Informatics for Integrating Biology and the Bedside

i2b2 Design Document

Data Repository (CRC) Cell

Document Version: *1.5.2*

I2b2 Software Release: *1.5*

Table of Contents

1. INTRODUCTION

The Data Repository Cell (also called the Clinical Research Chart, or CRC), is designed to hold data from clinical trials, medical record systems and laboratory systems, along with many other types of clinical data from heterogeneous sources. The CRC stores this data in the following tables:

Data Tables

1. Patient
2. Visit
3. Observation

Lookup Tables

1. Concept
2. Provider
3. Code

Mapping Tables

1. Patient mapping
2. Visit mapping

The three data tables, along with two of the lookup tables (concept and provider) make up the ***star schema*** of the warehouse. The code table is strictly a lookup table and is not part of the star schema. All of the tables that are part of the CRC are described in this document.

2. I2B2 DATA MART

The i2b2 data mart is a data warehouse modeled on the star schema structure first proposed by Ralph Kimball. The database schema looks like a star, with one central fact table surrounded by one or more dimension tables. The most important concept regarding the construction of a star schema is identifying what constitutes a fact.

In healthcare, a logical fact is an observation on a patient. It is important to note that an observation may not represent the onset or date of the condition or event being described, but instead is simply a recording or a notation of something. For example, the observation of 'diabetes' recorded in the database as a 'fact' at a particular time does not mean that the condition of diabetes began exactly at that time, only that a diagnosis was recorded at that time (there may be many diagnoses of diabetes for this patient over time).

The fact table contains the basic attributes about the observation, such as the patient and provider numbers, a concept code for the concept observed, a start and end date, and other parameters described below in this document. In i2b2, the fact table is called observation_fact.

Dimension tables contain further descriptive and analytical information about attributes in the fact table. A dimension table may contain information about how certain data is organized, such as a hierarchy that can be used to categorize or summarize the data. In the i2b2 data mart, there are four dimension tables that provide additional information about fields in the fact table:

1. patient_dimension
2. concept_dimension
3. visit_dimension
4. provider_dimension

3. I2B2 DATA MART TABLES

3.1 General Information

The **observation_fact** table has only required columns. The **patient_dimension** and **visit_dimension** tables have both required and optional columns. All the tables have the following five technically-oriented or administrative columns:

Column Name	Data Type	Nullable	Definition
update_date	datetime	Yes	Date the row was updated by the source system (date is obtained from the source system)
download_date	datetime	Yes	Date the data was downloaded from the source system
import_date	datetime	Yes	Date data was imported into the CRC
sourcesystem_cd	varchar(50)	Yes	Coded value for the data source system
upload_id	decimal(38,0)	Yes	A numeric id given to the upload

3.2 Observation_Fact

The **observation_fact table** is the fact table of the i2b2 star schema and represents the intersection of the dimension tables. Each row describes one observation about a patient made during a visit. Most queries in the i2b2 database require joining the observation_fact table with one or more dimension tables together.

observation_fact		
PK	**encounter_num**	**int**
PK	**concept_cd**	**varchar(50)**
PK	**provider_id**	**varchar(50)**
PK	**start_date**	**datetime**
	patient_num	**int**
	modifier_cd	varchar(50)
	instance_num	int
	valType_cd	varchar(50)
	tval_char	varchar(255)
	nval_num	decimal(18,5)

observation_fact		
	valueFlag_cd	varchar(50)
	quantity_num	decimal(18,5)
	units_cd	varchar(50)
	end_date	datetime
	location_cd	varchar(50)
	observation_blob	text
	confidence_num	decimal(18,5)
	update_date	datetime
	download_date	datetime
	import_date	datetime
	sourcesystem_cd	varchar(50)
	upload_id	int

Observation_Fact			
Key	**Column Name**	**Column Definition**	**Nullable? (Default = YES)**
PK	**encounter_num**	Encoded i2b2 patient visit number	NO
	patient_num	Encoded i2b2 patient number	NO
PK	**concept_cd**	Code for observation of interest (i.e. diagnoses, procedures, medications, lab test)	NO
PK	**provider_id**	Practitioner id or provider id	NO
PK	**start_date**	Starting date-time of observation *(mm/dd/yyyy)*	NO
	modifier_cd	Code for modifier of interest (i.e. “ROUTE”,” DOSE”), note that value columns are often used to hold the amounts such as “100” (mg) or “PO”	YES
	instance_num	Encoded instance number that allows more that one modifier to be provided for each concept_cd. Each row will have a different modifier_cd but a similar instance_num.	YES
	valType_cd	Format of the concept *N = Numeric* *T = Text (enums/short messages)*	

		B = Raw Text (notes/reports) *NLP = NLP result text*	
	tval_char	Used in conjunction with valType_cd = “T” or “N” When valtType_cd = “T” *Stores the text value* When valType_cd = “N” *E = Equals* *NE = Not equal* *L = Less Than* *LE = Less than and Equal to* *G = Greater Than* *GE = Greater than and Equal to*	
	nval_num	Used in conjunction with valType_cd = “N” to store a numerical value	
	valueFlag_cd	Used in conjunction with valType_cd = “B”, “NLP”, “N”, or “T” When valType_cd = “B” or “NLP” it is used to indicate whether or not the blob field is encrypted *X = Encrypted text in blob field* When valType_cd = “N” or “T” it is used to flag certain outlying or abnormal values *H = High* *L = Low* *A = Abnormal*	
	quantity_num	Quantity of nval	
	units_cd	Units of measurement of nval	
	end_date	The end date-time for the observation	
	location_cd	A location code, such as for a clinic	
	confidence_num	Assessment of accuracy of data	
	observation_blob	Holds any raw or miscellaneous data that exists, often encrypted PHI	
	update_date	As defined in the above section (“*General Information*”)	
	download_date	As defined in the above section (“*General*	

		Information")	
	import_date	As defined in the above section ("*General Information*")	
	sourcesystem_cd	As defined in the above section ("*General Information*")	
	upload_id	As defined in the above section ("*General Information*")	

3.2.1 Value Columns

The **observation_fact table** has six columns associated with values. This section contains additional information about each of these columns that contain value related data.

