Interreg
EUROPEAN UNION
Danube Transnational Programme
KnowING IPR

Fostering Innovation in the Danube Region through Knowledge Engineering and IPR Management

Output 3.2. Knowledge database

Project Acronym	KnowING IPR			
Project Number	DTP2-076-1.1			
Project URL	http://www.interreg-danube.eu/approved-projects/knowing-ipr			
Project Coordinator	Faculty of Information Studies in Novo mesto, Slovenia			
	Name	Tamara Valič	E-mail	tamara.valic@fis.unm.si
Output Name	Knowledge database			
Output Number	O.3.2			
Responsible Author(s)	FIS, all partners			
Contractual Date of Delivery	Period 3			
Status	Final Version			
Quality assurance readers	Klara Remec, Quality manager			

Table of Contents

Project co-funded by the European Union (ERDF, IPA, ENI)
http://www.interreg-danube.eu/knowing-ipr

Project co-funded by the European Union (ERDF, IPA, ENI)
http://www.interreg-danube.eu/knowing-ipr

Introduction

The KnowING IPR project targets the topic of technology transfer, with specific focus towards managing intellectual property rights in SMEs as a factor of innovation and competitiveness. The project is funded under Danube Transnational Programme and is addressing Programme Specific Objective 1.1. Improve framework conditions for innovation. Apart from that, the Project supports PA 8 Competitiveness of enterprises of the Danube region of the Action Plan of the European Strategy for the Danube Region (EUSDR).
This document serves as a description of the tool developed within WP3, Activities A3.2. Data acquisition and A3.3 Knowledge generation.
The Knowledge base will be used as a main data source for users' queries. It will serve as a repository where structured data about research queries will be stored after application of knowledge engineering methods will be applied to the data retrieved from the sources. The Output is extremely technological in its nature, and is a result of very specific knowledge (application of knowledge engineering algorithms). As such it fits well with overall scope and plan on services offered by KnowING HUB as will be developed at later stages of the KnowING IPR project.

Structure of the Knowledge database Output

The Output is structure in three chapters: first chapter describes the module that was developed to acquire the data (national and international patenting databases). Second part of the Output describes the sources database module (module that structures the databases so that in the later stage the queries are possible within several databases in one place). The last part of the Output presents the structure and functioning of the Knowledge database with all the data in it. It presents the structure how the queries are conducted and what can be the results by the queries.

Data acquisition module

Below is the description on the module that was developed to receive and structure the data.

Technologies Used

For the development of Data Administration Module (DAM) we use Windows 10 as the underlying operating system. The major technologies used across the application is MongoDB version 3.9.1 serving as the primary Sources database and Java 1.8 installed on it. The technologies chosen for the development stage were carefully selected based on the main priorities of the project and with the aim for the project source code to be easily readable and modifiable by any other developers.
The next table is an overview of free third-party libraries we use for development.

Name	**Version**
MariaDB JDBC	2.4.0
MongoDB Java Driver	3.9.1
Jettison	1.4.0
Apache Commons Lang	3.0
JSON	20180813
Apache Commons DbUtils	1.7
Jackson Core	2.9.8
TestFX	4.0.1
JUnit	4.12
Guava	14.0.1

