ISSN 2278-3091
Volume 9, No.5, September - October 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse237952020.pdf
https://doi.org/10.30534/ijatcse/2020/237952020

Hierarchical Database Construction and Retrieval

Monday Eze[1], Doris Nnakwuzie[2], Chinyere Akobundu[3], Nneka Richard-Nnabu[4]
[1]Department of Computer Science, Babcock University, Nigeria. ezem@babcock.edu.ng
[2]Dept. of Computer Science, Alex Ekwueme Federal University, Nigeria. okafordoris49@yahoo.com
[3]Dept. of Comp. Sc. Evangel University, Akaeze, Ebonyi State.
chinyerekingsleyfavour@evangeluniversity.edu.ng
[4]Dept. of Computer Science, Alex Ekwueme Federal University, Nigeria. nneka4him@gmail.com

ABSTRACT
There are a number of scientific issues whose solutions require advanced database construction and retrieval techniques. One of such is in dealing with hierarchical database projects. Researchers have attempted to use a number of technological tools and techniques in this regard. This work utilizes Recursive Common Table Expressions (RCTE) to tackle this problem. The three objectives of this work are to demonstrate the construction of a small scale enterprise hierarchical database from the scratch, and to demystify hierarchical database retrieval using recursive query techniques. The practical case used in this work is a Hypothetical Small Scale Financial Institution, termed the HSSFI Bank. The actual implementation of this research was done using PosgreSQL in a Windows Environment, with the conceptualization, hierarchical construction procedures and the system outputs clearly demystified.

Key words: Hierarchical Database, Recursive CTE, PostgreSQL, ORDBMS, Data Retrieval, SQL.

1. INTRODUCTION

Hierarchical database is one in which the data is stored using a tree-like approach [1]. In other words, the contents are arranged in hierarchies, where apart from the root, every other node has a parent node [2]. Again, apart from the terminals, every other node has at least a child [3]. Based on these attributes, it can be deduced that hierarchical databases support one-to-many relationship [4], but not many-to-many relationship, unlike networks [5]. Moreover, because they support one to many relationships, the possibility of data redundancy is very high, and thus the necessity for developing data retrieval strategies that could prevent unnecessary redundancies [6]. The construction of hierarchical databases is achieved through creating pointer-like relationships between the parent nodes and children nodes. While research has shown that navigation of hierarchical databases could be fast [7], other researches have also shown that deletion of a node could lead to a cascaded deletion [8] of all other lower level linked nodes. This is another reason why hierarchical database access should be carefully designed and implemented.

2. RESEARCH TOOLS UTILISED

The database tool used for this research is PostgreSQL, an open source object-relational database management system (ORDBMS) [9]. PostgreSQL offers diverse features of modern databases such as complex queries, triggers [10], procedural languages, function aggregation, well established data types, among others. PostgreSQL also supports standard SQL, including a very seamless interface [11] with modern programing languages such as Python, Java, and so on. The major prerequisite for a successful implementation or replication of this work is the installation of PostgreSQL version 12.

3. PRACTICAL CASE

The major deliverable of this research is to demystify the construction of hierarchical database for a small scale organization, and to ensure seamless retrieval of the contents. The practical case study used is a Hypothetical Small Scale Financial Institution (HSSFI) which will henceforth be termed HSSFI Bank. In order to successfully construct a hierarchical database of the company organogram, code design was done for all the existing positions as will be further explained.

The HSSFI Bank is has a total of 36 employees. A graphical summary of the human resource distribution based on the positional codes is shown in Fig.1. The Bank has a Head Office, and four subsidiary branches called the North, South, East and West Branches respectively.

