**Основные понятия реляционной модели данных**

Реляционная модель данных предполагает хранение данных в виде плоских таблиц, состоящих из строк и столбцов. Это интуитивно понятный способ, однако и здесь существует несколько условий, которым должны удовлетворять такие таблицы.

Например, таким ограничениям удовлетворяет следующая таблица:

|  |  |  |  |
| --- | --- | --- | --- |
| Таб\_Номер | ФИО\_Сотрудника | РазмерЗП | Отдел |
| 10065 | Иванов Иван Иванович | 30000 | Стратегического планирования |
| 12543 | Петров Петр Петрович | 50000 | Труда и заработной платы |
| 65423 | И т.д. |  |  |

Основными понятия в теории реляционного представления данных являются тип данных, домен, кортеж и отношение.

**Тип данных** – это способ описания физического хранения данных, т.е. тип данных определяет сколько места в памяти выделяется для хранения данных и как они трактуются. Это понятие ничем не отличается от понятия типа данных в программировании.

**Домен** характеризуется базовым типом данных и условием, которому должны удовлетворять данные, попадающие в этот домен. Условие обычно задает логическое описание тех данных, которые могут использоваться в базе. Например, поле «Возраст», естественно не может быть отрицательным и не может превышать определенной границы (110-120 лет). Таким образом, хранение возраста как целого числа недостаточно, так как этот тип данных может хранить и недопустимые значения по условию. Тогда определяются домены.

**Заголовок (схема, структура) отношения –** это множество пар <A,T>, где А – имя атрибута, Т – тип данных или домен этого атрибута. Т.е. схема отношения задает, какие поля используются для хранения информации о некотором объекте, для которого создается отношение.

**Кортеж** – это множество триплетов <A,T,V>, где каждому атрибуту в заголовке отношения А, имеющему тип Т, соответствует значение V. Таким образом, кортеж представляет собой описание одного объекта, соответствующего заголовку некоторого отношения. Кратко кортеж можно записать в виде <35267, Сидорова Мария Ивановна, 30000, Бухгалтерия>, предполагая очевидное соответствие атрибутов заголовка отношения указанным значениям.

**Тело отношения –** это совокупность (**множество**) кортежей в отношении. Кортеж может соответствовать схеме отношения, но не входить в его тело.

**Отношение** включает в себя схему отношения и тело отношения.

Часто в литературе кортеж называют записью, а отношение называют таблицей.

Таким образом, под отношением будет пониматься таблица, для которой справедливы условия:

1. отсутствуют кортежи-дубликаты
2. отсутствует упорядоченность кортежей
3. отсутствует упорядоченность атрибутов
4. атрибуты являются атомарными, т.е. неделимы (**первая нормальная форма**).

Прокомментируем немного эти свойства.

1. Отсутствие дубликатов отслеживается СУБД посредством введения понятия первичного ключа. Первичный ключ – это минимальный набор атрибутов заголовка отношения, составное значение которых уникально (однозначно) определяет кортеж отношения. В отношении не может быть двух записей с одинаковым значением первичного ключа. Так, в отношении из примера логично назначить первичным ключом табельный номер сотрудника. Может быть несколько ключей, обладающих этим свойством. Однако выбрать нужно только один из вариантов. Оставшиеся называются ключами-кандидатами.
2. Определение тела отношения определяет его как множество кортежей. Естественно, что для множества важно вхождение элементов в него, а не порядок их вхождения.
3. Аналогично в определении схемы отношения задействуется понятие множества. Так что логически от перестановки столбцов в таблице-отношении суть отношения не изменится.
4. Предполагается, что атрибуты задаются скалярными значениями (неделимыми, атомарными). Это облегчает представление и поиск. Это условие является очень важным и определяет первую нормальную форму – основу для применения теории нормализации отношений.

Таким образом, реляционная база данных состоит их набора таблиц- отношений. Помимо этого понятия для базы данных имеет значение понятие целостности. Выделяют два вида целостности – целостность сущности и целостность ссылок.

**Целостность сущности** фактически определяется первым требованием к отношению – наличию первичного ключа и уникальной идентификацией всего объекта (т.е. всех его атрибутов) через значения атрибутов первичного ключа. Не допускается также применение неопределенного значения (NULL) в качестве ключа кортежа.