Observation_Fact Values Columns					
valType_cd	**tval_char**	**nval_num**	**valueFlag_cd**	**units_cd**	**obs_blob**
N	E (equal), NE (not equal) L (less than) LE (less than and equal) G (greater than) GE (greater than and equal)	Actual numeric value of object	H (high) L (low) N (normal) [null] (unknown)	Units associated with object	Misc. encrypted information
T	Actual short text value of object	N/A	A (abnormal) N (normal) [null] (unknown)	Units associated with object	Misc. encrypted information
B	N/A	N/A	X (encrypted)	N/A	Raw text
NLP	N/A	N/A	X (encrypted)	N/A	NLP result XML

Here is an example of how the Numeric/Text/Flag value constrains used in the queries:

Value Constrain by Number:

Query Numeric Value Constrain	Numeric Constrain SQL
Greater than: <constrain_by_value> <value_operator>GT</value_operator> <value_constraint>99.9</value_constraint> <value_type>NUMBER</value_type> </constrain_by_value>	(valtype_cd = 'N' AND nval_num > 99.9 AND tval_char IN ('GE','E')) OR (valtype_cd = 'N' AND nval_num >= 99.9 AND tval_char = 'G')
Less than: <constrain_by_value> <value_operator>LT</value_operator> <value_constraint>99.9</value_constraint> <value_type>NUMBER</value_type> </constrain_by_value>	((valtype_cd = 'N' AND nval_num < 99.9 AND tval_char IN ('LE','E')) OR (valtype_cd = 'N' AND nval_num <= 99.9 AND tval_char = 'L'))
Between : <constrain_by_value> <value_operator>BETWEEN</value_operator> <value_constraint>1 and 100 </value_constraint> <value_type>NUMBER</value_type> </constrain_by_value>	valtype_cd = 'N' AND nval_num BETWEEN 1 and 100 AND tval_char = 'E'
Equal to : <constrain_by_value> <value_operator>EQ</value_operator> <value_constraint>99.9</value_constraint> <value_type>NUMBER</value_type> </constrain_by_value>	valtype_cd = 'N' AND nval_num = 99.9 AND tval_char = 'E'
Less than and Equal to: <constrain_by_value> <value_operator>LE</value_operator> <value_constraint>99.9</value_constraint> <value_type>NUMBER</value_type> </constrain_by_value>	valtype_cd = 'N' AND nval_num <= 99.9 AND tval_char IN ('L','E','LE')

Greater than and Equal to : <constrain_by_value> <value_operator>GE</value_operator> <value_constraint>99.9</value_constraint> <value_type>NUMBER</value_type> </constrain_by_value>	valtype_cd = 'N' AND nval_num >= 99.99 AND tval_char IN ('G','E','GE')
Not Equal : <constrain_by_value> <value_operator>NE</value_operator> <value_constraint>99.9</value_constraint> <value_type>NUMBER</value_type> </constrain_by_value>	(valtype_cd ='N' AND nval_num <> 99.9 AND tval_char <> 'NE') OR (valtype_cd = 'N' AND nval_num = 99.9 AND tval_char = 'NE')

Value Constrain by Text:

Query Text Value Constrain	Text value Constrain SQL
Equals : <constrain_by_value> <value_operator>EQ</value_operator> <value_constraint>H</value_constraint> <value_type>TEXT</value_type> </constrain_by_value>	valtype_cd = 'T' AND tval_char = 'H'
Not Equals : <constrain_by_value> <value_operator>NE</value_operator> <value_constraint>L</value_constraint> <value_type>TEXT</value_type> </constrain_by_value>	valtype_cd = 'T' AND tval_char <> 'L'
Like : <constrain_by_value> <value_operator>LIKE</value_operator> <value_constraint>L</value_constraint> <value_type>TEXT</value_type>	Valtype_cd = 'T' AND tval_char LIKE 'L%'

</constrain_by_value>	
IN : <constrain_by_value> <value_operator>IN</value_operator> <value_constraint>'A,'B'</value_constraint> <value_type>TEXT</value_type> </constrain_by_value>	Valtype_cd = 'T' AND tval_char IN ('A','B')
BETWEEN : <constrain_by_value> <value_operator>BETWEEN</value_operator> <value_constraint>'A' AND 'B'</value_constraint> <value_type>TEXT</value_type> </constrain_by_value>	valtype_cd = 'T' AND tval_char BETWEEN 'A' AND 'B'

Value constrain by Flag :

Query Flag Value Constrain	Flag value Constrain SQL
Equals: <constrain_by_value> <value_operator>EQ</value_operator> <value_constraint>H</value_constraint> <value_type>FLAG</value_type> </constrain_by_value>	valueflag_cd = 'H'
Not Equals : <constrain_by_value> <value_operator>NE</value_operator> <value_constraint>H</value_constraint> <value_type>FLAG</value_type> </constrain_by_value>	valueflag_cd = 'H'
IN : <constrain_by_value> <value_operator>IN</value_operator> <value_constraint>'A', 'B'</value_constraint> <value_type>FLAG</value_type> </constrain_by_value>	valueflag_cd IN ('A','B')

3.2.1.1 VALTYPE_CD

The **valType_cd** defines what type of object is being stored in the remaining value fields. The possible values are:

Value	Description
@	No value
N	Numeric objects such as those found in lab tests
T	Text objects such as labels, short messages, enum values
B	Raw text objects such as a doctor's note, discharge summary, and radiology report.
NLP	NLP result xml objects

3.2.1.2 TVAL_CHAR

The **TVal_char** column is used in conjunction with the **ValType_cd**. The information stored in the TVal_char column is dependent on what the ValType_cd is for the object.

