8

Глава 8 – Работа с обекти в базата от данни

Тази глава се занимава с обектите в базите от данни като схеми, таблици, изгледи, индекси, последователности и т.н. Някои сложни обекти в приложенията на базите от данни като тригери, дефинирани от потребителя функции (UDFs) и съхранени процедури са разгледани в Глава 14 – SQL PL съхранени процедури и Глава 15 – Вътрешен SQL PL, тригери и функции, дефинирани от потребителя.

Забележка:
За повече информация относно обектите в базата от данни гледайте следния видео материал:
http://www.channeldb2.com/video/video/show?id=807741:Video:4242

8.1 Схема

Схемите представляват пространства от имена за съвкупност от обекти. Използват се основно, за да:

- Предоставят информация за принадлежността на обекта или връзката му с дадено приложение.
- Групират логически взаимосвързаните обекти

Всички обекти в DB2 имат име, което ги идентифицира уникално – пълно име на обекта (fully qualified name), състоящо се от две части. Схемата е първата половина от името:

<schema_name>.<object_name>

Пълното име на обекта трябва да бъде уникално. Когато се свържете към базата от данни и създадете или реферирате обект, без да уточните схемата му, DB2 използва за име на схемата потребителското ID, с което сте се свързали към базата от данни. Например, ако се свържете към базата от данни SMAPLE с потребителско име „chobanov” и създадете таблица с командата CREATE TABLE:

```
CREATE TABLE artists …
```

Пълното име на таблицата е всъщност chobanov.artists.

8.2 Таблици

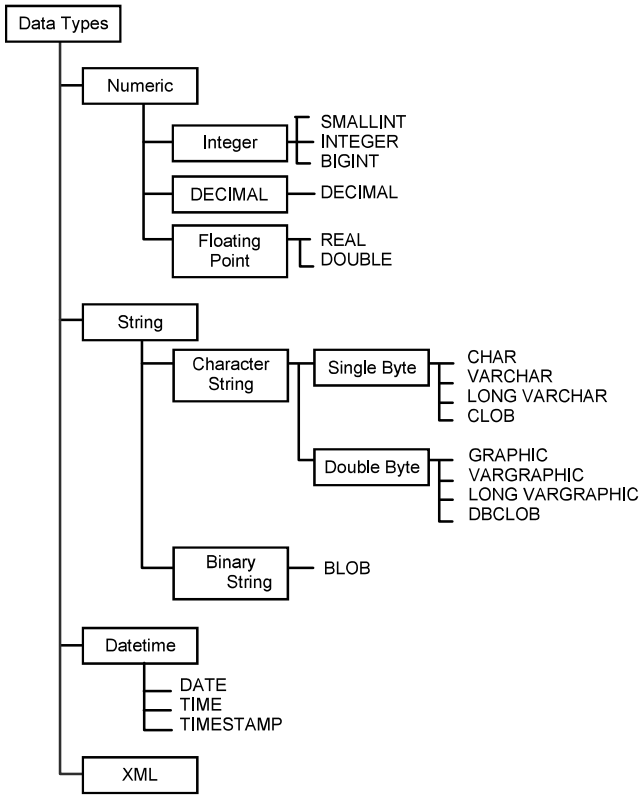
Таблицата е колекция от близки по смисъл данни, логически подредени в колони и редове. Примерът по-долу показва как се създава таблица с командата CREATE TABLE.

```
CREATE TABLE artists
 (artno           SMALLINT    not null,
  name            VARCHAR(50) with default 'abc',
  classification  CHAR(1)     not null,
  bio             CLOB(100K)  logged,
  picture         BLOB(2M)    not logged compact
 )
 IN mytbls1
```

В следващите секции ще опишем основните части на операцията CREATE TABLE.