**Целостность ссылок** важна в случаях связей между таблицами. Так возможна ситуация, когда ключ одного отношения является атрибутом другого отношения. Тогда для второго отношения он является **внешним ключом**. Так, в примере про сотрудников, очевидно, что будет много повторений названия отдела. Хранить информацию про отделы можно в отдельном отношении (номер отдела, название, начальник), а в отношении «Сотрудники» хранить только значение первичного ключа отдела, в котором работает конкретный сотрудник. Тогда важно, чтобы в это поле не попало значение, которого не существует в качестве ключа в отношении «Отделы».

Возможна проблема, связанная с ограничением ссылочной целостности. Допустим, информация про конкретный отдел должна быть удалена. Однако существуют кортежи о сотрудниках, которые работали в этом отделе. Что делается в этом случае? В этом случае должна поддерживаться политика поддержки ссылочной целостности. Среди них можно выделить:

* Restrict – политика запрета выполнения операции изменения или удаления кортежа, которая может привести к нарушению ссылочной целостности
* Cascade – политика каскадного удаления всех связанных кортежей.
* Ignore – игнорировать ограничение ссылочной целостности
* Set default – установить в качестве значения внешнего ключа в зависимой таблице значение по умолчанию
* Set NULL – установить в качестве значения внешнего ключа в зависимой таблице неопределенное значение NULL.

**Обзор возможностей языка SQL**

Язык SQL (Structured Query Language) был создан в 1986 году, а в 1987 году стал международным стандартом. Все современные СУБД поддерживают этот язык. Он ориентирован не только на работу с данными, но и на клиент-серверную архитектуру приложений работы с базами данных, что, в частности, и обусловило такую популярность.

Команды языка SQL можно разделить на группы:

* DDL (Data Definition Language) – язык определения метаданных. Команды позволяют создавать и изменять структуру базы данных. К этой группе относятся CREATE TABLE, DROP TABLE, ALTER TABLE и пр.
* DML (Data Manipulation Language) – язык манипулирования данными. В эту группу команд входят INSERT, DELETE, UPDATE.
* DQL (Data Query Language) – язык запросов. Основной представить этой группы – оператор SELECT.
* DCL (Data Control Language) – язык управления доступом
* Команды администрирования данных
* Команды управления транзакциями.

**Краткое описание стандарта записи наиболее важных SQL-команд.**

1. Создание базы данных

<определение\_базы\_данных> ::=

CREATE DATABASE имя\_базы\_данных

[ON [PRIMARY]

[ <определение\_файла> [,...n] ]

[,<определение\_группы> [,...n] ] ]

[ LOG ON {<определение\_файла>[,...n] } ]

[ FOR LOAD | FOR ATTACH ]

<определение\_файла>::=

([ NAME=логическое\_имя\_файла,]

FILENAME='физическое\_имя\_файла'

[,SIZE=размер\_файла ]

[,MAXSIZE={max\_размер\_файла |UNLIMITED } ]

[, FILEGROWTH=величина\_прироста ] )[,...n]

1. Изменение базы данных

<изменение\_базы\_данных> ::=

ALTER DATABASE имя\_базы\_данных

{ ADD FILE <определение\_файла>[,...n]

[TO FILEGROUP имя\_группы\_файлов ]

| ADD LOG FILE <определение\_файла>[,...n]

| REMOVE FILE логическое\_имя\_файла

| ADD FILEGROUP имя\_группы\_файлов

| REMOVE FILEGROUP имя\_группы\_файлов

| MODIFY FILE <определение\_файла>

| MODIFY FILEGROUP имя\_группы\_файлов

<свойства\_группы\_файлов>}

1. Удаление базы данных

DROP DATABASE имя\_базы\_данных [,...n]

1. Создание таблицы

<определение\_таблицы> ::=

CREATE TABLE имя\_таблицы

{(имя\_столбцатип\_данных[ NOT NULL ][ UNIQUE]

[DEFAULT <значение>]

[ CHECK (<условие\_выбора>)][,...n]}

[CONSTRAINT имя\_ограничения]

