Informatics for Integrating Biology and the Bedside

i2b2 Cell Messaging

Data Repository (CRC) Cell

Document Version: *1.7.08-004*

i2b2 Software Version: *1.7.08*

TABLE OF CONTENTS

TABLE OF CONTENTS **3**

DOCUMENT MANAGEMENT **5**

1 INTRODUCTION **6**

2 I2B2 DATA MART **7**

3 I2B2 DATA MART TABLES **8**

3.1 General Information 8
3.1.1 Table Requirements *8*
3.1.2 Administrative Columns *8*
3.1.3 Supported Databases and Data Types *8*
3.2 OBSERVATION_FACT Table 9
3.2.1 Value Columns *12*
3.2.1.1 VALTYPE_CD Column 13
3.2.1.2 TVAL_CHAR Column 13
3.2.1.3 NVAL_NUM Column 14
3.2.1.4 VALUEFLAG_CD Column 14
3.2.1.5 UNITS_CD Column 14
3.2.1.6 OBSERVATION_BLOB Column 15
3.2.2 Example of Value Constraints Used in Queries *15*
3.2.2.1 Value Constraint by Number 15
3.2.2.2 Value Constraint by Text 18
3.2.2.3 Value Constraint by Flag 19
3.3 PATIENT_DIMENSION Table 20
3.4 VISIT_DIMENSION Table 24
3.5 CONCEPT_DIMENSION Table 28
3.6 PROVIDER_DIMENSION Table 29
3.7 MODIFIER_DIMENSION Table 30
3.8 CODE_LOOKUP Table 31
3.9 PATIENT_MAPPING Table 32
3.10 ENCOUNTER_MAPPING Table 33
3.11 Joining Columns 34

4 PATIENT DATA OBJECT **36**

5 PATIENT AND EVENT MAPPING SCENARIOS **39**

5.1 Self-Mapping 40
5.2 New Mappings – Adding New Values 41
5.2.1 Use Case 1: Create new entry if PID not found *41*
5.2.2 Use Case 2: Create new entry if patient does not exist *42*
5.2.3 Use Case 3: Create new entry if hive entry does not exist for a patient *43*
5.3 Handling Existing Values 43
5.3.1 Use Case 1: Hive id found but <patient_map_id> is not mapped *44*
5.3.2 Use Case 2: Patient id, num and map_ids are not mapped *45*
5.3.3 Use Case 3: PATIENT_NUM in mapping table but with a different date *46*
5.3.4 Use Case 4: <patient> without a HIVE number *47*
5.4 Invalid XML 47
5.4.1 Use Case 1: <pid> without a PATIENT_ID) *47*

6 OBSERVATION FACT SCENARIOS 48

6.1 USE CASE 1: REPLACE OLD FACTS WITH NEW FACTS 48
6.2 USE CASE 2: ADD NEW FACTS 49

7 DATA PERMISSION 52

8 GLOSSARY 54

8.1 GENERAL TERMS 54

DOCUMENT MANAGEMENT

Revision Number	Date	Author	Description
1.7.001	09/13/12	Janice Donahoe	Created 1.7 version of the document
1.7.002	001/10/2014	Janice Donahoe	Updated primary key documentation for OBSERVATION_FACT table. Also updated various tables that had new columns added in 1.7
1.7.00-003	07/30/2015	Janice Donahoe	Fixed minor grammar and formatting issues.
1.7.08-004	09/06/2016	Janice Donahoe	Fixed revision number.

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 in this document. In the 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
5. MODIFIER_DIMENSION

3 I2B2 DATA MART TABLES

3.1 General Information

3.1.1 Table Requirements

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.

3.1.2 Administrative Columns

All the i2b2 tables have the following five technically-oriented or administrative columns.

Column Name	Data Type	Allow Nulls	Definition
UPDATE_DATE	datetime	Yes	Date the row was updated by the source system The 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 the data was imported into the CRC
SOURCESYSTEM_CD	datetime	Yes	A coded value for the data source system
UPLOAD_ID	datetime	Yes	A numeric id given to the upload

3.1.3 Supported Databases and Data Types

With the release of 1.7, i2b2 now supports three different types of databases;

1. SQL Server
2. Oracle 10g
3. PostgreSQL

For consistency purposes the data types listed in this document are those for a SQL Server database. For your convenience the following table displays a mapping of the data types used by the i2b2 tables for each of the supported types of databases.