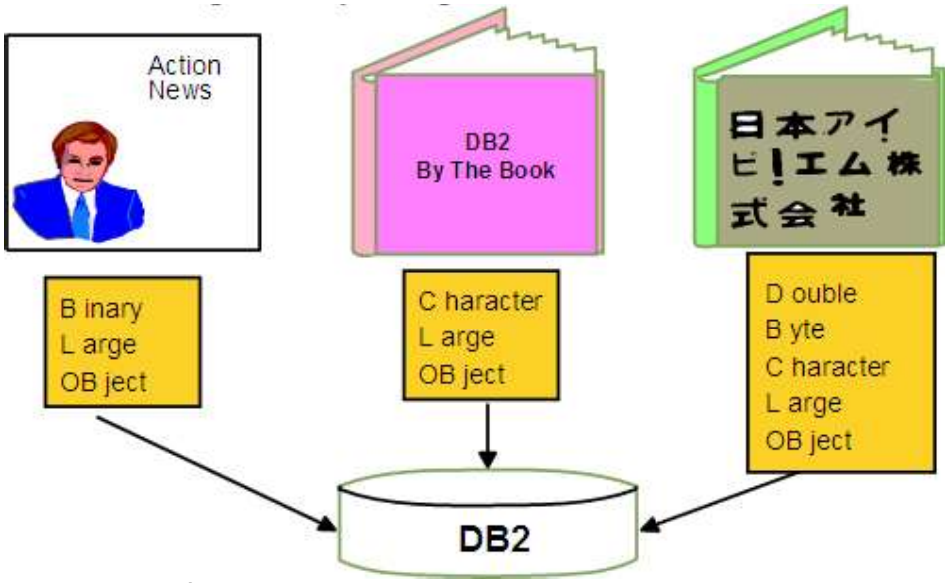
8.2.1 Типове данни

Фигура 8.1 показва типовете данни, поддържани от DB2.



Фигура 8.1 - Вградени типове данни в DB2

Типове данни за големи обекти (Large Objects(LOBs))
Тези типове данни се използват за съхранението на дълги символни последователности, големи двоични последователност или файлове, както е показано на Фигура 8.2



Фигура 8.2 – LOB типове данни

Названията на данните от тип големи обекти обикновено се съкращават за яснота: голям бинарен обект е BLOB, голям символен обект(character large object) е CLOB, а дву-битов голям символен обект(double byte character object) е известен още като DBCLOB.

Типове, дефинирани от потребителя
DB2 ви позволява да дефинирате свои типове данни, базирани на вградените типове. Те са известни като потребителски дефинирани типове (user-defined types - UDTs)*. Потребителските типове данни са полезни когато:

- има нужда да се създаде контекст за дадена стойност
- има нужда DB2 да приложи принудително типизиране*

Примерът по-долу показва кога и как се използват потребителски типове:

```
CREATE DISTINCT TYPE POUND AS INTEGER WITH COMPARISONS

CREATE DISTINCT TYPE KILOGRAM AS INTEGER WITH
COMPARISONS

CREATE TABLE person
   (f_name    VARCHAR(30),
    weight_p  POUND NOT NULL,
```

```
                weight_k  KILOGRAM NOT NULL )
```

В този пример се създават два потребителски типа: POUND и KILOGRAM. И двата използват за основа вградения тип INTEGER. Клаузата WITH COMPARISONS указва, че ще бъдат създадени функции за преобразуване на типовете със същото име като типовете данни.

Таблицата person използва двата нови потребителски типа данни в колоните `weight_p` и `weight_k` . Нека разгледаме следната заявка:

```
SELECT F_NAME FROM PERSON
  WHERE weight_p > weight_k
```

Ще получим съобщение за грешка, понеже се опитваме да сравним колони с различен тип данни. Въпреки че `weight_p` и `weight_k` използват типове данни, съответно POUND и KILOGRAM, които са базирани на типа INTEGER, чрез дефинирането на потребителски тип данни, правим това сравнение невъзможно. Това е точно, което искаме, защото какво би означавало сравнението между паундове и килограми? То не би имало смисъл.

В следващия пример се опитваме да сравним `weight_p` с цяло число (INTEGER). Тези два типа данни са различни и сравнението би предизвикало грешка, ако не използваме функция за преобразуване на типовете.

