**Саратовский Государственный университет им. Н.Г. Чернышевского**

Курсовая работа

на тему: «Особенности разработки триггеров и хранимых процедур в СУБД»

(на примере базы данных отдела кадров)

студентки I V курса заочного отделения КН и ИТ

(прикладная математика и информатика)

ФРОЛОВОЙ Марии Александровны

Содержание

1. Ведение
2. Реляционная база данных
3. Хранимые процедуры
4. Триггеры

Заключение

# Список использованной литературы

**1. ВВЕДЕНИЕ**

* + **Базы данных**

С появлением магнитных дисков началась история систем управления данными во внешней памяти. До этого каждая прикладная программа, которой требовалось хранить данные во внешней памяти, сама определяла расположение каждой порции данных на магнитной ленте или барабане и выполняла обмены между оперативной и внешней памятью с помощью программно-аппаратных средств низкого уровня (машинных команд или вызовов соответствующих программ операционной системы). Такой режим работы не позволяет или очень затрудняет поддержание на одном внешнем носителе нескольких архивов долговременно хранимой информации. Кроме того, каждой прикладной программе приходилось решать проблемы именования частей данных и структуризации данных во внешней памяти.

База данных (БД) – это структурированный набор постоянно хранимых данных. Постоянность означает, что данные не уничтожаются по завершении программы или пользовательского сеанса, в котором они были созданы.

База данных- это набор, совокупность файлов, в которых находится информация. Программная система (приложение), обеспечивающая работу с базой данных (файлами данных) называется системой управления базой данных (СУБД).

В зависимости от расположения программы, которая использует данные, и самих данных, а также от способа разделения данных между несколькими пользователями различают локальные и удаленные базы данных.

По структуре организации данных базы делятся на реляционные и нереляционные.

##### Поддержка языков БД

Для работы с базами данных используются специальные языки, в целом называемые языками баз данных. В ранних СУБД поддерживалось несколько специализированных по своим функциям языков. Чаще всего выделялись два языка - язык определения схемы БД (SDL - Schema Definition Language) и язык манипулирования данными (DML - Data Manipulation Language). SDL служил главным образом для определения логической структуры БД, т.е. той структуры БД, какой она представляется пользователям. DML содержал набор операторов манипулирования данными, т.е. операторов, позволяющих заносить данные в БД, удалять, модифицировать или выбирать существующие данные.

В современных СУБД обычно поддерживается единый интегрированный язык, содержащий все необходимые средства для работы с БД, начиная от ее создания, и обеспечивающий базовый пользовательский интерфейс с базами данных. Стандартным языком наиболее распространенных в настоящее время реляционных СУБД является язык SQL (Structured Query Language).

Прежде всего, язык SQL сочетает средства SDL и DML, т.е. позволяет определять схему реляционной БД и манипулировать данными. При этом именование объектов БД (для реляционной БД - именование таблиц и их столбцов) поддерживается на языковом уровне в том смысле, что компилятор языка SQL производит преобразование имен объектов в их внутренние идентификаторы на основании специально поддерживаемых служебных таблиц-каталогов. Внутренняя часть СУБД (ядро) вообще не работает с именами таблиц и их столбцов.

1. **РЕЛЯЦИОННЫЕ БАЗА ДАННЫХ (РБД)**

Реляционная база данных - это тело связанной информации, сохраняемой в двумерных таблицах. Напоминает адресную или телефонную книгу.

Были созданы таблицы DAN (Interbase) и данные (MS Access) со столбцами:

|  |  |
| --- | --- |
| Interbase | MS Access |
| TAB\_NO – табельный номер  FAM – фамилия работника  IMIA – имя работника  OTSH – отчество работника  G\_R - год рождения работника  ADRES - адрес работника  TEL - телефон  DATA\_P – дата приема  DATA\_U - дата увольнения  OBRAZ - образование  SEM\_POLOJ – семейное положение  S\_N\_PASPORT – паспортные данные  K\_V - код выслуги  K\_O - код оклада  K\_P - код профессии | табельный номер  фамилия работника  имя работника  отчество работника  год рождения работника  адрес работника  телефон  дата приема  дата увольнения  образование  семейное положение  паспортные данные  код выслуги  код оклада  код профессии |

и заполненными строками. Каждая строка (называемая также записью - основной элемент БД) будет соответствовать определенной особенности; каждый столбец будет содержать значение для каждого типа данных - имени, телефонного номера, и адреса представляемого в каждой строке.

То что мы получили - является основой реляционной базы данных как и было определенно, а именно, двумерной (строка и столбец) таблицей информации. Однако реляционные базы данных редко состоят из одной таблицы. Такая таблица меньше чем файловая система. Создав несколько таблиц взаимосвязанной информации, мы сможем выполнить более сложные и мощные операции с данными. Мощность базы данных зависит от связи, созданной между фрагментами информации, а не от самого фрагмента информации.

## Связывание одной таблицы с другой

Но на примере наших таблиц можно показать реальное использование в деловой ситуации. Предположим, что персонажи в наших первых таблицах - это работники МЧС России. В другой таблице, мы могли бы запомнить дополнительную информацию них. Столбцы второй таблицы PROF (**Interbase)** и профессия (**MS Access)** профессия выглядят так:

|  |  |
| --- | --- |
| SELECT \*  FROM PROF; | SELECT профессия.[код профессии], профессия.[название профессии]  FROM профессия; |
| Interbase | MS Access |
| K\_P NAZ  ==== ========  1 пожарный  2 водитель  3 бухгалтер  4 инспектор  5 начальник караула  6 диспетчер  7 секретарь  8 водометчик  9 командир отделения  10 начальник части  11 заместитель по тылу  12 наводчик | код название  профессии профессии  = = = = = = = = = = = == = = = = = = = ==  1 пожарный  2 водитель  3 бухгалтер  4 инспектор  5 начальник караула  6 диспетчер  7 секретарь  8 водометчик  9 командир отделения  10 начальник части  11 заместитель по тылу  12 наводчик |

