A Study of Library Databases by Translating Those SQL Queries Into Relational Algebra and Generating Query Trees

Santhi Lasya [1], Sreekar Tanuku [2]

[1] – Department of Electronics and Computer Science, Jawaharlal Nehru Technological University, SNIST, Hyderabad, India.

[2] – Amazon, Seattle, WA, USA.

Abstract

Even in this World Wide Web era where there is unrestricted access to a lot of articles and books at a mouse's click, the role of an organized library is immense. It is vital to have effective software to manage various functions in a library and the fundamental for effective software is the underlying database access and the queries used. And hence library databases become our use-case for this study.

This paper starts off with considering a basic ER model of a typical library relational database. We would also list all the basic use–cases in a library management system. The next part of the paper deals with the sql queries used for performing certain functions in a library database management system. Along with the queries, we would generate reports for some of the use cases. The final section of the paper forms the crux of this library database study, wherein we would dwell on the concepts of query processing and query optimization in the relational database domain. We would analyze the above mentioned queries, by translating the query into a relational algebra expression and generating a query tree for the same. By converting algebra, we look at optimizing the query, and by generating a query tree, we would come up a cheapest cost plan.

Keywords: *Library Databases, Query Optimization, Relational Algebra, Query Tree, SQL Server Management Studio, Microsoft Visual Studio, and Cost Analysis.*

1. Introduction

The Library Database is created to support the principal functions of a lending library's day-to-day operations. It provides access to resources across a wide spectrum of topic and subject areas. Such as: the arts, academic research, home improvement, auto repair, business and much more.

The very first step is to construct an ER-Diagram of library databases, which is just an approximate description of the information to be stored in the database. With that logical design schema, we implement our database design. The database design of the library consists of creating queries and stored procedures that satisfy some of the functionalities of library operations. Through these library operations, final reports are produced and the designed database results are given in the form of test forms.

For accessing and retrieving data from the database, we convert ER-Diagram into relational database model. Relational algebra is one of the two formal query languages associated with the relational model. As relational model supports powerful query languages, steps we take in handling the queries of library databases are:

- Initial SQL queries for library databases
- Converting these SQL queries into relational algebra based on collection of operators for manipulating relations and optimizing purpose.
- Formation of Query tree for estimating the cheapest cost plan.

2.1. LIBRARY DATABASES

A library database contains relevant and accurate information in a particular field. It is both an electronic catalog and the access point to information from published works where published information sources are of magazines, newspapers, encyclopedias, journals and other resources. Library databases are easily searchable. Database content may often be searched by: Keywords, Title, Author, or Subject. Each article or book can be given in the form of

- Full Text = entire article - Library databases sometimes omit photos, graphs, charts, and figures from articles, but most will indicate that these have been omitted.
- Abstract = summary provided by the author or database publisher.

Databases provide citation information about the items they index. A citation typically consists of information such as:

- Title
- Author
- Source (Title and type of Publication)
- Publisher
- Date of Publication

Any library visitor may access the library database collection and library card holders may access many of the library's databases from home. This database works closely with faculty administrators, computing services and the Research school to provide integrated support to researchers. There is a wide collection of electronic databases which provides full text access to eBooks, databases, and thesis.

2.2. ER- DIAGRAM

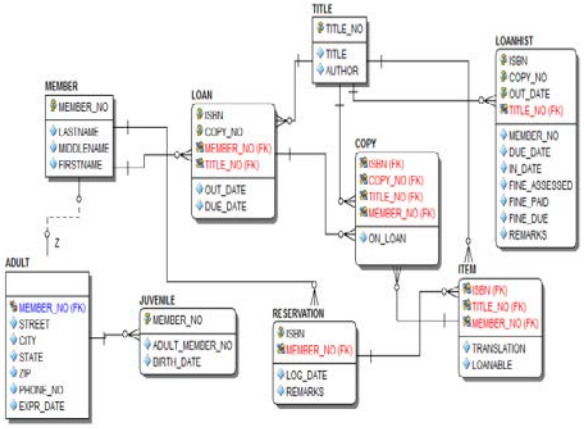


Fig.1. ER-Diagram of Library Databases

2.3. STEP BY STEP PROCEDURE

A library database contains a listing of authors that have written books on various subjects (one author per book).Here, this database has been used to-

*Create queries against the library databases that return a number of results which uses different types of joins, UNION statements, CASE statements, date manipulation, string concatenation, and aggregate functions.