[PRIMARY KEY (имя\_столбца [,...n])

{[UNIQUE (имя\_столбца [,...n])}

[FOREIGN KEY (имя\_столбца\_внешнего\_ключа

[,...n])

REFERENCES имя\_род\_таблицы

[(имя\_столбца\_род\_таблицы [,...n])],

[MATCH {PARTIAL | FULL}]

[ON UPDATE {CASCADE| SET NULL |SET DEFAULT

|NO ACTION}]

[ON DELETE {CASCADE| SET NULL |SET DEFAULT

|NO ACTION}]

{[CHECK(<условие\_выбора>)][,...n]})

Отдельно следует отметить ограничение FOREIGN KEY, так как именно в нем задается ссылочная целостность данных таблицы. Это ограничение внешнего ключа, которое связывает поле (поля) текущей таблицы с полем (полями) другой таблицы, которая в данном случае будет родительской. Требуется, чтобы значение связанного поля присутствовало в связанном столбце родительской таблицы. Секции ON UPDATE и ON DELETE определяют политику поведения СУБД при нарушении ссылочной целостности.

1. Изменение таблицы

<изменение\_таблицы> ::=

ALTER TABLE имя\_таблицы

[ADD [COLUMN]имя\_столбцатип\_данных

[ NOT NULL ][UNIQUE]

[DEFAULT <значение>][ CHECK (<условие\_выбора>)]]

[DROP [COLUMN] имя\_столбца [RESTRICT | CASCADE ]]

[ADD [CONSTRAINT [имя\_ограничения]]

[{PRIMARY KEY (имя\_столбца [,...n])

|[UNIQUE (имя\_столбца [,...n])}

|[FOREIGN KEY (имя\_столбца\_внешнего\_ключа [,...n])

REFERENCES имя\_род\_таблицы

[(имя\_столбца\_род\_таблицы [,...n])],

[ MATCH {PARTIAL | FULL}

[ON UPDATE {CASCADE| SET NULL |

SET DEFAULT | NO ACTION}]

[ON DELETE {CASCADE| SET NULL |

SET DEFAULT | NO ACTION}]

|[CHECK(<условие\_выбора>)][,...n]}]

[DROP CONSTRAINT имя\_ограничения

[RESTRICT | CASCADE]]

[ALTER [COLUMN] SET DEFAULT <значение>]

[ALTER [COLUMN] DROP DEFAULT]

1. Удаление таблицы

DROP TABLE имя\_таблицы

1. Создание индекса

<создание\_индекса>::=

CREATE [ UNIQUE ]

[ CLUSTERED | NONCLUSTERED ]

INDEX имя\_индекса ON имя\_таблицы(имя\_столбца

[ASC|DESC][,...n])

[WITH [PAD\_INDEX]

[[,] FILLFACTOR=фактор\_заполнения]

[[,] IGNORE\_DUP\_KEY]

[[,] DROP\_EXISTING]

[[,] STATISTICS\_NORECOMPUTE] ]

[ON имя\_группы\_файлов ]

1. Запрос на добавление информации

<оператор\_вставки>::=INSERT INTO <имя\_таблицы>

[(имя\_столбца [,...n])]

{VALUES (значение[,...n])|

<SELECT\_оператор>}

1. Запрос на удаление информации

<оператор\_удаления> ::=DELETE

FROM <имя\_таблицы>[WHERE <условие\_отбора>]

1. Запрос на изменение данных

<оператор\_изменения> ::=

UPDATE имя\_таблицы SET имя\_столбца=

<выражение>[,...n]

[WHERE <условие\_отбора>]

При модификации данных возможно нарушение ссылочной целостности. Возможны следующие ситуации;

* 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.
* Удаление строки из дочерней таблицы. Никаких нарушений ссылочной целостности не происходит.
* 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.
* Вставка строки в родительскую таблицу. Такая вставка не может вызвать нарушения ссылочной целостности. Добавленная строка просто становится родительским объектом, не имеющим дочерних объектов.
* 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 таблицыавтоматически 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.
* 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.

1. Оператор выборки SELECT.

