана простая определяемая пользователем функция Transact-SQL и обозначены ее главные компоненты. Функция на основе переданной даты вычисляет и возвращает день недели, соответствующий этой дате.

```
IF OBJECT_ID(N'dbo.GetWeekDay', N'FN') IS NOT NULL
    DROP FUNCTION dbo.GetWeekDay;
GO
CREATE FUNCTION dbo.GetWeekDay          -- function name
(@Date datetime)                       -- input parameter name and data type
RETURNS int                            -- return parameter data type
AS
BEGIN                                  -- begin body definition
    RETURN DATEPART (weekday, @Date)   -- action performed
```

```
END;  
GO
```

Вызов функции:

```
SELECT dbo.GetWeekDay(CONVERT(DATETIME,'20020201',101)) AS DayOfWeek;  
GO
```

Пример функции, заменяющей присоединение таблиц:

```
CREATE FUNCTION dbo.EmplPos (@id int)  
RETURNS nvarchar(50)  
WITH EXECUTE AS CALLER  
AS  
BEGIN  
    DECLARE @position nvarchar(50);  
    SET @position = (SELECT Position.Name FROM Position  
                    JOIN Empl on Position.ID = Empl.PositionID  
                    WHERE Empl.ID = @id);  
    RETURN(@position);  
END;  
GO  
  
--calling  
SELECT Name, dbo.EmplPos(ID) FROM Empl
```

Хранимые процедуры

Хранимая процедура в SQL Server — это группа из одной или нескольких инструкций Transact-SQL или ссылка на метод Microsoft .NET Framework среды CLR. Процедуры аналогичны конструкциям в других языках программирования, поскольку обеспечивают следующее:

- обрабатывают входные параметры и возвращают вызывающей программе значения в виде выходных параметров;
- содержат программные инструкции, которые выполняют операции в базе данных, включая вызов других процедур;
- возвращают значение состояния вызывающей программе, таким образом передавая сведения об успешном или неуспешном завершении (и причины последнего).

Преимущества использования процедур:

Снижение сетевого трафика между клиентами и сервером

Команды в процедуре выполняются как один пакет кода. Это позволяет существенно сократить сетевой трафик между сервером и клиентом, поскольку по сети отправляется только вызов на выполнение процедуры. Без инкапсуляции кода, предоставляемой процедурой, по сети бы пришлось пересылать все отдельные строки кода.

Большая безопасность

Многие пользователи и клиентские программы могут выполнять операции с базовыми объектами базы данных посредством процедур, даже если у них нет

прямых разрешений на доступ к базовым объектам. Процедура проверяет, какие из процессов и действий могут выполняться, и защищает базовые объекты базы данных. Это устраняет необходимость предоставлять разрешения на уровне индивидуальных объектов и упрощает формирование уровней безопасности.

Предложение EXECUTE AS может быть указано в инструкции CREATE PROCEDURE, чтобы разрешить олицетворение других пользователей или разрешить пользователям или приложениям выполнять определенные действия баз данных без необходимости иметь прямые разрешения на базовые объекты и команды. Например, для некоторых действий, таких как TRUNCATE TABLE, предоставить разрешения нельзя. Чтобы выполнить инструкцию TRUNCATE TABLE, у пользователя должны быть разрешения ALTER на нужную таблицу. Предоставление разрешений ALTER не всегда подходит, так как фактические разрешения пользователя выходят за пределы возможности усечения таблицы. Заклучив инструкцию TRUNCATE TABLE в модуль и указав, что этот модуль должен выполняться от имени пользователя, у которого есть разрешения на изменение таблицы, можно предоставить разрешение на усечение таблицы пользователю с разрешением EXECUTE для этого модуля.

При вызове процедуры через сеть виден только вызов на выполнение процедуры. Следовательно, злоумышленники не смогут видеть имена объектов таблиц и баз данных, внедрять свои инструкции Transact-SQL или выполнять поиск важных данных.

Использование параметров в процедурах помогает предотвратить атаки типа «инъекция SQL». Поскольку входные данные параметра обрабатываются как литеральные значения, а не как исполняемый код, злоумышленнику будет труднее вставить команду в инструкции Transact-SQL в процедуре и создать угрозу безопасности.

Процедуры могут быть зашифрованы, что позволяет замаскировать исходный код.

Повторное использование кода

Если какой-то код многократно используется в операции базы данных, то отличным решением будет произвести его инкапсуляцию в процедуры. Это устранил необходимость излишнего копирования того же кода, снизит уровень несогласованности кода и позволит осуществлять доступ к коду любым пользователям или приложениям, имеющим необходимые разрешения.

Более легкое обслуживание

Если клиентские приложения вызывают процедуры, а операции баз данных остаются на уровне данных, то для внесения изменений в основную базу данных будет достаточно обновить только процедуры. Уровень приложения остается незатронутым изменениями в схемах баз данных, связях или процессах.

