УДК 519.711.3

СИНТЕЗ СХЕМ БАЗ ДАННЫХ НА ОСНОВЕ ОНТОЛОГИИ

М.В. Кучуганов

Удмуртский государственный университет, Ижевск, Россия
qmikle1@yandex.ru

Аннотация

Рассматривается задача автоматического синтеза схемы реляционной базы данных по описанию онтологии предметной области в дескрипционной логике. Предлагается алгоритм экспликации и анализа сведений о связях концептов, релевантных задаче генерации схемы базы данных. Приводится пример генерации схемы фрагмента базы данных производственной организации. Отличие предлагаемого алгоритма от уже существующих алгоритмов генерации схем баз данных для онтологий предметной области, описанных на языке дескрипционной логики, заключается в том, что он ориентирован на создание базы данных для хранения информации (фактов) о состоянии предметной области, которая пригодна не только для решения логических запросов со стороны базы знаний (основанной на дескрипционной логике и логическом выводе), но и для обработки обычных SQL запросов со стороны любых других инженерных приложений (информационных систем), работающих с этой информацией. Полученная в результате логического анализа онтологии схема базы данных достаточна для хранения информации о любом не противоречащем онтологии состоянии предметной области, гарантирует ссылочную целостность данных и позволяет организовать эффективный доступ к ним.

Ключевые слова: *онтология, дескрипционная логика, база данных, реляционная модель, синтез баз данных.*

Цитирование: *Кучуганов, М.В*. Синтез схем баз данных на основе онтологии / М.В. Кучуганов // Онтология проектирования. – 2016. – Т. 6, №4(22). - С. 475-484. – DOI: 10.18287/2223-9537-2016-6-4-475-484.

Введение

Одним из наиболее популярных в последние годы средств описания онтологии предметной области (ПрО) является дескрипционная логика [1, 2].

Описание *Th* = [*C, R, TBox, ABox*] ПрО в дескрипционной логике (ДЛ) далее называется *теорией (ДЛ-онтологией*) ПрО и включает в себя следующее:

1) *C* – множество атомарных концептов;
2) *R* – множество атомарных ролей;
3) *TBox* – множество утверждений о свойствах концептов и ролей (*описание понятий*);
4) *ABox* – множество утверждений о конкретных объектах (*описание фактов*).

Информация о конкретном состоянии ПрО описывается утверждениями из *ABox*.

Рассмотрим задачу построения (схемы) реляционной базы данных (БД) для хранения информации о состоянии ПрО – фактов, которые описываются в ДЛ утверждениями из *ABox.* Очевидно, есть тривиальное решение – БД, состоящая из отдельных таблиц для каждого концепта и для каждой роли. В принципе, достаточно двух таблиц – одна для всех концептов, другая – для всех ролей. Но это решение никак не отражает сути онтологии – утверждений о свойствах концептов и ролей из *TBox* и никак не использует их для *эффективного* извлечения *релевантных к запросу* фактов из БД.

С другой, практической стороны, реляционные БД предназначены для *эффективного* хранения и извлечения нужных фактов о ПрО. И успешно используются для этого десятки лет. Не случайно задача построения ДЛ-описаний *реальных* реляционных БД рассматривается давно и остаётся актуальной (см. обзор этого направления исследований в работе [3]).

Имеющиеся в современных реляционных БД механизмы контроля целостности дают возможность построить *структурированную* схему БД для хранения фактов *ABox,* исходя из логических следствий аксиом *TBox* в ДЛ–описании ПрО.

Предлагаемые в настоящее время методы построения реляционной БД для хранения фактов *ABox,* ориентированы на эффективное выполнение *логических* запросов к *ABox*, основанных на понятиях и определениях из *TBox.* Так, например, в работах [4, 5] предлагаются различные способы трансляции терминологических выражений ДЛ в иерархически-реляционные структуры схемы данных, позволяющие существенно упростить верификацию утверждений ДЛ. Однако предлагаемые в этих работах принципы построения схемы БД предусматривают создание множества «дополнительных» таблиц и связей, повышающих эффективность вычисления логических запросов, но существенно усложняющих обычную, «повседневную» обработку данных традиционными программными системами.