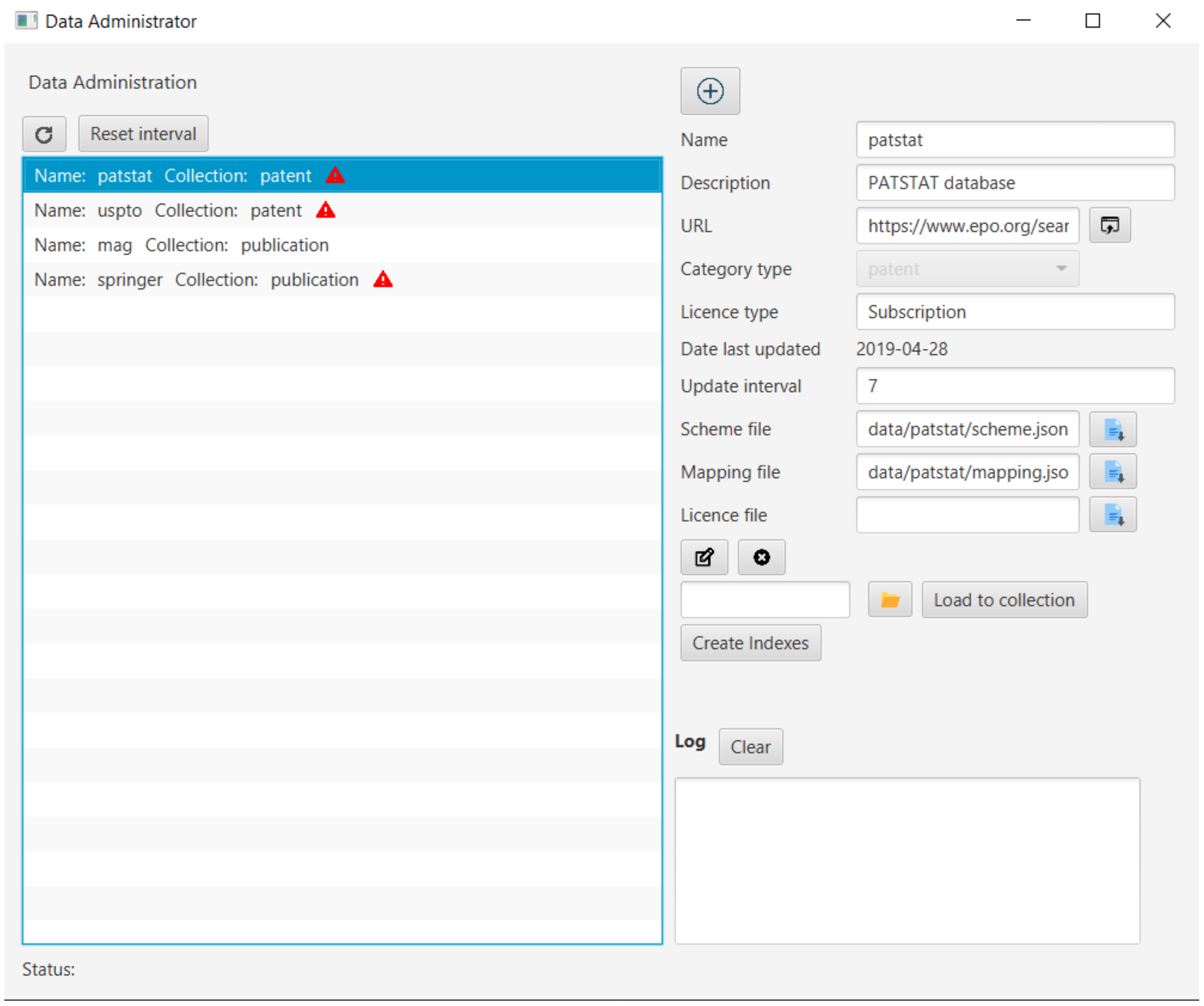
Use Cases

The main use cases in the Data Administration Module are:

- Loading data to the sources database
- Creating proper indexes on the sources database
- Creating metadata for new data sources
- Configuring file locations of files related to data sources – mapping files, structure files
- Managing the metadata for data sources – creation, deletion, update
- Managing expiration of data sources

User Interface

The interface of the application is pretty straight forward. The image below shows the user interface of the Data Administration Module.


Data Administrator

Data Administration

Reset interval

Name: patstat Collection: patent
Name: uspto Collection: patent
Name: mag Collection: publication
Name: springer Collection: publication

Name	patstat
Description	PATSTAT database
URL	https://www.epo.org/sear
Category type	patent
Licence type	Subscription
Date last updated	2019-04-28
Update interval	7
Scheme file	data/patstat/scheme.json
Mapping file	data/patstat/mapping.jso
Licence file	

Load to collection

Create Indexes

Log Clear

Status:

List view

The main part of the appplication consists of a list view, containing a list of data sources metadata present in the Data Sources DB. Aboe the list view, there is a **refresh button** which refreshes the list of data sources from the database and a **Refresh interval** button which resets the update interval for the selected list item.

Details view

The right side of the application contains the **detail view** for the selected item in the listview. It contains all the **metadata** stored for each data source. The details information about a data source can be edited using the *Edit* button and deleted using the *Delete* button. If the selected data source contains the implementation of a data loader, it is possible to load it to the database. This is done using the *Load to collection* button which attempts to load files from the file system path and **insert them to the destination collection**, specified in the ***category type* field**.

Indexes for each data source are created automatically, however it is possible to trigger the indexing manually by clicking on the *Create Indexes* button.

The following is the list of all available metadata for each data source:

- ***Name*** **–** This is a mandatory field, it serves as an identification name for the data source
- ***Description*** **–** Provides a brief description of the data source
- ***URL*** **–** Contains the source URL, where the data source comes from
- ***Category type*** **–** Specifies the type of data (patents, publications, etc.)
- ***Licence type*** **–** Specifies the type of licence (FREE, Subscription-based, Paid)
- ***Date last updated*** **–** The date the data source was last updated
- ***Scheme file*** – Points to the file with the structure of the particular data source
- ***Mapping file*** **–** Mandatory field pointing to the mapping file for the data source
- ***Licence file*** – Pooints to the file with licence, if exists

Each data field also contains information about its **validity**, specified by the ***update interval field***. The update date can be updated using the ***reset interval button***. The bottom of the page contains utility tools for bulk **zip extraction** of files from a directory and **bulk conversion** of files from a source directory to the target directory. At the very bottom, there is a label indicating the status of current operation.

Logs view

The bottom right corner displays **log messages** from the application. The bottom panel shows the current status of the application.

Project co-funded by the European Union (ERDF, IPA, ENI)
http://www.interreg-danube.eu/knowing-ipr

Data Structure

There needs to be a common set of fields, which will be present in each data source and each data source will 'map' its own structure to this common one.
The common structure for patent data contains the following fields:

- **Title**
- **Abstract**
- **Patent number**
- **Year granted**
- **Date**
- **List of authors**
- **List of owners**
- **Language**
- **Country**
- **Status**

The common structure for publication data:

- **Title**
- **Abstract**
- **Date**
- **List of authors**
- **Language**
- **Country**
- **Publisher**
- **Venue**
- **Keywords**
- **DOI**
- **Issue**
- **Fields of study**

Sources database module

This document should provide an overview of the process of **searching** relevant data on the Sources database. The main **strategies** are presented as well as **more advanced methods** that were used during the searching process.

Strategies

This section describes the strategies used when performing the searches on the target Sources database.

The process of query processing consists of several stages:

- Submit query
- Send query to the REST endpoint
- Accept query at Knowledge Generation Core
- Analyze and preprocess the query
- Check the cache for query
- If the query is found in the cache:
    1. Retrieve report from the cache
    2. Serialize report and send to the client
- If the query is not found in the cache:
    1. Create a simple query to the MongoDB or ElasticSearch database
    2. If results are found:
        1. Create and cache the new report
    3. If results are not found and MongoDB is used:
        1. Run extended query
        2. Create and cache the new report
- Serialize report and send back to the client

The image on Figure 1 Flow chart of the query processing shows the **flow of events** from the point the query is submitted to the server until the results are returned back to the client. This flow of events is valid if the MongoDB database is used for searching. In case the ElasticSearch is used, only a single query is run.

