A novel data storage logic in the cloud [version 1; referees: 1 approved, 1 not approved]

Bence Mátyás¹, Máté Szarka²,³, Gábor Járvás⁴, Gábor Kusper⁵, István Argay⁶,⁷

¹Kerpely Kalman Doctoral School, University of Debrecen, Debrecen, Hungary
²Medical and Health Sciences Center, Research Centre for Molecular Medicine, University of Debrecen, Debrecen, Hungary
³Vitrolink Biotechnological Researching, Development, Servicing and Trading Limited Liability Company, Debrecen, Hungary
⁴Hungarian Academy of Sciences MTA-PE, Translational Glycomics Group, University of Pannonia, Veszprem, Hungary
⁵Computer Science Department, Eszterhazy Karoly University, Eszterhazy, Hungary
⁶Department of Obstetric and Gynecology, UD MSHC, University of Debrecen Medical Center, Debrecen, Hungary
⁷IRCAD France Laparoscopic Training center, Strasbourg, France

Abstract
Databases which store and manage long-term scientific information related to life science are used to store huge amounts of quantitative attributes. Introduction of a new entity attribute requires modification of the existing data tables and the programs that use these data tables. The solution is increasing the virtual data tables while the number of screens remains the same. The main objective of the present study was to introduce a logic called Joker Tao (JT) which provides universal data storage for cloud-based databases. It means all types of input data can be interpreted as an entity and attribute at the same time, in the same data table.
Corresponding author: Bence Mátyás (bmatyas@ups.edu.ec)

Competing interests: No competing interests were disclosed.

How to cite this article: Mátyás B, Szarka M, Járvás G et al. A novel data storage logic in the cloud [version 1; referees: 1 approved, 1 not approved] F1000Research 2016, 5:93 (doi: 10.12688/f1000research.7727.1)

Copyright: © 2016 Mátyás B et al. This is an open access article distributed under the terms of the Creative Commons Attribution Licence, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Grant information: The first version of JT is a Hungarian product which was developed in 2008 (R.number: INNO-1-2008-0015 MFB-00897/2008) thanks to an INNOCSEK European Union application.

The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Introduction
Databases which store and manage long-term scientific information related to life science are used to store huge amount of quantitative attributes. This is especially true for medical databases\(^1\)\(^-\)\(^4\). One major downside of these data is that information on multiple occurrences of an illness in the same individual cannot be connected\(^5\)\(^-\)\(^7\). Modern database management systems fall into two broad classes: Relational Database Management System (RDBMS) and Not Only Structured Query Language (NoSQL)\(^8\)\(^-\)\(^9\). The primary goal of this paper is to introduce a novel database model which provides an opportunity to store and manage each input data in one (physical) data table while the data storage concept is structured. JT can be defined as a NoSQL engine on an SQL platform that can serve data from different data storage concepts without several conversions.

Methods
The technical environment is Oracle Application Express (Apex) 5.0 cloud-based technology. Workstation: OS (which is indifferent) + internet browser (Chrome). The Joker Tao logic (www.jokertao.com) can be applied in any RDBMS system (e.g. www.taodb.hu). Specification of the physical data table structure was determined with -ID (num) as the identifier of the entity, which identifies the entity between the data tables (not only in the given data table); -ATTRIBUTE (num) is the identifier of the attribute; -SEQUENCE (num) which is used in the case of a vector attribute; and -VALUE (VARCHAR2) which is used for storing values of the attributes. The codes which are stored in the Attribute column are also defined, sooner or later, in the ID column. At that time the attribute becomes an entity. In every case, the subjectivity determines the depth of the JT structure. From this stage, the developer uses identifiers which were defined in the previous steps instead of attribute names.

Following this, the presented data table has been modified step by step. At the end of these steps, the JT data storage structure is created. The first step is the technical data storage. In Table 2, technical data will be stored which describes exactly what the virtual data table stores in the physical data table.

In the second step, the identifiers assigned to the attributes are displayed (Table 3).

In the third step, identifiers assigned to the entities are also displayed (Table 4). These identifiers are assigned to each cell of the entity. These identifiers are determined by the developer. The values of these identifiers can be any natural number that has not already been used in the ID column.

In the fourth step, the attribute identifiers are also assigned to each cell (Table 5). These identifiers are determined by the developer. The values of these identifiers can be any natural number that has not already been used in the Attribute column.

In the fifth step, the initial value of the cell is inserted as the Value of the JT structure (Table 6). From this stage, the developer uses identifiers (which were defined in the previous steps) instead of attribute names.