Type of Data	SQL Server	Oracle 10g	PostgreSQL
Alphanumeric	CHAR	CHAR	CHAR
Alphanumeric	TEXT	CLOB	TEXT
Alphanumeric	VARCHAR	VARCHAR2	VARCHAR
Date	DATETIME	DATE	TIMESTAMP
Numeric	DECIMAL	NUMBER	DECIMAL
Numeric	INT	NUMBER	INT
Numeric	INT IDENTITY	NUMBER	SERIAL

3.2 OBSERVATION_FACT Table

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 together the OBSERVATION_FACT table with one or more dimension tables.

OBSERVATION_FACT		
PK	**ENCOUNTER_NUM**	**INT**
PK	**PATIENT_NUM**	**INT**
PK	**CONCEPT_CD**	**VARCHAR(50)**
PK	**PROVIDER_ID**	**VARCHAR(50)**
PK	**START_DATE**	**DATETIME**
PK	**MODIFIER_CD**	**VARCHAR(100)**
PK	**INSTANCE_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)

	END_DATE	DATETIME
	LOCATION_CD	VARCHAR(50)
	OBSERVATION_BLOB	TEXT
	CONFIDENCE_NUM	DECIMAL(18,5)
	UPDATE_DATE	DATETIME
	DOWNLOAD_DATE	
	IMPORT_DATE	
	SOURCESYSTEM_CD	VARCHAR(50)
	UPLOAD_ID	INT
	TEXT_SEARCH_INDEX	INT IDENTITY(1,1)

OBSERVATION_FACT			
Key	**Column Name**	**Column Definition**	**Allow Nulls? (Default = YES)**
PK	**ENCOUNTER_NUM**	Encoded i2b2 patient visit number	NO
PK	**PATIENT_NUM**	Encoded i2b2 patient number	NO
PK	**CONCEPT_CD**	Code for the observation of interest (i.e. diagnoses, procedures, medications, lab tests)	NO
PK	**PROVIDER_ID**	Practitioner or provider id	NO
PK	**START_DATE**	Starting date-time of the observation (*mm/dd/yyyy*)	NO
PK	**MODIFIER_CD**	Code for modifier of interest (i.e. "ROUTE", "DOSE"). Note that the value columns are often used to hold the amounts such as "100" (mg) for the modifier of DOSE or "PO" for the modifier of ROUTE.	NO
PK	**INSTANCE_NUM**	Encoded instance number that allows more than one modifier to be provided for each CONCEPT_CD. Each row will have a different MODIFIER_CD but a similar INSTANCE_NUM.	NO
	VALTYPE_CD	Format of the concept *N = Numeric* *T = Text (enums / short messages)*	YES

		B = Raw Text (notes / reports) *NLP = NLP result text*	
	TVAL_CHAR	Used in conjunction with VALTYPE_CD = "T" or "N" When the VALTYPE_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*	YES
	NVAL_NUM	Used in conjunction with VALTYPE_CD = "N" to store a numerical value	YES
	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 data in the blob column is encrypted. *X = Encrypted text in the blob column* When the VALTYPE_CD = "N" or "T" it is used to flag certain outlying or abnormal values *H = High* *L = Low* *A = Abnormal*	YES
	QUANTITY_NUM	Quantity of the value in the NVAL_NUM column	YES
	UNITS_CD	Units of measurement for the value in the NVAL_NUM column	YES
	END_DATE	The end date-time for the observation	YES
	LOCATION_CD	A location code, such as for a clinic	YES
	OBSERVATION_BLOB	Holds any raw or miscellaneous data that exists, often encrypted PHI	YES
	CONFIDENCE_NUM	Assessment of accuracy of data	YES
	UPDATE_DATE	As defined in the above section ("*General Information*")	YES
	DOWNLOAD_DATE	As defined in the above section ("*General Information*")	YES

	IMPORT_DATE	As defined in the above section ("*General Information*")	YES
	SOURCESYSTEM_CD	As defined in the above section ("*General Information*")	YES
	UPLOAD_ID	As defined in the above section ("*General Information*")	YES
	TEXT_SEARCH_INDEX	Used by SQL Server and PostgreSQL when doing a large text search. The column is automatically updated by the CRC. ** This column is not included in the table for an Oracle database.	

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 Value 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 to) G (greater than) GE (greater than and equal to)	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		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

3.2.1.1 VALTYPE_CD Column

The **VALTYPE_CD** defines what type of object is being stored in the remaining value columns. 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 message, enumerated 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 Column

The **TVAL_CHAR** column is used in conjunction with the **VALTYPE_CD** column. 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 in the VALTYPE_CD.