Както можете да видите от примера по-долу, използваме функцията за преобразуване POUND(), за да направим възможно сравнението. Както казахме преди малко, преобразуващата функция POUND() беше създадена с потребителския тип данни, чрез извикването на WITH COMPARISONS клаузата на CREATE DISTINCT TYPE командата.

```
SELECT F_NAME FROM PERSON
  WHERE weight_p > POUND(30)
```

NULL стойности

NULL стойността се използва за представяне на неизвестна стойност. За да се подсигури съществуването на стойност в дадена колона, можем да използваме NOT NULL клаузата на командата CREATE TABLE. Също така можем да зададем стойност по подразбиране, ако колоната е декларирана като NOT NULL. Следващия пример показва това поведение:

```
CREATE TABLE Staff (
     ID       SMALLINT NOT NULL,
     NAME     VARCHAR(9),
     DEPT     SMALLINT NOT NULL with default 10,
     JOB      CHAR(5),
     YEARS    SMALLINT,
     SALARY   DECIMAL(7,2),
     COMM     DECIMAL(7,2) with default 15
     )
```

8.2.2 IDENTITY колони

IDENTITY колоната е колона от числови стойности, тя автоматично генерира уникална стойност за всеки свой ред. Всяка таблица може да има само една IDENTITY колона.

Има два начина да се генерират стойностите в една IDENTITY колона, в зависимост от начина, по който тя е дефинирана:

- **Generated always:** стойностите се генерират винаги от DB2. Приложенията не могат да задават стойности на колоната.
- **Generated by default:** стойностите могат да бъдат въведени от приложение, но ако не е зададена стойност, DB2 генерира такава. DB2 не може да гарантира уникалност. Тази опция се използва при разпространение на данни, изпразване и презареждане съдържанието на таблица.

Нека разгледаме следния пример:

```
CREATE TABLE subscriber(subscriberID INTEGER GENERATED ALWAYS AS
                                     IDENTITY (START WITH 100
                                     INCREMENT BY 100),
                        firstname VARCHAR(50),
                        lastname  VARCHAR(50) )
```

В примера, колоната subscriberID е от тип INTEGER, дефинирана като IDENTITY колона, която винаги генерира сама стойностите си. Генерираните стойности ще започнат от 100 и ще нарастват със 100.

8.2.3 SEQUENCE обекти

Последователностите генерират уникален номер в рамките на цялата база от данни. За разлика от IDENTITY колоните, последователностите са независими от таблиците. Например:

```
CREATE TABLE t1 (salary int)

CREATE SEQUENCE myseq
   START WITH 10
   INCREMENT BY 1
   NO CYCLE

INSERT INTO t1 VALUES (nextval for myseq)
INSERT INTO t1 VALUES (nextval for myseq)
INSERT INTO t1 VALUES (nextval for myseq)

SELECT * FROM t1

SALARY
-----------
         10
         11
```

```
          12
  3 record(s) selected.


SELECT prevval for myseq FROM sysibm.sysdummy1

1
-----------
          12
  1 record(s) selected
```

PREVVAL е текущата стойност на последователността, а NEXTVAL е следващата.

Освен това горният пример използва SYSIBM.SYSDUMMY1. Това е таблица от системния каталог, която съдържа една колона и един ред. Може да бъде използвана в ситуации, когато заявката изисква извеждане на резултат, базиран на една единствена стойност. Таблиците от системния каталог са описани в следващата секция.

8.2.4 Таблици от системния каталог

Всяка база от данни има каталог със системни таблици и изгледи. Те съдържат *мета-данни* за обектите в базата. Можем да създаваме заявки за тези таблици по същия начин, както го правим за другите таблици от базата от данни. Използват се три схеми, за да се идентифицират таблиците от системния каталог:

- SYSIBM: основни таблици, оптимизирани за употреба от DB2
- SYSCAT: изгледи, базирани на таблиците от SYSIBM, оптимизирани за по-лесна работа
- SYSSTAT: статистики за базата от данни

А ето и някои примери за изгледи от системния каталог:

