A Framework for Systematic Database Denormalization

YMA PINTO

Goa University , India
ymapinto@gmail.com

***Abstract*- It is currently the norm that relational database designs should be based on a normalized logical data model. The primary objective of this design technique is data integrity and database extendibility. The Third Normal Form is regarded by academicians and practitioners alike to be point at which the database design is most efficient. Unfortunately, even this lower level normalization form has a major drawback with regards to query evaluation. Information retrievals from the database can result in large number of joins which degrades query performance. So you need to sometimes break theoretical rules for real world performance gains. Most existing Conceptual Level RDBMS data models provide a set of constructs that only describes "what data is used" and does not capture "how the data is being used". The question of "how data is used" gets embedded in the implementation level details. As a result, every application built on the existing database extracts the same or similar data in different ways. If the functional use of the data is also captured, common query evaluation techniques can be formulated and optimized at the design phase, without affecting the normalized database structure constructed at the Conceptual Design phase. This paper looks at denormalization as an effort to improve the performance in data retrievals made from the database without compromising data integrity. A study on a hierarchical database table shows the performance gain - with respect to response time – using a denormalization technique.**

Keywords: denormalization, database deign, performance tuning, materialized views, query evaluation

I. INTRODUCTION

Most of the applications existing today have been built, or are still being built using RDBMS or ORDBMS technologies. The RDBMS is thus not dead, as stated by Arnon-Roten [Roten_Gal, 2009]. Van Couver, a software engineer with vast experience in databases at Sun MicroSystems, emphasizes the fact that RDBMSs are here to stay but do require improvements in scalability and performance bottlenecks [Couver , 2009].

Normalization is the process of putting one fact and nothing more than one fact in exactly one appropriate place. Related facts about a single entity are stored together, and every attribute of each entity is non-transitively associated to the Primary Key of that entity. This design technique results in enhanced data integrity and removes insert, update and delete anomalies that would have otherwise been present in a non-normalized database. Another goal of normalization is to minimize redesign of the database structure. Admittedly, it is impossible to predict every need that your database design will have to fulfill and every issue that is likely to arise, but it is important to mitigate against potential problems as much as possible by a careful planning.

Arguably, normalizing your data is essential to good performance, and ease of development, but the question always comes up: "How normalized is normalized enough?" Many books on normalization, mention that 3NF is essential, and many times BCNF, but 4NF and 5NF are really useful and well worth the time required to implement them [Davidson, 2007]. This optimization, however, results in performance degradation in data retrievals from the database as a large number of joins need to be done to solve queries [Date, 1997] [Inmon, 1987] [Schkolnick and Sorenson ,1980].

"Third normal form seems to be regarded by many as the points where your database will be most efficient ... If your database is overnormalized you run the risk of excessive table joins. So you denormalize and break theoretical rules for real world performance gains." [Sql Forums, 2009].

There is thus a wide gap between the academicians and the database application practitioners which needs to be addressed. Normalization promotes an optimal design from a logical perspective. Denormalization is a design level that needs to be mitigated one step up from normalization. With respective to performance of retrieval, denormalization is not necessarily a bad decision if implemented following a systematic approach to large scale databases where dozens of relational tables are used.

Denormalization is an effort that seeks to optimize performance while maintaining data integrity. A denormalized database is thus not equivalent to a database that has not been normalized. Instead, you only seek to denormalize a data model that has already been normalized. This distinction is important to understand, because you go from normalized to denormalized, not from nothing to denormalized. The mistake that some software developers do is to directly build a denormalized database considering only the performance aspect. This only optimizes one part of the equation, which is database reads. Denormalization is a design level that is one step up from normalization and should not be treated naively. Framing denormalization against normalization purely in the context of performance

is unserious and can result in major application problems [Thought Clusters, 2009]. We need to understand how and when to use denormalization

This paper is organized as follows: Section 1 introduces the concept and current need for denormalization. Section 2 provides us a background of the related work in this area from the academic and the practitioners' point of view. Section 3 makes a strong case for denormalization while Section 4 presents the framework for a systematic denormalization. Section 5 elucidates some denormalization techniques that can be followed during the database design life cycle and shows the performance gain of this technique over a Hierarchical Normalized Relation.