Проблема синтеза БД, обеспечивающих работу обычных программных систем управления, например, производственными процессами не рассматривается. По этой причине бывает затруднительно обеспечить взаимосвязь информационных систем (ИС), использующих *логические* запросы к базе знаний ПрО (основанной на ДЛ и логическом выводе), с традиционными инженерными ИС, использующими обычные SQL запросы для обработки данных в реляционной базе данных.

Суть предлагаемого решения проблемы заключается в экспликации (извлечении) и анализе отношения вложенности концептов и свойств ролей в ДЛ-онтологии ПрО и последующей генерации SQL-скрипта для создания схемы БД. Предлагается алгоритм, позволяющий автоматически генерировать схему реляционной БД не только *достаточную* для хранения информации о любом допустимом состоянии ПрО, но и пригодную для работы обычных программных систем, не использующих ДЛ. Приводится пример генерации (фрагмента) схемы реляционной БД для некоторой производственной организации.

1 Алгоритм анализа онтологии предметной области

Предполагается, что ДЛ-онтология ПрО является *концептуально полной,* т.е. имеет достаточно явно выделенных *атомарных* концептов.

***Определение* 1.** Онтология $Th = [C, R, TBox, ABox]$ называется *концептуально полной*, если выполняются следующие условия:

1) для каждого концепта $c \in C$ имеется (единственный, представляющий собой функцию от концепта c) *наименьший* по отношению вложенности концепт $parent(c) \in C$ такой, что $c \subseteq parent(c)$;
2) для каждой роли $r \in R$ имеется (единственный, представляющий собой функцию от роли r) *наименьший* по отношению вложенности концепт $dom(r) \in C$, содержащий область *определения* роли r (как бинарного отношения);
3) для каждой роли $r \in R$ имеется (единственный, представляющий собой функцию от роли r) *наименьший* по отношению вложенности концепт $range(r) \in C$, содержащий область *значений* роли r (как бинарного отношения).

Структура концептуально полной онтологии наглядно представляется в виде ориентированного графа (*концептуальной диаграммы* онтологии), вершины которого представляют концепты, а дуги – определённые в онтологии роли.

Описываемый вариант алгоритма будем далее называть DLSQL1.

Предполагается, что имеется механизм логических рассуждений (или reasoner, рассуждатель), вычисляющий отношение логического следования $IMPLY(TBox, \varphi) = (TBox \models \varphi)$ - «утверждение φ *логически следует* из утверждений *TBox*».

Из приводимого далее описания алгоритма видно, что с логическим выводом связан только первый этап алгоритма – экспликация сведений о связях концептов из исходного ДЛ-описания ПрО.

Далее n – количество концептов, m – количество ролей теории *Th.*

1.1 Этап 1. Экспликация сведений о связях концептов в ДЛ-онтологии

ШАГ 1.1. Определение вложенности концептов и формирование вспомогательного массива индексов PARENT с учётом концептуальной полноты онтологии.

Массив PARENT содержит, для каждого концепта $C[i]$, индекс концепта $parent(C[i])$. Этот массив однозначно определяет отношение вложенности на атомарных концептах.

```
for i:=1 to n do begin
  PARENT[i]:=-1; //ещё не нашли концепт, больше i-го
  for j:=1 to n do
    if (i<>j) and IMPLY(Th, C[i] ⊆ C[j]) then
      if PARENT[i]=-1 then PARENT[i]:=j //нашли больший концепт
      else if IMPLY(Th, C[j] ⊆ C[PARENT[i]]) then PARENT[i]:=j;
           //уточнили больший
end;
```

ШАГ 1.2. Определение концептов-областей определения ролей и формирование вспомогательного массива индексов DOM с учётом концептуальной полноты онтологии.

Понятие области определения роли $r \in R$ является определяемым концептом в ДЛ *ALC* (*Attributive Language with Complement*) и описывается формулой $Dom(r) = \exists r.\mathbf{T}$[1].

