**04 Querying Multiple Tables**

В реальных условиях вполне вероятно, что данные, необходимые для запроса, хранятся в нескольких местах. Мы узнаем, как писать запросы, объединяющие данные из нескольких источников в Microsoft SQL Server . Вы будете писать запросы, содержащие соединения, которые позволяют извлекать данные из двух (или более) таблиц на основе связей данных между таблицами.

**Lesson 1**

**Understanding Joins**

The FROM Clause and Virtual Tables

Вы уже узнали о логическом порядке операций, выполняемых при обработке запроса SQL Server. Вы помните, что предложение FROM инструкции SELECT является первым этапом обработки. Это предложение определяет, какая таблица или таблицы будут источником строк для запроса.

Как вы увидите в этом модуле, это справедливо независимо от того, запрашиваете ли вы одну таблицу или объединяете несколько таблиц в качестве источника запроса. Чтобы узнать о дополнительных возможностях предложения FROM, полезно подумать о функции предложения как о создании и заполнении виртуальной таблицы. Эта виртуальная таблица будет содержать выходные данные предложения FROM и впоследствии будет использоваться другими этапами инструкции SELECT, такими как предложение WHERE. При добавлении дополнительных функций, таких как операторы соединения, в предложение FROM будет полезно подумать о назначении элементов предложения FROM как о добавлении строк в виртуальную таблицу или удалении строк из нее.

Виртуальная таблица, созданная предложением FROM, является только логической сущностью. В SQL Server не создается физическая таблица, будь то постоянная или временная, для хранения результатов предложения FROM при передаче в предложение WHERE или на других последующих этапах.

Синтаксис оператора SELECT, который вы использовали для предыдущих запросов в этом курсе, выглядит следующим образом:

SELECT ...

FROM <table> AS <alias>;

Сначала обрабатывается предложение FROM, и в результате на любые псевдонимы таблиц, которые вы там создаете, могут быть ссылки в предложении SELECT. В этом модуле вы увидите множество примеров псевдонимов таблиц. Хотя эти псевдонимы являются необязательными, за исключением запросов с самосоединением, вы быстро увидите, как они могут быть удобным инструментом при написании запросов. Сравните следующие два запроса, которые имеют одинаковые выходные данные, но отличаются использованием псевдонимов. (Обратите внимание, что в примерах используется предложение JOIN, которое будет рассмотрено позже в этом модуле.)

**Without Table Aliases**

USE TSQL ;

GO

SELECT Sales.Orders.orderid, Sales.Orders.orderdate,

Sales.OrderDetails.productid,Sales.OrderDetails.unitprice,

Sales.OrderDetails.qty

FROM Sales.Orders

JOIN Sales.OrderDetails ON Sales.Orders.orderid = Sales.OrderDetails.orderid ;

**With Table Aliases**

USE TSQL ;

GO

SELECT o.orderid, o.orderdate,

od.productid, od.unitprice,

od.qty

FROM Sales.Orders AS o

JOIN Sales.OrderDetails AS od ON o.orderid = od.orderid ;

Использование псевдонимов таблиц улучшает читабельность запроса, не влияя на производительность. Настоятельно рекомендуется использовать псевдонимы таблиц в запросах с несколькими таблицами.

После того, как таблица была обозначена псевдонимом в предложении FROM, рекомендуется использовать псевдоним при ссылке на столбцы из этой таблицы в других предложениях.

**Терминология объединения: Декартово произведение**

Когда вы узнаете о написании многотабличных запросов в T-SQL, важно понять концепцию декартовых продуктов. В математике это произведение двух множеств. Продукт набора из двух предметов и набора из шести—это набор из 12 предметов-или 6 х 2.

В базах данных декартово произведение является результатом соединения каждой строки одной входной таблицы со всеми строками другой входной таблицы. Произведение таблицы с 10 строками и таблицы со 100 строками - это результирующий набор с 1000 строками.

Для большинства запросов T-SQL декартово произведение не является желаемым результатом. Как правило, декартово произведение возникает, когда две входные таблицы соединяются без учета каких-либо логических связей между ними. При отсутствии информации о связях обработчик запросов SQL Server будет выводить все возможные комбинации строк.

Хотя это может иметь некоторые практические применения, такие как создание таблицы чисел или создание тестовых данных, обычно это не полезно и может иметь серьезные последствия для производительности.