II. BACKGROUND AND RELATED WORK

Relational Databases can be roughly categorized into Transaction Processing (OLTP) and Data Warehouse (OLAP). As a general rule, OLTP databases use normalized schema and ACID transactions to maintain database integrity as the data needs to be continuously updated when transactions occur. As a general rule, OLAP databases use unnormalized schema (the "star schema" is the paradigmatic OLAP schema) and are accessed without transactions because each table row is written exactly one time and then never deleted nor updated. Often, new data is added to OLAP databases in an overnight batch, with only queries occurring during normal business hours [Lurie M.,IBM, 2009] [Microsoft SQL Server guide] [Wiseth ,Oracle].

Software developers and practitioners mention that database design principles besides normalization, include building of indices on the data and denormalization of some tables for performance. Performance tuning methods like indices and clustering data of multiple tables exist, but these methods tend to optimize a subset of queries at the expense of the others. Indices consume extra storage and are effective only when they work on a single attribute or an entire key value .The evaluation plans sometimes skip the secondary indexes that are created by users if these indices are nonclustering [Khaldtiance , 2008].

Materialized Views can also be used as a technique for improving performance [Vincent et al,97] but these consume vast amount of storage and their maintenance results in additional runtime overheads. Blind application of Materialized Views can actually result in worse query evaluation plans and should be used carefully [Chaudhuri et al, 1995]. View update techniques have been researched and a relatively new method of updating using additional views has been proposed [Ross et al, 1996].

In the real world, denormalization is sometimes necessary. There have been two major trends in the approach to demoralization. The first approach uses a "non normalized ERD" where the entities in the ERD are collapsed to decrease the joins. In the second approach, denormalization is done at the physical level by consolidating relations, adding synthetic attributes and creating materialized views to improve performance. The disadvantage of this approach is the overheads required in view consistency maintenance. Denormalization is not necessarily a bad decision if implemented wisely [Mullins , 2009].

Some denormalization techniques have been researched and implemented in many strategic applications to improve query response times. These strategies are followed in the creation of data warehouses and data marts [Shin and Sanders, 2006] [Barquin and Edelstein] and are not directly applicable to an OLTP system. Restructuring a monolithic Web application composed of Web pages that address queries to a single database into a group of independent Web services querying each other also requires denormalization for improved performance [Wei Z et al, 2008].

Several researches have developed a list of normalization and denormalization types ,and have subsequently mentioned that denormalization should be carefully deployed according to how the data will be used [Hauns ,1994] [Rodgers, 1989].The primary methods that have been identified are : combining tables, introducing redundant data, storing derivable data, allowing repeating groups, partitioning tables, creating report tables, mirroring tables. These "denormalization patterns" have been classified as Collapsing Relations, Partitioning Relations, Adding Redundant Attributes and Adding Derived Attributes [Sanders and Shin ,2001]

III. A CASE FOR DENORMALIZATION

Four main arguments that have guided experienced practitioners in database design have been listed here [26]

The Convenience Argument

The presence of calculated values in tables' aids the evaluation of adhoc queries and report generation. Programmers do not need to know anything about the API to do the calculation.

The Stability Argument

As systems evolve, new functionality must be provided to the users while retaining the original. History data may still need to be retained in the database.

The Simple Queries Argument

Queries that involve join jungles are difficult to debug and dangerous to change. Eliminating joins makes queries simpler to write, debug and change

The Performance Argument

Denormalized databases require fewer joins in comparison to normalized relations. Computing joins are expensive and time consuming. Fewer joins directly translates to improved performance.

Denormalization of Databases, ie, a systematic creation of a database structure whose goal is performance improvement, is thus needed for today's business processing requirements. This should be an intermediate step in the DataBase Design Life Cycle integrated between the Logical DataBase Design Phase and the Physical DataBase Design Phase. Retrieval performance needs dictate very quick retrieval capability for

data stored in relational databases, especially since more accesses to databases are being done through Internet. Users are concerned with more prompt responses than an optimum design of databases. To create a Denormalization Schema the functional usage of the operational data must be analyzed for optimal Information Retrieval.

