www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 5 Issue 09 September 2016 Page No.17916-17920

A Review on Performance of Database Systems

Azfar Inayat Khan
Research Scholarsinghania University, Rajasthan
Azfarkhan_78@Hotmail.Com

Abstract

Presently most of the researchers focus on works in the area of pattern recognition, computer networks, mobile computing, information security, image processing, data mining, bio inspired computing, theoretical computer science and cloud computing. Research in the area of data modelling for relational databases has trimmed down considerably.

Since research in the area of data modelling for relational databases has trimmed down considerably, it has been comprehended that the fissure between user expectation and product delivered, in database systems design and development, has to be diminished. The field of modelling and design of databases is vast, with a surplus of methodologies as well as techniques, but designers need authentic guidelines that can be used for designing proficient database management systems. This paper reviews the different techniques used to perk up the performance of databases.

Keywords: Database, ER Diagram, Normalization, Refactoring

1. Introduction

Data modeling is the foundation for understanding consumer's necessities and designing information system of any organization. It is a process used to identify and examine data necessities needed to sustain the business processes within the scope of consequent information systems in organizations. During the information systems growth, process within an organization, data resources are usually analyzed in the form of a data model. When developing a data model, it is imperative to communicate with the stakeholders about the necessities.

It visually signifies the nature of data, business policies that are pertinent to data, and how it will be organized in the database. Data modelling assists the end-users to define their necessities, and the developers are able to develop a system to meet those specified necessities.

According to the types of concepts used to portray the database structure, data model can be classified as high-level or conceptual data model, representational or implementation model and low-level or physical data model [1].

A conceptual or high-level model is a model of the actual world expressed in terms of data necessities. This model provides conceptions that are close to the way many users recognize data. This model uses the conception of entities, attributes and relationship. Physical or low-level model describes the details of how data is stored in the computer by representing information such as record formats, record ordering and access path. Conception provided by physical models are generally intended for computer experts and not for typical end users. Representational or realization model are the intermediary model that provide conception that may be understood by end users and hiding some details of data storage. The conception of representational model fall between the above two models, balancing user views with some computer storage details.

The hierarchical data model is described as the data model which organizes data logically in accordance with the structural relationships of hierarchical definition trees. A hierarchical database therefore consists of an assortment of records that are connected with each other through links.

Each record is an assortment of fields or attributes, each of which contains one data value. A link is an association between precisely two records.

DOI: 10.18535/ijecs/v5i9.21

2. ER Model

There are several notations for data modeling. The most general and traditionally used conceptual data model is the Entity-Relationship (ER) model. The rationale of using ER model is to provide a common, informal and expedient model for communication between users and the database administrator for the purpose of modeling the data structures.

The ER model portrays data as entities, attributes and relationships among entities. The basic constituent that the ER model represents is an entity. An entity is a thing in the real world with an independent existence. An entity may be an object with a physical survival like car, house or employee. An entity may be an object with a conceptual existence like a company, a job or a university course. An attribute is a property that portrays the entity. A particular entity will have a value for each of its attributes. The attribute values become a major part of a data stored in the database. The various types of attributes available in the ER model are simple versus composite, single valued versus multi-valued, and stored versus derived.

An et al [2] presented a round-trip engineering framework and a set of principles and procedures that automatically and incrementally maintain conceptual-relational mappings as schemas. The first principle for mapping maintenance under schema evolution is to locate the appropriate elements in the conceptual model for adding new attributes. The location process is guided by analyzing the key and foreign key information in the original and new schemas. The second principle is to locate the anchors of the appropriate skeleton trees for discovering or adding relationships. The location process is guided by using key and foreign key structures in the schemas. The third principle is to arrange the primary key and foreign key constraints in the (new) schema with the cardinality constraints in the new conceptual model. The authors also propose two procedures for maintaining the mappings. The first procedure maintains the mappings when schemas evolve. Given a set of consistent conceptual-relational mappings as input, the procedure gives a synchronized conceptual model and a set of updated mappings as output. The second procedure obtains a set of consistent conceptual-relational mappings between a conceptual model and a relational schema as input and gives an updated new set of mappings.

De Lucia et al [3] in their work aimed to analyze the comprehensibility ER diagrams and UML class diagrams in data model maintenance. They performed three sets of controlled experiments. They stated that the results demonstrate that using UML class diagrams achieved better comprehension levels and the support given by the two notations during maintenance activities are same, while in general UML class diagrams provide a better support during verification activities.