Вся таблица:

Много мощных функций можно выполнить, извлекая информацию из этих таблиц согласно указанным параметрам, особенно когда эти параметры включают в себя фрагменты информации связанные в различных таблицах друг с другом. Например, возьмем запрос: Вывод сотрудников с соответствующими им профессиями:

|  |  |
| --- | --- |
| Interbase | SELECT TAB\_NO,FAM,IMIA,OTSH,G\_R,PROF.NAZ  FROM DAN,PROF  WHERE DAN.K\_P=PROF.K\_P; |
| MS Access | SELECT данные.[табельный номер], данные.фамилия, данные.имя, данные.отчество,  данные.[год рождения], профессия.[название профессии]  FROM профессия INNER JOIN данные ON профессия.[код профессии] = данные.[код профессии]; |
| Interbase | TAB\_NO FAM IMIA OTSH G\_R NAZ  ====== ========= ========== =============== ==== ================  1001 Петров Петр Петрович 01.12.1971 инспектор  1002 Сидоров Павел Сергеевич 10.03.1975 начальник караула  1003 Кортунов Сергей Владимирович 17.07.1963 начальник части  1004 Семенова Пелагея Никитична 01.06.1985 диспетчер  1005 Романова Евлампия Апполинарьевна 04.12.1982 диспетчер  1006 Несмелая Агрофена Агриповна 04.11.1976 секретарь  1007 Сердюков Игорь Игнатьевич 27.05.1978 пожарный  1008 Любимов Петр Николаевич 17.02.1967 водометчик  1009 Шарипов Руслан Имранович 14.08.1960 пожарный  1010 Ухабистова Авдотья Владиленовна 07.10.1956 бухгалтер  1011 Игумнов Андрей Дмитриевич 25.03.1962 заместитель по тылу  1012 Ильюшин Дмитрий Александрович 19.11.1958 наводчик  1013 Туполев Эдуард Валентинович 06.01.1969 пожарный  1014 Рабинович Ицхак Абрамович 23.09.1984 водитель  1000 Иванов Иван Иванович 12.04.1968 водитель |
| MS Access | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | Таб № | фамилия | имя | отчество | год рождения | название профессии | | | 1 | ИВАНОВ | Иван | Иванович | 1988 | пожарный |  | | 2 | ПЕТРОВ | Петр | Петрович | 1988 | пожарный |  | | 3 | СИДОРОВ | Сидор | Сидорович | 1977 | пожарный |  | | 4 | ФРОЛОВА | Мрия | Александровна | 1970 | секретарь |  | | 5 | ФРОЛОВА | Анастасия | Дмитриевна | 1990 | инспектор |  | | 5 | КУКУШКИН | Антон | Викторович | 1955 | заместитель по тылу | | |  |  |  | |

Схемы баз данных (Database Diagrams) — это тип объектов, который присутствует только в проектах Access. Они являются аналогом схемы данных в базах данных Access, однако в проектах Access это понятие существенно расширено.

Во-первых, таких схем в одном проекте может быть несколько. Это значит, что всю совокупность таблиц можно разделить на логические области и построить для каждой из них свою схему данных. Это очень важно, т. к. количество таблиц в базе данных на сервере может быть очень велико, и, если разместить их все на одной схеме, она может быть просто необозрима. Кроме того, могут быть созданы схемы данных для определенных целей, когда нужно выделить и выразить специфический взгляд на подмножество таблиц базы.

Во-вторых, сам инструментарий для создания схем баз данных более богат: он позволяет не только определять связи между существующими таблицами, но и изменять структуру и свойства таблиц, создавать новые таблицы и т. д. В этом смысле он является альтернативой обычному способу описания таблиц — с помощью режима Конструктора.

Вы можете выполнять операции как с отдельным объектом этой структуры, так и с группой объектов. Причем все изменения, которые вносятся в структуру базы данных, не сохраняются не сервере до тех пор, пока вы не сохраните измененную схему. Это удобно, когда нужно внести много изменений. Тогда сохранить можно только конечный результат, когда вы убедились, что все изменения согласованы.

Программы реляционной базы данных разрабатывались для того чтобы обрабатывать большие и сложные совокупности данных такого типа, что очевидно является более универсальным методом в деловом мире. Даже если бы база данных отдела кадров содержала сотни или тысячи имен - как это вероятно и бывает на практике - одна команда SQL может выдать мне информацию в которой он нуждался почти немедленно.

В отличии от строк, столбцы таблицы упорядочиваются и нумеруются, так в таблице № 3 OCLAD первый столбец слева K\_O содержит код оклада. Во избежание неоднозначности все столбцы таблицы должны иметь разные имена. Кроме того, он является первичным ключом - важным элементом в структуре базы данных. Каждый столбец имеет определенный тип данных. Все данные конкретного столбца относятся к одному типу: текст, число, дата и т.д., так как содержит однотипную информацию.