SELECT [ALL | DISTINCT ] {\*|[имя\_столбца

[AS новое\_имя]]} [,...n]

FROM имя\_таблицы [[AS] псевдоним] [,...n]

[WHERE <условие\_поиска>]

[GROUP BY имя\_столбца [,...n]]

[HAVING <критерии выбора групп>]

[ORDER BY имя\_столбца [,...n]]

FROM – определяются имена используемых таблиц;

WHERE – выполняется фильтрация строк объекта в соответствии с заданными условиями;

GROUP BY – образуются группы строк, имеющих одно и то же значение в указанном столбце;

HAVING – фильтруются группы строк объекта в соответствии с указанным условием;

SELECT – устанавливается, какие столбцы должны присутствовать в выходных данных;

ORDER BY – определяется упорядоченность результатов выполнения операторов.

Для сравнения строк актуально использование шаблонов – конструкция LIKE.

Символ % – вместо этого символа может быть подставлено любое количество произвольных символов.

Символ \_ заменяет один символ строки.

[] – вместо символа строки будет подставлен один из возможных символов, указанный в этих ограничителях.

[^] – вместо соответствующего символа строки будут подставлены все символы, кроме указанных в ограничителях.

Пример. Найти клиентов, у которых в номере телефона вторая цифра 2, 3 или 4.

SELECT Клиент.Фамилия, Клиент.Телефон

FROM Клиент

WHERE Клиент.Телефон LIKE '\_[2-4]%'

1. Использование подзапросов.

Подзапрос – это инструмент создания временной таблицы, содержимое которой извлекается и обрабатывается внешним оператором. Текст подзапроса должен быть заключен в скобки. К подзапросам применяются следующие правила и ограничения:

* фраза ORDER BY не используется, хотя и может присутствовать во внешнем подзапросе;
* список в предложении SELECT состоит из имен отдельных столбцов или составленных из них выражений – за исключением случая, когда в подзапросе присутствует ключевое слово EXISTS;
* по умолчанию имена столбцов в подзапросе относятся к таблице, имя которой указано в предложении FROM. Однако допускается ссылка и на столбцы таблицы, указанной во фразе FROM внешнего запроса, для чего применяются квалифицированные имена столбцов (т.е. с указанием таблицы);
* если подзапрос является одним из двух операндов, участвующих в операции сравнения, то запрос должен указываться в правой части этой операции.

Существует два типа подзапросов:

* Скалярный подзапрос возвращает единственное значение. В принципе, он может использоваться везде, где требуется указать единственное значение.
* Табличный подзапрос возвращает множество значений, т.е. значения одного или нескольких столбцов таблицы, размещенные в более чем одной строке. Он возможен везде, где допускается наличие таблицы.

Пример скалярного подзапроса:

SELECT Дата, Количество

FROM Сделка

WHERE Количество=(SELECT Max(Количество) FROM Сделка)

Пример табличного подзапроса:

SELECT Название

FROM Товар

WHERE КодТовара In

(SELECT КодТовара FROM Склад)

Ключевые слова ANY и ALL могут использоваться с подзапросами, возвращающими один столбец чисел. Если подзапросу будет предшествовать ключевое слово ALL, условие сравнения считается выполненным, только когда оно выполняется для всех значений в результирующем столбце подзапроса. Если записи подзапроса предшествует ключевое слово ANY, то условие сравнения считается выполненным, когда оно выполняется хотя бы для одного из значений в результирующем столбце подзапроса. Если в результате выполнения подзапроса получено пустое значение, то для ключевого слова ALL условие сравнения будет считаться выполненным, а для ключевого слова ANY – невыполненным. Ключевое слово SOME является синонимом слова ANY.

Пример

SELECT Клиент.Фамилия, Сделка.Количество

FROM Клиент INNER JOIN Сделка

ON Клиент.КодКлиента=Сделка.КодКлиента

WHERE Сделка.Количество>=ALL(SELECT Количество

FROM Сделка)

Ключевые слова EXISTSи NOT EXISTS предназначены для использования только совместно с подзапросами. Результат их обработки представляет собой логическое значение TRUE или FALSE. Для ключевого слова EXISTS результат равен TRUE в том и только в том случае, если в возвращаемой подзапросом результирующей таблице присутствует хотя бы одна строка. Если результирующая таблица подзапроса пуста, результатом обработки операции EXISTS будет значение FALSE. Для ключевого слова NOT EXISTS используются правила обработки, обратные по отношению к ключевому слову EXISTS. Поскольку по ключевым словам EXISTS и NOT EXISTS проверяется лишь наличие строк в результирующей таблице подзапроса, то эта таблица может содержать произвольное количество столбцов.