Массив DOM содержит, для каждой роли $R[i]$, индекс концепта $dom(R[i])$.

```
for i:=1 to m do begin
  DOM[i]:=-1; //еще не нашли концепт, больше i-го
  for j:=1 to n do
    if IMPLY(Th, Dom(R[i]) ⊆ C[j]) then
      if DOM[i]=-1 then DOM[i]:=j //нашли больший концепт
      else if IMPLY(Th, C[j] ⊆ C[DOM[i]]) then DOM[i]:=j;
           //уточнили больший
end;
```

ШАГ 1.3. Определение концептов-областей значений ролей и формирование вспомогательного массива индексов RANGE.

Понятие области значений роли $r \in R$ является определяемым концептом в ДЛ с обратными ролями *ALCI* и описывается формулой $Range(r) = \exists(r^{-}).\mathbf{T}$.

По условию концептуальной полноты онтологии для каждой роли $r \in$ R имеется (единственный) *наименьший* по отношению вложенности концепт $range(r) \in C$, содержащий область значений роли r (как бинарного отношения).

Массив RANGE содержит, для каждой роли $R[i]$, индекс концепта $range(R[i])$.

[1] $\mathbf{T}$ – символ *ALC*-логики, который интерпретируется как весь домен - множество индивидов, представители которого составляют множество объектов описываемой ПрО. Выражение $\exists r.\mathbf{T}$ обозначает на языке *ALC* область определения (т.е. определённые объекты рассматриваемой ПрО) роли *r*. *Прим. ред.*

```
for i:=1 to m do begin
  RANGE[i]:=-1; //еще не нашли концепт, больший i-го
  for j:=1 to n do
    if IMPLY(Th, Range(R[i]) ⊆ C[j]) then
      if RANGE[i]=-1 then RANGE[i]:= j //нашли больший концепт
        else if IMPLY(Th, C[j] ⊆ C[RANGE[i]]) then RANGE[i]:=j;
             //уточнили больший
end;
```

ШАГ 1.4. Определение функциональности ролей и формирование вспомогательного массива FUNC.

Понятие функциональности роли $r \in R$ является определяемым в ДЛ с численными ограничениями *ALCQ* и описывается формулой $Func(r) = (Dom(r) \subseteq (\leq 1.r.\mathbf{T}))$

Массив FUNC для каждой роли *R*[*i*] содержит признак - является ли она функциональной или нет.

В ДЛ атрибуты объектов ПрО описываются ролями-функциями. Фактически, массив FUNC определяет множество полей таблиц БД.

```
for i:=1 to m do FUNC[i]:=IMPLY(Th, Func(R[i]));
```

ШАГ 1.5. Определение свойства *Not Null* ролей-атрибутов и формирование вспомогательного массива NOTNULL.

Свойство *Not Null* означает, что атрибут является обязательным для объектов концепта – области определения и позволяет наложить соответствующее ограничение на значения поля таблицы, где будут храниться значения атрибута.

Массив NOTNULL для каждой роли *R*[*i*] содержит признак - обладает ли она свойством *Not Null* или нет.

```
for i:=1 to m do NOTNULL[i]:=FUNC[i] and IMPLY(Th, C[DOM[i]] ⊆ Dom(R[i]));
```

ШАГ 1.6. Определение свойства *Unique* ролей–атрибутов и формирование вспомогательного массива UNIQUE.

Свойство *Unique* означает, что значения атрибута обязательны и различны у различных объектов концепта-области определения и позволяет наложить соответствующее ограничение на значения поля таблицы, где будут храниться значения атрибута.

Массив UNIQUE для каждой роли *R*[*i*] содержит признак - обладает ли она свойством *Unique* или нет.

```
for i:=1 to m do UNIQUE[i]:=NOTNULL[i] and IMPLY(Th, 1.(R[i]⁻).C[DOM[i]]);
```

1.2 ЭТАП 2. Анализ сведений о связях концептов

ШАГ 2.1. Определение концептов – таблиц БД и формирование вспомогательного массива TABLE.

В ДЛ атрибуты объектов ПрО описываются ролями–функциями. Каждому концепту - области определения функциональной роли (*табличному*) соответствует таблица БД, в которой и будут храниться значения этого атрибута.