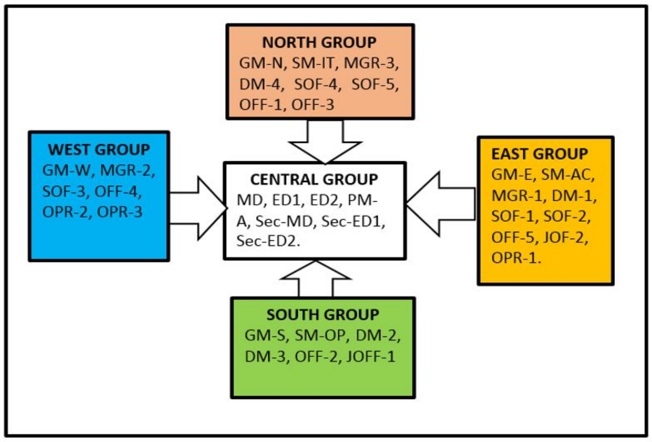

Figure 1: Graphical Distribution of HSSFI Bank Human Resources

At the top echelon of HSSFI Bank leadership is the Managing Director (MD), Executive Director for Finance(ED-F), Executive Director for Research (ED-R) and a Principal Manager for Administration (PM-A). The MD and the two EDs also have personal secretaries attached to them, designated as Sec-MD, Sec-ED1 and Sec-ED2 respectively. All these principal officers and their secretaries work in the Head Office as part of the Central Group. The four non-head office branches of HSSFI Bank are led by General Managers - GM-N for North, GM –S for South, GM-E for East and GM-W for the West respectively. At the lower cadres, there are three Senior Managers (SM-IT, SM-OP, and SM-AC) for Information Technology, Operations and Accounts respectively. Others are three Manager (MGR), four Deputy Managers (DM), five Senior Officers (SOF), five Officers (OFF), two Junior Officers (JOF) and three Operators (OPR).

4. HIERACHICAL DATABASE CONSTRUCTION

The construction of a typical hierarchical database requires a number of steps. Before delving into its, a researcher should carefully study the problem on ground, and confirm if indeed, it is most appropriate to use hierarchical database model to solve a particular problem, over other alternatives. This is because, at the moment, relational databases appear to be the most widely used of all the database models. However, the necessity to retain other database models alongside relational cannot be overemphasized. This is due to the fact that there are a number of scientific and technological problems that naturally fit into a hierarchical database model. Instances are in the areas of graph modeling [12], telecommunications, organizational organograms, automated strategic planning [13], geo-mapping, design of communities and collaborations [14], among others. A company organogram will be used in this work. It is also possible to maintain both relational and hierarchical database formats in a hybrid setting [15]. The system workflow is presented next.

4.1 System Workflow

The general system workflow for this research is shown in Fig. 2, and consists of five compartments. The first involves three major manual activities. Data Gathering entails collecting together the necessary human resources related information which will be used for further processing. Some of the information gathered at this stage are Staff Identification Number, Name of Staff, HSSFI Branch Location from where the staff operates, Position or grade of the personnel within the HSSFI Bank employment, information on whom a particular staff reports to, and so on. Code Design involves the manual generation of relevant codes, which are utilized at the hierarchy design stage, for the manual construction of the requisite organogram shown in Fig. 3. The use of node coloration is one way of increasing the clarity of the resulting node hierarchy. For instance, the five colours (White, Brown, Green, Blue and Yellow) were used in this work to clearly delineate the human resources reporting lines.