Пример

SELECT Название

FROM Товар

WHERE EXISTS (SELECT КодТовара

FROM Склад

WHERE Товар.КодТовара=Склад.КодТовара)

Вычислимые поля.

В общем случае для создания вычисляемого (производного) поля в списке SELECT следует указать некоторое выражение языка SQL. В этих выражениях применяются арифметические операции сложения, вычитания, умножения и деления, а также встроенные функции языка SQL. Можно указать имя любого столбца (поля) таблицы или запроса, но использовать имя столбца только той таблицы или запроса, которые указаны в списке предложения FROM соответствующей инструкции. При построении сложных выражений могут понадобиться скобки.

Стандарты SQL позволяют явным образом задавать имена столбцов результирующей таблицы, для чего применяется фраза AS.

Пользователю доступны следующие основные итоговые функции:

* Count (Выражение) - определяет количество записей в выходном наборе SQL-запроса;
* Min/Max (Выражение) - определяют наименьшее и наибольшее из множества значений в некотором поле запроса;
* Avg(Выражение) - эта функция позволяет рассчитать среднее значение множества значений, хранящихся в определенном поле отобранных запросом записей. Оно является арифметическим средним значением, т.е. суммой значений, деленной на их количество.
* Sum (Выражение) - вычисляет сумму множества значений, содержащихся в определенном поле отобранных запросом записей.

Предложение GROUP BY

Часто в запросах требуется формировать промежуточные итоги. Для этой цели в операторе SELECT используется предложение GROUP BY. Запрос, в котором присутствует GROUP BY, называется группирующим запросом, поскольку в нем группируются данные, полученные в результате выполнения операции SELECT, после чего для каждой отдельной группы создается единственная суммарная строка. При наличии в операторе SELECT фразы GROUP BY каждый элемент списка в предложении SELECT должен иметь единственное значение для всей группы. Более того, предложение SELECT может включать только следующие типы элементов: имена полей, итоговые функции, константы и выражения, включающие комбинации перечисленных выше элементов.

Все имена полей, приведенные в списке предложения SELECT, должны присутствовать и во фразе GROUP BY - за исключением случаев, когда имя столбца используется в итоговой функции. Обратное правило не является справедливым - во фразе GROUP BY могут быть имена столбцов, отсутствующие в списке предложения SELECT.

Если совместно с GROUP BY используется предложение WHERE, то оно обрабатывается первым, а группированию подвергаются только те строки, которые удовлетворяют условию поиска.

При помощи HAVING отражаются все предварительно сгруппированные посредством GROUP BY блоки данных, удовлетворяющие заданным в HAVING условиям. Это дополнительная возможность "профильтровать" выходной набор.

Условия в HAVING отличаются от условий в WHERE:

* HAVING исключает из результирующего набора данных группы с результатами агрегированных значений;
* WHERE исключает из расчета агрегатных значений по группировке записи, не удовлетворяющие условию;
* в условии поиска WHERE нельзя задавать агрегатные функции.

### **Представления**

Представления представляют собой временные таблицы и являются объектами базы данных, информация в которых не хранится постоянно, как в базовых таблицах, а формируется динамически при обращении к ним. Представление - это фактически тот же запрос, который выполняется всякий раз при участии в какой-либо команде. У пользователя создается впечатление, что он работает с настоящей, реально существующей таблицей.

Согласно стандарту SQL Команда создания представления имеет вид:

<определение\_представления> ::=

{ CREATE| ALTER} VIEW имя\_представления

[(имя\_столбца [,...n])]

[WITH ENCRYPTION]

AS SELECT\_оператор

[WITH CHECK OPTION]

Обращение к представлению осуществляется с помощью оператора SELECT как к обычной таблице.