The final step is to rotate the traditional data table structure 90 degrees. This means each virtual data table is defined in one physical data table. With these steps the developer can design one data table to store each entity, attribute and formula in a database.

### Table 1. Traditional data storage structure.

<table>
<thead>
<tr>
<th>Name</th>
<th>Attribute 1</th>
<th>Attribute 2</th>
<th>Attribute 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item 1</td>
<td>Value 1</td>
<td>Value 2</td>
<td>Value 3</td>
</tr>
<tr>
<td>Item 2</td>
<td>Value 4</td>
<td>Value 2</td>
<td>Value 5</td>
</tr>
</tbody>
</table>

### Table 2. Belonging to the virtual data tables.

<table>
<thead>
<tr>
<th>Tables</th>
<th>Name</th>
<th>Attr.1</th>
<th>Attr.2</th>
<th>Attr.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Item 1</td>
<td>Value 1</td>
<td>Value 2</td>
<td>Value 3</td>
</tr>
<tr>
<td>Table 1</td>
<td>Item 2</td>
<td>Value 4</td>
<td>Value 2</td>
<td>Value 5</td>
</tr>
</tbody>
</table>

### Table 3. Identifiers assigned to the attributes.

<table>
<thead>
<tr>
<th>Tables (1010)</th>
<th>Name (1019)</th>
<th>Attr.1 (1027)</th>
<th>Attr.2 (1028)</th>
<th>Attr.3 (1029)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Item 1</td>
<td>Value 1</td>
<td>Value 2</td>
<td>Value 3</td>
</tr>
<tr>
<td>Table 1</td>
<td>Item 2</td>
<td>Value 4</td>
<td>Value 2</td>
<td>Value 5</td>
</tr>
</tbody>
</table>

### Table 4. Identifiers assigned to the entities.

<table>
<thead>
<tr>
<th>Name (1019)</th>
<th>Attribute 1 (1027)</th>
<th>Attribute 2 (1028)</th>
<th>Attribute 3 (1029)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item 1</td>
<td>Value 1</td>
<td>Value 2</td>
<td>Value 3</td>
</tr>
<tr>
<td>Item 2</td>
<td>Value 4</td>
<td>Value 2</td>
<td>Value 5</td>
</tr>
</tbody>
</table>

### Table 5. Identifiers of records and columns.

<table>
<thead>
<tr>
<th>Table (1010)</th>
<th>Name (1019)</th>
<th>Attr.1 (1027)</th>
<th>Attr.2 (1028)</th>
<th>Attr.3 (1029)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Item 1</td>
<td>Value 1</td>
<td>Value 2</td>
<td>Value 3</td>
</tr>
<tr>
<td>Table 1</td>
<td>Item 2</td>
<td>Value 4</td>
<td>Value 2</td>
<td>Value 5</td>
</tr>
</tbody>
</table>

### Table 6. Data table representation in record ID, column ID and value structure.

<table>
<thead>
<tr>
<th>1010 (Tab 1)</th>
<th>1019 (Name)</th>
<th>Attr.1 (1027)</th>
<th>Attr.2 (1028)</th>
<th>Attr.3 (1029)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001, 1010, 1086</td>
<td>10001, 1019, 1 Item1</td>
<td>10001, 1027, 1 Value 1</td>
<td>10001, 1028, 1 Value 2</td>
<td>10001, 1029, 1 Value 3</td>
</tr>
<tr>
<td>10001, 1010, 1086</td>
<td>10002, 1019, 1 Item2</td>
<td>10003, 1027, 1 Value 4</td>
<td>10004, 1028, 1 Value 2</td>
<td>10005, 1029, 1 Value 5</td>
</tr>
</tbody>
</table>
The above described method can be applied manually. For the automatic conversion we created a Java code below:

```java
public static String getEntityName ( )
throws Exception
{
    Connection conn = broker.getConnection ( );
    PreparedStatement pstmt =
    conn.prepareStatement ("select *
    from joker");
    ResultSet rs = pstmt.executeQuery ( );
    int i = 0;
    while (rs.next ( )) {
        i++;
    }
    System.out.println ("number of records:" + i);
    broker.freeConnection (conn);
    return "";
}
public static void insertJokerRow
(Integr GROUP_ID, Integer UNIQ_ID,
Integer FIELD_ID, Integer ARRAY_INDEX,
String SEEK_VALUE, String FIELD_VALUE)
throws Exception {
    if (GROUP_ID == null) pstmt.setNull (1, 2);
    else pstmt.setInt (1, GROUP_ID.intValue ( ));
    if (UNIQ_ID == null) pstmt.setNull (2, 2);
    else pstmt.setInt (2, UNIQ_ID.intValue ( ));
    if (FIELD_ID == null) pstmt.setNull (3, 2);
    else pstmt.setInt (3, FIELD_ID.intValue ( ));
    if (ARRAY_INDEX == null) pstmt.setNull (4, 2);
    else pstmt.setInt (4, ARRAY_INDEX.intValue ( ));
    if (SEEK_VALUE == null) pstmt.setNull (5, 12);
    else pstmt.setString (5, SEEK_VALUE);
    if (FIELD_VALUE == null) pstmt.setNull (6, 12);
    else pstmt.setString (6, FIELD_VALUE);
    pstmt.execute ( );
}
public static void readFile ( ) throws Exception
{
    File f = new File ("data.txt");
    BufferedReader br = new BufferedReader
(new FileReader (f));
    while (br.ready ( )) {
        String line = br.readLine ( );
        int GROUP_ID = Integer.parseInt
(line.substring (0, 10));
        int UNIQ_ID = Integer.parseInt
(line.substring (11, 21));
        int ARRAY_INDEX = Integer.parseInt
(line.substring (22, 32));
        String FIELD_VALUE = line.length ( ) > 32?
        line.substring (23, line.length ( )): " ";
        insertJokerRow (Integer.valueOf (GROUP_ID),
        Integer.valueOf (UNIQ_ID), null,
        Integer.valueOf (ARRAY_INDEX), null, FIELD_VALUE);
    }
    br.close ( );
}
```