**Overview of Join Types 4-4**

Для заполнения виртуальной таблицы, созданной предложением FROM в инструкции SELECT, SQL Server использует операторы join. Они добавляют или удаляют строки из виртуальной таблицы, прежде чем она будет передана последующим логическим этапам инструкции SELECT:

Оператор перекрестного соединения (CROSS JOIN) добавляет все возможные комбинации строк двух входных таблиц в виртуальную таблицу. Любая фильтрация строк будет происходить в предложении WHERE. Для большинства запросов этого оператора следует избегать.

Оператор внутреннего соединения (INNER JOIN или просто JOIN) сначала создает декартово произведение, а затем фильтрует результаты, используя предикат, указанный в предложении ON, удаляя все строки из виртуальной таблицы, которые не удовлетворяют предикату. Внутреннее соединение-это очень распространенный тип соединения для извлечения строк с атрибутами, которые совпадают в разных таблицах, например, сопоставление Клиентов с Заказами по общему идентификатору.

Оператор внешнего соединения (LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN) сначала создает декартово произведение и, как и внутреннее соединение, фильтрует результаты, чтобы найти строки, соответствующие в каждой таблице. Однако все строки из одной таблицы сохраняются и добавляются обратно в виртуальную таблицу после применения начального фильтра. Нули помещаются в атрибуты, в которых не найдено совпадающих значений.

Если не указано иное с помощью CROSS или OUTER, оператор СОЕДИНЕНИЯ по умолчанию использует INNER join.

**T-SQL Syntax Choices**

На протяжении всей истории SQL Server продукт менялся, чтобы идти в ногу с изменениями стандартов Американского национального института стандартов (ANSI) для языка SQL. Одним из наиболее заметных мест, где видны эти изменения, является синтаксис оператора join в предложении FROM. В ANSI SQL-89 оператор ON не был определен.

Соединения были представлены в списке таблиц, разделенных запятыми, и любая фильтрация, например для внутреннего соединения, выполнялась в предложении WHERE. Этот синтаксис по-прежнему поддерживается SQL Server, но из-за сложности представления фильтров для внешнего соединения в предложении WHERE, в дополнение к любой другой фильтрации, здесь не рекомендуется.

Кроме того, если предложение WHERE случайно опущено, соединения в стиле ANSI SQL-89 могут легко превратиться в декартовы продукты и вызвать проблемы с производительностью.

Cartesian Product **4-5**

USE TSQL;

GO

/\* This is ANSI SQL-89 syntax for an inner join, with the filtering performed in the

WHERE clause. \*/

SELECT c.companyname, o.orderdate

FROM Sales.Customers AS c, Sales.Orders AS o

WHERE c.custid = o.custid;

....

(830 row(s) affected)

/\*

Это синтаксис ANSI SQL-89 для внутреннего соединения, опуская предложение wHERE и вызывая непреднамеренное декартово соединение.

\*/

SELECT c.companyname, o.orderdate

FROM Sales.Customers AS c, Sales.Orders AS o;

...

(75530 row(s) affected)

С появлением стандарта ANSI SQL-92 была добавлена поддержка предложения ON. T-SQL также поддерживает этот синтаксис. Соединения представляются в предложении FROM с помощью соответствующего оператора СОЕДИНЕНИЯ. Логическая связь между таблицами, которая становится предикатом фильтра, представлена предложением ON.

В следующем примере повторяется предыдущий запрос с более новым синтаксисом:

SELECT c.companyname, o.orderdate

FROM Sales.Customers AS c JOIN Sales.Orders AS o

ON c.custid = o.custid;

Синтаксис ANSI SQL-92 затрудняет создание случайных декартовых соединений. После добавления ключевого слова JOIN возникнет синтаксическая ошибка, если предложение ON отсутствует.

Demo 1

**Lesson 2 4-7**

**Querying with Inner Joins**

Запросы T-SQL, использующие внутренние соединения, являются наиболее распространенными типами для решения многих бизнес-задач, особенно в сильно нормализованных средах баз данных. Чтобы получить данные, которые были сохранены в нескольких таблицах, вам часто потребуется повторно собрать их с помощью внутренних запросов объединения. Как вы уже узнали, внутреннее соединение начинает свою фазу логической обработки как декартово произведение, которое затем фильтруется, чтобы удалить все строки, которые не соответствуют предикату.