Массив TABLE для каждого концепта содержит признак - является ли он табличным или нет.

```
for i:=1 to n do begin
  TABLE[i]:=false;
  for j:=1 to m do
    if FUNC[j] and (DOM[j]=i) then TABLE[i]:=true;
end;
```

ШАГ 2.2. Определение связей концептов с табличными концептами и формирование вспомогательного массива PARENT_TABLE.

Массив PARENT_TABLE для каждого концепта содержит индекс *табличного* концепта-родителя.

```
for i:=1 to n do begin
  PARENT_TABLE[i]:=i;
  while not TABLE[PARENT_TABLE[i]] do //ищем ближайший табличный концепт
    PARENT_TABLE[i]:=PARENT[PARENT_TABLE[i]];
end;
```

ШАГ 2.3. Определение связей ролей с табличными концептами и формирование вспомогательных массивов DOM_TABLE и RANGE_TABLE.

Массивы DOM_TABLE и RANGE_TABLE для каждой роли содержат индексы *табличного* концепта области определения и области значений роли соответственно.

```
for i:=1 to m do begin
  DOM_TABLE:=PARENT_TABLE[DOM[i]];
  RANGE_TABLE:=PARENT_TABLE[RANGE[i]];
end;
```

ШАГ 2.4. Определение концептов – доменов БД и формирование вспомогательного массива DB_DOMAIN.

Домен базы данных (тип данных, некоторое *множество значений атрибутов* объектов), как концепт, отличается от остальных следующими свойствами:

- не является подмножеством табличного концепта (табличные концепты представляют *множества разносортных объектов* ПрО);
- не является областью определения роли (роли связывают *объекты* ПрО с другими объектами или со значениями их атрибутов);

Массив DB_DOMAIN для каждого концепта содержит признак - является ли он доменом БД или нет.

```
for i:=1 to n do begin
  DB_DOMAIN[i]:=(PARENT_TABLE[i]=-1); //не является подмножеством
  for j:=1 to m do // не является областью определения
    if DOM[j]=i then DB_DOMAIN[i]:=false;
end;
```

2 Пример генерации схемы реляционной БД для предметной области

В качестве примера рассмотрим задачу синтеза БД для некоторой производственной организации.

2.1 Описание концептов и ролей

Укрупнённое описание деятельности, связанной с движением ресурсов, содержит, в частности, следующие понятия (концепты и роли).

2.1.1 Концепт Документы

Концепт связан с другими следующими ролями:

- *НомерДокумента*(*x: Документы, y: STRING*) – строка *y* является уникальным идентификатором (кодом, номером) *документа x,* функциональное отношение (атрибут);

- *НаименованиеДокумента*(*x: Документы, y: STRING*) – строка *y* является наименованием документа *x,* функциональное отношение (атрибут);
- *ДатаДокумента*(*x: Документы, y: DATETIME*) – дата документа *x,* функциональное отношение (атрибут).

2.1.2 Концепт ЕдиницыИзмерения

Концепт связан с другими следующими ролями:

- *НаименованиеЕдиницы*(*x: ЕдиницыИзмерения, y: STRING*) – строка *y* является идентификатором (уникальным наименованием) единицы измерения *x,* функциональное отношение (атрибут);
- *ОбозначениеЕдиницы*(*x: ЕдиницыИзмерения, y: STRING*) – строка *y* является обозначением (уникальным сокращением наименования) единицы измерения *x,* функциональное отношение (атрибут).

2.1.3 Концепт Ресурсы

Концепт связан с другими следующими ролями:

- *НаименованиеРесурса*(*x: Ресурсы, y: STRING*) – строка *y* является идентификатором (уникальным наименованием) ресурса *x,* функциональное отношение (атрибут);
- *ЕдиницыИзмеренияРесурса*(*x: Ресурсы, y: ЕдиницыИзмерения*) – единица измерения *y* используется для измерения количества ресурса *x,* атомарная роль, определяющая множество допустимых единиц измерения ресурса.

2.1.4 Концепт Спецификации

Концепт связан с другими следующими ролями:

- *РесурсСпецификации*(*x: Спецификации, y: Ресурсы*) – ресурс *y* является предметом спецификации *x,* функциональное отношение (атрибут);
- *ЕдиницаСпецификации*(*x: Спецификации, y: ЕдиницыИзмерения*) – единица измерения *y* используется для измерения количества ресурса спецификации *x,* функциональное отношение (атрибут);
- *КоличествоСпецификации*(*x: Спецификации, y: FLOAT)* – число *y* является количеством ресурса спецификации*,* функциональное отношение (атрибут).