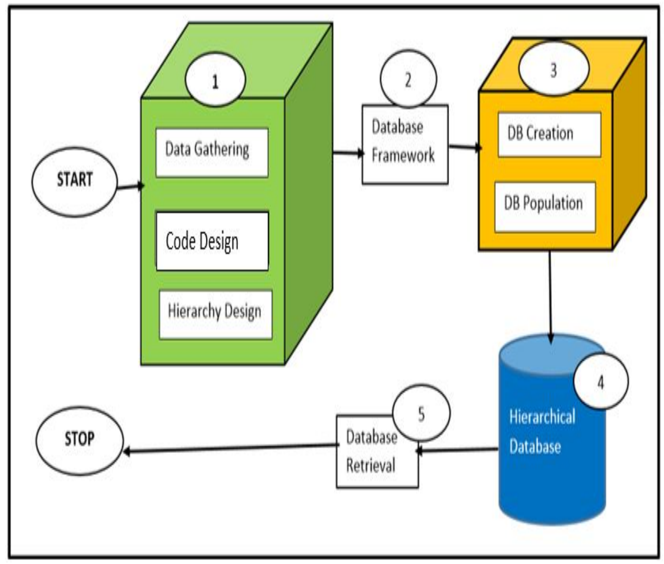

Figure 2: General System Workflow

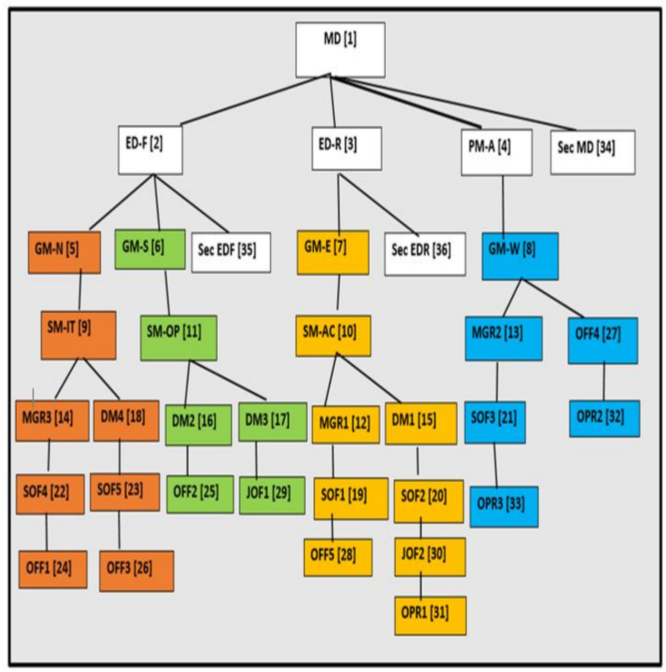

Figure 3: HSSFI Bank Organogram

The second step in the system workflow is to come up with the database frame work. The major deliverable at this stage is Table 1. The contents of this table is what is ported into the digitized format of the hierarchical database. Without any form of ambiguity, it can be deduced from the table that the Managing Director (MD) of NSSFI Bank Pius Nwada has Staff ID of value 1, works in the Central Branch of the organization. The columns of the table were designed to be as meaningful as possible. The column PCode stands for Position Code, signifying the grade of the personnel. Again, RepoCode (Reporting Code) represents the superior officer of a particular staff in question. The RepoCode for the MD is NULL because he does not report directly to any singular person, though he may report to the board of directors.

Table 1: Hierarchical Database Framework

StaffId	StaffName	PCode	Location	RepoCode
1	Pius Nwada	MD	Central	NULL
2	Nwanka Dalu	ED-F	Central	1
3	Eze Chidi	ED-R	Central	1
4	Kalu Roland	PM-A	Central	1
5	Ude Wolopa	GM-N	North	2
6	Yata Shikina	GM-S	South	2
7	Mapa Gayu	GM-E	East	3
8	Ukas Nkasi	GM-W	West	4
9	Yazi Layazi	SM-IT	North	5
10	Opuya Hope	SM-AC	East	7
11	Quad Gafa	SM-OP	South	6
12	Bolu Dozie	MGR-1	East	10
13	Bari Black	MGR-2	West	8
14	Mohama Rado	MGR-3	North	9
15	Lola Wole	DM-1	East	10
16	Paul Idiala	DM-2	South	11
17	Simon Moses	DM-3	South	11
18	Faith Gigi	DM-4	North	9
19	Adam Oshom	SOF-1	East	12
20	Gura Guri	SOF-2	East	15
21	Papa Idimma	SOF-3	West	13
22	Wisdom New	SOF-4	North	14
23	Jacob Baraba	SOF-5	North	18
24	Jare Zikenu	OFF-1	North	22
25	Coco Liko	OFF-2	South	16
26	Sarawak Biu	OFF-3	North	23
27	Jayeola Xio	OFF-4	West	8
28	Ishidima Welo	OFF-5	East	19
29	Peter Higha	JOF-1	South	17
30	John Ruela	JOF-2	East	20
31	Matthew Bam	OPR-1	East	30
32	Raba Ijeuwa	OPR-2	West	27
33	Ukwa Achina	OPR-3	West	21
34	Jaja Opoga	Sec-MD	Central	1
35	Igbere Faya	Sec-EDF	Central	2
36	Wada Naze	Sec-EDR	Central	3

The remaining three compartments of the system workflow will be discussed under the section for actual system implementation.

