Informatics for Integrating Biology and the Bedside

i2b2 Clinical Research Chart (CRC) Design Document

Document Version: *1.1*

I2b2 Software Release: *1.4*

Table of Contents

1. INTRODUCTION 3

2. I2B2 DATA MART 3

3. I2B2 DATA MART TABLES 4

3.1 GENERAL INFORMATION 4
3.2 OBSERVATION_FACT 4
3.3 PATIENT_DIMENSION 8
3.4 VISIT_DIMENSION 9
3.5 CONCEPT_DIMENSION 10
3.6 PROVIDER_DIMENSION 11
3.7 CODE_LOOKUP 12
3.8 PATIENT_MAPPING 12
3.9 ENCOUNTER_MAPPING 13
3.10 JOINING COLUMNS 13

4. PATIENT DATA OBJECT 15

5. PATIENT AND EVENT MAPPING SCENARIOS 17

5.1 SELF MAPPING 18
5.2 NEW MAPPINGS – ADDING NEW VALUES 18
5.2.1 Case 1: (<pid> not found, generate [max+1]) 19
5.2.2 Case 2: (<patient> not found, generate [max + 1]) 19
5.2.3 Case 3: (HIVE id (patient_num) not found) 20
5.3 20
5.4 HANDLING EXISTING VALUES 20
5.4.1 Case 1: (HIVE id found, but <patient_map_id> not mapped) 20
Case 2: (<patient_id> <> patient_num, and <patient_map_id> not mapped) 21
5.4.2 Case 3 : (patient_num already in mapping table, but with a different date) 21
5.4.3 Case 4: (<patient> without HIVE number) 22
5.5 INVALID XML 22
5.5.1 Case 1: (<pid> without patient_id - INVALID) 22

6. OBSERVATION FACT SCENARIOS 23

6.1 CASE 1 EXAMPLE 23
6.2 CASE 2 EXAMPLE 24

7. DATA PERMISSION 26

8. DEFINITIONS OF TERMS 27

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 three tables, the patient, visit and observation tables. In addition to these three tables, there are three lookup tables, the concept, provider and code tables, and two mapping tables, patient_mapping and 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 radially 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: patient_dimension, concept_dimension, visit_dimension, and provider_dimension.

3. I2B2 DATA MART TABLES

3.1 GENERAL INFORMATION

The observation table has only required columns. The patient_dimension and visit_dimension tables have both required and optional columns. All the tables have 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**
PK	**modifier_cd**	**varchar(50)**
	patient_num	**int**
	valType_cd	varchar(50)
	tval_char	varchar(255)
	nval_num	decimal(18,5)
	valueFlag_cd	varchar(50)
	quantity_num	decimal(18,5)
	units_cd	varchar(50)

observation_fact		
	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**	ID number 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
PK	**modifier_cd**	Ranking of Modifiers 1, 2, 3, … 1.1, 1.2. 1.3… 1.1.1, 1.1.2. 1.1.3…	NO
	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” to store a text value When valType_cd = “N” *EQ = Equals* *NE = Not equal* *LT = Less Than* *GT = Greater Than*	

	nval_num	Used in conjunction with valType_cd = "N" to store a numerical value	
	valueFlag_cd	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 Used in conjunction with valType_cd = "N" or "T" 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 ending 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 above)	
	download_date	(as above)	
	import_date	(as above)	
	sourcesystem_cd	(as above)	
	upload_id	(as above)	

Observation Fact values columns					
valType_cd	**tval_char**	**nval_Num**	**valueFlag_cd**	**Units_cd**	**obs_blob**
N	EQ (equal), NE (not equal), LT (less than), GT (greater than)	Actual numeric value of object	H (high), L (low)	Units associated with object	Misc. encrypted information
T	Actual short text value of object	N/A	A (abnormal)	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

The observation_fact table has 6 fields associated with VALUES:

1. ValType_Cd

ValType_cd tells us what type of object is being stored in the remaining value fields.

Possible Values = N,T,B,NLP,@

@ = no value

N = Numeric objects such as that found in lab tests

T = Text objects such as labels, short messages, enum values

B = Raw Text objects such as doctor's notes, discharge summaries, radiology reports

NLP = NLP Result xml objects

2. TVal_Char

If ValType_cd = 'T', then the text value associated with the concept_cd is stored here, in tval_char.

