Text search in i2b2

1.6.02

The i2b2 metadata allows a text search of the observation_fact table. Two columns may be searched, the TVal_Char column, and the observation_blob column (coming soon). The search is initiated by pulling concepts from the "Navigate Terms" or "Find Terms" in the Vocabulary Areas of the i2b2 Client for which you want to search for values in their associated text. There are several strategies undertaken while loading the data where values may have been included that one wants to search, such as:

- A lab test may have associated text that can be searched. For example, a lumbar puncture may include a description of white cell appearance and one wants to search for "multinucleated".

- A diagnosis may have a text name as well as the concept_cd code. The code will go into concept_cd, but if text as a description of the diagnosis is available, this can go into a text field and therefore allow it to be searched. Note that by dragging the general term "diagnosis" into a query tool panel, one could search this text field for all diagnoses.

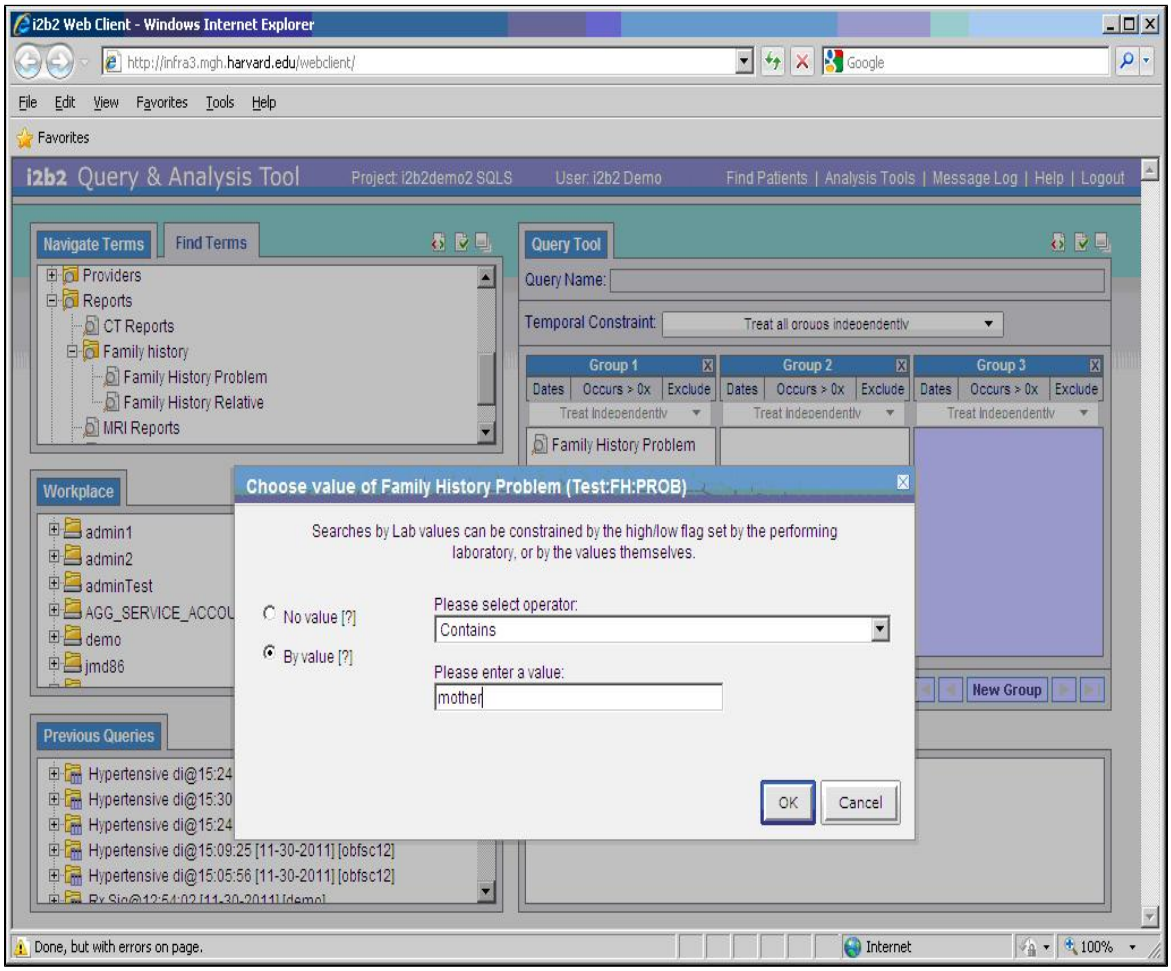
- A radiology report can be placed into the blob field (note that it would need to be de-identified or the column would need to be database-encrypted to prevent PHI from possibly being exposed). One could then search for text in the report, such as "fracture" or "erosion".