ValType_cd = "T"

- The text value associated with the concept_cd is stored

ValType_cd = "N"

- If an operator is associated with the numeric value then it is stored

The operators are:

Operator	Description
E	Equal
NE	Not equal
L	Less than
LE	Less than and Equal to
G	Greater than
GE	Greater than and Equal to

3.2.1.3 NVAL_NUM

If the **valType_cd** = "N" then the actual numeric value associated with the concept_cd is stored in the nval_num.

3.2.1.4 VALUEFLAG_CD

The **valueFlag_cd** is for storing flags associated with an object. It is usually seen used with a lab object to indicate that a lab value is high or low. It may also be used in conjunction with valType_cd = "B" or "NLP" to indicate encrypted data in the blob field.

The possible values are:

Value	Description
@	No value
A	Abnormal
H	High

L	Low
X	Blob field is encrypted

3.2.1.5 UNITS_CD

The **units_cd** stores the units associated with the object, such as mmol/l. It is usually used for lab test values.

3.2.1.6 OBSERVATION_BLOB

The **observation_blob** stores large text objects such as raw text (B) or NLP results (NLP). For these types of objects, ***valueFlag_cd*** indicates whether or not the data is encrypted. Other objects (numeric or short text) may store miscellaneous information about the object. For these objects, (N, T) the data in this field defaults to encrypted.

3.3 Patient_Dimension

Each record in the **patient_dimension table** represents a patient in the database. The table includes demographics fields such as gender, age, race, etc. Most attributes of the patient dimension table are discrete (i.e. Male/Female, Zip code, etc.).

Every patient_dimension table has the following four required columns:

1. patient_num
   - It is the primary key for the table therefore; it can ***not*** contain duplicates.
   - Can ***not*** be null.
   - Holds a reference number for the patient within the data repository.
   - Integer field.

2. birth_date
   - Can be null.
   - Contains the patient's date of birth (if it exists).
   - Date-time field.

3. death_date
   - Can be null.
   - Contains the patient's date of death (if it exists).
   - Date-time field.

*Note**: The birth_date and death_date fields are not standardized to a specific time zone, a limitation that may need to be addressed in the future.***

4. vital_status_cd
   - Contains a code that represents the vital status of the patient and the precision of the vital status data.
   - The values are:

Value	Description	
N	Living	corresponds to a *null* death_date
Y	Deceased	death_date accurate to *day*
M	Deceased	death_date accurate to *month*
X	Deceased	death_date accurate to *year*

*Note**: The codes for this field were determined arbitrarily as there was no standardized coding system for their representation.***

The patient_dimension table may have an unlimited number of optional columns and their data types and coding systems are specific to the local implementation. An example of a patient table is shown below. In the example table, there are eight optional columns.

patient_dimension		
PK	**patient_num**	**int**
	vital_status_cd	varchar(50)
	birth_date	datetime

patient_dimension		
	death_date	datetime
	sex_cd*	varchar(50)
	age_in_years_num*	int
	language_cd*	varchar(50)
	race_ cd *	varchar(50)
	marital_Status_cd *	varchar(50)
	religion_cd *	varchar(50)
	zip_cd *	varchar(10)
	stateCityZip_Path	varchar(700)
	patient_blob	text
	update_date	datetime
	download_date	datetime
	import_date	datetime
	sourcesystem_cd	varchar(50)
	upload_id	int

The rules for using the codes in the columns to perform queries are represented in the metadata. For example, the columns shown in the table example include a *race_cd* and a *statecityzip_cd*.

- The codes from the *race_cd column* are enumerated values that may be grouped together to achieve a desired result. For instance, if there are 4 codes to represent a race of “white”; W, WHITE, WHT, and WHITE-HISPANIC then all 4 codes can be counted directly to determine the number of white-race patients in the database.

- The codes from the *statecityzip_cd* are strings that represent hierarchical information. In this way, the string is queried from left to right in a string comparison to determine which patients are returned by the query, for example, if a code is MA\BOSTON\02114 and all the patients in BOSTON are desired, the string “MA\BOSTON*” (where * is a wildcard) would be queried.

3.4 Visit_Dimension

The **visit_dimension table** represents sessions where observations were made. Each row represents one session (also called a visit, event or encounter.) This session can involve a patient directly, such as a visit to a doctor's office, or it can involve the patient indirectly, as in when several tests are run on a tube of the patient's blood. More than one observation can be made during a visit. All visits must have a start date/time associated with them, but they may or may not have an end date. The visit record also contains specifics about the location of the session, such as the hospital or clinic the session occurred, and whether the patient was an inpatient or outpatient at the time of the visit.

The visit_dimension table has four required columns

1. visit_num
   - It is the primary key for the table; therefore it can ***not*** contain duplicates.
   - Holds a reference number for the visit within the data repository.
   - Integer field.

2. start_date
   - Can ***not*** be null.
   - Contains the date the event began.
   - Date-time field.

3. end_date
   - Can be null.
   - Contains the date the event ended
   - Date-time field.

*Note**: A visit is considered to be an event; there is a distinct beginning and ending date and time for the event. However, these dates may not be recorded and the active_status_cd is used to record whether the event is still ongoing.***

4. active_status_cd
   - Contains a code that represents the status of an event along with the precision of the available dates.
   - Conceptually it is very similar to the vital_status_cd column in the patient_dimension table.