Project co-funded by the European Union (ERDF, IPA, ENI)
http://www.interreg-danube.eu/knowing-ipr

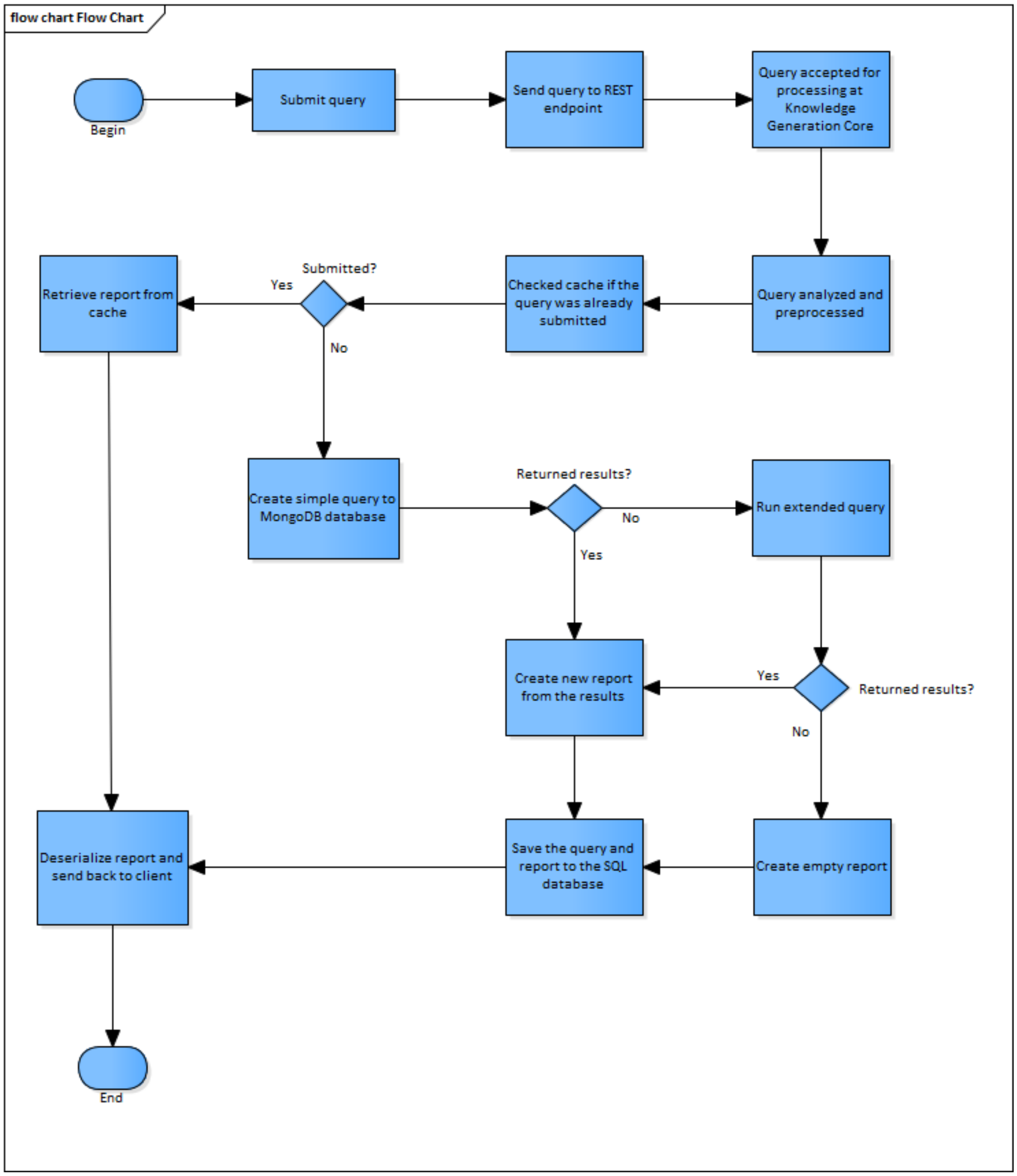

flow chart Flow Chart
Begin
Submit query
Send query to REST endpoint
Query accepted for processing at Knowledge Generation Core
Query analyzed and preprocessed
Checked cache if the query was already submitted
Submitted?
Yes
No
Retrieve report from cache
Create simple query to MongoDB database
Returned results?
No
Yes
Run extended query
Returned results?
Yes
No
Create new report from the results
Create empty report
Save the query and report to the SQL database
Deserialize report and send back to client
End

Figure 1 Flow chart of the query processing

Project co-funded by the European Union (ERDF, IPA, ENI)
http://www.interreg-danube.eu/knowing-ipr

MongoDB Query Creation

The creation of the query to the MongoDB database consists of **two stages**, the first one being a quick query, followed by an extended query, running a more thorough search.

- **Quick query**

  The quick query is performed at the start of each search request. It is limited to only a few seconds of execution and the method for searching is based only on exact match. If the quick query does not return any results by the end, the extended query is performed.

- **Extended query**

  Is not bound by a fixed timeout, the user however can specify after how much time the query searching should terminate.

Query Analysis

To retrieve enhanced results to the user's queries, an analysis of the query is performed. An implementation of the **WordNet** lexical database is being used. The program extracts words from the query and finds similar words, hypernyms, and hyponyms from the WordNet hierarchy tree. This enables us to **enhance the results** by not only searching for the specified terms but also words of similar meanings.

The following **example** illustrates the usage of the WordNet Lexical database:

The user searches for the expression "*fast car*". The query is analyzed and words "*fast*" and "*car*" are replaced by a sequence of similar words (synonyms, hypernyms, hyponyms…)

The resulting query run on the database will look something like this:

"*fast agile brisk rapid car vehicle auto automobile*". Between tokens, there is an OR operator, meaning any of the tokens need to be present in order for the document to match.