*Design back-end stored procedures that satisfy some of the functionalities of library operations such as Add Adult, Add juvenile, Check in a Book, Check out a book, Add Book, Renew Membership, Change juvenile to Adult, Update Adult. The procedures incorporated input validations and provide adequate error handling using TRY/CATCH.

When we consider the E-R diagram of library databases, anyone can visit the library database collection. And for accessing many of the library databases, particular visitor should have a library card. So with the help of library card, particular member can fill up the details in 'MEMBER' entity dataset (member_no, lastname, firstname, middlename). Each member is provided with member_no in library card and whenever a library visitor wants to access particular book or any

information about the library database, using member_no we can retrieve the information.

The personal details (like address (street, city, state, and zip), phone_no, expr_date) of a visitor can be filled in the entity dataset 'ADULT' and in that, the member is given a expiry date. Once the card is expired, then the visitor has to renew the library card. If the member's age is less than 18 then that particular visitor comes under the dataset 'JUVENILE'. Juvenile entity dataset (member_no, adult_member_no, birth_date) offer online access to age-appropriate books and magazines. With the help of adult_member_no i.e, using any adult member library card number, juvenile members can get their member_no.

In the 'TITLE' dataset (title_no, title, author), the search of a specific title or any author, provides the title number (title_no). Through the dataset 'TITLE', the details of any book to be loaned are known. 'LOAN' dataset (ISBN, title_no, member_no, copy_no, out_date, due_date) contains the information about a specific book, still how many copies are there in the database if no then which member_no loans that book and the date until it is reserved by a particular member_no is given. Here in 'LOAN' dataset ISBN i.e, International Standard Book Number, is present where specific number is given to the book which is called all over world as its number. ISBN is locked in each and every dataset wherever used. 'COPY ‘dataset (ISBN copy_no, title_no, on_loan) gives the number of copies of each standard book and its details. The total data of 'LOAN' and 'COPY' dataset with excess amount of data are provided in 'LOANHIST' i.e, loan history dataset. 'LOANHIST'(ISBN, copy_no, out_data, title_no, member_no, due_date, in_date, fine_assessed, fine_paid, fine_due, remarks) contains database book details i.e, copy number ,title number, book return date, if due date is completed the fine to be paid, overall fine to be paid, fine_due and remarks given for that member_no.

The details of a book or even translation of any language can be done in the 'ITEM' dataset (ISBN, title_no, translation, loanable). Finally in 'RESERVATION' dataset (ISBN, member_no, log_date, remarks), any member can reserve a book and the details when that particular book is reserved or any remarks written on that book can be seen here. And this concludes the explanation of the overall E-R diagram of library databases.

The front-end of the database queries include many cases-

- Return member info from member and adult tables.
- Display available books.
- Search member info using member_no.
- Use UNION, list all member reserve the specific book.
- Use CASE, list all member reserve the specific book.
- Create temporary table.
- Display members have past due loan use temporary table.
- Display members who pay highest fine.
- List of members who want to reserve the specific book.

All these queries are developed on SQL Server Management Studio (SQL Server 2008). When you consider the use-case 'Display Available Books' in the library databases, the code to be written in query language on SQL Server Management Studio is shown as,

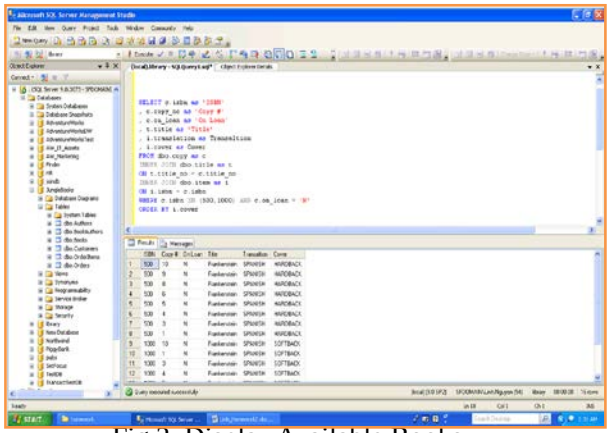


Fig.2. Display Available Books

Similarly, when we consider another case like 'Display members who pay highest fine', the output form is given as,

