information

MDPI

Article

Entity Attribute Value Style Modeling Approach for Archetype Based Data

Shivani Batra [1], Shelly Sachdeva [1,*] and Subhash Bhalla [2]

[1] Department of Computer Science and Engineering, Jaypee Institute of Information Technology University, Noida 201304, India; ms.shivani.batra@gmail.com

[2] Division of Information Systems, University of Aizu, Fukushima 965-8580, Japan; bhalla@u-aizu.ac.jp

* Correspondence: shelly.sachdeva@jiit.ac.in; Tel.: +91-120-2400973

Received: 31 October 2017; Accepted: 20 December 2017; Published: 29 December 2017

Abstract: Entity Attribute Value (EAV) storage model is extensively used to manage healthcare data in existing systems, however it lacks search efficiency. This study examines an entity attribute value style modeling approach for standardized Electronic Health Records (EHRs) database. It sustains qualities of EAV (i.e., handling sparseness and frequent schema evolution) and provides better performance for queries in comparison to EAV. It is termed as the Two Dimensional Entity Attribute Value (2D EAV) model. Support for ad-hoc queries is provided through a user interface for better user-interaction. 2D EAV focuses on how to handle template-centric queries as well as other health query scenarios. 2D EAV is analyzed (in terms of minimum non-null density) to make a judgment about the adoption of 2D EAV over n-ary storage model of RDBMS. The primary aim of current research is to handle sparseness, frequent schema evolution, and efficient query support altogether for standardized EHRs. 2D EAV will benefit data administrators to handle standardized heterogeneous data that demands high search efficiency. It will also benefit both skilled and semi-skilled database users (such as, doctors, nurses, and patients) by providing a global semantic interoperable mechanism of data retrieval.

Keywords: entity attribute value model; Electronic Health Records (EHRs); standardization; archetype based EHRs

1. Introduction

Efficient data management and faster access procedures are essential for healthcare application. Healthcare data management poses various challenges in terms of sparseness (high percentage of null values), frequent evolution (changes in schema, and thus, changes in corresponding healthcare application) and quick data access.

- *Sparseness:* Among the vast dimensions (attributes) of healthcare domain, only few are active for a patient [1]. For example, a patient with fever might not undergo any blood test, and thus, the corresponding attributes will contain null (sparse) values.
- *Frequent Evolution:* With time, medical knowledge evolves. This results in new diagnosis parameters for providing more accurate decisions. For example, a few years back, four-dimensional (4D) ultrasound technology had been introduced that assisted in a better understanding of the fetus. This requires changes in existing database schema (and thus, changes in corresponding healthcare application) for accommodating the new knowledge in terms of attributes/parameters to be recorded and presented to the user on demand.
- *Quick Data Access:* Data extraction can be for a specific patient or for a population. When patient data or population is extracted, target data can be characterized as rows and columns of a relational model, respectively. Extracting patient data instantly is highly demanded in healthcare

domain as it can lead to life loss. Whereas, population queries need not be answered in real time [1]. But, irrespective of the type of query, faster data access is always appreciable.

RDBMS follow the n-ary storage model (NSM) approach, in which a table consists of 'n' columns (one per attribute) [2]. A typical normalized relational database follows good ER design policy. A sparse table can be divided into multiple non-sparse normalized tables with a good ER design. However, a frequent evolution of schema in a good ER scenario is very expensive and seeks the involvement of a schema designer for changes in the schema. Changes in schema can be the addition of an attribute, deletion of an attribute or data type modification corresponding to an attribute. Addition of an attribute is the most expensive operation that might result in the creation of new tables and corresponding relationships. The existing data is then moved to the newly defined tables. In addition, every time that the schema is modified, the information system built on the underlying schema needs to be modified accordingly and retested for flawless operation.

In relational database literature, various data storage models (such as, entity attribute value model [3], decomposed storage model [4], wide table [5], and interpreted storage [6]) have been proposed for storing sparse datasets. Healthcare domain persisted data into various databases (such as XML, Node+Path, archetype relational mapping, and dynamic tables) based on these data storage models as well [1,7–10]. The Entity Attribute Value (EAV) model is observed to be the most widely adopted storage model in clinical systems [11]. The EAV [1] model has a fixed schema structure consisting of three columns, referred to as Entity, Attribute, and Value. The 'Entity' column will store the contents of the primary key, the 'Attribute' column will store the name of the attribute, and the 'Value' column will store the data value. For each non-null entry (except primary key entries) in the relational table, one row in the EAV table is constructed. Thus, EAV stores only non-null values. Also, EAV model enhances flexibility by allowing any number of attributes to be added by just specifying its name in the 'Attribute' column. This enables no changes in schema and the underlying information system.

Existing research [1,11–13] in healthcare/biomedical, as well as other domains, such as e-commerce [3] and semantic web [14], strongly favors EAV. EAV was first employed in the TMR (The Medical Record) system [15] and the HELP Clinical Data Repository [16–18]. The presence of a single 'value' column in EAV hinders the ability to use multiple data types. Thus, to deal with heterogeneity, the open-source TrialDB clinical study data management system [19,20] explored the use of multiple EAV tables (one table for each data type). Further, EAV has been extended to incorporate relationships among various medical concepts through the EAV/CR framework [21]. Yale's SenseLab [22,23] used the EAV/CR framework to build a publicly accessible neuroscience database. EAV is also utilized by Oracle's health sciences division in its commercial systems ClinTrial [24] and Oracle Clinical [25], for modeling clinical data attributes. Nowadays, many commercial applications utilize various aspects of EAV internally, including Oracle Designer [26] (for ER modeling), and Kalido [27] (for data warehousing and master data management).

A good ER design may result in thousands of tables. For example, in the clinical domain, hundreds to thousands of relational tables (one corresponding to one form) need to be generated [11]. Intermountain Healthcare's enterprise data warehouse involves 9000 tables and 10 terabytes of data [11,28]. Rows in tables (corresponding to a good ER design) may vary from few to thousands or millions in number. However, visually, tables with huge number of rows are emphasized equally to the tables with few rows. The tables with few rows are suitable candidates for EAV representation. For example, in ontology modelling environment, categories (classes) must often be created on the fly, and some classes are often eliminated in subsequent cycles of prototyping [29]. This situation is best candidate for EAV that can accommodate changes in classes without schema change.

Primarily, current research is focused on faster data retrieval and complex ad-hoc query support in addition to the characteristics (handling sparseness and frequent evolution) of EAV model.

- *Faster data retrieval:* Besides the extensive adaptability of EAV in the healthcare domain, EAV lacks search efficiency. For data extraction, an exhaustive scan of EAV tables is required, which adds a delay in processing the output. Thus, this research paper proposes an entity attribute value style modeling approach for standardized Electronic Health Record (EHR) databases. The new storage model proposed is termed as Two Dimensional Entity Attribute Value Model (2D EAV) that extends the EAV to provide faster accessibility for patient-specific and population queries in comparison to EAV. 2D EAV uses a mixture of the EAV model and the NSM of RDBMS to produce a generic storage system that stores only non-sparse data and can accommodate new knowledge with a better access speed.
- *Complex ad-hoc query support:* EAV database is complemented with metadata to store all schema related details. EAV model represents the physical structure (i.e., how data is actually stored on disk) and the metadata associated with it depicts the logical structure (i.e., how data is visible to end users). However, in the healthcare domain, end users (such as, doctors, nurses, patients, and pharmacists) need not require any knowledge about the physical and/or logical structure of data. Moreover, every user is not aware of SQL. Even if a user is aware of SQL, the query corresponding to EAV will be highly complex and error-prone [1]. Thus, a Graphical User Interface (GUI) must be provided to the medical domain user for accessing Electronic Health Records (EHRs) that are stored in the database of a healthcare application. Current research provides a GUI corresponding to 2D EAV storage system. The GUI generates the SQL query corresponding to ad-hoc queries on fly, such as query that is constructed by desktop resident query tools, to extract the desired information without any prior knowledge about the underlying schema (2D EAV in our case) provided to the proposed GUI as an input.

Healthcare domain demands for frequent unambiguous exchange of data for better medical assistance to patients. Semantic interoperability and standardized data representation are crucial tasks in the management of modern clinical trials [30]. Many standards (such as openEHR [31], ISO13606 [32–34], and HL7 [35]) are making guidelines for standardized EHRs. The dual model approach [36], has been introduced in the Synapses project [37] and adopted by the openEHR standard. The openEHR project has developed clinical model-driven architecture for future-proof interoperable EHRs systems [38] and can be harmonized with other standards [39]. In this study, we adopt a dual model approach for standardization of EHRs. It divides the framework of defining medical concepts electronically into two levels that segregate knowledge from information, as shown in Figure 1.

The reason behind adopting a dual model approach is the need for flexibility in adding new medical concepts without modifying the existing system. Level 1 in the dual model approach is referred to as the Reference Model (RM) [33,36], which defines the basic building blocks (such as data types and data structures) of various medical concepts. Level 2, referred to as the Archetype model (AM) [36], makes use of the information defined in RM to produce complete knowledge regarding a medical concept (in the form of online available deliverables known as an archetype in openEHR paradigm). AM applies constraints on the information that is provided in RM to build archetypes.

Medical concepts are presented in the form of archetypes. An archetype constitutes all knowledge regarding one medical concept, such as the various attributes that are included in a given medical concept, their data types, their range, and any other constraints [34,36]. For example, the Body Weight archetype constitutes two attributes, termed as 'Weight' and 'Comment', whose data types are 'Quantity' (quantifiable) and 'Text' (textual), respectively. An archetype may logically include other archetypes, and/or a specialization of another archetype. Thus, they are flexible and vary in form. In terms of scope, they are general-purpose, reusable and composable [40]. The openEHR archetypes are freely available for download from a standard online library such as Clinical Knowledge Manager (CKM) [41]. Archetypes also provide links to standard terminologies such as SNOMED-CT (Systematized Nomenclature of Medicine Clinical Terms) [42] to avoid any ambiguity of terms. An archetype is authored after a rigorous review process that involves a team constituting clinical experts and information technology experts [41]. For each attribute in an archetype, a maximum

possible occurrence is defined. Thus, an attribute can be optional or mandatory. Different healthcare organizations can tailor their needs by inheriting the information stored in AM (in the form of archetypes) through the use of templates [36]. For example, a gynecology department will have a template designed using archetypes that are related to pregnancy, menstruation, and other women-specific diseases. This study has made use of various templates to collect standardized EHRs data for experimentation. The current research aims to reduce the delay in accessing the standardized EHRs data.

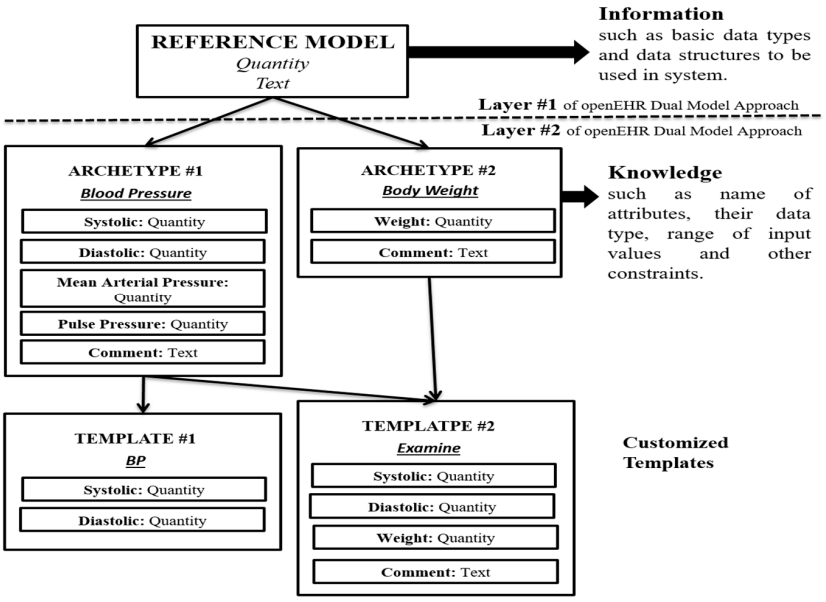


Figure 1. Dual model approach.

1.1. Related Studeis

Sparseness in medical domain is attributed towards the different procedures and processes being followed by distinguished healthcare providers. For example, a blood pressure (archetype) constitutes of five attributes- systolic, diastolic, mean arterial, pulse pressure, and comment. A doctor in one hospital may record blood pressure in terms of systolic and diastolic pressure, whereas another doctor may record blood pressure in terms of mean arterial pressure. Moreover, there are many situations where all the parameters are not recorded for a particular medical scenario. Thus, sparseness is introduced. Many approaches have been suggested to handle data efficiently as follows:

1. NoSQL systems have been introduced to overcome limitations of Relational Database Management Systems (RDBMS) (See the next following Section 1.1.1).
2. In the healthcare domain, dual model approach opens a new path for handling data to make a stable system that can capture future knowledge without making changes in the existing application (See the next following Section 1.1.2).
3. Various storage approaches over existing RDBMS are suggested to make the system compatible with future evolution and/or to avoid sparseness (see the next following Section 1.1.3).