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 Column

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 Column

The **VALUEFLAG_CD** column 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 *OBSERVATION_BLOB* column.

The possible values are:

Value	Description
@	No value
A	Abnormal
H	High
L	Low
X	Blob column is encrypted

3.2.1.5 UNITS_CD Column

The **UNITS_CD** column 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 Column

The **OBSERVATION_BLOB** column stores large text objects such as raw text (B) or NLP results (NLP). For these types of objects, the *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.

3.2.2 Example of Value Constraints Used in Queries

3.2.2.1 Value Constraint by Number

If the fact with a numerical value didn't have the normalized numerical value with a single *UNIT_CD* for a particular concept, then the user can tell the service to do the unit conversion of the *NVAL_NUM* column before applying the value constraints in the query.

The unit conversion of *NVAL_NUM* is calculated using the concept's metadata xml defined in the Ontology cell (<ConvertingUnits/>, <MultiplyingFactor/>).

To enable the unit conversion, set the following project parameter in the Project Management cell.

CRC_ENABLE_UNITCD_CONVERSION = ON | OFF

Value	Description
ON	Unit Conversion is enabled
OFF	Unit Conversion is not enabled

Note

This unit conversion option will slow down the query. For better query performance load, normalize the numerical fact values and do not enable this option.

Greater than operator	
Query Numeric Value Constraint:	<constrain_by_value> <value_operator>GT</value_operator> <value_constraint>99.9</value_constraint> <value_type>NUMBER</value_type> </constrain_by_value>
Numeric Constraint SQL:	(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') ***Unit Conversion Enabled*** (valtype_cd = 'N' AND case when unit_cd = 'mg/5ml' then nval_num > 99.9 * 5 when unit_cd = 'mg/15ml' then nval_num > 99.9 * 15 when unit_cd = 'mg/0.5ml' then nval_num > 99.9 * 0.5 AND tval_char IN ('GE','E')) OR (valtype_cd = 'N' AND case when unit_cd = 'mg/5ml' then nval_num > 99.9 * 5 when unit_cd = 'mg/15ml' then nval_num > 99.9 * 15 when unit_cd = 'mg/0.5ml' then nval_num > 99.9 * 0.5 AND tval_char = 'G')

Less than operator	
Query Numeric Value Constraint:	<constrain_by_value> <value_operator>LT</value_operator> <value_constraint>99.9</value_constraint> <value_type>NUMBER</value_type> </constrain_by_value>
Numeric Constraint SQL:	(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 operator	
Query Numeric Value Constraint:	<constrain_by_value> <value_operator>BETWEEN</value_operator> <value_constraint>1 and 100</value_constraint> <value_type>NUMBER</value_type> </constrain_by_value>
Numeric Constraint SQL:	(valtype_cd = 'N' AND nval_num BETWEEN 1 and 100 AND tval_char = 'E')

Equal to operator	
Query Numeric Value Constraint:	<constrain_by_value> <value_operator>EQ</value_operator> <value_constraint>99.9</value_constraint> <value_type>NUMBER</value_type> </constrain_by_value>
Numeric Constraint SQL:	(valtype_cd = 'N' AND nval_num = 99.9 AND tval_char = 'E')

Less than and Equal to operator	
Query Numeric Value Constraint:	<constrain_by_value> <value_operator>LE</value_operator> <value_constraint>99.9</value_constraint> <value_type>NUMBER</value_type> </constrain_by_value>
Numeric Constraint SQL:	(valtype_cd = 'N' AND nval_num <= 99.9 AND tval_char IN ('L','E','LE')

Greater than and Equal to operator	
Query Numeric Value Constraint:	<constrain_by_value> <value_operator>GE</value_operator> <value_constraint>99.9</value_constraint> <value_type>NUMBER</value_type> </constrain_by_value>
Numeric Constraint SQL:	(valtype_cd = 'N' AND nval_num >= 99.9 AND tval_char IN ('G','E','GE')

Not Equal operator	
Query Numeric Value Constraint:	<constrain_by_value> <value_operator>NE</value_operator> <value_constraint>99.9</value_constraint> <value_type>NUMBER</value_type> </constrain_by_value>
Numeric Constraint SQL:	(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')

3.2.2.2 Value Constraint by Text

Equals operator	
Query Text Value Constraint:	<constrain_by_value> <value_operator>EQ</value_operator> <value_constraint>H</value_constraint> <value_type>TEXT</value_type> </constrain_by_value>
Text Value Constraint SQL:	valtype_cd = 'T' AND tval_char = 'H'