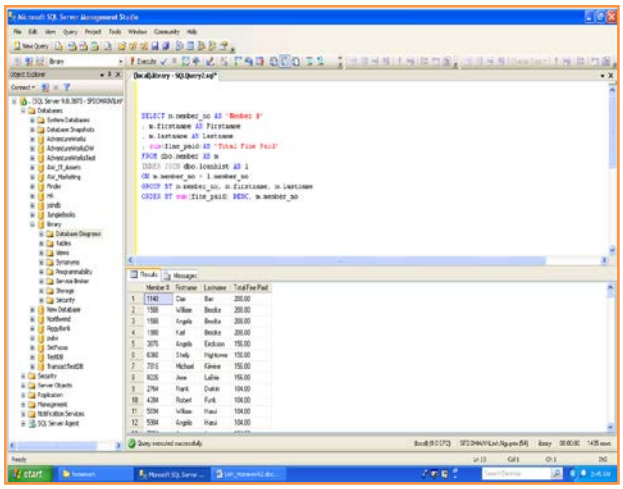


Fig.3. Display Members Who Pay Highest Fine

The back-end stored procedures support the following principal functions of a library's day-to-day operations:

Add Adult, Add Juvenile, Check In a book, Check out a book, Add Book, Renew Membership, Change Juvenile to Adult, Update Adult.

Here each and every case is designed and developed on Microsoft Visual Studio (ADO.NET). When comes to the result, Fig.4. shows 'Add Adult' test form output.

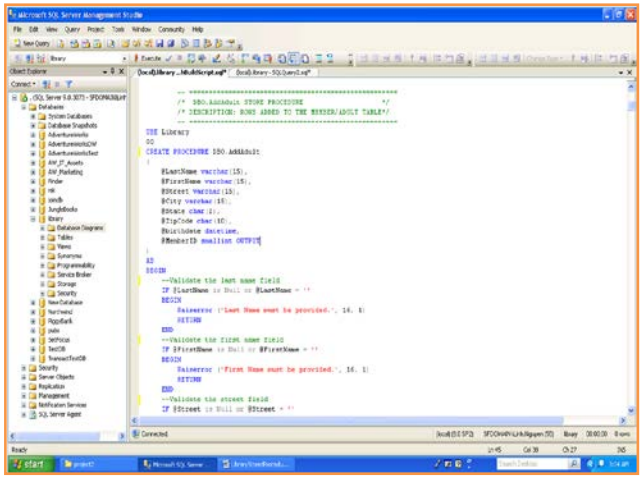


Fig.4. Add Adult

Similarly, the test form 'check out a book' output refers to be,

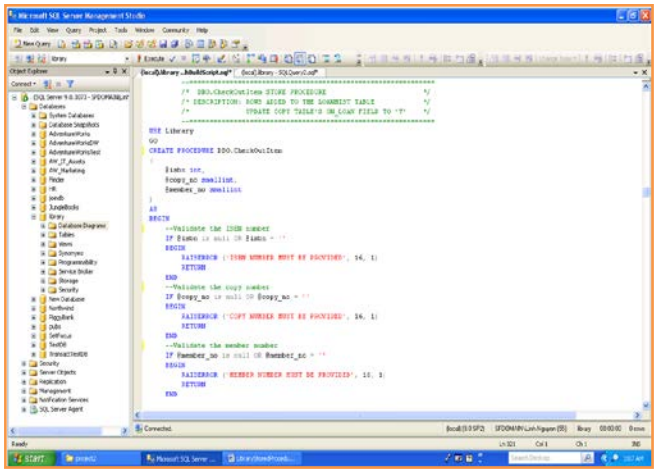


Fig.5. Check Out A Book

2.4. Reports Of Library Databases

The final reports are shown in Microsoft Visual Studios. Overall reports produced in library databases are-

- Complete list of books
- No of copies per title
- Most active members
- List of books on loan
- Adult member detail
- Dependents member detail
- Expired memberships
- Reference and special collection items
- Current fines for overdue books
- Total fines by member

The report 'List of books on loan' i.e, the list of total books under loan in the library database are shown as result in this fig.6.

