Лекция . Запросы модификации данных

Язык *SQL* ориентирован на выполнение операций над группами записей, хотя в некоторых случаях их можно проводить и над отдельной записью.

Запросы действия представляют собой достаточно мощное средство, так как позволяют оперировать не только отдельными строками, но и набором строк. С помощью *запросов действия пользователь* может добавить, удалить или обновить блоки данных. Существует три вида *запросов действия*:

- `INSERT INTO` – *запрос добавления* ;
- `DELETE` – *запрос удаления* ;
- `UPDATE` – *запрос обновления*.

Запрос добавления

Оператор `INSERT` применяется для *добавления записей* в таблицу. Формат оператора:

```
<оператор_вставки>::=INSERT INTO <имя_таблицы>
    [(имя_столбца [,...n])]
    {VALUES (значение[,...n])|
    <SELECT_оператор>}
```

Здесь `параметр имя_таблицы` представляет собой либо *имя таблицы базы данных*, либо имя обновляемого представления.

Первая форма оператора `INSERT` с параметром `VALUES` предназначена для вставки единственной строки в указанную таблицу. *Список* столбцов указывает столбцы, которым будут присвоены значения в добавляемых записях. *Список* может быть опущен, тогда подразумеваются все столбцы таблицы (кроме объявленных как *счетчик*), причем в определенном порядке, установленном при создании таблицы. Если в операторе `INSERT` указывается конкретный *список* имен полей, то любые пропущенные в нем столбцы должны быть объявлены при создании таблицы как допускающие *значение* `NULL` , за исключением тех случаев, когда при описании столбца использовался *параметр* `DEFAULT` . *Список* значений должен следующим образом соответствовать списку столбцов:

- количество элементов в обоих списках должно быть одинаковым;
- должно существовать прямое соответствие между позицией одного и того же элемента в обоих списках, поэтому первый элемент списка значений должен относиться к первому столбцу в списке столбцов, второй – ко второму столбцу и т.д.
- типы данных элементов в списке значений должны быть совместимы с типами данных соответствующих столбцов таблицы.

Пример 8.1. Добавить в таблицу `TOBAP` новую *запись*.

```
INSERT INTO Товар (Название, Тип, Цена)
    VALUES(" Славянский ", " шоколад ", 12)
```

```
Пример 8.1. Добавление в таблицу TOBAP новой записи.
```

Если столбцы таблицы `TOBAP` указаны в полном составе и в том порядке, в котором они перечислены при создании таблицы `TOBAP` , оператор можно упростить.

```
INSERT INTO Товар VALUES (" Славянский ",
    " шоколад ", 12)
```

Вторая форма оператора `INSERT` с параметром `SELECT` позволяет скопировать множество строк из одной таблицы в другую. Предложение `SELECT` может представлять собой любой допустимый оператор `SELECT` . Вставляемые в указанную таблицу строки в точности должны соответствовать строкам результирующей таблицы, созданной при выполнении вложенного запроса. Все ограничения, указанные выше для первой формы оператора `INSERT` , применимы и в этом случае.

Поскольку оператор `SELECT` в общем случае возвращает множество записей, то оператор `INSERT` в такой форме приводит к *добавлению* в таблицу аналогичного числа новых записей.

Пример 8.2. Добавить в итоговую таблицу сведения об общей сумме ежемесячных продаж каждого наименования товара.

```
INSERT INTO Итог
   (Название, Месяц, Стоимость )
SELECT Товар.Название, Month(Сделка.Дата)
   AS Месяц, Sum(Товар.Цена*Сделка.Количество)
   AS Стоимость
FROM Товар INNER JOIN Сделка
   ON Товар.КодТовара= Сделка.КодТовара
GROUP BY Товар.Название, Month(Сделка.Дата)
```

Пример 8.2. Добавление в итоговую таблицу сведений об общей сумме ежемесячных продаж каждого наименования товара.

Запрос удаления

Оператор `DELETE` предназначен для *удаления группы записей* из таблицы.

Формат оператора:

```
<оператор_удаления> ::=DELETE
    FROM <имя_таблицы>[WHERE <условие_отбора>]
```

Здесь *параметр* `имя_таблицы` представляет собой либо *имя таблицы базы данных*, либо имя обновляемого представления.

Если предложение `WHERE` присутствует, удаляются записи из таблицы, удовлетворяющие условию отбора. Если опустить предложение `WHERE` , из таблицы будут *удалены все записи*, однако сама *таблица* сохранится.