//Вывод общее количество кодов оклада // **Interbase**

**SELECT COUNT(\*)**

**FROM OCLAD**;

COUNT

=====

9

// Вывод максимального оклада **//MS Access**

**ELECT DISTINCTROW Max([оклады].[оклад]) AS [Max - оклад]**

**FROM оклады;**

|  |
| --- |
| оклады max |
| Max - оклад |
| 5 999,55р. |

Таблицы DAN, PROF, OCLAD образуют простую реляционную БД, она мала, с ней легко работать, но содержит все необходимые компоненты для иллюстрации основных принципов и приемов применения SQL. Первый столбец каждой таблицы содержит №№, разные для каждой строки (первичные ключи). Некоторые из номеров встречаются в других столбцах таблиц. Это внешние ключи, которые ссылаются на первичные ключи (для удобства имеют одинаковые имена). Выполнение условия «прикрепления» таблиц, говорит, что система находится в состоянии «ссылочной целостности»

**3. Хранимые процедуры**

база файл триггер информация

Это тоже элемент базы данных, наряду с таблицами. Это специальная программа, написанная на так называемом языке процедур и триггеров и хранимая в качестве части базы данных. Поскольку мы создали хранимую процедуру, мы можем напрямую вызвать на выполнение из своего приложения. Хранимая процедура может получать входные параметры и возвращать определенные значения приложению. Язык процедур и триггеров включает в себя операторы SQL и некоторые расширения, такие как IF THEN ELSE, MHILE DO, FOR SELECT DO, исключения и обработку ошибок.

Преимущества использования хранимых процедур:

* Модульное оформление;
* Приложения, обращающиеся к одной и той же базе данных, могут делит между собой хранимые процедуры (использовать одну и ту же хранимую процедуру), избегая дублирования кода (т.е. нет необходимости в повторном коде хранимой процедуры) и снижая таким образом размер самих приложений;
* Рациональная поддержка. Когда процедура обновляется, ее изменения автоматически отражаются во всех приложениях, которые ее используют, и новая компиляция этих приложений не требуется. Приложения только однажды компилируются для каждого клиента.
* Улучшение работы. И это – основное достоинство этого инструмента. Хранимые процедуры выполняются сервером, а не клиентом, что снижает загрузку сети и улучшает её работу, особенно в условиях клиентского доступа. Например, в системе управления кадрами, когда общая база данных хранится на сервере и работники отдела кадров и руководство предприятия пользуются ее услугами, у некоторого клиента возникает потребность получить аналитические данные в некотором разрезе. При наличии в БД хранимой процедуры, получающей требуемую аналитику, нет нужды клиенту вытаскивать к себе все необходимые данные для расчета, а достаточно вызвать на сервере соответствующую хранимую процедуру, передав ей необходимые параметры, и эта процедура, на месте исполнив всю работу и не перегоняя большое количества данных по сети клиенту, выдаст ему только результат в виде небольшой таблицы. Этой же процедурой может воспользоваться и другой клиент и получить для себя такую же аналитику.
* **Работа с хранимыми процедурами**

С помощью интерактивного SQL можно создавать, модифицировать и удалять процедуры и исключения. Существуют два способа создавать, видоизменять и удалять хранимые процедуры с помощью интерактивного SQL:

* Интерактивно;
* С помощью входного файла, содержащего операторы определения данных.

Обычно предпочтение отдают файлам, им легче модифицировать и обеспечивать для них соответствующую документацию. Для простых модификаций существующих процедур более подходит интерактивный режим. Пользователь, который создает процедуру, является собственником и может передавать привилегии исполнения процедуры другим пользователем, триггерам и хранимым процедурам.

Хранимые процедуры в базах данных, преимущества использования, CREATE PROC, мастер Create Stored Procedure Wizard, макросы и модули VBA в Microsoft Access

Хранимые процедуры - это сохраненные как объекты баз данных блоки кода на языке SQL. В важных базах хранимые процедуры используются очень активно. Часто на них реализуют всю бизнес-логику приложений и практически любое внесение изменений в базы данных должно производиться только при помощи хранимых процедур. Что дают нам хранимые процедуры:

модульность разработки приложений;

так же, как и представления, позволяют защитить клиентские приложения от изменений в структуре таблиц в базах данных;

позволяют реализовывать дополнительную логику проверок, протоколирования действий пользователей и т.п.;

позволяют производить каскадные обновления;

в отличие от представлений, хранимые процедуры могут принимать и возвращать параметры;

обеспечивают очень мощные механизмы обеспечения безопасности;

позволяют повысить производительность - за счет использования откомпилированных планов хранимых процедур из кэша и за счет сокращения сетевого трафика при отправке больших наборов команд;

при помощи хранимых процедур можно реализовывать изощренные схемы обработки ошибок на сервере.

Создание хранимых процедур производится на SQL Server командой CREATE PROC или - на графическом интерфейсе - в Enterprise Manager из контейнера Stored Procedures или при помощи мастера Create Stored Procedure Wizard.

В MS Access как таковых хранимых процедур нет, но их роль могут выполнять:

макросы (типа "Запуск запроса SQL")

модули VBA.

* **Использование файла определения данных**

Для создания и изменения процедуры с использованием файла определения данных используют текстовой редактор, что бы записать такой файл, затем сохраняют файл и исполняют с помощью интерактивного SQL утилиты IBConsole, в которой вставляют данные из файла, либо используют команду isgl в командной строке (режим DOS), задавая команду в виде:

C:\ISGL-INPUT FILENAME DATABASE\_NAME.

Здесь FILENAME – это имя файла определения данных, а DATABASE\_NAME – имя используемой БД. Файл определения данных должен включать:

* Операторы создания, модификации и удаления процедур и исключений;
* Любые другие ISGL – операторы. Исключения должны созданы до ссылки на них в процедурах.

Хранимая процедура меняет код профессии в таблице DAN соответствующий переменной MEN на код соответствующий переменной NOW.