2.1.5 Концепт План

Концепт связан с другими следующими ролями:

- *НачалоПериода*(*x: План, y: DATETIME*) – дата начала периода планирования плана *x,* функциональное отношение (атрибут);
- *КонецПериода*(*x: План, y: DATETIME*) – дата окончания периода планирования плана *x,* функциональное отношение (атрибут);
- *РесурсыПлана*(*x: План, y: Спецификации*) – спецификация *y* является спецификацией количества ресурса плана *x,* атомарная роль, определяет множество спецификаций ресурсов плана.

2.2 Анализ онтологии и генерация схемы БД

Рассмотрим более подробно анализ онтологии и генерацию SQL-скрипта на примере обработки концепта План и его связей.

Поскольку (Шаг 2.1. Алгоритма) концепт *План* является областью определения функциональных ролей – атрибутов *НачалоПериода* и *КонецПериода*, то он является *табличным*, и

в схеме БД будет (должна быть) таблица, в которой будут храниться эти атрибуты для каждого экземпляра документа вида *План*.

Соответственно, будет сгенерирована следующая SQL-команда создания таблицы:

```
create table PPlan(ID integer not null);
```

Здесь (как и в других основных таблицах) поле ID предназначено для хранения уникального идентификатора объекта ПрО и является главным ключом таблицы, поэтому добавляется соответствующее ограничение:

```
alter table PPlan add constraint PK_PPlan primary key(ID);
```

Концепт *План* (Шаг 1.1 Алгоритма) является подмножеством концепта *Документы*, т.е. каждый план является документом и, соответственно, идентификатор плана является идентификатором документа. Поэтому добавляем соответствующее ограничение *foreign key*:

```
alter table PPlan add constraint FK_PPlan foreign key(ID) references Documents(ID);
```

Далее в таблице создаются поля для хранения значений атрибутов:

```
alter table PPlan add BegDate TIMESTAMP not null;
alter table PPlan add EndDate TIMESTAMP not null;
```

Для хранения *всех* нефункциональных ролей (отношений «один ко многим») используется *одна* «системная» таблица ROLES. Она содержит колонку NAME, которая содержит имя роли и колонки для хранения идентификаторов объектов табличных концептов. В каждой записи таблицы хранятся имя роли и ID объектов из таблиц – областей определения и значений этой роли – в соответствующих колонках, чтобы можно было контролировать ссылочную целостность БД.

Таблица создаётся при наличии в ДЛ-онтологии нефункциональных ролей (как в данном случае) следующими SQL-командами:

```
create table roles(Name varchar(127) not null);
create index roles_name on roles(Name);
```

Концепт *План* (Шаг 2.2. Алгоритма) является областью определения *нефункциональной* роли *РесурсыПлана*, которая сопоставляет каждому плану *множество* его спецификаций (позиций плана) из концепта Спецификации.

Следовательно, в таблице ROLES, где хранятся все нефункциональные отношения, должны быть поля для хранения пар «идентификатор плана – идентификатор спецификации». Добавляем их в таблицу ROLES вместе с соответствующими ограничениями, гарантирующими ссылочную целостность:

```
alter table roles add PPlan_ID integer;
alter table roles add constraint FK_PPlan_ID foreign key (PPlan_ID) references PPlan(ID);
alter table roles add Specs_ID integer;
alter table roles add constraint FK_Specs_ID foreign key (Specs_ID) references Specs(ID);
```

Аналогично обрабатываются все табличные концепты и формируется SQL-скрипт, показанный на рисунке 1.

Скрипт может исполняться в любой из распространённых реляционных СУБД.

Разумеется, перед его исполнением нужно:

- инициализировать схему БД с желаемыми техническими параметрами (размер страницы, кодировка данных и прочие, в зависимости от особенностей используемой СУБД);
- создать домены данных для концептов-доменов, которые *не являются стандартными* в данной СУБД.