### Представление можно использовать в команде так же, как и любую другую таблицу. К представлению можно строить запрос, модифицировать его (если оно отвечает определенным требованиям), соединять с другими таблицами. Содержание представления не фиксировано и обновляется каждый раз, когда на него ссылаются в команде. Представления значительно расширяют возможности управления данными.

Не все представления в SQL могут быть модифицированы. Модифицируемое представление определяется следующими критериями:

* основывается только на одной базовой таблице;
* содержит первичный ключ этой таблицы;
* не содержит DISTINCT в своем определении;
* не использует GROUP BY или HAVING в своем определении;
* по возможности не применяет в своем определении подзапросы;
* не использует константы или выражения значений среди выбранных полей вывода;
* в просмотр должен быть включен каждый столбец таблицы, имеющий атрибут NOT NULL;
* оператор SELECT просмотра не использует агрегирующие (итоговые) функции, соединения таблиц, хранимые процедуры и функции, определенные пользователем;
* основывается на одиночном запросе, поэтому объединение UNION не разрешено.

**Хранимые процедуры**

База данных может хранить не только данные, но и определенный программный код, который реализует бизнес-логику приложения. Эти возможности актуальны для серверных СУБД, так как хранимые процедуры и функции позволяют выполнять сложные алгоритмы модификации и поиска данных без необходимости выполнять обмен данными с клиентскими приложениями.

В современных серверных СУБД имеется возможность создавать и хранимые процедуры, и хранимые функции. Общий вид команды SQL для создания таков:

<определение\_функции>::=

{CREATE | ALTER } FUNCTION [владелец.]

имя\_функции

( [ { @имя\_параметра скаляр\_тип\_данных

[=default]}[,...n]])

RETURNS тип\_данных

[WITH {ENCRYPTION | SCHEMABINDING}

[,...n] ]

[AS]

BEGIN

<тело\_функции>

RETURN скаляр\_выражение

END

<определение\_процедуры>::=

{CREATE | ALTER } PROCEDURE [владелец.]

имя\_процедуры

( [ { @имя\_параметра скаляр\_тип\_данных

[=default]}[,...n]])

[WITH {ENCRYPTION | SCHEMABINDING}

[,...n] ]

[AS]

BEGIN

<тело\_процедуры>

END

В качестве команд можно использовать команды SQL и команды управления потоком выполнения (условия, циклы и пр). Параметры функции могут быть входными (IN), выходными (OUT) и INOUT (т.е. и входными, и выходными). Соответствующее ключевое слово предваряет описание параметра функции или процедуры.

Разберем несколько примеров. Пусть нам требуется написать хранимую функцию или процедуры вычисления оценки по баллам. Возможны следующие варианты.

Использовался синтаксис MySQL 5.0.

Вариант 1. Напишем хранимую процедуру, которая получает в качестве входного параметра количество баллов и сравнивает их со шкалой оценок. Результат возвращается через выходной параметр.

CREATE PROCEDURE `GetMark\_v1`(in ball INT, out mark INT)

BEGIN

IF ball BETWEEN 55 AND 70 THEN

SET mark=3;

ELSEIF ball BETWEEN 71 AND 85 THEN

SET mark=4;

ELSEIF ball BETWEEN 86 AND 100 THEN

SET mark=5;

ELSE SET mark=2;

END IF;

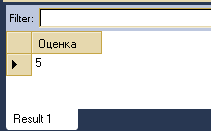
END

Вызов процедуры осуществляется следующим образом:

call GetMark\_v1(89,@m);

select ""+@m as "Оценка";

Результат выполнения:



Вариант 2. Чтобы при смене правил вычисления оценок не нужно было бы менять процедуру, создадим справочную таблицу для хранения всех оценок и их диапазонов.

create table marks

( Mark int,

Low int,

High int)

Если в данной таблице хранятся данные только о положительных оценках и их нижней и верхней границе, мы можем искать в ней нужную запись по параметру-баллам и возвращать найденную оценку. Оформим этот вариант в виде функции с одним параметром, хранящим набранные баллы, и возвращающую найденную оценку или 2 в случае, когда набранным баллам ничего в таблице не соответствует. В данной функции демонстрируется использование переменных и использование условного оператора.