4.2 System Implementation

The construction of Hierarchical database as a key objective of this research involves the preliminary manual activities as itemized in the system workflow, plus the remaining digitization and implementation steps as will be discussed at this stage. These are creation of the database, creation of requisite tables, and populating the database tables accordingly. Thereafter, the next concern will be the process and technical details needed for data retrieval and interpretation of outputs from the resulting system. The Structured Query Language (SQL) [16] is an important tool used at this stage. In line with a standard nomenclature adopted for the database objects, the database and tables were named *hssfi_organo_database* and *hssfi_organo_table* respectively. The SQL CREATE command [17] was used in creating these two objects. An important precaution is the issuance of a windows command \c in between database and table creation sessions. This is to ensure that the incoming tables are domiciled within the right schema. This is shown in Fig. 4.

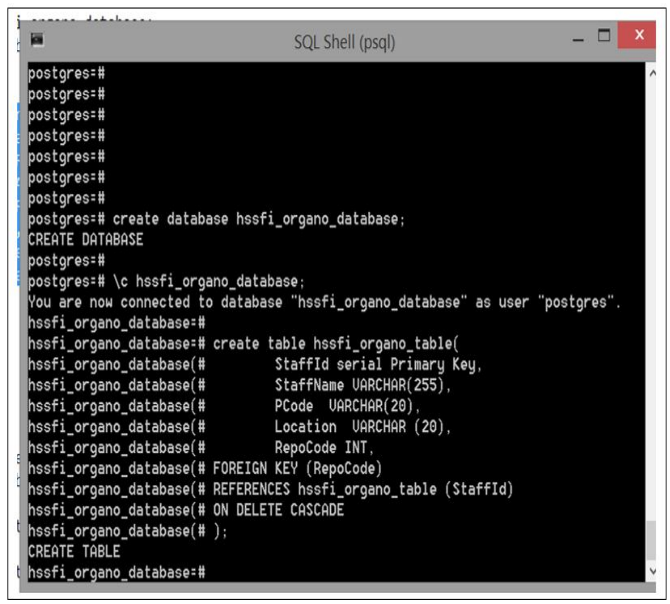

Figure 4: Database Object Creation Screen

It is also important to state that Foreign Key statement [18] was used to link the *RepoCode* to *StaffID*, with a Delete Cascade option [19]. This is to maintain referential integrity [20] in the resulting database table. Referential integrity is a very important issue in database security and accuracy [21], and especially in ensuring a smooth implementation of data retrieval through recursive common table expression (RCTE) [22] to be discussed at a later section of this paper.

The next stage in the construction of hierarchical database is to populate the *hssfi_organo_table*, which was achieved through the SQL Insert Command. To ensure that this operation was error free, the safe + multiple tuple insertion strategy was used [23] as will be briefly explained. As shown in Fig.5, there are four common paths for insertion into database tables. These are blind +single tuple, blind + multiple tuple, safe + single tuple, and safe + multiple tuple.

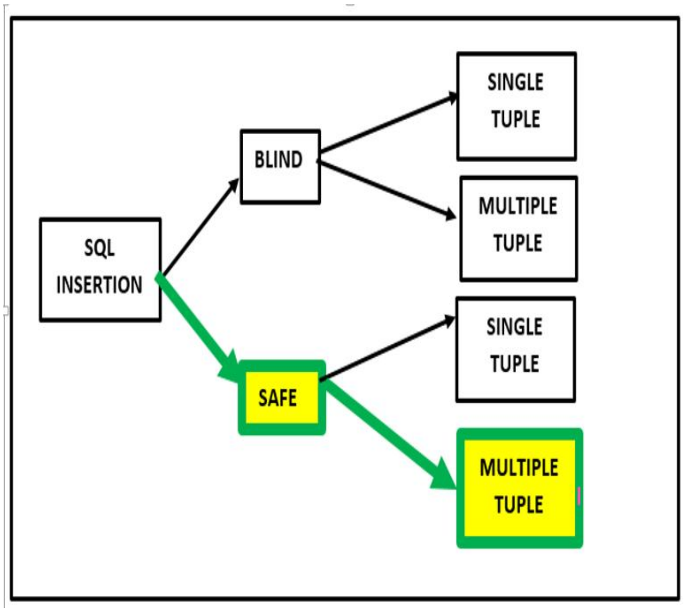

Figure 5: Safe and Multiple SQL Insertion Path

The safe + multiple strategy enforces the deliberate listing of the database fields in the insertion statement, and as well, ensures that the tuples to be inserted are listed in a multiple format, rather than listing them one by one. Two major advantages of taking this path are accuracy and speed, unlike the three other methods. The screen shots of this operation is shown in Fig. 6.