**SET TERM ^;**

**CREATE PROCEDURE IZM\_PROF(MEN INTEGER,NOW INTEGER)**

**RETURNS(TAB\_NO CHAR(5),FAM VARCHAR(30),K\_P INTEGER,NOW\_P INTEGER)AS**

**BEGIN**

**FOR**

**SELECT TAB\_NO,FAM,K\_P**

**FROM DAN**

**INTO:TAB\_NO,FAM,K\_P**

**DO**

**BEGIN**

**IF(K\_P=MEN)THEN**

**BEGIN**

**NOW\_P=NOW;**

**UPDATE DAN**

**SET K\_P=:NOW\_P**

**WHERE TAB\_NO=:TAB\_NO;**

**SUSPEND;**

**END**

**END**

**END^**

**SET TERM;^**

Вызов процедуры

**SELECT\***

**FROM IZM\_PROF(12,5);**

Удаление процедуры:

**DROP PROCEDURE <имя процедуры>;**

* **Вызов хранимых процедур.**

Приложения могут вызывать хранимые процедуры из SGL, DSGL, ISGL.

Существуют два типа хранимых процедур: процедуры-выборки (selectable procedures) и исполняемые процедуры (executable procedures). Отличие в работе этих двух видов ХП заключается в том, что процедуры-выборки обычно возвращают множество наборов выходных параметров, сгруппированных построчно, которые имеют вид набора данных, а исполняемые процедуры могут либо вообще не возвращать параметры, либо возвращать только один набор выходных параметров, перечисленных в Returns. Процедуры-выборки вызываются в запросах SELECT, а исполняемые процедуры - с помощью команды EXECUTE PROCEDURE.

* SELECT - процедуры, которые приложения может использовать вместо таблицы или вьюера в операторе SELECT; результатом исполнения оператора SELECT над такой процедурой будет обычная таблица данных, какая получается, например, когда в SELECT стоит символ\*. Например, если имеется SELECT - процедура с именем SP, то ее вызов на исполнение выглядеть так:

**SELECT\***

**FROM IZM\_PROF(12,5);**

Если воспользоваться запросом

**EXECUTE PROCEDURE** UV\_D (1.1);

то произойдет изменение и вывод только первой строки:

KOD\_D SUMA\_D N\_SUMA\_D

======== ======== ========

300 2500 2750

Вызов процедуры UV\_MZ\_D

**SELECT**\*

**FROM** UV\_MZ\_D (2600);

произведет изменения только двух строк таблицы DOLJN, в которых поле SUMA\_D меньше 2600, и выведет таблицу из этих строк.

KOD\_D SUMA\_D N\_SUMA\_D

======== ======== ========

300 2500 2600

307 2000 2600

//Поиск повторений //MS Access

**SELECT данные.фамилия, данные.имя, данные.отчество**

**FROM данные**

**WHERE (((данные.фамилия) In (SELECT [фамилия] FROM [данные] As Tmp GROUP BY [фамилия] HAVING Count(\*)>1 )))**

**ORDER BY данные.фамилия;**

|  |  |  |
| --- | --- | --- |
| Поиск повторений для данные | | |
| фамилия | имя | отчество |
| ИВАНОВ | Иван | Иванович |
| ИВАНОВ | Иван | Иванович |
| ФРОЛОВА | Анастасия | Дмитриевна |
| ФРОЛОВА | Мария | Александровна |

* **Удаление хранимой процедуры.**

Удаление хранимой процедуры выполняется с помощью оператора **DROP PROCEDURE**. Синтаксис оператора удаления таков:

**DROP PROCEDURE** name;

Здесь name – имя хранимой процедуры.

Хранимые процедуры — основное средство программирования серверной логики. Они представляют собой откомпилированный модуль, написанный на языке Transact-SQL. В коде хранимой процедуры можно использовать не только операции выборки и модификации данных, но и логику ветвления, переменные, вызовы других процедур и некоторые другие средства, характерные для языков программирования высокого уровня. Код процедуры синтаксически анализируется при компиляции, а оптимизированный план выполнения создается при первом вызове процедуры. В отличие от запросов, хранимые процедуры имеют возможность возвращать несколько наборов записей, а также значения.

Для вызова процедуры клиентская программа или другая процедура должны указать имя выполняемой процедуры и передать ей набор входных параметров.

Хранимые процедуры используются в проектах Access везде, где могут использоваться предложения SQL. Они могут служить: в качестве источников записей в формах, отчетах, активных страницах; в качестве источников строк для полей со списком.

Однако нужно помнить, что хранимые процедуры возвращают необновляемый набор записей. Хранимая процедура может содержать одну инструкцию SQL, и в этом случае для ее создания можно использовать графический режим Конструктора запросов. Если же она состоит из нескольких инструкций SQL, тогда она создается и изменяется в редакторе SQL.

В первом случае чтобы создать процедуру проще всего щелкнуть по ярлыку Создание сохраненной процедуры в режиме конструктора (Create stored procedure in designer). При этом открывается стандартное окно Конструктора запросов, в котором можно создать инструкцию SQL, составляющие тело процедуры. Так же как и представление, хранимая процедура в данном случае может иметь ряд дополнительных свойств. Чтобы отобразить или изменить эти свойства, необходимо открыть окно свойств. На вкладке Сохраненная процедура (Stored Procedure) содержится перечень свойств, которые аналогичны свойствам представления. А на второй вкладке можно посмотреть и изменить параметры процедуры.

Откройте процедуру в режиме Конструктора. Нажмите кнопку Свойства (Properties) на панели инструментов, чтобы отобразить диалоговое окно Свойства (Properties). Раскройте вкладку Параметры сохраненной процедуры.

На вкладке отображается таблица со списком параметров процедуры. Каждый параметр обязательно должен иметь имя и тип данных. Кроме того, для него может быть указано значение по умолчанию. Обратим внимание, что имя каждого параметра начинается со значка @. Но имя параметра не может быть изменено на этой вкладке, так же как нельзя добавить новый параметр. Это делается только в бланке запроса в окне Конструктора или прямо в инструкции SQL.