CREATE FUNCTION Get\_Mark2 (ball INT) returns INT

BEGIN

DECLARE k,m INT;

SELECT COUNT(\*) INTO k FROM Marks where ball between Low and High;

IF k>0 THEN

SELECT Mark INTO m FROM Marks WHERE ball between Low and High;

RETURN m;

END IF;

RETURN 2;

END

Вызов функции оформляется следующим образом:

set @m=Get\_Mark\_v2(79);

select ""+@m AS "Оценка";

В данной функции для определения наличия в таблице оценки с соответствующими баллами используется запрос на поиск количества записей, удовлетворяющих условию. Покажем еще один вариант, в котором для этой цели используется функция EXIST, которая часто используется в подзапросах. Результатом функции является логическое выражение, которое определяет, является ли выборка пустой.

CREATE FUNCTION Get\_Mark\_v3 (ball INT) returns INT

BEGIN

DECLARE m INT;

IF EXISTS (SELECT \* FROM Marks WHERE ball between Low and High) THEN

SELECT Mark INTO m FROM Marks WHERE ball between Low and High;

RETURN m;

END IF;

RETURN 2;

END;

**Триггеры**

Триггеры являются одной из специальных разновидностей хранимых процедур. Их исполнение происходит при выполнении для таблицы какого-либо оператора языка манипулирования данными (DML). Триггеры используются для проверки целостности данных, для заполнения значений по умолчанию, а также для отката транзакций в случае нарушения каких-либо правил целостности.

Триггер запускается сервером автоматически при попытке изменения данных в таблицах, с которыми триггеры связаны. Каждый триггер привязывается к конкретной таблице. Все производимые им модификации данных рассматриваются как одна транзакция. В случае обнаружения ошибки или нарушения целостности данных происходит откат этой транзакции. Тем самым внесение изменений запрещается. Отменяются также все изменения, уже сделанные триггером.

Создает триггер только владелец базы данных. Это ограничение позволяет избежать случайного изменения структуры таблиц, способов связи с ними других объектов и т.п.

Согласно стандарту команда создания триггера выглядит следующим образом:

<Определение\_триггера>::=

CREATE TRIGGER имя\_триггера

BEFORE | AFTER <триггерное\_событие>

ON <имя\_таблицы>

[REFERENCING

<список\_старых\_или\_новых\_псевдонимов>]

[FOR EACH { ROW | STATEMENT}]

[WHEN(условие\_триггера)]

<тело\_триггера>

триггерные события состоят из вставки, удаления и обновления строк в таблице. В последнем случае для триггерного события можно указать конкретные имена столбцов таблицы. Время запуска триггера определяется с помощью ключевых слов BEFORE (триггер запускается до выполнения связанных с ним событий) или AFTER (после их выполнения) (для MS SQL Server имеется тип INSTEAD OF).

Выполняемые триггером действия задаются для каждой строки ( FOR EACH ROW ), охваченной данным событием, или только один раз для каждого события ( FOR EACH STATEMENT ).

Существует два параметра, определяющих поведение триггеров:

* AFTER. Триггер выполняется после успешного выполнения вызвавших его команд. Если же команды по какой-либо причине не могут быть успешно завершены, триггер не выполняется. Можно определить несколько AFTER -триггеров для каждой операции ( INSERT, UPDATE, DELETE ).
* INSTEAD OF. Триггер вызывается вместо выполнения команд. В отличие от AFTER -триггера INSTEAD OF -триггер может быть определен как для таблицы, так и для представления. Для каждой операции INSERT, UPDATE, DELETE можно определить только один INSTEAD OF -триггер.

Триггеры различают по типу команд, на которые они реагируют:

* INSERT TRIGGER – запускаются при попытке вставки данных с помощью команды INSERT.
* UPDATE TRIGGER – запускаются при попытке изменения данных с помощью команды UPDATE.
* DELETE TRIGGER – запускаются при попытке удаления данных с помощью команды DELETE.

Отметим, что внутри триггера не допускается выполнение ряда операций, таких, например, как:

* создание, изменение и удаление базы данных;
* восстановление резервной копии базы данных или журнала транзакций.