1.1.1. Why Not Nosql?

NoSQL databases are gaining popularity as a storage option for highly sparse and frequently evolving data. NoSQL systems are aimed at providing schema-less support for data storage to attain flexibility. However, to attain flexibility abandoning schema entirely is not a good option [43]. Nandkarni stated that NoSQL database (such as Cassandra) came to same conclusion and introduced CQL for schema definition and data manipulation for productivity of developers.

For data analysis, most of the NoSQL databases leverage Hadoop MapReduce functionality. This takes user away from standard SQL, which renders a large number of SQL based third party analytical tools that are present in market (such as IBM Cognos, Tableau, SAP Business Objects, and Microstrategy) unusable [44]. Also, several systems (such as Hadapt, Hive, and Impala) that provide an SQL interface to the data residing in Hadoop require an external schema definition from the user to enable data analytics via SQL. In addition, some of NoSQL systems (such as Apache Cassandra) have introduced their own SQL-like query languages (such as Apache Cassandra introduced CQL) to provide a more user friendly and easier learning environment to developers (that are in practice of using SQL). Providing SQL-like query language enables schema definition at logical level. This schema support helps in building a constrained storage system to avoid incorrect data.

In order to utilize the capabilities of RDBMS, such as transactional support, storage-integrated access-control, read-write concurrency control, statistics gathering, and cost-based query optimization capabilities, authors choose RDBMS for building the proposed storage system. In addition, NewSQL system [45] also favors to maintain ACID properties, which is inherently provided in RDBMS. However, NSM is expensive to evolve [3,5,7–10], and is restricted up to a certain limit to avoid disk page overflow [46]. To attain the flexibility offered by NoSQL, we have adopted an approach similar to key-value approach (of NoSQL) i.e., EAV on RDBMS. In addition, a query builder is provided to make user free from the burden of writing complex queries, and thus, interact with the proposed system flawlessly.

1.1.2. Storage of openEHR Standard Based Data

There are three openEHR based approaches. These are compared in Table 1.

Table 1. openEHR based persistence approaches.

Persistence Approach	Modeling Level	Advantage	Limitation
Object Relational Model (ORM) [7]	Reference Model	Simple process of creating tables for classes defined in RM. Also, stable in nature since, RM is stable.	Deep hierarchy present in openEHR RM structure complicates the ORM scenario.
XML, JSON, & Node+Path (using BLOB) [8]	Reference Model	BLOB is used to denote the hierarchy in form of path.	Complicated paths that consume more space as well as cause delay in data access.
Archetype Relational Mapping (ARM) [9]	Archetype	One relational table is created corresponding to one archetype.	Schema evolution requires efforts from schema designer to modify schema and thus, changes in application code.

The first approach, Object Relational Mapping (ORM) provides a set of relational tables that can capture details of any object defined in healthcare domain. However, the ORM approach gets complicated as levels of hierarchy increases [7]. Since RM describes a deep hierarchy, ORM is not well suited for storing standard based health records. The second approach suggests capturing knowledge about hierarchy (path) in a BLOB (binary large objects). Various storage approaches, such as XML and JSON, exploit BLOB to store openEHR complaint data. One of the approaches utilizing BLOB in an RDBMS is Node+Path [8]. Node+Path approach is similar to EAV; both use semantic paths as attribute names. Wang et al. [9] suggest an Archetype Relation Mapping (ARM) approach as an optimization of ORM. It proposes to use one NSM table per archetype, with some additional metadata tables to support schema evolution. Each archetype is mapped to one relational table using a defined set of mapping rules.

1.1.3. Storage Approaches in RDBMS for Sparse Dataset

To deal with sparseness, many efforts are done at the physical layer [5,47,48], as well as the logical layer [1,10] of RDBMS. Various storage approaches suggested in RDBMS literature are detailed in the following subsections.

- *Physical Level Modification*

Beckmann et al. [5] proposed a modification of physical layer (termed as interpreted attribute storage format) of row oriented RDBMS. It replaced the fixed length tuple by variable length tuple storing non-null attribute-value pairs and associated length. This approach provides efficient storage but degrades the performance of population queries (due to variable length tuples).

Microsoft included 'Sparse Column' [47] functionality in SQL Server 2008 and later versions to mitigate the effect of sparseness. However, 'Sparse Column' falls short in following:

1. 'Sparse Column' functionality is not applicable to many data types that are quite common nowadays, such as 'String', 'Timestamp', and 'Geometry', etc.
2. No constraints can be applied to 'Sparse Columns'.
3. No data compression is possible for 'Sparse Columns'.
4. Copying data from one machine to another will result in a loss of the 'Sparse Column' functionality.

In addition to 'Sparse Column', Microsoft introduced column store index [49] in SQL Server 2012 and later version to enhance the performance of population queries. Also, many columnar RDBMS are incorporating various compression techniques to handle sparseness [48]. However, a little space is wasted even after compression. For example, null bitmap reserves one bit for each null value.

Existing solutions (discussed in this sub-section) deal well with sparseness and also enhance the performance of queries, but falls short in case of schema evolution.

As discussed before, with evolving schema, the schema designer needs to redesign the ER for incorporating newly evolved attributes. With a new good ER design, corresponding changes must be reflected to the database schema, as well as the application. This incurs extra cost and the loss of stability. Therefore, it has been suggested to adopt a generic schema for storing standardized EHRs [12,13] when considering frequent schema evolution.

Partition across (PAX) [50] divide the n-ary table into multiple pages and each page is vertically partitioned in cache. This enables the improvement in search efficiency of OLTP queries by keeping whole tuple in cache. Simultaneously, OLAP performance is improved since spatial locality of attribute data is improved by vertically partitioning.

Another approach, termed as fractured mirror [51], provides two disk images of same dataset. Each disk image has two same fragments of the dataset, but distinct physical organization. For instance, Disk 1 will save fragment 1 as n-ary table and fragment 2 as vertically partitioned dataset. Whereas, Disk 2 will save fragment 1 as per vertically partitioned dataset and fragment 2 as n-ary table. Depending upon the type of query (OLTP or OLAP), the best organization is chosen by the optimizer.

HYRISE [52] also works in the direction of providing efficient storage mechanism. It provides a storage hybrid architecture that inherits advantages of NSM and vertical partitioning. It creates variable length partitions of the whole database. Each partition can be stored either as n-ary table or vertically partitioned table as per the underlying requirements. If attributes are accessed frequently, the choice of storage should be vertical partitioning, such as in case of OLAP queries. For accessing row specific data (OLTP scenario), NSM is opted.

Pinnecke et al. [53] has presented a survey of various storage approaches, including PAX, fractured mirrors, and HYRISE, which do not deal with sparseness; however, they provide storage with improved search efficiency.

- *Logical Level Modification*

EAV is a widely adopted logical level approach. However, search inefficiency of EAV demands for other storage models, or enhancement to existing EAV structure. An alternate approach for storing sparse dataset is to create one binary table corresponding to each attribute of a relational table [10,54]. The first column of the binary table contains primary key and the second column defines the corresponding attribute. It improves the performance of population queries. However, the performances of patient-specific queries worsen. Patient specific queries extract entity specific data that needs to be scanned in all the binary tables of database irrespective of the fact that only a few binary tables are applicable for underlying entity.

Sparse dataset exhibits a special characteristic that entities are likely to have the same subset of non-null attributes [46]. The elements of this subset are termed as co-occurring attributes. To improve the performance of patient specific queries, a better approach (than constructing binary tables) is to group co-occurring attributes in one relational table. Information regarding co-occurring attributes needs to be discovered beforehand. Clustering algorithms, such as K-Nearest Neighbor can be applied to identify co-occurring attributes. Baumgartner et al. [55] presented an efficient technique, termed as SURFING (subspaces relevant for clustering), which identifies cluster based on relevance. Relevance is identified based on interestingness of a subspace using the k-nearest neighbor distances of the objects. Irrespective of the efficiency of clustering algorithm, as the schema evolves, the co-occurring attributes may also evolve, which in turn, may require rebuilding of an application built on the previous schema (as in case of good ER design). Current research also considers co-occurring attributes for the partitioning of data in different tables. However, partitions in the proposed approach follow EAV model and information regarding co-occurring attributes are extracted from the openEHR archetype definition (as detailed in next section).

- *Wide Table Approach*

Besides physical level and logical level modifications, a popular RDBMS approach is to store a large number of entities belonging to the same entity set in one wide table [4]. This approach relies on the compression techniques that are offered at physical layer by the underlying columnar RDBMS. The wide table approach eliminates the involvement of a schema designer as schema evolves. However, schema evolution is still expensive as modifications need to be reflected in the corresponding information system. Another point of concern related with wide table is querying dataset involving a huge number of attributes, since the user cannot remember thousands of attributes that are involved in the dataset. Moreover, a drop-down menu is not an efficient option because scrolling thousands of attributes do not seem to be a feasible solution. Thus, Chu et al. [4] proposed a keyword search mechanism to provide the user with potential desired attribute set. Keyword searching [56] is good as the user does not need to remember all the attributes. However, it is not always possible to retrieve only desired attributes (additional attributes are also extracted with matching keyword).

- *Materialized Views*

Materialized views [57] are very advantageous for storing results of queries to avoid long running calculations every time that a query is executed. At the physical storage level, materialized views behave like indexes. It is easy to add attributes to an NSM table without making changes to the materialized views. As long as the users access data through views, the relationships between the tables can be changed without disrupting queries. Thus, a good ER can be followed and the underlying tables that are physically stored in the database do not need to be sparse. Views can provide an efficient solution in situations where underlying queries are static (not changed). However, in the healthcare domain, parameters (attributes) evolve frequently. The newly evolved parameters need to be accommodated in an existing database and must be inquired to reflect patients' situation. Thus, having a static view cannot resolve the issue of frequent evolution.

1.1.4. Performing Analytical Operations

Databases are designed to support two categories of operations, i.e., transactional (that usually demands patient specific data) and analytical (that requires population data). A popular tool, termed as Informatics for Integrating Biology and the Beside (i2b2), uses EAV for the purpose of data storage to perform analytical queries [58]. I2b2 is a self-service tool that has been designed (and becoming a de facto standard) specifically for patients' cohort identification. It allows to perform a population-wide search to identify the amount of patient existing as per a study-specific criteria for feasibility analysis of the underlying study [58]. To provide functionality of cohort identification and feasibility analysis, i2b2 performs analytical operations on population data. However, it lacks support for querying data related to specific patient. The system proposed in current research (termed as 2D EAV) supports extraction of patient specific, as well as population, data with a better speed than that of EAV.

I2b2 implements an EAV based star schema that enables integration of healthcare data from disparate sources. However, the use of EAV in i2b2 hinders constraint definition, and thus, i2b2 is completely dependent upon the individual contributing systems for the implication of constraints [59]. In contrast, openEHR provides a dual layer modelling approach that segregates the information (in reference model) from knowledge (in archetype model). Archetypes define data quality constraints to be placed on the individual system and the content of record entries [40]. Individual systems can use openEHR archetypes for implementing constraint definitions. This constrained dataset can be migrated to i2b2 for patients' cohort identification. Haarbrandt et al. [60] proposed an approach to automate the process of populating i2b2 clinical data warehouse with openEHR complaint dataset. In future, authors will try to export 2D EAV complaint data to i2b2 (by following a similar approach as suggested by Haarbrandt et al.) for further enhancing the capabilities of 2D EAV.

1.2. Objective of This Research

Objectives of current research are to provide (1) better access speed; (2) adherence to standards; (3) capability to accommodate new knowledge without modifying existing schema and information system; (4) less storage with no sparseness; and, (5) ease of ad-hoc query. Current research aims to build an EAV style modeling approach, termed as 2D EAV, that can be used for storage of highly sparse and evolving data belonging to healthcare as well as other domains.

2. Method

Our study proposes a modified entity attribute value storage model named 2D EAV. It is especially designed for storing heterogeneous and archetype-based data. In addition to the capabilities of EAV, such as handling sparseness, generic structure, and frequent schema evolution, 2D EAV also enhances searching and querying capabilities with support for querying data related to the desired templates.

2.1. Design and Implementation of 2D EAV

Foundations of 2D EAV lies in partitioning data stored in a single EAV table into multiple EAV tables using two dimensions. The parameters chosen for partitioning are data semantics, improving spatial locality of co-occurring attributes and different types of data. The first dimension chosen is archetype (based on data semantics and improving spatial locality), and the second dimension chosen is data type (based on storing heterogeneous data). Hence, the name 2D EAV.

Partitioning based on data semantics plays an important role in improving search efficiency [61]. Thus, 2D EAV utilizes the data semantics of medical concepts that are defined in an archetype to create partitions of data. Knowledge representation of clinical concepts is through archetypes that enable semantic interoperability of heterogeneous systems [40]. Another motive behind choosing archetype as a dimension for partitioning is improving spatial locality of co-occurring attributes [46]. Archetype corresponds to the parameters that belong to the underlying medical concept (which tends to be recorded together, i.e., co-occurring attributes). Thus, partitioning based on archetypes enables

presence of co-occurring data in one table. Adoption of archetypes eliminate the overhead of applying clustering algorithms for extracting co-occurring attributes details.

openEHR follows a rigorous archetype definition process thus an archetype is considered as an authenticated standard definition of a medical concept [41]. Any evolution in knowledge is released as a versioned archetype. Thus, changes in existing co-occurring group of attributes can be handled through 2D EAV without making any changes (or rebuilding) in the schema and existing healthcare application. Detail about version handling in 2D EAV is explained in Section 4.3.

