An effective approach to learn Logical Joins operator in relational theory and SQL: A Case study

Ashfaq Ahmed
Department of Computer Science
University of Management and Technology
Lahore, Pakistan
ashfaq.ahmed024@gmail.com

Dr.Muhammad Shoaib Farooq
Department of Computer Science
University of Management and Technology
Lahore, Pakistan
Shoaib.farooq@umt.edu.pk

Muhammad Ishaq Raza
Department of Computer Science
University of Management and Technology
Lahore, Pakistan
ishaq.raza@nu.edu.pk

Adnan Abid
Department of Computer Science
University of Management and Technology
Lahore, Pakistan
Adnan.abid@umt.edu.pk

Ali Raza
Department of Computer Science
University of Management and Technology
Lahore, Pakistan
aliirazii@gmail.com

Abstract*— *Teaching joins to beginners is a complex task .The join is one of the most significant query operations of the relational database. Joins help in accessing the data from one or more relations based on Cartesian product. Join operators are used for access the data from relations. Join operators are the most important concept and skill for students taking a course in database design and implementation specifically those majoring in CS/IT/SE/EE/BBIT. All of the database textbooks partially cover the topics of Joins and operators but the full set of join operators are not fully exemplified. The concept of join is the most challenging operation for students to implement efficient and effective relational algebra expression. The join is an expensive operation in term of I/O. A lot of research has been applied to the optimization of join query operation. Usually, query execution takes more time than query optimization. In the case of a highly normalized relation, very large databases, the frequency of join operation is very high. To perform join query more efficiently the selection of join operators for an optimal result is very important. With the best of our knowledge, no text or curriculum fully cover the topics of join and there related operations, it creates a deficiency in students to write efficient and effective queries, on industry real life problems. Occasionally it is very tough for readers, research scholars, students to learn and teach Join operators. The aim of this research is the evaluation of join operators for query optimization for teachers, learners to improve their database concepts, implements the join operators in Relational database, visualize join table's diagrams with real life practical examples, and define the best sequence of teaching. I also clarified which types of join require ANSI Syntax and which could be constructed with traditional syntax

Keywords— *Cartesian product, ANSI, Join operators, Relational algebra*

I. INTRODUCTION

In relational database management system, join is one of the most important query operation [1]. E.F. Codd at IBM in 1970 invented relational database which is a collective set of multiple data sets organized based on relational model [18]. The software system, which is used to maintain relational database, are known as relational database management systems (RDBMS). Almost all relational database systems used structured query language (SQL) for maintain and querying the database. Therefore, a join is a query operation that retrieves information from more than one table [2]. Join operator is required when we need information which is not retrieved in a single relation [2]. When we require to join more than two tuples from two or more different relations then we require join [1]. Tuples are combining when they fulfill the join condition. The resulting tuples always become a new relation [2], which might be made from more than one relation. There is no available defined sequence of teaching for these contents and sub contents in the course outlines given by instructors and in computing curriculum. Due to this issue, faculty defines its own variants and teaching sequence. To avoid these uncertainties and ambiguities we propose a

framework for the evaluation of join in relational database. Our research will facilitate Teachers to improve their outline, implements the join in Relational database, visualize join table's diagrams with practical examples, and define the best sequence of teaching.

This paper presents each features using the following ingredients:(i) define a type of join formally; (ii) discuss join with SQL syntax according to ISO/ICE-2011 standers; (iii) discuss join with relation algebra (iv)discuss join with example case study (v) discuss DMBS support to joins. The rest of this dissertation is organized as follows. Section 2 is dedicated to Literature review, section 3 present logical joins formally with SQL syntax according to ISO/ICE-2011 standers, and Section 4 presents the conclusion and future direction.

II. LITERATURE REVIEW

Codd introduced the join operator in relational algebra [26]. Join is the most widely useful operator to retrieved data from multiple relation. The joins were discovered in the 1980s and implemented in the most of the relational database management systems since then, and provide many improvements [5] [6] [4].Mishra et al. [1] surveyed the different types of joins and various implementation-joining techniques. Gotlieb et al. [16] evaluate how one relational operation of join could be computed efficiently. The evaluation is based on computation time and storage usage.