Пример 8.3. Удалить все прошлогодние сделки.

```
DELETE
FROM Сделка
WHERE Year(Сделка.Дата)=Year(GETDATE())-1
```

Пример 8.3. Удаление всех прошлогодних сделок.

В приведенном примере условие отбора формируется с учетом года (*функция* `Year`) от текущей даты (*функция* `GETDATE()`).

Запрос обновления

Оператор `UPDATE` применяется для *изменения* значений в группе записей или в одной записи указанной таблицы.

Формат оператора:

```
<оператор_изменения> ::=
    UPDATE имя_таблицы SET имя_столбца=
        <выражение>[,...n]
    [WHERE <условие_отбора>]
```

Параметр `имя_таблицы` – это либо *имя таблицы базы данных*, либо имя обновляемого представления. В предложении `SET` указываются имена одного и более столбцов, данные в которых необходимо изменить. Предложение `WHERE` является необязательным. Если оно опущено, значения указанных столбцов будут изменены во всех строках таблицы. Если предложение `WHERE` присутствует, то обновлены будут только те строки, которые удовлетворяют условию отбора. *Выражение* представляет собой новое *значение* соответствующего столбца и должно быть совместимо с ним по типу данных.

Пример 8.4. Для товаров первого сорта установить цену в *значение* 140 и *остаток* – в *значение* 20 единиц.

```
UPDATE Товар SET Товар.Цена=140, Товар.Остаток=20
WHERE Товар.Сорт=" Первый "
```

Пример 8.4. Обновление выбранных записей.

Пример 8.5. Увеличить цену товаров первого сорта на 25%.

```
UPDATE Товар SET Товар.Цена=Товар.Цена*1.25
WHERE Товар.Сорт=" Первый "
```

Пример 8.5. Обновление выбранных записей.

Пример 8.6. В сделке с максимальным количеством товара увеличить число товаров на 10%.

```
UPDATE Сделка SET Сделка.Количество=
    Сделка.Количество*1.1
WHERE Сделка.Количество=
    (SELECT Max(Сделка.Количество) FROM Сделка)
```

Пример 8.6. Обновление выбранных записей.

Введение в понятие "целостность данных"

Выполнение операторов модификации данных в таблицах *базы данных* `INSERT` , `DELETE` и `UPDATE` может привести к нарушению *целостности данных* и их корректности, т.е. к потере их достоверности и непротиворечивости.

Чтобы *информация*, хранящаяся в базе данных, была однозначной и непротиворечивой, в реляционной модели устанавливаются некоторые ограничительные условия – правила, определяющие возможные значения данных и обеспечивающие логическую основу для поддержания корректных значений. *Ограничения целостности* позволяют свести к минимуму ошибки, возникающие при обновлении и обработке данных.

В базе данных, построенной на реляционной модели, задается ряд правил целостности, которые, по сути, являются ограничениями для всех допустимых состояний *базы данных* и гарантируют *корректность* данных. Рассмотрим следующие *типы ограничений целостности данных*:

обязательные данные;

ограничения для доменов полей;

корпоративные ограничения;

целостность сущностей;

ссылочная целостность.

Обязательные данные

Некоторые поля всегда должны содержать одно из допустимых значений, другими словами, эти поля не могут иметь пустого значения.

Ограничения для доменов полей

Каждое поле имеет свой домен, представляющий собой набор его допустимых значений.

Корпоративные ограничения целостности

Существует понятие "корпоративные *ограничения целостности* " как дополнительные правила поддержки *целостности данных*, определяемые пользователями, принятые на предприятии или администраторами баз данных. Ограничения предприятия называются бизнес-правилами.

Целостность сущностей

Это *ограничение целостности* касается *первичных ключей* базовых таблиц. По определению, ***первичный ключ*** – минимальный идентификатор (одно или несколько полей), который используется для уникальной идентификации записей в таблице. Таким образом, никакое подмножество *первичного ключа* не может быть достаточным для уникальной идентификации записей.

Целостность сущностей определяет, что в базовой таблице ни одно поле *первичного ключа* не может содержать отсутствующих значений, обозначенных `NULL` .

Если допустить присутствие определителя `NULL` в любой части *первичного ключа*, это равносильно утверждению, что не все его поля необходимы для уникальной идентификации записей, и противоречит определению *первичного ключа*.

