Создание таблиц с помощью SQL-запросов

Содержание

Обзор

Создание таблиц с помощью SQL

Заключение

Обзор

На данном уроке мы познакомимся еще с одной возможностью создания таблиц - через посылку SQL-запросов. Как Вы, наверное, могли заметить на предыдущем уроке, Database Desktop не обладает всеми возможностями по управлению SQL-серверными базами данных. Поэтому с помощью Database Desktop удобно создавать или локальные базы данных или только простейшие SQL-серверные базы данных, состоящие из небольшого числа таблиц, не очень сильно связанных друг с другом. Если же Вам необходимо создать базу данных, состоящую из большого числа таблиц, имеющих сложные взаимосвязи, можно воспользоваться языком SQL (вообще говоря, для этих целей лучше всего использовать специализированные CASE-средства, которые позволяют в интерактивном режиме сгенерировать всю структуру базы данных и сформировать все связи; описание двух наиболее удачных CASE-средств - System Architect и S-Designor - дано в дополнительных уроках). При этом можно воспользоваться компонентом Query в Delphi, каждый раз посылая по одному SQL-запросу, а можно записать всю последовательность SQL-предложений в один так называемый скрипт и послать его на выполнение, используя, например, Windows Interactive SQL (WISQL.EXE) - интерактивное средство посылки SQL-запросов к InterBase (в том числе и локальному InterBase), входящее в поставку Delphi. Конечно, для этого нужно хорошо знать язык SQL, но, уверяю Вас, сложного в этом ничего нет! Конкретные реализации языка SQL незначительно отличаются в различных SQL-серверах, однако базовые предложения остаются одинаковыми для всех реализаций. Практика показывает, что если нет необходимости создавать таблицы во время выполнения программы, то лучше воспользоваться WISQL.

Создание таблиц с помощью SQL

Если Вы хотите воспользоваться компонентом TQuery, сначала поместите его на форму. После этого настройте свойство DatabaseName на нужный Вам алиас (если базы данных еще не существует, удобней создать ее в WISQL командой File|Create Database..., а затем уже настроить на нее новый алиас). После этого можно ввести SQL-предложение в свойство SQL. Для выполнения запроса, изменяющего структуру, вставляющего или обновляющего данные на сервере, нужно вызвать метод ExecSQL компонента TQuery. Для выполнения запроса, получающего данные с сервера (т.е. запроса, в котором основным является оператор SELECT), нужно вызвать метод Open компонента TQuery. Это связано с тем, что BDE при посылке запроса типа SELECT открывает так называемый курсор, с помощью которого осуществляется навигация по выборке данных (подробней об этом см. в уроке, посвященном TQuery).

Как показывает опыт, проще воспользоваться утилитой WISQL. Для этого в WISQL выберите команду File|Run an ISQL Script... и выберите файл, в котором записан ваш скрипт, создающий базу данных. После нажатия кнопки "OK" ваш скрипт будет выполнен, и в нижнее окно будет выведен протокол его работы.

Приведем упрощенный синтаксис SQL-предложения для создания таблицы на SQL-сервере InterBase (более полный синтаксис можно посмотреть в online-справочнике по SQL, поставляемом с локальным InterBase):

CREATE TABLE table

(<col\_def> [, <col\_def> | <tconstraint> ...]);

где

table - имя создаваемой таблицы,

<col\_def> - описание поля,

<tconstraint> - описание ограничений и/или ключей (квадратные скобки [] означают необязательность, вертикальная черта | означает "или").

Описание поля состоит из наименования поля и типа поля (или домена - см. урок 9), а также дополнительных ограничений, накладываемых на поле:

<col\_def> = col {datatype | COMPUTED BY (<expr>) | domain}

[DEFAULT {literal | NULL | USER}]

[NOT NULL] [<col\_constraint>]

[COLLATE collation]

Здесь

col - имя поля;

datatype - любой правильный тип SQL-сервера (для InterBase такими типами являются - см. урок 11 - SMALLINT, INTEGER, FLOAT, DOUBLE PRECISION, DECIMAL, NUMERIC, DATE, CHAR, VARCHAR, NCHAR, BLOB), символьные типы могут иметь CHARACTER SET - набор символов, определяющий язык страны. Для русского языка следует задать набор символов WIN1251;

COMPUTED BY (<expr>) - определение вычисляемого на уровне сервера поля, где <expr> - правильное SQL-выражение, возвращающее единственное значение;

domain - имя домена (обобщенного типа), определенного в базе данных;

DEFAULT - конструкция, определяющая значение поля по умолчанию;

NOT NULL - конструкция, указывающая на то, что поле не может быть пустым;

COLLATE - предложение, определяющее порядок сортировки для выбранного набора символов (для поля типа BLOB не применяется). Русский набор символов WIN1251 имеет 2 порядка сортировки - WIN1251 и PXW\_CYRL. Для правильной сортировки, включающей большие буквы, следует выбрать порядок PXW\_CYRL.