Cao et al. [17] explains that self joins are cases of similar relation with advantage to using both inputs. Bhanuprakash et al. [3] introduce the concept of SQL joins in relational algebraic notations with examples. The aim of this paper was to learn the all fundamental types of joins with algebraic notation.

Syrdal et al. [8] calculate the joins cost that intended to guess the extent of the discontinuity of audible explains by the combination of two particular unit. Swami [13] produce optimize result for large join queries based on combinatorial and heuristic technique. Yang et al. [9], compare the performance of all types of join methods and provide an opportunity for choosing the best one based on performance and cost. Starner [11] used time interval in combination with timestamp data type, implement SQL-92 and provide a stander in which introduce a useful technique for addressing the period of time. Define the difference between internal data type and scalar data type, and conclude internal data type is much better than the scalar data type because the internal data type, which is central to the inner, join operation. Gold et al. [7] proposed a partitioned SQL join that reduced inter-stock reads.

Bernstein et al.[25], explains the concrete class of the relational queries that are resolved using semi-join operator. Further, described linear-time membership test. Xu bet.al [27] introduces an efficient and easiest outer join algorithm known as "outer-join Skew optimization (OJSO) so that the scalability is improved and enhance the performance of outer joins. Shinichi Morishita introduce a novel algorithm that develop applications from CPF ordering of joins, use applications consisting of joins, semi-joins and estimates for calculating same relation join.[30]. Wang et al.[31] presented a hierarchy of joins operation which are related to REA data model, it then further explain the combination of different modules related to REA model, the paper also presents findings on join operation and their linked to AIS document and report.

III. THE TAXONOMY OF LOGICAL JOIN OPERATOR

According to "ISO/IEC CD 9075-2:2013(E)" ANSI-Stander SQL classify seven types of join operators cross join, inner join (theta join, equi-join, natural join, named column join), outer join (left, right, full) [32]. Some join operators like semi-join (right, left), anti-join are no direct operator support in relational algebra, we can get semi-join and anti-join by using EXIST/IN/ALL functions in correlated sub queries [18]. As a special case, self-

As a special case, self-join can be join a table to itself [36][35] [34]. SQL describe joins in two distinct syntactical ways: explicit and implicit notations or syntax. Implicit join syntax abhorred in 1992, its uses is not a batter approach but some DBMS still support it. We have made a taxonomy of join operators to describe how join operators are related to each other. In the following (figure 2) taxonomy scheme, possibly cover all types of join in relational database.

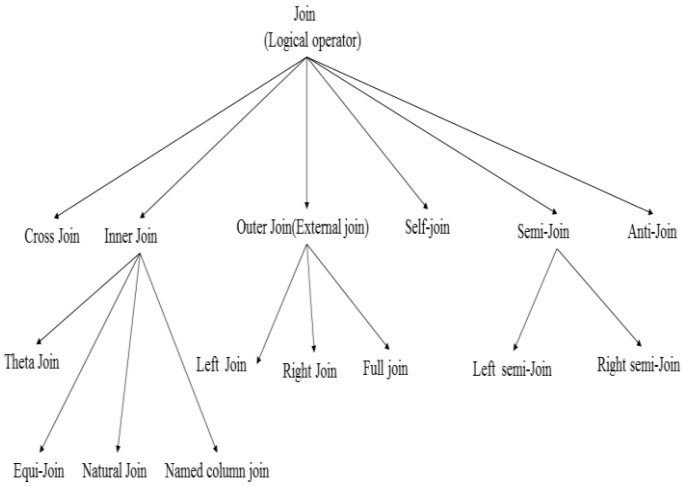

Figure 1. Taxonomy of joins

IV. CASE STUDY

A part of university database schema which includes student and society relation (table). University organize different student societies in such a way that one student is batch representative in each batch and each society is managed by a student known as head_student. We are interested only the head_student information in society table. Some students are participating in societies individually. In this schema, roll_no is the primary key of student table and batch_rep is foreign key, society_id is the primary key and head_student is foreign key in society table. The sample tables illustrate that batter understanding of this relation.