Not equals operator	
Query Text Value Constraint:	<constrain_by_value> <value_operator>NE</value_operator> <value_constraint>L</value_constraint> <value_type>TEXT</value_type> </constrain_by_value>
Text Value Constraint SQL:	valtype_cd = 'T' AND tval_char <> 'L'

Like operator	
Query Text Value Constraint:	<constrain_by_value> <value_operator>LIKE</value_operator> <value_constraint>L</value_constraint>

	<value_type>TEXT</value_type> </constrain_by_value>
Text Value Constraint SQL:	valtype_cd = 'T' AND tval_char LIKE 'L%')

In operator	
Query Numeric Value Constraint:	<constrain_by_value> <value_operator>IN</value_operator> <value_constraint>'A','B'</value_constraint> <value_type>TEXT</value_type> </constrain_by_value>
Text Value Constraint SQL:	valtype_cd = 'T' AND tval_char = ('A','B')

Between operator	
Query Text Value Constraint:	<constrain_by_value> <value_operator>BETWEEN</value_operator> <value_constraint>'A' and 'B'</value_constraint> <value_type>TEXT</value_type> </constrain_by_value>
Text Value Constraint SQL:	valtype_cd = 'T' tval_char BETWEEN 'A' AND 'B'

3.2.2.3 Value Constraint by Flag

Equals operator	
Query Flag Value Constraint:	<constrain_by_value> <value_operator>EQ</value_operator> <value_constraint>H</value_constraint> <value_type>FLAG</value_type> </constrain_by_value>
Flag Value Constraint SQL:	valueflag_cd = 'H'

Not equals operator

Query Flag Value Constraint:	<constrain_by_value> <value_operator>NE</value_operator> <value_constraint>L</value_constraint> <value_type>FLAG</value_type> </constrain_by_value>
Flag Value Constraint SQL:	valueflag_cd <> 'H'

In operator	
Query Flag Value Constraint:	<constrain_by_value> <value_operator>IN</value_operator> <value_constraint>'A','B'</value_constraint> <value_type>FLAG</value_type> </constrain_by_value>
Flag Value Constraint SQL:	valueflag_cd IN ('A', 'B')

3.3 PATIENT_DIMENSION Table

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

Starting from version 1.6, this table will support custom columns apart from the required ones. The PDO service will return the custom fields in the <param> tag within the <patient> element. Please refer to the section called *CODE_LOOKUP Table* for adding the descriptions to the custom fields.

The following table shows the rules for mapping the custom field's database type to the xml type.

Database to XML type mapping:

XML Type	Oracle Type	SQL Server Type	PostgreSQL Type
string	varchar(n), varchar2(n), char(n), nchar(n), nchar2(n), clob, nclob	varchar(n), nvarchar, char(n), nchar, nvarchar, text, ntext	varchar(n), char(n), text
dateTime	Date	date, datetime, datetime2, smalldatetime, timestamp	timestamp(p), date, interval(fieds,p)

int	int, number(p,s), shortinteger, longinteger	int, bigint, tinyint, smallint	int, bigint, smallint
decimal	number(p,s), decimal, shortdecimal, float(n), binary_float, binary_double	decimal(p,s), numeric(p,s), float(n)	number(p,s), decimal(p,s), real, float4, double precision, float8

The PATIENT_DIMENSION table has the following four **REQUIRED columns**:

1. PATIENT_NUM
    - It is part of the primary key for the table; therefore, it cannot contain duplicates.
    - Cannot be null.
    - Holds a reference number for the patient within the data repository.
    - Integer field.

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

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

Note

The BIRTH_DATE and DEATH_DATE columns 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 code consists of two characters; the first one represents the validity of the DEATH_DATE and the second one is for the BIRTH_DATE.

The VITAL_STATUS_CD values are:

KEY:

"*" means that a second character should be the birth date indicator (if exists)

"_" means that a first character should be the death date indicator (if exists)

Date Explained	Value	Description	
Death date	N*	Living	corresponds to a *null* DEATH_DATE
Death date	(null)*	Living	corresponds to a *null* DEATH_DATE
Death date	U*	Unknown	corresponds to a *null* DEATH_DATE
Death date	Z*	Deceased	corresponds to a *null* DEATH_DATE
Death date	Y*	Deceased	DEATH_DATE accurate to *day*
Death date	M*	Deceased	DEATH_DATE accurate to *month*
Death date	X*	Deceased	DEATH_DATE accurate to *year*
Death date	R*	Deceased	DEATH_DATE accurate to *hour*
Death date	T*	Deceased	DEATH_DATE accurate to *minute*
Death date	S*	Deceased	DEATH_DATE accurate to *second*
Birth date	_L	Unknown	corresponds to a *null* BIRTH_DATE
Birth date	_(null)	Known	BIRTH_DATE accurate to *day*