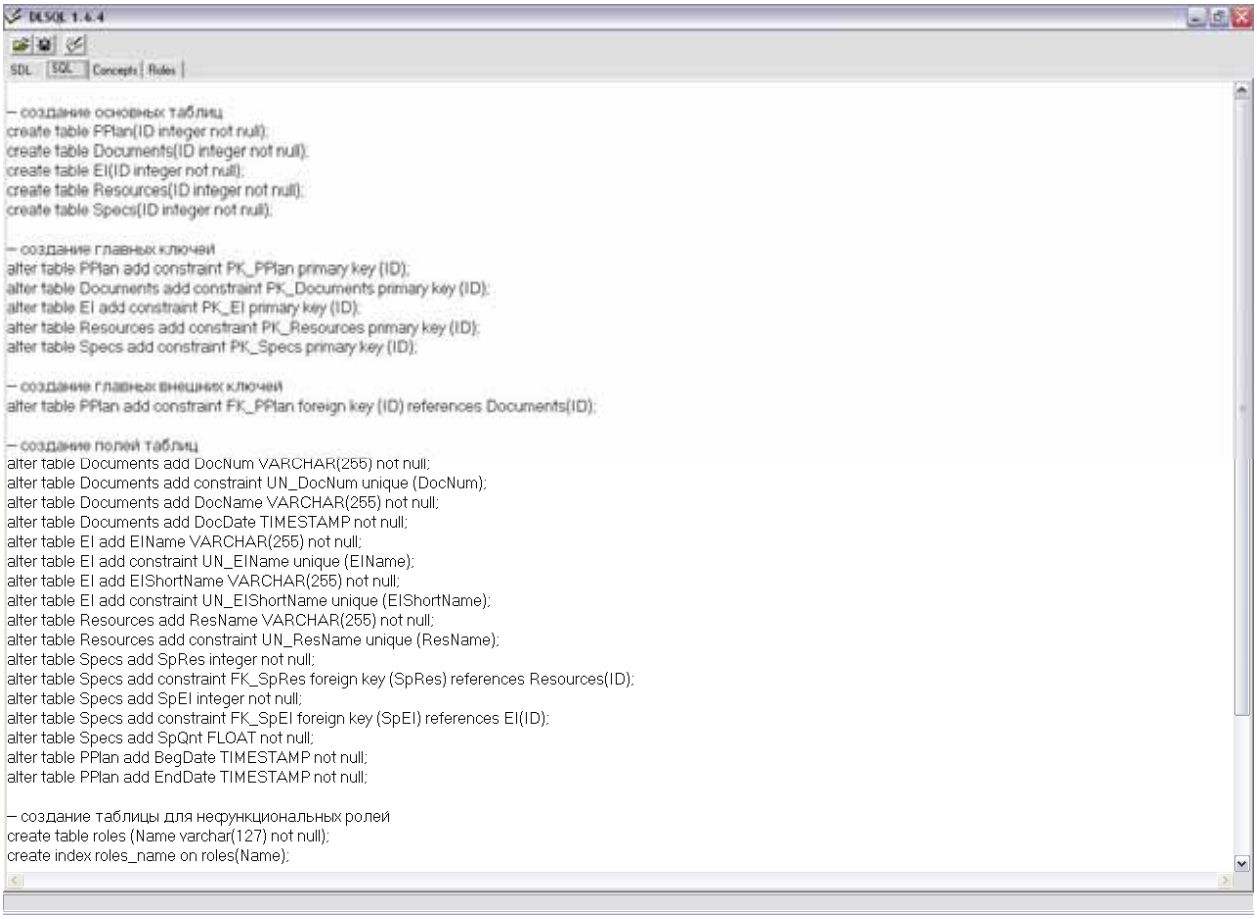

Рисунок 1 – SQL-скрипт, полученный в результате анализа онтологии

На рисунке 2 показана диаграмма БД, созданной в результате исполнения скрипта в СУБД Firebird в программе-консоли EMS SQL Manager Lite for InterBase and Firebird. Она наглядно показывает таблицы документов, планов, спецификаций, ресурсов, единиц измерения ресурсов и ролей (отношений «один ко многим») и связи между ними.