2D EAV segregates the data based on data types (the second dimension) to support heterogeneity. In the absence of partitioning based on data types, multiple value columns corresponding to different data types (such as, value_int, value_text, value_boolean) are required in each archetype table. In such a scenario, only one value column will contain the entry and all of the others will be null, resulting in sparseness. This motivated us to partition the archetype table corresponding to data types (followed by the Value column of underlying tables).

Let A be an archetype with attributes of three distinct data types, i.e., DT1, DT2, and DT3. The set of attributes (say, with elements a1, a2 . . . , a8) of A is divided into three subsets corresponding to three distinct data types (DT1, DT2, and DT3).

At1 = {a1, a2, a3}
At2 = {a4, a5, a6, a7}
At3 = {a8}

Each attribute subset is mapped to one Archetype table, as shown in Figure 2.

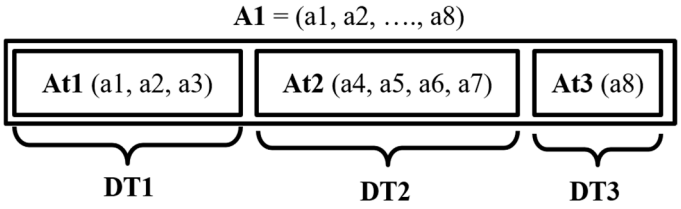


Figure 2. Data partitioning mechanism according to data type.

For example, blood pressure archetype constitutes five attributes, with two distinct data types, i.e., 'Quantity' (four attributes) and 'Text' (one attribute). Thus, two archetype tables are defined. In this example, the first archetype table (corresponding to 'Quantity' data type) contains four attributes only. The second archetype table contains one attribute (corresponding to 'Text' data type) only. The process is repeated for all archetypes involved in building the database of healthcare application.

2D EAV segregates the data based on data types (the second dimension) to support heterogeneity. In absence of partitioning based on data types, multiple value columns corresponding to different data types are required in each archetype table. In such a scenario, only one value column will contain the entry and all of the others will be null, resulting in sparseness. This motivated us to partition archetype table corresponding to data types (as reflected in Value column of underlying tables). Presently, we consider only four basic data types for the purpose of demonstration: Integer, String, Real, and Boolean. The set of data types can be enhanced by simply adding tables that are related to the desired data types.

Each archetype table is uniquely termed as a concatenated string of Archetype_ID, an underscore ("_"), and its corresponding Data_Type. openEHR provides a unique identification code to each archetype. However, this unique code is a long string that consumes more space (when stored repeatedly) and introduces a delay in data processing (needs to be de-serialized at time of access). Thus, for 2D EAV, the long string identification code allotted by openEHR is mapped to a new unique identification code through a mapping table. This mapped code serves as Archetype_ID.

Following a generic approach (EAV) for each archetype table, facilitates the addition of any number of attributes to existing information system, and the freedom from sparseness. As new

archetypes are added to the archetype repository, schema evolves automatically in 2D EAV storage system and requires no amendments to the definition of existing information system.

Partitioning of data into multiple tables (as per 2D EAV) results in many tables. Archetypes are advantageous because 10–20 basic archetypes are sufficient to build the core of a health application [62–64]. Around 100 archetypes can constitute a primary care electronic health record [63,64]. Similarly, around 2000 archetypes can constitute hospital EHRs, as compared to more than 400,000 active concepts in SNOMED CT [42]. So, the resultant 2D EAV storage can have thousands of tables. However, the data that needs to be accessed will be limited to a few tables. This happens because a patient data recordings will generally correspond to 10–20 basic archetypes (as required to build the core of a health application). Other recorded parameters (if any) will contain a null value (not stored in EAV).

In simple EAV, an exhaustive search to complete data might be required for extracting desired data. In contrast, 2D EAV restricts the search for desired data to the tables containing it. To enable this restriction, 2D EAV is complemented with metadata support as two tables, namely: Master table and Template table (as shown in Figure 3).

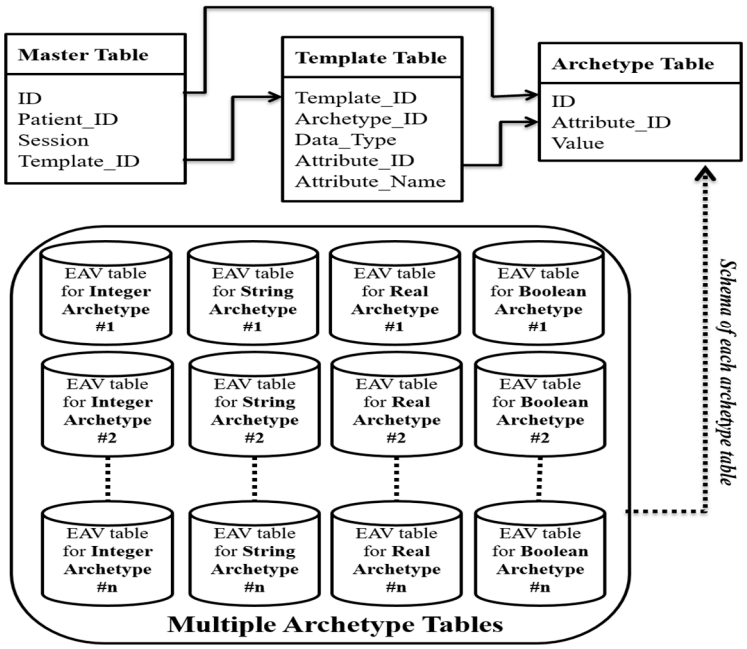


Figure 3. Modified entity attribute value storage model.

The Master table features four columns: ID, Patient_ID, Session, and Template_ID. The Template table features five columns: Template_ID, Archetype_ID, Data_Type, Attribute_ID, and Attribute_Name. 2D EAV stores a single EAV table as a collection of disjoint EAV partitions. Each partition (termed as Archetype table) corresponds to a distinct data type attributes of an archetype. Various archetype tables features three columns: ID, Attribute_ID, and Value.

- Master Table: The Master table is designed with the aim of uniquely identifying a patient's admittance to the hospital. The Master table follows the relational approach, since it stores data that contains no null values and have a fixed schema. ID column is the primary key of the Master table that stores auto-generated sequential numbers to identify each entry in the Master table uniquely. Patient_ID is unique for a particular patient, but it cannot serve as a candidate key in a Master table since a patient can have multiple admittances to a hospital, and thus, many entries in the Master table. Session is recorded to support a temporal behavior of standardized EHRs. Each entry in the Session column consists of a date (using ddmmyyyy format) followed by time

(hhmm), at which the underlying data is stored in the database. Template_ID reserves the ID of the template through which the data is stored in the database.

- Template Table: Every organization customizes their template as per their needs using Template Designer [65]. To identify each template uniquely, the Template_ID is maintained as it is. The Template table follows the EAV approach. However, the 'Attribute' column (of the EAV model) is defined using two columns (Archetype_ID and Data_Type) in the Template table to account for the fact that the EAV storage model is divided into two dimensions (archetype and data type). Each archetype constitutes of a set of attributes. To identify the particular attribute that belongs to a defined template, Attribute_ID is used. Attribute_Name specifies the name of the attribute corresponding to Attribute_ID.

Indexing aids in faster access of data. The ID column in a Master table serves as a primary key. To deal with queries that enquire about data related to some specific patient or template, a search on Patient_ID and Template_ID is performed (explained in the next following Section 2.2). As a result, two indexes (i.e., on Patient_ID and Template_ID) are created for the Master Table to facilitate faster data access. The Template table and Archetype tables have no single column primary key. A combination of Template_ID, Archetype_ID, and Attribute_ID forms a primary key for the Template table, whereas a combination of ID and Attribute_ID defines the primary key for every Archetype table. To facilitate faster execution of queries (elaborated on in the next following Section 2.2) that enquire about data related to some specific template, archetype, and attribute, three indexes (i.e., on Template_ID, Attribute_ID, and Archetype_ID) are defined for the Template table. Finally, the index for an ID attribute is defined for every Archetype table for rapid data access. In the absence of indexing, the whole table needs to be searched, which adds a time delay in accessing the required data. These indexes are managed by the DBMS through SQL (CREATE INDEX) command.

Building a 2D EAV storage system corresponding to various archetypes in an archetype repository requires to follow below listed steps.

- For each data type (of elements) in an archetype, we construct one EAV table (known as Archetype table). If a new version of an archetype is released with some new data type, a new table can be accommodated in the existing architecture; otherwise, existing tables can capture newer version elements.
- Basic data items represented by the archetype basic data type are mapped to the corresponding equivalent data type of the underlying RDBMS. Single-valued and Multi-valued attributes can be easily captured in the same Archetype table since, one row of Archetype table corresponds to one data entry.
- In case of a multi-valued attribute, a combination of ID, Attribute_ID, and Value defines the primary key for the underlying Archetype tables. Otherwise, the combination of ID, Attribute_ID serves as the primary key for the various Archetype table.
- ID is the primary key in Master table (metadata of 2D EAV). ID column in each Archetype table is declared as the foreign key that refers to ID column of Master table.
- Use of ID enables unique identification of each data instance. For rapid access, ID column of each Archetype table is indexed.
- An archetype can inherit knowledge from existing archetypes (as inheritance in Templates). This type of inheritance is maintained through a special attribute type, termed as Archetype slot. Archetype slot is supported in 2D EAV through metadata, i.e., Template table. A template derives knowledge from archetypes. A detail of archetypes participating in a template is stored in 'Template' table. All of the archetype slots are viewed as embedded within the same archetype. Thus, all of the details are stored within 'Template' table.
- Collection data items (such as CLUSTER, ITEM_TREE, ITEM_LIST) [36] are considered to be embedded within the archetype. Collection data items are flattened to store corresponding data. Thus, Archetype tables storing data can also hold collection data items (viewed as flattened).

- Aggregation relationship is supported in 2D EAV through metadata, i.e., template table. A template derives knowledge from archetypes. A detail of archetypes participating in a template is stored in 'Template' table. All the relationships of participating archetypes to other archetypes are viewed as embedded within the same template. Thus, all details are stored within 'Template' table.
- Each Archetype table is termed as a concatenated string of archetype name, an underscore and the underlying data type.
- openEHR defines a semantic path for each attribute within an archetype. This path provides a mechanism to uniquely identify an attribute within an archetype. In 2D EAV, each attribute of an archetype is mapped to a unique code through a manually designed mapping table. The use of attribute codes in place of long semantic paths help in achieving a better readability and saving storage space. Set of codes can be replicated for some other archetype. The use of replicated codes does not create any problem since the codes are unique within an archetype, and 2D EAV uses the combination of Archetype_ID and Attribute_ID to identify an element.

2.2. Evaluation of Performance

To give an abstract view to the user, our study proposes a query builder that interacts with metadata tables to dynamically present the user with the options to enquire data based on archetypes that are stored in archetype repository. A query builder is supported as an end user query interface. Users of this query interface can work without having any knowledge of the underlying query language. The presence of a query interface supports naive users. A blueprint of the query interface is shown in Figure 4.

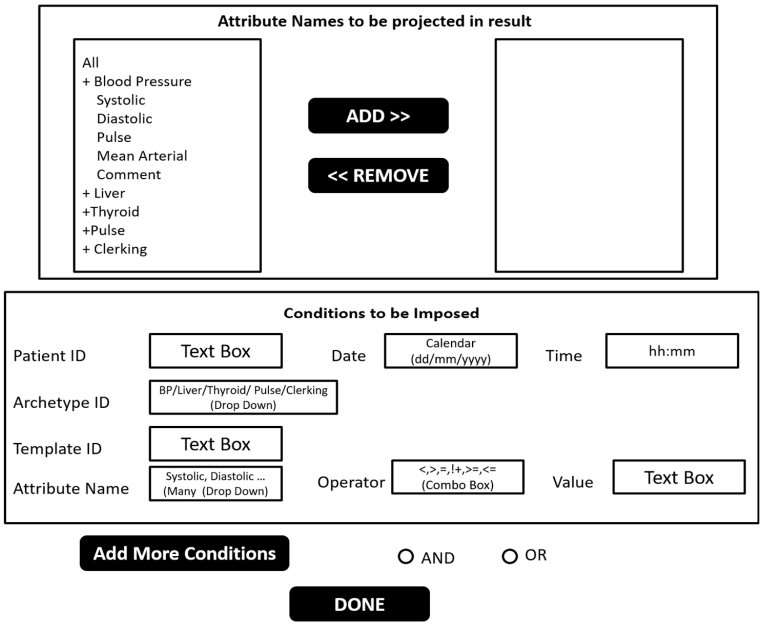


Figure 4. A Blueprint of query interface for the Two Dimensional Entity Attribute Value (2D EAV) storage system.