Для того чтобы создать хранимую процедуру, состоящую из нескольких инструкций SQL, нужно открыть список запросов в окне проекта и нажать кнопку Создать (New). В диалоговом окне Новый запрос (New Query) следует выбрать элемент Ввод сохраненной процедуры (Create Text Stored Procedure) .

Нужно изменить имя процедуры в первой строке шаблона и ввести текст процедуры в соответствии с правилами языка Transact-SQL.

Если в теле процедуры нужно ввести инструкцию SQL SELECT, то можно воспользоваться для этого Конструктором запросов. Установите курсор на строке редактора, в которой должна быть введена инструкция SELECT и нажмите кнопку Вставить SQL (Insert SQL) на панели инструментов . Появится окно Конструктора запросов. После того как запрос будет создан, нужно закрыть это окно, подтвердив сохранение изменений инструкции SQL.

Пример использования хранимой процедуры в качестве источника записей вы увидите, если откроете отчет "2". В качестве источника записей в этом отчете используется одноименная хранимая процедура. В свойстве Входные параметры (Input Parameters) этого отчета описаны два параметра: @Beginning\_date И @Ending\_date.

При запуске отчета эти параметры будут запрашиваться так же, как и при использовании запроса с параметрами в базе данных Access.

SELECT данные.фамилия, данные.имя, данные.отчество, данные.образование

FROM данные LEFT JOIN оклады ON данные.[код оклада] = оклады.[код оклада]

WHERE (((оклады.[код оклада]) Is Null));

|  |  |  |  |
| --- | --- | --- | --- |
| 'данные' без подчиненных в 'оклады' | | | |
| фамилия | имя | отчество | образование |
| СИДОРЧУК | Валентина | Викторовна |  |

**4. Триггеры**

Триггер - это особый вид хранимой процедуры, которая выполняется автоматически при вставке, удалении или модификации записи таблицы или вьюера. Триггеры никогда не вызываются напрямую, а только при модификации таблиц или вьюеров и могут "срабатывать" непосредственно до или сразу же после указанного события.

Существует три основных SQL-операции, применимые к данным, - INSERT/DELETE/UPDATE. А поскольку срабатывание триггера может происходить "до" (**BEFORE**)и "после"(**AFTER**) операции, то получается шесть возможных видов триггеров на таблицу - до и после каждой из трех возможных SQL-операций.

Триггер может быть активный (**ACTIVE**) и неактивный (**INACTIVE**).

* **Польза от применения триггеров.**

- Принудительная обработка ограничений, что позволяет пользователю быть уверенным, что он вводит в строки только достоверные данные.

- Изменения в триггере автоматически отображаются у всех приложений, связанных с таблицей, что не требует их перекомпиляции, так как все происходит в базе данных, а не у каждого приложения в отдельности.

- Автоматическая регистрация изменений в таблицах. Приложение может регистрировать с помощью триггера изменения в таблице.

- Автоматическая регистрация изменений в базе данных с помощью обработчиков событий в триггерах.

* **Примеры триггеров.**

Триггер (англ. trigger «собачка, защёлка, спусковой крючок»):

в русском языке первоначально — термин из области радиосхем, позже электронной техники: пусковая схема, схема с несколькими устойчивыми состояниями, см. триггер (электроника); в последнее время заимствовались и другие значения английского оригинала:

спусковой крючок пистолета, арбалета и т. п., кнопка на джойстике, находящаяся под указательным пальцем; в более широком смысле: некая причина возникновения события вообще условия, при наступлении которых должно происходить предписанное действие Триггеры являются особой разновидностью хранимых процедур, которые активизируются не по прямому вызову пользователя, а при наступлении определенного события в базе данных, а именно — вставки, удаления или модификации записи в таблице. Триггеры привязаны к конкретной таблице. Каждая таблица может иметь до трех триггеров — по одному на каждый вид операции с данными (INSERT, UPDATE, DELETE). Но один триггер может также обрабатывать и два и все три события.

Триггер может быть использован для соблюдения определенных правил модификации данных. При нарушении этих правил он может откатить все произведенные изменения и вернуть данные в исходное состояние. Кроме того, триггер может быть использован для автоматического осуществления каких-либо сопутствующих операций, например ведения журнала изменений данных или автоматического обновления некоторых итоговых значений (например, значения поля, в которое вносится количество сотрудников в фирме).

Триггер срабатывает после удаления строки из таблицы DAN и записывает удаленную строку в таблицу UD\_DAN.

**SET TERM^;**

**CREATE TRIGGER CHRAN\_UD\_DAN FOR DAN**

**ACTIVE AFTER DELETE POSITION 0**

**AS**

**BEGIN**

**INSERT INTO UD\_DAN**

**VALUES(OLD.TAB\_NO,OLD.FAM,OLD.IMIA,OLD.OTSH,OLD.G\_R,OLD.ADRES,OLD.TEL,OLD.DATA\_P,OLD.DATA\_U, OLD.OBRAZ,OLD.SEM\_POLOJ, OLD.S\_N\_PASPORT, OLD.K\_V, OLD.K\_O, OLD.K\_P);**

**END^**

**SET TERM;^**

Теперь если выполнить команду

**DELETE FROM DAN**

**WHERE TAB\_NO=1014; //** (и 1013)

произойдёт конечно удаление строки из таблицы DAN, сработает триггер и удаленная строка запишется в таблицу UD\_DAN, которая собственно и предназначена для хранения удаленных данных.

Что бы отключить триггер следует выполнить команду

**ALTER TRIGGER <имя триггера> INACTIVE;**

Что бы включить триггер следует выполнить команду

**ALTER TRIGGER <имя триггера> ACTIVE**;

Что бы удалить триггер команда

**DROP TRIGGER <имя триггера>;**

Следующий триггер, срабатывает ДО ВСТАВКИ (**BEFORE INSERT**) в таблицу DAN и заполняет поле первичного ключа (TAB\_NO).