Полученная в результате анализа ДЛ-онтологии БД позволяет хранить информацию о любом не противоречащем онтологии состоянии ПрО и позволяет организовать эффективный доступ к данным об этом состоянии не только со стороны базы знаний ПрО, но и со стороны любых других инженерных приложений, работающих с информацией о ПрО.

Заключение

Предложенный алгоритм позволяет автоматически синтезировать схему реляционной БД по ДЛ-онтологии ПрО.

Приведённый пример показывает, что синтезируемая схема БД получается «естественной» – понятной и удобной для разработчиков прикладных программных систем, работающих с этими данными. Взаимосвязь задач через онтологию ПрО полезна при разработке корпоративных информационных систем. Описанная версия алгоритма экспликации и анализа сведений о связях понятий ПрО является базовой и допускает дальнейшее развитие с целью извлечения из ДЛ-онтологии более точных сведений о связях концептов и генерации более сложных ограничений в схеме БД.

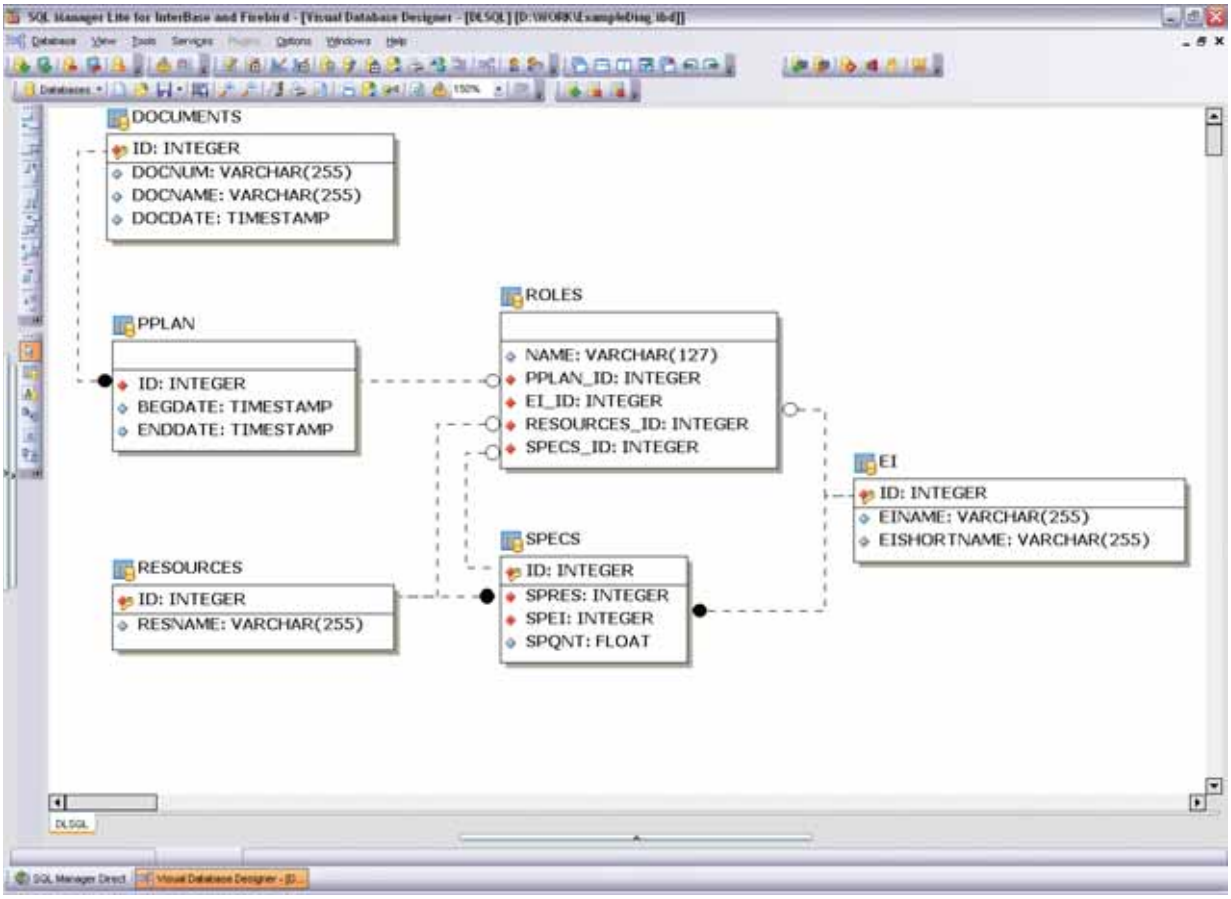

Рисунок 2 - Диаграмма схемы БД, полученной в результате исполнения скрипта

Список источников

[1] ***Baader, F.*** The Description Logic Handbook / F. Baader. - New York: Cambridge University Press, 2003.

[2] ***Конев, Б.Ю.*** Онтология и представление знаний / Б.Ю. Конев. - Department of Computer Science, Liverpool University, 2010. – URL: http://logic.pdmi.ras.ru/csclub/courses/ontology (Дата обращения 12.10.2016).

[3] ***Sequeda, J.*** Integrating relational databases with the Semantic Web / J. Sequeda. - [Электронный ресурс]. – URL: https://repositories.lib.utexas.edu/handle/2152/30537 (Дата обращения 16.08.2016).

[4] ***Auer, S. and I., Zachary, G.*** Integrating Ontologies and Relational Data / S. and I. Auer, G. Zachary // Technical Reports (CIS). Paper 716. – URL: http://repository.upenn.edu/cis_reports/716 (Дата обращения 16.08.2016).

[5] ***Левков, А.А.*** Организация эффективной системы хранения фактов в онтологиях /А.А. Левков // Информационные технологии и вычислительные системы. – 2011. - №4. – С. 3-9.

ONTOLOGY BASED SYNTHESIS OF DATABASE SCHEMES

M.V. Kuchuganov

Udmurt State University, Izhevsk, Russian
qmikle1@yandex.ru

Abstract

The article discusses the problem of automatic synthesis of a relational database schema to store information about states of a subject area by using its description logic ontology. It presents an algorithm for explication and analysis of

