ХИМИКОТЕХНОЛОГИЧЕН И МЕТАЛУРГИЧЕН УНИВЕРСИТЕТ

ФАКУЛТЕТ ПО ХИМИЧНО И СИСТЕМНО ИНЖЕНЕРСТВО

КАТЕДРА: "АВТОМАТИЗАЦИЯ НА ПРОИЗВОДСТВОТО"

Д И П Л О М Н А Р А Б О Т А

Тема: **„Трансформации на релационни бази данни в онтологии"**

Дипломант: Емилия Николова АтанасоваФак. № МС0815

София, 2016 г.

Специалност: *“Автоматика и Информационни технологии”*

Образователно-квалификационна степен: *МАГИСТЪР*

Дипломант: Емилия Николова Атанасова Фак. № МС0815

/...../

Ръководител на дипломната работа: .....

/гл.ас. д-р Даниела Гочева/

За рецензент определям: **.....**

Ръководител на катедра: .....

/доц. д-р В. Цочев /

UNIVERSITY OF CHEMICAL TECHNOLOGY AND METALLURGY - SOFIA
FACULTY OF CHEMICAL AND SYSTEM ENGINEERING
DEPARTMENT OF INDUSTRIAL AUTOMATION

DIPLOMA THESIS

Title:* *"Mapping relational data to OWL ontologies"

Specialty: **Automation and Information Technology**

Student: **Emiliq Nikolova Atanasova, Fac. № MS 0815**

/...../

Supervisor:**.....**
/ Assist. Prof. Daniela Gocheva, PhD/

Head of Department**…….**
/Assoc. Prof. V. Tzotchev, PhD/

Sofia, July 2016

ХИМИКОТЕХНОЛОГИЧЕН И МЕТАЛУРГИЧЕН УНИВЕРСИТЕТ
ФАКУЛТЕТ ПО ХИМИЧНО И СИСТЕМНО ИНЖЕНЕРСТВО
Катедра“АВТОМАТИЗАЦИЯ НА ПРОИЗВОДСТВОТО”

Специалност: *“Автоматика и информационни технологии“*

Образователно-квалификационна степен: *МАГИСТЪР*

Утвърдил:

Ръководител на катедра АП :.....

/доц.д-р.В.Цочев/

З А Д А Н И Е

ЗА ДИПЛОМНА РАБОТА

на студента: Емилия Николова Атанасова фак.№ МС 0815

Тема на дипломната работа:

„Трансформации на релационни бази данни в онтологии“

Описание на конкретните задачи:

1. Запознаване със системите бази данни и езика SQL
2. Запознаване със семантичните модели на данните и езика SPARQL
3. Запознаване с методи за трансформации на релационни в семантични модели
4. Трансформация на примерна база данни чрез D2RQ. Извличане на информация чрез SQL и SPARQL.

Студент:.....

Ръководител:.....

/гл.ас.д-р Даниела Гочева/

ДЕКЛАРАЦИЯ

от Емилия Николова Атанасова фак. № МС 0815

Декларирам, че съм разработила и написала представената дипломна работа самостоятелно, следвайки напътствията и съветите на дипломния си ръководител и използвайки само цитираните литературни източници и средства.

Дата: …….. Подпис: .....

СЪДЪРЖАНИЕ

УВОД

Основните данни на производствените предприятия включват данни за клиенти, продукти, служители, материали, доставчици и организационни структури. Данните обикновено са разпределени в различни системи и са достъпни в изключително разнообразни структури и формати: бази данни, таблици, документи и неструктурирана информация.

Семантичният уеб осигурява нови функционалности за интеграция на данни, за търсене и семантично посредничество, подпомага вземането на решения въз основа на интегрирани данни. Платформата на консорциума W3C за свързаните данни стандартизира начините за запис и достъп до данните. Платформата използва формат RDF, който е машинно обработваем и независим от приложенията, които могат да го използват. Извличането на данни се осъществява чрез SPARQL- протокол и език за заявки върху данни, записани или представени в RDF формат.

Необходимостта да се обменят данни мотивира търсенето на възможности релационните данни да се предоставят в мрежата с цел достъп и извличане на информация. Релационните бази данни са много широко разпространени поради тяхната ефективност и точни дефиниции, което позволява утвърдени софтуерни средства да манипулират и управляват данните предсказуемо и ефективно. Resource Description Framework (RDF) е формат за данни, базирани на уеб архитектура за идентификация и интерпретация на понятията. Дипломната работа изследва определя установяването на съответствие (mapping) на релационна база данни и семантичен RDF модел на данните.

ПЪРВА ГЛАВА

1. ВЪВЕДЕНИЕ В СИСТЕМИТЕ БАЗИ ДАННИ

1.1. Увод в системите бази данни

Понятието бази данни се появява в началото на шестдесетте години без да се знае кой за първи път го използва във връзка с компютърната обработка на данните [1]. За рождена дата на базата данни може да приеме юни на 1963г., когато фирмата SDC (Sistem Development Corporation) организира симпозиум на тема: "Разработка и използване на машинно управляеми бази от данни". Фундаментални са идеите на Ч. Бахман, които проличават в публикациите му от 1964 и 1965г. С тях той поставя първите проблеми на една нова информатика, позната днес като "Управление на бази от данни".

Понятието бази данни се отнася до съвкупност от данни, между които има вътрешни връзки и позволява да се получи достъп до информацията, изхождайки от различни гледни точки. Исторически базите данни са се развивали като начин за интеграция на системите за съхраняване на данни. Първоначално при създаване на програмните приложения е надделявала тенденцията всяко приложение да работи със собствен набор от данни. Такива програмни системи имат ограничена ефективност за използване на ресурсите на машините и информацията. Затова, скоро се преминава към създаване на интегрирани системи за съхраняване на данни и информация, при които една и съща информация се използва за различни цели. В такива системи се усъвършенства контролът за достоверност на информацията, тъй като основната база се съхранява на едно място и се ползва от много потребители, които следят за достоверността на данните. Съвременните програмни приложения, работещи с бази данни, използват програмни системи за манипулация на информацията на по-ниско ниво. Те се наричат програмни системи за управление на бази данни (СУБД). Такова разделение на функциите за използване на бази данни има някои преимущества. Първото преимущество е, че по този начин се опростява процесът на обработка на информацията. Това е особено явно при така наречените разпределени бази данни (данни разпределени на няколко компютъра).

Друго преимущество е възможността за управление на достъпа до базата данни. Системите за управление на бази данни имат собствени схеми за обозначаване на

йерархията в данните и осигуряват контрола за достъп до всяко ниво в тази йерархия. Освен това, чрез разделяне на функциите на отделните програмни средства се осигурява необходимата независимост на данните. Това означава, че може да се внасят изменения в организацията на данните, без да се променя програмното осигуряване за достъп до данните.

1.2. Приложение на базите данни

Информацията от базите от данни е много разнообразна: текстова информация, числова информация, статистика, таблици, графична, аудио, видео, и др. т.е. в базите от данни може да се съхранява и обработва всякакъв тип от информация, която се обработва от компютърна система. Това голямо разнообразие на типа информация в базите от данни произтича от разнообразните области на приложение: управление на фирми и организации, медицината, науката и образованието и други сфери на обществения живот. Научни работници, медици, прависти и финансисти ежедневно използват бази от данни за получаване на бърз селективен достъп до актуализирана информация. Чрез заявки за търсене всеки ръководител на фирма или счетоводител може да получи бърз достъп до последните изменения на конкретен нормативен документ. За да се издаде заявка на СУБД, не е нужно да се познава езика, с който работи системата. Дори и сложна завка може да се конструира само с щракване с мишката върху името на полето и логически оператори, с които се построява завката.

1.3. Характеристика на базата данни

- **Избягване дублиране на данните**

Един път съхранени, данните могат да се използват многократно в едно или няколко приложения

- **Поддържане съответствие на данните**

Една естествена последица от премахването на дублирането е, че се намалява възможността от несъответствие при актуализиране на данните.

- **Независимост на данните и програмите**

Промените в структурата на данните или в програмните приложения не са взаимозависими.

- **Логически аспект на данните за потребителите и потребителските програми**

Представянето на данни е независимо от начина на физическото им съхраняване. Възможни са различни логически представяния на едни и същи данни за различни потребители и програми. Това освобождава програмистите от нуждата да се съобразяват с физическите подробности на съхраняването им и позволява да се съсредоточат върху приложението.

- **Обслужване на широк кръг приложения**

Веднъж запомнени, данните могат да се обработват по-различни начини и в произволна желана форма, което разширява областта на възможните приложения.

- **Използване на правила и стандарти**

Фактът, че достъпът до данните става само чрез СУБД, позволява на администратора на базата данни да изисква и следи за изпълняването на определени стандарти при представянето на данните от приложенията.

- **Сигурността на данните**

Администраторът на базата данни контролира достъпа до нея. Той може да следи за задаването на кодове за достъп на потребителите, ограничаващи ги само до онези части и функции от базата от данни, които са им разрешени.

- **Елементи на базата данни**
    - Данни
    - Информация
    - Таблици
    - Полета
    - Изглед
    - Ключове
    - Първичен ключ
    - Външен ключ
    - Индекс
    - Релации

1.4. Предимства и недостатъци при подхода „база данни”

1.4.1. Предимства

- Осъществява се контрол на излишъка и устойчивостта на данните;
- Поддръжка на каталог, който съдържа описание на данните, както и описание на ограниченията, наложени върху тях;
- Осигурява независимост между данни и програми;
- Моделира данните;
- Поддържа целостта на БД така, че в нея да се съдържа само проверена и утвърдена информация;
- Може да се работи с различни потребителски изгледи (view). - потребителският изглед е подмножество на базата данни, с който работи даден потребител;
- Осигурена е разделимост на данните – всички данни са на едно място и се ползват от всички потребители;
- Недопускане на неупълномощен достъп до БД;
- Осигуряване на диференциран достъп само до тези елементи от БД от които отделните потребители се нуждаят в процеса на тяхната работа;
- Сигурност на данните при хардуерни и софтуерни сривове;
- Възможност за използване на разнородни интерфейси.

1.4.2. Недостатъци

- Допълнителни инвестиции за хардуер и софтуер;
- Разработката на бази от данни изисква време и разходи-изпълняват се системни функции като защита, паролен достъп, проверка на ограничения за цялост, които забавят обработката на данни;
- Моделът предоставя еднотипни справки за всички видове приложения.

1.5. Спомагателни средства на базите данни

Спомагателните средства на базата данни се използват за подпомагане на потребителите за решаване на проблемите, свързани с базата данни.

- **Езици за запитване** - Те са разработени за обикновените потребители, правещи запитвания към базата данни. Тези езици могат да се използват за специфични запитвания към базата данни, за разлика от командите за манипулиране на данните. Те са лесни за разбиране и употреба и обикновено са съставени от изречения, близки до говоримия английски език.

- **Речници за данни** - представляват набор от данни за самите данни в базата. Те съхраняват информацията за типа на записите, за имената и типовете на полетата, както и друга информация за структурата на базата данни. В съвременните системи речниците за данни често са част от самите бази данни.
- **Спомагателни средства за наблюдение и оценка** - позволяват на администратора на базата данни да определя степента на използването ѝ от отделни потребители, от отдели и други потребителски центрове. Тази информация е полезна за разпределяне на разходите между различните потребителски отдели.
- **Генератори на отчети** - понякога е необходимо да се изведат данни в специфичен формат. СУБД обикновено поддържат обикновени генератори за някои типове отчети. Възможно е да бъдат създавани и потребителски генератори на отчети. Генераторите са мощни и лесни за използване средсва за достъп до данните.
- **Архивиране и възстановяване** - в процеса на експлоатация на една база данни е възможно да настъпи повреда. За избягване на невъзвратимата загуба на информацията обикновено данните се копират или архивират в някаква сигурна среда. В интервала между две архивирания се поддържат записи за всички дейности засягащи базата данни (вмъквания, изтривания и модификации). Комбинацията от архивираната база данни и файла, съхраняващ записите за тези дейности позволява да се възстанови състоянието на базата данни, която тя е имала преди повреждането.
- **Управление на конкурентния достъп** - базата съдържа данни, които могат да бъдат едновременно достъпни за много потребители или потребителски приложения. Понякога може да има искания за конкурентен достъп. Подобен пример може да бъде едновременния достъп от двама потребители от данните за наличностите в една многопотребителска система. Няма проблем, ако исканията са за прочитане на запис, но възниква трудност, когато и двамата потребители се опитват да променят записа в едно и също време. Спомагателно средство, предвидено за конкурентното използване, „блокира“ единия потребител, докато влезе в сила промяната на другия потребител.
- **Физическа реорганизация** - Това средство подпомага администратора на базата данни при ефективното преструктуриране на физическата база данни, когато това е необходимо. Такова преструктуриране се налага, когато модификации като добавяния и изтривания на записи, променят физическите характеристики на

съхраняваните данни и причиняват увеличаване на времето за достъп до тях или при неефективно съхраняване на данните върху много носители.

1.6. Основни понятия на СБД

Системите Бази Данни (СБД) се състоят от две основни компоненти: База Данни (БД) и Система за Управление на Бази Данни (СУБД), които могат да бъдат дефинирани по следния начин:

- **База данни (БД) -** представлява колекция от логически свързани данни в конкретна предметна област, които са структурирани по определен начин. Базата от данни се състои от записи, подредени систематично, така че компютърна програма да може да извлича информация по зададени критерии.
- **Система за управление на бази от данни или система за управление на бази данни, СУБД (на английски: Database management system, DBMS)** е набор от компютърни програми, контролиращи изграждането, поддръжката и използването на бази от данни
- **Система БД (СБД) –** представлява съвкупността от БД и софтуера на СУБД.

1.7. Релационни бази данни

1.7.1. Същност на релационна база данни

Релационна база данни за първи път е предложен през 1970 година от **Едгар Код**, учен в **IBM**. Релационна база данни е тип база данни, която съхранява множество данни във вид на релации, съставени от записи и атрибути (полета) и възприемани от потребителите като таблици. Релационните бази данни понастоящем преобладават при избора на модел за съхранение на финансови, производствени, лични и други видове данни. Софтуерът, който се използва за организиране и управление на този вид бази данни се нарича най-общо система за управление на релационни бази данни (СУРБД).

2.1.2. Релация

Релацията (relation) се дефинира като множество от записи, които имат едни и същи атрибути. Записът обикновено представя обект и информация за обекта, който обичайно е физически обект или понятие. Релацията обикновено се оформя като таблица, организирана по редове и колони. Всички данни, които се съдържат за даден атрибут, принадлежат на едно и също множество от допустими стойности, наречено домейн и съблюдават едни и същи ограничения.

Характеристики на релацията:

- Всяка релация (таблица) в базата данни носи уникално име.
- Всеки атрибут носи уникално име в рамките на дадена релация.
- Всяка релация съдържа уникални записи, не може да съдържа повтарящи се идентични записи.
- Няма определен ред, в който се разполагат записите в дадена релация или атрибутите в даден запис.
- Стойностите в записите са атомарни – те не могат да се състоят от различни типове данни (данни от различни домейни). Физическата организация на данните в паметта няма значение за релационния модел, в който важи само логическата им организация.

Приложенията за бази данни осъществяват достъп до данните, като отправят заявки, които използват операции като `select` (селекция), за да се идентифицират записите, `project` (проекция), за да се идентифицират атрибутите, и `join` (съединение), за да се комбинират релации. Релациите се манипулират чрез операторите `create` (създаване), `insert` (вмъкване), `delete` (изтриване) и `update` (актуализиране).

2.1.3. Основни понятия

Теорията на релационните бази данни използва набор от математически термини, които имат съответствия с термините, използвани при SQL базите данни:

- релация, релационна схема (`relation`) ↔ таблица (`table`),
- запис, кортеж (`tuple`) ↔ ред (`row`),
- атрибут, поле (`attribute`) ↔ стълб, колона (`column`).

Като синоними се използват и понятията клас в смисъла на релация с точно определени атрибути и екземпляр на класа в смисъл на един от записите на тази релация.