Table 1: Student(R)

roll_no	student_name	cgpa	batch_rep
101	Tahreem	3.3	103
102	Izaan	3.0	103
103	Isbah	3.4	Null
104	Ismail	2.5	105
105	Alia	2.7	Null

Table 2: Society(S)

Society_id	Society_ name	Head_student
1	Softec	102
2	ACM Lahore Chapter	104
3	Creative	Null
4	Sports	102

Cross-join

Cross-join returns the cartesian product of tuples from relations (table) in the join. Cartesian product produces all the combination of tuples from two relations. A general form of Cartesian product is (R×S) [22][15][33]. If a join query process has no joining condition between two tables, Query engine return their Cartesian product [15].

Table 3: Cross join syntax in relational algebra and SQL

RA	Result← R×S.
SQL	ISO/IEC-2013(E) syntax [32] SELECT<column list> FROM<left joined table>CROSS JOIN<right joined table> Alternate syntax[22]: SELECT<column list> FROM<left joined table>, <right joined table>

Table 4: Example of cross join in Relational Algebra and SQL

RA	Result← Student× Society
SQL	SELECT * FROM student CROSS JOIN society; Alternate example: SELECT * FROM student, society;

Table 3 show the join syntax with relational algebra and SQL. Table 4 shows the example query of our schema in relational algebra expression and SQL .

Inner Join (⋈):

Joins that generate a result set that contain only tuples of joining table that fulfil the joining condition are known as inner joins [20]. Tuples that do not fulfil the join condition are no included in the joined table or resultant table [20]. Inner join is a widely useful join operator in applications but should not be consider a good approach in all situations.

Table 5: Inner Join Syntax in Relational Algebra and SQL

RA	Result← R $\bowtie_c$ S.
SQL	ISO/IEC-2013(E) SQL syntax [32]. SELECT <column list> FROM<Left Joined Table>[INNER] JOIN <Right joined table> ON <join condition> Alternate syntax[22]: SELECT <column list>s FROM<Left Joined Table>, <Right joined table> WHERE<join condition>

Table 6: Inner Join example in Relational Algebra and SQL

RA	Result← student $\bowtie_{roll\_no=head\_student}$ Society.
SQL	SELECT * FROM student INNER JOIN society ON student.roll_no=society.head_student; Alternate Example: SELECT * FROM student, society WHERE student.roll_no=society.head_student;

Table 7: Result of Inner Join operation

roll_no	Student _name	cgpa	Batch _rep	Society _id	Society _name	Head _student
102	Izaan	3.0	103	1	Softec	102
102	Izaan	3.0	103	4	Sports	102
104	Ismail	2.5	105	2	ACM Lahore Chapter	104

Natural join (⋈):

Natural join introduce by E.F coded in relational algebra [26].The natural join (⋈) is a binary operation that is transcribed as R ⋈S where R and S both are relations [20]. The natural join returns the result set of all combination of rows in S and R relation that equivalent on their common attribute names. Natural join is always an equi-join because it has concealed (hidden) join condition-similar of common named attributes [29]

Table 8: Natural Join syntax in Relational Algebra and SQL

RA	Result ←R ⋈S
SQL	ANSI (ISO/IEC-2013(E)) syntax [32]. SELECT<column list> FROM<right Joined Table> NATRUAL <join type> JOIN <left joined Table>

Table 9: Example of Natural Join in Relational Algebra and SQL

RA	Result← student⋈($\rho_{(society_id,society_name,roll_no)}$Society)
SQL	ANSI (ISO/IEC-2013(E)): SELECT * FROM student NATURAL [INNER] JOIN society; OR SELECT * FROM student NATURAL [OUTER] JOIN society; Note: Inner Keyword is optional

Table 10: Result of Natural Join Operator

Roll _no	Student _name	cgpa	batch_rep	Society _id	Society _name
102	Izaan	3.0	103	1	Softec
102	Izaan	3.0	103	4	Sports
104	Ismail	2.5	105	2	ACM Lahore Chapter

Named columns join

Name column join is a new (enhance) version of natural join [31]. It is more flexible to natural join. It is mandatory to use the name columns in the joins condition. Then, it is not as dangerous [31]. Named column joins always are equi-joins .The type of join can be one of the following: blank (no join type define)-means inner join by default, outer is an optional keyword in right [outer], left [outer], full [outer] [31].