information about those relations between concepts, which are relevant to the problem. Example of generating of a database schema fragment is presented. The difference between the proposed algorithm and other ones, which are generating database schemes by using a description logic ontology, oriented to logical queries to database only, but not to usual SQL queries from usual engineer information systems, is outlined. Database schemes, which are generated by the algorithm, are sufficient to store any facts about a subject area which are consistent with description logic ontology, grants references constraints and effective access to data by means SQL queries.

Key words: *ontology, description logic, database, relational model, data integrity constraints, database synthesis.*

Citation: *Kuchuganov MV*. Ontology based synthesis of database schemes. *Ontology of Designing*. 2016; 6(4): 475-484. DOI: 10.18287/2223-9537-2016-6-4-475-484.

References

[1] *Baader F.* The Description Logic Handbook. - New York: Cambridge University Press, 2003.
[2] *Konev B.* Ontology and knowledge representation. [In Russian]. Department of Computer Science, Liverpool University, 2010. - http://logic.pdmi.ras.ru/csclub/courses/ontology.
[3] *Sequeda J.* Integrating relational databases with the Semantic Web. - https://repositories.lib.utexas.edu/handle/2152/30537.
[4] *Auer S and I, Zachary G.* Integrating Ontologies and Relational Data. Technical Reports (CIS). 716 p. - http://repository.upenn.edu/cis_reports/716>.
[5] *Levkov A.* Effective fact storage organization for ontology. [In Russian]. - http://www.jitcs.ru/index.php?option=com_content&view=article&id=348.

Сведения об авторе

Кучуганов Михаил Валерьевич, 1966 г. рождения. Окончил Ижевский механический институт в 1983 г. Ст.преподаватель кафедры информатики и математики Удмуртского государственного университета. В списке научных трудов несколько работ в области конструктивных логик, синтеза программ и планов действий.
Kuchuganov Mihail Valerievich (b. 1966) graduated from the Izhevsk Mechanical Institute in 1983. Hi is senior lecturer at Udmurt State University (Department of informatics and mathematics). Hi is author of some scientific articles in the field of constructive logics, program synthesis and action planning.