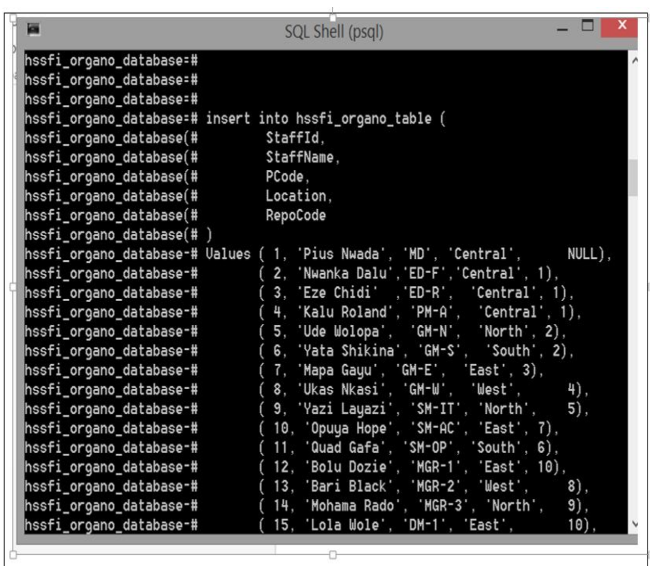

Figure 6: Database Data Population Screen

Thus, the organogram data of all the 36 staff of NSSFI Bank were inserted in the database in a one safe + multiple tuple insertion statement.

5. HIERACHICAL DATABASE RETRIEVAL

Beyond database design, creation and population of the resulting structure with requisite data, the next very important issue is the seamless retrieval of hierarchical data. Hierarchical Database retrieval is achieved using a technique known as Recursive Common Table Expression (RCTE) as will be demonstrated in this work. RCTE is an advancement in Common Table Expression (CTE). CTE are a special database query mechanism used to build temporary named result sets, which can be re-used over and over again. A typical CTE is constructed by embedding other queries within a WITH.. AS statement [24]. Research has shown that the use of CTE can improve data retrieval speed appreciably over a normal sequential SELECT statement, especially with appropriate indexing [25]. Research has also shown that apart from data retrieval through SELECT, other statements such as INSERT, UPDATE, DELETE, among others can be conveniently used within a CTE [26]. A number of modern databases support CTE syntax, some of which are PostgreSQL, Microsoft SQL Server, Teradata, DB2, Firebird, Oracle, SQLite, among others [27]. However, in Oracle Installations, the term recursive subquery factoring is used for CTE [28], and in higher versions of Oracle such as 10g and above, the CONNECT BY [29] hierarchical data retrieval feature is also supported as an alternative retrieval strategy.

5.1 Recursive CTE Structure

The general syntax of a RCTE in PostgreSQL is

```
WITH RECURSIVE CTE-Name AS (
NR-Section
UNION | UNION ALL
RC-Section
)
Main Query Invoking CTE_Name
```

It is clear from the syntax that every RCTE begins with the keyword WITH RECURSIVE followed by the name of the CTE, the keyword AS and then a bracket that encloses the body of the CTE. The body of an RCTE consists of two sections. These are the Non-Recursive (NR) Section and the Recursive (RC) Sections respectively [30]. Another name for the former is the anchor part. One major differences between an RCTE and an ordinary CTE is that the former has two sections, while the later has only one section. The two sections in an RCTE are joined using a UNION or UNION ALL statement [31]. The difference between these two options is that UNION ALL allows repetition of outputs, unlike UNION keyword that ensures that all records in the final result occur only once [32]. The power of RCTE is hinged on the fact that it has the ability to invoke itself [33], and thus the recursively concept of CTE. The final part of the RCTE is the main query definition which makes reference to the CTE Name. It is important to note that no comma or semi-colon is allowed in between the final definition of CTE and the main query [34], a rule that many programmers usually forget, which commonly causes system errors in RCTEs.