**SET** TERM ^;

**CREATE TRIGGER** VS\_T **FOR** DAN

**ACTIVE BEFORE INSERT POSITION** 0

**AS**

**BEGIN**

**IF** (NEW.TAB\_NO **IS NULL**) **THEN**

NEW.TAB\_NO=**GEN\_ID** (TAB, 1);

**END** ^

**SET** TERM; ^

Для этого триггера потребуется генератор с начальным значением 1012, так как это максимальный табельный номер уже заполненной таблицы DAN и, новым сотрудникам, вносимым в таблицу, будут присвоены табельные номера, следующие за 1012.

**CREATE GENERATOR** TAB**;**

**SET GENERATOR** TAB **TO** 1012**;**

При вставки новой строки в таблицу DAN, даже не указав значение поля TAB\_NO,

**INSERT INTO** DAN (FAM, IMIA, OTSH, G\_R, ADRES, TEL, DATA\_P, DATA\_U, OBRAZ,

SEM\_POLOJ, S\_N\_PASPORT, K\_V, K\_O, K\_P)

**VALUES** ('Кротов', 'Андрей', 'Федорович', '12.08.1961', 'Кирова 52-124', '281344', '21.02.2008', '28.03.2008','среднее''холост', '9999 22', '900' , '50' , '1');

То есть триггер перед вставкой произвел проверку, указал ли пользователь значение поля TAB\_NO, и не обнаружив такового, воспользовался генератором, увеличив его значение на 1, занес полученное в поле TAB\_NO нового сотрудника.

Генератор, в свою очередь, теперь хранит значение 1014.

Кроме контекстной переменной NEW, существует переменная OLD. В отличие от NEW, OLD содержит старые значения записи, которые удаляются или изменяются. Например, можно использовать переменную OLD для получения значений записей, которые удаляются из таблицы.

Следующий триггер срабатывает ПОСЛЕ УДАЛЕНИЯ (**AFTER DELETE)** и записывает строки, удаленные из таблицы в отдельную таблицу.

Сначала необходимо создать таблицу для хранения удаленных данных.

**CREATE TABLE** DAN\_D

(TAB\_NO\_D **Float**,

FAM\_D **Varchar** (20),

IMIA \_D **Varchar** (20),

OTSH\_D **Varchar** (20),

G\_R\_D **Date**,

ADRES\_D **Varchar** (20),

TEL\_D **Char** (6) );

**SET** TERM ^;

**CREATE TRIGGER** DEL\_DAN **FOR** DAN

**ACTIVE AFTER DELETE POSITION** 0

**AS**

**BEGIN**

**INSERT INTO** DAN\_D

**VALUES** (OLD.TAB\_NO, OLD.FAMILIA, OLD. IMIA E, OLD.OTSHO, OLD.GOD\_R,

OLD.ADRES, OLD.TEL);

**END** ^

**SET** TERM; ^

Как только происходит удаление строки из таблицы DANNIE,

**DELETE FROM** DANNIE

**WHERE** TAB\_NO=1013;

Так как, в данной БД при удалении строки из таблицы DAN удаляется и строка из таблицы NASH (начисления) с идентичным табельным номером (TAB\_NO), уместно было бы создать триггер, который сохранял бы и данные из таблицы NASH. Чтобы не создавать лишние таблицы для хранения удаленных данных можно расширить уже созданную таблицу DAN\_D.

**ALTER TABLE** DAN\_D **ALL** KOD\_Z\_D **FLOAT;**

**ALTER TABLE** DAN\_D **ALL** KOD\_D\_D **FLOAT;**

**ALTER TABLE** DAN\_D **ALL** KOD\_V\_D **FLOAT;**

**ALTER TABLE** DAN\_D **ALL** SLOJ\_D **FLOAT;**

**ALTER TABLE** DAN\_D **ALL** PAIK\_D **FLOAT**;

Следующий триггер срабатывает после удаления строки из таблицы NASH.

**SET** TERM ^;

**CREATE TRIGGER** DEL\_NASH **FOR** NASH

**ACTIVE AFTER DELETE POSITION** 0

**AS**

**BEGIN**

**UPDATE** DAN\_D

**SET** KOD\_Z\_D=OLD.KOD\_Z, KOD\_D\_D=OLD.KOD\_D,

KOD\_V\_D=OLD.KOD\_V, SLOJ\_D=OLD.SLOJ,

PAIK\_D=OLD.PAIK

**WHERE** TAB\_NO\_D=OLD.TAB\_NO;

**END** ^

**SET** TERM; ^

Теперь после команды

**DELETE FROM** DAN

**WHERE** TAB\_NO=1012;

произойдет следующее:

1. Произойдет удаление строки с TAB\_NO=1014 из таблицы DAN.
2. Сработает триггер DEL\_DAN, который запишет удаленную строку в таблицу DAN\_D.
3. Произойдет удаление строки с TAB\_NO=1014 из таблицы NASH.
4. Сработает триггер DEL\_NASH и допишет в, уже наполовину заполненную строку таблицы DAN\_D, данные из таблицы NASH, соответствующие этому сотруднику (**WHERE** TAB\_NO\_D=OLD.TAB\_NO).

Следующий триггер срабатывает после добавления строки в таблицу DAN и запускает хранимую процедуру, которая, в свою очередь, заполняет строку с таким же табельным номером (TAB\_NO) в таблице начислений (NASH). Поскольку нельзя предугадать, какими будут начисления того или иного работника, по умолчанию поля таблицы NASH будут заполняться значениями соответствующими минимальным начислениям предприятия, которые потом, безусловно, следует изменить. Хранимая процедура, вызванная триггером, лишь предварительно заполняет таблицу NASH, как бы занимая место для нового сотрудника, уже занесенного в таблицу DAN.

