

# Relational Database with Multicore-Framework

**Venkatesh Kakalwar<sup>1</sup>, Dipanjan De<sup>2</sup>, Arup Kumar Biswas<sup>3</sup>, Sivanesan S<sup>4</sup>**

First Year M. Tech. (SCOPE), Computer Science and Engineering, VIT University, Vellore<sup>1</sup>

Assistant Professor (Selection Grade), School of Computer Engineering, VIT University, Vellore<sup>4</sup>

**Abstract:** Database Management is one of the most important tasks for an organization today. The ultimate goal is not only storing and/or managing data, but to do so efficiently, in order to reduce the overall cost and expenditure of the organization. Using the right database system to handle multiple types of data, increases the overall efficiency of business operation. These systems are built to be extremely versatile.

Through this paper, we present an alternate way of storing data in a database, in pursuit of optimizing space and reducing the time complexity during data search or retrieval.

## I. INTRODUCTION

The gigantic size of today's databases makes it increasingly difficult to optimise the space during storage. Unequivocally, the performance of the database is affected as the space optimisation degrades. Furthermore, as the size of the database grows, the more time it takes to search and/or retrieve data, and the time complexity increases. This paper focuses on storing data virtually in three-dimension in the memory. We use an additional dimension to depict the structure of the database, and the basic structure is a solid  $l \times m \times n$  block, where 'l', 'm' and 'n' are all real numbers greater than or equal to 1. We first provide a conceptual framework, followed by some applications. Then we move to section where we explain the advantages of having an additional virtual dimension and we explain some scenarios where this kind of modelling could be implemented. Later we conclude with the scope, and further possibilities in this domain.

## II. CONCEPTUAL FRAMEWORK

### A. Basic Structure

The idea of the structure came from the Rubik's cube and we thought that many two-dimensional relations can be condensed together in the form of a cube and all relations inside the cube can be linked through foreign keys. Also, the memory allocation for storing the data would be sequential, and in a three-dimensional matrix format. To save memory, related data can be aggregated inside the cube next to one another. To bind the cube together, link lists can be used and the concept of indexing can be employed.

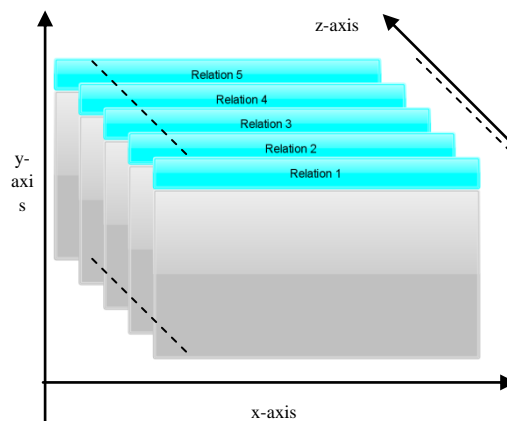


Figure 2.1. An example of a set of relations, stored in a three-dimensional manner, connected by foreign keys with the primary key of the main relation

The design is inspired by three dimensional matrices and can be taken to higher dimensions as well, to form a tesseract in four-dimensions. However, visualising such structure is very difficult and the conundrum limits us to stick to three-dimensions for now. However, the structure of a tesseract has enormous potentials for storing multivalued attributes and depth enabled data.

**B. Storage and Memory Management**

The storage of such databases can be implemented using indexed allocation of memory. A simple array can be used to store the starting address of each relation in the cube. The first block would point to the first relation, the next would point to the second plane in the x-y axis, and so on.