If ValType_cd = 'N' and there is an operator associated with the numeric value, such as 'equal to', 'less than', or 'greater than', then the operator is represented in tval_char as either 'EQ' (equal), 'NE' (not equal), 'LT' (less than) or 'GT' (greater than).

3. NVal_Num

If ValType_cd = 'N', then the actual numeric value associated with the concept_cd is stored here, in nval_num.

4. ValueFlag_Cd

ValueFlag_cd is for flags associated with an object. It is usually 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 content in the blob field.

Possible Values = A,H,L,X,@

@ = no value
A = Abnormal
H = High
L = Low
X = blob field is encrypted

5. Units_Cd

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

6. Observation_Blob

Large text objects such as Raw Text (B) or NLP results (NLP) are stored here. 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 four required columns, Patient_Num, Birth_Date, Death_Date, and Vital_Status_Cd. The patient_num column must be filled (it must not be null). The patient_num column is the primary key for the table and therefore can not contain duplicates. This column holds a reference number for the patient within the data repository. It is an integer. The birth_date and death_date columns can be null, and are date-time fields. They contain the birth and death dates for the patient if they exist. They are not standardized to a specific time zone, a limitation that may need to be addressed in the future. The vital_status_cd column contains a code that represents the vital status of the patient, and the precision of the vital status data. The codes for this field were determined arbitrarily, as there was no standardized coding system for their representation. The values for the vital_status_cd column are N for living (corresponds to a null death_date) and Y for deceased (death_date accurate to day), M for deceased (death_date accurate to month), and X for deceased (death_date accurate to year).

The Patient table may have an unlimited number of optional columns and their data types and coding systems are local implementation-specific. An example of a Patient _Dimension table is shown below. In the example table, there are eight optional columns. The rules for using the codes in the columns to perform queries are represented in the metadata. For example, the columns shown below include a race_cd column and a statecityzip_cd column. Codes from the race_cd column are enumerated values that may be grouped together to achieve a desired result, such as if there are 4 codes for the “white” race such as W, WHITE, WHT, and WHITE-HISPANIC, they can be counted directly to determine the number of white-race patients in the database. 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.

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

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 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 at which 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 Patient_Num, Start_Date, End_Date, and Active_status_cd. The visit_num column is the primary key for the table and therefore can not contain duplicates. This column holds a reference number for the visit within the data repository. It is an integer. The start_date and end_date columns can be null, and are date-time fields. Because 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,along with the precision of the available dates. Conceptually, this makes it very similar to the vital_status_cd column in the patient table. The codes for this field were determined arbitrarily, as there was no standardized coding system

for their representation. The codes are ‘F’ for final, ‘P’ for preliminary, ‘A’ for active (indicating there is no end_date) and null if there are no dates.

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

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

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_path is a path that delineates the concept’s hierarchy. The concept_code is the code that represents the diagnosis, procedure, or any other coded value. Name_char is the actual 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 doctor or provider at an institution. The provider_path is the path that describes the how the provider fits into the institutional hierarchy. Institution, department, provider name and a code may be included in the path.

provider_dimension		
PK	**provider_id**	**varchar(50)**
PK	**provider_path**	**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_looup 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 types hospital locations and these values would be stored in the code lookup table. The first four fields of the table might look like this:

	table_cd	column_cd	code_...	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	SRH	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 from the source_system,patient_ide (the ‘e’ in ide is for ‘encrypted’.) Patient_ide_source contains the name of the source system. Patient_ide_status gives the status of the patient number in the source system, for example, if it is Active or Inactive or 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 from the source system,encounter_ide_source (the 'e' in ide is for 'encrypted'.) Encounter_ide_source contains the name of the source system. 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 **PK**	**encounter_ide** **encounter_ide_source**	**varchar(200)** **varchar(50)**
	encounter_num patient_ide patient_ide_source encounter_ide_status upload_date download_date import_date sourcesystem_cd upload_id	**int** varchar(200) varchar(50) varchar(50) datetime datetime datetime varchar(50) 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_Dimesnion 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 Visit_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>1.1</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 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 numbers came. The patient_mapping and event_mapping tables are populated by existing hive numbers when the database is created and are also added to 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. (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 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 Massachussetts 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>
```

Row in patient_mapping table:

patient_ide	patient_ide_source	patient_num	patient_ide_status
1	HIVE	1	A

5.2 NEW MAPPINGS – ADDING NEW VALUES

The following cases address three different situations where there is a number that does not already 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 get 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>
```

Rows in patient_mapping table:

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 ('xyz') combination do not exist, a new patient_num with value max(patient_num)+1 should be generated all the patient_nums for this patient will get this value. The new patient number must also be added to the patient_dimension table.