**SET** TERM ^;

**CREATE TRIGGER** Z\_NASH **FOR** DANNIE

**ACTIVE AFTER INSERT POSITION** 0

**AS**

**BEGIN**

**EXECUTE PROCEDURE** VSTAV\_NASH (NEW.TAB\_NO);

**END**^

**SET** TERM;^

Хранимая процедура, вызываемая триггером.

**SET** TERM ^;

**CREATE PROCEDURE** VSTAV\_NASH (T **FLOAT**)

**AS**

**BEGIN**

**INSERT INTO** NASH

**VALUES** (: T, 200, 307, 100, 0.5, 20);

**END**^

**SET** TERM; ^

При вставке строки

**INSERT INTO** DAN (FAM, IMIA,OTSH,G\_R,ADRES,TEL)

**VALUES** ('Кротов', 'Андрей', 'Федорович', '12.08.1961', 'Кирова 52-124', '281344');

произойдет следующее:

1. Сработает триггер VS\_T, который присвоит новому сотруднику,

следующий по порядку, табельный номер.

1. Заполнятся строка в таблице DANNIE соответствующими данными о новом сотруднике.

3. Срабатывает триггер Z\_NASH, который вызывает ХП.

4. Хранимая процедура VSTAV\_NASH вставляет в таблицу NASH данные (принятые по умолчанию) для нового сотрудника.

* **Управление состоянием триггера.**

По умолчанию триггер создается активным, т. е. он будет срабатывать при осуществлении соответствующей операции. Состоянием триггера управляет ключевое слово ACTIVE в заголовке. Если же триггер сделать неактивным, то он не будет исполняться при возникновении операции. Это бывает полезным при осуществлении каких-либо внеплановых операций над данными, например массовой заливке данных или ручном исправлении данных. Чтобы отключить триггер, необходимо выполнить команду DDL:

**ALTER TRIGGER** <trigger name> **INACTIVE;**

* **Удаление триггеров.**

Удаление триггера выполняется с помощью оператора **DROP TRIGGER**. Синтаксис оператора удаления таков:

**DROP TRIGGER** name;

Здесь name – имя триггера.

Это отдельная подпрограмма, связанная с таблицей или вьюером , которая автоматически выполняет действие, когда модифицируется таблица или вьюер на уровне строки. Они не вызываются напрямую, только при модификации таблиц или вьюеров.

//Триггер срабатывает после удаления строки из таблицы DAN и записывает удаленную строку в таблицу UD\_DAN.

**SET TERM^;**

**CREATE TRIGGER CHRAN\_UD\_DAN FOR DAN**

**ACTIVE AFTER DELETE POSITION 0**

**AS**

**BEGIN**

**INSERT INTO UD\_DAN**

**VALUES(OLD.TAB\_NO,OLD.FAM,OLD.IMIA,OLD.OTSH,OLD.G\_R,OLD.ADRES,OLD.TEL,OLD. DATA\_P, OLD.DATA\_U,OLD.OBRAZ,OLD.SEM\_POLOJ,OLD.S\_N\_PASPORT,OLD.K\_V,OLD.K\_O,OLD.K\_P);**

**END^**

**SET TERM;^**

Выполняя команду

**DELETE FROM DAN**

**WHERE TAB\_NO=1000;**

произойдет, конечно, удаление строки из таблицы DAN, сработает триггер, и удаленная строка запишется в таблицу UD\_DAN, которая собственно и предназначена для хранения удаленных данных.

Что бы отключить триггер следует выполнить команду

**ALTER TRIGGER <имя триггера> INACTIVE;**

Включить триггер следует выполнить команду

ALTER TRIGGER <имя триггера> ACTIVE;

Для удаления триггера команда

DROP TRIGGER <имя триггера>;

* **Триггеры в MS Access**

БД всегда содержит какую-нибудь особенность, которая не вписывается в схему возможных ограничений Access на значение конкретного поля или совокупности полей в записи таблицы. Кое-какие ограничения, конечно, задать можно, но их сложность ограничивается сложностью выражения, которое Access способен переварить в качестве такого ограничения. Никакой сложной обработки, вроде обращения к другим таблицам, там нет.

Пожелание разработчика в целом можно сформулировать так: "хочу иметь пользовательскую функцию, которая срабатывает всякий раз, когда данные в таблице меняются (точнее – хотят измениться!), такую, которая может либо запретить изменение, либо что-то в нем скорректировать, и такую, которую никак нельзя обойти". В современных СУБД такая функция называется триггером (trigger). Триггеры есть, в SQL Server.

В Access предлагалось достаточно примитивное решение проблемы: для заполнения любой таблицы обычно существует форма, у формы есть события, повесив на которые обработчики VBA можно достичь необходимого результата. Все это, в общем, было бы не плохо, если бы можно закрыть для пользователя доступ к редактированию данных непосредственно в таблицах. Но вот этого-то Access сделать до конца корректно не позволяет.

Часто лучше потерять все данные сразу, чем иметь ошибочные данные и думать, что они правильные. Поэтому проблема контроля ввода данных в таблицы является весьма и весьма насущной.

В основе Access лежит ядро баз данных Microsoft Jet. Объекты Access, доступные из VBA (в Access 2000 – из VB), являются на самом деле объектами ядра Jet. Те же обработчики событий в формах, например, мы имеем возможность писать благодаря тому, что объект ядра Jet Form поддерживает эти события. Ядро Jet имеет невысокие по сравнению с большими СУБД требования к ресурсам системы, обладая при этом достаточно большим потенциалом. Именно поэтому Access выглядит таким мобильным и годится для решения многих практических задач на достаточно современном уровне с минимальным количеством накладных расходов.

