УРОК 12

Объединение таблиц

На этом уроке вы узнаете, что такое объединения, для чего они нужны и как создавать инструкции SELECT, использующие объединения.

Что такое объединение

Одной из ключевых особенностей SQL является возможность на лету объединять таблицы при выполнении запросов, связанных с извлечением данных. Объединения — это самые мощные операции, которые можно выполнить с помощью инструкции SELECT, поэтому понимание объединений и их синтаксиса является важной частью процесса изучения SQL.

Прежде чем вы сможете эффективно применять объединения, следует разобраться, что такое реляционные таблицы и как проектируются реляционные базы данных. В столь маленькой книге полностью осветить такую обширную тему не удастся, но нескольких уроков будет вполне достаточно для того, чтобы вы смогли получить общее представление.

Что такое реляционные таблицы

Понять, что представляют собой реляционные таблицы, поможет пример из реальной жизни.

Предположим, определенная таблица базы данных содержит каталог товаров, в котором для каждого элемента выделена одна строка. Информация, хранящаяся о каждом товаре, должна включать описание товара и его цену, а также сведения о компании, выпустившей данный товар.

Теперь предположим, что в каталоге имеется целая группа товаров от одного поставщика. Где следует хранить информацию о поставщике (такую, как название компании, адрес и контактная информация)? Эти сведения не рекомендуется хранить вместе с данными о товарах по нескольким причинам.

- Информация о поставщике одна и та же для всех его товаров. Повторение этой информации для каждого товара приведет к напрасной потере времени и места на диске.
- Если информация о поставщике изменяется (например, если он переезжает или изменяется его почтовый код), вам придется обновить все записи о его товарах.
- Когда данные повторяются (а такое происходит, когда информация о поставщике указывается для каждого товара), высока вероятность того, что где-то данные будут введены с ошибкой. Несовместимые данные очень трудно использовать при создании отчетов.

Отсюда можно сделать вывод, что хранить множество экземпляров одних и тех же данных крайне нежелательно. Именно этот принцип и лежит в основе реляционных баз данных. Реляционные таблицы разрабатываются таким образом, что вся информация распределяется по множеству таблиц, причем для данных каждого типа создается отдельная таблица. Эти таблицы соотносятся (связываются) между собой через общие поля.

В нашем примере можно создать две таблицы: одну — для хранения информации о поставщике, другую — о его товарах. Таблица `Vendors` содержит информацию о поставщиках, по одной строке для каждого поставщика, с обязательным указанием его уникального идентификатора. Это значение называется *первичным ключом*.

В таблице `Products` хранится только информация о товарах, но нет никакой конкретной информации о поставщиках, за исключением их идентификаторов (первичного ключа таблицы `Vendors`). Этот ключ связывает таблицу `Vendors` с таблицей `Products`. Благодаря применению идентификатора поставщика можно использовать таблицу `Vendors` для поиска информации о соответствующем поставщике.

Что это дает? Ниже указаны ключевые преимущества.

- Информация о поставщике никогда не повторяется, благодаря чему экономится время, требуемое для заполнения базы данных, а также место на диске.
- Если информация о поставщике изменяется, достаточно обновить всего одну запись о нем, единственную в таблице `Vendors`. Данные в связанных с нею таблицах изменять не нужно.

- Поскольку никакие данные не повторяются, они, очевидно, оказываются непротиворечивыми, благодаря чему составление отчетов значительно упрощается.

Таким образом, данные в реляционных таблицах хранятся достаточно эффективно, и ими легко манипулировать. Вот почему реляционные базы данных масштабируются значительно лучше, чем базы данных других типов.

> **Масштабирование**
>
> Возможность справляться со все возрастающей нагрузкой без сбоев. О хорошо спроектированных базах данных или приложениях говорят, что они хорошо масштабируются.

Зачем нужны объединения

Распределение данных по многим таблицам обеспечивает их более эффективное хранение, упрощает обработку данных и повышает масштабируемость базы данных в целом. Однако эти преимущества не достигаются даром — за все приходится платить.

Если данные хранятся во многих таблицах, то как их извлечь с помощью одной инструкции SELECT?

Ответ таков: посредством объединений. *Объединение* представляет собой механизм слияния таблиц в инструкции SELECT. Используя особый синтаксис, можно объединить несколько исходных таблиц в одну общую, которая будет на лету связывать нужные строки из каждой таблицы.