В синтаксисе SQL-89 этот предикат находится в предложении WHERE; в синтаксисе SQL-92 этот предикат находится в предложении FROM в предложении ON:

SQL-89 and SQL-92 Join Syntax Compared

--ANSI SQL-89 syntax

SELECT c.companyname, o.orderdate

FROM Sales.Customers AS c, Sales.Orders AS o

WHERE c.custid = o.custid;

--ANSI SQL-92 syntax

SELECT c.companyname, o.orderdate

FROM Sales.Customers AS c JOIN Sales.Orders AS o

ON c.custid = o.custid;

С точки зрения производительности оптимизатор запросов в SQL Server не отдает предпочтение одному синтаксису перед другим. Рекомендуется использовать новый стандарт

Давайте проследим за обработкой, начиная со строки 2:

Предложение FROM обозначает Продажи.Таблица клиентов в качестве одной из входных таблиц, присвоив ей псевдоним “c”.

Оператор JOIN в строке 3 отражает использование ВНУТРЕННЕГО соединения (тип по умолчанию в T-SQL) и обозначает Продажи.Заказы в качестве другой входной таблицы, которая имеет псевдоним “o”.

SQL Server выполнит логическое декартово объединение этих таблиц и передаст результаты на следующий этап в виртуальной таблице. (Обратите внимание, что физическая обработка запроса может фактически не выполнять операцию декартова произведения, в зависимости от решений оптимизатора.)

Используя предложение ON, SQL Server будет фильтровать виртуальную таблицу, сохраняя только те строки, в которых значение custid из таблицы “c” (Sales.Клиенты были заменены псевдонимом) соответствует идентификатору custid из таблицы “o” (Sales.Заказы были заменены псевдонимом).

Оставшиеся строки остаются в виртуальной таблице и передаются на следующий этап в инструкции SELECT. В этом примере виртуальная таблица затем обрабатывается предложением SELECT, и только два столбца возвращаются в клиентское приложение.

Итог: Список клиентов, разместивших заказы. Любые клиенты, которые никогда не размещали заказ, были отфильтрованы предложением ON, как и любые заказы, у которых есть идентификатор клиента, который не соответствует записи в списке клиентов.

**Inner Join Syntax**

При написании запросов с использованием внутренних соединений учитывайте следующие рекомендации:

Как вы уже видели, псевдонимы таблиц предпочтительны не только для списка ВЫБОРА, но и для выражения предложения ON.

Внутренние соединения могут выполняться с одним соответствующим атрибутом, таким как orderid, или с несколькими соответствующими атрибутами, такими как комбинация orderid и productid. Соединения, соответствующие нескольким атрибутам, называются составными соединениями.

Порядок, в котором таблицы перечислены и объединены в предложении FROM, не имеет значения для оптимизатора SQL Server. (Это не будет иметь место для ВНЕШНИХ запросов СОЕДИНЕНИЯ в следующем разделе.) Концептуально соединения будут оцениваться слева направо.

Используйте ключевое слово JOIN один раз для каждых двух таблиц в списке FROM. Для запроса с двумя таблицами укажите одно соединение. Для запроса с тремя таблицами вы будете использовать JOIN дважды-один раз между первыми двумя таблицами и еще раз между выводом первых двух таблиц и третьей таблицей.

This query performs a join on a single matching attribute, relating the categoryid from the Production.Categories table to the categoryid from the Production.Products table:

SELECT c.categoryid, c.categoryname,

p.productid, p.productname

FROM Production.Categories AS c

JOIN Production.Products AS p

ON c.categoryid = p.categoryid;

Этот запрос выполняет составное объединение двух совпадающих атрибутов, связывая атрибуты города и страны из Продаж.Клиенты в отдел кадров.Работники. Обратите внимание на использование оператора DISTINCT для фильтрации повторяющихся вхождений города, страны:

SELECT DISTINCT e.city, e.country

FROM Sales.Customers AS c

JOIN HR.Employees AS e

ON c.city = e.city AND c.country = e.country;

В следующем примере показано, как внутреннее соединение может быть расширено, чтобы включить более двух таблиц. Обратите внимание, что Продажи.Таблица OrderDetails присоединяется не к продажам.Таблица заказов, но к выходу СОЕДИНЕНИЯ между продажами.Клиенты и продажи.Заказы. Каждый экземпляр JOIN ... ON выполняет собственное заполнение и фильтрацию виртуальной выходной таблицы. Оптимизатор запросов SQL Server определяет порядок, в котором будут выполняться соединения и фильтрация.