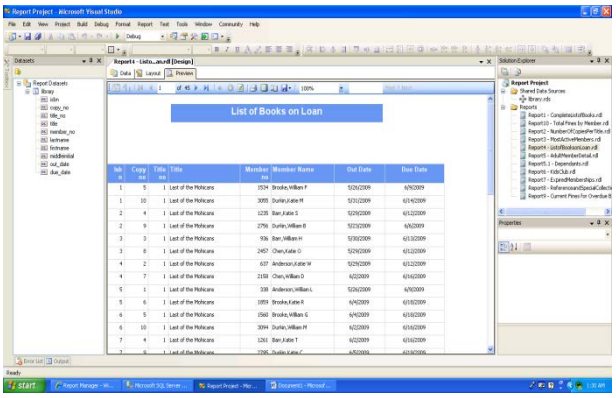


Fig.6. List Of Books On Loan

Similarly, Fig.7 shows another report 'Current fines for overdue books' where the overdue book fines are given as result in the test form.

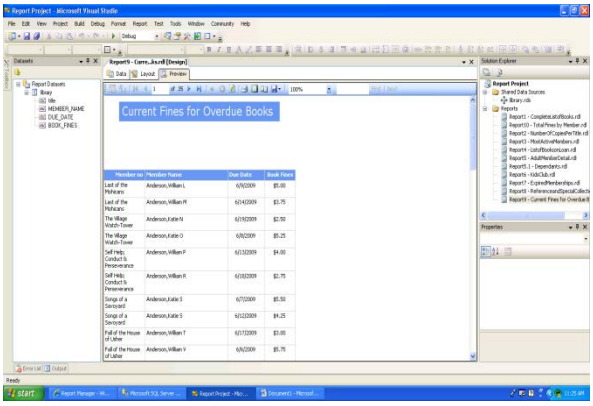


Fig.7. Current Fines For Overdue Books

We have seen two examples in each and every case of front-end queries and back-end stored procedures. In the similar way, the rest of the output test forms and reports in the library database are produced.

3.1. Translation Of SQL Queries In To Relational Algebra And Generating Query Tree

SQL queries are optimized by decomposing them into a collection of smaller units, called blocks. A typical relational query optimizer concentrates on optimizing a single block at a time. When a user submits an SQL query, the query is parsed into a collection of query blocks and then passed onto the query optimizer.

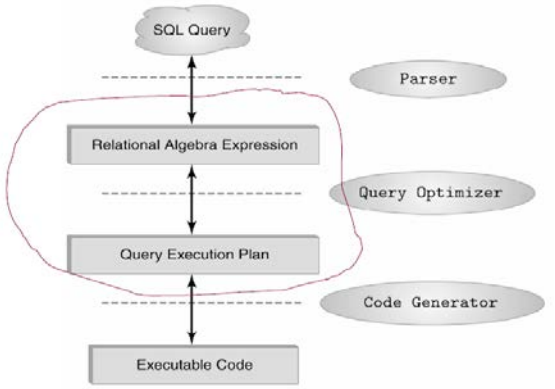


Fig.8. Query Optimizer

The optimizer examines the system catalogs to retrieve information about the types and lengths of fields, statistics about the referenced relations, and the access paths available for them. The optimizer then considers each query block and chooses a query evaluation plan for that block.

In given query, it essentially enumerates a certain set of plans and chooses the plan with the least estimated cost. The query blocks that contain two or more relations in the FROM clause require joins (or cross-products). Finding a good plan for such queries is very important because these queries can be quite expensive. Regardless of the plan chosen, the size of the final result can be estimated by taking the product of the sizes of the relations in the FROM clause and the reduction factors for the terms in the WHERE clause. But, depending on the order in which relations are joined, intermediate relations of widely varying sizes can be created, leading to plans with very different costs. In that process of enumerating multiple relation queries plan, query is taken as example for forming the cheapest plan.

3.1.1. Relational Algebra

Relational algebra is the query language associated with the relational model. Queries in algebra are composed using a collection of operators. Each relational query describes a step-by-step procedure for computing the desired output, based on the order in which operators are applied in the query.

For optimizing a query block, the SQL query must be converted into relational algebra expression.

3.1.2. Query Tree

Query tree is a tree data structure that corresponds to a relational algebra expression. It represents the input relations of the query as leaf nodes of the tree, and represents the relational algebra operations as internal nodes.