- The values are:

Value	Description	
F	Final	
P	Preliminary	
A	Active	indicating there is no end_date
null	No dates	

*Note**: The codes for this field were determined arbitrarily as there was no standardized coding system for their representation.***

The visit_dimension table may have an unlimited number of optional columns, but their data types and coding systems are specific to the local implementation. An example of a visit table is shown below. In the example table, there are four optional columns.

visit_dimension		
PK	**encounter_num**	**int**
	patient_num	int
	active_status_cd	varchar(50)
	start_date	datetime
	end_date	datetime
	inout_cd*	varchar(50)
	location_cd*	varchar(50)
	visit_blob	text
	update_date	datetime
	download_date	datetime
	import_date	datetime
	sourcesystem_cd	varchar(50)
	upload_id	int

The rules for using the codes in the columns to perform queries are represented in the metadata, and the values within the columns follow a similar pattern as described above for the ***patient_dimension table***.

3.5 Concept_Dimension

The **concept_dimension table** contains one row for each concept. Possible concept types are diagnoses, procedures, medications and lab tests. The structure of the table gives enough flexibility to store virtually any concept type, such as demographics and genetics data.

The concept_dimension table has three required columns

1. concept_path
   - a path that delineates the concept's hierarchy

2. concept_code
   - a code that represents the diagnosis, procedure, or any other coded value

3. name_char
   - the name of the concept

concept_dimension		
PK	**concept_path**	**varchar(700)**
	concept_cd	**varchar(50)**
	name_char	varchar(2000)
	concept_blob	text
	update_date	datetime
	download_date	datetime
	import_date	datetime
	sourcesystem_cd	varchar(50)
	upload_id	int

3.6 Provider_Dimension

Each record in the **provider_dimension table** represents a physician or provider at an institution. The provider_path is the path that describes how the provider fits into the institutional hierarchy. Institution, department, provider name and a code may be included in the path.

provider_dimension		
PK **PK**	**provider_id** **provider_path**	**varchar(50)** **varchar(700)**
	name_char	varchar(850)
	provider_blob	text
	update_date	datetime
	download_date	datetime
	import_date	datetime
	sourcesystem_cd	varchar(50)
	upload_id	int

3.7 Code_Lookup

The **code_lookup table** contains coded values for different fields in the CRC. For example, in the *visit_dimension table*, there is the *location_cd field* that may have different values for different hospital locations that would be stored in the code lookup table. The first few fields of the table might look like this:

	table_cd	column_cd	code_cd	name_char
1	VISIT_DIMENSION	LOCATION_CD	@	zz not recorded
2	VISIT_DIMENSION	LOCATION_CD	BWH	Brigham and Womens Hospital
3	VISIT_DIMENSION	LOCATION_CD	FH	Faulkner Hospital
4	VISIT_DIMENSION	LOCATION_CD	MGH	Massachusetts General Hospital
5	VISIT_DIMENSION	LOCATION_CD	NWH	Newton Wellesley Hospital
6	VISIT_DIMENSION	LOCATION_CD	SPH	Spaulding Rehabilitation Hospital

code_lookup		
PK **PK** **PK**	**table_cd** **column_cd** **code_cd**	**varchar(100)** **varchar(100)** **varchar(50)**
	name_char lookup_blob update_date download_date import_date sourcesystem_cd upload_id	varchar(650) text datetime datetime datetime varchar(50) int

3.8 Patient_Mapping

The **patient_mapping table** maps the *i2b2* ***patient_num*** to an encrypted number, ***patient_ide****,* from the *source_system* (the 'e' in ide is for 'encrypted').

The ***patient_ide_source*** contains the name of the source system.

The ***patient_ide_status*** gives the status of the patient number in the source system. For example, if it is *Active, Inactive, Deleted,* or *Merged*.

patient_mapping		
PK **PK**	**patient_ide** **patient_ide_source**	**varchar(200)** **varchar(50)**
	patient_num patient_ide_status update_date download_date import_date sourcesystem_cd upload_id	**int** varchar(50) datetime datetime datetime varchar(50) int

3.9 Encounter_Mapping

The **encounter_mapping table** maps the *i2b2* ***encounter_number*** to an encrypted number, ***encounter_ide***, from the *source_system* (the 'e' in ide is for 'encrypted').

The ***encounter_ide_source*** contains the name of the source system.

The ***encounter_ide_status*** gives the status of the encounter in the source system. For example, if it is *Active*, *Inactive*, *Deleted* or *Merged*.

encounter_mapping		
PK	**encounter_ide**	**varchar(200)**
PK	**encounter_ide_source**	**varchar(50)**
	encounter_num	**int**
	patient_ide	varchar(200)
	patient_ide_source	varchar(50)
	encounter_ide_status	varchar(50)
	upload_date	datetime
	download_date	datetime
	import_date	datetime
	sourcesystem_cd	varchar(50)
	upload_id	int

3.10 Joining Columns

All of the tables above can be linked together using SQL joins to obtain more data. For example, a concept will have a code in the ***observation_fact.concept_cd*** field, but will have to be joined to the ***concept_dimension.concept_cd*** field to find the name_char and/or concept_path that define the concept. Below are some examples of common columns used to join tables in the star schema.

observation_fact

encounter_num in observation_fact	can be joined to	encounter_num in the visit_dimension table
patient_num in observation_fact	can be joined to	patient_num in the patient_dimension and visit_dimension tables
provider_id in observation_fact	can be joined to	provider_id in the provider_dimension table

patient_dimension

patient_num in patient_dimension	can be joined to	patient_num in the observation_fact and visit_dimension tables

visit_dimension

encounter_num in visit_dimension	can be joined to	encounter_num in the observation_fact table
patient_num in visit_dimension	can be joined to	patient_num in the observation_fact and patient_dimension tables

concept_dimension

concept_cd in concept_dimension	can be joined to	concept_cd in the observation_fact table

provider_dimension

provider_id in provider_dimension	can be joined to	provider_id in the observation_fact table