Table 11 show the join syntax with relational algebra and SQL. Table 12 shows the example query of our schema in relational algebra expression and SQL. Table 13 shows the result of Named columns join operation on our designed schema.

Table 11: Named Column in Relational Algebra and SQL

RA	Result ←R ⋈S.
SQL	ANSI (ISO/IEC-2013(E)) syntax [32] SELECT* FROM R JOIN S USING(column R, column S);

Table 12: Example of Named Column join in Relational Algebra and SQL

RA	Result← student⋈($\rho_{(society_id,society_name,roll_no)}$Society)
SQL	SELECT* FROM student JOIN society USING(roll_no, head_student);

Table 13: Result of named Column join operator

roll _no	student_ name	Cgpa	batch _rep	society _id	Society _name
102	Izaan	3.0	103	1	Softec
102	Izaan	3.0	103	4	Sports
104	Ismail	2.5	105	2	ACM Lahore Chapter

Theta join (θ):

Theta join is a more meaningful way of combining two relations [37]. It Generate all the combination of tuples from R and S that fulfill the joining condition, proper representation is $R \bowtie_{<joining\ condition>} S$.it is a binary operation which is denoted by $\bowtie_{\theta,\ Where}$ ⋈is the join notation and the notation in the subscript is the Greek latter theta. Theta join can be represented by using a Cartesian product and a selection expression while selection checks the joining condition is fulfill [1]. It allows us to merge the Cartesian product and selection into one operation. It produce the cartesian product of two relation, and then performs the selection using the predicate θ **Error! Reference source not found.**.

$$R \bowtie_\theta S = \sigma_\theta (R \times S)$$

There are some join conditions possible in a theta join i.e. (<, >, =,>=, <=) matching tuples from different tables or relations. Theta join has potentially (n*m) tuples in the resultant table, but in some cases the cardinality of result is much lesser [37].

Roll _no	Student _name	Cgpa	Batch _rep	Society _id	society name	Head _student
102	Izaan	3.0	103	2	Softec	102
102	Izaan	3.0	103	4	Sports	102
104	Ismail	2.5	105	4	ACM Lahore Chapter	104

Table 14: Theta Join Syntax in Relational Algebra and SQL

RA	Result← R $\bowtie_{condition}$ S
SQL	ANSI (ISO/IEC-2013(E)) syntax [32] SELECT* FROM R JOIN S ON <joining condition>

Table 15: Example of Theta Join operation in Relational Algebra and SQL

RA	Result← $(\rho_S(Student) \bowtie (\rho_{BR}$Student) $\sigma an\ rk = CGPA(Student\ (S)) \times Student(BR)$
SQL	SELECT * FROM S JOIN Student BR ON S.cgpa<BR.cgpa

Table 16: Result of Theta Join Operator

roll_no	student_name	Cgpa	batch_rep
101	Tahreem	3.3	103
102	Izaan	3.0	103
104	Ismail	2.5	105

Equi-join (=)

Theta join has a sub type called Equi join. Equi join type is a unique type of join that provides us with the combination of rows from two relation such as R and S that meet joining condition based on equality comparison [18]. The equijoin operation is denoted by $Student \bowtie_{<joining\ condition>} Society$ [13].

Table 17: Equi join syntax in Relational algebra and SQL

RA	Result← R$\bowtie_{R.c1=S.c2}$ S
SQL	ANSI (ISO/IEC-2013(E)) syntax [32] SELECT *FROM R JOIN S ON R.c1=S.c2

Table 18: Example of Equi Join in Relational Algebra and SQL

RA	$Result \leftarrow Student \bowtie_{student.Roll_{No}=Society.Head\_Student} Society$
SQL	SELECT * FROM student JOIN society ON R.roll_no=S.batch_rep

Table 19: Result of Equi Join operator

roll_ no	student_ name	Cgpa	batch_ rep	society_ id	society name	head_ student
102	Izaan	3.0	103	2	Softec	102
102	Izaan	3.0	103	4	Sports	102
104	Ismail	2.5	105	4	ACM Lahore Chapter	104

Table 17 show the join syntax with relational algebra and SQL. Table 18 shows the example query of our schema in relational algebra expression and SQL .Table 19 shows the result of Equi-join operation based on our designed schema.