Some of the benefits of denormalization can be listed:

(a)Performance improvement by

- Precomputing derived data
- Minimizing joins
- Reducing Foreign Keys
- Reducing indices and saving storage
- Smaller search sets of data for partitioned tables
- Caching the Denormalized structures at the Client for ease of access thereby reducing query/data shipping cost.

(b)Since the Denormalized structures are primarily designed keeping in mind the functional usage of the application, users can directly access these structures rather then the base tables for report generation. This also reduces bottlenecks at the server.

A framework for denormalization needs to address the following issues:

(i) Identify the stage in the DataBase Design Life Cycle where Denormalization structures need to be created.

(ii) Identify situations and the corresponding candidate base tables that cause performance degradation.

(iii) Provide strategies for boosting query response times.

(iv) Provide a method for performing the cost-benefit analysis.

(v) Identify and strategize security and authorization constraints on the denormalized structures.

Although (iv) and (v) above are important issues in denormalization, they will not be considered in this paper and will be researched on later.

IV. A DENORMALIZATION FRAMEWORK

The framework presented in this paper differs from the papers surveyed above in the following respects:

It does not create denormalized tables with all contributing attributes from the relevant entities, but instead creates a set of Denormalized Structures over a set of Normalized tables. This is an important and pertinent criteria as these structures can be built over existing applications with no "side effects of denormalization" over the existing data.

The entire sets of attributes from the contributing entities are not stored in the Denormalized structure. This greatly reduces the storage requirements and redundancies.

The Insert, Update and Delete operations (IUDs) are not done to the denormalized structures directly and thus do not violate data integrity. The IUDs to data are done on the Base Tables and the denormalized structures are kept in synch by triggers on the base tables.

Since the denormalized structures are used for information retrieval , they need to consider the authorization access that users have over the base tables.

The construction of the "Denormalization View" is not an intermediate step between the Logical and the Physical Design phases, but needs to be consolidated by considering all 3 views of the SPARC ANSI architectural specifications.

Most existing Conceptual Level RDBMS data models provide a set of constructs that describes the structure of the database [Elmashree and Navathe]. This higher level of conceptual modeling only informs the end user "what data is used" and does not capture "how the data is being used". The question of "how data is used" gets embedded in the implementation level details. As a result, every application built on the existing database extracts the same or similar data in different ways. If the functional use of the data is also captured, common query evaluation techniques can be formulated and optimized at the design phase, without affecting the normalized database structure constructed at the Conceptual Design phase. Business rules are descriptive integrity constraints or functional (derivative or active) and ensure a well functioning of the system. Common models used during the modeling process of information systems do not allow the high level specification of business rules except a subset of ICs taken into account by the data model [Amghar and Mezaine, 1997].

The ANSI 3 level architecture stipulates 3 levels – The External Level and the Conceptual Level, which captures data at rest, and the Physical Level which describes how the data is stored and depends on the DBMS used. External Schemas or subschemas relate to the user views. The Conceptual Schema describes all the types of data that appear in the database and the relationships between data items. Integrity constraints are also specified in the conceptual schema. The Internal Schema provides definitions for stored records, methods of representation, data fields, indexes, and hashing schemes. Although this architecture provides the application development environment with logical and physical data independence, it does not provide an optimal query evaluation platform. The DBA has to balance conflicting user requirements before creating indices and consolidating the Physical schema.