ElasticSearch

ElasticSearch is a highly scalable open-source full-text search and analytics engine. It allows for storing, searching, and analyzing big volumes of data quickly and in near real time. It is generally used as the underlying engine/technology that powers applications that have complex search features and requirements.

The majority of user queries is performed using ElasticSearch. By default, we use the implementation of *simple_query_string* for searching. The simple query string supports the following operators:

- + signifies AND operation
- | signifies OR operation
- - negates a single token
- " wraps a number of tokens to signify a phrase for searching
- * at the end of a term signifies a prefix query
- (and) signify precedence
- ~N after a word signifies edit distance (fuzziness)
- ~N after a phrase signifies slop amount

Data synchronization

ElasticSearch (ES) is used as a complementary database to MongoDB. In order to be able to search using ES, we needed to implement a synchronization mechanism that would allow to transfer the data from MongoDB to ES. The final solution uses a third party library Monstache which allows for real time data synchronization from MongoDB to ES. However, it was not necessary to synchronize whole documents, as it would only increase the storage size occupied by the data sources as well as slowing down the overall search performance. Therefore while MongoDB stores whole documents, in ES we only index a subset of fields present in MongoDB. These fields are called *filters,* as they are used for filtering by users.

The following list shows the fields synchronized from MongoDB:

For patents, the common fields extracted from available data sources: (The fields after the dash in italics are the exact field names which should be used for filtering)

- Patent number – *number*
- Title - *title*
- Abstract - *abstract*
- Date granted - *date*
- Authors - *authors.name*
- Owners - *owners.name*

- Country - *country*
- Language - *lang*
- Status - *status*

For publications, we extracted the following fields:

- Title - *title*
- Abstract (Microsoft Academic Graph does not have them) - *abstract*
- Year (date) - *date*
- Authors - *authors.name*
- Publisher - *publisher*
- Fields of study - *fos*
- Keywords - *keywords*
- Issue - *issue*
- URL - *url*
- DOI - *doi*
- Venue - *venue*
- Language - *lang*

This means that the intersection between patent and publication data produces the following common fields:

- Title
- Abstract (Again, not all documents have it)
- Date
- Authors
- Language

Category System

Sometimes users do not want to search for anything specific, but rather browse through all the patents in a certain category. For these purposes, the server implements a category system which enables users to interact with patents in specific categories. A category tree was implemented, storing various categories in a **tree structure**. The client can request to print the whole tree or a subtree of a specific category. The output can be either in **plaintext** or in **JSON** format.

Category endpoints

The server provides several **endpoints** methods to interact with the category system. The exact specifications of the endpoints are described in the document *REST API Specification.*

The most important endpoint methods are the following:

- *\/category/tree/plaintext* - Prints the whole category tree in plaintext
- */category/tree/json* - Prints the whole category tree in JSON format
- */category/tree/json?name=Wheels* - Prints category subtree from the specified category name (Wheels)
- */category/get/rims/1* - Returns documents associated with the specified category (rims)

Confirmed results

The process of retrieving the documents in a certain category is **automated**, meaning it is not necessary to manually categorize each patent to a category, but it is done automatically. However, to increase the accuracy of the results, it is possible to manually mark patents as belonging to a certain category. These patents will then be placed above others in the list of results. Patents with the manually assigned category are called ***confirmed results***.

The URLs of confirmed results are stored in the database, ensuring they can be retrieved as fast as possible. The whole process of retrieving patents for a certain category looks like this:

- The user submits a request to the endpoint (e.g. */category/get/rims/1)* on the server.
- The server validates if the name of the category is valid

- If the category is valid, the *Knowledge database* is queried and a list of confirmed results for the category is fetched.
- If there are enough results to fill the page, the results can be returned immediately.
- If the number of results is less than the size of the page, the rest is fetched directly from the *Sources database.*
- Next, duplicate records are found and removed.
- Finally, the confirmed results are put to the front of the list.

The Figure 2 shows the process of retrieving existing and storing new confirmed results.