> ПРИМЕЧАНИЕ: **использование интерактивных инструментов СУБД**
>
> Важно понимать, что объединение не является физической таблицей — другими словами, оно не существует как реальная таблица в базе данных. Объединение создается СУБД по мере необходимости и сохраняется только на время выполнения запроса. Многие СУБД предлагают графический интерфейс, который можно использовать для интерактивного определения связей таблицы. Эти инструменты могут оказаться чрезвычайно полезными для поддержания ссылочной целостности.

При использовании реляционных таблиц важно, чтобы в связанные столбцы заносились только корректные данные. Вернемся к нашему примеру: если в таблице Products хранится недостоверный идентификатор поставщика, соответствующие товары окажутся недоступными, поскольку они не будут относиться ни к одному поставщику. Во избежание этого база данных должна позволять пользователю вводить только достоверные значения (т.е. такие, которые представлены в таблице Vendors) в столбце идентификаторов поставщиков в таблице Products. Ссылочная целостность означает, что СУБД заставляет пользователя соблюдать правила, обеспечивающие непротиворечивость данных. И контроль этих правил часто обеспечивается благодаря интерфейсам СУБД.

Создание объединения

Создание объединения — очень простая процедура. Нужно указать все таблицы, которые должны быть включены в объединение, а также подсказать СУБД, как они должны быть связаны между собой. Рассмотрим следующий пример.

Ввод ▼

```
SELECT vend_name, prod_name, prod_price
FROM Vendors, Products
WHERE Vendors.vend_id = Products.vend_id;
```

Вывод ▼

```
vend_name       prod_name           prod_price
--------------- ------------------- ----------
Doll House Inc. Fish bean bag toy   3.4900
Doll House Inc. Bird bean bag toy   3.4900
Doll House Inc. Rabbit bean bag toy 3.4900
Bears R Us      8 inch teddy bear   5.9900
Bears R Us      12 inch teddy bear  8.9900
Bears R Us      18 inch teddy bear  11.9900
Doll House Inc. Raggedy Ann         4.9900
Fun and Games   King doll           9.4900
Fun and Games   Queen doll          9.4900
```

Анализ ▼

Инструкция SELECT начинается точно так же, как и все инструкции, которые мы до сих пор рассматривали, — с указания столбцов, которые должны быть извлечены. Ключевая разница состоит в том, что два из указанных столбцов (prod_name и prod_price) находятся в одной таблице, а третий (vend_name) — в другой.

Взгляните на предложение FROM. В отличие от предыдущих инструкций SELECT, оно содержит две таблицы: Vendors и Products. Это имена двух таблиц, которые должны быть объединены в данном запросе. Таблицы корректно объединяются в предложении WHERE, которое заставляет СУБД связать идентификатор поставщика vend_id из таблицы Vendors с полем vend_id таблицы Products.

Обратите внимание на то, что эти столбцы указаны как Vendors.vend_id и Products.vend_id. Полностью определенные имена необходимы здесь потому, что, если вы укажете только vend_id, СУБД не сможет понять, на какие именно столбцы vend_id вы ссылаетесь (их два, по одному в каждой таблице). Как видно из представленных результатов, одна инструкция SELECT сумела извлечь данные из двух разных таблиц.

> ПРЕДУПРЕЖДЕНИЕ: **полностью определенные имена столбцов**
>
> Используйте полностью определенные имена столбцов (в которых названия таблиц и столбцов разделяются точкой) всякий раз, когда может возникнуть неоднозначность относительно того, на какой столбец вы ссылаетесь. В большинстве СУБД будет выдано сообщение об ошибке, если вы введете неоднозначное имя столбца, не определив его полностью путем указания имени таблицы.

Важность предложения WHERE

Использование предложения WHERE для установления связи между таблицами может показаться странным, но на то есть весомая причина. Вспомните: когда таблицы объединяются в инструкции SELECT, отношение создается на лету. В определениях таблиц базы данных ничего не говорится о том, как СУБД должна объединять их. Вы должны указать это сами. Когда вы объединяете две таблицы,

то в действительности создаете пары, состоящие из каждой строки первой таблицы и каждой строки второй таблицы. Предложение WHERE действует как фильтр, позволяющий включать в результат только строки, которые соответствуют указанному условию фильтрации — в данном случае условию объединения. Без предложения WHERE каждая строка в первой таблице будет образовывать пару с каждой строкой второй таблицы независимо от того, есть ли логика в их объединении или нет.