- **Домейн** в базите данни означава множеството от допустимите стойности на даден атрибут на релация, т.е. представлява известно ограничение върху стойностите и името на атрибута. Математически погледнато, прилагането на домейн към даден атрибут означава, че атрибутът приема за стойности елементите от дадено множество.

- **Ограниченията** (constraints) позволяват в още по-голяма степен да се специфицират стойностите, които атрибутите от даден домейн могат да приемат. Например, за атрибут от домейна на целите числа, може да е валидно ограничение на стойностите между 1 и 10. Ограниченията предлагат един от методите за реализиране на бизнес правилата в базите данни. Ограниченията стесняват обхвата на данните, които могат да се съхранят в релациите. Те могат да се приложат над отделни атрибути, над записи или над цялата релация.
- **Ключ** (key) се наричат един или повече атрибута със специално предназначение в таблицата на релацията. Видът на ключа определя предназначението му. Съществуват четири различни вида ключове в базите данни:
  - кандидат ключ (възможен ключ);
  - първичен ключ;
  - външен ключ;
  - алтернативен ключ;

Най-важните сред тях са първичният и външният ключове. Всяка таблица трябва да съдържа поне един първичен ключ.

Първичен ключ (`primary key`) е атрибут (по-рядко група атрибути), който служи да идентифицира по уникален начин всеки запис (екземпляр) на релацията. Когато измежду атрибутите на релацията няма един подходящ за първичен ключ атрибут, вариантите са:

- да се прибегне към множество от два и повече атрибути, които заедно идентифицира т записите еднозначно, т.нар. сложен първичен ключ (`composite primary key`);
- да се добави нов атрибут, по който да се прави идентификацията на записите.

Външният ключ (`foreign key`) е необходим, когато е налице връзка между две таблици (релации). Релационната връзка се създава, като копие от първичния ключ на едната таблица се включи в структурата на втората таблица, за която той се явява външен (понеже тя вече си има свой собствен първичен ключ). Освен да помогне в установяването на връзка между две таблици, външният ключ помага да се осигури и целостта на ниво връзка.

2.1.4. Кардиналност на релационна връзка

Релационна връзка (relationship) се нарича зависимост, съществуваща между две таблици, когато записи от първата таблица могат да се свържат по някакъв начин със записи от втората таблица. Три са възможните видове **кардиналност** (cardinality):

- „едно към едно“ (1:1),
- „едно към много“ (1:N),
- „много към много“ (M:N).

Кардиналност „едно към едно“

Кардиналност от вид „едно към едно“ е налице, когато всеки запис от една таблица е свързан с най-много един запис от втора таблица и всеки запис от втората таблица е свързан най-много един запис от първата таблица. Този вид връзка е специална, защото е единствената, при която двете таблици могат да споделят един общ първичен ключ. Възможно е обаче и първичните им ключове да са различни и връзката да се създава с използване на външен ключ, като първичният ключ на едната таблица, без значение коя, се включва в структурата на другата.

Кардиналност „едно към много“

Кардиналност „едно към много“ между две таблици съществува тогава, когато един запис от първата таблица, наречена родителска, може да бъде свързан с много записи от втората таблица, наречена дъщерна, но запис от дъщерната таблица може да бъде свързан само с един запис от родителската таблица. Връзката между двете таблици се създава като копие на първичния ключ на родителската таблица се включи в структурата на дъщерната таблица, за която той представлява външен ключ. В литературата се среща и кардиналността „много към едно“ (N:1), която е вариант на „едно към много“. Това е най-често срещаният вид връзка между таблици.

Кардиналност „много към много“

Кардиналност „много към много“ съществува когато един запис от едната таблица може да се свърже с много на брой записи от втората таблица, и един запис от втората може да се свърже с много на брой записи от първата таблица. За да се създаде на практика тази връзка, се използва нова, свързваща или асоцииращa таблица, която съдържа копия на първичните ключове на двете таблици. От една страна свързващата таблица представлява сложен първичен ключ на отношението, а от другата страна, всеки

от първичните ключове на изходните таблици играе ролята на външен ключ за свързващата таблица.

2.1.5. Операции над релационни бази данни

- **Релационна алгебра**

Заявките, които се отправят към релационната база данни и извлечените в резултат подтаблици се изразяват със средствата на релационната алгебра (релационно смятане). В своята оригинална релационна алгебра, Код въвежда осем релационни оператора в две групи по четири. Първите четири оператора са били базирани на традиционните математически операции над множества:

- Оператор обединение (`union`) комбинира записи от две релации и премахва от резултата всички евентуални повтарящи се записи. Релационният оператор обединение е еквивалентен на `SQL`-оператора `UNION`.
- Оператор сечение (`intersection`) извежда множеството от записи, които са общи за двете релации. Сечението в `SQL` е реализирано чрез оператора `INTERSECT`.
- Оператор разлика (`difference`) се прилага над две релации и в резултат връща множеството от записите от първата релация, които не съществуват във втората релация. В `SQL` разликата е имплементирана посредством оператора EXCEPT или `MINUS`.
- Оператор декартово произведение или само произведение (`Cartesian product, cross join, cross product`) на две релации представлява съединение (`join`), при което всеки запис от първата релация се конкатенира с всеки запис от втората релация. В `SQL` операторът е реализиран под името `CROSS JOIN`.

Останалите оператори, предложени от Код, включват операции, които са специфични за релационните бази данни.

- Операцията селекция (`selection, restriction`) връща само онези записи от дадена релация, които отговарят на избрани критерии, т.е. подмножество в термините на теорията на множествата.
- Операцията проекция (`projection`) е по своя смисъл селекция, при която повтарящите се записи се отстраняват от резултата.

- Операцията съединение (`join`), дефинирана за релационни бази данни, често се нарича и естествено съединение (`natural join`). При този вид съединение две релации са свързани посредством общите им атрибути.
- Операцията деление (`division`) е малко по-сложна операция, при която записи от една релация в ролята на делител се използват, за да се раздели втора релация в ролята на делимо. По смисъла си, тази операция е обратна на операцията (декартово) произведение.

В добавка към въведените от Едгар Код осем оригинални оператора, в последствие са въведени и други, включително оператори за релационно сравнение и разширения, които реализират йерархични, темпорални, размити данни и т.н.

2.1.6. Теория на нормализацията

Нормализацията е процес на подреждане на данните в релации. Целта на логическият дизайн на базата данни е да представи обектите в базата данни чрез релации, които да съдържат необходимите данни, за да може представяният обект да бъде конструиран; освен това да могат да бъдат лесно добавяни, изтривани и модифицирани записи, без нарушаване на консистентността на записаните данни. Нормализацията е един от най-предизвикателните аспекти на традиционния дизайн на бази данни. Въпреки че е смятан за труден процес, той всъщност е относително прост, след като веднъж са разбрани три основни концепции:

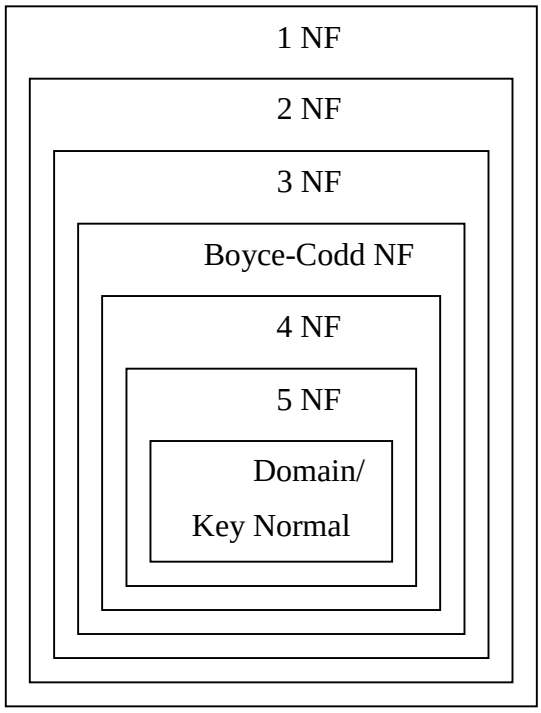
- Цялостните предпоставки за нормализация;
- Аномалиите на промените;
- Зависимостите между данните.

Процесът на нормализация е декомпозиция на големи, неефективно структурирани таблици, в по-малки, по-ефективно структурирани таблици, като при този процес не трябва да има каквито и да било загуби на данни. Нормализацията поддържа твърдението, че добре дефинираната база данни не съдържа дублиращи се информация и излишните данни са сведени до минимум. Това гарантира интегритета (съгласуваността) на данните и тяхната коректност при извличането им. Декомпозирането на некоректно структурираните таблици не е произволен процес. Това е методичен и последователен процес на привеждане на определена таблица в целевата нормална форма. Нормалната форма е алгоритъм, който се използва за тестване на структурата на таблицата, помагащ за премахване на аномалиите на промените и ефективно структуриране на таблиците и

полетата в тях. Съществуват седем нормални форми, всяка от които е създадена да се справи със специфични типове проблеми (фиг.1):

- Първа нормална форма – базирана на функционалните зависимости;
- Втора нормална форма - базирана на функционалните зависимости;
- Трета нормална форма - базирана на функционалните зависимости;
- Четвърта нормална форма – базирана на мулти-стойностни (`multi-valued`) зависимости;
- Пета нормална форма – базирана на `JOIN` зависимости;
- `Boyce/Codd` нормална форма – базирана на функционалните зависимости;
- `Domain/Key` нормална форма – базирана на дефиницията на домейни и ключове.

На фиг.1 са представени нормалните форми. Смисълът на това подреждане е, че ако искаме да достигнем 3-та нормална форма, то задължително трябва да преминем през първа и втора нормални форми преди това. По такъв начин всяка следваща нормална форма специфицира по-тясно изискванията, на които трябва да отговаря релация в тази нормална форма.



Фиг.1: Нормални форми

Теорията на нормализацията се изгражда около концепцията за нормални форми. Нормалната форма е една добре дефинирана стандартна мярка за степента на

нормализация, която една релация (таблица) притежава. Нормализацията се извършва на степени. Всяка следваща нормална форма означава по-висока степен на нормализация. Въпреки факта, че нормализацията е желателна, възможно е да изпаднем в свръхнормализация. Колкото по-висока е степента на нормализация, толкова повече са таблиците, което прави схемата по-сложна, а също така може да намали ефективността и производителността на базата данни. В оригинала на своите разработки Codd дефинира:

- първа;
- втора;
- трета нормална форма.

Мотивацията според дефинициите на Codd е, че 2NF е "по-желана" от 1NF и 3NF е по-желана от 2NF, т.е. проектантът ще цели предимно 3NF. Но това твърдение (че проектантът ще се стреми към 3NF) не трябва да се приема като закон. Понякога съществуват добри причини да се променят принципите на нормализацията. Единственото твърдо изискване е релациите да бъдат най-малко в първа нормална форма. В действителност това е добра причина за твърдения от рода, че проектирането на БД е много сложен проблем. Теорията за нормализацията е използваемо помощно средство в този процес, но тя не е панацея. Всяко проектиране на БД трябва да е точно обмислено да кореспондира с основните техники на нормализацията, но ние не искаме да внушаваме, че то се базира само на принципите на нормализацията. Има няколко неща, които трябва да бъдат проверени преди да започне процесът на нормализация:

- Всяка таблица трябва да има първичен ключ;
- Таблица не може да съдържа повтарящи се групи от данни.

А. Първа нормална форма (1NF)

Дефиниция: релационна променлива е в първа нормална форма, ако във всеки един момент всички атрибути на всеки ред съдържат единични стойности.

Целта на първа нормална форма е да гарантира, че полетата на таблицата не съдържат данни от много части или мулти-стойностни данни, а всяко поле съдържа единична стойност за всеки запис.

Б. Втора нормална форма (2NF)

Дефиниция: релационна променлива е във втора нормална форма, ако тя е в първа нормална форма и всеки не ключов атрибут е несъкратимо зависим от първичния ключ, т.е. всеки не ключов атрибут е зависим от целия първичен ключ.

Втора нормална форма гарантира, че всяко не ключово поле в таблицата е функционално зависимо от целия първичен ключ и че таблицата не съдържа полета с изчислени (калкулирани) стойности. Ако първичният ключ на таблицата се състои от едно поле, то тя е автоматично във втора нормална форма.

В. Трета нормална форма(3NF)

Дефиниция: релационна променлива е в трета нормална форма тогава, когато тя е във втора нормална форма и всеки не ключов атрибут е не-транзитивно зависим от първичния ключ, т.е. няма транзитивни зависимости между атрибутите.

Трета нормална форма гарантира, че таблицата притежава следните характеристики:

- Всяко не-ключово поле в таблицата е функционално зависимо от целия първичен ключ;
- Всеки атрибут представлява специфична характеристика на обекта, който таблицата представя;
- Таблицата представя само един обект, т.е. няма транзитивни зависимости.

Ако приемем, че всяка таблица с ключ от една колона е във втора нормална форма, то тогава на пръв поглед лесно ще решим проблема за достигане на тази нормална форма чрез добавяне на подобен ключ за всяка таблица. Но тогава ще се изправим пред основния проблем, който трета нормална форма е предназначена да разреши – функционалните зависимости между не ключови атрибути.

Г. Boyce/Codd нормална форма (BCNF)

Дефиниция: релация е в BCNF, ако и само всеки детерминант е ключ-кандидат.

BCNF е друга версия на трета нормална форма и е създадена, за да я усъвършенства и допълни. Целта на BCNF е от две части:

- Да гарантира, че поле, което определя стойността на някое или всички не-ключови полета от таблицата трябва да бъде кандидат-ключ за тази таблица;
- Да гарантира, че таблицата представя точно един обект.

BCNF е малко по-строга от трета нормална форма по отношение на ситуацията, когато една таблица има повече от едно поле, което може да бъде първичен ключ. Дефинирана е за точно определяне на всички кандидат-ключове, като по този начин

може да бъде гарантирана липсата на транзитивни зависимости и като следствие от това таблицата да не е субект на аномалиите на промените.

Д. Четвърта нормална форма(4NF)

Дефиниция: релационната променлива **R** е в четвърта нормална форма тогава и само тогава, когато съществуват подмножества **A** и **B** от атрибути на **R** такива, че МСЗ **A→→B** да е удовлетворена, тогава всички атрибути на **R** са функционално зависими от **A**.

Целта на четвърта нормална форма е да гарантира, че таблицата не съдържа МСЗ и представя един единствен обект.

Е. Пета нормална форма(5NF)

Дефиниция: релационна променлива **R** е в пета нормална форма, също наричана `Projection/Join Normal Form (PJ/NF)`, ако и само всяка нетривиална `JOIN` зависимост, която е в сила за **R**, се базира на кандидат-ключовете на **R**.

След като таблицата е достигнала четвърта нормална форма приемаме, че тя няма транзитивни и мулти-стойности зависимости. В повечето случаи няма да е необходимо таблицата да бъде декомпозирана по-нататък. Обаче, ако трябва да се декомпозира таблицата още веднъж, трябва да се провери дали е налице валидна `JOIN` зависимост в таблицата. Има няколко въпроса, на които трябва да се отговори преди да се премине към по-нататъшна декомпозиция:

- Може ли да бъде създадена таблица (или таблици), използвайки първичния или кандидат-ключ като част от новата таблица? (Това беше изискване за решаване на транзитивните и мулти-стойностните зависимости.)
- Може ли да бъде пресъздадена оригиналната таблица чрез `SQL JOIN` операция, която съединява всички таблици, създадени след декомпозицията?
- Ще бъде ли декомпозицията без загуба на записи?

Ако отговорът на всички тези въпроси е “да”, то таблицата е в пета нормална форма и декомпозицията може да бъде направена уверено. Обаче фактът, че таблицата може да бъде декомпозирана без загуба на данни, не означава, че това трябва да бъде направено.

Ж. Domain/Key нормална форма (DKNF)

Дефиниция: релационна променлива **R** е в DKNF, ако и само ако всяко ограничение на **R** е логическо последствие от ограниченията за домейни и ключове, приложени в **R**.