При выполнении команд добавления, изменения и удаления записей SQL Server создает две специальные таблицы: inserted и deleted. Например, в MySQL это будут таблицы old и new. В них содержатся списки строк, которые будут вставлены или удалены по завершении транзакции. Структура таблиц inserted и deleted идентична структуре таблиц, для которой определяется триггер. Для каждого триггера создается свой комплект таблиц inserted и deleted, поэтому никакой другой триггер не сможет получить к ним доступ. В зависимости от типа операции, вызвавшей выполнение триггера, содержимое таблиц inserted и deleted может быть разным:

* команда INSERT работает только с таблицей inserted (new).–
* команда DELETE – только с таблицей deleted(old)
* команда UPDATE – с обеими таблицами.

В MS SQL Server триггер выполняется как неявно определенная транзакция, поэтому внутри триггера допускается применение команд управления транзакциями. В частности, при обнаружении нарушения ограничений целостности для прерывания выполнения триггера и отмены всех изменений, которые пытался выполнить пользователь, необходимо использовать команду ROLLBACK TRANSACTION. Для MySQL такой механизм еще не доступен, поэтому для отката транзакции применяют прием выполнения некорректного оператора, например, вставки.

Для удаления триггера используется команда

DROP TRIGGER {имя\_триггера} [,...n]

Приведем примеры.

Пример 1. Создадим триггер для вставки в таблицу результатов сессии, в котором проверяются ограничения целостности (студент с заданном колом существует, предмет с заданным кодом существует, дисциплину нужно сдавать именно в этом семестре). Если произойдет нарушение этих ограничений, то требуется откатить транзакцию, т.е. не выполнять вставку записи. Если же все данные будут корректными, проведем заполнение значений полей даты сдачи зачета/экзамена как текущей и вычислим оценку по указанным баллам.

Для проверки корректности данных для вставки создадим хранимую функцию, чтобы код триггера был не очень сложным. (Для некоторых версий MySQL требуется, чтобы в триггере было упоминание только текущей записи, обращение к другим таблицам и записям недоступно). Тип INT(11) в MySQL является логическим типом.

create function IsCorrect (idStud INT, idSub INT, Sem INT, idTeach INT) RETURNS INT(11)

BEGIN

RETURN EXISTS (SELECT \* from students INNER JOIN sessions ON students.NumGroup=sessions.NumGroup INNER JOIN subjects on sessions.idSubject=subjects.idSubject INNER JOIN Teachers on sessions.idTeacher=Teachers.idTeacher where students.idStudent=idStud and subjects.idSubject=idSub AND teachers.idTeacher=idTeach and NumSemestr=Sem);

END

Триггер будет вызывать функцию проверки корректности, передавая в функцию поля из новой записи таблицы Results (запись называется New). В случае необходимости отката действия вставки осуществляется вставка записи в таблицу кафедр с уже существующим первичным ключом (для SQL Server здесь можно вызвать команду ROLLBACK TRAN).

create trigger tr1\_results before insert on results for each row

begin

if IsCorrect(New.idStudent, New.idSubject, New.NumSemestr, New.idTeacher) then

set New.Balls=Get\_Mark\_v3(New.Balls);

set New.DateExam=Now();

else

insert into departments values (1,"","");

end if;

end

Приведем еще один пример триггера на вставку новой записи в таблицу результатов. Этот триггер должен срабатывать после вставки и быть связан с подсчетом рейтинга студентов. Для этого введем в базу новую таблицу:

create table Reyting

( idStudent INT PRIMARY KEY,

summ\_balls INT)

При вставке нового результата рейтинг студента должен меняться. Таким образом, нужно проанализировать, есть ли запись о студенте – в случае положительного ответа произвести суммирование баллов, иначе добавить новую запись в таблицу рейтинга.

create trigger tr3\_results after insert on results for each row

begin

if exists(select \* from reyting where idStudent=new.idStudent) then

update reyting set summ\_balls=summ\_balls+new.Balls

where idStudent=new.idStudent;

else

insert into reyting values (New.idStudent, New.balls);

end if;

end