The reason denormalization is at all possible in relational databases is because, courtesy of the relational model, which creates lossless decompositions of the original relation, no Information is lost in the process. The Denormalized structure can be reengineered and populated from the existing Normalized database and vice-versa. In a distributed application development environment the Denormalization Views can be cached on the client resulting in a major performance boost by saving run time shipping

costs. It would require only the Denormalization View Manager to be installed on the Client.
A High Level Architecture that this framework considers is defined as follows:

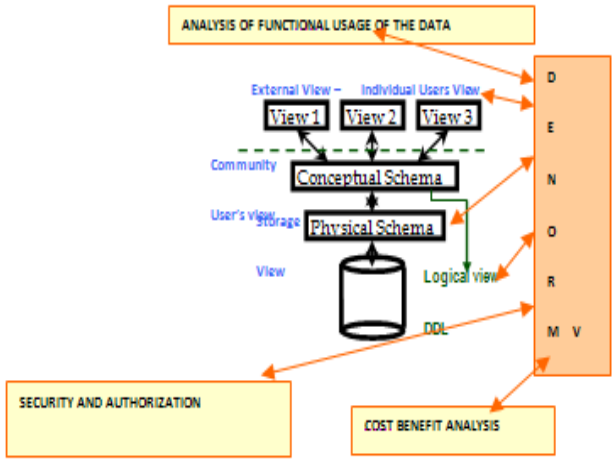

Figure 1 : A High Level Architecture of the Relational Data Information Retrieval Model

To realize the potential of the Denormalization View, efficient solutions to the three encompassing issues are required:

Denormalization View design: Determining what data and how it is stored and accessed in the Denormalization Schema
Denormalization View maintenance: Methods to efficiently update the data in the Denormalized schema when base tables are updated.
Denormalization View exploitation: Making efficient use of denormalization views to speed up query processing (either entire queries or sub queries)
Extensive research has been done on subquery evaluation on materialized views [Afrati et al, 2001] [Chirkova et al, 2006] [Halevy , 2001]

The inputs that are required for the construction of the Denormalized schema can be identified as:

- the logical and external views schema design,
- the physical storage and access methods provided by the DBMS,
- the authorization the users have on the manipulation and access of the data within the database,
- the interaction (inter and intra) between the entities,
- the number of entities the queries involve,
- the usage of the data (ie, the kind of attributes and their frequency of extraction within queries and reports),
- the volume of data being analyzed and extracted in queries (cardinality and degree of relations, number and frequency of tuples, blocking factor of tuples, clustering of data, estimated size of a relation),
- the frequency of occurrence and the priority of the query,
- the time taken by the queries to execute(with and without denormalization).

The problem can now be stated as "Given a logical schema with its corresponding database statistics and a set of queries with their frequencies, arrive at a set of denormalized structures that enhances query performance"

A few definitions are required
Defn 1: A ***Relational Data Information Retrieval System (RDIRS)*** has as its core components (i) a set of Normalized Relations {R} (ii) a set of Integrity Constraints {ICs} (iii) a set of data access methods {A} (iv) a set of Denormalization Structures {DS} and (v) a set of queries and subqueries that can be defined and evaluated on these relations.
Each component of the RDIRS, by definition, can have dynamic elements resulting in a flexible and evolvable system.

Defn 2: A ***"Denormalized Structure" (DSM)*** is a relvar [Date ,Kannan , Swamynathan] comprising of the Denormalized Schema Design and the Denormalized Structure Manager.

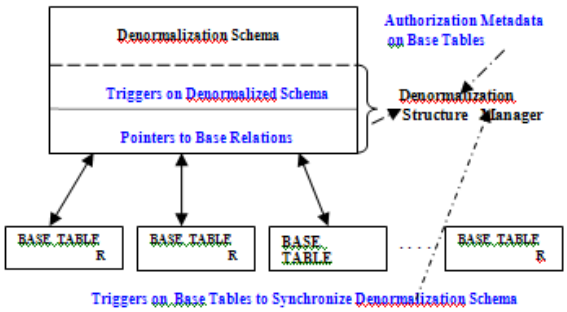

Figure 2 : The Denormalization View

A system cannot enforce truth, only consistency. Internal Predicates (IPs) are what the data means to the system and External Predicates (EPs) are what data means to a user. The EPs result in criterion for acceptability of IUD operations on the data, which is an unachievable goal [Date, Kannan, Swamynathan], especially when Materialized Views are created. In the framework presented in this paper, IUDs on the Denormalized Structures are never rejected as these are automatically propagated to the base relations where the

Domain and Table level ICs are enforced. Once the base relations are updated, the Denormalized Schema Relation triggers are invoked atomically to synchronize the data, ensuring simultaneous consistency of Base and Denormalized tables. Further, the primary reason for the Denormalization Structures is faster information retrieval and not data manipulation; hence no updates need be made to the Denormalization Schema directly.