Results
The resulting table structure is called JT structure (Table 7).

<table>
<thead>
<tr>
<th>ID (Record)</th>
<th>Attribute</th>
<th>Sequence</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001</td>
<td>1010</td>
<td>1</td>
<td>1086</td>
</tr>
<tr>
<td>10001</td>
<td>1019</td>
<td>1</td>
<td>Item 1</td>
</tr>
<tr>
<td>10001</td>
<td>1027</td>
<td>1</td>
<td>Value 1</td>
</tr>
<tr>
<td>10001</td>
<td>1028</td>
<td>1</td>
<td>Value 2</td>
</tr>
<tr>
<td>10001</td>
<td>1029</td>
<td>1</td>
<td>Value 3</td>
</tr>
<tr>
<td>10002</td>
<td>1010</td>
<td>1</td>
<td>1086</td>
</tr>
<tr>
<td>10002</td>
<td>1019</td>
<td>1</td>
<td>Item 2</td>
</tr>
<tr>
<td>10002</td>
<td>1027</td>
<td>1</td>
<td>Value 1</td>
</tr>
<tr>
<td>10003</td>
<td>1028</td>
<td>1</td>
<td>Value 2</td>
</tr>
<tr>
<td>10004</td>
<td>1029</td>
<td>1</td>
<td>Value 3</td>
</tr>
</tbody>
</table>

From the JT physical data table, the following definitions can be read out:

- Virtual record is the set of the physical data tables which have the same ID value.
- Virtual data table is the set of the virtual records which have the same value of the belonging to the virtual data table (code 1010) attribute.

Thesis: In the JT structure, each attribute needs only one index for indexing in the database.

Proof using mathematical induction: It is obvious the statement is true for the case of one record stored in a data table (according to the RDBMS structure where the developers use more indexes to indexing more attributes). In this case the data table appears as shown in Figure 1.

Index = attribute (num) + value (varchar 2)

In view of entity, an ID (numerical) index is also used in JT logic-based systems. This ID does not depend (no transitive dependency) on any attribute. Thus, the entities of the virtual data tables meet the criteria of the third normal form (Figure 2).

The modes of the expansion of a data table are: -input new entity (Figure 3); -input new attribute (Figure 4); -input new virtual data table (Figure 5).
**Figure 1.** Indexing a record.

![Indexing a record](image1.png)

**Figure 2.** ID usage.

![ID usage](image2.png)
The indexing is correct in case of n+1 record expansion also. With JT logic the user is able to use only one physical data table to define each virtual data table in a database. Therefore, since only one index is required to index each attribute, the statement of the thesis is true in every case of the JT logic-based data table according to the principle of mathematical induction below. Thesis: For n=1 ergo;

\[ 1 + 2 + \ldots + n = \frac{n(n+1)}{2} \]

substituting one into the equation we get:

\[ 1 = 1 \times (1 + 1)/2 \]

result of the operation is 1=1, that is, the induction base is true.