Декартово произведение

Результаты, возвращаемые при слиянии таблиц без указания условия объединения. Количество полученных строк будет равно числу строк в первой таблице, умноженному на число строк во второй таблице.

Для того чтобы разобраться в этом, рассмотрим следующую инструкцию SELECT и результат ее выполнения.

Ввод ▼

```
SELECT vend_name, prod_name, prod_price
FROM Vendors, Products;
```

Вывод ▼

```
vend_name       prod_name           prod_price
--------------- ------------------- ----------
Bears R Us      8 inch teddy bear   5.99
Bears R Us      12 inch teddy bear  8.99
Bears R Us      18 inch teddy bear  11.99
Bears R Us      Fish bean bag toy   3.49
Bears R Us      Bird bean bag toy   3.49
Bears R Us      Rabbit bean bag toy 3.49
Bears R Us      Raggedy Ann         4.99
Bears R Us      King doll           9.49
Bears R Us      Queen doll          9.49
Bear Emporium   8 inch teddy bear   5.99
Bear Emporium   12 inch teddy bear  8.99
Bear Emporium   18 inch teddy bear  11.99
Bear Emporium   Fish bean bag toy   3.49
Bear Emporium   Bird bean bag toy   3.49
Bear Emporium   Rabbit bean bag toy 3.49
```

```
Bear Emporium       Raggedy Ann          4.99
Bear Emporium       King doll            9.49
Bear Emporium       Queen doll           9.49
Doll House Inc.     8 inch teddy bear    5.99
Doll House Inc.     12 inch teddy bear   8.99
Doll House Inc.     18 inch teddy bear   11.99
Doll House Inc.     Fish bean bag toy    3.49
Doll House Inc.     Bird bean bag toy    3.49
Doll House Inc.     Rabbit bean bag toy  3.49
Doll House Inc.     Raggedy Ann          4.99
Doll House Inc.     King doll            9.49
Doll House Inc.     Queen doll           9.49
Furball Inc.        8 inch teddy bear    5.99
Furball Inc.        12 inch teddy bear   8.99
Furball Inc.        18 inch teddy bear   11.99
Furball Inc.        Fish bean bag toy    3.49
Furball Inc.        Bird bean bag toy    3.49
Furball Inc.        Rabbit bean bag toy  3.49
Furball Inc.        Raggedy Ann          4.99
Furball Inc.        King doll            9.49
Furball Inc.        Queen doll           9.49
Fun and Games       8 inch teddy bear    5.99
Fun and Games       12 inch teddy bear   8.99
Fun and Games       18 inch teddy bear   11.99
Fun and Games       Fish bean bag toy    3.49
Fun and Games       Bird bean bag toy    3.49
Fun and Games       Rabbit bean bag toy  3.49
Fun and Games       Raggedy Ann          4.99
Fun and Games       King doll            9.49
Fun and Games       Queen doll           9.49
Jouets et ours      8 inch teddy bear    5.99
Jouets et ours      12 inch teddy bear   8.99
Jouets et ours      18 inch teddy bear   11.99
Jouets et ours      Fish bean bag toy    3.49
Jouets et ours      Bird bean bag toy    3.49
Jouets et ours      Rabbit bean bag toy  3.49
Jouets et ours      Raggedy Ann          4.99
Jouets et ours      King doll            9.49
Jouets et ours      Queen doll           9.49
```

Анализ ▼

Как видно из представленных результатов, декартово произведение вы, скорее всего, будете использовать очень редко. Данные, полученные таким способом, ставят в соответствие каждому товару

каждого поставщика, включая товары с указанием "не того" поставщика (и даже поставщиков, которые вообще не предлагают никаких товаров).

> ПРЕДУПРЕЖДЕНИЕ: **не забудьте указать предложение WHERE**
>
> Проверьте, включили ли вы в запрос предложение WHERE, иначе СУБД вернет намного больше данных, чем вам нужно. Кроме того, убедитесь в том, что предложение WHERE сформулировано правильно. Некорректное условие фильтрации приведет к тому, что СУБД выдаст неверные данные.