DKNF е сравнително нова нормална форма и е подобна на BCNF по това, че частично се базира на правилата, наложени от първични и кандидат-ключове. Но тя също се базира на идеята за домейните. Домейните не са просто множество от стойности – те имат логическа и физическа част. Логическата част дефинира стойност по подразбиране, диапазон от стойности, дали наличието на стойност да бъде задължително или може да има NULL стойност. Физическата част определя неща като тип на данните, дължина, позиция на десетичната запетая и позволени символи. За да бъде една таблица в DKNF, тя трябва да отговаря на следните изисквания:

- Всяко поле трябва да бъде цялостно и коректно дефинирано;
- Всяко поле трябва да представя характеристика на обекта, който таблицата представя;
- Всяко не ключово поле от таблицата трябва да бъде функционално зависимо от целия първичен ключ;
- Всяка таблица трябва да представя само един обект.

Таблица, намираща се в DKNF няма да има транзитивни и мулти-стойностни зависимости, няма да бъде субект на аномалии на промените.

2.1.7. Денормализация

Едни от най-често срещаните причини за денормализация, са недостатъчна бързина и ефективност на базата данни. Тази стъпка трябва да е винаги последна, когато всички други варианти са неприложими. Преди нея могат да бъдат предприети редица действия за повишаване на ефективността на работа на базата данни. Някои от тях са:

- Обновяване на компютърното оборудване;
- Оптимизиране на операционната среда, увеличаване на скоростта на мрежовия трафик;
- Оптимизиране на софтуера на СУБД – осигуряване на това, че няма стартирани допълнителни процеси от софтуера на СУБД, които не са необходими, но заемат процесорно време и памет;
- Ефективно използване на индекси – индексите могат реализират огромно ускорение процеса на извличане на данни;

- Писане на добър, стегнат и ефективен процедурен код;
- Писане на добре структурирани и ефективни SQL заявки.

Когато се извършва денормализация се прави стъпка назад към проблемите с целостта на данните. Затова трябва добре да се прецени и реши дали ползите от денормализацията ще си струват допълнителните усилия, които ще коства поддържането на такава база данни.

2.1.8. Предимства и недостатъци на релационните модели на БД

А. Предимства:

- Осигуряват ясно и концептуално пряко представяне на сложни релации на данни;
- Позволяват мощни манипулации с данните чрез езици за запитване;
- Базата от данни се поддържа в таблична форма, която е естествена за представяне на данни за потребители с бизнес-ориентация;
- Могат да бъдат разработени езици за запитване, използващи тази таблична форма;
- Достъпът до данните е по стойности и условия и базата от данни е достъпна за от всяка точка;
- Пътищата на достъп не се виждат от потребителите на базата от данни;
- Представянето на данните е изцяло логическо, което още веднъж подсилва простотата на представянето от гледна точка на потребителите и приложните програмисти.

Б. Недостатъци:

- Използваните при конкретното изпълнение индекси често са големи и изискват допълнителна памет с голям обем;
- Скоростта на действие на съществуващите комерсиални системи за релационни бази от данни е по-ниска, отколкото на техните йерархични или мрежови аналози. Това ги прави неподходящи за голями по обем оперативни дейности.

ВТОРА ГЛАВА

2. СЕМАНТИЧНИ СТАНДАРТИ, ЕЗИЦИ И ТЕХНОЛОГИИ

2.1. Семантичен стек

Инициативата за създаването на Семантичен уеб (семантична мрежа) е на създателя на World Wide Web – Тим Бърнърс-Лий и основания от него консорциум World Wide Web Consortium (W3C) при участието на голям брой изследователи и индустриални партньори. Според публикации на сайта на консорциума W3C http://www.w3.org/standards в настоящия момент технологии, свързани със Семантичният уеб се използват в голям брой разнообразни приложения: за семантично търсене, за интеграция на данни и едновременно извличане на данни от различни източници в различни формати, за откриване и класификация на ресурси, за създаване на по-добри, специфични за някои области информационни системи (здравеопазване, науките за земята, администрация и електронно управление, енергетика, добив на нефт и газ и др.), за каталогизиране на съдържанието и връзките на уеб сайтове и дигитални библиотеки, за улесняване на споделянето и обмена на знания чрез интелигентни агенти на софтуерни приложения; за оценяване на уеб съдържание и др.

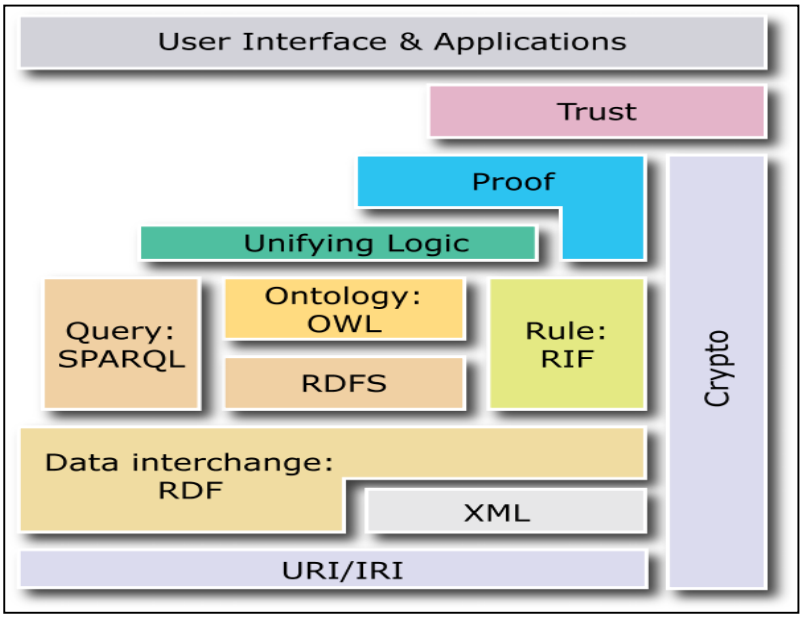
Семантични технологии позволяват да се създават складове от данни в интернет, да се изграждат формализирани речникови структури за конкретни области, да се пишат правила за обработка на данни.

Въпреки, че идеите на Семантичния уеб първоначално са разглеждани единствено като разширение на инфраструктурата на World Wide Web, софтуерът и приложенията, базирани на семантични технологии незадължително са предназначени за работа в глобалната мрежа и незадължително интегрират уеб съдържание. Визията на семантичната мрежа за обмен на данни, доколкото е възможно чрез използване на наличните инструменти и работни практики не е приложима само в една отворена среда като интернет, а също така и в затворени системи, като например в индустрията и бизнеса.

Семантичният уеб осигурява нови функционалности чрез технологиите през мрежата, нови възможности за интеграция на данни, търсене и семантично

посредничество, подпомага вземането на решения и др. Семантичните уеб стандарти се развиват активно в момента и приложенията на Семантичния уеб навлизат във всички области и сфери на реалността.

Семантичният стек (фиг.2) е илюстрация на йерархията на различни езици, където всеки слой използва функциите на всеки един слой под него. Показва как технологиите които са стандартизирани за Семантичния уеб са организирани и взаимно свързани; а също така как Семантичният Уеб е продължение на класическия хипертекстов уеб.



Семантичен Стек – Архитектура и стандарти [2]

Технологиите които се намират най-отдолу в стека нагоре са стандартизирани и са приети да изграждат семантичните приложения. Все още не е ясно как технологиите от върха на стека ще бъдат изпълнени.

(1) Унифицираните идентификатори на ресурси URIs (Uniform Resource Identifiers) позволяват всеки ресурс да се идентифицира глобално по еднозначен начин чрез определен символен низ.

(2) Extensible Markup Language (XML) - език, който позволява описание на структурирани Web документи чрез дефинирана от потребителя лексика. XML осигурява синтаксиса за дефиниране на структурата на документите, без да има отношение към смисъла на съдържанието.

(3) Resource Description Framework (RDF) - Resource Description Framework е разработен като модел на данни и метаданни на базата на синтаксиса на XML. Формално RDF е модел на данни за ресурси, техните свойства и стойностите на тези свойства чрез

т. нар. „модел на тройките“ (subject, predicate, object). В действителност RDF е универсален машинно-читаем формат. RDF дава общата структура на информацията, така, че тя да може да бъде достъпна от приложения, различни от тези, за които тя оригинално е създадена и по този начин всяка информация в RDF може да бъде свързана с всяка друга информация в RDF. Ресурсите се идентифицират еднозначно и глобално чрез Uniform Resource Identifiers (URIs). RDF-изразите се представят чрез етикетирани ориентирани графи.

(4) RDF Schema (RDFS) - RDF Schema предоставя структурата за моделиране на данни, записани като RDF. RDF Schema дава възможност да данните да се структурират в йерархии от класове и свойства, позволява да се зададат ограничения върху свойствата. RDF Schema се основава на RDF и в същността си е език за дефиниране на лексика и може да се разглежда като формален език за онтологии.

(5) Web Ontology Language (OWL) - По-мощните в сравнение с RDFS езици за онтологии (Web Ontology Language – OWL и OWL2) разширяват RDF Schema и позволяват описание на по-сложни характеристики и взаимовръзки в модела на данните.

(6) Rule Interchange Format (RIF) - Rule Interchange Format (RIF) е език за комбиниране на онтологии и правила.

(7) SPARQL е протокол и език за запитвания върху данни, записани или представени в RDF формат. С модификациите и приемането през 2013 год. на SPARQL 1.1., езикът може да бъде използван като език за създаване на правила чрез възможностите си за конструиране на графи.

2.2. RDF – модел на данните

RDF е основен модел на данните за записване на прости изрази за Web обекти (ресурси). RDF моделът на данните не разчита на XML, но използва XML-базиран синтаксис. За разлика от йерерхичия модел на XML, технологията RDF представлява базирано на графи представяне на знания. RDF е модел на данни.

- RDF е признат за W3C standard recommendation през 2004;
- RDF е език за представяне на информация за ресурсите в Web;
- RDF е универсален машинно-читаем формат;
- RDF дава общата структура на информацията;

- Информацията може да бъде достъпна от приложения, различни от тези, за които тя оригинално е създадена;
- Всяка информация в RDF може да бъде свързана с всяка друга информация в RDF;
- Ресурсите се идентифицират еднозначно и глобално чрез Uniform Resource Identifiers (URIs).

RDF-изразите се представят чрез етикетирани ориентирани мултиграфи. Възлите в графа представляват ресурсите, дъгите са релациите, които също както ресурсите притежават имена и се идентифицират чрез унифицираните идентификатори на ресурси URIs. RDF данните се записват чрез машинно-читаем формат (RDF/XML), или чрез други формати (Turtle, N-Triples и др.), които се поддържат от голям брой програмни средства

2.3. SPARQL –език за запитвания към RDF данни -

Езикът SPARQL има широко приложение върху RDF модели – за запитвания, за модифициране на данните, за създаване на правила, за трансформации на данни и др.

Семантичният език SPARQL е подобен на SQL език, който реализира търсене на зададен модел (шаблон) в RDF модел на данни чрез HTTP. Чрез езика SPARQL е възможно извличане на данни, конструиране на RDF графи, проверка на твърдения.

SPARQL (Simple Protocol and RDF Query Language) работи с всякакъв тип източници на информация които могат да бъдат представени в RDF.

SPARQL позволява да се събира информация от RDF данни по следният начин:

- Да се извличат стойности от структурирани и полу-структурирани източници на данни
- Да се изследват данни чрез запитвания и да се откриват неизвестни релации
- Да се обединяват разнородни източници на данни
- Да се трансформират RDF данни от една домейн лексика (vocabulary) в друга

Една SPARQL заявка (фиг.3) се състои от:

- Деклариране на префикси за съкращаване на URIs
- Дефиниране на източник на данни, указвайки графи които ще бъдат запитвани
- Резултат, каква информация трябва да се получи от заявката
- Шаблон на заявката, показващ какви изрази се търсят

- Модификатори за изграждане, подреждане, пренареждане на резултатите от заявката

```
# prefix declarations
PREFIX                                  foo:
<http://example.com/resources/>
...
# dataset definition
FROM ...
# result clause
SELECT ...
# query pattern
WHERE {
    ...
}
# query modifiers
```

Фиг. 3 Структура на една SPARQL заявка [2]

Основни клаузи в SPARQL са:

- **SELECT** – търсене на данни по зададен шаблон
- **ASK** – запитване дали заявката се изпълнява (резултатът е true/false)
- **DESCRIBE** – връща всички RDF изрази за зададен ресурс
- **CONSTRUCT** – създава нови RDF изрази на базата на резултатите от заявката

Наборът от SPARQL спецификации, приети като стандарти (WC3 recommendations) през 2013 година включва:

- **SPARQL Query language SPARQL 1.1.** Спецификацията дефинира синтаксиса и семантиката на езика SPARQL за заявки към RDF източници на данни, независимо дали данните се съхраняват в RDF или се разглеждат като RDF. SPARQL поддържа агрегиране, подзаявки, отрицание, изчисляване на стойности на изрази, тестване на стойности и др. Резултатите от SPARQL заявки могат да бъдат набори от данни или RDF графи.
- **SPARQL Update**. Езикът представлява език за манипулиране (update) на RDF графи на базата на синтаксиса на SPARQL Query Language и операциите за манипулиране (създаване, актуализиране и отстраняване) на колекции от графи в Graph Store.

- **SPARQL Protocol** дефинира средства за предаване на SPARQL заявки към услуги за обработване на заявките и получавана на резултати чрез HTTP.
    - **SPARQL Protocol client** е HTTP client, който изпраща HTTP заявки за операциите на SPARQL Protocol (client)
    - **SPARQL Protocol service** Един HTTP сървър който обслужва HTTP заявки и изпраща чрез HTTP резултатите. URI за връзка на SPARQL Protocol service е известен като SPARQL endpoint. (service)
    - **SPARQL endpoint** е URI на който SPARQL Protocol service получава заявки от SPARQL Protocol clients.
    - **SPARQL Protocol operation**. Една HTTP заявка и резултатът от нея съответстващи на SPARQL Protocol.
    - **RDF Dataset** Набор от един default graph и нула или повече named graphs, както е дефинирано според SPARQL 1.1 Query Language.
- **SPARQL Result formats** са форматите за записване на резултатите от SPARQL заявките (SELECT и ASK)
    - SPARQL Query Results JSON Format
    - SPARQL Query Results CSV (comma separated values) и TSV (tab separated values) формати за таблични данни.
    - SPARQL Query Results XML Format
- **SPARQL точки за достъп** (SPARQL endpoint)

SPARQL точки за достъп е URI до който могат да бъдат изпращани заявките, и който връща отговор на заявката чрез HTTP. SPARQL точка за достъп може да бъде:

- Genetic – записва произволни Web данни в RDF
- Specific – предназначени за частни източници на данни.

Като резултатите от тези заявки могат да се представят в различни формати:

- **XML**. SPARQL дефинира XML лексика за резултатите.
- **JSON**. Има практическо значение за Web приложения.
- **RDF**. Тези формати са (RDF/XML, N-Triples, Turtle)
- **HTML**. При интерактивните форми за SPARQL заявки. Често се имплементира чрез XSL трансформация върху XML формат на резултатите

ТРЕТА ГЛАВА

3. СЕМАНТИЧНИ И РЕЛАЦИОННИ МОДЕЛИ НА ДАННИТЕ – СРАВНЕНИЕ И АНАЛИЗ

3.1. Семантични и релационни модели на данните

Семантични модели наричаме моделите на данни, които се създават, интегрират и запитват чрез технологиитте на семантичния уеб, систематзирани чрез семантичния стек (фиг.1).

В [3] и [4] са анализирани отношенията на семантичните модели във връзка с релационните модели на системите бази данни, систематизирани са общите черти и различията, посочени са ползите от онтологично базираното управление на данни: явна семантика, ниска степен на свързаност, възможности за интеграция и оперативна съвместимост, за автоматични проверки за консистентност, лесна разширимост и поддръжка.

Едно от предимствата за използването на RDF/OWL като модел на данните е продължителността на жизнения цикъл на системата – семантичните технологии и онтологиите в частност са безспорно по-гъвкави при промяна на изискванията към системата в сравнение с подходите на базата на фиксирани схеми като релационните бази данни. Форматът на данните е лесен за поддържане, данните от различни източници могат да бъдат свързвани и многократно използваеми в различни приложения.