4. PATIENT DATA OBJECT

The Patient Data Object (PDO) is the XML representation of patient data. This data corresponds to the values in the star schema tables in the database. Below is a sample PDO. Definitions of the fields can be found in the last section of this document.

```
<repository:patient_data xmlns:repository="">
    <event_set>
        <event *>
            <event_id source="hive">1256</event_id>
            <patient_id source="hive">4</patient_id>
            <start_date>1999-02-28T13:59:00</start_date>
            <end_date>1999-02-28T13:59:00</end_date>
            <active_status_cd>F<active_status_cd>
            <param name="admission status">Inpatient</param>
            <param name="site">MGH</param>
            <param name="location">Oral Surgery</param>
            <event_blob/>
        </event>
    </event_set>
    <concept _set>
        <concept *>
            <concept_path>Diagnoses\athm\C0004096\</concept_path>
            <concept_cd>UMLS:C0004096</concept_cd>
            <name_char>Asthma</name_char>
            <concept_blob/>
        </concept>
    </concept_set>
    <observer_set>
        <observer *>
            <observer_path>MGH\Medicine\C0004096\</observer_path>
            <observer_cd>M00022303</observer_cd>
            <name_char>Shawn Murphy MD</name_char>
            <observer_blob/>
        </observer>
    </observer_set>
    <pid_set>
        <pid>
            <patient_id source="hive">4</patient_id>
            <patient_map_id source="MGH" status="A" *>0051382</patient_map_id>
            <patient_map_id source="EMPI" status="A" *>10034586</patient_map_id >
        </pid>
    </pid_set>
    <eid_set>
```

```
    <eid>
        <event_id source="hive">1256</event_id>
        <event_map_id source="MGHTSI" status="A"
        patient_id="0051382" patient_id_source="MGH" *>KST004</event_map_id>
    /eid>
</eid_set>
<patient_set>
    <patient *>
        <patient_id source='hive">4</patient_id>
        <birth_date>1930-02-28</birth_date>
        <death_date>2001-02-28</death_date>
        <vital_status_cd>Y<vital_status_cd>
        <param name="gender">Female</param>
        <param name="age in years">71</param>
        <param name="language">English</param>
        <param name="race">Black</param>
        <param name="marrital status">Married</param>
        <param name="zipcode">12345-1234</ param>
        <patient_blob/>
    </patient>
</patient_set>
<observation_set path="">
    <observation *>
        <event_id source="hive">1256</event_id>
        <patient_id source='hive">4</patient_id>
        <concept_cd name="Asthma">UMLS:C0004096</concept_cd>
        <observer_cd name="Doctor, John A.,
    MD">B001234567</observer_cd>
        <start_date>1999-02-28T13:59:00</start_date>
        <modifier_cd>@</modifier_cd>
        <valtype_cd>N</valtype_cd>
        <tval_char>E</tval_char>
        <nval_num units="ml">1.0</nval_num>
        <valueflag_cd name="High">H</valueflag_cd>
        <quantity_num>1.0</quantity_num>
        <units_cd>ml</units_cd>
        <end_date>1999-02-28T13:59:00</end_date>
        <location_cd name="Oral Surgery">MT045</location_cd>
        <confidence_num></confidence_num>
        <observation_blob/>
    </observation>
</observation_set>
<code_set>
    <code *>
```

```
            <table_cd>observation_fact</table_cd>
            <column_cd>ValueType_CD</dimension_path>
            <code_cd>N</dimension_cd>
            <name_char>Numeric</name_char>
            <code_blob/>
        </code>
    </code_set>
</repository:patient_data>
```

* indicates the following technical metadata parameters may be included in the tag (shown here with sample data values):

update_date="1999-02-28T13:59:00"

download_date="1999-02-28T13:59:00"

import_date="1999-02-28T13:59:00"

sourcesystem_cd="RPDRASTHMA"

5. PATIENT AND EVENT MAPPING SCENARIOS

A patient may have more than one identifier in different source systems and will be given a single unique i2b2 identifier. All of these identifiers are grouped together in the XML **Patient Data Object (PDO)** in the <pid_set> and are also added to the ***patient_mapping*** *table* in the database. A similar process occurs for encounters from different systems grouped together in the <eid_set> in the PDO and in the ***encounter_mapping*** *table* in the database.

The patient and event mapping tables link the values used in the i2b2 database to their counterparts in the source systems from which the identifiers came. The patient_mapping and event_mapping tables are populated by existing hive numbers when the database is created; they are also updated as new patients and encounters are added. Each patient number corresponds to a row in the patient table and each encounter or event has a row in the encounter_mapping table. The following examples review different scenarios for adding data to the mapping tables.

*NOTE****: The examples refer to the patient_mapping table, but can be applied to the encounter_mapping table in the same way; i.e. patient_num is to patient_ide as encounter_num is to encounter_ide.***

Encrypted identifiers are indicated by appending '_e' to the name of the source system. So, for example, if the identifier is an encrypted number from Massachusetts General Hospital, the source will be 'MGH_e'. The scenarios below refer both to the XML objects in the PDO and to the dimension tables and mapping tables in the database. Patient_num is the field name for the i2b2 identifier in the database and corresponds to the value of <patient_id> when the source is 'HIVE'.

Below is a generic <pid_set> from the XML Patient Data Object (PDO).

```
<pid_set>
    <pid>
        <patient_id source=”source”>value</patient_id>
        <patient_map_id source="source” status="A“>value</patient_map_id>
        <patient_map_id source=”source” status="A”>value</patient_map_id >
        ...
    </pid>
</pid_set>
```

The following cases describe possible scenarios for different combinations of <patient_id> source and value and <patient_id_map> source and value for both the <pid> and the <patient> objects. An id source and its value are both needed to determine the parameters

inserted into the mapping tables. These two fields are called the source/value pair. The patient_id in the <pid> must have the same source/value pair as in the <patient> object and the rest of the PDO. There may be multiple <patient_map_ids> in one <pid>, with each one representing a different source system and identifier value for the same patient.