> **Перекрестное объединение**
>
> Иногда объединение, которое возвращает декартово произведение, называют перекрестным объединением.

Внутренние объединения

Объединение, которое мы до сих пор использовали, называется объединением по равенству — оно основано на проверке равенства записей двух таблиц. Объединение такого типа называют также *внутренним объединением*. Для подобных объединений можно применять несколько иной синтаксис, явно указывающий на тип объединения. Следующая инструкция SELECT возвращает те же самые данные, что и в предыдущем примере.

Ввод ▼

```
SELECT vend_name, prod_name, prod_price
FROM Vendors INNER JOIN Products
  ON Vendors.vend_id = Products.vend_id;
```

Анализ ▼

Предложение SELECT здесь точно такое же, как и в предыдущем случае, а вот предложение FROM другое. В данном запросе отношение между двумя таблицами определяется в предложении FROM,

содержащем спецификацию INNER JOIN. При использовании такого синтаксиса условие объединения задается с помощью специального предложения ON, а не WHERE. Фактическое условие, указываемое в предложении ON, то же самое, которое задавалось бы в предложение WHERE.

Обратитесь к документации своей СУБД, чтобы узнать, какой синтаксис предпочтительнее использовать.

> ПРИМЕЧАНИЕ: **"правильный" синтаксис**
>
> Согласно спецификации ANSI SQL, предпочтителен синтаксис INNER JOIN. В то же время большинство СУБД поддерживает оба синтаксиса. Изучите оба формата и применяйте тот из них, который кажется вам более удобным.

Объединение нескольких таблиц

SQL не ограничивает число таблиц, которые могут быть объединены посредством инструкции SELECT. Основные правила создания объединения остаются теми же. Вначале перечисляются все таблицы, а затем определяются отношения между ними. Рассмотрим пример.

Ввод ▼

```
SELECT prod_name, vend_name, prod_price, quantity
FROM OrderItems, Products, Vendors
WHERE Products.vend_id = Vendors.vend_id
  AND OrderItems.prod_id = Products.prod_id
  AND order_num = 20007;
```

Вывод ▼

```
prod_name          vend_name       prod_price  quantity
-----------------  --------------  ----------  --------
18 inch teddy bear Bears R Us      11.9900     50
Fish bean bag toy  Doll House Inc. 3.4900      100
Bird bean bag toy  Doll House Inc. 3.4900      100
Rabbit bean bag toy Doll House Inc. 3.4900     100
Raggedy Ann        Doll House Inc. 4.9900      50
```

Анализ ▼

В этом примере выводятся элементы заказа номер 20007. Все они находятся в таблице OrderItems. Каждый элемент хранится в соответствии с идентификатором, который ссылается на товар в таблице Products. Эти товары связаны с соответствующими поставщиками в таблице Vendors по идентификатору поставщика, который хранится вместе с каждой записью о товаре. В предложении FROM данного запроса перечисляются три таблицы, а предложение WHERE определяет оба названных условия объединения. Дополнительное условие служит для фильтрации только элементов заказа 20007.

> ПРЕДУПРЕЖДЕНИЕ: **к вопросу о производительности**
>
> Все СУБД обрабатывают объединения динамически, затрачивая время на обработку каждой указанной таблицы. Этот процесс может оказаться очень ресурсоемким, поэтому не следует использовать объединения таблиц без особой надобности. Чем больше таблиц вы объединяете, тем ниже производительность.

> ПРЕДУПРЕЖДЕНИЕ: **максимальное число таблиц в объединении**
>
> Несмотря на то что SQL не накладывает каких-либо ограничений на число таблиц в объединении, многие СУБД на самом деле имеют такие ограничения. Обратитесь к документации своей СУБД, чтобы узнать, какие ограничения такого рода она налагает (если они есть).

Теперь самое время вернуться к примеру из урока 11, в котором инструкция SELECT возвращала список клиентов, заказавших товар RGAN01.

Ввод ▼

```
SELECT cust_name, cust_contact
FROM Customers
WHERE cust_id IN (SELECT cust_id
                  FROM Orders
                  WHERE order_num IN (SELECT order_num
                                      FROM OrderItems
                                      WHERE prod_id =
                                            'RGAN01'));
```

Анализ ▼

Как упоминалось на уроке 11, подзапросы не всегда являются самым эффективным способом выполнения сложных инструкций SELECT, поэтому тот же самый запрос можно переписать с использованием синтаксиса объединений.