Моделирането на данни е процес на създаване на модел на данните за информационна система чрез прилагане на формални техники за моделиране [5]. Моделирането на данни се използва за дефиниране и анализиране на данните и изискванията към данните, необходими за подпомагане на бизнес процесите в рамките и обхвата на съответната информационна система. В процеса на моделиране на данни се включват както тесни специалисти, така и потенциалните потребители на информационната система.

- **Концептуални модели** на данни. Тези модели, понякога се наричат домейн модели и обикновено се използват за проучване на особеностите на областта със

заинтересованите страни по проекта. Представляват един набор от технологично независими спецификации.

- **Логическите модели** на данни се използват за анализиране и дефиниране на понятията в областта на моделиране, както и техните връзки. Логическият модел на данните документира структурите на данните, описва видовете логически същности, атрибути, описващи тези същности, и релациите между тях.
- **Физични модели** на данни се използват за проектиране на вътрешната схема на базата данни, изобразяваща таблиците с данни, колоните от данни на тези таблици и връзките между таблиците

Трите модела са относително независими един от друг, технологията за съхранение може да се промени, без да засяга или логическия или на концептуалния модел. Структурите на таблици и атрибути могат да бъдат променяни без това (задължително) да повлияе на концептуалния модел.

Когато информационното моделиране се извършва с цел за да се създаде релационна база данни, концептуалният модел трябва да бъде различен от логическия модел, защото за някои компоненти няма място в релационна структура на базата данни, напр. за дефинираните правила. Тази семантична информация, събрана и документирана като част от първоначалното моделиране не е забравена, когато се преминава към създаване на логическия модел на данните. "Забравената" част се използва от разработчиците на софтуер, тъй като те се кодират бизнес семантика директно в потребителските програми [6].

Логическият модел на данните е подмножество на концептуалния модел, който може да се изрази с помощта на конкретна технология за моделиране на данни. Въпреки това, винаги има някои съображения за ефективност, които изискват допълнителни промени в логическия модел на данните, преди да може той да се реализира в релационна база данни. Следователно, някои от аспектите на логически модел се изоставят, когато се преминава към физически модел на данните.

Тъй като уеб онтологиите са модел на дадена приложна област, описващ обектите в областта, може да се смята, че всички три вида модели на данни могат да бъдат сравнени с онтологии. Те варират от най-изразителния, който описва бизнес концепции и процеси (концептуалния модел) към по-малко изразителния като постепенно се преминава от описание на бизнес семантиката към описание на физическата структура на данните, тъй

като тя се съхранява в базите данни (логически и физически модел на данните). Физическият модел може да се счита като онтология на определена база данни [6].

Семантичните уеб технологии дават възможност да се имплементира директно концептуалния модели [6]. Това е възможно благодарение на пластовата структура на семантичните технологии, представени във вид на стек (фиг.1), състояща се от: RDF - каноничен модел на данните, който подобно на релационния модел на данните може да свърже обекти, но за разлика от релационния модел на данните, ресурсите в RDF са силно „раздробени“. Най-малката единица на информация в RDF не е таблица, или един ред в таблица, а индивидуалните изрази (тройки) - един-единствен факт за един ресурс.

Тъй като RDF моделът е силно каноничен, при RDF модела схемата е сведена до прости тройки. Не съществуват ограничения, за да се комбинират данните в таблици или йерархии. RDF данните са просто мрежа от свързани тройки. Като такава, тя може да се използва за представяне, ако е необходимо и на таблични структури и на йерархии. Стандартни преобразувания лесно се дефинират за трансформации от релационни таблици и XML йерархии в RDF [6].

Друг ключов фактор за RDF модела на данни е, че той е "роден в мрежата". Всеки RDF ресурс има глобално уникална идентичност – унифициран идентификатор (URI). В резултат на това е възможно да се свържат RDF данни по начин, подобен на начина, чрез който документите се свързват чрез хипервръзки в мрежата и това важи за всички HTTP базирани мрежи, включително интранет и екстранет [6].

Точно като има стандартни езици за заявки към релационни и XML данни, има стандарт за заявки към RDF данни - SPARQL. Също като създадения в уеб RDF модел, SPARQL е не само език за заявки, но също така и протокол, който дава възможност за достъп до данни RDF чрез HTTP.

RDFS (RDF Schema) и OWL (Web Ontology Language) са RDF-базирани езици за изразяване на бизнес семантика. Съвместно RDFS и OWL дават възможност да се дефинират класове, йерархии от класове и свойства, характеристики на класовете и свойствата и ограничения върху тях. Възможностите на RDFS и OWL позволяват напълно да изрази смисъла на бизнес понятията и техните връзки. Моделите на данните в RDFS и OWL се съхраняват по същия начин, както и данните, т.е. във вид на RDF тройки.

Тъй като данните и схемата се съхраняват по един и същи начин, възможно е да се запитват схемите по същия начин както и данните и да се съчетаят критерии за търсене на схеми с критерии за търсене за данни.

Използването на RDF означава, че конструкциите на моделиране и дефинициите могат да бъдат свързани. Организациите могат да използват взаимно едни и същи бизнес дефиниции. Моделите могат да бъдат модулни, могат да бъдат описани и повторно използвани, когато е уместно. Различията между свързани, но не идентични понятия може да бъдат дефинирани. Всичко това може да се направи в стандартен и оперативно съвместим начин.

В [3] е направен паралел между онтологичните модели и утвърдените логически модели на базите данни. Онтологичните аксиоми (аналогични на схемите на базите данни) описват структурата на модела, докато фактите (данните в базите данни) описват конкретна ситуация. Основните разлики според [3] са представени в таблица 1.

Таблица 1. Сравнение между модели на данни

Логически модел на данни	Онтологичен модел на данни
Допускане Closed World Assumption (CWA) (липсващата информация се третира като грешна)	Допускане Open World Assumption (OWA) (липсващата информация се третира като неизвестна)
Допускане Unique Name Assumption (всеки индивид има единствено уникално име)	Без допускане Unique Name Assumption (индивидите могат да имат повече от едно име)
Схемата функционира като ограничение върху структурата на данните (дефинира състоянието на данните)	Аксиомите функционират като правила (за извеждане на логически заключения)

При онтологичното търсене аксиомите играят основна роля, като позволяват извличането на явна и неявна информация, но алгоритмите за търсене се характеризират с високо ниво на сложност в зависимост от размера на входните данни. Според [3], онтологиите като модел на данните би следвало да се използват когато схемата е голяма и сложна и е необходимо да се използва по време на запитването, когато не е възможно да се предположи вида на цялата информация (сложни структури) и когато не е възможно да се реализира изпълнението с „конвенционални“ средства. Системите бази данни се използват когато схемата е малка, ясна и не се използва по време на заявката,

когато цялата информация е достъпна от самото начало (момента на проектирането) и когато са необходими гаранции за осъществяване на точност и бързина.

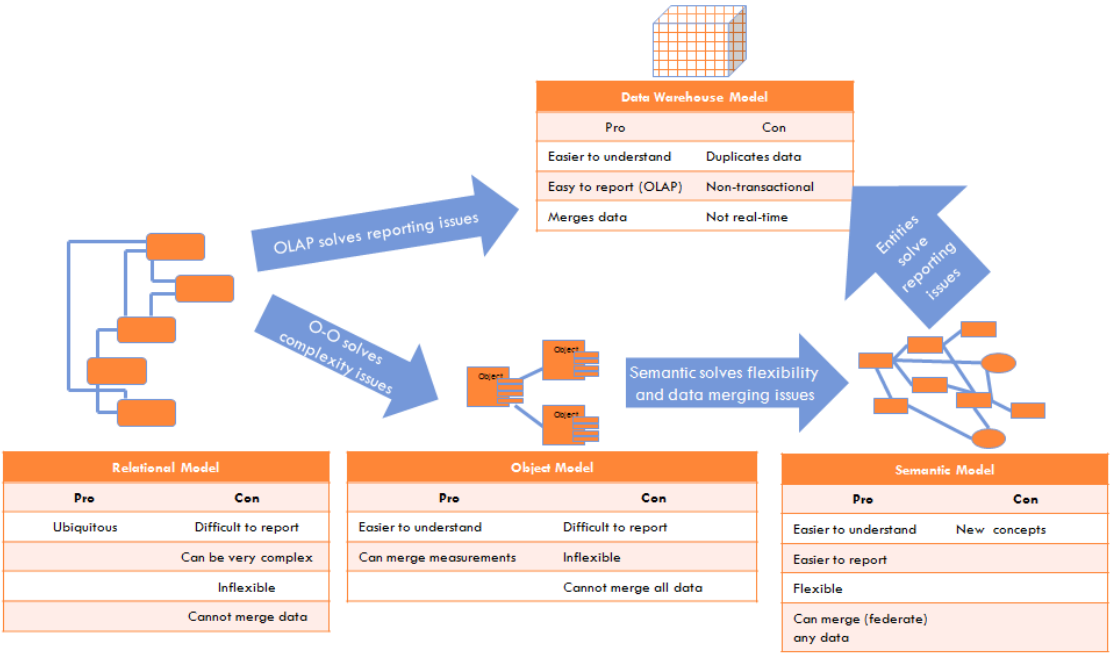
В [7] са посочени пет причини за използване на семантични модели на данните.

- Семантичните технологии позволяват да се използват едновременно различни модели и средства за съхранение и управление на данните.
- Осигуряват еднозначност на термините и хармонизиране на данни от различни източници.
- Семантичните модели на данните се създават постепенно (не еднократно) в момента, когато данните са налични и достъпни.
- Семантичните модели са “леки” и гъвкави.
- Семантичните технологии оставят данните да се съхраняват в техните оригинални хранилища, а обединяването се осъществява при поискване.

Трите особености на семантичните модели на данните, които релационните, йерархичните и складовете за данни не притежават според [7] са:

- Традиционните релационни метамодели имат голяма степен на гъвкавост до момента, в който те са пуснати „в работа”. След като схемите на данните, отчетите и заявките са изградени, става трудно да се променят таблиците и релациите. Тези ранни решения относно структурата на таблиците, ключовете и типовете данни влияят на всяко бъдещо решение за изграждане на приложения. Семантичните модели позволяват да се дефинира структура, когато това е необходимо, а също и да се направи анализ и логическа проверка на структурата на модела на данните по всяко време. Семантичните модели са гъвкави, поддържат сложността на структурите на данните и подпомагат осъществяването на промени в тях
- С помощта на семантичните технологии може да се създаде виртуален склад за данни, оставяйки на данните, там където са и в какъвто формат са. Семантичният модел използва възможностите на обединяване на данни, без да е необходимо данните да бъдат копирани. Федерацията на данните е "временна" - тя се създава за продължителността на заявката
- Семантичните технологии позволяват да се създаде стандартно базиран модел на данните, използвайки данните от алтернативни източници на данни.

Някои характеристики на различните модели на данните са показани на фиг.4.



Фиг.4 Сравнение на модели на данните [7]

Според проект CUBIST (Комбиниране и съчетаване на семантични технологии и бизнес интелигентност) [8], интелигентните системи за бизнес анализи се сблъскват с проблеми породени от комплексността на програмните средсва от една страна и от невключването на неструктурирани източници на данни за осъществяване на анализите от друга страна. Голям брой анализирани в [8] семантични методи, технологии и софтуерни продукти поддържат федерации от данни от неструктурирани и структурирани източници, съхраняване на данните в семантични хранилища, прилагане на визуални анализи, навигация и визуални заявки към данните. В CUBIST като примери за разнообразни източници на данни, които успешно се трансформират чрез налични софтуерни средства (включително и open source) са посочени: HTML страници, структурирани и неструктурирани текстови документи (word, txt, pdf), таблици (CSV, TSV), XML, бази данни, онтологии.

Технологията на Свързаните данни (Linked Data) позволява разработване, свързване и управление на различни структури от данни: бази данни, таблици, документи и друга информация, намираща се в най-разнообразни източници. Данните се представят във формат, който е машинно обработваем и независим от приложенията, които го използват.

Поради факта, че в момента огромно количество структурирана информация е във вид на релационни бази данни, използването на създадените, утвърдени и работещи релационни бази данни чрез технологиите на семантичния уеб (RDB2RDF) е актуална

задача. През септември 2012 като W3C standard recommendation се приема **Direct Mapping of Relational Data to RDF** - стандартен пряк метод за трансформация на релационни данни в RDF структури, като структурата на RDF графите пряко отразява структурата на базите данни. Direct Mapping разчита на автоматични методи, които създават "предполагаема" онтология от схемата и трансформират на данните в съответствие с тази схема. Преките методи зависят от „богатството" на схемата на данните по отношение на семантиката на данните. През 2012 като W3C standard recommendation се приема и езика **R2RML** - език за преобразуване от релационни бази данни в RDF масиви от данни. Чрез езика R2RML се създава съответствие между схемата на данните и съществуваща онтология.

3.2. Езици за запитване и модифициране на релационни и семантични модели на данните

В [9] е направен сравнителен анализ на езиците SPARQL и SQL. Докато SPARQL първоначално може да изглежда като SQL, има съществени различия, тъй като данните в RDF се базират на графи, съответно SPARQL заявки търсят съответствие с модели на графи вместо релационни SQL операции. Синтаксисът на двата езика е много близък, но SPARQL запитва данни във вид на графи, а SQL - релационни данни в таблици. Всеки от двата езика за заявки има своите предимства и недостатъци. SQL и релационния модел са добре проектирани, за да представляват регулярни структурирани данни, и по тази причина се използват много широко за управление на данни на производствени и бизнес процеси. Целта на изследването в [9] е продиктувана от възможността да се транслират SPARQL изрази в SQL изрази, като по този начин се е възможно да съхраняват RDF модели в релационни бази данни, и обратно – за да се запитват чрез SPARQL релационни данни от различни таблици, представени чрез RDF модели.

SPARQL и SQL езиците за заявки [10] са предназначени за извличане на информация съответно от данни, представени чрез RDF модел или релационни данни. Основно въпросът какво повече може да се направи с SPARQL, в сравнение с SQL, в действителност засяга това, какво може да бъде направено на базата на RDF модели и не може да бъде направено с релационни бази данни. И двата езика за търсене позволяват на потребителя да създава, комбинира и използва структурирани данни. SQL осигурява достъп до таблиците в релационните бази данни, и SPARQL - достъп до мрежа от свързани данни. SPARQL е проектиран за да обедини различни разнородни източници на

данни (различни формати и модели на данните могат да се представят чрез RDF „в движение“ без данните да се трансформират „физически“).

Релационните данни се състоят от редове, събрани в таблици. Редовете в таблицата съответстват на зададените типове данни и ограничения и според [10] се наричат **схема на данните** (schema). Частта от SQL наречена DDL (data definition language) дефинира схемата.

```
CREATE TABLE Person (
    ID INT,
    fname CHAR(10),
    addr INT,
    FOREIGN KEY(addr) REFERENCES Address(ID));

CREATE TABLE Address (
    ID INT,
    city CHAR(10),
    state CHAR(2))
```

По този начин чрез SQL се дефинира че таблица Person има три колони-"ID", "fname" и "addr", които са съответно INT, CHAR(10) и INT; таблица Address с non-NULL в поле "addr" в таблица Person съответстващо на стойността на "ID" в таблица Address.

Записите в двете таблици биха изглеждали примерно така: Боб живее в Кеймбридж, Масачузетс и не се знае къде живее Сю:

ID	city	addr
7	Bob	18
8	Sue	NULL

ID	sity	addr
18	**Cambridge**	**MA**

Чрез SQL заявка е дефинирано ограничението **Person.addr = Address.ID**. Чрез RDF релациите може да се представят като:

```
entity1 has propertyA relationship to entity2
```

Чрез използване на езика Turtle, в RDF можем да дефинираме, че има човек с име „Bob”, с адрес в Cambridge, MA по следния начин:

```
<PersonA> a <Person> .
<AddressB> a <Address> .
<PersonA> <Person#fname> "Bob" .
```

```
<AddressB> <Address#city> "Cambridge" .
<PersonA> <Person#addr> <AddressB> .
<AddressB> <Address#state> "MA" .
        <PersonF> a <Person> .
<PersonF> <Person#fname> "Sue"
```

Термините, използвани в твърденията по-горе, са относителни URL адреси - в ъглови скоби (<> и), литерали - в кавички ("" и) и символ "a", който е съкратен запис за дефиниране на релацията "has type".