Outer Join

The joins that generate a joined relation that include all tuples from the outer table, nevertheless of whether or not the section that generate matching tuples, are known as outer joins[20]. There are three types of outer joins: Left outer join, Right outer join, and full outer join.

a) **Left outer join (⟕)**

This joining query mainly looking for matching records and remaining records from left table [2] [12]. A left outer join returns all the values from the left table and matched values from the right table (NULL in the case of no matching join predicate) [13]. Table 20 show the join syntax with relational algebra and SQL. Table 201 shows the example query of our schema in relational algebra expression and SQL .

Table 20: Syntax of Left Outer Join in Relational algebra and SQL

RA	$Result \leftarrow R \bowtie_{R.c1=S.c2} S$
SQL	ANSI (ISO/IEC-2013(E))syntax [32]: SELECT <column list> FROM<left joined Table> [OUTER] JOIN <right joined table> ON <Join condition> Alternate syntax: Alternate syntax**Error! Reference source not found.**:oracle support this deprecated syntax SELECT * FROM R, S WHERE R.column1=S.column1 (+); IBM Informix supports this syntax **Error! Reference source not found.**: SELECT * FROM R, [OUTER] S WHERE R.Column1=S.Column1;

Table 21: Example Left Outer Join in Relational algebra and SQL

RA	Result← Student $\bowtie_{roll\_no=head\_student}$ Society

SQL	ANSI (ISO/IEC-2013(E)) syntax [32].
	SELECT *
	FROM student
	LEFT OUTER JOIN society
	ON
	student.roll_no=society.head_student;
	Oracle supported syntax example:
	SELECT *
	FROM society, student
	WHERE
	student.roll_no=society.head_student (+);
	IBM Informix supported syntax example:
	Select *
	From student, OUTER society
	WHERE
	student.roll_no=society.head_student;

Right outer join (⟖)

This joining query mainly looking for matching records and remaining records from right table [2] [12]. A right outer join returns all the values from the right table and matched values from the left table (NULL in the case of no matching join predicate) [13].The right outer join syntax in relational algebra. Table 22 show the join syntax with relational algebra and SQL. Table 23 shows the example query of our schema in

relational algebra expression and SQL. Table 24 shows the result of right outer join operation based on our designed schema.

Table 22: Right outer Join in Relational Algebra and SQL

RA	$Result \leftarrow R \bowtie_{R.c1=S.c2} S$
SQL	ANSI (ISO/IEC-2013(E)) syntax [32]. SELECT * FROM R RIGHT OUTER JOIN S ON R.c1=S.c1;

Table 23: Example of Right Outer Join in Relational Algebra and SQL

RA	Result← Student $\bowtie_{roll\_no=head\_student}$ Society
SQL	SELECT * FROM society, OUTER student WHERE student.roll_no=society.head_student;

Table 24: Right outer Join Operation Result

Roll_no	Student_name	cgpa	Batch_rep	Society_id	Society_name	Head_student
102	Izaan	3.0	103	1	Softec	102
102	Izaan	3.0	103	4	Support	102
Null	Null	Null	Null	3	Creative	Null
104	Ismail	2.5	105	2	ACM Lahore chapter	104

Full outer join (⋈)

This join query mainly looking for matching records and remaining records from both table [2] [13]. Conceptually, a full outer join combines the effect of applying both left and right outer joins. Where records in the full outer joined tables do not match, the result set will have null values for every column of the table that lacks a matching row. For those records that do match, a single row will be produced in the result set (containing fields populated from both tables). Table 25 show the join syntax with relational algebra and SQL. Table 26 shows the example query of our schema in relational algebra expression and SQL. Table 27 shows the result of full outer join operation on our designed schema.