Ссылочная целостность

Указанное *ограничение целостности* касается *внешних ключей*. ***Внешний ключ*** – это поле (или множество полей) одной таблицы, являющееся ключом другой (или той же самой) таблицы. *Внешние*

ключи используются для установления логических связей между таблицами. Связь устанавливается путем присвоения значений *внешнего ключа* одной таблицы значениям ключа другой.

Между двумя или более таблицами базы данных могут существовать *отношения подчиненности*, которые определяют, что для каждой записи *главной таблицы* (называемой еще ***родительской***) может существовать одна или несколько записей в *подчиненной таблице* (называемой также ***дочерней***).

Существует три разновидности связи между таблицами базы данных:

"один-ко-многим";

"один-к-одному";

"многие-ко-многим".

Отношение "один-ко-многим" имеет место, когда одной записи *родительской таблицы* может соответствовать несколько записей *дочерней*. Связь "один-ко-многим" иногда называют связью "многие-к-одному". И в том, и в другом случае сущность связи между таблицами остается неизменной.

Связь "один-ко-многим" наиболее распространена для реляционных баз данных. Она позволяет моделировать также иерархические структуры данных.

Отношение "один-к-одному" имеет место, когда одной записи в *родительской таблице* соответствует одна запись в *дочерней*. Это отношение встречается намного реже, чем отношение "один-ко-многим". Его используют, если не хотят, чтобы таблица БД "распухала" от второстепенной информации. Использование связи "один-к-одному" приводит к тому, что для чтения связанной информации в нескольких таблицах приходится производить несколько операций чтения вместо одной, когда данные хранятся в одной таблице.

Отношение "многие-ко-многим" имеет место в следующих случаях:

одной записи в *родительской таблице* соответствует более одной записи в *дочерней таблице* ;

одной записи в *дочерней таблице* соответствует более одной записи в *родительской таблице*.

Считается, что всякая связь "многие-ко-многим" может быть заменена на связь "один-ко-многим" (одну или несколько).

Часто связь между таблицами устанавливается по *первичному ключу*, т.е. значениям *внешнего ключа* одной таблицы присваиваются значения *первичного ключа* другой. Однако это не является обязательным – в общем случае связь может устанавливаться и с помощью вторичных ключей. Кроме того, при установлении связей между таблицами не требуется непременная уникальность ключа, обеспечивающего установление связи. Поля *внешнего ключа* не обязаны иметь те же имена, что и имена ключей, которым они соответствуют. *Внешний ключ* может ссылаться на свою собственную таблицу – в таком случае *внешний ключ* называется рекурсивным.

Ссылочная целостность определяет: если в таблице существует *внешний ключ*, то его значение должно либо соответствовать значению *первичного ключа* некоторой записи в базовой таблице, либо задаваться определителем `NULL` .

Существует несколько важных моментов, связанных с *внешними ключами*. Во-первых, следует проанализировать, допустимо ли использование во *внешних ключах* пустых значений. В общем случае, если участие *дочерней таблицы* в связи является обязательным, то рекомендуется запрещать применение пустых значений в соответствующем *внешнем ключе*. В то же время, если имеет место частичное участие *дочерней таблицы* в связи, то помещение пустых значений в поле *внешнего ключа* должно быть разрешено. Например, если в операции фиксации сделок некоторой торговой фирмы необходимо указать покупателя, то поле `КодКлиента` должно иметь атрибут `NOT NULL` . Если допускается продажа или покупка товара без указания клиента, то для поля `КодКлиента` можно указать атрибут `NULL` .

Следующая проблема связана с организацией *поддержки ссылочной целостности* при выполнении операций модификации данных в базе. Здесь возможны следующие ситуации:

1. Вставка новой строки в *дочернюю таблицу*. Для обеспечения *ссылочной целостности* необходимо убедиться, что значение *внешнего ключа* новой строки *дочерней таблицы* равно пустому значению либо некоторому конкретному значению, присутствующему в поле *первичного ключа* одной из строк *родительской таблицы*.
2. Удаление строки из *дочерней таблицы*. Никаких нарушений *ссылочной целостности* не происходит.
3. Обновление *внешнего ключа* в строке *дочерней таблицы*. Этот случай подобен описанной выше первой ситуации. Для сохранения *ссылочной целостности* необходимо убедиться, что значение *внешнего ключа* в обновленной строке *дочерней таблицы* равно пустому значению либо некоторому конкретному значению, присутствующему в поле *первичного ключа* одной из строк *родительской таблицы*.
4. Вставка строки в *родительскую таблицу*. Такая вставка не может вызвать нарушения *ссылочной целостности*. Добавленная строка просто становится родительским объектом, не имеющим дочерних

объектов.