В RDF не съществува понятие съответстващо на SQL NULL, тъй като няма изискване в RDF, съответстващо на структурното ограничение на SQL, че всеки ред в релационната база данни трябва да отговаря на една и съща схема.

В RDF модела на данните обект (object) на едно твърдение, например <AddressB>, могат да бъдат субект (subject) или обект (object) на други твърдения. По този начин, наборът от RDF твърдения се свързва във вид на RDF граф. Тези графи може да бъде циклични, например да се посочи, че Боб живее някъде, където той е собственик:

```
        <PersonC> <Person#homeAddress> <AddressK> .
<PerconC> <Person#fname> "Bob" .
<AddressK> <Address#owner> <PersonC> .
```

Основна разлика в моделите на данните (RDF и релационния модел) е, че RDF е Web език, т.е. позволява да се използват уеб идентификатори за ресурсите, които се описват, за атрибутите и релациите между ресурсите. Основна характеристика и предимство е че RDF моделът много лесно може да обединява информация от различни източници. Пример за това, ако някой има <http://xmlns.com/foaf/0.1/givenName>, тази стойност ще бъде наистина собствено име на този човек. Това означава, че RDF данните се „смесват“ докато, за релационните данни е необходим междинен процес на установяване на съответствие на термините между бази данни и проверки, когато таблиците използват едно и също име за дадено поле, когато то би следвало да означава различни неща.

Една SQL заявка за извличане на адресите на всички, които живеят в MA може да изглежда така:

```
        SELECT Person.fname, Address.city
FROM Person, Address
WHERE Person.addr = Address.ID
AND Address.state = "MA"
```

Концептуално, избира се списък с атрибути от набор от таблици, когато са изпълнени определени ограничения. Тези ограничения улавят отношенията, които се съдържат в схемата, Person.addr = Addresses.ID и критериите за избор, например Address.state = "MA".

А SPARQL заявка за търсене на същите тези данни може да изглежда така

```
SELECT ?name ?city
WHERE {
    ?who <Person#fname> ?name ;
    <Person#addr> ?adr .
    ?adr <Address#city> ?city ;
    <Address#state> "MA"}
```

SPARQL използва някои ключови думи, познати на SQL потребители: SELECT, FROM, WHERE, UNION, GROUP BY, HAVING, както и имена на функции.

С поглед към примерите по-горе, може да се покаже общия вид на една SQL заявка:

```
SELECT <attribute list>
FROM <table list>
WHERE <test expression>
```

Изразите обхващат както редовете, свързани с конкретна заявка (редовете за хората, които живеят в "MA") така и структурата на базата данни (Person.addr свързан с Address.ID). В резултат на заявката се дава списък от редове, всеки с избрания списък от атрибути. Изпълнението на конкретната заявка дава едно решение, което съответства на един човек живее в "MA":

fname	city
Bob	**Cambridge**

SPARQL заявката има подобна структура:

```
SELECT <variable list>
WHERE {<graph pattern>}
```

Променливите в <variable list> са обвързани с модела на графа. Моделът <graph pattern> прилича на твърденията чрез тройки, но субектът, предиката или обекта може да бъдат заменени с променлива (термин започващ с "?") .

```
SELECT ?name ?city
WHERE {
```

```
    ?who <Person#fname> ?name ;
    <Person#addr> ?adr .
    ?adr <Address#city> ?city ;
    <Address#state> "MA"}
```

name	city
" Bob "	" Cambridge "

Заявката дава един резултат с променливи name и city: показаните като литерали " Bob" и " Cambridge ". Терминте в резултатите от SPARQL заявките са представени по същия начин като в SPARQL заявките или Turtle твърденията на модела. Заглавията на колоните в SPARQL резултатите са променливите, които се появяват в WHERE {<graph pattern>}, докато заглавията на колоните в SQL резултатите са имената на атрибути (колони) в SQL схемата.

SQL използва символа **NULL** за да покаже, че данните не са на разположение или не са подходящи. Обаче **join constraints** обикновено елиминират редове ако няма съответстващи стойности. Операторът LEFT OUTER JOIN осъществява свързване но не елиминира резултати, ако join constraints не са спазени. Избира се съответстващия на търсенето ред от таблицата, дори ако избраните атрибути са NULL.

Следната заявка ще изведе стойностите на атрибута fname за всички записи, дори и ако нямат запис за атибута city:

```
SELECT Person.fname, Address.city
FROM Person
LEFT OUTER JOIN Address ON Person.addr=Address.ID
WHERE Address.state="MA"
```

fname	city
Bob	**Cambridge**
Sue	**NULL**

SPARQL използва ключовата дума OPTIONAL вместо LEFT OUTER JOIN, но ефектът е подобен:

```
SELECT ?name ?city
```

```
        WHERE {
           ?who < Person#fname> ?name .
           OPTIONAL {
              ?who < Person#addr> ?adr .
              ?adr < Address#city> ?city ;
              < Address#state> "MA"    }
```

Въпреки, че семантиката на двата езика е аналогична, има видима разлика в третирането на липсващите данни. Липсващите данни просто не се изразяват в RDF. Също така (а впоследствие), графите няма да се свържат, ако има липсващи атрибути, например като адресът на Sue горе. В една SPARQL заявка търсеща имената на всички и техните адреси трябва да направи атрибута adrr OPTIONAL, за да намери на запис на Sue:

```
        SELECT ?name ?city
WHERE {
   ?who < Person#fname> ?name .
   OPTIONAL {?who < Person#addr> ?adr }
```

А наистина аналогична SQL заявка е:

```
        SELECT Person.fname, Person.addr
FROM Person
LEFT OUTER JOIN Address ON Person.addr=Address.ID
WHERE Address.state="MA" AND Person.addr IS NOT NULL
```

SPARQL и SQL имат много сходни UNION и MINUS оператори.

Тъй като типовете данни на таблица при SQL се приемат за еднакви във всички редове, трябва да се подравнят типовете данни в SELECT. При някои SQL бази данни, прилагането на това правило се оставя на потребителите с някои полезни съобщения за грешка ако се установи разминаване.

SQL базите данни са хранилища на данни, с набор от таблици, запълнени с редове с данни. SQL заявки работят в дадена база данни. SPARQL заявките се различават по това дали имат или нямат предварително определена RDF база данни. Примерът по-горе предполага, че RDF моделът е достъпен за заявки. Ако базата данни RDF, указана по подразбиране е празна, или не съдържа данните, необходими за заявката, ще трябва да се посочи откъде да се вземат данните. За това SPARQL използва SQL клаузата FROM, за да се идентифицират уеб ресурси, които трябва да бъдат налични, за да извърши заявката:

```
        SELECT ?name ?city
FROM<http://example.org/AddressBook>
WHERE { ... }
```

Комбинирането на навигация с изследване е много лесно в SPARQL. Ако искаме да разберем какво се знае за реакции, включващи **illudium phosdex** , може да се напиша заявката:

```
        SELECT ?reaction ?p ?o
WHERE {
   ?compound ex:name "illudium phosdex" ;
   ?reaction ex:involves ?compound ;
   ?reaction ?p ?o }
```

В SQL, това ще бъде като:

```
        SELECT reactions.*
        FROM reactions, compounds
        WHERE reactions.compoundID=compounds.ID
        AND compounds.name="illudium phosdex"
```

Основната характеристика на SPARQL която впечатлява SQL потребителите е възможността за федеративни заявки в различни хранилища. RDF осигурява основата за интегриране на графи и RDF софтуерните средства позволява тривиално извличане на множество документи с данни. Интегриране на големи бази данни е също лесно, но вместо да се извличат данни и да се обединяват локално, една SPARQL заявка може да делегира части от заявката към отдалечени SPARQL услуги, например поръчките, които предстои да бъдат изпратени тази седмица поръчки се извличат чрез следната заявка:

```
        SELECT ?order ?handler
WHERE {
    SERVICE<sales> {
       ?order ex:soldBy ?handler ;
       ex:dueDate ?due
       FILTER (?due> "2012-02-22"&& ?due< "2012-02-29"
    } MINUS {
       SERVICE<fulfillment> { ?order ex:shipped ?shipped }
       } }
```

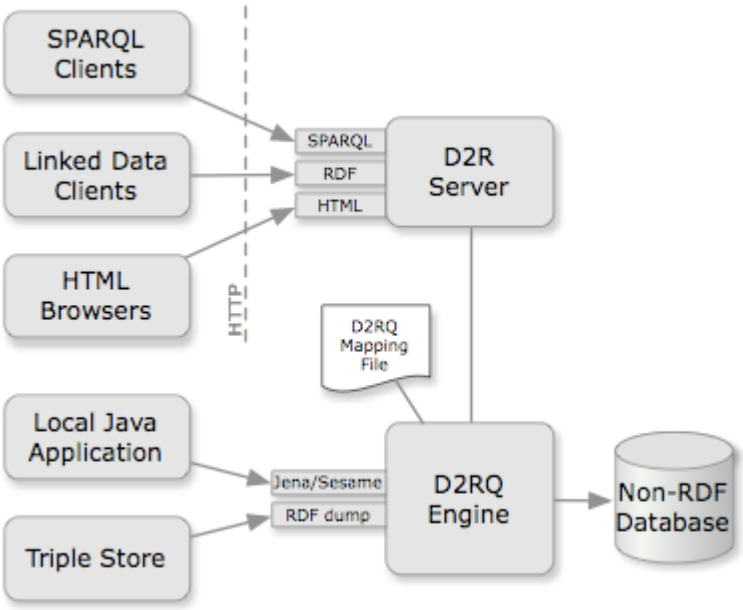
SQL не е стандартно средство за федеративни заявки. Различни средства, повече или по-малко чрез SQL синтаксис осигуряват достъп до предварително определен набор от бази данни. Чрез MySQL FROM може да се присъединят данни от различни виртуални бази данни, изпълнявани на същия MySQL сървър. Oracle Database Streams и SQL Server Integration Services използват този синтаксис за свързване с бази данни, с които е било установено съответствие към локалните схеми чрез ръчно конфигуриране.

3.3. Софтуерна продукти за трансформация на релационни бази данни в онтологии

D2RQ (http://d2rq.org/) представлява софтуерна платформа за достъп до релационни бази данни чрез генериране на виртуален read-only RDF граф. Тя предоставя RDF-базиран достъп до съдържанието на релационна база данни без да създава реплика в RDF хранилища. D2R процесор представя данните от базата данни в RDF формат. Прототипът на D2R процесор е публикуван под GNU Public License лиценз.

Платформата D2RQ се състои от:

- Език **D2RQ Mapping Language –** декларативен **XML**-базиран език за описание на съответствието между релационните схеми и OWL/RDFS онтологиите
- **D2RQ Engine** - plug-in в Jena **Semantic** Web toolkit, който използва езика за пренаписване на заявките в SQL чрез Jena API към базата данни и получаване на резултатите от тях
- **D2R Server** - **HTTP** server, който позволява изглед към Linked Data - един HTML изглед за създаване на SPARQL заявки върху базата данни



Имплементацията на D2RQ е на езика Java и е базирана на Jena API, като поддържаните експортни формати са RDF, N3, N-TRIPLES и Jena models, поддържат се всички релационни бази данни, които предлагат JDBC или ODBC достъп. Процесорът може да се използва в servlet среда за динамично публикуване на XHTML страници, съдържащи RDF като конектор с бази данни в приложения с Jena модели, или като a command line tool.

D2R Server може да бъде стартиран като stand-alone server application.

```
d2r-server [-p port] [-b serverBaseURI]
    [--fast] [--verbose] [--debug]
    mapping-file.ttl
```

Чрез *generate-mapping* се създава D2RQ трансформиращ файл, анализирайки схемата на базата данни. Всяка таблица се трансформира в нов RDF клас със същото име, всяко поле се трансформира в свойство тип данни. Този файл може да бъде редактиран от потребителя.

```
generate-mapping [-u user] [-p password] [-d driver]
    [-l script.sql] [--[skip-](schemas|tables|columns) list]
    [--w3c] [-v] [-b baseURI] [-o outfile.ttl]
    [--verbose] [--debug]
    jdbcURL
```

Чрез *d2r-query* се осъществяват SPARQL заявки към релационната база данни от команден ред. Ако е избран трансформиращ файл се отправя запитване към виртуален RDF граф, дефиниран при трансформацията. Ако не е избран трансформиращ файл, създава се такъв по подразбиране.

```
d2r-query [-f format] [-b baseURI] [-t timeout] [--verbose] [--debug] mapping-file.ttl query
```

Заявките може да са предварително създадени и записани във файл с име query.sparql:

```
d2r-query mapping.ttl @query.sparql
```

Чрез dump-rdf се записва във файл моментан снимка „dump” на данните в базата данни в избран RDF синтаксис. Това може да се направи със или без D2RQ mapping file.

```
dump-rdf [-f format] [-b baseURI] [-o outfile.ttl] [--verbose] [--debug]  mapping-file.ttl
```

Достъпът до трансформираните чрез D2RQ данни може да се осъществи чрез:

- **RDF dumps** в RDF хранилища за данни в RDF/XML или N-Triples формат
- **RDF APIs**, където D2RQ е вграден в Java приложения за достъп до релационни данни в чрез Jena and Sesame APIs. Приложенията

„пренаписват“ в SQL заявките и ги изпълняват в базата данни. Проектът R2D2 предлага достъп до D2RQ -трансформирани данни от PHP приложения

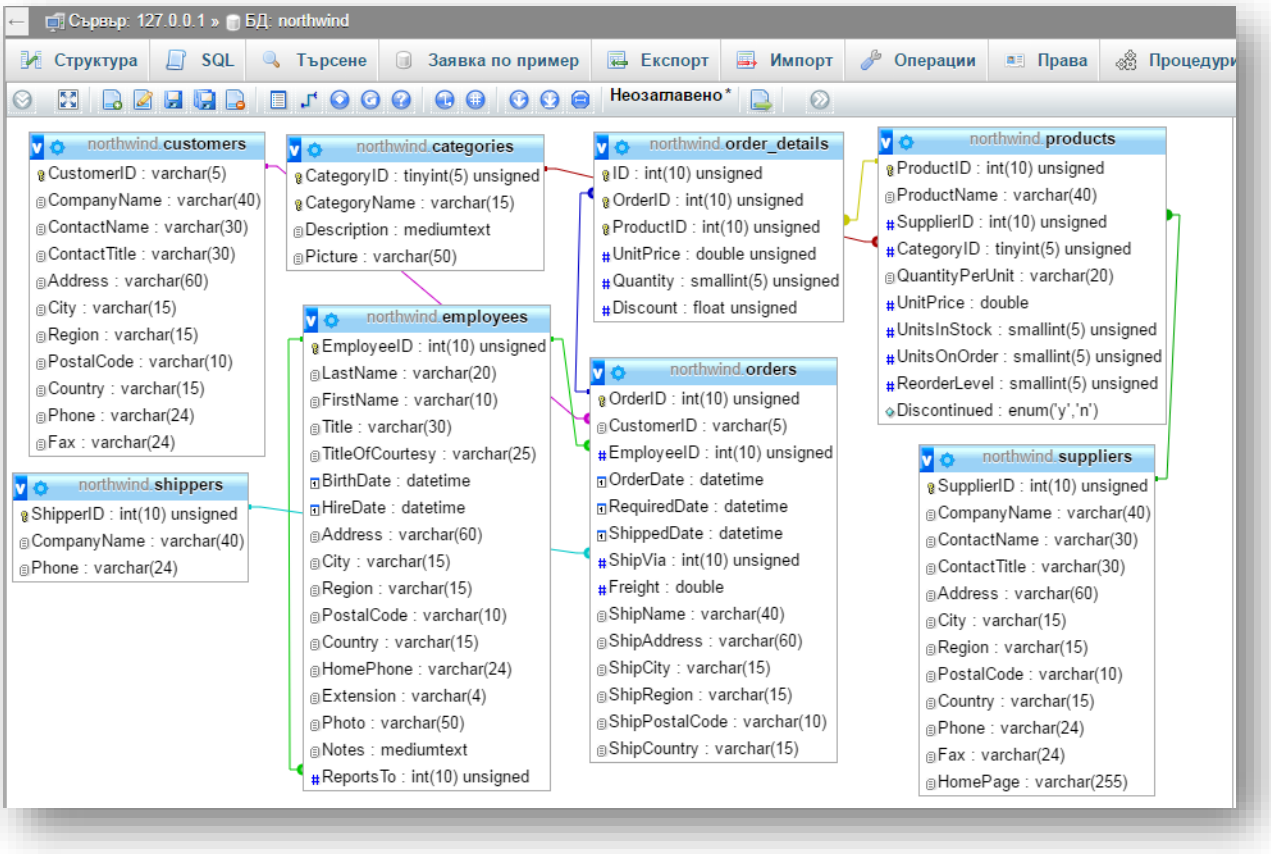
- **Чрез SPARQL** протокол D2R сървър осигурява отдалечен достъп до D2RQ - трансформирани база данни през **SPARQL крайни точки**.
- **D2R** сървър осигурява и достъп чрез **HTML страница**.
- **D2RQ** е интегриран в софтуерните среди **TopBraid Composer** и Gnowsis Semantic Desktop.