An execution of the query tree consists of executing an internal node operation whenever its operands are available and then replacing that internal node by the relation that results from executing the operation.

When we consider a case "Display available books" from front-end database queries as an example, As shown in fig.2,

CASE1: Display available books

CASE1: SQL QUERY:

```
SELECT i.ISBN, copy_no, t.title_no,
translation, title, author
FROM copy As c
INNER JOIN item As i
ON i.isbn = c.isbn
AND i.title_no = c.title_no
INNER JOIN title As t
ON t.title_no = i.title_no;
```

A query block is an SQL query with no nesting and exactly one SELECT clause and one FROM clause and at most one WHERE clause, GROUP BY clause, and HAVING clause.

So, the first step in optimizing a query block is to express it as a relational algebra expression.

Every SQL query block can be expressed as an extended algebra expression having this form. The SELECT clause corresponds to the projection operator, the WHERE clause corresponds to the selection operator, the FROM clause corresponds to the cross-product of relations, and the remaining clauses are mapped to corresponding operators in a straightforward manner.

The alternative plans examined by a typical query optimizer can be understood by recognizing that a query is essentially treated as an algebra expression. Translating that "CASE1: SQL QUERY" into relational algebra expression, we have

CASE1: RELATIONAL ALGEBRA:

$$\pi_{\text{I.ISBN, copy\_no, t.title\_no, translation, title, author}} \left(\left(\text{Copy} \bowtie_{\substack{C.ISBN = I.ISBN \\ C.title\_no = I.title\_no}} \text{Item} \right) \bowtie_{I.title\_no = T.title\_no} \text{Title} \right)$$

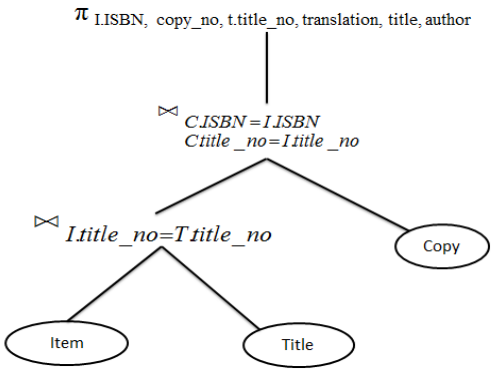
For estimating the cost of an evaluation plan for a query block, we consider query tree. For each node in the tree, we must estimate the cost of performing the corresponding operation. Costs are affected significantly by whether pipelining is used or temporary relations are created to pass the output of an operator to its parent.

Steps in converting a query tree during optimization involve:

- Initial (canonical) query tree for sql query.
- Moving SELECT operations down the query.
- Applying the more restrictive SELECT operation first.
- Replacing CARTESIAN PRODUCT and SELECT with JOIN operations.
- Moving PROJECT operations down the query tree.

So the converted query tree for "CASE1: RELATIONAL ALGEBRA" is,

CASE1: QUERY TREE:



Suppose that the following indexes are available: for Item and Title, a B+ tree index on the -(ISBN and title_no) fields and a hash tree on the title_no field(join method of item and title);similarly for Copy, a B+ tree index on the -(ISBN and title_no) fields and a hash tree on the ISBN and title_no fields.

The best plan is found for accessing each relation, regarded as the first relation in an execution. so the best plan for copy, item and title is obviously a file scan because no selections match an available index. Still now the plans generated are taken as outer relation and we consider joining another relation as the inner one. Hence, the following joins are processed: file scan of Item(outer) with Title(inner),file scan of Item(outer) with Copy(inner),file scan of Copy(outer) with Title(inner), file scan of Copy(outer) with Item(inner).

For each such pair, we consider every join method, and for each join method, we consider every available access path for the inner relation. For each pair of relations, we retain the cheapest of the plans considered for every sorted order in which the tuples are generated. Note that, since the result of the first join is produced in sorted order by title_no, whereas the second join requires its inputs to be sorted by ISBN and title_no, the result of the first join must be sorted by ISBN and title_no before being used in the second join. The tuples in the result of the second join are generated in sorted order by ISBN and title_no fields. For each plan retained, if the result is not sorted on ISBN and title_no, we add the cost of sorting on the ISBN and title_no fields.

The sample plan generated produces tuples in ISBN and title_no order, therefore, it may be the cheapest plan for the query even if a cheaper plan joins all three relations but does not produce tuples in ISBN and title_no order.