5. Удаление строки из *родительской таблицы*. *Ссылочная целостность* окажется нарушенной, если в *дочерней таблице* будут существовать строки, ссылающиеся на удаленную строку *родительской таблицы*. В этом случае может использоваться одна из следующих стратегий:

   `NO ACTION` . Удаление строки из *родительской таблицы* запрещается, если в *дочерней таблице* существует хотя бы одна ссылающаяся на нее строка.

   `CASCADE` . При *удалении* строки из *родительской таблицы* автоматически удаляются все ссылающиеся на нее строки *дочерней таблицы*. Если любая из удаляемых строк *дочерней таблицы* выступает в качестве родительской стороны в какой-либо другой связи, то операция *удаления* применяется ко всем строкам *дочерней таблицы* этой связи и т.д. Другими словами, *удаление* строки *родительской таблицы* автоматически распространяется на любые *дочерние таблицы*.

   `SET NULL` . При удалении строки из *родительской таблицы* во всех ссылающихся на нее строках дочернего отношения в поле *внешнего ключа*, соответствующего *первичному ключу* удаленной строки, записывается пустое значение. Следовательно, удаление строк из *родительской таблицы* вызовет занесение пустого значения в соответствующее поле *дочерней таблицы*. Эта стратегия может использоваться, только когда в поле *внешнего ключа* *дочерней таблицы* разрешается помещать пустые значения.

   `SET DEFAULT` . При удалении строки из *родительской таблицы* в поле *внешнего ключа* всех ссылающихся на нее строк *дочерней таблицы* автоматически помещается значение, указанное для этого поля как значение по умолчанию. Таким образом, удаление строки из *родительской таблицы* вызывает помещение принимаемого по умолчанию значения в поле *внешнего ключа* всех строк *дочерней таблицы*, ссылающихся на удаленную строку. Эта стратегия применима лишь в тех случаях, когда полю *внешнего ключа* *дочерней таблицы* назначено некоторое значение, принимаемое по умолчанию.

   `NO CHECK` . При удалении строки из *родительской таблицы* никаких действий по сохранению *ссылочной целостности* данных не предпринимается.

6. Обновление *первичного ключа* в строке *родительской таблицы*. Если значение *первичного ключа* некоторой строки *родительской таблицы* будет обновлено, нарушение *ссылочной целостности* случится при том условии, что в дочернем отношении существуют строки, ссылающиеся на исходное значение *первичного ключа*. Для сохранения *ссылочной целостности* может применяться любая из описанных выше стратегий. При использовании стратегии `CASCADE` *обновление* значения *первичного ключа* в строке *родительской таблицы* будет отображено в любой строке *дочерней таблицы*, ссылающейся на данную строку.

Существует и другой вид целостности – *смысловая (семантическая) целостность* базы данных. Требование ***смысловой целостности*** определяет, что данные в базе данных должны изменяться таким образом, чтобы не нарушалась сложившаяся между ними смысловая связь.

Уровень поддержания *целостности данных* в разных системах существенно варьируется.

Идеология архитектуры клиент-сервер требует переноса максимально возможного числа правил *целостности данных* на сервер. К преимуществам такого подхода относятся:

гарантия целостности базы данных, поскольку все правила сосредоточены в одном месте (в базе данных);

автоматическое применение определенных на сервере *ограничений целостности* для любых приложений;

отсутствие различных реализаций ограничений в разных клиентских приложениях, работающих с базой данных;

быстрое срабатывание ограничений, поскольку они реализованы на сервере и, следовательно, нет необходимости посылать данные клиенту, увеличивая при этом сетевой трафик;

доступность внесенных в ограничения на сервере изменений для всех клиентских приложений, работающих с базой данных, и отсутствие необходимости повторного распространения измененных приложений клиентов среди пользователей.

К недостаткам хранения *ограничений целостности* на сервере можно отнести:

отсутствие у клиентского приложения возможности реагировать на некоторые ошибочные ситуации, возникающие на сервере при реализации тех или иных правил (например, ошибок при выполнении хранимых процедур на сервере);

ограниченность возможностей языка SQL и языка хранимых процедур и триггеров для реализации всех возникающих потребностей определения *целостности данных*.

На практике в клиентских приложениях реализуют лишь такие правила, которые тяжело или невозможно реализовать с применением средств сервера. Все остальные *ограничения целостности* данных переносятся на сервер.