5.2 RCTE Implementation

In this research, the system nomenclature [35] was made as meaningful as possible. The RCTE name is *HierachicalDemo*, and the raw code is shown Fig. 7. In the source code shown, the anchor and recursive sections can be clearly identified as separated by the UNION keyword. Another SQL keyword that finds relevance in the development of RCTE is INNER JOIN.

```
WITH RECURSIVE HierachicalDemo AS (
        SELECT StaffId, StaffName,
        PCode, Location, RepoCode
        FROM hssfi_organo_table
        WHERE StaffId = 5

  UNION

  SELECT emp.StaffId, emp.StaffName, emp.PCode,
        emp.Location, emp.RepoCode
        FROM hssfi_organo_table emp
        INNER JOIN HierachicalDemo
        ON HierachicalDemo.StaffId = emp.RepoCode
)
SELECT * FROM HierachicalDemo;
```

Figure 7: Code for RTE Definition

In SQL, a JOIN operation [36] is used to retrieve data from multiple tables in a single SELECT query. Thus, two tables can be combined by a single join operator, and the result can as well be joined again with other tables. One important condition is that the tables being joined must have a same or similar columns in order for them to be joined. There are many types of JOINS methods, however, the INNER JOIN [37] eliminates all rows that failed to match the join condition exactly. The main query part of the RCTE is the last select statement. In the code listing shown, the required input is the StaffID. Based on the hard-coded input, the system will recursively display all the personnel of HSSFI Bank that report to the staff having StaffID = 5. This input can be changed for other cases or inputs.

5.3 System Output

The system outputs after execution of the RCTE for StaffID =5 is shown in Fig. 8. This result compares favorably with the HSSFI Bank organogram shown in Fig. 3, and the Hierarchical Framework shown in Table 1. In order to ensure a complete evaluation for system accuracy, the source code [38] was executed for all the 36 possibilities StaffID = {x: x =1, 2, …. 36}, and the outcome coincided with the contents of the manually generated organogram as well as the framework in Table 1.

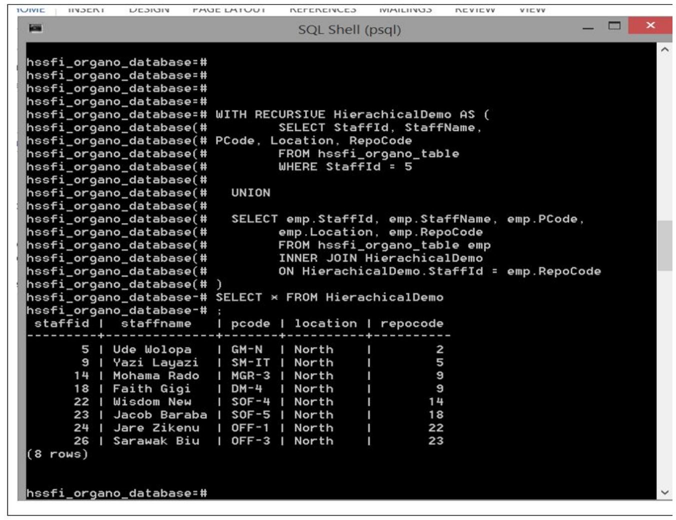

Figure 8: System Listing of all Subordinates to Staff Number 5

6. CONCLUSION

This research has presented the theory and procedures for design and construction of hierarchical databases. A practical case was demonstrated using a Hypothetical Small Scale Financial Institution (HSSFI) referred to in this work as HSSFI Bank. The use of RCTE for data retrieval from Hierarchical Database was also presented. At this stage of this work, it is clear that a CTE is a kind of temporary result set, quite similar to a derived or subquery table [39]. Again, unlike stored tables, a CTE only lasts within the query duration. Thus, a CTE is much more versatile and powerful than a derived table because it can be self-referencing, and could be referenced several times in a single query [40]. The recursively of RCTE gives it more speed and versatility than an ordinary CTE, and far above those of ordinary sub-queries. In conclusion, this work has presented a concise methodology [41] for construction, data population and retrieval in hierarchical databases. It is hoped that this work will be useful to other researchers who may find it a foothold for further research in hierarchical database construction and retrieval.