На уровне ядра Jet в Microsoft Access 2000 триггеров нет! Это продиктовано чисто коммерческими соображениями и связано с желанием Microsoft стремительно продвигать на рынок технологии, использующие SQL Server 7.0. В документации по Access триггером иногда называется группа кнопок, выполняющих функции переключателей – так это не те триггеры.

И в то же время, возможность использовать триггеры в Access 2000 есть. Microsoft теперь считает, что время Jet прошло и настала очередь больших корпоративных СУБД. Таких как SQL Server 7.0. Предусмотрено постепенное вытеснение Jet технологиями клиент-сервер. Появилось новое ядро баз данных MSDE (Microsoft Database Engine), очень тесно совместимое с SQL Server 7.0, практически это сильно усеченный SQL Server. Работая через это ядро можно хранить данные в формате SQL Server 7.0. Именно это рекомендуется делать тем, кто хочет иметь триггеры в Access 2000.

Загрузить в Access 2000 БД, которую мы хотим конвертировать в формат SQL Server 7.0, и запустить Upsizing Wizard . Wizard сделает все, что нужно, БД будет храниться далее в формате MS SQL Server 7.0, а не в mdb-файле. В Access вместо mdb мы будем загружать файл проекта adp.

Мы получаем возможность использовать виды (views), триггеры (triggers) и хранимые процедуры (stored procedures) SQL Server, редактируя их непосредственно из Access. Повышается уровень безопасности транзакций, появляются еще кое-какие плюсы, которые разработчика под Access обычно мало волнуют. Поскольку БД становится полностью совместима с SQL Server, можем пользоваться всем. При этом у нас остается весь инструментарий Access по разработке пользовательского интерфейса. Мы можем использовать также защиту данных на уровне SQL Server. Таблица Access в режиме дизайна начинает выглядеть так, как в Enterprise Manager SQL Server 7.0. Исчезает также вкладка Queries, вместо нее появляются вкладки Views и Stored Procedures.

Какие минусы?

Система становится заметно более тормозной. Хорошо для корпоративных решений, но совсем плохо для настольных баз данных. Чтобы писать триггеры, придется освоить язык, на котором они пишутся – Transact-SQL. Чтобы полноценно использовать предоставленные Вам возможности, нужно знать SQL Server 7.0

// **MS Access**

**Отчет 6**

**Фамилия название профессии**

ИВАНОВ начальник караула

пожарный

КУКУШКИНзаместитель по тылу

ПЕТРОВпожарный

СИДОРОВпожарный

СИДОРЧУКначальник части

ФРОЛОВАинспектор

секретарь

Option Compare Database

Sub Form\_Current()

On Error GoTo Form\_Current\_Err

If ChildFormIsOpen() Then FilterChildForm

Form\_Current\_Exit:

Exit Sub

Form\_Current\_Err:

MsgBox Error$

Resume Form\_Current\_Exit

End Sub

Sub ToggleLink\_Click()

On Error GoTo ToggleLink\_Click\_Err

If ChildFormIsOpen() Then

CloseChildForm

Else

OpenChildForm

FilterChildForm

End If

ToggleLink\_Click\_Exit:

Exit Sub

ToggleLink\_Click\_Err:

MsgBox Error$

Resume ToggleLink\_Click\_Exit

End Sub

Private Sub FilterChildForm()

If Me.NewRecord Then

Forms![данные].DataEntry = True

Else

Forms![данные].Filter = "[код профессии] = " & Me.[код профессии]

Forms![данные].FilterOn = True

End If

End Sub

Private Sub OpenChildForm()

DoCmd.OpenForm "данные"

If Not Me.[ToggleLink] Then Me![ToggleLink] = True

End Sub

Private Sub CloseChildForm()

DoCmd.Close acForm, "данные"

If Me![ToggleLink] Then Me![ToggleLink] = False

End Sub

Private Function ChildFormIsOpen()

ChildFormIsOpen = (SysCmd(acSysCmdGetObjectState, acForm, "данные") And acObjStateOpen) <> False

End Function

**ЗАКЛЮЧЕНИЕ**

Мы рассмотрели несколько типов объектов баз данных — хранимые процедуры, триггеры. Мы узнали следующее:

Хранимая процедура — это скомпилированный набор SQL-предложений, сохраненный в базе данных как именованный объект и выполняющийся как единый фрагмент кода. Для создания хранимой процедуры применяется предложение CREATE PROCEDURE, для изменения — ALTER PROCEDURE, а для удаления — DROP PROCEDURE.

Tриггер — это специальный тип хранимой процедуры, которая автоматически вызывается, когда данные в определенной таблице добавляются, удаляются или изменяются с помощью SQL-предложений INSERT, DELETE или UPDATE. Триггеры создаются с помощью предложения CREATE TRIGGER. Для изменения триггера используется предложение ALTER TRIGGER, а для удаления — предложение DROP TRIGGER.

**Список использованной литературы**

1. С.Д. Кузнецов, информационно-аналитические материалы Центра Информационных Технологий пользованием интерфейса ядра СУБД.
2. Грубер М. Понимание SQL- **МОСКВА 1993 (**Под редакцией Булычева В.Н.Перевод Лебедева В.Н.)
3. Грубер М. SQL. - М.: Лори, 2003.
4. К.Дейт. Введение в системы баз данных. 6-е издание. «Диалектика» Киев-Москва, 1998.
5. Т.Каннолли, К.Бегг, А. Страчан -Базы данных: проектирование, реализация и сопровождение.Теория и практика
6. Пахомов Б.И. Interbase b C++Builder на примерах./Б.И.Пахомов. – СПб.; БХВ- Петербург, 2006
7. КомпьютерПресс 12'2000. Copyright © КомпьютерПресс 2008

webmaster@compress.ru