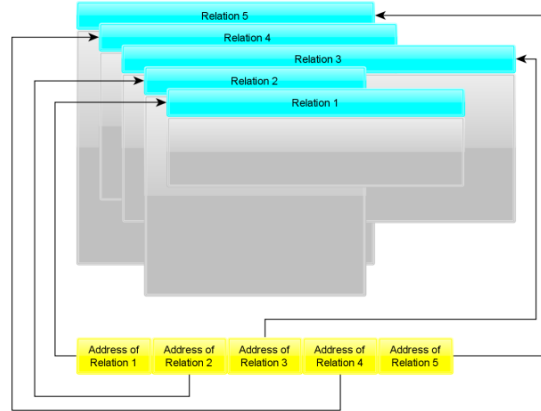


Figure 2.2. An Indexing based Memory Management technique

As we can see from figure 2.3, a simple one-dimensional array points to the starting addresses of the relations in the cube. This enables us to have relations of different sizes inside a cube, without wasting memory blocks.

**C. Search and Retrieve Operations in Multicore Environment**

Multicore environment can increase the performance of this system greatly. Each core can be used to traverse a relation in parallel, which implies that ‘n’ cores can traverse ‘n’ relations in a time quantum. However, in case of large relations, a single core traversing the entirety of the relation may take a long time. In such a scenario, a different core can be tasked to traverse the second half of the relation and the first core would search the first half. Hyperthreading concepts can also be applied to improve the performance of searches.

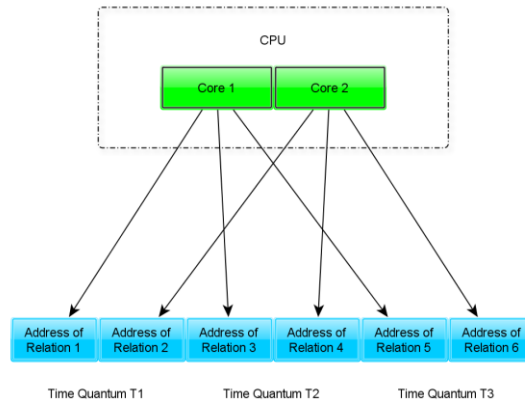
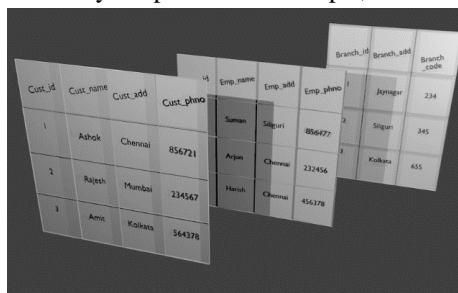


Figure 2.3. An example of multicore scheduling

**III. APPLICATIONS AND POTENTIALS**

**A. Visualisation through a whole new perspective**

The use of the third dimension to depict the structure of a database can be a very effective and straightforward approach in displaying information to the user. It not only simplifies the concepts, but also enhances the views.



| Cust_id | Cust_name | Cust_add | Cust_phno | Emp_name | Emp_add | Emp_phno | Branch_id | Branch_add | Branch_code |
|---------|-----------|----------|-----------|----------|---------|----------|-----------|------------|-------------|
| 1       | Ashok     | Chennai  | 856721    | Suman    | Siguri  | 856477   | 1         | Siguri     | 234         |
| 2       | Rajesh    | Mumbai   | 234547    | Aryam    | Chennai | 232456   | 2         | Kolkata    | 455         |
| 3       | Amic      | Kolkata  | 564378    | Harish   | Chennai | 456378   | 3         |            |             |

Figure 3.1. A sample three dimensional graphic visualisation of connected relations through foreign keys

**B. Special Joins**

Joining two relations would sometimes yield a very special and interactive relation, which would be three dimensional. As we sometimes need to create link tables during joins, this feature will come in handy for many users and developers.