The mapping process requires checking to see if the source/value pairs for <patient_id> and <patient_map_id> already exist in the i2b2 hive and then following the appropriate scenario below. The dates associated with the object must also be checked in order to determine the most recent values.

5.1 Self Mapping

Self mapping occurs when the <patient_id> source is HIVE and the <patient_id> value already exists in the hive. All hive patient and encounter numbers are mapped to themselves and inserted into their respective tables (either patient_mapping or encounter_mapping). The default mapping status is 'A' for ACTIVE and the source value is 'HIVE'.

Example:

```
<pid_set>
    <pid>
        <patient_id source="HIVE">1</patient_id>
    </pid>
</pid_set>
```

The row in the patient_mapping table will appear as follows:

patient_ide	patient_ide_source	patient_num	patient_ide_status
1	HIVE	1	A

5.2 New Mappings – Adding New Values

The following use cases address three different scenarios where at least one number does not exist in the i2b2 hive.

*NOTE**: that in these cases, the new number must be added to the patient_dimension table as well as to the patient_mapping table in the database.***

5.2.1 Case 1: (<pid> not found, generate [max+1])

If the <patient_id> source/value pair has not been added to the mapping table, a new patient_num with value max(patient_num)+1 should be generated and all the patient_nums for this patient will receive this value. The new patient number must also be added to the patient_dimension table.

Example:

New <patient_id> source/value pair = 'EMPI'/1000000

Select max(patient_num) from patient_mapping = 527

New patient_num = max(patient_num) +1 = 528

```
<pid>
    <patient_id source="EMPI">1000000</patient_id>
    <patient_map_id source="MGH">123</patient_map_id>
    <patient_map_id source="BWH">777</patient_map_id>
</pid>
```

The rows in the patient_mapping table will appear as follows:

patient_ide	patient_ide_source	patient_num	patient_ide_status
1000000	EMPI	528	A
123	MGH	528	A
777	BWH	528	A
528	HIVE	528	A

5.2.2 Case 2: (<patient> not found, generate [max + 1])

If the <patient_id> source in the <patient> object is not ‘HIVE’ **and** the patient_id source (‘MGH’) **and** value (‘123’) combination do not exist, then a new patient_num with value max(patient_num)+1 will be generated. All the patient_nums for this patient will receive this value. The new patient number must also be added to the patient_dimension table.

Example:

New <patient_id> source/value pair = ‘MGH’/123

Select max(patient_num) from patient_mapping = 527

New patient_num = max(patient_num) +1 = 528

```
<patient>
    <patient_id source="MGH">xyz</patient_id>
</patient>
```

The rows in patient_mapping table will appear as follows:

patient_ide	patient_ide_source	patient_num	patient_ide_status
123	MGH	528	A
528	HIVE	528	A

5.2.3 Case 3: (HIVE id (patient_num) not found)

Here the <patient_id> source is ‘HIVE’, but the value (528) does not exist in the mapping table. In this case, generating max+1 is not necesssary, the value 528 is not already in the table so it can be directly added. This new patient number must also be added to the patient_dimension table.

Example:

```
<pid>
    <patient_id source="HIVE">528</patient_id>
    <patient_map_id source="MGH ">123</patient_map_id>
    <patient_map_id source="BWH ">777</patient_map_id>
</pid>
```

The rows in the patient_mapping table will appear as follows:

patient_ide	patient_ide_source	patient_num	patient_ide_status
528	HIVE	528	A
123	MGH	528	A
777	BWH	528	A

5.3 Handling Existing Values

The following cases address situations where the patient_num has already been added to the mapping table.

5.3.1 Case 1: (HIVE id found, but <patient_map_id> not mapped)

In this case the patient_num (528) has been added to the mapping table, but the <patient_map_id> from both BWH and MGH have not been added; so the hive id (patient_num) is applied to all of the <patient_map_id>s that are not currently mapped for this patient.

Example:

```
<pid>
    <patient_id source="HIVE">528</patient_id>
    <patient_map_id source="MGH">123</patient_map_id>
    <patient_map_id source="BWH">777</patient_map_id>
</pid>
```

The rows in the patient_mapping table before the update:

patient_ide	patient_ide_source	patient_num	patient_ide_status
528	HIVE	528	A

The rows in the patient_mapping table after the update:

patient_ide	patient_ide_source	patient_num	patient_ide_status
528	HIVE	528	A
123	MGH	528	A
777	BWH	528	A

5.3.2 Case 2: (<patient_id>, <patient_num>, & <patient_map_id> not mapped)

In this case, the <patient_id> source and value ('EMPI'/100000) is already mapped to a patient_num, but the <patient_map_id>s are not, so use that patient_num for any of the <patient_map_id>s that are not already mapped.

Example:

```
<pid>
    <patient_id source="EMPI">100000<patient_id>
    <patient_map_id source="MGH">123</patient_map_id>
    <patient_map_id source="BWH">777</patient_map_id>
</pid>
```

The rows in the patient_mapping table before the update:

patient_ide	patient_ide_source	patient_num	patient_ide_status
1000000	EMPI	528	A
528	HIVE	528	A

The rows in the patient_mapping table after the update:

patient_ide	patient_ide_source	patient_num	patient_ide_status
1000000	EMPI	528	A
123	MGH	528	A
777	BWH	528	A
528	HIVE	528	A

5.3.3 Case 3: (patient_num already in mapping table, but with a different date)

If the <patient_id> value already exists in the mapping table, compare the update_date with the current patient record's update date. If the new record has a more recent date, then update the current patient record with this data.

Example:

```
<patient update_date="2008-05-0418:13:51.00">
    <patient_id source="HIVE">100</patient_id>
</patient>
```

The row in the patient_mapping table before the update:

patient_ide	patient_ide_source	patient_num	patient_ide_status	update_date
100	HIVE	100	A	2006-12-03 00:00:00

The row in the patient_mapping table after the update:

patient_ide	patient_ide_source	patient_num	patient_ide_status	update_date
100	HIVE	100	A	2008-05-04 18:13:51

5.3.4 Case 4: (<patient> without HIVE number)

If the <patient_id> source and value are already mapped to a patient_num, then the update date should be compared to the existing record's update date. If the new record has a more recent date, then update the current patient record with this data.