The upper portion implements the PROJECTION operation and the lower portion implements the SELECTION operation of relational algebra. The projection part provides the list of all the possible attributes of the EHRs. The attribute list is categorized based on medical concepts (archetypes) that are used in building healthcare application. Users can easily add or remove attributes of a choice for projection in output. The selection part enables the user to specify various criteria's for data to be presented as output. Selection criteria are categorized into five categories: namely patient-centric queries, attribute-centric queries, archetype-centric queries, template-centric queries, and hybrid queries.

Our query builder provides a potential solution to keyword search and drop-down menu. It can deal with thousands of attributes through the use of two drop down menus. It utilizes the co-occurring attributes information provided in archetypes. The first drop down menu lists various medical concepts (archetypes). Based on the item selection in the first menu, the second drop down menu is populated with the attributes list defined in the underlying archetype. Items in one drop down menu scale to an average of 10–50 attributes.

- Patient-Centric Query

Based on Patient_ID, Session, ID, and Template_ID (there may be multiple instances in the case of unknown Session), the ID of various encounters can be retrieved from the Master table. For each ID retrieved from the Master table, a search is performed in the Template table for matching Template_ID. Using Template_ID, a list of archetypes, underlying attributes, and data types can be extracted from the Template table.

Templates being used by a unit are very limited in number. For example, an eye care hospital will customize a template using eye-specific archetypes. Similarly, a cancer hospital will build a template using archetypes, such as, lung cancer, breast cancer, and other cancer related archetypes. Thus, the time to access template details from the Template table will be negligible. By concatenating Archetype_ID, an underscore ('_'), and Data_Type, a particular Archetype EAV table can be identified, where a search for the ID (from the Master table) and Attribute_ID (from the Template table) can be done for the desired value. This approach reduces searching time by a magnitude of 'n' (i.e., the total number of archetypes participating in EHRs) relative to an EAV approach. To search an element in EAV, an exhaustive search within the whole database is done. In contrast, for 2D EAV only the nth portion (sometimes even less) of the data (for patient specific queries) is searched.

- Attribute-Centric Query

Performing analytics (through population queries) demand attribute specific extraction. Using an attribute name (Attribute_Name), details related to attribute, such as its unique identification code (Attribute_ID), archetype identity (Archetype_ID), and data type (Data_Type) can be retrieved from the Template table. Using Archetype_ID and Data_Type, a particular Archetype EAV table can be identified where a search for the Attribute_ID can be done for the matching values. Adopting this approach reduces the time to search an attribute-centric query (for population queries) by a magnitude of 'n' (i.e., the total number of archetypes participating in EHRs), for the same reason as above.

- Archetype-Centric Query

There may be a scenario that needs to access attributes corresponding to an archetype (rather than individual attributes) for the purpose of analytics. In such a scenario, population queries are performed seeking details of all attributes corresponding to an archetype.

Irrespective of data type, all of the tables that are related to a particular archetype can be accessed instantly. The time complexity for accessing data related to an archetype is O(1) (i.e., a constant time) in the 2D EAV approach, which is quite fast. EAV approach scans the whole table to extract archetype specific rows, as exhibited in our previous research study [66].

- Template-Centric Query

Template centric is another scenario for performing analytics where data to be analyzed is entered using an underlying template. The Template_ID can be used in population query to identify a list of archetypes, their corresponding attributes, data types from the Template table, and ID from the Master table (for identifying the patients' list). Knowing ID is necessary since attributes that are related to one template can also participate in another template. To distinguish between the entries made through both templates, ID should be known. Using Archetype_ID and Data_Type, a particular Archetype table can be identified, where a search for the ID (from the Master table) and Attribute_ID (from the

Template table) can be done for the corresponding values. To the best of our knowledge, no approach has been proposed for template-centric queries till date.

- Hybrid Queries

There are many scenarios where queries are not only patient-centric, attribute-centric, archetype-centric, or template-centric; rather, any possible combination of the four categories defined above may apply. Specifying a hybrid query for data storedm as per the 2D EAV approach, imposes multiple conditions. The multiple conditions must be logically connected through some logical operator ('AND'/'OR'). The selection part can be used to specify a combination of a maximum of four (patient-, attribute-, archetype-, and template-related) conditions that are conjoined logically with each other via 'AND'. To specify more conditions, the "ADD MORE CONDITIONS" button can be used flexibly whenever required. The "ADD MORE CONDITIONS" button stores the current conditions specified in various input boxes in the system, and let the user specify the next condition in the same selection part. Each combination of conditions specified in one selection part is considered to be one sub-query by the query builder. Various sub-queries are connected logically through 'AND'/'OR' operator. Queries are executed using Algorithm given in Appendix A. The complete query support enables adherence of 2D EAV to data model definition (See Appendix B).

- Sparseness Evaluation

Applicability of 2D EAV in domains other than healthcare can be considered whenever a generic schema is required to deal with frequent evolution and a huge amount of sparseness.

As the amount of sparseness increases, the 2D EAV storage system performs more efficiently. The analysis is done considering the worst case scenario, where a single table is accommodating all of the attributes without any compression (rather than having a normalized structure). This analysis provides a rough estimation of the worst case scenario for 2D EAV in terms of minimum non-null density. In a real scenario, the non-null density should be much larger than evaluated here. A real time scenario that is suitable for 2D EAV is CNET product directory [67], for which recorded sparseness is 99.6%.

Let A_{tot} be the total number of attributes, R be the total number of entities (rows in the relational table), and A_{nn} be the average number of non-null entries per row.

- ➢ The total number of entries in the Relational table, $T_r = R \times A_{tot}$.
- ➢ The total number of entries in the Archetype table, $T_a = 3 \times (R \times A_{nn} - R)$ (as there are three entries corresponding to each non-null entry except the Patient_ID).
- ➢ The total number of entries in the Master table, $T_m = R \times 4$ (four entries corresponding to one row of the relational table).
- ➢ The total number of entries in the Template table is negligible, since the number of templates used is much smaller.

To adopt the 2D EAV storage system over the NSM approach, total entries in 2D EAV should be less than the total entries in the relational table.

So, $T_r > T_m + T_a$
$R \times A_{tot} > R \times 4 + 3 \times (R \times A_{nn} - R)$
$A_{tot} > 4 + 3 \times (A_{nn} - 1)$
$A_{tot} > 3 \times A_{nn} - 1$
$A_{tot} > 3 \times A_{nn}$ (neglecting -1 since $3 \times A_{nn} >> 1$)

Thus, the average number of non-null entries per row should be less than one third of the total attributes in the system, i.e., a minimum of 67% sparseness is appreciable. This analysis helps in the elimination of scenario where 2D EAV should not be adopted.

2.3. Environment

Our experiments compare performance of 2D EAV versus popular NoSQL solutions and basic EAV system for extracting standardized EHRs under various clinical query scenarios.

- Hardware and Software Configuration

All of the experiments are executed on a pair of 2.66 GHz dual-core Intel Xeon processors, with 16 GB RAM running Windows 8. Java version 1.8 has been used for implementation purpose. Query builder builds a SQL statement to be executed on 2D EAV.

- Dataset Collection

In total, 2.1 million records have been collected in accordance with the relational approach from three different sources, such as two private clinics, the UCI machine learning repository (Liver Disorder and Thyroid) [68,69], and self-synthesized, using knowledge that is available from reliable resources and internet (such as, if systolic pressure ranges between 120 to 139 and diastolic pressure ranges between 80 to 89, then the patient may have prehypertension [70]) for two medical concepts: namely, blood pressure and heart pulse. Data related to clerical tasks are not provided through Sources #1 and #2 due to ethical and security issues related to EHRs. Therefore, to facilitate the experiments, clerical data has been synthesized to simulate a realistic scenario. For the standardization of collected data, five openEHR archetypes (Clerking, Blood Pressure, Pulse, Thyroid, and Liver) have been adopted (See Appendix C).

- Storage Variants

We evaluated the performance of 2D EAV versus two alternatives: a system using the basic EAV model and MongoDB. Comparison of 2D EAV to NSM of RDBMS (following a good ER design) is not considered because the primary aim of this research is to provide a solution to sparseness, frequent evolution, faster query access, and ad-hoc query support altogether. Performance and adoption of 2D EAV in comparison to NSM is purely dependent upon the amount of sparseness in the dataset and frequent evolving nature of the underlying information system. To predict about the worst case scenario of 2D EAV in comparison to NSM, we performed an analysis in the previous section.

(1) 2D EAV: Our experiment version of 2D EAV is built on the top of PostgreSQL version 9.5, and our installation preserves the default configuration parameters. The query builder has been implanted using Java SE Development Kit 8. For 2D EAV, one Master table, one Template table, and 10 Archetype tables have been constructed. Among the 10 Archetypes tables, 2 tables (one real for QUANTITY and one string for TEXT) per archetype are included. Basic archetype data types can be mapped to SQL data types using the mapping rules suggested by Wang et al. [9].

(2) EAV: The standard EAV model has been extended for experiments to accommodate heterogeneity (through columns 'Value_Real', and 'Value_String'), temporal behavior (through column 'Session') and support for template-centric queries (through columns 'Template_ID', and Archetype_ID). Thus, one EAV table is constituted by six columns (Patient_ID, Template_ID, Archetype_ID, Attribute_ID, Value_Real, Value_String, and Session). For query support of EAV, one metadata table (See Appendix D) consists of four columns (Archetype_ID, Attribute_ID, Attribute_Name, and Data_Type). Indexes are defined on Patient_ID Template_ID, and Archetype_ID columns (for the EAV table), and on Archetype_ID column (for the metadata table) for faster execution of queries. EAV system is built also built on the top of PostgreSQL version 9.5.

(3) MongoDB: The most popular NoSQL database system as per db ranking system is MongoDB [71]. It provides same flexibility as EAV. Hundreds of well-known production systems uses MongoDB [72]. It is a document oriented NoSQL database that inherently store data as key-value pairs (key being the combination of Entity and Attribute). Thus, we choose MonogoDB to evaluate the performance of proposed approach i.e., 2D EAV. The default configuration parameters of MongoDB has been preserved during experimentation.

- Queries Formulated

In total, forty queries have been formulated for experiments while considering eight query scenarios in the health domain (five different queries per scenario). The eight query scenarios are (1) Patient-centric only; (2) Attribute-centric only; (3) Archetype-centric only; (4) Template-centric only; (5) Hybrid (Any 2); (6) Hybrid (Any 3); (7) Hybrid (All 4); and (8) Hybrid (2 Same) (for complete query set See Appendix E). 'Any 2' and 'Any 3' considers permutation/combination of any group of two or three criteria among patient, attribute, archetype, and template, respectively. The 'All 4' category considers permutation of patient, attribute, archetype, or template criteria. The '2 Same' category identifies the combination of patient, attribute, archetype, or template with itself. Queries for 2D EAV are executed using query builder by passing the desired parameters. Equivalent SQL queries (for EAV) and query operations (for MongoDB) has been composed manually to perform experiment. A sample set of query formulated for 2D EAV (SQL build by query builder), EAV, and MongoDB is given in Appendix F.

3. Result

3.1. Efficiency of Storage

Transformation of the collected 2.1 million relational records in EAV to 2D EAV structure produced 12.9 million records. Table 2 presents the load time and storage acquired by various storage variants. 2D EAV loads data faster than MongoDB and EAV. Although EAV and 2D EAV has been implemented on the top of PostgreSQL, they took different load time since the size of EAV dataset is larger than the size of 2D EAV dataset. Reason behind the larger size of EAV is the presence of multiple 'Value' columns (for different data types). However, the load time for EAV will get reduced if we partition the EAV table, as per data type. MongoDB converts the dataset into BSON objects. Thus, there is an increase in the overall load time, in case of MongoDB. Whereas, storage has been observed to be smallest in case of MongoDB.

Table 2. Load time and storage acquired by various storage variants.

S.No.	Storage Variant	Load Time (Milliseconds)	Size (MB)
1	EAV	6462	98.6
2	MongoDB	60,000	28.5
3	2D EAV	5800	63.2

3.2. Performance

The average access time of all test queries (five queries per query scenarios) is considered to represent the comparative results. The result of the queries executed on 12.9 million records, as per EAV and 2D EAV (shown in Table 3 and Figure 5), clearly indicates that the 2D EAV approach performs better than the EAV approach.

Table 3. Comparing 2D EAV with EAV and MongoDB.