Table 25: Full outer Join in Relational Algebra and SQL

RA	$Result \leftarrow R \bowtie_{R.c1=S.c2} S$
	ANSI (ISO/IEC-2013(E))syntax [32]: SELECT * FROM <left joined table> [FULL OUTER JOIN] <right joined table> ON <joined condition> Alternate syntax[15]: SELECT * FROM <left joined table>, <right joined table> WHERE <joined condition>

Table 26: Example of Full outer Join in Relational Algebra and SQL

RA	Result← Student $\bowtie_{roll\_no=head\_student}$ Society
SQL	ANSI (ISO/IEC-2013(E))syntax [32]: SELECT * FROM student FULL OUTER JOIN society ON student.roll_no=society.head_student; Alternate syntax[15]: SELECT * FROM society, student WHERE student.roll_no=society.head_student (+);

Table 27: Result of Full outer Join in Relational Algebra and SQL

Roll _no	Student _name	cgpa	Batch _rep	Society _id	Society _name	Head _student
101	Tahreem	3.3	103	Null	Null	Null
102	Izaan	3.0	103	1	Softec	102
102	Izaan	3.0	103	4	Sports	102
103	Isbah	3.4	Null	Null	Null	Null
104	Ismail	2.5	105	2	ACM Lahore Chapter	104
105	Alia	2.7	Null	Null	Null	Null
Null	Null	Null	Null	3	Creative	Null

Self-join (⋈)

The self-join is a type of inner join that join a single table to itself, especially when table has a FOREIGN KEY, which is reference to its own PRIMARY KEY. In this situation, we are using the same table twice [2].

Table 28 show the join syntax with relational algebra and SQL. Table 29 shows the example query of our schema in relational algebra expression and SQL .

Table 28: Self Join in Relational Algebra and SQL

RA	Result← $(\rho_{R1}(R)) \bowtie_{R1.c3=R2.c1} (\rho_{R2}(BR))$
SQL	ANSI (ISO/IEC-2013(E))syntax [32]: Select * From <column list>R R.column1=R.column2 Alternate syntax: Select R1.c1, R2.c2, R3.c3, R2.c2 From R As R1 JOIN R As R2 ON R1.c3= r2.c1 Note:c1 is PK and c3 is FK of relation R

Table 29: Example of Self Join Operator

RA	Result←$(\rho_S(\text{student}))$ $\bowtie_{S.batch\_rep=BR.roll\_no} (\rho_{BR}(\text{student}))$
SQL	SELECT S.roll_no, S.student_name, S.batch_rep, Br.student_name As batch_rep_name FROM student As S JOIN student As BR ON S.batch_rep=Br.roll_no

Semi-join (⋉)

The semi-join is a relational algebra operation that choose a set of rows in one relation that matches multiple rows of another relation on the joining domains [25]. A semi-join between two or more relation return rows the first relation where multiple matches are found in second relation. The difference between conventional join and semi-join is that rows in the first relation will be returned at most once. Even if the second relation contain multiple matches

For a row in the first relation, only one copy of the row will be returned. Semi-joins are implemented using the EXISTS or IN, ALL constructing in correlated sub queries in SQL [28].

Table 30: Left Semi Join in Relational Algebra and SQL

RA	Result← R ⋉S
SQL	ANSI (ISO/IEC-2013(E))syntax [32]: SELECT * FROM R WHERE EXISTS (SELECT *FROM S WHERE R.c1=S.c2);

Table 31: Example of Left Semi Join in Relational Algebra and SQL

RA	Result← Student ⋉Society.
SQL	SELECT * FROM student WHERE EXISTS (SELECT *FROM society WHERE student.roll_no=society.head_student);

Table 32: Result of Left Semi Join operator in Relational Algebra AND SQL

roll_no	student_name	Cgpa	batch_rep
102	Izan	3.0	103
104	Ismail	2.5	105

Right Semi-Join (⋊)

The right semi join logical join operator returns tuples from the right relation when there is a matching tuples in left relation. When there is no join predicate found in the argument column, every row is a matching row [23]. Table 33 show the join syntax with relational algebra and SQL. Table 34 shows the example query of our schema in relational algebra expression and SQL .Table 34 shows the result of right semi-join operation on our designed schema.