Using proof by induction we can now show that this is true for the following equation:

\[ n = k \text{ where } k \text{ is a optional but fixed natural number. Therefore, we know that the following operation is true:} \]

\[ 1 + 2 + \ldots + k = k \times (k + 1)/2 \]

Finally using n=k+1 we can prove our assumption to be true:

\[ 1 + 2 + \ldots + k + (k + 1) = (k + 1) \times (k + 2)/2 \]

The above induction proof shows:

\[ 1 + 2 + \ldots + k + (k + 1) = k \times (k + 1)/2 + (k + 1) \]

Conducting the mathematical operations we obtain the following:

\[ 1 + 2 + \ldots + k + (k + 1) = (k \times ((k + 1)/2) + 2 \times (k + 1))/2 = (k \times k + 2k + 2)/2 = (k \times k + 3k + 3)/2 \]

Conducting the mathematical operations on the other side we obtain the same:

\[ (k+1) \times (k+2)/2 = (k \times k + 2k + 2)/2 = (k \times k + 3k + 2)/2 \]

Thus, the induction step is true. Given that both the induction base and the induction step are true, the original statement is therefore true. In the present study, we explained the JT data storage logic. In our other study we focused on the query tests. Our previous results show that from 10000 records the relational model generates slow (more than 1 second) queries in a cloud-based environment while JT can remain with the one second time frame.
Discussion and conclusions
Using the developed database management logic, each attribute needs only one index for indexing in the database. JT allows any data whether entity, attribute, data connection or formula, to be stored and managed even under one physical data table. Thanks to this flexibility, a formula which is stored in a database can be utilized for problem solving in another field regardless of the difference in data storage method used in the present environment. In the JT data model, the entity and the attribute are used interchangeably, so users can expand the database with new attributes after or during the development process. With JT logic, NoSQL engine is ensured in SQL database systems for the storage and management of long term scientific information.

Author contributions
BM, MSZ, GJ, IA conceived the study. MSZ, GJ, IA, GK tested and the developed method. GK developed the mathematical proof. BM prepared the first draft of the manuscript. All authors were involved in the revision of the draft manuscript and have agreed to the final content.

Competing interests
No competing interests were disclosed.

Grant information
The first version of JT is a Hungarian product which was developed in 2008 (R.number: INNO-1-2008-0015 MFB-00897/2008) thanks to an INNOCSEK European Union application.

I confirm that the funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Acknowledgements
The corresponding author is thankful to György Mátyás, the idea owner of JT framework. The authors are thankful to the Call Tec Consulting Ltd. (the first one with highest Oracle certification in Hungary) and the first one that validated JT.

References
Open Peer Review

Current Referee Status:  

Version 1

Referee Report 25 February 2016

doi:10.5256/f1000research.8321.r12373

Kavita Sunil Oza
Department of Computer Science, Shivaji University, Kolhapur, Maharashtra, India

Work demonstrated in the paper is good and well explained. Complexity of work is not mentioned (algorithmic complexity) but this is not necessary as we already have high speed processors and time complexity may not matter much. Some more references should have been added but not mandatory as number of references are sufficient.

Competing Interests: No competing interests were disclosed.

I have read this submission. I believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Referee Report 15 February 2016

doi:10.5256/f1000research.8321.r12375

Jan Lindström
MariaDB Corporation, Espoo, Finland

In this paper authors introduce a new logic called Joker Tao (JT) which provides universal data storage for cloud-based databases. However, the paper is very poorly written. Firstly, the proposed logic is not presented detailed enough for the reader to understand and validate the method. Authors should research how relational model is presented and based on rigorous relational calculus and algebra. Based on this research, this paper should be rewritten based on rigorous mathematical foundation and give clear examples. Secondly, one table based example is far from convincing and provided Java-program is unnecessary. Length of the paper should be greatly increased to contain detailed description of JT method and give examples. Lastly, presentation is so poor that is not even clear how queries to resulting JT structure can be executed. To be honest, currently paper looks more like computer generated rubbish than a real scientific paper.

Competing Interests: No competing interests were disclosed.

I have read this submission. I believe that I have an appropriate level of expertise to state that I do not consider it to be of an acceptable scientific standard, for reasons outlined above.
The benefits of publishing with F1000Research:

- Your article is published within days, with no editorial bias
- You can publish traditional articles, null/negative results, case reports, data notes and more
- The peer review process is transparent and collaborative
- Your article is indexed in PubMed after passing peer review
- Dedicated customer support at every stage

For pre-submission enquiries, contact research@f1000.com