Birth date	_D	Known	BIRTH_DATE accurate to *day*
Birth date	_B	Known	BIRTH_DATE accurate to *month*
Birth date	_F	Known	BIRTH_DATE accurate to *year*
Birth date	_H	Known	BIRTH_DATE accurate to *hour*
Birth date	_I	Known	BIRTH_DATE accurate to *minute*
Birth date	_C	Known	BIRTH_DATE accurate to *second*

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**
	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)
	INCOME_CD	VARCHAR(50)
	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 four codes to represent a race of “white”; W, WHITE, WHT, and WHITE-HISPANIC then all four 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 the 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 patient in BOSTON are desired, the string “MA\BOSTON*” (where * is a wildcard) would be queried.

3.4 VISIT_DIMENSION Table

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 an outpatient at the time of the visit.

Starting from version 1.6, this table will support custom columns apart from the required ones. The custom column in the table follows the same setup rule as the ones in the

PATIENT_DIMENSION table. Please refer to the *PATIENT_DIMENSION* section for the data type mapping information.

The VISIT_DIMENSION table has the following four **REQUIRED columns**:

1. ENCOUNTER_NUM
    - It is part of the primary key for the table; therefore, this column in combination with the PATIENT_NUM cannot contain duplicate combinations.
    - Cannot be null.
    - Holds a reference number for the patient within the data repository.
    - Integer field.

2. PATIENT_NUM
    - It is part of the primary key for the table; therefore, this column in combination with the ENCOUNTER_NUM cannot contain duplicate combinations.
    - Cannot be null.
    - Holds a reference number for the patient within the data repository.
    - Integer field.

3. START_DATE
    - Can be null.
    - Contains the date the event began.
    - Date-time field.

4. 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 going on.

5. 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 code consists of two characters; the first one represents the validity of the END_DATE and the second one is for the START_DATE.

The ACTIVE_STATUS_CD values are:

KEY:

"*" means that a second character should be the start date indicator (if exists)

"_" means that a first character should be the end date indicator (if exists)

Date Explained	Value	Description	
End date	U*	Unknown	corresponds to a *null* END_DATE
End date	O*	Ongoing	corresponds to a *null* END_DATE
End date	(null)*	Known	END_DATE accurate to *day*
End date	Y*	Known	END_DATE accurate to *day*
End date	M*	Known	END_DATE accurate to *month*
End date	X*	Known	END_DATE accurate to *year*

End date	R*	Known	END_DATE accurate to *hour*
End date	T*	Known	END_DATE accurate to *minute*
End date	S*	Known	END_DATE accurate to *second*
Start date	_L	Unknown	corresponds to a *null* START_DATE
Start date	_A	Active	corresponds to a *null* START_DATE
Start date	_(null)	Ongoing	START_DATE accurate to *day*
Start date	_D	Ongoing	START_DATE accurate to *day*
Start date	_B	Known	START_DATE accurate to *month*
Start date	_F	Known	START_DATE accurate to *year*
Start date	_H	Known	START_DATE accurate to *hour*
Start date	_I	Known	START_DATE accurate to *minute*
Start date	_C	Known	START_DATE accurate to *second*

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 eight optional columns.

VISIT_DIMENSION		
PK	**ENCOUNTER_NUM**	**INT**
PK	**PATIENT_NUM**	**INT**
	ACTIVE_STATUS_CD	**VARCHAR(50)**
	START_DATE	**DATETIME**

	END_DATE	**DATETIME**
	INOUT_CD	VARCHAR(50)
	LOCATION_CD	VARCHAR(50)
	LOCATION_PATH	VARCHAR(900)
	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 previously described for the *PATIENT_DIMENSION* table.

3.5 CONCEPT_DIMENSION Table

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 type of concept, 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_CD
    - 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 Table

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	**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 MODIFIER_DIMENSION Table

The **MODIFIER_DIMENSION** table contains one row for each modifier. The modifier has the similar hierarchical organization as the concept type. The Ontology cell maintains the mapping of how the modifier applies to the types of concepts and at what level; whether it is applied to a particular concept or to all its children.

The MODIFIER_DIMENSION table has three **REQUIRED columns**.