Повышенная производительность

По умолчанию компиляция процедуры и создание плана выполнения, используемого для последующих выполнений, производится при ее первом запуске. Поскольку обработчику запросов не нужно создавать новый план, обычно обработка процедуры занимает меньше времени.

Если в таблицах или данных, на которые ссылается процедура, произошли значительные изменения, то наличие предварительно скомпилированного плана может вызвать замедление работы процедуры. В этом случае перекомпиляция процедуры и принудительное создание нового плана выполнения может улучшить производительность.

Типы хранимых процедур:

Пользовательские процедуры

Пользовательские процедуры могут быть созданы в пользовательской базе данных или любых системных базах данных, за исключением базы данных **Resource**. Процедура может быть разработана либо на языке Transact-SQL, либо как ссылка на метод Microsoft .NET Framework среды CLR.

Временные процедуры

Временные процедуры — это один из видов пользовательских процедур. Временные процедуры схожи с постоянными процедурами, за исключением того, что они хранятся в базе данных **tempdb**. Существует два вида временных процедур: локальные и глобальные. Они отличаются друг от друга именами, видимостью и доступностью. Имена локальных временных процедур начинаются с одного знака диеза (#); они видны только текущему соединению пользователя и удаляются, когда закрывается соединение. Имена глобальных временных процедур начинаются с двух знаков диеза (##); они видны любому пользователю и удаляются после окончания последнего сеанса, использующего процедуру.

Система

Системные процедуры включены в SQL Server. Физически они хранятся во внутренней скрытой базе данных **Resource**. Логически они отображаются в схеме **sys** каждой системной и пользовательской базы данных. В дополнение к этому, база данных **msdb** также содержит системные хранимые процедуры в схеме **dbo**. Эти процедуры используются для планирования предупреждений и заданий. Поскольку названия системных процедур начинаются с префикса **sp_**, этот префикс не рекомендуется использовать при создании пользовательских процедур.

SQL Server поддерживает системные процедуры, обеспечивающие интерфейс между SQL Server и внешними программами для выполнения различных действий по обслуживанию системы. Эти расширенные процедуры имеют префикс **xp_**.

Расширенные пользовательские процедуры

Расширенные процедуры позволяют создавать внешние подпрограммы на языках программирования (например, на C). Они представляют собой библиотеки DLL, которые могут динамически загружаться и выполняться экземпляром SQL Server.

Пример хранимой процедуры:

```
CREATE PROCEDURE dbo.NewEmployee
@EmployeeName nvarchar(50),
@DeptName nvarchar(50),
@PositionName nvarchar(50),
@EmployeeID int OUTPUT
AS
BEGIN
INSERT INTO Empl (Name, DepartmentID, PositionID)
VALUES
(
@EmployeeName,
(SELECT ID FROM Depts WHERE Depts.Name = @DeptName),
(SELECT ID FROM Position WHERE Position.Name = @PositionName)
)
```

```
);  
SET @EmployeeID = SCOPE_IDENTITY()  
END
```

Триггеры

Триггеры бывают двух видов: триггеры DML и триггеры DDL

Триггеры DML — это хранимые процедуры особого типа, автоматически вступающие в силу, если происходит событие языка обработки данных DML, которое затрагивает таблицу или представление, определенное в триггере. События DML включают инструкции INSERT, UPDATE или DELETE. Триггеры DML могут использоваться для предписания бизнес-правил и правил целостности данных, выполнения запросов к другим таблицам и включения сложных инструкций Transact-SQL. Триггер и инструкция, при выполнении которой он срабатывает, считаются одной транзакцией, которую можно откатить назад внутри триггера. При обнаружении серьезной ошибки (например, нехватки места на диске) вся транзакция автоматически откатывается назад.

Триггеры DML аналогичны ограничениям в том, что могут предписывать целостность сущностей или целостность домена. Вообще говоря, целостность сущностей должна всегда предписываться на самом нижнем уровне с помощью индексов, являющихся частью ограничений PRIMARY KEY и UNIQUE или создаваемых независимо от ограничений. Целостность домена должна быть предписана через ограничения CHECK, а ссылочная целостность — через ограничения FOREIGN KEY. Триггеры DML наиболее полезны в тех случаях, когда функции ограничений не удовлетворяют функциональным потребностям приложения.

В следующем списке приведено сравнение триггеров DML с ограничениями и указано, в чем триггеры DML имеют преимущества.

- Триггеры DML позволяют каскадно проводить изменения через связанные таблицы в базе данных; но эти изменения могут осуществляться более эффективно с использованием каскадных ограничений ссылочной целостности. Ограничения FOREIGN KEY могут проверить значения столбца только на предмет точного совпадения со значениями другого столбца, за исключением случаев, когда с помощью предложения REFERENCES задаются каскадные ссылочные действия.