- SYSCAT.TABLES
- SYSCAT.INDEXES
- SYSCAT.COLUMNS
- SYSCAT.FUNCTIONS
- SYSCAT.PROCEDURES

8.2.5 Деклариране на временни таблици

Временните таблици са таблици, които се създават в оперативната памет и се използват единствено за работата на дадено приложение. След приключване работата на това приложение, те биват изтрити. Тези таблици могат да бъдат достъпвани само от приложението, което ги е създало. За тях няма никакви записи в таблиците от каталога на DB2. Достъпът до временните таблици предоставя висока ефективност, понеже няма конфликти при обръщение към каталога, заключване на редовете, водене на дневник (воденето на дневник не е задължително) и проверка на правата за достъп. Също така има и поддръжка на индексите за тези таблици, това означава, че всеки стандартен индекс, може да бъде създаден в една временна таблица. С тези таблици също така може да използвате командата RUNSTATS.

Временните таблици се съхраняват в пространството за временни таблици на потребителя, което трябва да бъде заделено, преди създаването на временните таблици. Следващия пример показва как се създават три временни таблици:

```
CREATE USER TEMPORARY TABLESPACE apptemps
     MANAGED BY SYSTEM USING ('apptemps');

DECLARE GLOBAL TEMPORARY TABLE temployees
     LIKE employee NOT LOGGED;

DECLARE GLOBAL TEMPORARY TABLE tempdept
      (deptid CHAR(6), deptname CHAR(20))
       ON COMMIT DELETE ROWS NOT LOGGED;

DECLARE GLOBAL TEMPORARY TABLE tempprojects
     AS ( fullselect ) DEFINITION ONLY
     ON COMMIT PRESERVE ROWS NOT LOGGED
     WITH REPLACE IN TABLESPACE apptemps;
```

Когато се създаде временна таблица, нейната схема е SESSION и трябва да се означи. Потребителското ID, с което е създадена таблицата, ще има всички права върху нея. Всяко приложение, което създава временна таблица, има собствено нейно копие, както е показано на Фигура 8.5.

Фигура 8.5 – Обхват на глобални временни таблици

Лабораторно упражнение #7: Създаване на нова таблица

Цел:

До сега използвахме таблиците от базата от данни *SAMPLE*, за да илюстрираме основните идеи. В крайна сметка, ще ви се наложи да създадете свои таблици във вашата база от данни. В това упражнение ще използваме *Create Table Wizard*, за да създадем две нови таблици в базата от данни *SAMPLE*.

Стъпки:

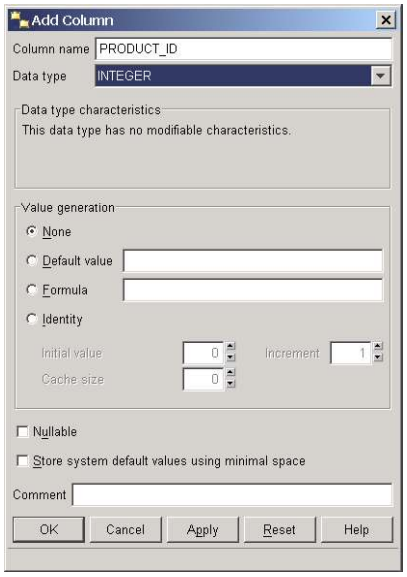
1. Стартирайте Create Table Wizard, както беше показано в презентацията. (*Control Center -> All Databases -> SAMPLE -> (десен бутон на мишката)Tables object -> Create …* опцията)

2. Задайте име на таблицата, имената и типът на колоните и ограниченията, които искате да са изпълнени. Таблицата ще пази информация за офис материали, използвани в проект от базата от данни *SAMPLE*. Всеки път, когато бъдат поръчани офис материали, ще се добавя нов ред в таблицата. Тя съдържа следните шест колони:

   - product_id: уникален номер за продукта, който е поръчан
   - description: описание на продукта
   - quantity: количеството, което е поръчано
   - cost: цена на продукта
   - image: снимка (ако има такава)
   - project_num: проекта, за който е поръчан дадения продукт

3. На първата страница от помощника, задайте за схема потребителския идентификатор (ID), с който сте се записали. Задайте “*SUPPLIES*” за име на таблицата. Може да добавите коментар, ако желаете. Натиснете бутона *Next*, за да преминете на следваща страница.

4. От тази страница можете да добавяте колони към таблицата. Натиснете бутона *ADD*, за да добавите колона.