1. MODIFIER_PATH
    - A path that delineates the modifier's hierarchy

2. MODIFIER_CD
    - A code that represents the modifier

3. NAME_CHAR
    - The name of the modifier

MODIFIER_DIMENSION		
PK	**MODIFIER_PATH**	**VARCHAR(700)**
	MODIFIER_CD	**VARCHAR(50)**
	NAME_CHAR	**VARCHAR(2000)**
	MODIFIER_BLOB	TEXT
	UPDATE_DATE	DATETIME

	DOWNLOAD_DATE	DATETIME
	IMPORT_DATE	DATETIME
	SOURCESYSTEM_CD	VARCHAR(50)
	UPLOAD_ID	INT

3.8 CODE_LOOKUP Table

The **CODE_LOOKUP** table contains coded values for different columns in the CRC. For example, in the *VISIT_DIMENSION* table there is the *LOCATION_CD* column that may have different values for different hospital locations that would be stored in the *CODE_LOOKUP* table. The first few records of the table might look like this:

	TABLE_CD	COLUMN_CD	CODE_CD	NAME_CHAR
1	VISIT_DIMENSION	LOCATION_CD	@	Not recorded
2	VISIT_DIMENSION	LOCATION_CD	BWH	Brigham and Women's 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

Starting from version 1.6.00, the description of custom columns is also stored in this table. To store the column descriptor, the value of CODE_CD column should be "CRC_CUSTOM_DESCRIPTOR".

TABLE_CD	COLUMN_CD	CODE_CD	NAME_CHAR
PATIENT_DIMENSION	INCOME_CD	CRC_COLUMN_DESCRIPTOR	Income
VISIT_DIMENSION	ACTIVE_STATUS_CD	CRC_COLUMN_DESCRIPTOR	Date Accuracy

CODE_LOOKUP		
PK	**TABLE_CD**	**VARCHAR(100)**
PK	**COLUMN_CD**	**VARCHAR(100)**
PK	**CODE_CD**	**VARCHAR(50)**
	NAME_CHAR	VARCHAR(650)
	LOOKUP_BLOB	TEXT
	UPLOAD_DATE	DATETIME
	UPDATE_DATE	DATETIME
	DOWNLOAD_DATE	DATETIME
	IMPORT_DATE	DATETIME
	SOURCESYSTEM_CD	VARCHAR(50)
	UPLOAD_ID	INT

3.9 PATIENT_MAPPING Table

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	**PATIENT_IDE**	**VARCHAR(200)**
PK	**PATIENT_IDE_SOURCE**	**VARCHAR(50)**
	PATIENT_NUM	**INT**
	PATIENT_IDE_STATUS	VARCHAR(50)
	PROJECT_ID	**VARCHAR(50)**

	UPDATE_DATE	DATETIME
	DOWNLOAD_DATE	DATETIME
	IMPORT_DATE	DATETIME
	SOURCESYSTEM_CD	VARCHAR(50)
	UPLOAD_ID	INT

3.10 ENCOUNTER_MAPPING Table

The **ENCOUNTER_MAPPING** table maps the *i2b2* **ENCOUNTER_NUM** 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 patient number 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)**
PK	**PROJECT_ID**	**VARCHAR(50)**
	ENCOUNTER_NUM	**INT**
	PATIENT_IDE	VARCHAR(200)
	PATIENT_IDE_SOURCE	VARCHAR(50)
	ENCOUNTER_IDE_STATUS	VARCHAR(50)
	UPLOAD_DATE	DATETIME
	UPDATE_DATE	DATETIME
	DOWNLOAD_DATE	DATETIME
	IMPORT_DATE	DATETIME
	SOURCESYSTEM_CD	VARCHAR(50)

	UPLOAD_ID	INT

3.11 Joining Columns

All of the tables above can be linked together using SQL joins to obtain more data.

Example:

A concept will have a code in the **OBSERVATION_FACT.CONCEPT_CD** column but will have to be joined to the **CONCEPT_DIMENSION.CONCEPT_CD** column to find the *NAME_CHAR* and / or *CONCEPT_PATH* defined for the concept.