Описание ограничений и/или ключей включает в себя предложения CONSTRAINT или предложения, описывающие уникальные поля, первичные, внешние ключи, а также ограничения CHECK (такие конструкции могут определяться как на уровне поля, так и на уровне таблицы в целом, если они затрагивают несколько полей):

<tconstraint> = [CONSTRAINT constraint

<tconstraint\_def>]

<tconstraint>

Здесь

<tconstraint\_def> = {{PRIMARY KEY | UNIQUE} (col[,col...])

| FOREIGN KEY (col [, col ...]) REFERENCES other\_table

| CHECK (<search\_condition>)}

search\_condition =

{<val> operator {<val> | (<select\_one>)}

| <val> [NOT] BETWEEN <val> AND <val>

| <val> [NOT] LIKE <val> [ESCAPE <val>]

| <val> [NOT] IN (<val> [, <val> ...] |

<val> = {

col [array\_dim] | <constant> | <expr> | <functiont>

| NULL | USER | RDB$DB\_KEY } [COLLATE collation]

<constant> = num | "string" | charsetname "string"

<functiont> = {

COUNT (\* | [ALL] <val> | DISTINCT <val>)

| SUM ([ALL] <val> | DISTINCT <val>)

| AVG ([ALL] <val> | DISTINCT <val>)

| MAX ([ALL] <val> | DISTINCT <val>)

| MIN ([ALL] <val> | DISTINCT <val>)

| CAST (<val> AS <datatype>)

| UPPER (<val>)

| GEN\_ID (generator, <val>)

}

<operator> = {= | < | > |<= |>= | !< | !> | <> | !=}

<select\_one> = выражение SELECT по одному полю, которое возвращает в точности одно значение.

Приведенного неполного синтаксиса достаточно для большинства задач, решаемых в различных предметных областях. Проще всего синтаксис SQL можно понять из примеров. Поэтому мы приведем несколько примеров создания таблиц с помощью SQL.

Пример A: Простая таблица с конструкцией PRIMARY KEY на уровне поля

CREATE TABLE REGION (

REGION REGION\_NAME NOT NULL PRIMARY KEY,

POPULATION INTEGER NOT NULL);

Предполагается, что в базе данных определен домен REGION\_NAME, например, следующим образом:

CREATE DOMAIN REGION\_NAME

AS VARCHAR(40) CHARACTER SET WIN1251 COLLATE

PXW\_CYRL;

Пример B: Таблица с предложением UNIQUE как на уровне поля, так и на уровне таблицы

CREATE TABLE GOODS (

MODEL SMALLINT NOT NULL UNIQUE,

NAME CHAR(10) NOT NULL,

ITEMID INTEGER NOT NULL, CONSTRAINT MOD\_UNIQUE

UNIQUE (NAME, ITEMID));

Пример C: Таблица с определением первичного ключа, внешнего ключа и конструкции CHECK, а также символьных массивов

CREATE TABLE JOB (

JOB\_CODE JOBCODE NOT NULL,

JOB\_GRADE JOBGRADE NOT NULL,

JOB\_REGION REGION\_NAME NOT NULL,

JOB\_TITLE VARCHAR(25) CHARACTER SET WIN1251

COLLATE PXW\_CYRL NOT NULL,

MIN\_SALARY SALARY NOT NULL,

MAX\_SALARY SALARY NOT NULL,

JOB\_REQ BLOB(400,1) CHARACTER SET WIN1251,

LANGUAGE\_REQ VARCHAR(15) [5],

PRIMARY KEY (JOB\_CODE, JOB\_GRADE, JOB\_REGION),

FOREIGN KEY (JOB\_REGION) REFERENCES REGION (REGION),

CHECK (MIN\_SALARY < MAX\_SALARY));

Данный пример создает таблицу, содержащую информацию о работах (профессиях). Типы полей основаны на доменах JOBCODE, JOBGRADE, REGION\_NAME и SALARY. Определен массив LANGUAGE\_REQ, состоящий из 5 элементов типа VARCHAR(15). Кроме того, введено поле JOB\_REQ, имеющее тип BLOB с подтипом 1 (текстовый блоб) и размером сегмента 400. Для таблицы определен первичный ключ, состоящий из трех полей JOB\_CODE, JOB\_GRADE и JOB\_REGION. Далее, определен внешний ключ (JOB\_REGION), ссылающийся на поле REGION таблицы REGION. И, наконец, включено предложение CHECK, позволяющее производить проверку соотношения для двух полей и вызывать исключительное состояние при нарушении такого соотношения.

Пример D: Таблица с вычисляемым полем

CREATE TABLE SALARY\_HISTORY (

EMP\_NO EMPNO NOT NULL,

CHANGE\_DATE DATE DEFAULT "NOW" NOT NULL,

UPDATER\_ID VARCHAR(20) NOT NULL,

OLD\_SALARY SALARY NOT NULL,

PERC\_CHANGE DOUBLE PRECISION DEFAULT 0 NOT NULL

CHECK (PERC\_CHANGE BETWEEN -50 AND 50),

NEW\_SALARY COMPUTED BY

(OLD\_SALARY + OLD\_SALARY \* PERC\_CHANGE / 100),

PRIMARY KEY (EMP\_NO, CHANGE\_DATE, UPDATER\_ID),

FOREIGN KEY (EMP\_NO) REFERENCES EMPLOYEE (EMP\_NO));

Данный пример создает таблицу, где среди других полей имеется вычисляемое (физически не существующее) поле NEW\_SALARY, значение которого вычисляется по значениям двух других полей (OLD\_SALARY и PERC\_CHANGE).

На диске приведен пример скрипта, создающего базу данных, осуществляющую ведение контактов между людьми и организациями.

Заключение

Итак, мы рассмотрели, как создавать таблицы с помощью SQL-выражений. Этот процесс, хотя и не столь удобен, как интерактивное средство Database Desktop, однако обладает наиболее гибкими возможностями по настройке Вашей системы и управления ее связями.