S.No.	Query Category	Data Access Time (Milliseconds)		
		EAV	MongoDB	2D EAV
1	Patient Centric	873.8	123.6	60.2
2	Attribute Centric	941.4	274.2	176
3	Archetype Centric	3280	1446	1315.8
4	Template Centric	875.2	168.4	86
5	Hybrid (Any 2)	847	150.4	38
6	Hybrid (Any 3)	1045.8	159.4	43.2
7	Hybrid (All 4)	981.6	121.8	28.4
8	Hybrid (2 Same)	1013.6	299.8	137.4

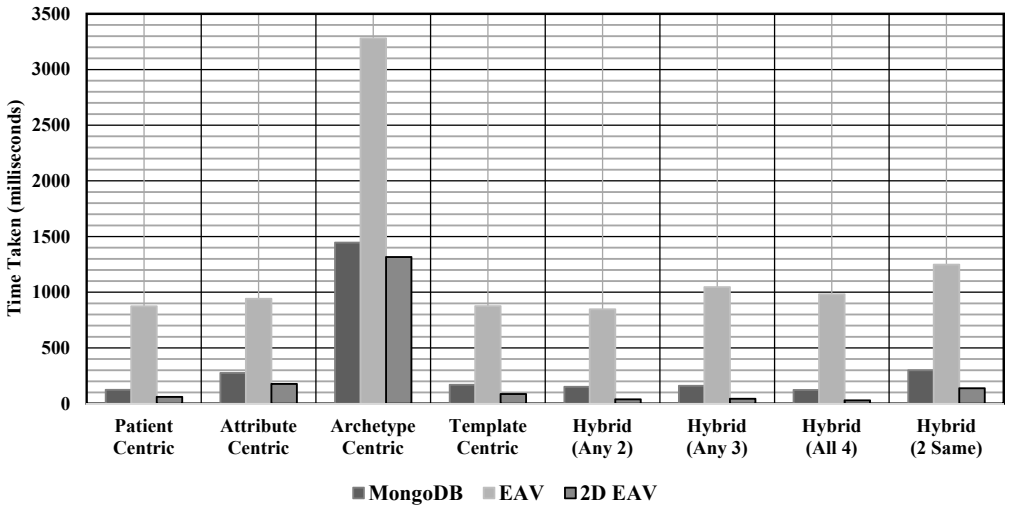
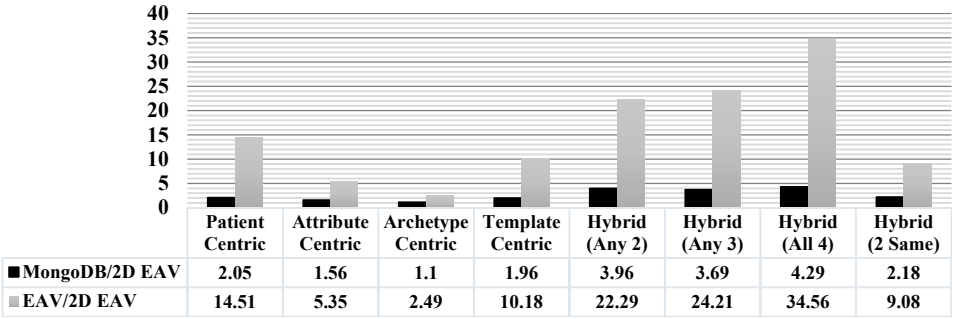


Figure 5. Experimental results for time taken to access standardized EHRs.

For predicting the efficiency of 2D EAV with respect to MongoDB and EAV, we analyzed the results presented in Table 3. We considered the ratio of time taken by MongoDB and EAV to the time taken 2D EAV as a measure of efficiency. This ratio provides a factor by which 2D EAV performs better than MongoDB and EAV. Results of the ratio calculated for various categories is presented in Figure 6. From the results presented in Figure 6, it has been calculated that 2D EAV performs better than MongoDB and EAV by a factor of 2.5 and 15.3, with a standard deviation of 1.2 and 10.8, respectively.



	Patient Centric	Attribute Centric	Archetype Centric	Template Centric	Hybrid (Any 2)	Hybrid (Any 3)	Hybrid (All 4)	Hybrid (2 Same)
MongoDB/2D EAV	2.05	1.56	1.1	1.96	3.96	3.69	4.29	2.18
EAV/2D EAV	14.51	5.35	2.49	10.18	22.29	24.21	34.56	9.08

Figure 6. Efficiency results of 2D EAV.

To further analyze the performance variation of 2D EAV with respect to EAV, we perform experiments considering selection and projection operation on the different configurations of datasets. Various dataset configurations consider different number of tuples (rows) and a different number participating archetypes.

The two operations (selection and projection) are chosen when considering the fact that dataset is mostly accessed to perform patient-specific queries (extracting multiple rows i.e., selection) or population oriented queries (extracting multiple columns i.e., projection). Results of experiments performed are shown in Figures 7 and 8 (where 'na' presents the number of participating archetypes).

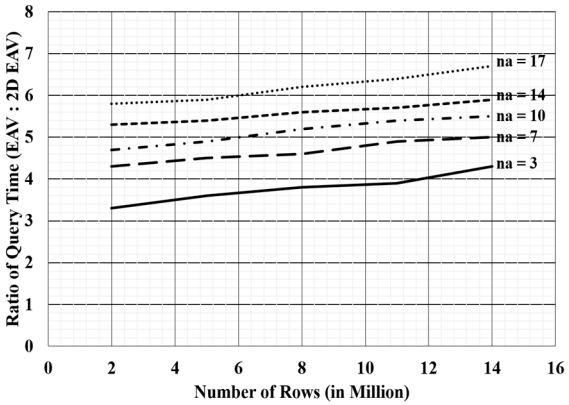


Figure 7. Selection operation results.

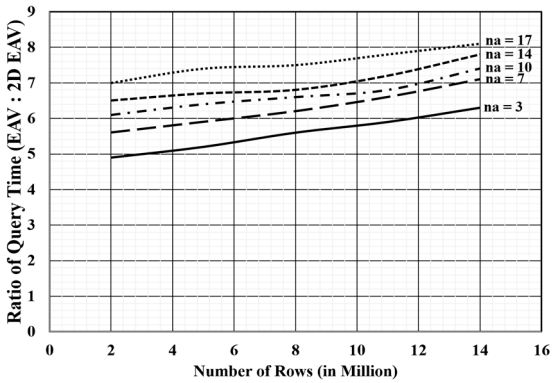


Figure 8. Projection operation results.

4. Discussion

4.1. Advantages of 2D EAV

1. Faster Data Retrieval: 2D EAV stores data in various partitions and provides metadata to communicate with these partitions flawlessly. This restricts the search space of desired data to a few partitions, and thus, improves the speed of data retrieval.
2. Adaptability to Other Domains: In this paper, we highlight the use of 2D EAV specifically for EHRs; however, an example process of creating an archetype for a subject schedule and for hotels is discussed in [73]. Once the archetype system is ready, 2D EAV can be easily adopted for the underlying domain by simply renaming the 'patient_id' as 'entity_id'. All other semantics of 2D EAV will remain unchanged for the desired domain. We have mainly focused on standardized EHRs due to the availability of archetypes for the healthcare domain, and the need for generic storage for EHRs.

4.2. Comparison with Other Studies

2D EAV excels other persistence approaches, as detailed below.

- *ARM:* ARM maps each archetype to a relational table and 2D EAV maps each archetype to a distinct EAV table that is further categorized based on data type. Loss of stability (since, schema is

not built using RM) in 2D EAV is compensated with the generic behavior of capturing any future evolution without modifying the existing system. ARM requires prior knowledge of identification attributes and frequently enquired data items for building indexing support. ARM also requires building a separate table for supporting multiple occurrences of collection data structures. 2D EAV in contrast to ARM requires no prior knowledge of data items. It does not construct separate tables for multiple occurrences of collection data structure.

- *Node+Path (using BLOB):* In 2D EAV, unique archetype and attribute names are coded to identify various hierarchies, rather using BLOB. Thus, it requires reduced storage and provides faster data access.
- *EAV:* In contrast to EAV, 2D EAV is focused on improving the access speed of standardized EHRs, rather than dealing with a complex query structure. It overcomes complex query difficulties through an efficient user interface.

4.3. Limitations

1. Complexity Due to Multiple Tables

In a real scenario, the number of tables can easily reach hundreds or thousands. Huge number of tables can cause complexity to the system in terms of managing inter-relationship and costly access (due to JOINing of multiple tables). However, 2D EAV reduces the complexity in the following ways.

- *Managing Inter-relationship:* The inter-relationship among the tables is stored in Template table of 2D EAV. It helps in managing this complexity. Template table defines the set of attributes corresponding to each template, and thus, inter-related archetypes. For instance, there are 'm' hospitals that are involved in the information system; each having their own customized templates and a template on an average constitutes 'n' attributes. Thus, template table will contain 'm $\times$ n' rows. If a new template is introduced into the existing system with 'z' attributes, then 'z' rows are added to existing 'm $\times$ n' rows of template table giving a total of 'm $\times$ n + z' rows. In contrast to ARM approach, 2D EAV handles the introduction of a new table by simply inserting some rows in the metadata table, eliminating the need of manually defining the inter-relationship of existing archetypes with a newly introduced archetype.
- *Data Accessibility Cost:* EHRs are extracted for either clinical purpose or research purpose. A clinical activity normally involves the extraction of patient specific data to provide care services. Query interface of 2D EAV produces output in the form of EAV table. The results obtained are visually more appropriate for doctors due to document-like view of medical records. Thus, resultant records (following EAV) need not be self-JOINed to be presented in accordance with a relational approach. When EHRs are used for research objectives (such as finding the effect of some drug or growth rate of any disease, etc.), epidemiological queries are involved for the extraction of records. Such queries need not be answered in real time [1], and thus, a delay in data is acceptable. However, 2D EAV access data more quickly than EAV.

2. Versioning of Archetypes

The management of different versions of the archetypes along time can be envisioned through metadata tables. 2D EAV is designed when considering templates. 2D EAV stores a unique identification of each template in a column, termed as Template_ID (in Template table). Also, each entry that is made in the system is uniquely identified by the ID column (primary key of MASTER table). A combination of Template_ID and ID helps in uniquely identifying the data that corresponds to a specific version of the archetype. A template inherits knowledge for one or many archetypes according to local healthcare application requirement. It is assumed that the template corresponds to a form that is presented to end user for data entry. Data is thus organized in 2D EAV storage system at the backend. As knowledge evolves, existing archetypes can be redefined as per the new knowledge. The redefined archetypes are

released as a new version over existing one. A template might inherit knowledge from the previous or new version based on the availability of the corresponding archetype in the local archetype repository. It seems to be impractical that a template inherits knowledge from both previous and new version of an archetype.

5. Conclusions

The study proposed an EAV style modeling approach, termed 2D EAV. The EAV provides a generic structure. It deals with sparseness but lacks in supporting heterogeneity, and quick data access. 2D EAV is an extension of the existing EAV approach that overcomes search inefficiency and provides a mechanism for enabling a template-centric query. Experiments have been performed based on various categories of a query (patient-centric queries, attribute-centric queries, archetype-centric queries, template-centric queries, and hybrid queries). Till date, no other approach has offered support for template-centric queries. Current research provides a mechanism to retrieve template-oriented data for standardized EHR databases.

Results illustrate that performance of 2D EAV is enhanced by a factor of 2.5 in case of Mongo DB, and by a factor of 15.3 in case of EAV. The corresponding standard deviation calculated is 1.2 and 10.8, respectively. The performance of the 2D EAV in comparison to the NSM approach is dependent upon the amount of sparseness. We analyze the scenario where 2D EAV is not preferable in terms of non-null density. The proposed solution will benefit users in handling standard-based heterogeneous data requiring high search efficiency. A user interface has been developed to support ease of complex ad-hoc query. It provides various query parameters to the proposed query builder for building a query corresponding to the desired output. The proposed user interface enables skilled and semi-skilled database users (such as doctors, nurses, and patients) to query EHRs data without any knowledge of the underlying storage system and query language. Applicability of 2D EAV can be extended from standardized EHRs to other domains as well by utilizing a mapping mechanism.

Author Contributions: Shivani Batra and Shelly Sachdeva conceived, formulated and verified the 2D EAV data model. Shivani Batra, Shelly Sachdeva and Subhash Bhalla analyzed the results, proof read the manuscript and corrected the manuscript. Shelly Sachdeva and Subhash Bhalla contributed on experience about openEHR architecture. All authors have read and approved the final manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Algorithm A1

Input: Master Table (M), Template Table (T), set of Archetype Tables (A), Patient_ID (P), Session (S), set of Attribute_Name (N) to be projected, Condition (Attribute Name: AN, Operator: OP and Value: V), Archetype_ID (I) and Template_ID (TI)
Output: EAV table containing enquired data (T_{out}).
//$TEMP_I$, $TEMP_{TI}$, $TEMP_{AT}$, $TEMP_{AR}$, $TEMP_{TI}$ and $TEMP_F$ are temporary tables. Two Variables $Name_T$ and AI for storing name of table and Attribute_ID for AN

(1) For each sub-query follow Step 2 to Step 4
(2) Call **FUNCTION IDS** to identify list of desired Template_ID.
(3) Call **FUNCTION ARCHTAB** to identify archetype table that needs to be accessed for required data extraction.
(4) Call **FUNCTION EXTRACT** to extract desired records from the archetypes table identified in Step 3.
(5) Following the precedence rules (AND before OR) merge results of sub-queries as INERSECTION for 'AND' and UNION for 'OR'.