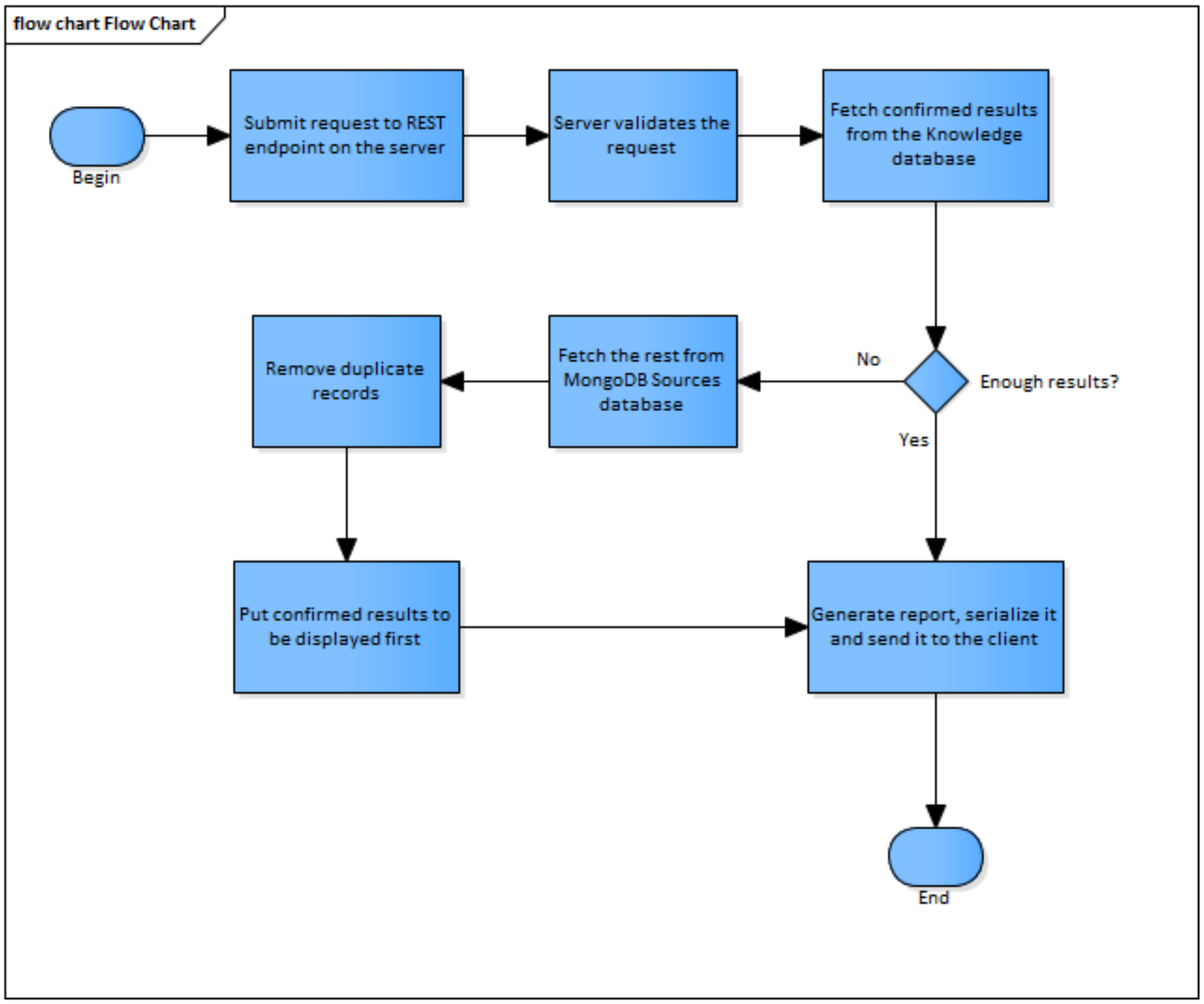

flow chart Flow Chart
Begin
Submit request to REST endpoint on the server
Server validates the request
Fetch confirmed results from the Knowledge database
Enough results?
No
Yes
Fetch the rest from MongoDB Sources database
Remove duplicate records
Put confirmed results to be displayed first
Generate report, serialize it and send it to the client
End

Figure 2 Flow chart of retrieving and storing confirmed results

Knowledge database model

The Knowledge database is a relational database running on MariaDB. It is a complementary database to the Sources database. Apart from containing metadata about the acquired data sources, it also provides a cache mechanism to speed up the performance of retrieving common queries by storing previously run queries along with the returned results.

Use Cases

The Knowledge Database (KD) represents a temporary data storage for queries frequently asked by users.

The main use cases are the following:

- Storing metadata about data sources present in the Sources database
- Storing common queries and responses to them
- Keeping relationships between generated reports and user queries

It is a repository where structured data about searched queries is being stored after knowledge engineering methods are applied to the data retrieved from the data sources. Therefore, the main advantage of the *Knowledge Database* is the ability to speed up the retrieval of results to the user. This is especially noticeable in queries that run longer. The *Knowledge Database* is also used to store metadata about the acquired data sources. The metadata contain fields like name of the data source, category of data, source URL, type of licence etc.

Context

The integration of the *Knowledge Database* plays a major role in the process of searching the data. The following is an example of when the *Knowledge Database* can be used in practice:

- User sends query to the *Knowledge Generation Core* application's endpoint
- Query is analyzed and preprocessed
- *Knowledge Database* does a check if query already present
    - If the query is found in the cache:
        - Retrieve report from the cache directly from *Knowledge Database*

Project co-funded by the European Union (ERDF, IPA, ENI)
http://www.interreg-danube.eu/knowing-ipr

    - If the query is not found in the cache:
        - Create a query to the ElasticSearch database
        - Create and cache the new report to the *Knowledge Database*
- Serialize report and send back to the client