Dhabe et al [4] proposed an Articulated Entity-Relationship (AER) diagram which is an extension of Entity-Relationship (ER) diagram to accommodate the functional dependency (FD) information as its integral part for complete automation of normalization. As the proposed AER diagram is designed by taking into account the normalization process, normalization up to BCNF becomes an integral part of conceptual design. Any modifications made to the AER diagram will automatically be reflected in its FD information. FDs are diagrammatically represented using two types of connectors: attribute connectors and functional dependency connectors. They have shown complete AER diagram for banking enterprise. They concluded that AER diagrams could be extended to include multi-valued dependencies and join dependencies. They stated that it should be possible to the domain and key constraints to automate normalization up to DKNF.

Cuadra et al [5] in their work provided a methodological framework to inspire the database designer to use ternary relationship, the constraint which database designers find it very difficult to detect, represent and manage according to the domain requirements. The three approaches are taken namely Chen’s approach, Merise’s method and their proposed strategy consisting of a combination of these approaches. Calculation of cardinality constraints in binary and ternary relationships is shown in their method. Based on the study, it is proved that their approach has a high level of participant confidence.

3. Database Normalization

The purpose of any data model, relational or otherwise, is to allow the user of the model to describe and manipulate those relationships among entities in the real world that the user intends to store in the database. In the relational model, such a collection of relationships is represented in a relational schema. A relational schema consists of a set of database relations and for each relation the specification of candidate key(s) and relationship among relations represented through foreign keys. It is vital to consider functional relationships when selecting grouping of attributes into relations. Functional relationships among database attributes are formalized in the concept of functional dependency. The resulting relations may contain undesirable structures unless data is normalized. Such a database design often leads to undesirable properties called data anomalies. These anomalies often lead to repetition of information, inability to present certain information, and loss of information. Normalization is essential to avoid insertion, deletion and updation anomalies. Normalization is the process of grouping attributes into well structured relations free of anomalies.

4. Code modularization

A widely used representational data model is the relational model. In software engineering discipline and practice, the modules and their relationships are established in the architectural design before coding takes place. Modularization of code is similar to data normalization which gives the benefits of reusability, reliability, manageability, readability. In order to acclimatize to the constantly evolving necessities, software designers and developers add new features or alter the existing design. Because of this, the software becomes more and more complex and drift away from its original design thereby the quality of the software is getting reduced and hence effort on maintenance is increased. A huge portion of the total software development cost is usually spent on software maintenance[6]. To alleviate the maintenance, the existing code is to be restructured so that the changes made in a module will not create any adverse effect in any other module, which brings down the ripple effect. This diminishes the software intricacy by perking up the internal software quality.

5. Refactoring

Refactoring is "the process of changing an object-oriented software system in such a way that it does not alter the external behaviour of the code, yet improves its internal behaviour. The key idea behind refactoring or restructuring is to promote code reuse. Clean, modular, well written code is easy to recycle and diminishes future programming efforts. Furthermore, refactoring aims to perk up several factors of quality namely, understandability, portability, maintainability, testability, reliability, usability, reusability and adaptability.

Fokaefs et al [7] proposed a methodology which identifies extract class refactoring opportunities by a class decomposition method. An agglomerative clustering algorithm is used based on the Jaccard distance between class members. In terms of cohesion this work facilitates to recognize novel conceptions and rank the solutions according to their impact on the design quality of the system. Specific kind of bad smells called "God Class" is considered for this reason. Data god class and behavioural god class are defined. A class which has many system's data in terms of number of attributes is called data god class. When it has greater portion of the systems functionality in terms of number and complexity of methods is called behavioural god class. Behavioural god class may be avoided by splitting the class by extracting a cohesive and independent piece of functionality. This refactoring is called "Extract Class". Two projects namely eRisk, an electronic adaptation of the well known board game and SelfPlanner[8], an intelligent web based calendar application have been taken for this purpose and the result shows that this methodology identifies relatively large number of new concepts that can be potentially extracted in new classes.

Tsantalis and Chatzigeorgiou et al [9] have proposed the placement of attributes or methods within classes in an object-oriented system which is typically guided by conceptual criteria and aided by suitable metrics. Moving state and behaviour between classes can assist to diminish coupling and to increase cohesion, but it is nontrivial to recognize. They proposed a methodology for the recognition of move method refactoring opportunities that comprise a way for solving numerous familiar feature envy awful smells.