Example:

```
<patient update_date="2006-05-04T18:13:51.0Z">
    <patient_id source="MGH">123</patient_id>
</patient>
```

5.4 Invalid XML

5.4.1 Case 1: (<pid> without patient_id - INVALID)

This example is *invalid*, because it contains patient_map_ids without a patient_id. Every <pid> must have a <patient_id>. In this case the <patient_id> should be added to the PDO.

Example:

```
<pid>
    <patient_map_id source="MGH">123</patient_map_id>
    <patient_map_id source="BWH">777</patient_map_id>
</pid>
```

6. OBSERVATION FACT SCENARIOS

The updates to the observation fact can be classified into the two cases outlined in the next sections.

6.1 Case 1: Replace Old Facts with New Facts

In this case the old set of facts is replaced with a new set of facts for the matching encounter.

Example:

```
<observation update_date="2008-05-04T18:13:51.498-04:00" sourcesystem_cd="PFT">
    <event_id source="HIVE">100</event_id>
    <patient_id source="HIVE">100</patient_id>
    <concept_cd>LCS-I2B2:pulweight</concept_cd>
    <nval_num>100.9</nval_num>
</observation>
<observation update_date="2008-05-04T18:13:51.498-04:00" sourcesystem_cd="PFT">
    <event_id source="HIVE">100</event_id>
    <patient_id source="HIVE">100</patient_id>
    <concept_cd>LCS-I2B2:pulheight</concept_cd>
    <nval_num>6.0</nval_num>
</observation>
<observation update_date="2008-05-04T18:13:51.498-04:00" sourcesystem_cd="PFT">
    <event_id source="HIVE">100</event_id>
    <patient_id source="HIVE">100</patient_id>
    <concept_cd>LCS-I2B2:pulfev1pred</concept_cd>
    <nval_num>76</nval_num>
</observation>
```

The row in the observation fact table before the update:

encounter_num	patient_num	concept_cd	Nval_num	update_date
100	100	FC30.00620	10.9	2008-05-04 18:13:51
100	100	FC30.00621	20.2	2008-05-04 18:13:51
100	100	FC30.00622	6.0	2008-05-04 18:13:51

The row in the observation fact table after the update:

encounter_num	patient_num	concept_cd	Nval_num	update_date
100	100	LCSI2B2:pulweight	100.9	2008-05-04 18:13:51
100	100	LCSI2B2:pulheight	6.0	2008-05-04 18:13:51
100	100	LCSI2B2:pulfev1pred	76	2008-05-04 18:13:51

6.2 Case 2: Add New Facts

In this case new facts are added regardless of whether or not the fact's encounter exists. This involves overwriting any matching fields. i.e. if the incoming fact matches a particular stored fact and its update date is greater than the update of the matching fact, then the new fact will overwrite the old fact.

Example:

```
<observation update_date="2008-05-04T18:13:51.498-04:00" sourcesystem_cd="PFT">
    <event_id source="HIVE">100</event_id>
    <patient_id source="HIVE">100</patient_id>
    <concept_cd>FC30.00620</concept_cd>
    <nval_num>10.9</nval_num>
</observation>
<observation update_date="2008-05-04T18:13:51.498-04:00" sourcesystem_cd="PFT">
    <event_id source="HIVE">100</event_id>
    <patient_id source="HIVE">100</patient_id>
    <concept_cd> FC30.00620</concept_cd>
    <nval_num>20.2</nval_num>
</observation>
<observation update_date="2008-10-04T18:13:51.498-04:00" sourcesystem_cd="FC">
    <event_id source="HIVE">100</event_id>
    <patient_id source="HIVE">100</patient_id>
    <concept_cd>FC30.00622</concept_cd>
    <nval_num>76.0</nval_num>
</observation>
```

The row in the observation fact table before the update:

encounter_num	patient_num	concept_cd	Nval_num	update_date
100	100	FC30.00620	10.9	2008-05-04 18:13:51
100	100	FC30.00621	20.2	2008-05-04 18:13:51
100	100	FC30.00622	6.0	2008-05-04 18:13:51

The row in the observation fact table after the update:

encounter_num	patient_num	concept_cd	Nval_num	update_date
100	100	FC30.00620	10.9	2008-05-04 18:13:51
100	100	FC30.00621	20.2	2008-05-04 18:13:51
100	100	FC30.00622	76.0	2008-10-08 18:13:51

***Assumption**: the record(s) in the update file (new record) has the same primary key as a record(s) in the associated table (existing record).*

Primary Key includes:

- Encounter number
- Patient number
- Concept code
- Start date
- Modifier code
- Observer code

Append Flag = True

Following conditions will result in the new record **replacing** the existing record:

new record update date	equal to (=)		update date on the existing record	
new record update date	greater than (>)		update date on the existing record	
new record update date	is not null	AND	update date on the existing record	null
new record update date	null	AND	update date on the existing record	null

Following conditions will result in **ignoring** the new record and **not** updating the existing record:

new record update date	less than (<)		update date on the existing record	
new record update date	null	AND	update date on the existing record	is not null

7. DATA PERMISSION

The CRC determines when and how data is presented to a user based on their user role, which is specified in the PM Cell. The following table summarizes the user roles and their access permissions in the hierarchical order of least to most access.