The following 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 table
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 *Definition of Terms* section.

```
<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>
            <param column="inout_cd" type="string" name=""
            column_descriptor="in vs. outpatient code"></param>
            <param column="location_cd" type="string" name=""
            column_descriptor="location code"></param>
            <param column="location_path" type="string" name=""
            column_descriptor="location hierarchy"></param>
            <param column="active_status_cd" type="string" name=""
            column_descriptor="date accuracy code"></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>
            <param column="vital_status_cd" type="string"
                 column_descriptor="date accuracy code"
                 name=" "></param>
            <param column="sex_cd"  type="string"
                column_descriptor="gender" name=""></param>
            <param column="age_in_years_num" type="int"
                column_descriptor="age"></param>
            <param column="language_cd" type="string"
                column_descriptor="language" name=""></param>
            <param column="race_cd" type="string"
                column_descriptor ="race" name=""></param>
            <param column="religion_cd" type="string"
                column_descriptor ="religion" name=""></param>
            <param column="marital_status_cd" type="string"
                column_descriptor ="marital status" name=""></param>
            <param column="statecityzip_path_char" type="string"
                column_descriptor="zip code hierarchy"></param>
            <param column="gender_cd" type="string" name=""
                    column_descriptor="Gender"></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</column_cd>
            <code_cd>N</code_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=”DEMO”

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 ENCOUNTER_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. For example, if the identifier is an encrypted number from Massachusetts General Hospital, the source will be 'MGH_e'.

The scenarios below refer to both the XML objects in the PDO and to the dimension and mapping tables in the database. PATIENT_NUM is the column 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	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**
>
> In these cases, the new number must be added to the PATIENT_DIMENSION table as well as the PATIENT_MAPPING table in the i2b2 database.

5.2.1 Use Case 1: Create new entry if PID not found

(<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
1	1000000	EMPI	528	A
2	123	MGH	528	A
3	777	BWH	528	A
4	528	HIVE	528	A

5.2.2 Use Case 2: Create new entry if patient does not exist

(<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

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

1	123	MGH	528	A
2	528	HIVE	528	A

5.2.3 Use Case 3: Create new entry if hive entry does not exist for a patient

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:

```
<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
1	528	HIVE	528	A
2	123	MGH	528	A
3	777	BWH	528	A

5.3 Handling Existing Values

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

5.3.1 Use Case 1: Hive id found but <patient_map_id> is 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
1	528	HIVE	528	A

The rows in the PATIENT_MAPPING table **after** the update:

	PATIENT_IDE	PATIENT_IDE_SOURCE	PATIENT_NUM	PATIENT_IDE_STATUS
1	528	HIVE	528	A
2	123	MGH	528	A
3	777	BWH	528	A

5.3.2 Use Case 2: Patient id, num and map_ids are 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">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 **before** the update:

	PATIENT_IDE	PATIENT_IDE_SOURCE	PATIENT_NUM	PATIENT_IDE_STATUS
1	1000000	EMPI	528	A
2	528	HIVE	528	A

The rows in the PATIENT_MAPPING table **after** the update:

	PATIENT_IDE	PATIENT_IDE_SOURCE	PATIENT_NUM	PATIENT_IDE_STATUS
1	1000000	EMPI	528	A
2	528	HIVE	528	A
3	123	MGH	528	A
4	777	BWH	528	A

5.3.3 Use Case 3: PATIENT_NUM in mapping table but with a different date

If the <patient_id> value already exists in the mapping table then compare the UPDATE_DATE 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 ="2008-05-0418:13:51.00">
    <patient_map_id source="HIVE">100</patient_map_id>
    <patient_map_id source="BWH">777</patient_map_id>
</patient>
```

The rows in the PATIENT_MAPPING table **before** the update:

	PATIENT_IDE	PATIENT_IDE_ SOURCE	PATIENT_NUM	PATIENT_IDE _STATUS	UPDATE_DATE
1	100	HIVE	100	A	2006-12-03 00:00:00

The rows in the PATIENT_MAPPING table **after** the update:

	PATIENT_IDE	PATIENT_IDE_ SOURCE	PATIENT_NUM	PATIENT_IDE _STATUS	UPDATE_DATE
1	100	HIVE	100	A	2008-05-04 18:13:51

5.3.4 Use Case 4: <patient> without a 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 ="2006-05-0418:13:51.00">
    <patient_map_id source="MGH">123</patient_map_id>
</patient>
```

5.4 Invalid XML

5.4.1 Use Case 1: <pid> without a PATIENT_ID)

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

Updates to the OBSERVATION_FACT table can be classified into two cases; (1) Replace old facts with new facts and (2) Add new facts. Both of these cases are further defined in the next two sections.