FUNCTION IDS
Input: Patient_ID (P), Session (S) and Template_ID (TI)
Output: List of ID ($TEMP_I$) to be in output and List of Template_ID ($TEMP_{TI}$).
if (P != NULL) **then**
  if (S != NULL) **then**
    if (TI != NULL) **then**
      foreach record r ϵ M **do**
        if (r.Patient_ID == P) && (r.Session == S) && (r.Template_ID == TI) **then**
        APPEND r.ID to $TEMP_I$ and r.Template_ID to $TEMP_{TI}$;
  else foreach record r ϵ M **do**
        if (r.Patient_ID == P) && (r.Session == S) **then**
        APPEND r.ID to $TEMP_I$ and r.Template_ID to $TEMP_{TI}$;
  else if (TI != NULL) **then**
      foreach record r ϵ M **do**
        if (r.Patient_ID == P) && (r.Template_ID == TI) **then**
        APPEND r.ID to $TEMP_I$ and r.Template_ID to $TEMP_{TI}$;
  else foreach record r ϵ M **do**
        if (r.Patient_ID == P) **then**
        APPEND r.ID to $TEMP_I$ and r.Template_ID to $TEMP_{TI}$;
else if (S != NULL) **then**
  if (TI != NULL) **then**
    foreach record r ϵ M **do**
      if (r.Session == S) && (r.Template_ID == TI) **then**
      APPEND r.ID to $TEMP_I$ and r.Template_ID to $TEMP_{TI}$;
else foreach record r ϵ M **do**
      if (r.Session == S) **then**
      APPEND r.ID to $TEMP_I$ and r.Template_ID to $TEMP_{TI}$;
else if (TI != NULL) **then**
  foreach record r ϵ M **do**
    if (r.Template_ID == TI) **then**
    APPEND r.ID to $TEMP_I$ and r.Template_ID to $TEMP_{TI}$;
  else $TEMP_I$ =NULL and $TEMP_{TI}$ =NULL;

FUNCTION ARCHTAB
Input: List of ID ($TEMP_I$) and List of Template ID ($TEMP_{TI}$).
Output: List of archetype tables containing desired data ($TEMP_F$), Attribute_ID corresponding to attribute in condition part (AI).
foreach record q ϵ T **do**
if (q.Attribute_Name IN N) **then**
APPEND **CONCAT**(q.Archetype_ID, "_", q.data_Type) to $TEMP_{AT}$;
if (q.Archetype_ID == I) **then**
APPEND **CONCAT**(q.Archetype_ID, "_", q.data_Type) to $TEMP_{AR}$;
if (q.Template_ID ϵ $TEMP_{TI}$) **then**
APPEND **CONCAT**(q.Archetype_ID, "_", q.data_Type) to $TEMP_T$;
if (q.Attribute_Name ==AN) **then**
$Name_T$ = **CONCAT**(q.Archetype_ID, "_", q.data_Type) ;
  AI = Attribute_ID;
if ($TEMP_{AT}$!= NULL) **then**
  if ($TEMP_{AR}$!= NULL) **then**
    if ($TEMP_T$!= NULL) **then**
    TEMPF = **INTERSECTION**($TEMP_{AT}$, $TEMP_{AR}$, $TEMP_T$);
    else $TEMP_F$ **= INTERSECTION** ($TEMP_{AT}$, $TEMP_{AR}$);
  else if ($TEMP_T$!= NULL) **then**
    $TEMP_F$ **= INTERSECTION** ($TEMP_{AT}$, $TEMP_T$);
    else $TEMP_F$ **=** $TEMP_{AT}$;
else if ($TEMP_{AR}$!= NULL) **then**
      if ($TEMP_T$!= NULL) **then**
        $TEMP_F$ **= INTERSECTION** ($TEMP_{AR}$, $TEMP_T$);
      else $TEMP_F$ **=** $TEMP_{AR}$;
else if ($TEMP_T$!= NULL) **then**
      $TEMP_F$ **=** $TEMP_T$;
else $TEMP_F$ **=** NULL;

```
FUNCTION EXTRACT
Input: List of ID (TEMP_I) and List of archetype tables (TEMP_F), Attribute_ID (AI)
Output: EAV table containing enquired data (T_out).
if (TEMP_I == NULL) then
  foreach record u of table name stored in Name_T do
    if (u.Attribute_ID = AI) && (u.Value OP V) then
    APPEND u.ID to TEMP_I;
if (TEMP_I != NULL) then
  if (TEMP_F != NULL) then
    foreach table X whose name lies IN TEMP_F
      foreach record s ϵ X do
        if (s.ID IN TEMP_I ) && (s.Attribute_ID IN N) then
        APPEND s to T_out;
  else foreach table X whose name lies IN TEMP_F do
      foreach record s ϵ X do
        if (s.ID IN TEMP_I ) then
        APPEND s to T_out;
else if (TEMP_F != NULL) then
      foreach table X whose name lies IN TEMP_F
        foreach record s ϵ X do
          if (s.Attribute_ID IN N) then
          APPEND s to T_out;
  else foreach table X whose name lies IN TEMP_F
        foreach record s ϵ X do
          APPEND s to T_out;
```

Appendix B

A data model consists of three components: a set of data structure types, a set of operators or inference rules, and a set of integrity rules given by Codd. 2D EAV adheres to this definition.

Set of data structures types: Currently, 2D EAV is implemented for four basic data types (Integer, String, Real, and Boolean) for the purpose of demonstration. The set of data types can be easily extended by incorporating archetype tables corresponding to a desired data type. 2D EAV being specifically designed for storing standardized EHRs and is capable of supporting basic data types defined for archetypes. Basic archetype data types can be mapped to SQL data types using the mapping rules as shown in Table A1.

Table A1. Archetype basic data types and mapping rules [9].

Data Type	Field	Field Data Type	SQL Type
CodePhrase	codeString	String	NVARCHAR
DvBoolean	Value	Boolean	INTEGER
DvCodedText	definingCode	CodePhrase	#
DvCount	magnitude	Integer	INTEGER
DvDateTime	Value	String	NVARCHAR
DvEHRURI	Value	URI	NVARCHAR
DvIdentifier	Id	String	NVARCHAR
DvMultimedia	uri	DvURI	#
DvProportion	precision	Integer	INTEGER
DvQuantity	magnitude	Double	FLOAT
	Units	String	NVARCHAR
DvText	Value	String	NVARCHAR
DvURI	Value	URI	NVARCHAR
GenricID	Value	String	NVARCHAR
	Name	String	NVARCHAR
Link	Target	DvEHRURI	#

"#" represents a non-preliminary data type and "NVARCHAR" represents character array of length "N".

Set of operators: 2D EAV is well defined for relational algebra operations (SELECT and PROJECT using Query builder). In addition, 2D EAV exploits the three additional relational algebra operators (CARTESIAN PRODUCT, UNION and MINUS) available in the underlying RDBMS. To redefine "CARTESIAN PRODUCT, UNION and MINUS" operators for 2D EAV (that produce result same as in case of relational model), an equivalent query can be build using tables aliasing and logical operators (such as, 'AND', 'OR', and 'NOT').

Set of integrity rules: The three most popular integrity rules (entity integrity, domain integrity, and referential integrity) specified for the conventional relational model are also followed in the case of 2D EAV.

1. Entity Integrity: Every record stored in the 2D EAV database is uniquely identified by a combination of ID and Attribute_ID. In other words, the ID and Attribute_ID columns in Archetype Tables compose a PRIMARY KEY. A primary attribute can never be NULL, since 2D EAV is built for storing non-null values.
2. Domain Integrity: 2D EAV is defined for archetype based system. Domain constraints are well defined in archetypes. Any data entered in the system conforms to the semantics of constraints specified in AM and RM.
3. Referential Integrity: References are made from the Master table to the various Archetype tables, where the ID column serves as the primary key.

Appendix C. Archetypes Used in Data Collection

Snapshots of various archetypes (only the data part) being used for data collection are shown in Figure A1.

Various attributes contributing to different archetypes are as follows:

- Four TEXT attributes, eleven QUANTITY attributes, one MULTIMEDIA attribute, and two CLUSTERS are present in openEHR openEHR-EHRs-OBSERVATION.lab_test-liver_function.v1archetype.
- Four TEXT attributes, six QUANTITY attributes, one MULTIMEDIA attribute, and two CLUSTERS are present in openEHR-EHRs-OBSERVATION.lab_test-thyroid.v1 archetype.
- One TEXT attribute, and four QUANTITY attributes are present in openEHR-EHRs-OBSERVATION. blood_pressure.v1 archetype.
- Five TEXT attributes, and one QUANTITY attribute is present in openEHR-EHRs-OBSERVATION. pulse.v1 archetype.
- Six TEXT attributes, and one COMPOSITION attribute (containing 3 TEXT attributes) is present in openEHR-EHRs-OBSERVATION.soap_clerking8.v8 archetype.

Authors are not considering (for liver and thyroid archetype) multimedia representation attributes and cluster attributes (specimen detail and per-result annotation) due to non-availability of multimedia data and device information in the dataset collected.

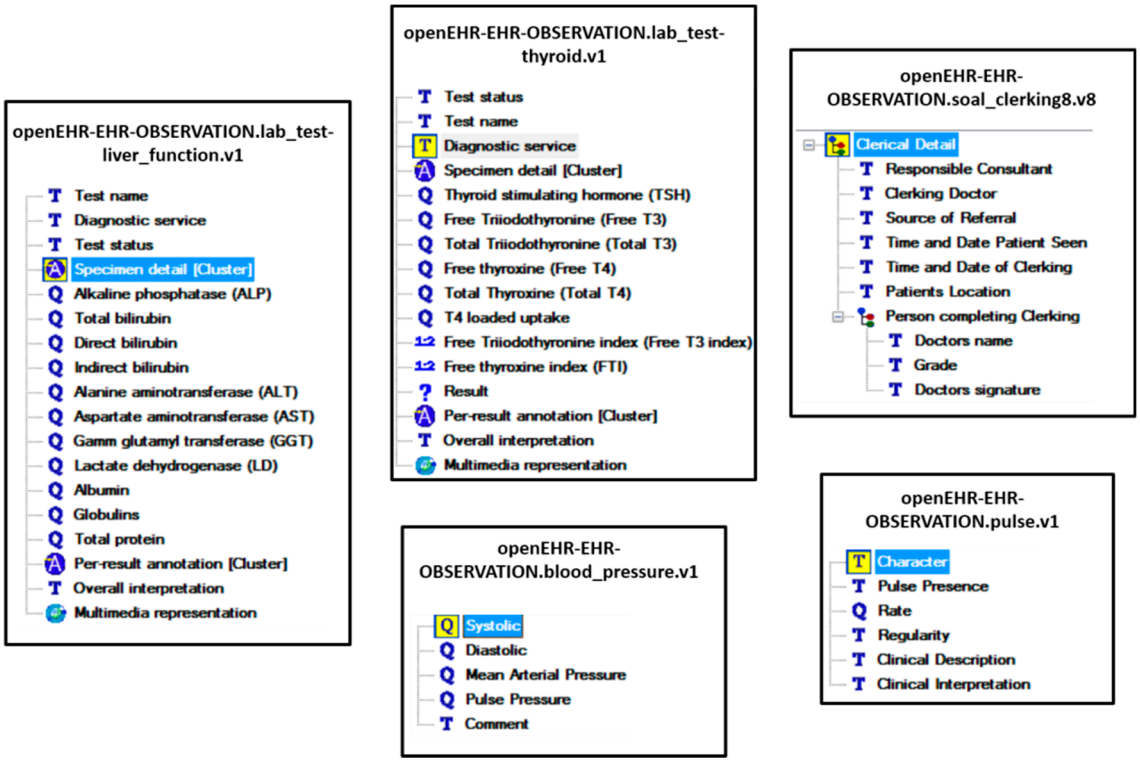


Figure A1. The data part of various archetypes used for data collection (T signifies TEXT (textual) data type and Q signifies QUANTITY (quantifiable) data type defined in RM of openEHR).

Appendix D. Modified EAV for Experimentation

The EAV schema (consisting of three columns: Entity, Attribute and Value) is extended as shown in Figure A2. The change to the basic EAV approach is done to accommodate heterogeneity (through columns 'Value_Real', and 'Value_String'), temporal behavior (through column 'Session') and support for template-centric queries (through columns 'Template_ID', and Archetype_ID). Corresponding to the modified EAV schema, the metadata table is also updated (as shown in Figure A2).

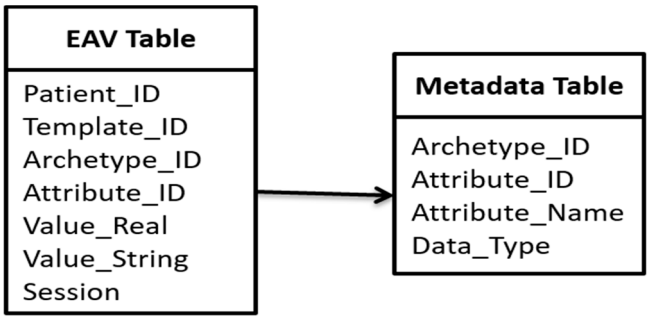


Figure A2. EAV storage schema for experimentation.

Appendix E

In total forty queries are formulated considering eight categories for the purpose of experiment as presented in Table A2.

Table A2. Query set for experiment.