Ввод ▼

```
SELECT cust_name, cust_contact
FROM Customers, Orders, OrderItems
WHERE Customers.cust_id = Orders.cust_id
  AND OrderItems.order_num = Orders.order_num
  AND prod_id = 'RGAN01';
```

Вывод ▼

```
cust_name          cust_contact
-------------------  -------------------
Fun4All            Denise L. Stephens
The Toy Store      Kim Howard
```

Анализ ▼

Как уже говорилось на уроке 11, получение необходимых для этого запроса данных требует обращения к трем таблицам. Однако вместо подзапросов здесь были применены два объединения таблиц, а также указаны три условия WHERE. Первые два связывают таблицы в объединение, а последнее фильтрует данные по товару RGAN01.

> СОВЕТ: **экспериментируйте**
>
> Как видите, часто существует несколько способов выполнения одного и того же SQL-запроса. И редко удается однозначно сказать, какой из них правильный. Производительность может зависеть от типа операции, используемой СУБД, количества данных в таблицах, наличия либо отсутствия индексов и ключей, а также целого ряда других критериев. Следовательно, зачастую бывает целесообразно поэкспериментировать с различными типами запросов для выяснения того, какой из них работает быстрее.

Резюме

Объединения — одно из самых важных и востребованных средств SQL, но их эффективное применение возможно только на основе знаний о структуре реляционной базы данных. На этом уроке вы ознакомились с основами построения баз данных, а также узнали, как создавать объединение по равенству (называемое также внутренним объединением), которое используют чаще всего. На следующем уроке вы научитесь создавать объединения других типов.

УРОК 13

Создание расширенных объединений

На этом уроке вы узнаете о дополнительных типах объединений — что они собой представляют и когда они нужны. Вы также узнаете, как применять псевдонимы таблиц и использовать итоговые функции совместно с объединениями.

Использование псевдонимов таблиц

На уроке 7 вы узнали, как использовать псевдонимы в качестве ссылок на извлекаемые столбцы таблицы. Синтаксис псевдонимов столбцов выглядит следующим образом.

Ввод ▼

```
SELECT RTRIM(vend_name) + ' (' + RTRIM(vend_country) + ')'
       AS vend_title
FROM Vendors
ORDER BY vend_name;
```

Псевдонимы можно применять не только для имен столбцов и вычисляемых полей, но и вместо имен таблиц. На то есть две основные причины:

- сокращение синтаксиса запросов;
- возможность многоразового использования одной и той же таблицы в инструкции SELECT.

Рассмотрим следующую инструкцию SELECT. В основном она такая же, как и в примерах предыдущего урока, но здесь она модифицирована с учетом псевдонимов.

Ввод ▼

```
SELECT cust_name, cust_contact
FROM Customers AS C, Orders AS O, OrderItems AS OI
WHERE C.cust_id = O.cust_id
  AND OI.order_num = O.order_num
  AND prod_id = 'RGAN01';
```

Анализ ▼

Заметьте, что все три таблицы в предложениях FROM имеют псевдонимы. Например, выражение Customers AS C задает C в качестве псевдонима для таблицы Customers, что позволяет использовать сокращение C вместо полного имени Customers. В данном примере псевдонимы таблиц указаны только в предложении WHERE, но их можно применять и в других местах, например в списке извлекаемых таблиц, в предложении ORDER BY, а также в любой другой части инструкции SELECT.

> СОВЕТ: **в Oracle нет ключевого слова AS**
>
> Oracle не поддерживает ключевое слово AS. Чтобы создать псевдоним в Oracle, просто укажите его без ключевого слова AS, например Customers C вместо Customers AS C.

Следует отметить, что псевдонимы таблиц существуют только во время выполнения запроса. В отличие от псевдонимов столбцов, они никогда не сообщаются клиенту.

Объединения других типов

До сих пор мы применяли только простые объединения, которые называются внутренними. Теперь рассмотрим три других типа объединения: самообъединение, естественное объединение и внешнее объединение.

Самообъединения

Одна из основных причин для использования псевдонимов таблиц состоит в возможности обращения к одной и той же таблице несколько раз в одной инструкции SELECT. Продемонстрируем это на примере.