Data Protection Track Role	Access Description	Example
DATA_OBFSC	OBFSC = Obfuscated The user can see aggregated results that are obfuscated. An example of an aggregated result is *patient count*. The user is limited on the number of times they can run the same query within a specified time period. If the user exceeds the maximum number of times then their account will be locked and only the Admin user can unlock it.	<query_result_instance> <result_instance_id>0</result_instance_id> <query_instance_id>0</query_instance_id> <query_result_type> <name>PATIENTSET</name> </query_result_type> <set_size>101</set_size> **<obfuscate_method>OBTOTAL</obfuscate_method>** <start_date>2000-12 30T00:00:00</start_date> </query_result_instance>
DATA_AGG	AGG = Aggregated The user can see aggregated results like the patient count. The results are not obfuscated and the user is not limited to the number of times they can run the same query.	<query_result_instance> <result_instance_id>0</result_instance_id> <query_instance_id>0</query_instance_id> <query_result_type> <name>PATIENTSET</name> </query_result_type> <set_size>99</set_size> **<obfuscate_method> </obfuscate_method>** <start_date>2000-12 30T00:00:00</start_date> </query_result_instance>
DATA_LDS	LDS = Limited Data Set The user can see all fields except for those that are encrypted. An example of an encrypted field is the *blob fields* in the *fact* and *dimension tables*.	PDO request: <observation_set blob="false" onlykeys="false"/>
DATA_DEID	DEID = De-identified Data The user can see all fields including	PDO request: <observation_set blob="true" onlykeys="false"/>

	those that are encrypted. An example of an encrypted field is the *blob fields* in the *fact* and *dimension tables*.	
DATA_PROT	PROT = Protected The user can see all data, including the identified data that resides in the Identity Management Cell.	

8. DEFINITION OF TERMS

patient_data	The root element that holds data from the patient data tables. May contain any number of observation_set's, and none or one patient_set, event_set, concept_set, observer_set, code_set, pid_set, or eid_set. They can occur in any order.
event_set	Data from the visit_dimension table.
event	One row of data from the visit_dimension table.
event_id	A choice between encounter_num (if source is HIVE) and encounter_id if another source. A source with "_e" at the end is encrypted.
patient_id	A choice between patient_num, (if source is HIVE) and patient_id if another source. A source with "_e" at the end is encrypted.
start_date	The date-time that the event started.
end_date	The date-time that the event ended.
active_status_cd	A code to represent the meaning of the date fields above.
param	
event_blob	XML data that includes partially structured and unstructured data about a visit.
concept_set	Data from the concept_dimension table
concept	One row of data from the concept_dimension table.
concept_path	
concept_cd	A unique code that represents a concept.
name_char	A string name that represents this concept, idea or person.
concept_blob	XML data that includes partially structured and unstructured data about a concept.
observer_set	Data from the provider_dimension table.
observer	One row of data from the provider_dimension table.
observer_path	

observer_cd	A unique code that represents a observer.
name_char	A string name that represents the observer, it could be person or machine.
observer_blob	XML data that includes partially structured and unstructured data about an observer.
code_set	Data from the code_lookup table.
code	One row of data from the code_lookup table.
table_cd	The name of one of the 8 tables represented in the PDO.
column_cd	The column name of the table where the code is found.
code_cd	The code itself.
name_char	The human-readable description of what the code represents.
pid_set	Data from the patient_mapping table.
pid	One set of mappings on a single patient_num.
patient_id	A choice between patient_num, (if source is HIVE) and patient_id if another source. A source with "_e" at the end is encrypted.
patient_map_id	A patient_id that should have the same patient_num as the patient_id in this pid.
eid_set	Data from the encounter_mapping table.
eid	One set of mappings on a single visit_num.
event_id	A choice between visit_num, (if source is HIVE) and visit_id if another source. A source with "_e" at the end is encrypted.
event_map_id	A visit_id that should have the same patient_num as the visit_id in this eid.
observation_set	
observation	One row of data from the observation_fact table.
event_id	A choice between encounter_num (if source is HIVE) and encounter_id if another source. A source with "_e" at the end is encrypted.
patient_id	A choice between patient_num, (if source is HIVE) and patient_id if another source. A source with "_e" at the end is encrypted.

concept_cd	A unique code that represents a concept.
observer_cd	An ID that represents the provider, which could be a physician or a machine such as an MRI machine.
start_date	The date that the observation was made, or that the observation started. If the data is derived or calculated from another observation (like a report) then the start date is the same as the observation it was derived or calculated from.
modifier_cd	Modifier code for a concept or provider
ValType_cd	A code representing whether a value is stored in the TVal column, NVal column, or observation_blob column.
TVal_Char	A text value.
NVal_Num	A numerical value.
ValueFlagCd	A code that represents the type of value present in the NVal_Num, the TVal_Char or observation_blob column.
quantity_num	The number of observations represented by this fact.
units_cd	A textual description of the units associated with a value.
end_date	The date that the observation ended. If the data is derived or calculated from another observation (like a report) then the end_date is the same as the observation it was derived or calculated from.
location_cd	A code representing the hospital associated with this visit.
confidence_num	A code or number representing the confidence in the accuracy of the data.
observation_blob	XML data that includes partially structured and unstructured data about an observation.
patient_set	Data from the visit_dimension table.
patient	One row of data from the visit_dimension table.
patient_id	A choice between Patient_Num, (if source is HIVE) or Patient_Id if another source. A source with "_e" at the end is encrypted.
start_date	The date-time that the patient was born.
end_date	The date-time that the patient died.
vital_status_cd	A code to represent the meaning of the date fields above.
param	
patient_blob	XML data that includes partially structured data about a patient.

annotationGroup	A group of fields that appear together at the end of all tables and store annotation or administrative information.
update_date	The date the data was last updated according to the source system from which the data was obtained. If the source system does not supply this data, it defaults to the download_date.
download_date	The date that the data was obtained from the source system. If the data is derived or calculated from other data, then the download_date is the date of the calculation.
import_date	The date the data is placed into the table of the data mart.
sourceSystem_cd	A code representing the source system that provided the data.
upload_id	Tracking number assigned to any file uploaded.