Задайте „product_id" за име на колоната и изберете да е от тип: INTEGER. Махнете отметката от опцията *Nullable* и натиснете бутона *Apply*, за да добавите колоната.

Повторете тази стъпка за останалите колони, като използвате данните от таблицата по-долу. След като добавите всички колони, натиснете бутона *OK*, за да се появи списък с колоните, които сте създали. Натиснете бутона *Next*, за да преминете на следващата страница.

Име на колона	**Атрибути**
product_id (completed)	INTEGER, NOT NULL
description	VARCHAR, length 40, NOT NULL
quantity	INTEGER, NOT NULL
cost	DECIMAL, Precision 7, Scale 2, NOT NULL
image	BLOB, 1MB, NULLABLE, NOT LOGGED
project_num	CHAR, length 6, NOT NULL

Забележка: Опцията NOT LOGGED може да бъде зададена, когато колоната е от тип голям обект (LOB). Задължителна е за колоните, надхвърлящи 1GB. Също така се препоръчва за колони над 10MB, понеже при промяна към по-голям размер лог файлът може да бъде запълнен по-бързо. Дори и да бъде използвана опцията NOT LOGGED, транзакциите върху LOB файловете пак могат да бъдат

отменяни успешно. Забележете също, че единствено колоната *image* е зададена като „NULLABLE”. Защо е така според вас?

5. До този момент цялата необходима информация за дефиниране на таблица е въведена. Пропускайки другите страници от помощника, вие избирате стойностите по подразбиране за съответните опции. Винаги можете да зададете ключове и ограничения в таблицата, след като сте я създали.

6. Добавете ограничение към колоната *quantity* от таблицата. На страницата *Constraint* от помощника, натиснете бутона *ADD*. В полето *Check Name* въведете: valid_quantities. В полето *Check Condition* въведете: *quantity* > 0.

   Натиснете бутона *OK*. Би трябвало да видите обобщение на ограничението, което добавихте току-що, на страницата *Constraint*. Натиснете бутона *Next*, за да продължите към следващата страница от помощника.

7. Може да продължите към следващите страници на помощника и да промените другите параметри на таблицата. Също така можете да преминете направо към страницата *Summary* или просто да натиснете бутона *Finish*, за да завършите таблицата.

8. От Контролния Център, натиснете върху папката *Tables* в базата от данни *SAMPLE* в Панела с дървото на обектите. Таблицата, която създадохте току-що, трябва да е в списъка. Би могло да се наложи да презаредите Контролния Център, за да видите промените.

8.3 Изгледи

Изгледът е начин за представяне на данните в таблиците. Информацията за изгледа не се пази отделно, а се извлича от таблиците, при извикване на даден изглед. Поддържат се и вложени изгледи или изгледи, които се базират на други изгледи. Цялата информация за изгледите се пази в следните изгледи от DB2 каталога: SYSCAT.VIEWS, SYSCAT.VIEWDEPT и SYSCAT.TABLES. Ето пример как да създадете и използвате изглед.

```
CONNECT TO MYDB1;

CREATE VIEW MYVIEW1
  AS SELECT ARTNO, NAME, CLASSIFICATION
  FROM ARTISTS;

SELECT * FROM MYVIEW1;

Output:

ARTNO            NAME                 CLASSIFICATION
------           -----------------    --------------
  10             HUMAN                  A
  20             MY PLANT               C
  30             THE STORE              E
  ...
```

8.4 Индекси

Индексът е подредено множество от ключове, всеки от които сочи към ред от таблица. Индексът позволява уникалност, също така подобрява производителността. Ето някои от характеристиките на индексите, които можете да дефинирате:

- Подредбата в индекса може да е нарастваща или намаляваща
- Ключовете от индекса могат да бъдат или да не бъдат уникални
- Няколко колони могат да бъдат използвани, за да формират индекса (това се нарича съставен индекс)
- Ако индексът и физическите данни са групирани в подобна последователност, те са групов (клъстерен) индекс.