REFERENCES

1. S. James, P. Prakash and R. Nandakumar. **The Tree List – Introducing a Data Structure.** International Journal of Recent Technology and Engineering, 2019, Vol 7, Issue 6, 1093-1096.

2. R. Dhankah, S. Kamra and V. Jangra. **Tree Concepts in Data Structure**. International Journal of Innovative Research in Technology. 2014. Vol 1, Issue 7, 183-185.
3. A. Gupta and A. Lata. **Dynamic Trees in Data Structure.** International Journal of Engineering Research and Management Technology. 2014. Vol 1, Issue 5, 7-12.
4. E. Gummesson. **From One-to-One to Many-to-Many Marketing**. In Proceedings from QUIS 9, Karlstad, Sweden. 2004. pp1-11.
5. C. Mancas. **On Database Relationships Vs Mathematical Relations**. Global journal of computer science and technology: C Software and data engineering, 2016. Vol 16, issue 12-16
6. S. Saha. **Efficient Methods for Reducing Data Redundancy in the Internet.** A PhD Dissertation submitted to Aalto University School of Science, October 2015.
7. R. Ngolikar, R. Khandal and R. Mohare. **Comparison of HDBMS, NDBMS, RDBMS and OODBMS.** International Journal of Advanced Research in Computer Science and Management Studies, **2015**. Vol 3, Issue 6, p119-126
8. K. Raval. **A study on Oracle data constraints**. International Journal of Advanced Research in Computer Science and Management Studies, 2013. Vol 1, Issue 1, p1-4
9. M. Velicanu and I. Botha. **Solutions for the Object – Relational Databases Design**. Database Systems Journal, 2011, Vol 2, No 4, p51-64
10. H. Paci, E. Kajo and A. Xhuvan. **Protecting Oracle PL/SQL Source Code from a DBA User.** International Journal of Database Management Systems. 2012. Vol 4, No. 4, 43-52
11. Y. Abass, Z. Zahoor and S. Irfan. **Common Database Interface with NLP**. International Journal of Computer Science and Mobile Computing, 2017. Vol 6, Issue 6, 195-199.
12. N. Tyagi and N. Singh. **Graph Database – An Overview of its Applications and its types**. International Journal of Computer Science Engineering Techniques, 2017. Vol2, Issue 3, p6-10
13. G. Valentim and C. Henrique. **How to develop Technology Roadmap? The case of a Hospital Automation Company.** Production, 2016. Vol 26, No. 2, 354-358
14. K. Haddad, R. Lindquist-Grantz, H. Vivens, A. Boards, F. Jacquez and L. Vaughn. **Empowering Youths to build Bridges: Youth Leadership in Suicide Prevention. Collaborations**: A Journal of Community – based Research and Practice, 2020. Vol 3, No.1, p1-10
15. D. Martinez-Mosquera, R. Navarrete and S. Lujan-Mora. **Modeling and Management Big-Data in databases – A Systematic Literature Review.** Sustainability, 2020. Vol 12, No 634. 1-41.
16. A. Kumar. **Structured Query Language (SQL) Answering Model for User Queries based on Intuitionistic Fuzzy Logic.** International Journal of Computer Applications, 2015. Vol 129, No.1, 32-36
17. K. Elshazly, Y. Fouad, M. Saleh and A. Sewisy. **A Survey of SQL Injection Attack Detection and Prevention.** Journal of Computer and Communication. 2014. Vol 2. 1-9
18. L. Jiang and F. Naumann. **Holistic Primary Key and Foreign Key Detection**. Journal of Intelligent Information Systems, 2020. Vol 54, 439-461.
19. K. Raval. **A Study on Oracle Data Constraints**. International Journal of Advance Research in Computer Science and Management Studies, 2013. Vol 1, Issue 1, 1-4
20. C. Ordenez and J. Garcia-Garcia. **Referential Integrity Quality Metrics**. Decision Support System Journal, 2008. Vol 44, No. 2, 495-508.
21. H. Pourzargham. **Importance of Security in Database.** International Journal of Computer Science and Network Security, 2015. Vol 15, No.5, 29-31.
22. S. Sowndarya and A. Sridhar. **Performance Comparisons of Common Table Expressions and Cursors**. International Journal of Computer Science and Engineering, 2012. Vol 4, No. 7. 1362-1365.
23. IBM Corp. **Database SQL Programming.** International Business Machine. IBM i, Version 7.3, 2015. Product Number 5770-SS1.
24. P. Garner. **Learning SQL in Steps**. Systemics, Cybernetics and Informatics, 2015. Vol 13, No. 4, 19-24.
25. R. Talib. **A Comparative Study of Indexing using Oracle and MS-SQL Server for Relational Database Management Systems.** International Journal of Computer Science and Mobile Computing, 2018. Vol 7, Issue 12, 341-350.
26. Daniel Bartholomew. **MariaDB and MySQL Common Table Expressions and Window Functions Revealed.** Apress Publishers, 2017. Berkeley, CA.
27. K. Kolonko. **Performance Comparison of the most popular relational and non-relational database management systems.** An MSc in Software Engineering Dissertation submitted to the Faculty of Computing, Blekinge Institute of Technology, Karlskrona, Sweden, February 2018.
28. K. Morton, K. Osborne, R. Sands, R. Shamsudeen and J. Still. **Sub Query Refactoring. In: Pro Oracle SQL.** Apress Publishers, 2013. Berkeley CA, pp261-298
29. M. Cyran. **Oracle Database Concepts,** 10g Release 2(10.2). Oracle Corporation, 2005.
30. M.K. Rohil and N. Gupta. **Visualization of Recursive Database Query Output Using a Declarative System: An Illustration through Parts – Assembly Tree Visualization**. 2019 IEEE 5th International Conference for convergence in technology, Bombay, 2019. p1-4.
31. N. Kumari. **SQL Server Query Optimization Techniques – Tips for writing Efficient and Faster Queries.** International Journal of Scientific and Research Publications, 2012. Vol. 2, Issue 6, 1-4.