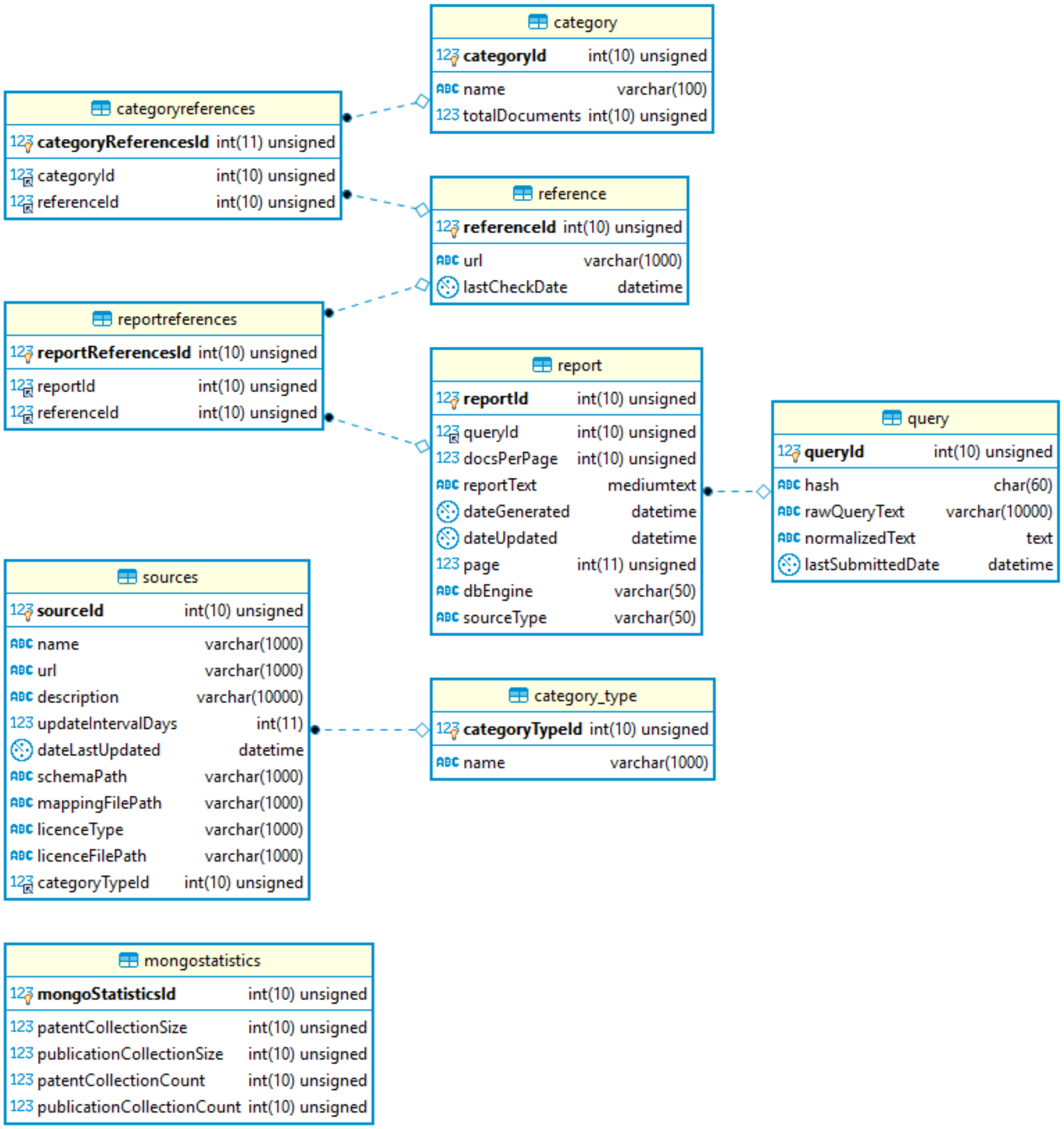
Technologies Used

The *Knowledge Database* is a relational SQL database which runs on MariaDB. The *Knowledge Generation Core* application and the *Data Administration Module* use MariaDB JDBC driver in version 2.4.0 to connect to the *Knowledge Database*.

Model

The following image shows the entity relationship model of the Knowledge database. As of now, the *Knowledge Database* consists of 9 tables, mainly concerning about storing user queries and reports generated by the *Reports Generation Module.*

Project co-funded by the European Union (ERDF, IPA, ENI)
http://www.interreg-danube.eu/knowing-ipr


category
categoryId int(10) unsigned
name varchar(100)
totalDocuments int(10) unsigned
categoryreferences
categoryReferencesId int(11) unsigned
categoryId int(10) unsigned
referenceId int(10) unsigned
reference
referenceId int(10) unsigned
url varchar(1000)
lastCheckDate datetime
reportreferences
reportReferencesId int(10) unsigned
reportId int(10) unsigned
referenceId int(10) unsigned
report
reportId int(10) unsigned
queryId int(10) unsigned
docsPerPage int(10) unsigned
reportText mediumtext
dateGenerated datetime
dateUpdated datetime
page int(11) unsigned
dbEngine varchar(50)
sourceType varchar(50)
query
queryId int(10) unsigned
hash char(60)
rawQueryText varchar(10000)
normalizedText text
lastSubmittedDate datetime
sources
sourceId int(10) unsigned
name varchar(1000)
url varchar(1000)
description varchar(10000)
updateIntervalDays int(11)
dateLastUpdated datetime
schemaPath varchar(1000)
mappingFilePath varchar(1000)
licenceType varchar(1000)
licenceFilePath varchar(1000)
categoryTypeId int(10) unsigned
category_type
categoryTypeId int(10) unsigned
name varchar(1000)
mongostatistics
mongoStatisticsId int(10) unsigned
patentCollectionSize int(10) unsigned
publicationCollectionSize int(10) unsigned
patentCollectionCount int(10) unsigned
publicationCollectionCount int(10) unsigned

Project co-funded by the European Union (ERDF, IPA, ENI)
http://www.interreg-danube.eu/knowing-ipr

Category table

The Category table contains available categories for searching with the number of documents present for this category.

COLUMN_NAME	IS_NULLABLE	COLUMN_COMMENT	COLUMN_TYPE	CHARACTER_MAXIMUM_LENGTH	COLUMN_KEY
categoryId	NO		int(10) unsigned	\N	PRI
name	NO	Name of the category.	varchar(100)	100	
totalDocuments	NO	Temporary total number of documents assigned to this category.	int(10) unsigned	\N	

Category references table

Associates categories with their respective referenced documents which fall under that category.

COLUMN_NAME	IS_NULLABLE	COLUMN_COMMENT	COLUMN_TYPE	CHARACTER_MAXIMUM_LENGTH	COLUMN_KEY
categoryReferencesId	NO		int(11) unsigned	\N	PRI
categoryId	NO		int(11) unsigned	\N	
referenceId	NO		int(11) unsigned	\N	

Category type table

A code table for different categories of data sources. The options can be *publications, patents, trademarks* etc.

COLUMN_NAME	IS_NULLABLE	COLUMN_COMMENT	COLUMN_TYPE	CHARACTER_MAXIMUM_LENGTH	COLUMN_KEY
categoryTypeId	NO		int(10) unsigned	\N	PRI
name	NO	Name of the data source category. Patent or publication are valid examples of data source categories.	varchar(1000)	1000	

Mongo statistics table

Contains statistics specific to MongoDB, like the sizes of different collections etc.