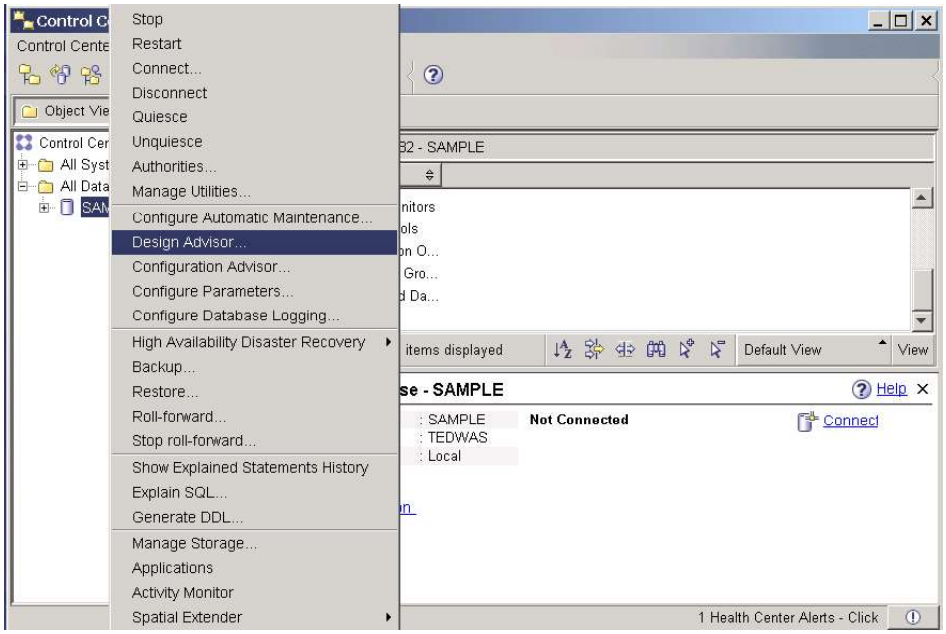
Например:

```
CREATE UNIQUE INDEX artno_ix ON artists (artno)
```

8.4.1 Съветник по дизайна

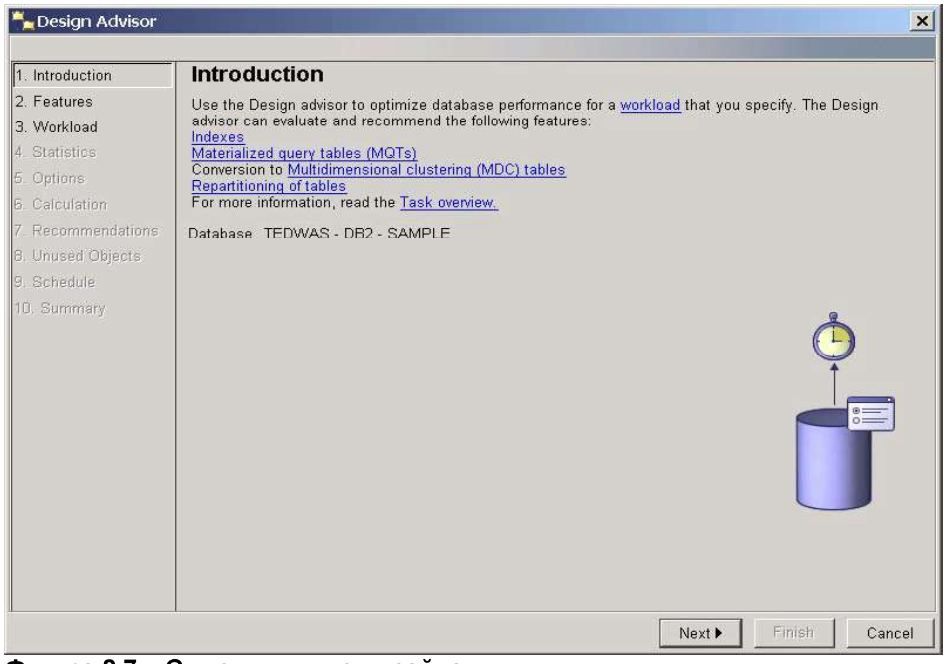
Съветникът по дизайна (Design Advisor) е чудесен инструмент, който може да ви посъветва как да направите оптимален дизайн на своята база от данни при дадено натоварване. Съветникът по дизайна може да ви помогне с дизайна на вашите индекси, Материализирани таблици от заявки (Materialized Query Tables - MQTs), Многомерно групиране (Multi-dimension clustering - MDC), и функцията за разпределяне на базата от данни. Съветникът по дизайна може да бъде извикан от

Контролния Център. Натиснете с десен бутон на мишката върху базата от данни, след което изберете „Design Advisor", както е показано на Фигура 8.6.