6.1 Use 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 rows in the OBSERVATION_FACT table **before** the update:

	ENCOUNTER_ NUM	PATIENT_NUM	CONCEPT_CD	NVAL_NUM	UPDATE_DATE
1	100	100	FC30.00620	10.9	2008-05-04 18:13:51

2	100	100	FC30.00621	20.2	2008-05-04 18:13:51
3	100	100	FC30.00622	6.0	2008-05-04 18:13:51

The rows in the OBSERVATION_FACT table **after** the update:

	ENCOUNTER_NUM	PATIENT_NUM	CONCEPT_CD	NVAL_NUM	UPDATE_DATE
1	100	100	LCSI2B2:pulweight	100.9	2008-05-04 18:13:51
2	100	100	LCSI2B2:pulheight	6.0	2008-05-04 18:13:51
3	100	100	LCSI2B2:pulfev1pred	76	2008-05-04 18:13:51

6.2 Use Case 2: Add new facts

In this case new facts are added to the OBSERVATION_FACT table 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>FC30.00621</concept_cd>
    <nval_num>20.2</nval_num>
```

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

The rows in the OBSERVATION_FACT table **before** the update:

	ENCOUNTER_NUM	PATIENT_NUM	CONCEPT_CD	NVAL_NUM	UPDATE_DATE
1	100	100	FC30.00620	10.9	2008-05-04 18:13:51
2	100	100	FC30.00621	20.2	2008-05-04 18:13:51
3	100	100	FC30.00622	6.0	2008-05-04 18:13:51

The rows in the OBSERVATION_FACT table **after** the update:

	ENCOUNTER_NUM	PATIENT_NUM	CONCEPT_CD	NVAL_NUM	UPDATE_DATE
1	100	100	FC30.00620	10.9	2008-05-04 18:13:51
2	100	100	FC30.00621	6.0	2008-05-04 18:13:51
3	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:

Description	Column Name	XML tag
Patient number	PATIENT_NUM	<patient_id>
Concept code	CONCEPT_CD	<concept_cd>
Modifier code	MODIFIER_CD	<modifier_cd>
Start date	START_DATE	<start_date>
Encounter number	ENCOUNTER_NUM	<event_id>
Instance number	INSTANCE_NUM	<instance_num>
Observer code	PROVIDER_ID	<observer_cd>

Append Flag = True

The 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

The 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 Project Management (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>OBSUBTOTAL</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>101</set_size> <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 columns* 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 those that are encrypted. An example of an encrypted field is the *blob columns* in the *fact* and *dimension tables*.	PDO request: <observation_set blob="true" onlykeys="false"/>

DATA_PROT	PROT = Protected Data The user can see all data, including the identified data that resides in the Identity Management Cell.	

8 GLOSSARY

8.1 General Terms

The following table contains terms that are used throughout this document.

Term	Definition
patient_data	The root element that holds data from the patient data tables. May contain any number of observation_sets, 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. As 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. As 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 (START_DATE & END_DATE).
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 an observer (provider).
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 in the table where there 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 the source is HIVE) and PATIENT_ID (if another source is defined). A source with "_e" at the end is encrypted.
patient_map_id	A PATIENT_ID that should have the same PTIENT_NUM as the PATIENT_ID in the PID.
eid_set	Data from the ENCOUNTER_MAPPING table.
eid	One set of mappings on a single ENCOUNTER_NUM.
event_id	A choice between ENCOUNTER_NUM (if the source is HIVE) and ENCOUNTER_ID (if another source is defined). A source with "_e" at the end is encrypted.
event_map_id	An ENCOUNTER_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 the source is HIVE) and ENCOUNTER_ID (if another source is defined). A source with "_e" at the end is encrypted.
patient_id	A choice between PATIENT_NUM (if the source is HIVE) and PATIENT_ID (if another source is defined). A source with "_e" at the end is encrypted.

concept_cd	A unique code that represents a concept.
observer_cd	A unique code that represents an observer (provider).
start_date	The date that the observation was made or the observation started. If the date 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	The modifier code for a concept or provider.
valtype_cd	A code representing whether a value is stored in the TVAL_CHAR, NVAL_NUM, or OBSERVATION_BLOB columns.
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 columns.
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 date 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 PATIENT_DIMENSION table.
patient	One row of data from the PATIENT_DIMENSION table.
patient_id	A choice between PATIENT_NUM (if the source is HIVE) and PATIENT_ID (if another source is defined). A source with "_e" at the end is encrypted.
birth_date	The date-time the patient was born.
death_date	The date-time the patient died.
vital_status_cd	A code to represent the meaning of the date fields above (BIRTH_DATE & DEATH_DATE).
param	
patient_blob	XML data that includes partially structured and unstructured 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 dos not supply this data, it defaults to the DOWNLOAD_DATE.
download_date	The date 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	A tracking number assigned to any file uploaded.