SELECT c.custid, c.companyname, o.orderid, o.orderdate, od.productid, od.qty

FROM Sales.Customers AS c

JOIN Sales.Orders AS o

ON c.custid = o.custid

JOIN Sales.OrderDetails AS od

ON o.orderid = od.orderid;

Demo 2

**Lesson 3 4-11**

**Querying with Outer Joins**

Understanding Outer Joins

На предыдущем уроке вы узнали, как использовать внутренние соединения для сопоставления строк в отдельных таблицах. Как вы видели, SQL Server построил результаты запроса внутреннего соединения, отфильтровав строки, которые не соответствовали условиям, выраженным в предикате предложения ON. В результате были отображены только строки, совпадающие из обеих таблиц. С помощью внешнего соединения вы можете выбрать отображение всех строк из одной таблицы, а также тех, которые совпадают со второй таблицей.

SELECT c.CustomerID, soh.SalesOrderID

FROM Sales.Customer AS c JOIN Sales.SalesOrderHeader AS soh

ON c.CustomerID = soh.CustomerID

В результатах будут отображаться только те идентификаторы клиентов, которые находятся в обеих таблицах. Будут возвращены только те клиенты, которые разместили заказы

**Outer Left Join**

SELECT c.CustomerID, soh.SalesOrderID

FROM Sales.Customer AS c LEFT OUTER JOIN Sales.SalesOrderHeader AS soh

ON c.CustomerID = soh.CustomerID

В этом примере используется оператор LEFT OUTER JOIN, который, направляет обработчик запросов на сохранение всех строк из таблицы слева (Sales.Клиент) и отображает значения идентификатора SalesOrderID для совпадающих строк в Sales.SalesOrderHeader. Однако в этом примере возвращается больше строк. Все клиенты возвращаются, независимо от того, сделали они заказ или нет. Как вы увидите в этом уроке, внешнее соединение будет отображать все строки с одной или другой стороны соединения, независимо от того, совпадают они или нет.

Что показывает запрос внешнего соединения в столбцах, где не было совпадения? В этом примере нет совпадающих заказов для 701 клиента. Вместо столбца SalesOrderID SQL Server выведет значение NULL, если в противном случае значения отсутствуют.

**Рекомендации:**

Внешние соединения выражаются с помощью ключевых слов LEFT, RIGHT или FULL, предшествующих ВНЕШНЕМУ СОЕДИНЕНИЮ. Цель ключевого слова-указать, какая таблица (на какой стороне СОЕДИНЕНИЯ ключевых слов) должна быть сохранена и все ее строки должны отображаться, совпадают они или нет.

Как и в случае с внутренними соединениями, внешние соединения могут выполняться с одним соответствующим атрибутом, таким как orderid, или с несколькими соответствующими атрибутами, такими как orderid и productid.

В отличие от внутренних соединений, порядок, в котором таблицы перечислены и объединены в предложении FROM, имеет значение, поскольку он будет определять, выбираете ли вы ЛЕВОЕ или ПРАВОЕ соединение для вашего соединения.

Многопозиционные соединения более сложны, когда присутствует ВНЕШНЕЕ СОЕДИНЕНИЕ. Наличие нулей в результатах внешнего соединения может вызвать проблемы, если промежуточные результаты затем соединяются с помощью внутреннего соединения с третьей таблицей. Строки с нулевыми значениями могут быть отфильтрованы предикатом второго соединения.

Чтобы отображать только строки, в которых нет совпадения, добавьте тест на NULL в предложение WHERE после предиката ВНЕШНЕГО СОЕДИНЕНИЯ.

This query displays all customers and provides information about each of their orders if any exist:

SELECT c.custid, c.companyname, o.orderid,

o.orderdate

FROM Sales.Customers AS c

LEFT OUTER JOIN Sales.Orders AS o

ON c.custid =o.custid;

This query displays only customers who have never placed an order:

SELECT c.custid, c.companyname, o.orderid, o.orderdate

FROM Sales.Customers AS c

LEFT OUTER JOIN Sales.Orders AS o

ON c.custid =o.custid

WHERE o.orderid IS NULL;

Demo 3

**Lesson 4 4-15**

**Querying with Cross Joins and Self Joins**

Understanding Cross Joins

Запросы перекрестного соединения создают декартово произведение, которого, как вы уже узнали в этом модуле, следует избегать. Хотя вы видели способ создать его с синтаксисом ANSI SQL-89, вы не видели, как и зачем это делать с ANSI SQL-92. В этой теме мы вернемся к перекрестным соединениям и декартовым произведениям.