COLUMN_NAME	IS_NULLABLE	COLUMN_COMMENT	COLUMN_TYPE	CHARACTER_MAXIMUM_LENGTH	COLUMN_KEY
mongoStatisticsId	NO		int(10) unsigned	\N	PRI
patentCollectionSize	NO	The size of the patent collection in bytes.	int(10) unsigned	\N	
publicationCollectionSize	NO	The size of the publication collection in bytes	int(10) unsigned	\N	
patentCollectionCount	NO	Number of records in the patent collection.	int(10) unsigned	\N	

publicationCollectionCount	NO	Number of records in the publication collection.	int(10) unsigned	\N	

Query table

This *Query* table contains data about user queries. The stored queries were previously run on the Sources database and were cached for faster later access. Duplicate checks are performed by the *rawQueryText* field.

COLUMN_NAME	IS_NULLABLE	COLUMN_COMMENT	COLUMN_TYPE	CHARACTER_MAXIMUM_LENGTH	COLUMN_KEY
queryId	NO	Unique key of a query	int(10) unsigned	\N	PRI
hash	NO	The hashed value of the query string	char(60)	60	
rawQueryText	NO	Raw query text	varchar(10000)	10000	
normalizedText	YES	The normalized text of the query	text	65535	
lastSubmittedDate	YES	The date of last query submission	datetime	\N	

Reference table

Represents a document by its URL. In MongoDB, the unique URL is the document'd identification number.

COLUMN_NAME	IS_NULLABLE	COLUMN_COMMENT	COLUMN_TYPE	CHARACTER_MAXIMUM_LENGTH	COLUMN_KEY
referenceId	NO		int(10) unsigned	\N	PRI
url	NO	The URL to the reference	varchar(1000)	1000	
lastCheckDate	YES		datetime	\N	

Report table

This table contains the actual report generated from the *Report Generation Module.* Each report is associated with a query. The Knowledge Generation Core application first attempts to retrieve data from this table upon receiving a user request query. If the desired query report is not found in the *Knowledge Database (KD),* a new query is constructed to the *Sources Database* and the result is saved to the *KD.*

COLUMN_NAME	IS_NULLABLE	COLUMN_COMMENT	COLUMN_TYPE	CHARACTER_MAXIMUM_LENGTH	COLUMN_KEY
reportId	NO		int(10) unsigned	\N	PRI
queryId	NO		int(10) unsigned	\N	MUL
docsPerPage	NO	Number indicating the number of results per page	int(10) unsigned	\N	
reportText	NO	The actual text of the report. Contains documents data with other metadata about the search.	mediumtext	16777215	
dateGenerated	NO	Date the report was first generated.	datetime	\N	
dateUpdated	NO	Date the report was updated.	datetime	\N	
page	NO	Number indicating the page of results contained in the report	int(11) unsigned	\N	
dbEngine	NO	The database engine used to generate the report, it can be either MongoDB or ElasticSearch. Each search engine uses slightly different format of the reports.	varchar(50)	50	

sourceType	NO	The type of data the report was generated from. It can be for example report from USPTO data, Microsoft Academic Graph data or a report from ALL data available.	varchar(50)	50	

Report references table

Contains a list of associations between reports and the target document references.

COLUMN_NAME	IS_NULLABLE	COLUMN_COMMENT	COLUMN_TYPE	CHARACTER_MAXIMUM_LENGTH	COLUMN_KEY
reportReferencesId	NO		int(10) unsigned	\N	PRI
reportId	NO	The ID of the report.	int(10) unsigned	\N	MUL
referenceId	NO	The ID of the document reference.	int(10) unsigned	\N	MUL

Sources table

Each data source contains various information about its origin and about its format and structure. The *Sources* table contains various metadata about each of the data source.

COLUMN_NAME	IS_NULLABLE	COLUMN_COMMENT	COLUMN_TYPE	CHARACTER_MAXIMUM_LENGTH	COLUMN_KEY
sourceId	NO		int(10) unsigned	\N	PRI
name	NO	The name of the data source.	varchar(1000)	1000	
url	NO	The URL where the updated data for download can be found.	varchar(1000)	1000	