Every Normalized Relation requires a Primary Key which satisfies the Key Integrity Constraint. This PK maintains uniqueness of tuples in the database and is not necessarily the search key value for users. For the **RDIRS** we define

Defn 3: An *Information Retrieval Key (IRK)* is a (set of) attributes that the users most frequently extract from an entity. The IRK is selected from amongst the mandatory attribute values which gives the end user meaningful information about the entity.
For ex, an employee table may have an Empid as its PK, but the IRK could be EmpName and Contact No.

Defn 4: An ***Information Retrieval Tree (IRT)*** is a Query Evaluation Tree which has as its components the operators required to extract the information from the database and the relvars that contribute to an optimized Data Extraction Plan. The IRT consists of relational algebra operations along the intermediate nodes and the relvars in the leaf nodes (base relations, views, materialized views or denormalization structures) and is a requisite for cost benefit analysis and query rewrites.

Researchers and Practitioners [Inmon, 1987] [Shin and Sanders, 2006] [Mullins, 2009] create the denormalized tables by creating a schema with all the attributes from the participating entities. This results in (i) additional storage and redundancy (ii) slows down the system on updates to data (iii) creates a scenario for data anomalies.

Defn 5: The ***Denormalization Schema (DS)*** in the RDIR Model is a relation that has as its attributes only the PKs, the IRKs and the URowIds (Universal Row Id) of the participating or contributing Base Relations.

The storage of only the PK, IRKs and URowIds is justifiable as most often, end users are interested in only the significant attributes of an entity. If required, the remaining attributes can be obtained from the base table using the RowId field stored in the Denormalized Scheme. The URowIds are chosen as they can even support row-ids on remote foreign tables.
It is interesting to note that even when a "select * "clause is used in an adhoc query, it is either because the user is unaware of the attributes of the entity or is uninterested in the attribute per se, but is actually looking for other information.

The Denormalization Schema Design is an input to the Query Optimizer for collapsing access paths, resulting in the IRT which is then submitted to the Query Evaluation Engine.

Although the metadata tables are query able at the server, the Denormalized Structure Manager can have its own metadata stored locally (at the node where the DSs are stored).
DS_Metadata_Scheme(DS_Name,DS_Trigger_Name,DS_Procedure_Name, DS_BT1_Name, Creator,DS_BT1_Trigger_Name,DS_BT2_Trigger_Name,DS_BT1_Authorization,DS_BT2_Authorization)

V. Denormalization Techniques

Denormalization looks at normalized databases which have operational data, but whose performance degrades during query evaluation. There are several indicators which will help to identify systems and tables which are potential denormalization candidates.

The techniques that can be used are summarized below:

a. Pre joined Tables

Application: When two or more tables need to be joined on a regular basis and the cost of joins is prohibitive.
This happens when Foreign Keys become a part of a relation or when transitive dependencies are removed.
Denormalization Technique: Collapse the relations.

b. Report Tables

Application: When the application requires creation of specialized reports that requires lot of formatting and data manipulation.
Denormalization Technique: The report table must contain the mandatory columns required for the report

c. Fragmenting Tables

Application: If separate pieces of a normalized table are accessed by different and distinct groups of users or applications, then the original relation can be split into two (or more) denormalized tables; one for each distinct processing group. The relation can be fragmented horizontally or vertically by preserving losslessness.
Denormalization Technique: When horizontal fragmentation is done, the predicate must be chosen such that rows are not duplicated.
When vertical fragmentation is done, the primary key must be included in the fragmented tables. Associations between the attributes of the relation must be considered. Projections that eliminate rows in the fragmented tables must be avoided.