Предположим, вы хотите послать письма по всем контактным адресам клиентов, которые работают с той же компанией, с которой работает Джим Джонс. Такой запрос требует, чтобы вначале вы выяснили, с какой компанией работает Джим Джонс, а затем — какие клиенты работают с этой же компанией. Один из способов решения данной задачи приведен ниже.

Ввод ▼

```
SELECT cust_id, cust_name, cust_contact
FROM Customers
WHERE cust_name = (SELECT cust_name
                   FROM Customers
                   WHERE cust_contact = 'Jim Jones');
```

Вывод ▼

```
cust_id        cust_name      cust_contact
----------     -------------  -------------
1000000003     Fun4All        Jim Jones
1000000004     Fun4All        Denise L. Stephens
```

Анализ ▼

В первом решении используются подзапросы. Внутренняя инструкция SELECT возвращает название компании (cust_name), с которой работает Джим Джонс. Именно это название используется в предложении WHERE внешнего запроса, благодаря чему извлекаются имена всех служащих, работающих с данной компанией. (О подзапросах см. урок 11.)

Теперь рассмотрим тот же самый запрос, но с использованием объединений.

Ввод ▼

```
SELECT c1.cust_id, c1.cust_name, c1.cust_contact
FROM Customers AS c1, Customers AS c2
WHERE c1.cust_name = c2.cust_name
  AND c2.cust_contact = 'Jim Jones';
```

Вывод ▼

```
cust_id          cust_name      cust_contact
----------       -----------    --------------
1000000003       Fun4All        Jim Jones
1000000004       Fun4All        Denise L. Stephens
```

> СОВЕТ: **в Oracle нет ключевого слова AS**
>
> Пользователи Oracle, не забывайте убирать из своих инструкций ключевое слово AS.

Анализ ▼

Две таблицы, необходимые для выполнения запроса, на самом деле являются одной и той же таблицей, поэтому таблица Customers появляется в предложении FROM дважды. И хотя это совершенно законно, любые ссылки на таблицу Customers оказались бы неоднозначными, потому что СУБД не знает, на какую именно таблицу Customers вы ссылаетесь.

Для решения данной проблемы и предназначены псевдонимы. Первый раз для таблицы Customers назначается псевдоним c1, а второй раз — псевдоним c2. Теперь эти псевдонимы можно применять в качестве имен таблиц. В частности, инструкция SELECT использует префикс c1 для однозначного указания полного имени нужного столбца. Если этого не сделать, СУБД выдаст сообщение об ошибке, потому что имеется по два столбца с именами cust_id, cust_name и cust_contact. СУБД не может знать, какой именно столбец вы имеете в виду (даже если в действительности это один и тот же столбец). Первое предложение WHERE объединяет обе копии таблицы, а затем фильтрует данные второй таблицы по столбцу cust_contact, чтобы вернуть только нужные данные.

СОВЕТ: **самообъединения вместо подзапросов**

Самообъединения часто применяются для замены инструкций с подзапросами, которые извлекают данные из той же таблицы, что и внешняя инструкция. Несмотря на то что конечный результат получается тем же самым, многие СУБД обрабатывают объединения гораздо быстрее, чем подзапросы. Стоит поэкспериментировать с тем и другим, чтобы определить, какой запрос работает быстрее.

Естественные объединения

Всякий раз, когда объединяются таблицы, по крайней мере один столбец будет появляться более чем в одной таблице (по нему и выполняется объединение). Обычные объединения (внутренние, которые мы рассмотрели на предыдущем уроке) возвращают все данные, даже многократные вхождения одного и того же столбца. Естественное объединение просто удаляет эти многократные вхождения, и в результате возвращается только один столбец.

Естественным называется объединение, в котором извлекаются только не повторяющиеся столбцы. Обычно это делается с помощью метасимвола (`SELECT *`) для одной таблицы и указания явного подмножества столбцов для всех остальных таблиц. Рассмотрим пример.

Ввод ▼

```
SELECT C.*, O.order_num, O.order_date,
       OI.prod_id, OI.quantity, OI.item_price
FROM Customers AS C, Orders AS O,
     OrderItems AS OI
WHERE C.cust_id = O.cust_id
  AND OI.order_num = O.order_num
  AND prod_id = 'RGAN01';
```

СОВЕТ: **в Oracle нет ключевого слова AS**

Пользователи Oracle, не забывайте убирать из кода ключевое слово `AS`.