- Для предотвращения случайных или неверных операций INSERT, UPDATE и DELETE и реализации других более сложных ограничений, чем те, которые определены при помощи ограничения CHECK.

В отличие от ограничений CHECK, DML-триггеры могут ссылаться на столбцы других таблиц. Например, триггер может использовать инструкцию SELECT для сравнения вставленных или обновленных данных и выполнения других действий, например изменения данных или отображения пользовательского сообщения об ошибке.

- Чтобы оценить состояние таблицы до и после изменения данных и предпринять действия на основе этого различия.

- Несколько DML-триггеров одинакового типа (INSERT, UPDATE или DELETE) для таблицы позволяют предпринять несколько различных действий в ответ на одну инструкцию изменения данных.
- Ограничения могут сообщать об ошибках только с помощью соответствующих стандартных системных сообщений. Если для пользовательского приложения требуются более сложные методы управления ошибками и, соответственно, пользовательские сообщения, то необходимо использовать триггер.
- При использовании триггеров DML может произойти откат изменений, нарушающих ссылочную целостность, что приводит к запрету модификации данных. Подобные триггеры могут применяться при изменении внешнего ключа в случаях, когда новое значение не соответствует первичному ключу. Обычно в указанных случаях используются ограничения FOREIGN KEY.
- Если в таблице триггеров существуют ограничения, то их проверка осуществляется между выполнением триггеров INSTEAD OF и AFTER. В случае нарушения ограничений выполняется откат действий триггеров INSTEAD OF, а триггер AFTER не срабатывает.

Виды DML триггеров:

Триггер AFTER

Триггеры AFTER выполняются после выполнения действий инструкции INSERT, UPDATE, MERGE или DELETE. Триггеры AFTER никогда не выполняются, если происходит нарушение ограничения; поэтому эти триггеры нельзя использовать для какой-либо обработки, которая могла бы предотвратить нарушение ограничения.

Триггер INSTEAD OF

Триггеры INSTEAD OF переопределяют стандартные действия инструкции, вызывающей триггер. Поэтому они могут использоваться для проверки на наличие ошибок или проверки значений на одном или нескольких столбцах и выполнения дополнительных действий перед вставкой, обновлением или удалением одной или нескольких строк. Например, если обновляемое значение в столбце почасовой оплаты в таблице учетной ведомости начинает превышать определенное значение, то с помощью этого триггера можно либо задать вывод сообщения об ошибке и откатить транзакцию, либо сделать вставку новой записи в след аудита до вставки записи в таблицу учетной ведомости. Главное преимущество триггеров INSTEAD OF в том, что они позволяют поддерживать обновления для таких представлений, которые обновлять невозможно. Например, в представлении, основанном на нескольких базовых таблицах, должен использоваться триггер INSTEAD OF для поддержки операций вставки, обновления и удаления, которые ссылаются на данные больше чем в одной таблице. Другое преимущество триггера INSTEAD OF состоит в том, что он обеспечивает логику кода, при которой можно отвергать одни части пакета и принимать другие.

Триггеры DDL

Можно использовать триггеры DDL для аудита операций базы данных или сервера, с помощью которых происходит создание, изменение или удаление объектов базы данных, или для обеспечения того, чтобы инструкции DDL предписывали конкретные бизнес-правила до их запуска. Триггеры DDL активируются в ответ на различные события языка DDL. Эти события в основном

соответствуют инструкциям Transact-SQL, которые начинаются с ключевых слов CREATE, ALTER, DROP, GRANT, DENY, REVOKE или UPDATE STATISTICS. Системные хранимые процедуры, выполняющие операции, подобные операциям DDL, также могут запускать триггеры DDL.

Используйте триггеры DDL, если хотите сделать следующее.

- Предотвратить внесение определенных изменений в схему базы данных.
- Выполнить в базе данных некоторые действия в ответ на изменения в схеме базы данных.
- Записывать изменения или события схемы базы данных.

Триггеры DDL срабатывают в ответ на событие Transact-SQL, обработанное текущей базой данных или текущим сервером. Область триггера зависит от события. Например, триггер DDL, созданный для срабатывания на событие CREATE TABLE, может срабатывать каждый раз, когда в базе данных или в экземпляре сервера возникает событие CREATE_TABLE. Триггер DDL, созданный для запуска в ответ на событие CREATE_LOGIN, может выполнять это только при возникновении события CREATE_LOGIN в экземпляре сервера.

В следующем примере триггер DDL safety будет срабатывать каждый раз, когда в базе данных будет выполняться инструкция DROP_TABLE или происходить событие ALTER_TABLE.

```
CREATE TRIGGER safety
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
    PRINT 'You must disable Trigger "safety" to drop or alter tables!'
    ROLLBACK;
```