5.1: An illustration of the above techniques

Consider the following Normalized database (3NF) relations:
(Primary Keys are in Red , Foreign keys are in Blue)

Customer (CustomerNo, CustomerName, ContactId)

Order OrderNo, CustomerNo, OrderDate, ShipRecdDate, VATax, Local_Tax, ShiptoContactId, BillToContactId)

ContactInfo (ContactId, Name, Street, City, State Country, Zip)

ContactPhone (ContactId, PhoneNo)

Item (ItemNo, ItemName, ItemPrice, ItemPart, SubItemNo)

OrderItem (OrderNo, ItemSerialNo, ItemNo, Quantity)

d. Redundant Data

Application: Sometimes one or more columns from one table are accessed whenever data from another table is accessed. If this happens frequently they could be stored as redundant data in the tables.
Denormalization Technique: The columns that are duplicated in the relation to avoid a lookup (join) should be used by a large number of users but should not be frequently updated.

e. Repeating Groups

Application: When repeating groups are normalized they are implemented as distinct rows instead of distinct columns resulting in less efficient retrieval. These repeating groups can be stored as a nested table within the original parent table.
Before deciding to implement repeating groups, it is important to consider if the data will be aggregated or compared within the row or if the data would be accessed collectively, otherwise SQL may slow down query evaluation.
Denormalization Technique: Repeating groups can be stored as "setoff(values)" - SQL Extensions - within the table removing the restriction on the number of values that can repeat.

f. Derivable Data

Application: If the cost of deriving data using complicated formulae is prohibitive then the derived data can be stored in a column. It is imperative that the stored derived value needs to be changed when the underlying values that comprise the calculated value change.
Denormalization Technique: Frequently used aggregates can be precomputed and materialized in an appropriate relation.

g. Hierarchical Speed Tables

Application: A hierarchy or a recursive relation can be easily supported in a normalized relational table but is difficult to retrieve information from efficiently. Denormalized "Speed Tables" are often used for faster data retrieval.
Denormalization Technique: Not only the immediate parent of a node is stored, but all of the child nodes at every level are stored.

Some of the major reports identified and that need to be generated from this database:

- What are the current outstanding orders along with their shipping and Billing details
- For a given order, find all the parts that are ordered along with the subparts of that part.
- Prepare a voucher for a given order.
- For orders that were paid for on the same date that the Shipment was received, give a 10% discount if the amount exceeds a value 'x' and a 20% discount if the amount exceeds a value 'y'.
- Retrieve all sub items that item number 100 contains
- Find all subparts that have no subpart.

The Denormalized Schema thus constructed over the Normalized Tables to improve performance and using the techniques described above:

DN_Oust_Order *(OrderNo, CustomerNo, OrderDate, ShipToContactInfo_Name, ShipToContactPhone_PhNo, BillToContactInfo_Name, BillToContactPhone_PhNo, ShipToContactInfo_URowId, BillToContactInfo_URowId)*

DN_Aggregate (OrderNo, OrderDate, TotalAmt, Discount)

DN_Voucher (OrderNo, OrderDate, ItemName, ItemPrice, Quantity, DN_Aggregate_RowId)

DN_Item_Hierarchy (Main_ItemId, Sub_ItemId, Child_Level, Is_Leaf, Item_URowId)
These tables can be created using the

create materialized view
build immediate
refresh fast on commit
enable query rewrite

clauses provided by the DBMSs. The URowIds of the Base Table rows can also be selected and inserted into the Denormalized Schema Extensions.

The DN_Aggregate Tables need to be created using the

withschemabinding

clause .

The Denormalized Hierarchy tables can be created using the

connect by prior
start with
level

clauses.

The CONNECT BY prior clause can automatically handle insertions.

5.2: A Performance Study on Hierarchical Queries

The Hierachical Technique for Denormalization needs to be further illustrated.