Type	#	Query Description
Patient-centric	Q1	List records of patient with Patient_ID 1004
	Q2	List records of patient with Patient_ID 924
	Q3	List records of patient with Patient_ID 14306
	Q4	List records of patient with Patient_ID 14
	Q5	List records of patient with Patient_ID 5126
Attribute-centric	Q6	List all stored values of systolic pressure
	Q7	List all stored values of diastolic pressure
	Q8	List all stored values of Total Thyroxine
	Q9	List all stored values of T4 loaded uptake
	Q10	List all stored values of Albumin
Archetype-centric	Q11	List all blood pressure archetype records
	Q12	List all liver archetype records
	Q13	List all thyroid archetype records
	Q14	List all liver and thyroid archetype records
	Q15	List all blood pressure and liver archetype records
Template-centric	Q16	List all records having Template_ID 18
	Q17	List all records having Template_ID 25
	Q18	List all records having Template_ID 1
	Q19	List all records having Template_ID 13
	Q20	List all records having Template_ID 8
Hybrid (Patient + Attribute)	Q21	List Thyroid stimulating hormone of the patient with Patient_ID 927
	Q22	List Systolic pressure of the patient with Patient_ID 15003
	Q23	List Diastolic pressure of the patient with Patient_ID 14969
	Q24	List Alkaline Phosphatase of the patient with Patient_ID 5
	Q25	List Test Name of the patient with Patient_ID 556
Hybrid (Patient + Archetype + Template)	Q26	List all Blood Pressure archetype records of the patient with Patient_ID 14987 and Template 25
	Q27	List all Blood Pressure archetype records of the patient with Patient_ID 15384 and Template 25
	Q28	List all Thyroid archetype records of the patient with Patient_ID 6209 and Template 10
	Q29	List all Liver archetype records of the patient with Patient_ID 590 and Template 1
	Q30	List all Liver archetype records of the patient with Patient_ID 561 and Template 3
Hybrid (Patient + Attribute + Archetype+ Template)	Q31	List the Alkaline Phosphatase for Template_ID 1, Archetype Liver and Patient_ID 606
	Q32	List the Globulins for Template_ID 3, Archetype Liver and Patient_ID 432
	Q33	List the Result for Template_ID 17, Archetype Thyroid and Patient_ID 11833
	Q34	List the Comment for Template_ID 25, Archetype Blood Pressure and Patient_ID 14938
	Q35	List the Comment for Template_ID 25, Archetype Blood Pressure and Patient_ID 15008
Hybrid (2 Same)	Q36	List the systolic pressure of patients with diastolic pressure >90
	Q37	List the systolic and diastolic pressure of patients with comment as Hypotension
	Q38	List records belonging to Blood Pressure archetype or Liver archetype
	Q39	List records belonging to Template_ID 1 or Template_ID 3
	Q40	List records of patients with Patient_ID > 14942 and Patient _ID < 15293

Appendix F. Sample Query Set for Experiment

The sample queries with its equivalent query syntax of NoSQL (i.e., MongoDB in our case) and SQL (for EAV and 2D EAV) for various query categories used in experimentation is presented in Table A3.

Table A3. Sample query syntax.

Query No.	MongoDB	EAV	2D EAV
Q1	db.mycol.find({"Patient_ID" :1004})	SELECT Patient_ID, Attribute_ID, Value_Real, Value_String FROM EAV WHERE Patient_ID = 1004;	SELECT Thyroid_Text.ATTRIBUTE_ID, Thyroid_Text.VALUE FROM Thyroid_Text WHERE Thyroid_Text.PATIENT_ID IN (SELECT MasterTable.ID FROM MasterTable WHERE MasterTable.Patient_ID = 1004) UNION SELECT Thyroid_Numeric.ATTRIBUTE_ID CAST (Thyroid_Numeric.VALUEAS character(8)) FROM Thyroid_Numeric WHERE Thyroid_Numeric.PATIENT_ID IN (SELECT MasterTable.ID FROM MasterTable WHERE MasterTable.Patient_ID = 1004);

Table A3. *Cont.*

Query No.	MongoDB	EAV	2D EAV
Q6	db.mycol.find({}, {"Systolic" :1})	SELECT Patient_ID, Attribute_ID, Value_Real, Value_String FROM EAV, Metadata WHERE Attribute_ID = Metadata.Attribute_ID AND Metadata.Attribute_Name = 'Systolic';	SELECT BP_Numeric.PATIENT_ID, BP_Numeric.ATTRIBUTE_ID, BP_Numeric.VALUE FROM BP_Numeric WHERE BP_Numeric.ATTRIBUTE_ID IN (SELECT TemplateTable.Attribute_ID FROM TemplateTable WHERE TemplateTable.Attribute_Name = 'Systolic');
Q11	db.mycol.find({"TestName": "BP"}, {"Systolic" :1, "Diastolic" :1, "MeanArterial" :1, "PulsePressure":1, "Comment":1 })	SELECT Patient_ID, Attribute_ID, Value_Real, Value_String FROM EAV WHERE Archetype_ID = 'BP';	SELECT BP_Numeric.PATIENT_ID, BP_Numeric.ATTRIBUTE_ID, BP_Numeric.VALUE FROM BP_Numeric UNION SELECT BP_Text.PATIENT_ID, BP_Text.ATTRIBUTE_ID, BP_Text.VALUE FROM BP_Text;
Q16	db.mycol.find({TestName :"'Thyroid'"}, {" Patient_ID" :1, "Result" :1 })	SELECT Patient_ID, Attribute_ID, Value_Real, Value_String FROM EAV WHERE Template_ID = 18;	SELECT Thyroid_Text.PATIENT_ID, Thyroid_Text.ATTRIBUTE_ID, Thyroid_Text.VALUE FROM Thyroid_Text WHERE Thyroid_Text.PATIENT_ID IN (SELECT MasterTable.ID FROM MasterTable WHERE MasterTable.Template_ID = '18');
Q21	db.mycol.find({"Patient_ID" : 927}, {"Patient_ID" :1, "ThyroidStimulaingHormone" :1 })	SELECT Patient_ID, Attribute_ID, Value_Real, Value_String FROM EAV, Metadata WHERE Attribute_ID = Metadata.Attribute_ID AND Patient_ID = 927 AND Metadata.Attribute_Name = 'Thyroid stimulaing hormone';	SELECT Thyroid_Numeric.PATIENT_ID, Thyroid_Numeric.ATTRIBUTE_ID, Thyroid_Numeric.VALUE FROM Thyroid_Numeric WHERE Thyroid_Numeric.ATTRIBUTE_ID IN (SELECT TemplateTable.Attribute_ID FROM TemplateTable WHERE TemplateTable.Attribute_Name = 'Thyroid stimulating hormone') AND Thyroid_Numeric.PATIENT_ID IN (SELECT MasterTable.ID FROM MasterTable WHERE MasterTable.Patient_ID = 927);
Q26	db.mycol.find({"TestName": "BP", "Patient_ID" : 14987}, {"Patient_ID":1, "Systolic" :1, "Diastolic" :1, "Comment":1 })	SELECT Patient_ID, Attribute_ID, Value_Real, Value_String FROM EAV WHERE Archetype_ID = 'BP' AND Patient_ID = 14987 AND Template_ID = 25;	SELECT BP_Numeric.PATIENT_ID, BP_Numeric.ATTRIBUTE_ID, BP_Numeric.VALUE FROM BP_Numeric WHERE BP_Numeric.PATIENT_ID IN (SELECT MasterTable.ID FROM MasterTable WHERE MasterTable.Patient_ID = 14987 AND MasterTable.Template_ID = '25') UNION SELECT BP_Text.PATIENT_ID, BP_Text.ATTRIBUTE_ID, BP_Text.VALUE FROM BP_Text WHERE BP_Text.PATIENT_ID IN (SELECT MasterTable.ID FROM MasterTable WHERE MasterTable.Patient_ID = 14987 AND MasterTable.Template_ID = '25');
Q31	db.mycol.find({"TestName": "Liver", "Patient_ID" : 606}, {"AlkalinePhosphatase":1})	SELECT Patient_ID, Attribute_ID, Value_Real, Value_String FROM EAV, Metadata WHERE Attribute_ID = Metadata.Attribute_ID AND Archetype_ID = 'Liver' AND Patient_ID = 601 AND Template_ID = 1 AND Metadata.Attribute_Name = 'Alkaline Phosphatase';	SELECT Liver_Numeric.PATIENT_ID, Liver_Numeric.ATTRIBUTE_ID, Liver_Numeric.VALUE FROM Liver_Numeric WHERE Liver_Numeric.PATIENT_ID IN (SELECT MasterTable.ID FROM MasterTable WHERE MasterTable.Patient_ID = 601 AND MasterTable.Template_ID = '1') AND Liver_Numeric.ATTRIBUTE_ID IN (SELECT TemplateTable.Attribute_ID FROM TemplateTable WHERE TemplateTable.Attribute_Name = 'Alkaline Phosphatase');
Q36	db.mycol.find({"Diastolic": {$gt:90}}, {"Systolic" :1})	SELECT Patient_ID, Attribute_ID, Value_Real, Value_String FROM EAV, Metadata WHERE Patient_ID IN (SELECT Patient_ID FROM EAV, Metadata WHERE Attribute_ID = Metadata.Attribute_ID AND Metadata.Attribute_Name = 'Diastolic' AND Value_Real > 90.00) AND Attribute_ID = Metadata.Attribute_ID AND Metadata.Attribute_Name = 'Systolic';	SELECT BP_Numeric.PATIENT_ID, BP_Numeric.ATTRIBUTE_ID, BP_Numeric.VALUE FROM BP_Numeric WHERE BP_Numeric.PATIENT_ID IN (SELECT BP_Numeric.PATIENT_ID FROM BP_Numeric WHERE BP_Numeric.ATTRIBUTE_ID IN (SELECT TemplateTable.Attribute_ID FROM TemplateTable WHERE TemplateTable.Attribute_Name = 'Diastolic') AND BP_Numeric.VALUE > 90) AND BP_Numeric.ATTRIBUTE_ID IN (SELECT TemplateTable.Attribute_ID FROM TemplateTable WHERE TemplateTable.Attribute_Name = 'Systolic');

References

1. Dinu, V.; Nadkarni, P. Guidelines for the effective use of entity–attribute–value modeling for biomedical databases. *Int. J. Med. Inform.* **2007**, *76*, 769–779. [CrossRef] [PubMed]

2. Ramakrishnan, R.; Gehrke, J. *Database Management Systems*; McGraw Hill: New York, NY, USA, 2000.
3. Agrawal, R.; Somani, A.; Xu, Y. *Storage and Querying of E-Commerce Data*; VLDB: Roma, Italy, 2001; Volume 1, pp. 149–158.
4. Chu, E.; Beckmann, J.; Naughton, J. The case for a wide-table approach to manage sparse relational data sets. In Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, Beijing, China, 11–14 June 2007; pp. 821–832.
5. Beckmann, J.L.; Halverson, A.; Krishnamurthy, R.; Naughton, J.F. Extending RDBMSs to support sparse datasets using an interpreted attribute storage format. In Proceedings of the 22nd International Conference on Data Engineering (ICDE'06), Atlanta, GA, USA, 3–7 April 2006; p. 58.
6. Copeland, G.P.; Khoshafian, S.N. A decomposition storage model. *ACM SIGMOD Rec.* **1985**, *14*, 268–279. [CrossRef]
7. Freire, S.M.; Sundvall, E.; Karlsson, D.; Lambrix, P. Performance of xml databases for epidemiological queries in archetype-based ehrs. In Proceedings of the Scandinavian Conference on Health Informatics 2012, Linköping, Sweden, 2–3 October 2012; Linköping University Electronic Press: Linköping, Sweden, 2012. No. 070. pp. 51–57.
8. Node+Path Persistence. Available online: https://openehr.atlassian.net/wiki/pages/viewpage.action?pageId=6553626 (accessed on 18 February 2016).
9. Wang, L.; Min, L.; Wang, R.; Lu, X.; Duan, H. Archetype relational mapping-a practical openEHR persistence solution. *BMC Med. Inform. Decis. Mak.* **2015**, *15*, 88. [CrossRef] [PubMed]
10. Corwin, J.; Silberschatz, A.; Miller, P.L.; Marenco, L. Dynamic tables: An architecture for managing evolving, heterogeneous biomedical data in relational database management systems. *J. Am. Med. Inform. Assoc.* **2007**, *14*, 86–93. [CrossRef] [PubMed]
11. Luo, G.; Frey, L.J. Efficient execution methods of pivoting for bulk extraction of Entity-Attribute-Value-modeled data. *IEEE J. Biomed. Health Inform.* **2016**, *20*, 644–654. [CrossRef] [PubMed]
12. Duftschmid, G.; Wrba, T.; Rinner, C. Extraction of standardized archetyped data from Electronic Health Record Systems based on the Entity-Attribute-Value Model. *Int. J. Med. Inform.* **2010**, *79*, 585–597. [CrossRef] [PubMed]
13. Johnson, S. Generic data modeling for clinical repositories. *J. Am. Med. Inform. Assoc.* **1996**, *3*, 328–339. [CrossRef] [PubMed]
14. Abadi, D.J.; Marcus, A.; Madden, S.R.; Hollenbach, K. SW-Store: A vertically partitioned DBMS for Semantic Web data management. *VLDB J.* **2009**, *18*, 385–406. [CrossRef]
15. Stead, W.; Hammond, W.; Straube, M. A chartless record—Is it adequate? *J. Med. Syst.* **1983**, *7*, 103–109. [CrossRef] [PubMed]
16. Warner, H.; Olmsted, C.; Rutherford, B. HELP—A program for medical decision making. *Comput. Biomed. Res.* **1972**, *5*, 65–74. [CrossRef]
17. Pryor, T. The HELP medical record system. *MD Comput.* **1988**, *5*, 22–33. [PubMed]
18. Huff, S.M.; Haug, D.J.; Stevens, L.E.; Dupont, C.C.; Pryor, T.A. HELP the next generation: A new clientserver architecture. In Proceedings of the 18th Symposium on Computer Applications in Medical Care, Washington, DC, USA, 5–9 November 1994; IEEE Computer Press: Los Alamitos, CA, USA, 1994; pp. 271–275.
19. Nadkarni, P.M.; Brandt, C.; Frawley, S.; Sayward, F.G.; Einbinder, R.; Zelterman, D.; Schacter, L.; Miller, P.L. Managing attribute-value clinical trials data using the ACT/DB client—Server database system. *J. Am. Med. Inform. Assoc.* **1998**, *5*, 139–151. [CrossRef] [PubMed]
20. Brandt, C.; Nadkarni, P.; Marenco, L.; Karras, B.T.; Lu, C.; Schacter, L.; Fisk, J.M.; Miller, P.L. Reengineering a database for clinical trials management: Lessons for system architects. *Control. Clin. Trials* **2000**, *21*, 440–461. [CrossRef]
21. Nadkarni, P.M.; Marenco, L.; Chen, R.; Skoufos, E.; Shepherd, G.; Miller, P. Organization of Heterogeneous Scientific Data Using the EAV/CR Representation. *J. Am. Med. Inform. Assoc.* **1999**, *6*, 478–493. [CrossRef] [PubMed]
22. Shepherd, G.M.; Healy, M.D.; Singer, M.S.; Peterson, B.E.; Mirsky, J.S.; Wright, L.; Smith, J.E.; Nadkarni, P.M.; Miller, P.L. Senselab: A project in multidisciplinary, multilevel sensory integration. In *Neuroinformatics: An Overview of the Human Brain Project*; Koslow, S.H., Huerta, M.F., Eds.; Lawrence Erlbaum Associates, Inc.: Mahwah, NJ, USA, 1997; pp. 21–56.