As we have seen one example of SQL query translating into relational algebra including the formation of query tree, we'll see another example for some more information.

CASE2: use UNION, list all member reserve the specific book.

CASE2: SQL QUERY:

```
SELECT i.ISBN, title, m.member_no, lastname
+','+ firstname As Name,
'Adult' As [MemberType] FROM adult As a
INNER JOIN member As m
ON a.member_no = m.member_no
INNER JOIN reservation As r
ON m.member_no = r.member_no
INNER JOIN item As i
ON r.isbn = i.isbn
INNER JOIN title As t
ON t.title_no = i.title_no WHERE i.isbn = 500;
```

UNION

```
SELECT i.ISBN, title, m.member_no, lastname
+','+ firstname As Name,
'Juvenile' As [MemberType] FROM juvenile As j
INNER JOIN member As m
ON j.member_no = m.member_no
INNER JOIN reservation As r
ON m.member_no = r.member_no
INNER JOIN item As i
ON r.isbn = i.isbn
INNER JOIN title As t
ON t.title_no = i.title_no WHERE i.isbn = 500;
```

Here we use the UNION operation to club two datasets ADULT and JUVENILE. When consider the relational algebra for optimizing that SQL query, the result can be given as,

CASE2: RELATIONAL ALGEBRA:

$$\pi_{\text{t.ISBN, title, m.member\_no, lastname + ',' + firstname As Name, a.MemberType = 'Adult'}}$$

$$((((\text{Adult} \bowtie_{Amember\_no=Mmember\_no} \text{Member}) \bowtie_{Mmember\_no=Rmember\_no} \text{Reservation}) \bowtie_{RISBN=IISBN} \text{Item}) \bowtie_{Ititle\_no=Ttitle\_no} (\sigma_{\text{t.ISBN = '500'}} \text{Title}))$$

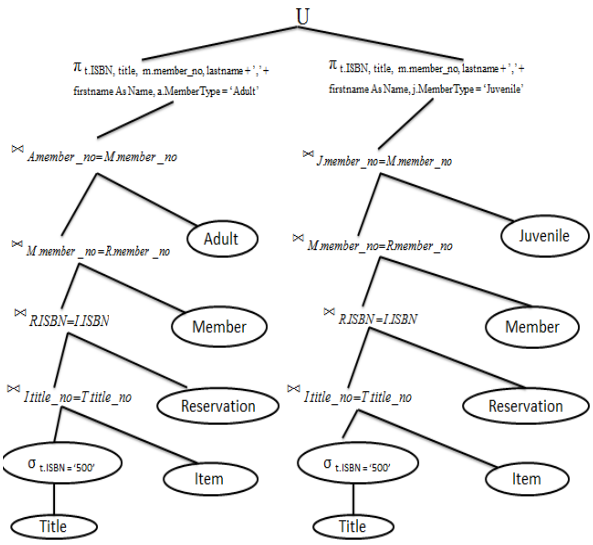
U

$$\pi_{\text{t.ISBN, title, m.member\_no, lastname + ',' + firstname As Name, j.MemberType = 'Juvenile'}}$$

$$((((\text{Juvenile} \bowtie_{Jmember\_no=Mmember\_no} \text{Member}) \bowtie_{Mmember\_no=Rmember\_no} \text{Reservation}) \bowtie_{RISBN=IISBN} \text{Item}) \bowtie_{Ititle\_no=Ttitle\_no} (\sigma_{\text{t.ISBN = '500'}} \text{Title}))$$

Many inner joins are consider for obtaining the required information taking into account the selection - t.ISBN = '500' and the optimizer uses UNION operation to gather wanted information from the datasets ADULT and JUVENILE. For estimating the cost from this query, consider query tree,

CASE2: QUERY TREE:



We have two datasets ADULT and JUVENILE combining with UNION operation. And the same information i.e, selection operator "t.ISBN = '500' " is required as a result in both the datasets. So the tree continues from UNION operation as shown in the figure, and continues with the left depth tree and the right depth tree which have the same indexes and joins to be considered.

First we'll estimate the cost for the left depth tree and the same cost is taken for the right depth tree which combines with a UNION operation.