4. ТРАНСФОРМАЦИЯ НА РЕЛАЦИОННА БАЗА ДАННИ В ОНТОЛОГИЯ

4.1. Релационна база данни и SQL заявки

Използвана е примерна база данни Northwind и phpMyAdmin - уеб базиран софтуер за администрация на MySQL.

Базата данни Northwind се състои от 7 таблици “**categories**“, “**customers**“,“**emploees**“,“**orders**“,“**order_details**“,“**produsts**“,“**shippers**“ и “**supliers**“. Релациите базата данни са представени на фиг.5.



Фиг.5 Релации в базата данни Northwind

Таблица “**categories”** се състои от следните колони : „CategoryID” – това е първичен ключ, който е уникален и служи за ID и връзки с другите таблици. „CategoryName“ – в това поле са показани различните категории храни и напитки. „Description“ – това поле показва описанието на категориите храни и напитки. “Picture” – снимков материал на отделните категории.

CategoryID	CategoryName	Description	Picture
1	Beverages	Soft drinks, coffees, teas, beers, and ales	beverages.gif
2	Condiments	Sweet and savory sauces, relishes, spreads, and se...	condiments.gif
3	Confections	Desserts, candies, and sweet breads	confections.gif
4	Dairy Products	Cheeses	diary.gif
5	Grains/Cereals	Breads, crackers, pasta, and cereal	cereals.gif
6	Meat/Poultry	Prepared meats	meat.gif
7	Produce	Dried fruit and bean curd	produce.gif
8	Seafood	Seaweed and fish	seafood.gif

Фиг.6 Таблица „**categories**“

Таблица **„customers“** се състои от следните колони: „CustomerID“ - това е първичен ключ. Полето „CompanyName“ показва списък с имената на компаниите клиенти. „ContactName“ покзава информация с лицата за контакти, представящи всяка компания. „ContactTitle“ показва работната позиция която заема всеки представител в компанията за която работи. „Address“ – дава информация за адреса на компанията клиент. „City“ градът, в който се помещава компанията. „Region“- в конкретният случай няма стойности- null. „PostalCode“ – пощенският код на компанията. „Country“ – страната в която е регистрирана. „Phone“ – телефон за връза с компанията. „Fax“ – факс номер на компанията.

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
ALFAA	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	030-0074321	030-0076545
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico	(5) 555-4729	(5) 555-3745
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexico	(5) 555-3932	
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1 1DP	UK	(171) 555-7788	(171) 555-6750
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958 22	Sweden	0921-12 34 65	0921-12 34 67
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim		68306	Germany	0621-08460	0621-08924
BLONP	Blondel père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg		67000	France	88.60.15.31	88.60.15.32
BOLID	Bólido Comidas preparadas	Martín Sommer	Owner	C/ Araquil, 67	Madrid		28023	Spain	(91) 555 22 82	(91) 555 91 99
BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers	Marseille		13008	France	91.24.45.40	91.24.45.41
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsawassen	BC	T2F 8M4	Canada	(604) 555-4729	(604) 555-3745
BSBEV	B's Beverages	Victoria Ashworth	Sales Representative	Fauntleroy Circus	London		EC2 5NT	UK	(171) 555-1212	
CACTU	Cactus Comidas para llevar	Patricio Simpson	Sales Agent	Cerrito 333	Buenos Aires		1010	Argentina	(1) 135-5555	(1) 135-4892

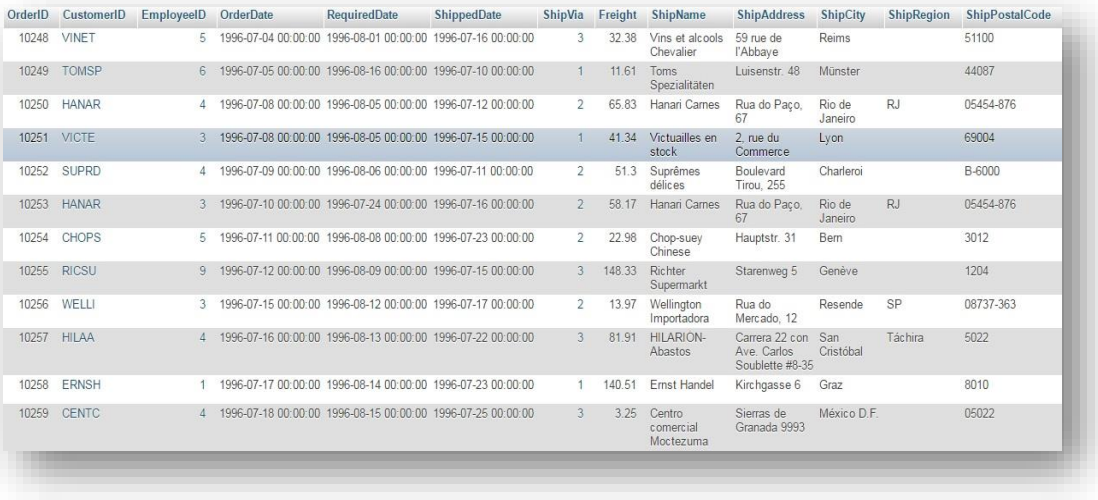
Фиг.7 Таблица „**customers**“

Таблица **„employees“** притежава следните полета: „EmployeeID“ – първичен ключ. „LastName“- информация за фамилиите на служителите. „FirstName“ – информация за собствените имена на служителите. „Title“ – работна позиция. „TitleOfCourtesy“- г-н или г-жа. „BirthDate“ – рожденна дата и час(няма информация за часовете). „HireDate“ – дата на постъпване на работа. „Address“ – адрес на местоживеене. „City“ – град на место живеене. „Region“ – регион на местоживеене. „PostalCode“ – пощенски код. „ Country“ – държава. „HomePhone“ – домашен телефон за връзка. „Extension“ – удължение на номер на който отговаря на работа. „Photo“ – снимка на служителя. „Notes“ – допълнителна информация на служителя за образование и придобити квалификации. „ReportsTo“ – с този номер е означен началникът пред който отговарят слижителите.

EmployeeID	LastName	FirstName	Title	TitleOfCourtesy	BirthDate	HireDate	Address	City	Region	PostalCode	Country	HomePhone	Extension	Photo
1	Davolio	Nancy	Sales Representative	Ms.	1968-12-08 00:00:00	1992-05-01 00:00:00	507 - 20th Ave. E. Apt. 2A	Seattle	WA	98122	USA	(206) 555-9857	5467	nancy.jpg
2	Fuller	Andrew	Vice President, Sales	Dr.	1952-02-19 00:00:00	1992-08-14 00:00:00	908 W. Capital Way	Tacoma	WA	98401	USA	(206) 555-9482	3457	andrew.jpg
3	Leverling	Janet	Sales Representative	Ms.	1963-08-30 00:00:00	1992-04-01 00:00:00	722 Moss Bay Blvd.	Kirkland	WA	98033	USA	(206) 555-3412	3355	janet.jpg
4	Peacock	Margaret	Sales Representative	Mrs.	1958-09-19 00:00:00	1993-05-03 00:00:00	4110 Old Redmond Rd.	Redmond	WA	98052	USA	(206) 555-8122	5176	margaret.jpg
5	Buchanan	Steven	Sales Manager	Mr.	1955-03-04 00:00:00	1993-10-17 00:00:00	14 Garrett Hill	London		SW1 8JR	UK	(71) 555-4848	3453	steven.jpg
6	Suyama	Michael	Sales Representative	Mr.	1963-07-02 00:00:00	1993-10-17 00:00:00	Coventry House Miner Rd.	London		EC2 7JR	UK	(71) 555-7773	428	michael.jpg

Фиг.8 Таблица „**employees**“

Таблица **„orders“** се състои от следните колони: „OrderID“ – дава информация за всяка една поръчка под какъв номер е записана. „CustomerID“ – кой е клиентът за всяка една поръчка, „EmployeeID“ – кой служител е поел поръчката, „OrderDate“ – на коя дата е приета поръчката, „RequiredDate“- изискуема дата, „ShippedDate“ – дата на която е тръгнала пратката, „ShipVia“ – метод на транспортиране, „Freight“ - тегло на пратката, „ShipName“ – име на получател на пратката, „ShipAddress“ – адрес на доставка на пратката, „ShipCity“ – градът до койтп трябва да се достави пратката, „ ShipRegion“ – региона до който трябва да стигне пратката, „ShipPostalCode“ – пощенски код на адреса за доставка на пратката, „ShipCountry“ – държавата до която трябва да се достави пратката.



OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate	ShipVia	Freight	ShipName	ShipAddress	ShipCity	ShipRegion	ShipPostalCode
10248	VINET	5	1996-07-04 00:00:00	1996-08-01 00:00:00	1996-07-16 00:00:00	3	32.38	Vins et alcools Chevalier	59 rue de l'Abbaye	Reims		51100
10249	TOMSP	6	1996-07-05 00:00:00	1996-08-16 00:00:00	1996-07-10 00:00:00	1	11.61	Toms Spezialitäten	Luisenstr. 48	Münster		44087
10250	HANAR	4	1996-07-08 00:00:00	1996-08-05 00:00:00	1996-07-12 00:00:00	2	65.83	Hanari Carnes	Rua do Paço, 67	Rio de Janeiro	RJ	05454-876
10251	VICTE	3	1996-07-08 00:00:00	1996-08-05 00:00:00	1996-07-15 00:00:00	1	41.34	Victuailles en stock	2, rue du Commerce	Lyon		69004
10252	SUPRD	4	1996-07-09 00:00:00	1996-08-06 00:00:00	1996-07-11 00:00:00	2	51.3	Suprêmes délices	Boulevard Tirou, 255	Charleroi		B-6000
10253	HANAR	3	1996-07-10 00:00:00	1996-07-24 00:00:00	1996-07-16 00:00:00	2	58.17	Hanari Carnes	Rua do Paço, 67	Rio de Janeiro	RJ	05454-876
10254	CHOPS	5	1996-07-11 00:00:00	1996-08-08 00:00:00	1996-07-23 00:00:00	2	22.98	Chop-suey Chinese	Hauptstr. 31	Bern		3012
10255	RICSU	9	1996-07-12 00:00:00	1996-08-09 00:00:00	1996-07-15 00:00:00	3	148.33	Richter Supermarkt	Starenweg 5	Genève		1204
10256	WELLI	3	1996-07-15 00:00:00	1996-08-12 00:00:00	1996-07-17 00:00:00	2	13.97	Wellington Importadora	Rua do Mercado, 12	Resende	SP	08737-363
10257	HILAA	4	1996-07-16 00:00:00	1996-08-13 00:00:00	1996-07-22 00:00:00	3	81.91	HILARION-Abastos	Carrera 22 con Ave. Carlos Soublette #8-35	San Cristóbal	Táchira	5022
10258	ERNSH	1	1996-07-17 00:00:00	1996-08-14 00:00:00	1996-07-23 00:00:00	1	140.51	Ernst Handel	Kirchgasse 6	Graz		8010
10259	CENTC	4	1996-07-18 00:00:00	1996-08-15 00:00:00	1996-07-25 00:00:00	3	3.25	Centro comercial Moctezuma	Sierras de Granada 9993	México D.F.		05022

Фиг.9 Таблица „**orders**“

Таблица **„products“** се състои от следните колони: „ProductID“ – поредният номер по който е записан всеки продукт който се предлага за продаване, „ProductName“ – клиенцкото името на продукта, „SupplierID“ – поредният номер с който се оказва кой е доставчика на този продукт, „CategoryID“ – показва към коя хранителна категория се приспада конкретният продукт, „QuanityPerUnit“ – количестово на продукта за единична опаковка, „UnitPrice“ – цената на всяка една опаковка, „UnitsInStock“ – какво е текущото количество на склад, „UnitOnOrder“ – колко опаковки са поръчвани от доставчиците. „ReorderLevel“ – ниво на поръчваемост. Полето „Discontinued“ - показва кои стоки са спрени от производство или не са налични.

ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued
1	Chai	1	1	10 boxes x 20 bags	18	39	0	10	n
2	Chang	1	1	24 - 12 oz bottles	19	17	40	25	n
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10	13	70	25	n
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22	53	0	0	n
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35	0	0	0	y
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	25	120	0	25	n
7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 lb pkgs.	30	15	0	10	n
8	Northwoods Cranberry Sauce	3	2	12 - 12 oz jars	40	6	0	0	n
9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	97	29	0	0	y
10	Ikura	4	8	12 - 200 ml jars	31	31	0	0	n
11	Queso Cabrales	5	4	1 kg pkg.	21	22	30	30	n
12	Queso Manchego La Pastora	5	4	10 - 500 g pkgs.	38	86	0	0	n
13	Konbu	6	8	2 kg box	6	24	0	5	n
14	Tofu	6	7	40 - 100 g pkgs.	23.25	35	0	0	n
15	Genen Shouyu	6	2	24 - 250 ml bottles	15.5	39	0	5	n
16	Pavlova	7	3	32 - 500 g boxes	17.45	29	0	10	n

Фиг.10 Таблица „**products**“

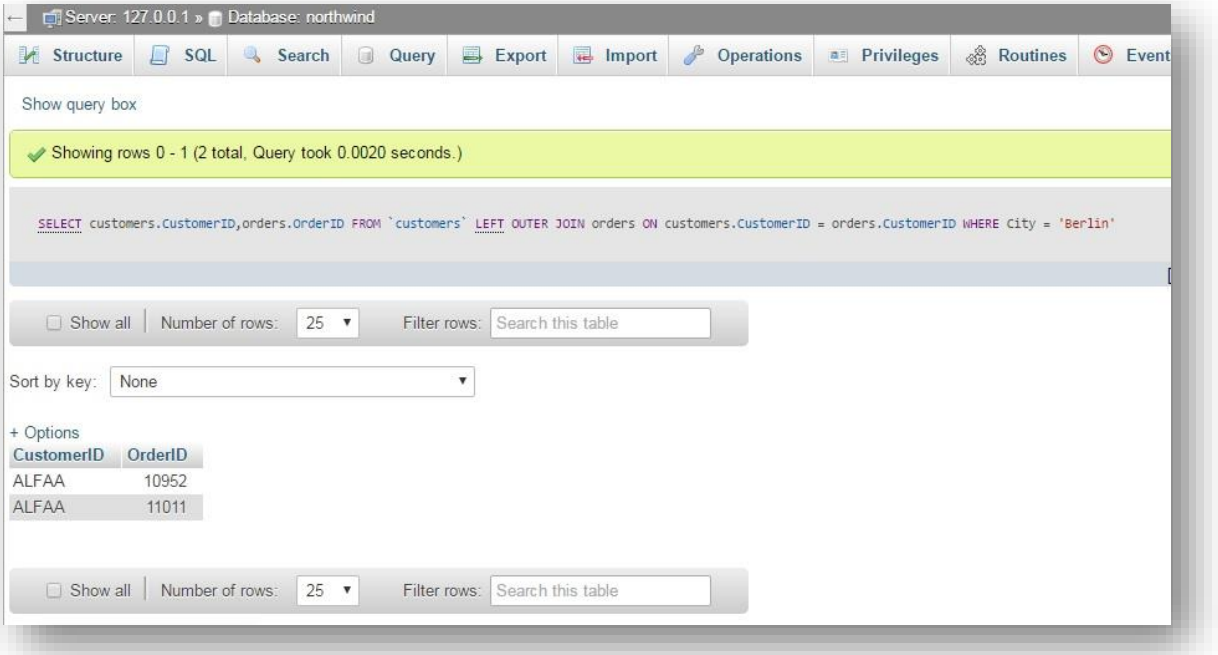
Таблица **„suppliers“** се състои от следните колони: „SupplierID“ – поредният номер с който е означен всеки един доставчикът на даден продукт(и), „CompanyName“ – името на всяка компания от даден пореден номер, „ContactName“ - лице за връзка към доставчика, „ContactTitle“ – позицията която заема лицето за контакти в фирмата доставчик, „ Address“ – адреса на който се помещава доставчика, „City“ – града в който се помещава доставчика, „Region“ – региона в който се помещава доставчика, „ PostalCode“ – пощенският адрес на фирмата доставчик, „Country“ – страната в която е регистрирана фирмата, „Phone“ - телефон за контакти с доставчика. „Fax“ – факс. „HomePage“ – уеб страница на доставчика.