Bavota et al [10] have proposed an approach using game theory to identify extract-class refactoring opportunities. Game theory techniques can often be used to deal contrasting goals. Game theory is successfully used to propose solutions to strategic situations, in which an individual's achievement in making choices depends on the choices of others. It is very common in software engineering to find a solution often for problems having competing goals, like integrity versus efficiency, reusability versus reliability, reusability versus integrity, quality versus cost, cohesion versus coupling by the developers and managers. The extract-class refactoring setback can be modelled as a non-cooperative game involving two players. Given a class to be refactored, the two players compete with the methods of the original class to construct two novel classes with high cohesion and poorer coupling than the original class. Their approach considers S and T as two players in charge of building a novel class selecting methods from the original class to be refactored. The process starts by assigning to S and T two methods that have the lowest structural and semantic similarity. S and T will then iteratively contend with the remaining methods of the class to be refactored. In single iteration, S or T could add at least one method to its class by considering the impact of adding the method on the cohesion and coupling of its class. The process stops when each method of the original class is assigned to either S or T. The move to be performed during iteration is chosen by finding the Nash equilibrium in the payoff matrix.

6. Game Theory

Game Theory is the branch of mathematics that is widely used in many fields especially in economics for the purpose of making decision on conflicting goals and to obtain a single optimum solution. The games studied in game theory are well-defined mathematical objects involving two or more usually non-cooperating players each of which playing a set of strategies. More precisely, a game consists of a set of players, a set of moves or strategies available to those players, and a specification of payoffs for each combination of moves. For a given game, it is necessary to determine the game solutions in which no player gains anything by changing only her or his own scheme unilaterally (situation of equilibrium). Game theory is fruitfully used in other fields, particularly in economics, to mathematically propose solutions to strategic situation, in which an individual's accomplishment in making choices depends on the choices of others.

In game theory, the Nash equilibrium is a elucidation conception of a non-cooperative game involving two or more players, in which each player is assumed to be acquainted with the equilibrium strategies of the other players, and no player has anything to gain by changing only his own strategy unilaterally. If each player has chosen a strategy and no player can gain by changing strategies while the other players keep theirs unchanged, then the current set of strategy choices and the corresponding payoffs compose Nash equilibrium. Game theorists use the Nash equilibrium conception to analyze the result of the strategic interaction of numerous decision makers. In other words, it provides a way of envisaging what will happen if numerous people or several institutions are making decisions at the same time and if the outcome depends on the decisions of the others.

7. Conclusion

Database design is a challenging and complex task due to its importance in the overall performance of the database system. Recently, various techniques have been under use across different stages of database application modelling, design and implementation and they have been under continuous study and improvements by various researchers.

In this paper we reviewed the work done in the ER modelling, code modularization, refactoring and gaming theory techniques.

References

[1]. Elmasri, R. and Navathe, S.B. Fundamentals of Database Systems, sixth ed., Pearson, 2010.

[2]. An, Y., Hu, X. and Song. Y. "Maintaining Mappings between Conceptual Models and Relational Schemas", Journal of Database Management, Vol.21, No.3, pp.36-68, 2010.

[3]. De Lucia, A., Gravino, C., Oliveto, R. and. Tortora, G. "An experimental comparison of ER and UML Class diagrams for data modeling", Empirical Software Engineering, Vol.15, No.5, pp.455-492, 2010.

[4]. Dhabe, P. S., Patwardhan, M. S., Asavari

Deshpande, A., Dhore, M. L., Barbadekar, B.V. and Abhaynkar, H.K. “Articulated Entity Relationship (AER) Diagram for complete automation of relational database normalization”, International Journal of Database Management Systems , Vol.2, No.2, pp.84-100, 2010.

[5]. Cuadra, D., Martínez, P. and Castro, E. “Guidelines for representing complex cardinality constraints in binary and ternary relationships”, Software & Systems Modeling, 2012.

[6]. Mens, T. and Tourwe, T. “A Survey of Software Refactoring”, IEEE Transactions on Software Engineering, Vol.30, No.2, pp.126-139, 2004.

[7]. Marios Fokaefs, Nikolaos Tsantalis and Jorg Sander Alexander Chatzigeorgiou, “Decomposing Object-Oriented Class Modules using an Agglomerative Clustering Technique”, IEEE Conference, 2009.

[8]. Refanidis, I. and Alexiadis, A. “SelfPlanner: Planning your time”, ICAPS Workshop on Scheduling and Planning Applications, 2008.

[9]. Tsantalis, N. and Chatzigeorgiou, A.“Identification of Move Method Refactoring Opportunities”, IEEE Transactions on Software Engineering, Vol.35, No.3, pp.347-367, 2009.

[10]. Bavota, G., Oliveto, R. and Lucia, A.D. “Playing with refactoring: Identifying extract class opportunities through game theory”, in ICSM ‘10 Proc. of the IEEE International Conference on Software Maintenance, pp 1-5, 2010.