Example:

```
<patient>
   <patient_id source="MGH">xyz</patient_id>
   <param  name="zipcode>02149/param>
</patient>
```

Parameters:
New <patient_id> source/value pair = 'MGH'/xyz

Select max(patient_num) from patient_mapping = 527
New patient_num = max(patient_num) +1 = 528

Rows in patient_mapping table:

patient_ide	patient_ide_source	patient_num	patient_ide_status
xyz	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 (1000000) does not exist in the mapping table. In this case, generating max+1 is not necesssary, the value 1000000 can be added directly to the table, since it is not already in the table. This new patient number must also be added to the patient_dimension table.

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

Rows in patient_mapping table:

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

5.3

5.4 Handling Existing Values

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

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

In this case the patient_num (1000000) has been added to the mapping table, but the <patient_map_id>s from BWH and MGH for the patient have not so the hive id (patient_num) is applied to all of the <patient_map_id>s that are not currently mapped.

```
<pid>
    <patient_id source="HIVE">1000000</patient_id>
    <patient_map_id source="MGH">123</patient_map_id>
    <patient_map_id source="bwh">123</patient_map_id>
</pid>
```

Rows in patient_mapping table before :

patient_ide	patient_ide_source	patient_num	patient_ide_status
1000000	HIVE	1000000	A

Rows in patient_mapping table after:

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

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

In this case, the <patient_id> source and value ('EMPI'/100000) are 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.

```
<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>
```

Rows in patient_mapping table before :

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

Rows in patient_mapping table after:

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

5.4.2 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.

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

```
    <param  name="zipcode">02149</param>
</patient>
```

Row in patient_mapping table before :

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

Row in patient_mapping table after:

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

5.4.3 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.

```
<patient update_date="2006-05-04T18:13:51.0Z">
    <patient_id source="MGH">xyz</patient_id>
    <param  name="zipcode">02149</param>
</patient>
```

5.5 Invalid XML

5.5.1 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.

```
<pid>
    <patient_map_id source="MGH">123</patient_map_id>
    <patient_map_id source="bwh">123</patient_map_id>
</pid>
```

6. OBSERVATION FACT SCENARIOS

The updates to the observation fact can be classified into two cases.
Case 1) Replace the old set of facts with the new set of facts for the matching encounter.
Case 2) Add new facts, irrespective of whether the fact's encounter exists or not.

The case (2) also involves overwriting any matching facts. i.e. if the incoming fact matches a particular stored fact and its update date greater than the matched fact's update date, then the new fact will overwrite the old fact.

6.1 Case 1 example

Row in observation_fact table before :

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

```
<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: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:pulfev1pred</concept_cd>
      <nval_num>76</nval_num>

</observation>
```

Row in observation_fact table after:

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 example

Row in observation_fact table before:

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

```
<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:pulweight</concept_cd>
        <nval_num>100.9</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>
```

Row in observation_fact table after:

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	2008-10-04 18:13:51

100	100	LCSI2B2:pulweight	100.9	2008-05-04 18:13:51
100	100	LCSI2B2:pulheight	6.0	2008-05-04 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

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 **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 the data presented to the user based on the user roles, specified in the PM Cell. The following table summarizes the user roles and their access permissions in the hierarchical order.

Data Protection Track Role	Access Description	Example
DATA_OBFSC	Access to aggregate result like the patient counts after it is obfuscated. Also the data obfuscation user can maximum run a same query, M times within a specified time period, after which the user account will be locked. Only the Admin user can unlock the account.	<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	Can see the aggregate result like the patient count without obfuscation.	<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	Can see all the fields except the blob field in fact and dimension tables.	PDO request: <observation_set blob="false" onlykeys="false"/>
DATA_DEID	Can see the blob field in the fact and the dimension tables.	PDO request: <observation_set blob="true" onlykeys="false"/>
DATA_PROT	Can see identified data. The identified data resides in the Identity Management Cell.	

8. DEFINITIONS 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) or 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) or 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 this observer, could be person or machine.

observer_blob: XML data that includes partially structured and unstructured data about a 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) or 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) or 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) or Encounter_Id if an other source. A source with "_e" at the end is encrypted.

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.

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: hierarchical derivations of a common observation

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.

ValueFlag_Cd: 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.