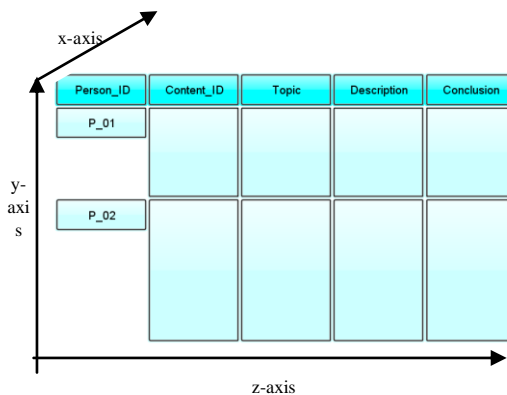


Figure 3.2. Special Join using Foreign Key

An example of such joins is visualised in figure 3.2, where the x-axis contains the Persons Relation, depicted in figure 3.3(a) and the link with the Content Relation, depicted in figure 3.3(b), is merged along the z-axis. The above representation is two-dimensional, but the goal is to represent this in three-dimensions, so that the users can hover back and forth across dimensions and view both tables at the same time.

| Serial Number | Person_ID | Name | Phone | E-Mail |
|---------------|-----------|------|-------|--------|
|               |           |      |       |        |
|               |           |      |       |        |
|               |           |      |       |        |
|               |           |      |       |        |

(a) Persons Table, here Person\_ID is the Primary Key

| Person_ID | Content_ID | Topic | Description | Conclusion |
|-----------|------------|-------|-------------|------------|
|           |            |       |             |            |
|           |            |       |             |            |
|           |            |       |             |            |
|           |            |       |             |            |

(b) Content Table, here Content\_ID is the Primary Key and Persons\_ID is the Foreign Key

Figure 3.3. Persons and Content Relations in x-axis and y-axis

**C. Faster search and/or retrieval using Multicore Processors**

The search and retrieval time complexities can be significantly improved using a multicore processor environment. We can imagine searching ‘n’ relations with ‘n’ cores at the same time. This can further be customised using scheduling algorithms like Round Robin or the Priority Scheduling, with or without pre-emption. Such parallel programming features can be added and optimised.

**D. Depth Enabled Spatial Databases**

This concept can also be applied to enhance Spatial Databases to store information, not only about a plane, but above and below the plane as well. This can be optimised and used by experts in various institutions and fields like weather forecasting, geosciences, carbon dating, etc.



Figure 3.4. An example of 3D representation of density of matter and composition under the surface by  $1 \times 1 \times 1$  metre, where each pixel stores information of  $1 \times 1 \times 1$  milli-metre. The pale blue pixels represents fossil remains found in designated areas during the study.

Such representations can help store gigantic data effortlessly and efficiently in certain research like “The study of composition of Antarctic landmass up to a depth of 500 metres”. Researchers can easily mark where they found fossils and different compositions of soil in different depths.

#### IV. CONCLUSIONS

Our endeavour is to add a new aspect to the tools and techniques and we do not aim to disband the existing systems. We hope to improve the way how data is stored in the memory, and we hope that this concept will be realised for creating more robust systems. Furthermore, our idea can prove to be helpful in creating special joins and visualise them through a whole new perspective. Also, providing more visual information can improve the understanding of the users and developers.

However, this will reach its relative complexity when compared to basic techniques employed currently, but hopefully the merits of this system will outnumber the shortcomings. We hope for this concept to thrive and help professionals build more efficient and robust systems in the future.

#### REFERENCES

1. S. Dixit and M. K. Singh, “3D Database Modelling”, ICWET’11 Proceedings of the International Conference & Workshop on Emerging Trends in Technology, ACM New York Publication
2. Nadika Habaraduwa Liyanage, “Advanced query model design concept to support multi-dimensional data analytics for relational database management systems”, 2017 International Conference on Big Data Analytics and Computational Intelligence (ICBDAC), IEEE Conference Publication
3. Marc Gyssens and Laks V. S. Lakshmanan, “A Foundation for Multi-Dimensional Databases”, VLDB’97 Proceedings of the 23<sup>rd</sup> International Conference on Very Large Databases, ISBN: 1-55860-470-7, ACM San Francisco Publication
4. Helen Hasan, Peter Hyland, David Dodds and Raja Veeraraghavan, “Approaches to the development of multi-dimensional databases: lessons from four case studies”, Published in ACM SIGMIS Database, Volume 31, Issue 3, Summer 2000, ACM New York Publication