SupplierID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax	HomePage
1	Exotic Liquids	Charlotte Cooper	Purchasing Manager	49 Gilbert St.	London		EC1 4SD	UK	(171) 555-2222		
2	New Orleans Cajun Delights	Shelley Burke	Order Administrator	P.O. Box 78934	New Orleans	LA	70117	USA	(100) 555-4822		#CAJUN.HTM#
3	Grandma Kelly's Homestead	Regina Murphy	Sales Representative	707 Oxford Rd.	Ann Arbor	MI	48104	USA	(313) 555-5735	(313) 555-3349	
4	Tokyo Traders	Yoshi Nagase	Marketing Manager	9-8 Sekimai Musashino-shi	Tokyo		100	Japan	(03) 3555-5011		
5	Cooperativa de Quesos 'Las Cabras'	Antonio del Valle Saavedra	Export Administrator	Calle del Rosal 4	Oviedo	Asturias	33007	Spain	(98) 598 76 54		
6	Mayumi's	Mayumi Ohno	Marketing Representative	92 Setsuko Chuo-ku	Osaka		545	Japan	(06) 431-7877		Mayumi's (on the World Wide Web)#http://www.micros...
7	Pavlova, Ltd.	Ian Devling	Marketing Manager	74 Rose St. Moonie Ponds	Melbourne	Victoria	3058	Australia	(03) 444-2343	(03) 444-6588	
8	Specialty Biscuits, Ltd.	Peter Wilson	Sales Representative	29 King's Way	Manchester		M14 GSD	UK	(161) 555-4448		
9	PB Knäckebröd AB	Lars Peterson	Sales Agent	Kaloadagatan 13	Göteborg		S-345 67	Sweden	031-987 65 43	031-987 65 91	
10	Refrescos Americanas LTDA	Carlos Diaz	Marketing Manager	Av. das Americanas 12.890	São Paulo		5442	Brazil	(11) 555 4640		
11	Heli Süßwaren GmbH & Co. KG	Petra Winkler	Sales Manager	Tiergartenstraße 5	Berlin		10785	Germany	(010) 9984510		
12	Plutzer Lebensmittelgroßmärkte AG	Martin Bein	International Marketing Mgr.	Bogenallee 51	Frankfurt		60439	Germany	(069) 992755		Plutzer (on the World Wide Web)#http://www.microso...
13	Nord-Ost-Fisch Handelsgesellschaft mbH	Sven Petersen	Coordinator Foreign Markets	Frahmredder 112a	Cuxhaven		27478	Germany	(04721) 8713	(04721) 8714	
14	Formaggi Fortini s.r.l.	Elio Rossi	Sales Representative	Viale Dante, 75	Ravenna		48100	Italy	(0544) 60323	(0544) 60603	#FORMAGGI.HTM#
15	Norske Meierier	Beate Vileid	Marketing Manager	Hatlevegen 5	Sandvika		1320	Norway	(0)2-953010		
16	Bigfoot Breweries	Cheryl Saylor	Regional Account Rep.	3400 - 8th Avenue	Bend	OR	97101	USA	(503) 555-9931		

Фиг.11 Таблица „**suppliers**“

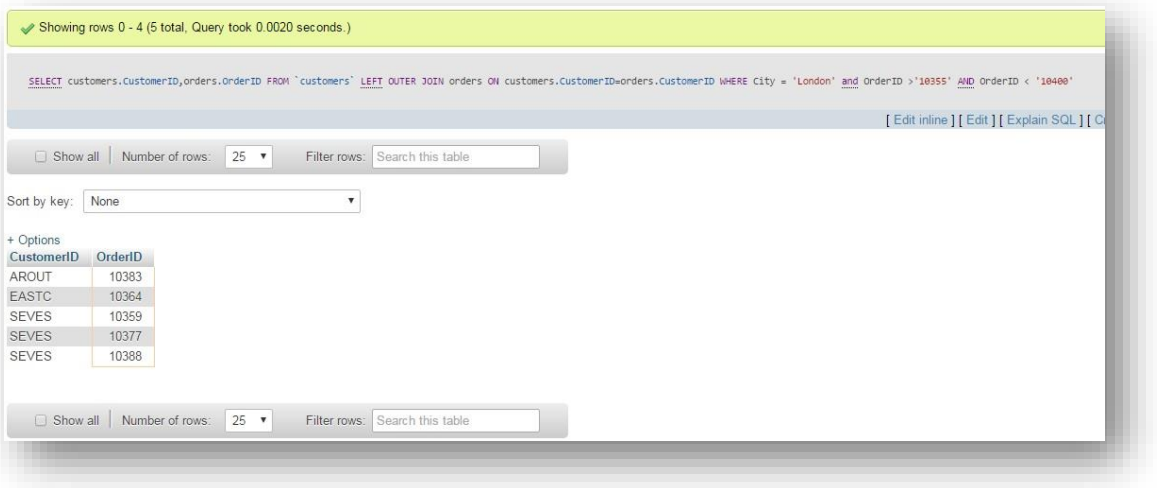
Фиг.12 Заявка в phpMyAdmin (1)

На фигура 12 е показана заявка за извличане на данни от базата данни в средата на phpMyAdmin: заявка за търсене на имената и градовете, в които живеят всички служители от Великобритания (в случая са само от Лондон)



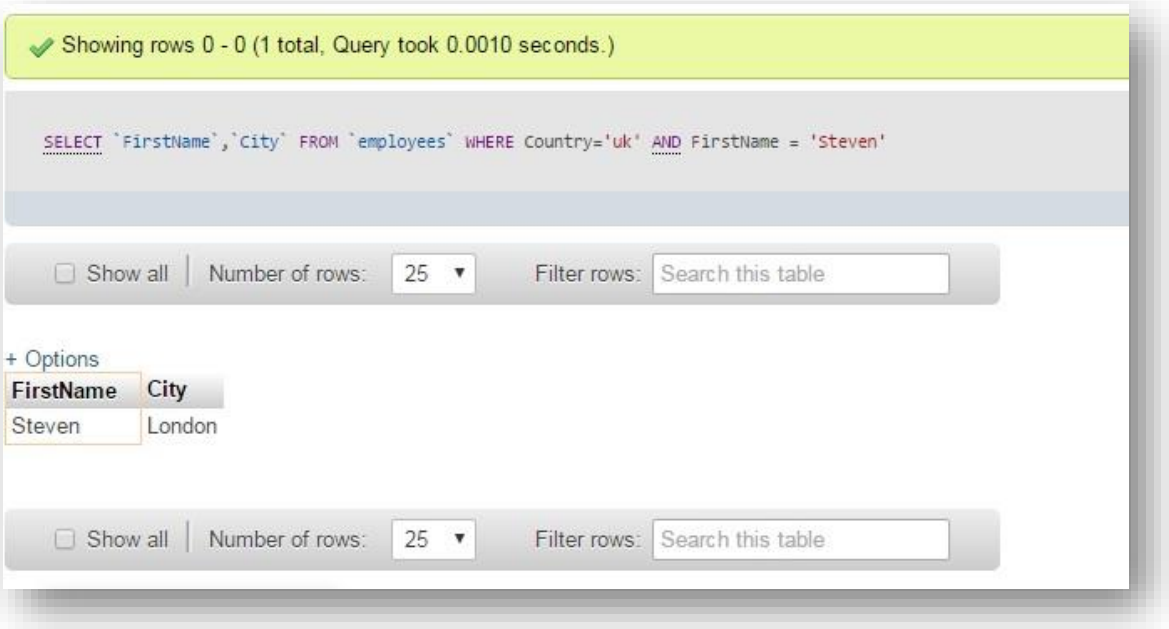
Фиг.13 Заявка в phpMyAdmin (2)

На фигура 13 е показана заявка за извличане на информация за всички клиентски идентификационни ключове и техните поръчки от таблица „Customers“.Ако няма информация за тях в таблицата да се свърже чрез техните поръчки с таблица „Orders“ само за клиентите от град Берлин.



Фиг.14 Заявка в phpMyAdmin (3)

На фигура 14 е показана заявка за извличане на информация за всички клиентски идентификационни ключове и техните поръчки от таблица „Customers“ и ако няма информация за тях в таблицата да се свърже чрез техните поръчки с таблица „Orders“ само за клиентите от град Лондон като поръчките са ограничени в рамките от 10300 до 10400 OrderID.



Фиг.15 Заявка в phpMyAdmin (4)

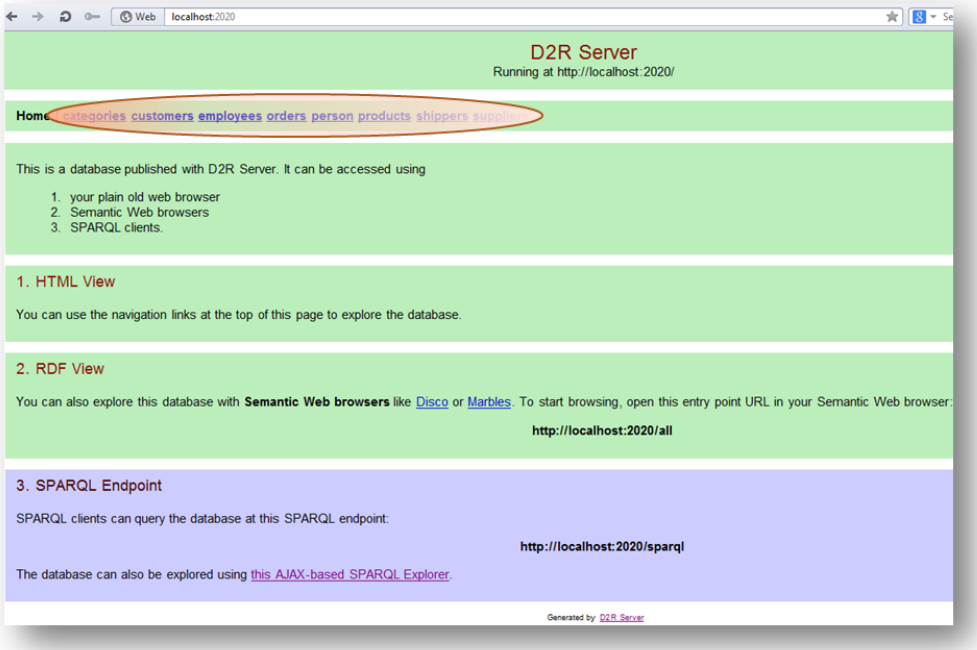
Заявката от фигура 13 показва като резултат първото име и града на служители чиято страна на местообитаване е UK и имената им са “Steven”. В случая е само един служител с това име в тази държава.

4.2. Трансформация на релационната база данни чрез D2RQ

Свалени са приложението d2rq-0.8.1 и JDBC driver за mySQL. За представяне на релационната база данни Northwind в RDF/OWL формат и за достъп до интерфейса на D2R сървър са стартирани следните команди:

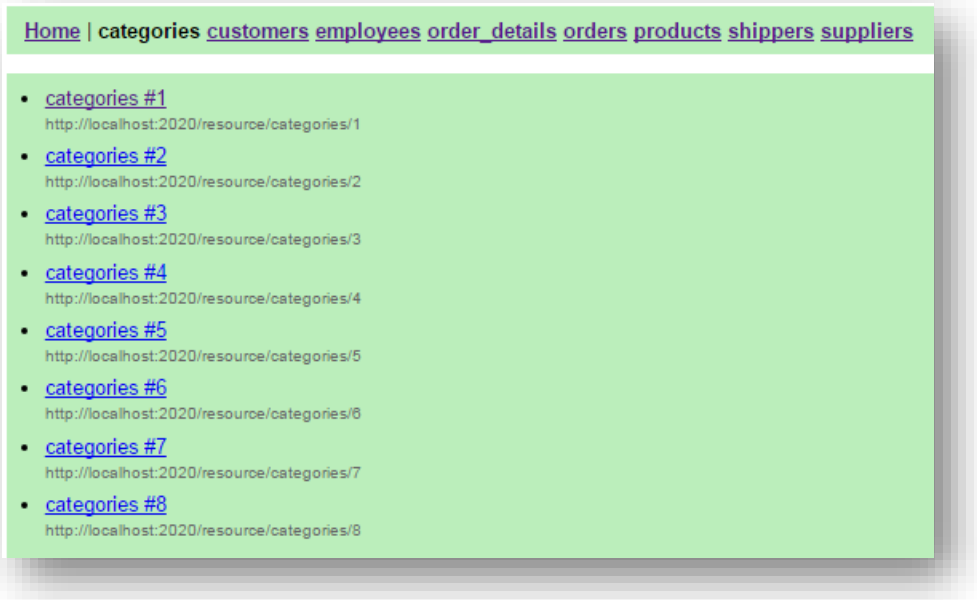
- ***generate-mapping -o mapp1.ttl -u root jdbc:mysql://localhost/northwind*** – Създава се трансформиращ файл с име mapp1.ttl за съответствието с базата данни Northwind.
- ***d2r-server mapp1.ttl*** – Стартира се D2R сървър на http://localhost:2020/ с информацията от трансформиращия файл mapp1.ttl.

Интерфейсът на D2R-server на http://localhost:2020/ е показан на фиг.16.

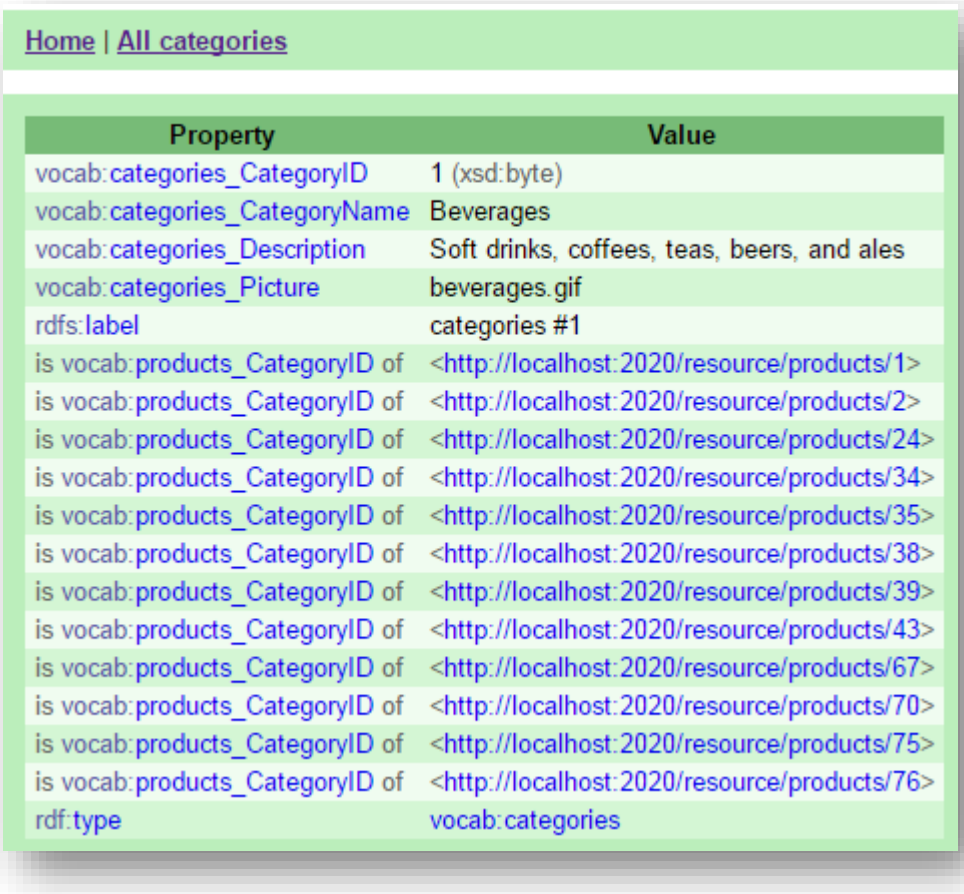


Фиг.16 Уеб страница на d2r-server с хипервръзки към таблиците от базата данни ***Northwind***

Чрез хипервръзките са достъпни таблиците, атрибутите и всички записи в базата данни; напр. на фиг.17 е представена таблицата "**categories"** като оттук може да се прегледат всички записи в таблицата (фиг.18).