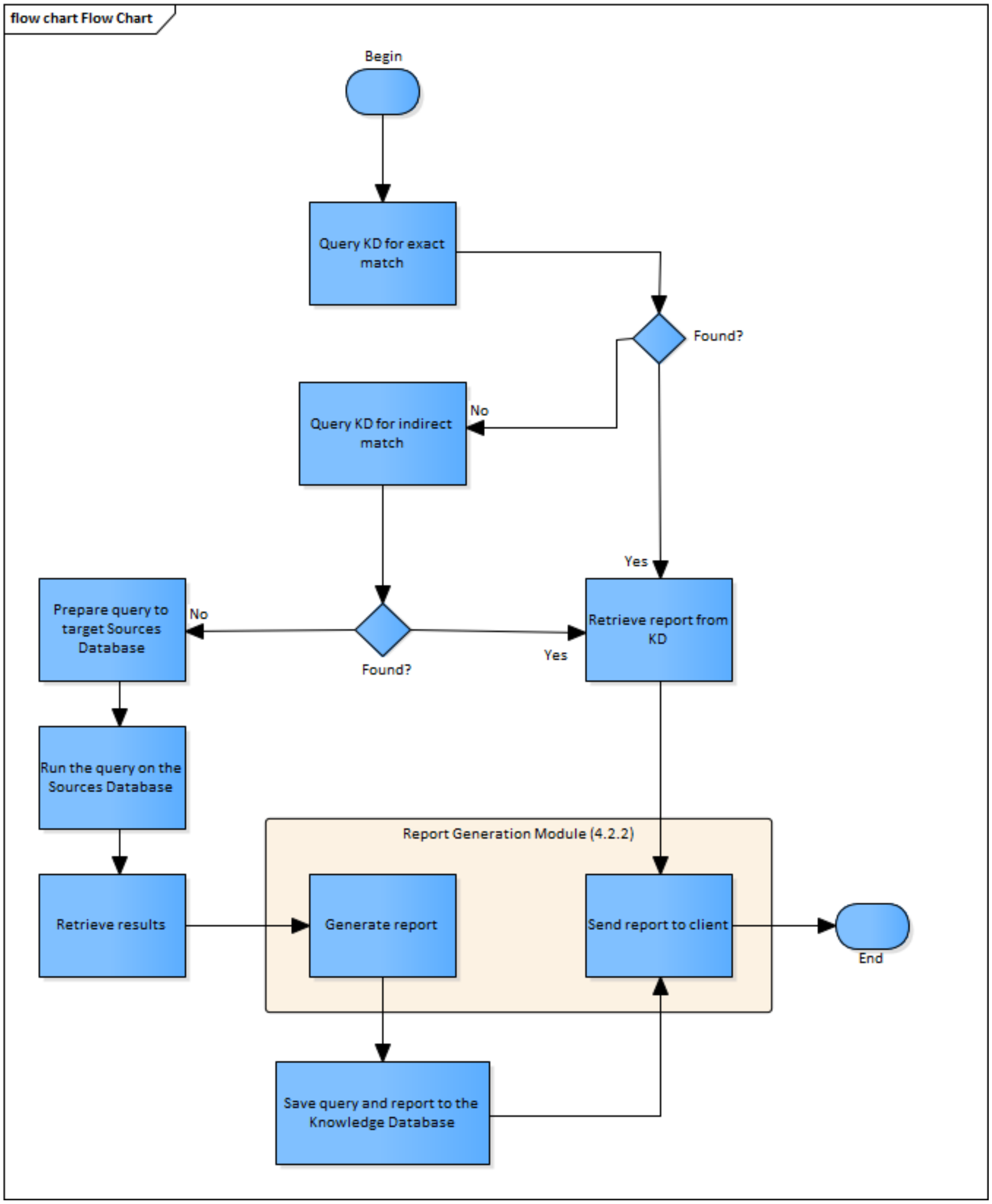
Project co-funded by the European Union (ERDF, IPA, ENI)
http://www.interreg-danube.eu/knowing-ipr

description	YES	The description of the data source	varchar(10000)	10000	
updateIntervalDays	YES	The update interval in days. The user is notified when it is time to update.	int(11)	\N	
dateLastUpdated	NO	Date of the last update.	datetime	\N	
schemaPath	NO	Path to the schema for the data source.	varchar(1000)	1000	
mappingFilePath	NO	Path to the mappings file for the data source	varchar(1000)	1000	
licenceType	YES	Type of licence associated with the data source	varchar(1000)	1000	
licenceFilePath	YES	Path to the licence file.	varchar(1000)	1000	
categoryTypeId	NO	The type of category assigned to this data source. It can be patents, publication, trademarks etc.	int(10) unsigned	\N	MUL

Examples

The following examples demonstrate the usage of Knowledge Database in the contect of the whole system. The most common scenario is when a user is searching for some specific documents. The user specifies a query which is then sent to the REST API from the client web application.

The flow of events can be displayed using the following flow chart.


flow chart Flow Chart
Begin
Query KD for exact match
Found?
No
Query KD for indirect match
Yes
Prepare query to target Sources Database
No
Found?
Yes
Retrieve report from KD
Run the query on the Sources Database
Report Generation Module (4.2.2)
Retrieve results
Generate report
Send report to client
End
Save query and report to the Knowledge Database

Slika 1 The flow of events when using the Knowledge Database

The following are examples of how the Knowledge Database can aid in retrieving the desired data.

For illustration, let's say the user is searching for »electric cars in Slovenia«.

Example 1 – The query is found exactly in the Knowledge Database

The first scenario which can happen is when the user (or multiple users) submit the same query multiple times. This scenario assumes the searched texts from the queries are exactly the same.

- The query is passed to the *Knowledge Generation Core*
- Knowledge database is queried for the exact match
- **The exact match query is found – Contains the exact search text »electric cars in Slovenia«**
- The report is identified and retrieved for that query
- The Report Generation Module appends metadata about the search
- The report is returned to the user in an unchanged way

Example 2 – The query is found indirectly in the Knowledge Database

- The query is passed to the *Knowledge Generation Core*
- Knowledge database is queried for the exact match
- The exact match is NOT found
- **Knowledge database is queried for the indirect match**
  - The keywords of the query are analysed using WordNet dictionary
  - The keywords are replaced with a combination of synonymous, hypernymous and hyponymous words – For example, the text query »electric cars in Slovenia« can be edited to: »electric cars OR vehicles OR BMW OR Mercedes in Slovenia«
- The indirect match query is found
- The report is identified and retrieved for that query
- The Report Generation Module appends metadata about the search
- The report is returned to the user in an unchanged way

Example 3 – The query is not found in the Knowledge Database

- The query is passed to the *Knowledge Generation Core*
- Knowledge database is queried for the exact match
- The exact match is NOT found
- Knowledge database is queried for the indirect match
- The indirect match is NOT found
- **The query to target Sources database is constructed**
- The query is run on the target database
- The results are retrieved

Project co-funded by the European Union (ERDF, IPA, ENI)
http://www.interreg-danube.eu/knowing-ipr

- **The *Report Generation Module* generates a new report from the retrieved results**
- The generated report along with the original query is saved to *Knowledge Database* for later retrieval
- The Report Generation Module appends metadata about the search
- The report is returned to the user

Project co-funded by the European Union (ERDF, IPA, ENI)
http://www.interreg-danube.eu/knowing-ipr