23. Marenco, L.; Nadkarni, P.; Skoufos, E.; Shepherd, G.; Miller, P. Neuronal database integration: The Senselab EAV data model. In Proceedings of the AMIA Symposium, Washington, DC, USA, 6–10 November 1999; pp. 102–106.
24. Oracle Health Sciences Clintrial | Oracle. Available online: http://www.oracle.com/us/industries/life-sciences/health-sciences-clintrial-363570.html (accessed on 10 August 2016).
25. Oracle Clinical—Overview | Oracle. Available online: http://www.oracle.com/us/products/applications/health-sciences/e-clinical/clinical/index.html (accessed on 14 August 2016).
26. Oracle Designer Product Information. Available online: http://www.oracle.com/technetwork/developer-tools/designer/overview/index-082236.html (accessed on 14 August 2016).
27. Kalido—Home. Available online: http://kalido.com/ (accessed on 14 August 2016).
28. Evans, R.S.; Lloyd, J.F.; Pierce, L.A. Clinical use of an enterprise data warehouse. In Proceedings of the AMIA Annual Symposium Proceedings, Chicago, IL, USA, 3–7 November 2012; Volume 2012, p. 189.
29. Entity–Attribute–Value Model—Wikipedia. Available online: https://en.wikipedia.org/wiki/Entity%E2%80%93attribute%E2%80%93value_model (accessed on 12 September 2017).
30. Paraiso-Medina, S.; Perez-Rey, D.; Bucur, A.; Claerhout, B.; Alonso-Calvo, R. Semantic normalization and query abstraction based on SNOMED-CT and HL7: Supporting multicentric clinical Trials. *IEEE J. Biomed. Health Inform.* **2015**, *19*, 1061–1067. [CrossRef] [PubMed]
31. OpenEHR Community. Available online: http://www.openehr.org/ (accessed on 10 October 2016).
32. CEN—European Committee for Standardization: Standards. Available online: http://www.cen.eu/CEN/Sectors/TechnicalCommitteesWorkshops/CENTechnicalCommittees/Pages/Standards.aspx?param=6232&title=CEN/TC+251 (accessed on 11 July 2016).
33. ISO 13606-1. *Health Informatics: Electronic Health Record Communication. Part 1: RM*, 1st ed.; International Organization for Standardization (ISO): Geneva, Switzerland, 2008.
34. ISO 13606-2. *Health Informatics: Electronic Health Record Communication. Part 2: Archetype Interchange Specification*, 1st ed.; International Organization for Standardization (ISO): Geneva, Switzerland, 2008.
35. HL7. Health Level 7. Available online: www.hl7.org (accessed on 23 October 2013).
36. Beale, T.; Heard, S. The openEHR architecture: Architecture/overview. In *openEHR Release 1.0.2*; openEHR Foundation: London, UK, 2008.
37. Grimson, J.; Grimson, W.; Berry, D.; Stephens, G.; Felton, E.; Kalra, D.; Toussaint, P.; Weier, O.W. A CORBA-based integration of distributed electronic healthcare records using the synapses approach. *IEEE Trans. Inf. Technol. Biomed.* **1998**, *2*, 124–138. [CrossRef] [PubMed]
38. Kobayashi, S.; Kimura, E.; Ishihara, K. Archetype model-driven development framework for EHR web system. *Healthc. Inform. Res.* **2013**, *19*, 271–277. [CrossRef] [PubMed]
39. Trigo, J.D.; Kohl, C.D.; Eguzkiza, A.; Martinez-Espronceda, M.; Alesanco, A.; Serrano, L.; Garcia, J.; Knaup, P. On the seamless, harmonized use of ISO/IEEE11073 and openEHR. *IEEE J. Biomed. Health Inform.* **2014**, *18*, 872–884. [CrossRef] [PubMed]
40. Sachdeva, S.; Bhalla, S. Semantic interoperability in standardized electronic health record databases. *J. Data Inf. Qual. (JDIQ)* **2012**, *3*. [CrossRef]
41. CKM. Clinical Knowledge Manager. Available online: http://www.openehr.org/knowledge/ (accessed on 12 December 2016).
42. International Health Terminology Standards Development Organisation. Systematized Nomenclature of Medicine-Clinical Terms (SNOMED CT). Available online: http://www.ihtsdo.org/snomed-ct/ (accessed on 14 September 2016).
43. Nadkarni, P. *Clinical Research Computing: A Practitioner's Handbook*; Academic Press: Cambridge, MA, USA, 2016; Chapter 3.
44. Tahara, D.; Diamond, T.; Abadi, D.J. Sinew: A SQL system for multi-structured data. In Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, Snowbird, UT, USA, 22–27 June 2014; pp. 815–826.
45. Pavlo, A.; Aslett, M. What's Really New with NewSQL? *ACM SIGMOD Rec.* **2016**, *45*, 45–55. [CrossRef]
46. Cui, B.; Zhao, J.; Yang, D. Exploring correlated subspaces for efficient query processing in sparse databases. *IEEE Trans. Knowl. Data Eng.* **2010**, *22*, 219–233. [CrossRef]
47. Use Sparse Columns | Microsoft Docs. Available online: https://docs.microsoft.com/en-us/sql/relational-databases/tables/use-sparse-columns (accessed on 24 September 2017).

48. Abadi, D.J. *Column Stores for Wide and Sparse Data*; CIDR: Asilomar, CA, USA, January 2007; pp. 292–297.
49. Larson, P.A.; Clinciu, C.; Fraser, C.; Hanson, E.N.; Mokhtar, M.; Nowakiewicz, M.; Papadimos, V.; Price, S.L.; Rangarajan, S.; Rusanu, R.; et al. Enhancements to SQL server column stores. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, 22–27 June 2013; pp. 1159–1168.
50. Ailamaki, A.; DeWitt, D.J.; Hill, M.D.; Skounakis, M. Weaving Relations for Cache Performance. In Proceedings of the 27th International Conference on Very Large Data Bases, Rome, Italy, 11–14 September 2001; Volume 1, pp. 169–180.
51. Ramamurthy, R.; DeWitt, D.J.; Su, Q. A case for fractured mirrors. *VLDB J.* **2003**, *12*, 89–101. [CrossRef]
52. Grund, M.; Krüger, J.; Plattner, H.; Zeier, A.; Cudre-Mauroux, P.; Madden, S. HYRISE: A main memory hybrid storage engine. *Proc. VLDB Endow.* **2010**, *4*, 105–116. [CrossRef]
53. Pinnecke, M.; Broneske, D.; Durand, G.C.; Saake, G. Are Databases Fit for Hybrid Workloads on GPUs? A Storage Engine's Perspective. In Proceedings of the 2017 IEEE 33rd International Conference on Data Engineering (ICDE), San Diego, CA, USA, 19–22 April 2017; pp. 1599–1606.
54. Bornhoevd, C.; Werner, H. Dynamic Database Schemas for Highly Irregularly Structured or Heterogeneous Data. U.S. Patent 8,352,510, 8 January 2013.
55. Baumgartner, C.; Plant, C.; Railing, K.; Kriegel, H.P.; Kroger, P. Subspace selection for clustering high-dimensional data. In Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM'04), Brighton, UK, 1–4 November 2004; pp. 11–18.
56. Zhang, D.; Chee, Y.M.; Mondal, A.; Tung, A.K.; Kitsuregawa, M. Keyword search in spatial databases: Towards searching by document. In Proceedings of the IEEE 25th International Conference on Data Engineering (ICDE'09), Shanghai, China, 29 March–2 April 2009; pp. 688–699.
57. Abadi, D.J.; Myers, D.S.; DeWitt, D.J.; Madden, S.R. Materialization strategies in a column-oriented DBMS. In Proceedings of the IEEE 23rd International Conference on Data Engineering (ICDE 2007), Istanbul, Turkey, 15–20 April 2007; pp. 466–475.
58. Murphy, S.N.; Weber, G.; Mendis, M.; Gainer, V.; Chueh, H.C.; Churchill, S.; Kohane, I. Serving the enterprise and beyond with informatics for integrating biology and the bedside (i2b2). *J. Am. Med. Inform. Assoc.* **2010**, *17*, 124–130. [CrossRef] [PubMed]
59. Deshmukh, V.G.; Meystre, S.M.; Mitchell, J.A. Evaluating the informatics for integrating biology and the bedside system for clinical research. *BMC Med. Res. Methodol.* **2009**, *9*, 70. [CrossRef] [PubMed]
60. Haarbrandt, B.; Tute, E.; Marschollek, M. Automated population of an i2b2 clinical data warehouse from an openEHR-based data repository. *J. Biomed. Inform.* **2016**, *63*, 277–294. [CrossRef] [PubMed]
61. Ramachandran, R.; Nair, D.P.; Jasmi, J. A horizontal fragmentation method based on data semantics. In Proceedings of the 2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), Chennai, India, 15–17 December 2016; pp. 1–5.
62. OpenEHR—Wikipedia. Available online: https://en.wikipedia.org/wiki/OpenEHR (accessed on 10 December 2016).
63. Ocean Informatics. Available online: https://code4health.org/_attachment/modellingintro/2015_11_Modelling_Intro.pdf (accessed on 10 December 2016).
64. Poll Results—Top 10 Archetypes for Use in an Emergency—Health Information Model—openEHR Wiki. Available online: https://openehr.atlassian.net/wiki/display/healthmod/Poll+Results+Top+10+archetypes+for+use+in+an+Emergency (accessed on 10 December 2016).
65. openEHR-Modelling Tools. Available online: http://www.openehr.org/downloads/modellingtools (accessed on 10 December 2016).
66. Batra, S.; Sachdeva, S.; Mehndiratta, P.; Parashar, H.J. Mining standardized semantic interoperable electronic healthcare records. In *Biomedical Informatics and Technology*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 179–193.
67. CNET Networks, Inc. CNET Product Directory. Available online: http://shopper.cnet.com/4296-3000_9-0-0-0.html (accessed on 12 September 2016).
68. UCI Machine Learning Repository: Liver Disorders Data Set. Available online: https://archive.ics.uci.edu/ml/datasets/Liver+Disorders (accessed on 12 September 2016).
69. UCI Machine Learning Repository: Thyroid Disease Data Set. Available online: https://archive.ics.uci.edu/ml/datasets/Thyroid+Disease (accessed on 12 September 2016).

70. The Facts about High Blood Pressure—American Heart Association. Available online: http://www.heart.org/HEARTORG/Conditions/HighBloodPressure/GettheFactsAboutHighBloodPressure/The-Facts-About-High-Blood-Pressure_UCM_002050_Article.jsp#.WSpqPGiGPIU (accessed on 12 September 2016).
71. DB-Engines Ranking—Popularity Ranking of Database Management Systems. Available online: http://db-engines.com/en/ranking (accessed on 15 November 2017).
72. Our Customers | MongoDB. Available online: https://www.mongodb.com/who-uses-mongodb (accessed on 15 November 2017).
73. Beale, T.; Heard, S. The openEHR Archetype Model—Archetype Definition Language ADL 1.4. In *openEHR Release 1.0.2*; openEHR Foundation: London, UK, 2008.