Фигура 8.6 – Стартиране на Съветника по дизайна (Design Advisor) от Контролния Център

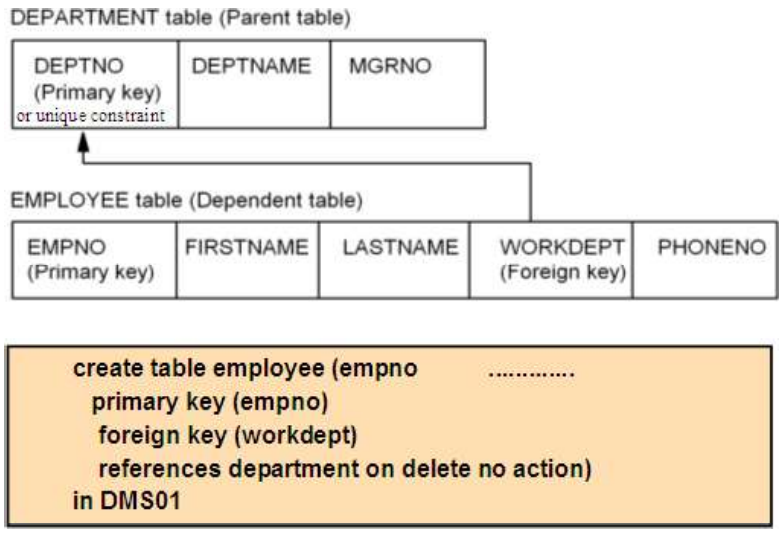
Фигура 8.7 показва Съветника по дизайна. Следвайте стъпките в помощника, за да получите съветите за дизайн от DB2.



Фигура 8.7 – Съветникът по дизайна

8.5 Ограничения за референтна цялостност

Ограниченията за референтна цялостност позволяват на базата от данни да поддържа връзки между таблиците. Можете да осъществите връзка от тип родител-дете между таблици, както е показано на Фигура 8.8. На фигурата има 2 таблици, DEPARTMENT и EMPLOYEE, свързани по номера на отдела. Колоната WORKDEPT на таблицата EMPLOYEE може да съдържа единствено номера на отдели, които съществуват в таблицата DEPARTMENT. Това е така, защото в примера таблицата DEPARTMENT е родителската таблица, а EMPLOYEE е зависима или още дъщерна таблица. Фигурата също така показва командата CREATE TABLE,
необходима на таблицата EMPLOYEE, за да се създаде връзката.



Фигура 8.8 – Ограничения за референтна цялостност върху таблиците

При ограниченията за референтна цялостност често се използват следните концепции:

Концепция	Описание
Родителска таблица	Контролираща таблица, в която се намира родителският ключ
Зависима таблица	Таблица, зависима от данните в родителската таблица. Освен това съдържа външен ключ. За да съществува даден ред в зависимата таблица, в родителската таблица трябва да съществува ред, съдържащ същия ключ.
Първичен ключ	Дефинира родителския ключ в родителската таблица. Не може да съдържа NULL или неуникални стойности. Първичният ключ се състои от една или повече колони от таблицата.
Външен ключ	Свързва зависимата таблицата с първичния ключ на родителската таблица.

Данни от една таблица могат да бъдат свързвани с данни от други таблици посредством ограниченията за референтна цялостност. Могат да бъдат въведени и допълнителни ограничения в стойностите на полетата, за да отговарят на определени изисквания и цели. Например, ако една колона в таблица трябва да пази пола на даден човек, ограничението може да регламентира въвеждането само на стойности “M” за мъже и “F” за жени.