The indexes available for this left depth tree query are: for Member, a B+ tree index on the member_no and hash tree on the member_no; for Reservation, a B+ tree index on the member_no and hash tree on the ISBN; for Item, a B+ tree index on the member_no and a clustered B+ tree on the ISBN field; for title, a B+ tree index on the ISBN(join method with title) field and a hash tree index on the ISBN(join method with title) field.

For (Adult, Member, Reservation, and Item), the best plan is obviously a file scan as seen in earlier case. The best plan for Title is to hash index on ISBN, which matches the selection t.ISBN = '500'. The B+ tree on ISBN also matches this selection and is retained even though the hash index is cheaper, because it returns tuples in stored order by ISBN field. Similarly, the joins processed in CASE2 are: all possible file scans for Adult, Member, Reservation, Item, Title as seen in earlier CASE1 and; Title accessed via B+ tree index on ISBN(outer) with Adult(inner), Title accessed via hash tree index on ISBN(outer) with Adult(inner),Title accessed via B+ tree index on ISBN(outer) with Member(inner),Title accessed via hash tree index on ISBN(outer) with Member(inner),Title accessed via B+ tree index on ISBN(outer) with Reservation(inner), Title accessed via hash tree index on ISBN(outer) with Reservation(inner), Title accessed via B+ tree index on ISBN(outer) with Item(inner), Title accessed via hash tree index on ISBN(outer) with Item(inner).

We consider here two examples for cheapest plan. First example, with Title accessed via hash index on ISBN as the outer relation, an index nested loops join accessing Item via the B+ tree index on ISBN(join method title) is likely to be a good plan; observe that there is no hash index on this field of Item. Another plan for joining Item and Title is to

access Title using the hash index on ISBN, access Item using the B+ tree on ISBN (join method title), and use a sort-merge order by ISBN (join method title).It is retained even if the previous plan is cheaper, unless an even cheaper plan produces the tuples in sorted order by ISBN (join method title). As told in the previous CASE1, considering the cost of sorting on member_no field, the cheapest plan is generated. From this, even right depth tree sorting cost is considered to be cheapest plan. And the result cost is clubbed by the UNION operation which overall is generated to be cheapest cost plan.

Similarly rest of the queries built in library databases are translated into relational algebra and the formation of query tree for estimating the least cost plan are done.

4. Conclusion

In this paper, we have described the query optimization of a single query block, which is expressed by translating SQL queries into relational algebra expression. We have implemented the design of library databases and used those SQL queries for the purpose of our optimization. Generating the cheapest cost plan is estimated in the formation of query tree. The process of finding a good plan for any SQL query gets complex when we require joins. Even for cases like this, we evaluated the cheapest query plan using left deep plans. But, the downside to this method of left deep plans is that, if the number of joins is more than 15 or so, analyzing the cost of optimization becomes complex.

5. References

[1] Raghu Ramakrishnan and Johannes Gehrke: 'Database Management Systems' - Introduction to database design and translation into relational algebra including tree structures, third edition, 2003, pages 25-110,344-385,478-507.

[2] Elmasri, Navathe: ' Fundamentals of Database systems', 2nd Edition,1994.

[3] A Swami, Optimization of Large join Queries Combining Heuristics and Combinatorial Techniques, in Proceedings of the 1989 ACM-SIGMOD Conference, Portland, OR, June 1989

[4] Harrington, Jan L.: 'Relational database design and implementation | clearly explained, third edition.

[5]Henk Ernst Blok, Djoerd Hiemstra and sunil choenni, Franciska de jong, Henk M. Blanken and peter M.G. Apers. Predicting the cost-quality tradeoff for information retrieval queries: Facilitating database and query optimization. Proceedings of the tenth international conference on information and knowledge management, October 2001, pages 207-214.

[6]Micheal L. Rupley, Jr.: 'Introduction to query processing and optimization'.

[7]Jamie MacLennan, ZhaoHui Tang, Bogdan Crivat: 'Data Mining with Microsoft SQL Server 2008'.

[8]Roger Jennings: 'Professional ADO.NET 3.5 with LINQ and the entity Framework', 2009.

[9]Carlos Coronel, Steven Morris, Peter Rob: ‘Database Systems- Design, Implementation and Management', ninth edition.

[10] 'Introduction to Databases and Programming with ADO.NET'- www.philadelphia.edu.jo/courses/ADO.NET/0782141838-1.pdf