Анализ ▼

В этом примере метасимвол * используется только для первой таблицы. Все остальные столбцы указаны явно, поэтому никакие дубликаты столбцов не извлекаются.

В действительности каждое внутреннее объединение, которое мы использовали до сих пор, представляло собой естественное объединение, и, возможно, вам никогда не понадобится внутреннее объединение, не являющееся естественным.

Внешние объединения

Большинство объединений связывают строки одной таблицы со строками другой, но в некоторых случаях вам может понадобиться включать в результат строки, не имеющие пар. Например, объединения можно использовать для решения следующих задач:

- подсчет количества заказов каждого клиента, включая клиентов, которые еще не сделали заказ;
- составление перечня товаров с указанием количества заказов на них, включая товары, которые никем не были заказаны;
- вычисление средних объемов продаж с учетом клиентов, которые еще не сделали заказ.

В каждом из этих случаев объединение должно включать строки, не имеющие ассоциированных с ними строк в связанной таблице. Объединение такого типа называется внешним.

> ПРЕДУПРЕЖДЕНИЕ: **различия в синтаксисе**
>
> Важно отметить, что синтаксис внешнего объединения может несколько отличаться в разных реализациях SQL. Различные формы синтаксиса, описанные далее, охватывают большинство реализаций, но все же, прежде чем начинать работу, обратитесь к документации своей СУБД и уточните, какой синтаксис необходимо применять.

Следующая инструкция SELECT позволяет выполнить простое внутреннее объединение. Она извлекает список всех клиентов и их заказы.

Ввод ▼

```
SELECT Customers.cust_id, Orders.order_num
FROM Customers INNER JOIN Orders
  ON Customers.cust_id = Orders.cust_id;
```

Синтаксис внешнего объединения похож на этот. Для получения имен всех клиентов, включая тех, которые еще не сделали заказов, можно сделать следующее.

Ввод ▼

```
SELECT Customers.cust_id, Orders.order_num
FROM Customers LEFT OUTER JOIN Orders
  ON Customers.cust_id = Orders.cust_id;
```

Вывод ▼

```
cust_id       order_num
----------    ---------
1000000001    20005
1000000001    20009
1000000002    NULL
1000000003    20006
1000000004    20007
1000000005    20008
```

Анализ ▼

Аналогично внутреннему объединению, которое мы рассматривали на прошлом уроке, в этой инструкции SELECT используется спецификация OUTER JOIN для указания типа объединения (в предложении FROM, а не WHERE). Но, в отличие от внутренних объединений, которые связывают строки двух таблиц, внешние объединения включают в результат также строки, не имеющие пар. При использовании спецификации OUTER JOIN необходимо указать ключевое слово RIGHT или LEFT, чтобы определить таблицу, все строки которой будут включены в результаты запроса (RIGHT для таблицы, имя которой стоит справа от OUTER JOIN, и LEFT — для той, имя которой значится слева). В предыдущем примере используется спецификация LEFT OUTER JOIN для извлечения всех строк таблицы, указанной в левой части предложения FROM (таблицы Customers).

Чтобы извлечь все строки из таблицы, указанной справа, используйте правое внешнее объединение (RIGHT OUTER JOIN), как показано в следующем примере.

Ввод ▼

```
SELECT Customers.cust_id, Orders.order_num
FROM Customers RIGHT OUTER JOIN Orders
  ON Orders.cust_id = Customers.cust_id;
```

> ПРЕДУПРЕЖДЕНИЕ: **внешние объединения в SQLite**
>
> SQLite поддерживает левое внешнее объединение, но не правое. К счастью, существует очень простое решение, объясняемое в следующем совете.

> СОВЕТ: **типы внешних объединений**
>
> Существуют две основные формы внешнего объединения: левое и правое. Единственная разница между ними состоит в порядке указания связываемых таблиц. Другими словами, левое внешнее объединение может быть превращено в правое просто за счет изменения порядка указания имен таблиц в предложении FROM или WHERE. А раз так, то эти два типа внешнего объединения могут заменять друг друга, и решение о том, какое именно из них нужно использовать, определяется личными предпочтениями.