Чтобы явно создать декартово произведение, необходимо использовать оператор ПЕРЕКРЕСТНОГО СОЕДИНЕНИЯ.

This will create a result set with all possible combinations of input rows:

Cross Join

SELECT ...

FROM table1 AS t1 CROSS JOIN table2 AS t2;

Хотя обычно это не является желаемым результатом, существует несколько практических приложений для написания явного перекрестного соединения:

Создание таблицы чисел со строкой для каждого возможного значения в диапазоне.

Создание больших объемов данных для тестирования. При перекрестном соединении с самой собой таблица, содержащая всего 100 строк, может легко генерировать 10 000 выходных строк с очень небольшим количеством работы с вашей стороны.

SELECT e1.firstname, e2.lastname

FROM HR.Employees e1 CROSS JOIN

HR.Employees e2;

**Understanding Self Joins**

До сих пор объединения, включали в себя несколько отдельных таблиц. Могут существовать сценарии, в которых необходимо сравнить и извлечь данные, хранящиеся в одной и той же таблице. Например, в классическом приложении для управления персоналом таблица сотрудников может содержать информацию о руководителе каждого сотрудника в отдельной строке сотрудника. Каждый руководитель также указан в качестве сотрудника. Чтобы получить информацию о сотруднике и сопоставить ее с соответствующим руководителем, вы можете использовать таблицу дважды в своем запросе, присоединив ее к себе для целей запроса.

Существуют и другие сценарии, в которых вы захотите сравнить строки в таблице друг с другом. Как вы видели, довольно легко сравнить столбцы в одной строке с помощью T-SQL, но как сравнить значения из разных строк (например, строка, в которой хранится время начала, с другой строкой в той же таблице, в которой хранится соответствующее время остановки), менее очевидно. Самосоединения являются полезным методом для таких типов запросов.

Для выполнения подобных задач вам следует учитывать следующие рекомендации:

Создайте два экземпляра одной и той же таблицы в предложении FROM и соедините их по мере необходимости, используя внутренние или внешние соединения.

Используйте псевдонимы таблиц, чтобы создать два отдельных псевдонима для одной и той же таблицы. По крайней мере, у одного из них должен быть псевдоним.

Используйте предложение ON для предоставления фильтра с использованием отдельных столбцов из одной таблицы.

Следующий пример, который вы подробно рассмотрите в следующем разделе, иллюстрирует эти рекомендации:

Этот запрос извлекает информацию о сотрудниках и их соответствующих менеджерах из таблицы Сотрудники, присоединенной к самой себе:

SELECT e.empid ,e.lastname AS empname,e.title,e.mgrid, m.lastname AS mgrname

FROM HR.Employees AS e

JOIN HR.Employees AS m

ON e.mgrid=m.empid;

This yields results like the following:

empid empname title mgrid mgrname

----- ------------ --------------------- ----- -------

2 Funk Vice President, Sales 1 Davis

3 Lew Sales Manager 2 Funk

4 Peled Sales Representative 3 Lew

5 Buck Sales Manager 2 Funk

6 Suurs Sales Representative 5 Buck

7 King Sales Representative 5 Buck

8 Cameron Sales Representative 3 Lew

9 Dolgopyatova Sales Representative 5 Buck

The following are some examples of self joins:

This query returns all employees, along with the name of each employee’s manager, when a manager exists (inner join). Note that an employee with no manager listed will be missing from the results:

SELECT e.empid ,e.lastname AS

empname,e.title,e.mgrid, m.lastname AS

mgrname

FROM HR.Employees AS e

JOIN HR.Employees AS m

ON e.mgrid=m.empid;

This query returns all employees with the name of each manager (outer join). This restores the missing employee, who turns out to be a CEO with no manager:

SELECT e.empid ,e.lastname AS empname,e.title,e.mgrid, m.lastname AS mgrname

FROM HR.Employees AS e

LEFT OUTER JOIN HR.Employees AS m

ON e.mgrid=m.empid;

demo