Фиг.17 Записи в таблица "**cateogries**" в уеб страница на d2r-server с хипервръзки към записите в таблицата



Фиг.18 Данни за първия запис в таблица "**cateogries**" в уеб страница на d2r-server с хипервръзки към данните от таблица "products"

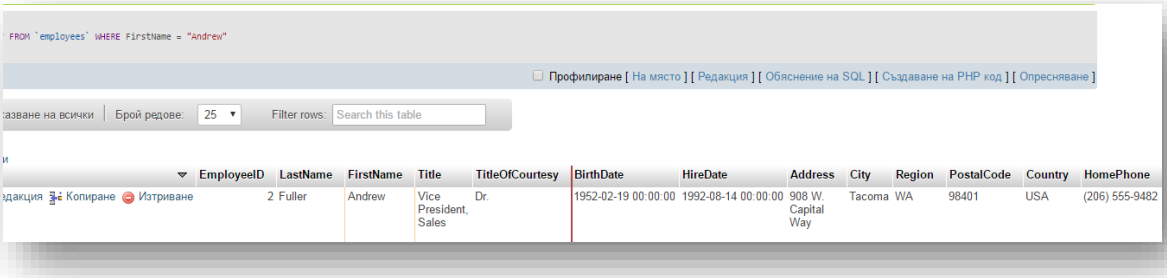
4.3. RDF модел на релационната база данни и SPARQL заявки

Чрез интерфейса на D2R server от http://localhost:2020/sparql могат да се създават SPARQL заявки към базата данни, представена в RDF формат от D2RQ mapping language. Сървърът D2R автоматично транслира SPARQL заявки в SQL заявки. Връзката с базата данни е динамична и при промяна на данните в базата данни се променя и резултата от заявката. Заявките може да бъдат генерирани автоматично чрез установените от системата D2RQ хипервръзки, или да се пишат чрез езика SPARQL в съответното поле.

На фиг.19 е представена SPARQL заявка за извличане на всички данни за един служител от таблица "**employees**"– Andrew Fuller.

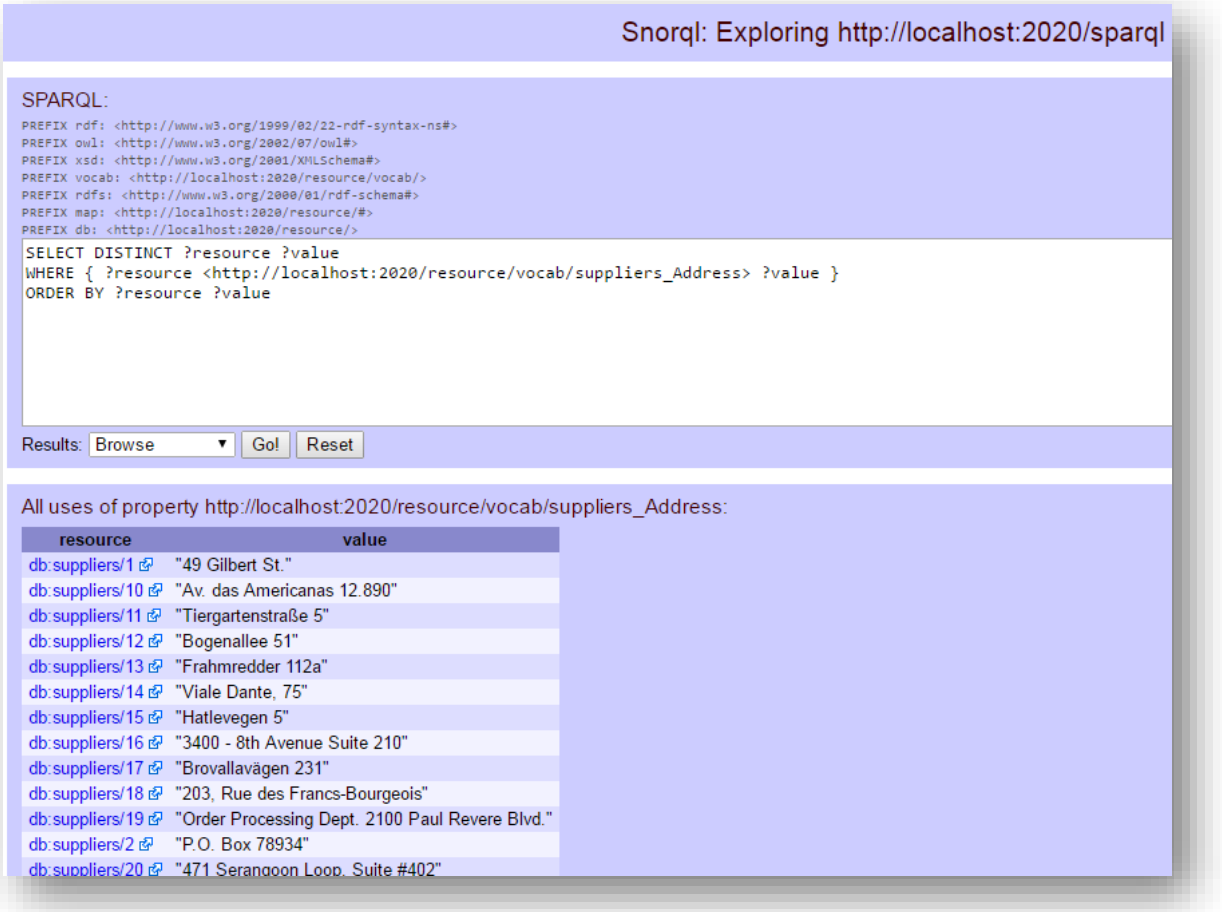
Фиг.19 Заявка и резултати от нея в интерфейса на D2Q сървър – всички данни за служител в компанията

На фиг.20 е представена същата заявка в SQLв средата на phpMyAdmin - заявка за извличане на всички данни за един служител от таблица “**employees**”– Andrew Fuller



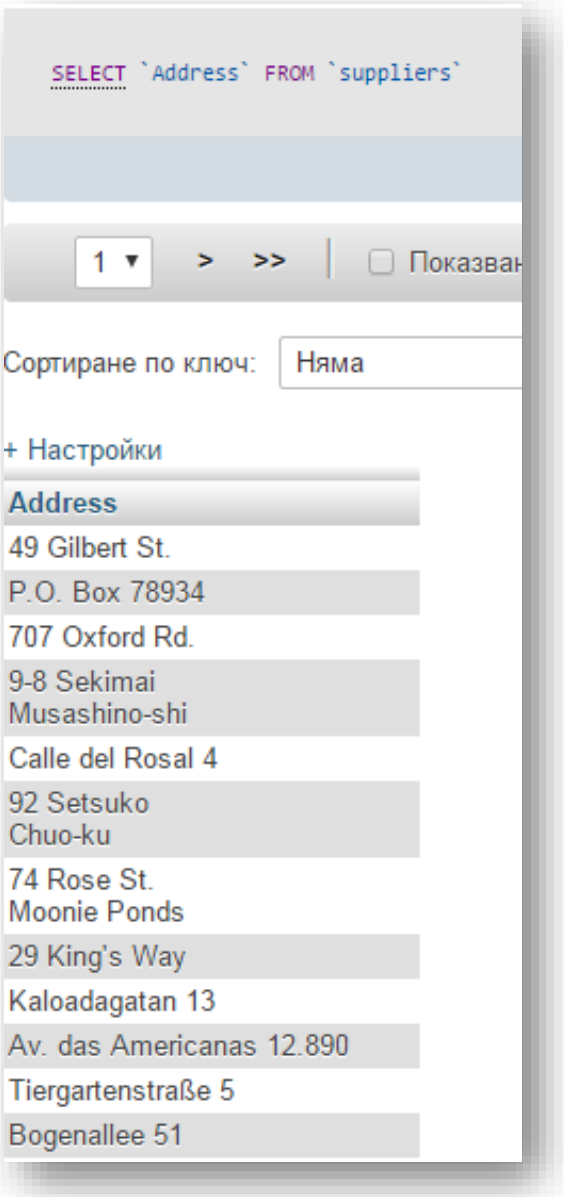
Фиг.20 SQL заявка за извличане на всички данни за един служител от таблица “**employees**”– Andrew Fuller

На фиг. 21 са представени резултатите от SPARQL заявка за извличане на всички адреси на доставчици.



Фиг.21 Резултати от SPARQL заявка за извличане на данни за адресите на доставчиците

На фиг.22 е представена същата заявка в SQLв средата на phpMyAdmin.



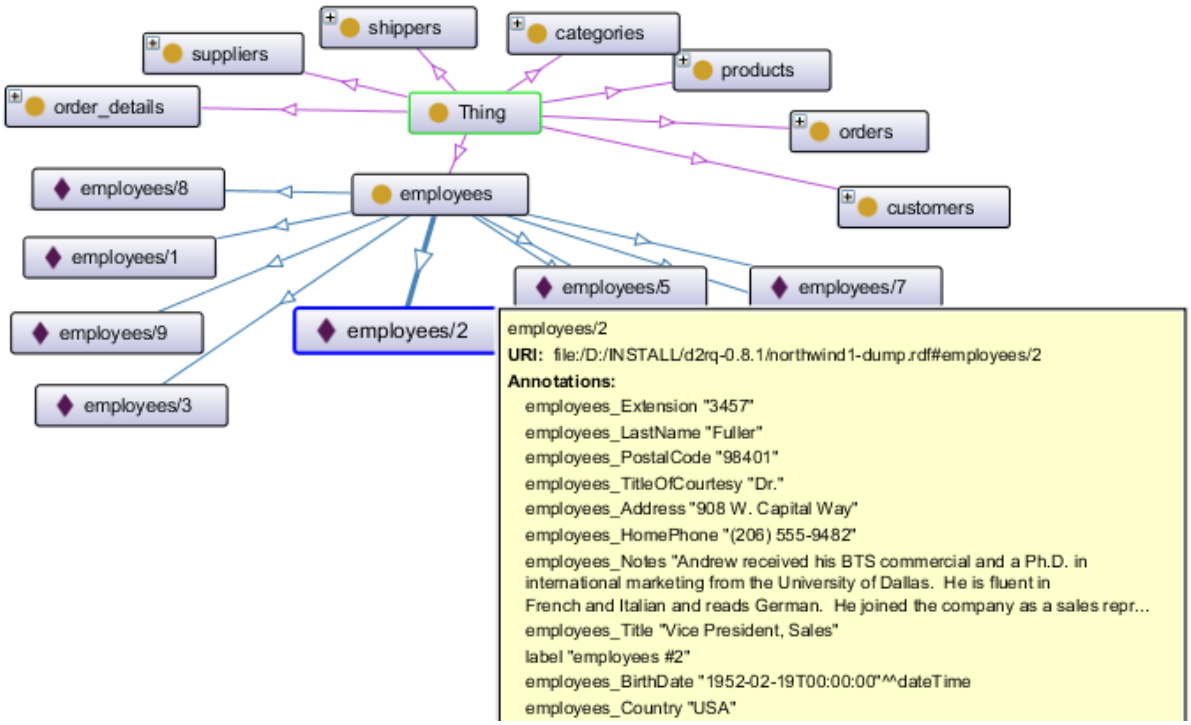
Фиг.22 Резултати от SQL заявка за извличане на данни за адресите на доставчиците

Чрез командата dump-rdf системата D2RQ създава и записва в RDF файл съдържанието на цялата база данни. Възможно е да се специфицира и конкретен RDF синтаксис: “TURTLE”, “RDF/XML”, “RDF/XML-ABBREV”, “N3”.

Чрез командата:

dump-rdf -u root -f RDF/XML-ABBREV -o northwind1-dump.rdf jdbc:mysql:///northwind

е създаден и записан файлът northwind1-dump.rdf, част от който е представен в продукта за онтологичен инженеринг Protégé 4.0 на фиг. 23. Показани са също и данните за един служител на компанията Andrew Fuller.



5. ЗАКЛЮЧЕНИЕ

Настоящата дипломна работа има за цел да представи и анализира възможностите за използване на съвременните технологии за управление на данни и начините на трансформиране на модели на данни.

Във връзка с поставените цели и задачи е постигнато следното:

- Направено е кратко въведение в системите бази данни: увод в системите бази данни; предимства и недостатъци на подхода „база данни”; основни понятия и компоненти на системите бази данни. Направено е кратко представяне на релационните бази данни и езика SQL за заявки към релационни данни
- Направено е кратко описание на технологиите за Семантичен уеб, семантични модели на данните и езика SPARQL за търсене на информация и модифициране на семантични модели на данните.
- Направен е кратък сравнителен анализ на семантичен и релационен модели на данните, както и на езиците за заявки към давата модела.
- Представена е софтуерна система за установяване на динамично съответствие между релационни и семантични модели на данните.
- На основа на примерна релационна база данни чрез динамично съпоставяне е получен семантичен модел на данните.
- Представени са примерни заявки за извличане на информация от двата модела на данните.

В дипломната работа е показано как чрез генериране на RDF граф от релационна база данни за управление на основните данни на производствено предприятие е възможно реализиране на търсене и извличане на информация за служители, доставчици, производствени заявки, данни за продукти и др.

Прилагането на семантичните технологии за управление на основните данни в предприятията улеснява обмена и споделянето на информация, подпомага унифицирането на лексиката и терминологията в предприятията, води до ефективно интегриране на хетерогенни източници на данни и подпомага извличането на точната информация в точния момент, на правилното място.

6. ЛИТЕРАТУРА

1. И. Бачкова, Системи Бази Данни, Лекционни записки, 2010, София.
2. Patel-Scheider, P. F., Antoniou, G., & Harmelen, F. (2007). A Semantic Web Primer.
3. I. Horrocks Ontologies and Databases, Presentation at the Semantic Days, 2008, posccaesar.org.
4. Michael Uschold, Ontologies and DB Schema: What's the Difference?, SemanticArts, 2011.
5. Data Modeling 101, http://www.agiledata.org/essays/dataModeling101.html
6. http://www.topquadrant.com/2011/09/30/ontologies-and-data-models-are-they-the-same/
7. Jay Funnell, David Dickson, Joel Chacon, Demystifying the Intuition Semantic Model, Control Engineering, 25 March 2012, http://www.controlengeurope.com
8. CUBIST - Combining and Uniting Business Intelligence with Semantic Technologies, CUBIST, Project No: 257403, Medium-scale Focused Research Project, FP7-ICT-2009-5, Duration: 2010/10/01-2013/09/30, Deliverable 2.2.1Semantic ETL from structured data sources.
9. Kumar, A. P., Kumar, A., & Kumar, V. N. (2011). A Comprehensive Comparative study of SPARQL and SQL. International Journal of Computer Science and Information Technologies, 2(4), 1706- 1710.
10. SPARQL vs. SQL, Technical Lesson, http://www.cambridgesemantics.com/semantic-university/sparql-vs-sql-intro