Considering the Normalized Item Data consisting of data shown below (partial view of the database)

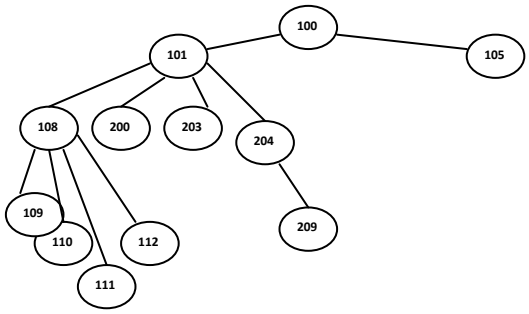

Figure 3: Partial Hierarchical Item Data

The **Normalized Relation** for the Hierarchical Item Table would be stored as

ItemNo	ParentItemNo	OtherItemDetails
100		…
101	100	…
105	100	…
108	101	…
200	101	…
203	101	…
204	101	…
109	108	…
110	108	…
111	108	…
112	108	…
209	204	…

2.

```
select itemno,itemname,parentitem from item start with
parentitem=100 connect by prior itemno=parentitem ;

69 rows selected.

Elapsed: 00:00:00.17
```

3.

```
select parentitem,childitemno,itemname from dn_item_hier
where parentitem=100
```

Consider a query ***"Find all items that are contained in item 100"*** that requires to be run on the above table. This involves finding the child nodes at every level of the hierarchy.

A Solution to the above query:

```
Select ItemNo from item where ParentItemNo='100'
 Union
Select ItemNo from item where ParentItemNo in
                (Select ItemNo from item where ParentItemNo='100')
 Union
Select ItemNo from item where ParentItemNo in
                (Select ItemNo from item where ParentItemNo in
                (Select ItemNo from item where ParentItemNo='100'))
```

This retrieval query, besides being extremely inefficient, one needs to know the maximum depth of the hierarchy.

The **Denormalized Schema** for the Item Information in the RDIRS :

DN_Item_Hierarchy (ParentItemNo, ChildItemNo, ItemName, ChildLevel, IsLeaf, Item_URowId)

The ChildLevel ascertains the level in the hierarchy that the child node is at; IsLeaf specifies if that node has further child nodes and makes queries like ***"Find all items that have no subparts"*** solvable efficiently.

The (part) extension of the DN_Item_Hierarchy Schema

ParentItemNo ChildItemNo ItemName ChildLevel IsLeaf ItemRowId

100			101		SubPart1
	1	N	….		
100		105		SubPart2	1
	N	….			
100			108		SubPart3 2
	N	….			
100			200		SubPart4
	2	Y	….		
100			203		SubPart5
	2	Y	….		
100			204		SubPart6
	2	N	**….**		
100			109		SubPart7
	3	Y	….		
100			110		SubPart8
	3	Y	….		
100			111		SubPart9
3	Y	….			
100			112		SubPart10
	3	Y	….		
100			209		SubPart11
	3	Y	…		
101			108		SubPart3
	2	N	…		
101			200		SubPart4
	2	N	…		
101			203		SubPart5
	2	N	…		
101			204		SubPart6
	2	N	…		
108			109		SubPart7
	3	Y	…		
108			110		SubPart8
	3	Y	…		
108			111		SubPart9
	3	Y	…		
108			112		SubPart10
	3	Y	…		
204			209		SubPart11
	3	Y	…		

…….
……

A Solution to the above query "***Find all items that are contained in item 100"*** can now be written as:

Select itemno from dn_item_hierarchy where parentitemno=100;

To study the performance improvement using denormalization, the normalized item table was created with 100 tuples, 70 tuples had the main root level as 100.The maximum child level nodes was 4.

The results are as shown :

```
1.

Set timing on;

select itemno,itemname,parentitem from item1 where itemno in

(select itemno from item1 where parentitem=100

union

select itemno from item1 where parentitem in

(select itemno from item1 where parentitem=100)

union

select itemno from item1 where parentitem in

(select itemno from item1 where parentitem in

(select itemno from item1 where parentitem=100)));


69 rows selected.

Elapsed: 00:00:00.31
```

VI. CONCLUSIONS AND FUTURE WORK

Although each new RDBMS release usually brings enhanced performance and improved access options that may reduce the need for denormalization, there will be many occasion where even these popular RDBMSs will require denormalized data structures. Denormalizatio will continue to remain an integral part of DataBase Design. A detailed authorization and access matrix which is stored along with the Denormalization view will further enhance performance. This and a detailed strategy for cost benefit analysis will be the next stage in the subject of my research.