32. C. Gabriel, M. Mihai, V. Luca and O.Teodor. **Query Optimization Techniques in Microsoft SQL Server**. Database Systems Journal, 2014. Vol 5, No. 2, p33-48.
33. T. Green, S. Huang, B. Thau and W. Zhou. **Data Log and Recursive Query Processing**. Foundations and Trends in Databases, 2012. Vol. 5, No.2, 105-195.
34. L. Jachiet, P. Greneves, N. Gesbert, and N. Layaida. **On the Optimization of Recursive Relational Queries: Application to Graph Queries**. SIGMOD 2020 – ACM International Conference on Management of Data. June 2020, Portland, USA. Pp 1-23.
35. S. Mubeen. **Synthesis of Circular Array Antennas Using Accelerated Particle Swarm Optimization**. International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE). Vol. 8, No. 5, Sep-Oct 2019, 2121-2125.
36. R. Batra. **SQL Primer - An Accelerated Introduction to SQL Basics.** APress Publishers, Haryana, India, 2018.
37. A. Taylor. **SQL For Dummies 9th Edition**, Published by: John Wiley & Sons, Inc., 2019. Hoboken, New Jersey.
38. S. Padda, A. Arora, S. Gupta and P. Sharma. **Review of Software Methodologies used in Software Design**. International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE). Vol. 3, No. 5, Sep - Oct. 2014, 88-93.
39. S. Thanh and L. Nguyen. **Query – subquery nets for Horn Knowledge-Bases in First – Order Logic.** Journal of Information and Telecommunication, 2017. Vol. 1, Issue 1. Pp79-99
40. I. Ben-Gan. **Inside Microsoft SQL Server 2008: T-SQL Querying.** Published by Microsoft Press, Redmond Washington. 2009.
41. V. Nam, P. Chuan, L. Manh and Q. Khanh. **A Review on Security-Aware Routing Protocol for Mobile Adhoc Network**. International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE). Vol. 9, No. 3, June 2020, 3655-3661