The use of the TVal_Char field is straightforward. In the observation_fact table, the data is imported into a row and the text value is placed into the TVal_Char column and the letter "T" is placed in the ValueType column. The limitation of the TVal_Char filed is that it can only have the number of characters that the VARCHAR field allows. By default it is 255, and can be changed to up VARCHAR(MAX) in SQL SERVER to make it up to 2 billion + characters, and changed to VARCHAR(2000) in ORACLE to make it up to 2000 characters.

The use of the observation_blob field enables more complex text searches if supported by the database implementation. The text value is placed into the observation_blob column. For the row of the fact table whose observation_blob field contains the text, perhaps a vist note or a radiology report, the letter "B" is placed in the ValueType, and the MIME type of the text value is (optionally) placed in the TVal_Char column, the default type is "plain/text" if null or blank. The number of bytes is (optionally) placed in the NVal_num column, the default is the database column size limit if not present.

Once the database has the entries above, one must set up the metadata in the ontology tables to drive queries on the text fields. Value query boxes are driven by the contents in the c_metadataxml column. The XML has two relevant tags that need to be set. The first is the <DataType> tag which should be set to STRING if the text value is placed in the TVal_Char column, and LARGESTRING if the text value is placed in the observation_blob column (coming soon). The second relevant tag is the <MaxStringLength> tag which should include the maximum length of a string to be queried for, usually the length of the datatype.

When the term is dragged, the value selection box will be presented to the user, the STRING tag will cause the following box to be presented:



Note that the "Family History Problem" is a concept code that has a value, which is text in the TVal_Char column, and the value will be searched for the word/phrase "mother". The STRING query box allows the following options for how the string value will be searched:

Begins with: Search for the word at the beginning of the string value.

Ends with: Search for the word at the end of the string value.

Contains: Search for the word somewhere within the string value.

Exact: Search for an exact match to the string value.

The Querytool packages the request into XML, which is specific for each item dragged into the query interface. The specification of the XML that defines a STRING value looks like:

Text value constrain options:

Text constrain options	Sql
LIKE[exact] Case insensitive search. <constrain_by_value> <value_type>TEXT</value_type> <value_operator> LIKE[exact]</value_operator> <value_constraint> NEG </value_constraint> </constrain_by_value>	Select * from observation_fact where Concept_cd in (select concept_Cd from concept_dimension where ..) and valtype_cd = 'T' and upper(tval_char) = upper('NEG')

LIKE[begin] Begin will be the default option if the optional tag is not present. This will support backward compatibility. <constrain_by_value> <value_type>TEXT</value_type> <value_operator>LIKE[begin]</value_operator> <value_constraint> NEG </value_constraint> </constrain_by_value>	Select * from observation_fact where Concept_cd in (select concept_Cd from concept_dimension where ..) and valtype_cd = 'T' and upper(tval_char) LIKE upper('NEG%')
LIKE[end] <constrain_by_value> <value_type>TEXT</value_type> <value_operator>LIKE[end]</value_operator> <value_constraint> NEG </value_constraint> </constrain_by_value>	Select * from observation_fact where Concept_cd in (select concept_Cd from concept_dimension where ..) and valtype_cd = 'T' and tval_char LIKE '%NEG'
LIKE[contains] <constrain_by_value> <value_type>TEXT</value_type> <value_operator>LIKE[contains] </value_operator> <value_constraint>NEG </value_constraint> </constrain_by_value>	Select * from observation_fact where concept_cd in (select concept_Cd from concept_dimension where ..) and valtype_cd = 'T' and tval_char LIKE '%NEG%'