Существует и другой вариант внешнего объединения — полное внешнее объединение, которое извлекает все строки из обеих таблиц и связывает между собой те, которые могут быть связаны. В отличие от левого и правого внешних объединений, которые включают в результат несвязанные строки только из одной таблицы, полное внешнее объединение включает в результат несвязанные строки из обеих таблиц. Синтаксис полного внешнего объединения таков.

Ввод ▼

```
SELECT Customers.cust_id, Orders.order_num
FROM Orders FULL OUTER JOIN Customers
  ON Orders.cust_id = Customers.cust_id;
```

> ПРЕДУПРЕЖДЕНИЕ: **поддержка полного внешнего объединения**
>
> Синтаксис `FULL OUTER JOIN` не поддерживается в Access, MariaDB, MySQL, OpenOffice Base и SQLite.

Использование объединений совместно с итоговыми функциями

Как было показано на уроке 9, итоговые функции служат для получения базовых статистических показателей. Во всех рассмотренных до сих пор примерах итоговые функции применялись только для одной таблицы, но их можно использовать и по отношению к объединениям.

Рассмотрим пример. Допустим, вы хотите получить список всех клиентов и число сделанных ими заказов. Для этого в следующем запросе применяется функция `COUNT()`.

Ввод ▼

```
SELECT Customers.cust_id,
       COUNT(Orders.order_num) AS num_ord
FROM Customers INNER JOIN Orders
  ON Customers.cust_id = Orders.cust_id
GROUP BY Customers.cust_id;
```

Вывод ▼

```
cust_id      num_ord
----------   --------
1000000001   2
1000000003   1
1000000004   1
1000000005   1
```

Анализ ▼

В этой инструкции используется спецификация `INNER JOIN` для связи таблиц `Customers` и `Orders` между собой. Предложение

GROUP BY группирует данные по клиентам, и, таким образом, вызов функции COUNT(Orders.order_num) позволяет подсчитать количество заказов для каждого клиента и вернуть результат в виде столбца num_ord.

Итоговые функции можно также использовать с объединениями других типов.

Ввод ▼

```
SELECT Customers.cust_id,
       COUNT(Orders.order_num) AS num_ord
FROM Customers LEFT OUTER JOIN Orders
  ON Customers.cust_id = Orders.cust_id
GROUP BY Customers.cust_id;
```

> СОВЕТ: **в Oracle нет ключевого слова AS**
>
> Еще раз напоминаю пользователям Oracle о необходимости удаления ключевого слова AS из кода запроса.

Вывод ▼

```
cust_id       num_ord
----------    -------
1000000001    2
1000000002    0
1000000003    1
1000000004    1
1000000005    1
```

Анализ ▼

В этом примере используется левое внешнее объединение для включения в результат всех клиентов, даже тех, которые не сделали ни одного заказа. Как видите, клиент 1000000002 также включен в список, хотя на данный момент у него ноль заказов.

Правила создания объединений

Прежде чем завершить обсуждение объединений, которое заняло два урока, имеет смысл напомнить о ключевых моментах, касающихся объединений.

- Будьте внимательны при выборе типа объединения. Возможно, что чаще вы будете применять внутреннее объединение, хотя в зависимости от ситуации это может быть и внешнее объединение.
- Посмотрите в документации к СУБД, какой именно синтаксис объединений она поддерживает. (Большинство СУБД поддерживает одну из форм синтаксиса, описанных на этих двух уроках.)
- Проверьте, правильно ли указано условие объединения (независимо от используемого синтаксиса), иначе будут получены неверные данные.
- Не забывайте указывать условие объединения, в противном случае вы получите декартово произведение таблиц.
- Можно включать в объединение несколько таблиц и даже применять для каждой из них свой тип объединения. Несмотря на то что это допустимо и часто оказывается полезным, желательно проверить каждое объединение отдельно, прежде чем применять их вместе. Это намного упростит поиск ошибок.

Резюме

Этот урок стал продолжением предыдущего, посвященного объединениям. Вначале было показано, как и для чего используют псевдонимы, а затем мы продолжили рассмотрение объединений различных типов и вариантов синтаксиса для каждого из них. Вы также узнали, как применять итоговые функции совместно с объединениями и какие правила важно соблюдать при использовании объединений.