REFERENCES

[*1] Afrati F., Chen Li, and Ullman J D. "Generating efficient plans using views". In SIGMOD, pages 319–330, 2001.*

[2] Amghar Y. and Mezaine M., "Active database design" ,Comad 97, Chennai, India.

[3] Chaudhuri, Krishnamurthy R, Potamianos S, and Shim K,"Optimizing Queries using materialized views", In Proceedings of the 11th International Conference on Data Engineering (Taipei, Taiwan, Mar.), ,1995,pp. 190--200.

[4] Chirkova R., Chen Li, and J Li, "Answering queries using materialized views with minimum size" ,. Vldb Journal 2006, 15 (3), pp. 191-210.

[5] Date C.J, "The Normal is so …interesting", DataBase programming and Design, Nov 1997,pp 23-25

[6] Halevy A. "Answering queries using views: A survey." In VLDB, 2001.

[8] Inmon W.H, "Denormalization for Efficiency, "ComputerWorld", Vol 21 ,1987 pp 19-21

[9] Ross K., Srivastava D. and Sudarshan S., "Materialized View Maintenace and integrity constraint checking : trading space for time", ACM Sigmod Conference 1996,pp 447 -458

[10] Rodgers U., "Denormalization: why, what and how?" Database Programming and Design,1989 (12) ,pp 46-53

[11] Sanders G. and Shin S.K, "Denormalization Effects on Performance of RDBMS", Proceedings of the 34th International Conference on Systems Sciences, 2001

[12] Schkolnick M., Sorenson P. , "Denormalization :A performance Oriented database design technique" , Proceedings of the AICA 1980 Congress ,Italy.

[13] Shin S.K and Sanders G.L., " Denormalization strategies for data retrieval from data warehouses ",Decision support Systems, VolVol. 42, No. 1, pp. 267-282, 2006

[14] Vincent M., Mohania M. and Kambayashi Y., "A Self-Maintainable View maintenance technique for data warehouses" ,8th Int. Conf on Management of Data, Chennai,India

[15] Wei Z., Dejun J., Pierre G.,Chi C.H, Steen M.,"Service-Oriented Data Denormalization for Scalable Web Applications" , Proceedings of the 17th International WWW Conference 2008, Beijing, China

[16] Barquin R., Edelstein H., "Planning and Designing the Data Warehouse", Prentice Hall

[7] Hauns M., "To normalize or denormalize, that is the question", Proceedings of 19th Int.Conf for Management and Performance Evaluation of Enterprise computing Systems, San Diego,CA,1994,pp 416-423

[17] Date C.J. ,Kannan A., Swamynathan S.,"An Introduction to Database Systems ", ,8th Ed.,Pearson Education

[18] Elmashree R. and Navathe S.,"Fundamentals of Database Systems",3rd Ed, Addison Weisley.

[19] Davidson L., "Ten common design mistakes ", software engineers blog, Feb 2007

[20] Downs K.,"The argument for Denormalization",The Database Programmer,Oct 2008

[21] Khaldtiance S., "Evaluate Index Usage in Databases", SQL Server Magazine, October 2008

[22] Lurie M.,IBM, "Winning Database Configurations

[23] Mullins C, "Denormalization Guidelines ", Platinum Tecnology Inc.,Data administration Newsletter, Accessed June 2009.

[24] Microsoft - SQL Server 7.0 Resource Guide "Chapter 12 - Data Warehousing Framework"

[25] Roten-Gal-Oz A. Cirrus minor in "Making IT work" Musings of an Holistic Architect, Accessed June 2009

[26] Van Couver D. on his blog "Van Couvering is not a verb", Accessed June 2009

[27] Wiseth K, Editor-in-Chief of Oracle Technology News, in "Find Meaning",Accessed June 2009

[28] Thought Clusters on software, development and programming, website -– March 2009

*[29] website – **http://www.sqlteam.com/Forums/**, Accessed July 2009*