Table 33: Right semi join operator in Relational Algebra and SQL

RA	Result← R ⋊ S
SQL	ANSI (ISO/IEC-2013(E))syntax [32]: SELECT * FROM S WHERE EXISTS (SELECT * FROM R WHERE S.c1=R.c2);

Table 34: Example of Right Semi join in Relational Algebra and SQL

RA	Result← student⋊ society
SQL	SELECT * FROM society WHERE EXISTS (SELECT * FROM student WHERE batch_rep=roll_no);

Table 35: Result of Right Semi Join Operator

society_id	society_name	head_student
1	Softec	102
2	Acm Lahore chapter	104
4	Sports	102

Anti-join (▷)

The anti-join return the non-matching rows from two relations. An anti-join between two relation returns tuples from the first relation where no matches are found in second relation [28]. Basically the anti-join is opposite of a semi-join, an anti-join return one copy of each rows in the first relation which no match is found while a semi-join return one copy of each rows in the first relation which at least one match is found. Anti-join is implemented using NOT IN or NOT EXISTS constructs in correlated sub queries [28]. Table 36 show the join syntax with relational algebra and SQL. Table 37 shows the example query of our schema in relational algebra expression and SQL .Table 38 shows the result of anti-join operation on our designed schema.

Table 36: Anti Join operator in Relational algebra and SQL

RA	Result← R ▷ S
SQL	ANSI (ISO/IEC-2013(E))syntax [32]: SELECT * FROM R WHERE NOT EXISTS (SELECT *FROM S WHERE R.c1=S.c2);

Table 37: Example of Anti Join operator in Relational Algebra and SQL

f RA	Result← student ▷ Society
SQL	SELECT * FROM student WHERE NOT EXISTS (SELECT *FROM society WHERE student.roll_no=society.head_student);

Table 38: Result of Anti Join operator

Roll_No	Student_Name	Batch_Rep
101	Tahreem	103
103	Isbah	Null
105	Alia	Null

Table 39: Join Operator support to DBMS

Join operators	Relational algebra Support(direct)	ISO/IEC-2013(E) Support	Oracle	Microsoft SQL	MySQL	TeraData	PostgreSQL	IBM DB2	IBM Informix
Cross join	✓	✓	✓	✓	✓	✓	✓	✓	✓
Inner join	✓	✓	✓	✓	✓	✓	✓	✓	✓
Left outer	✓	✓	✓	✓	✓	✓	✓	✓	✓
Right outer	✓	✓	✓	✓	✓	✓	✓	✓	✓
Full outer	✓	✓	✓	✓	✓	✓	✓	✓	✓
Theta join	✓	×	✓	✓	✓	✓	✓	✓	✓
Equi-join	✓	×	✓	✓	✓	✓	✓	✓	✓
Named column join	✓	✓	✓	✓	✓	✓	✓	✓	✓
Natural join	✓	✓	✓	✓	✓	✓	✓	✓	✓
Self-join	✓	✓	✓	✓	✓	✓	✓	✓	✓
Left semi-join	×	×	✓	✓	✓	✓	✓	✓	✓
Right semi-join	×	×	✓	✓	✓	✓	✓	✓	✓
Anti-join	×	×	✓	✓	✓	✓	✓	✓	✓

The Table 39 show which logical join operator support with relational algebra and ISO/IEC-2013 standard with top DBMS.

IV. CONCLUSION AND FUTURE WORK

In this paper, we have worked to minimize the complexities that were faced by the learners of Database System in CS1 module. The complexity of joins is being reduced with the help of details examples according to the taxonomy. This taxonomy of the joins being presented will help Readers, students and researchers in identifying join operators, which are being discussed in a simple way.

ACKNOWLEDGMENT

I am very thankful to department of computer science at University of Management and Technology for providing me peaceful and ambient environment. I am also very thankful to all the reviewers who took time out of their busy schedule for reviewing this article.

REFERENCES

[1]Mishra, P., & Eich, M. H. (1992). Join processing in relational databases. ACM Computing Surveys (CSUR), 24(1), 63-113.

[2] Oracle® Database SQL Reference 10g Release 1 (10.1), Documentation. https://docs.oracle.com/cd/B12037_01/server.101/b10759/toc.htm .Accessed-05 November 2015.

[3]C.Bhanuprakash, Y.S.Nijagunarya, M.A.Jayaram,"A Simple Approach to SQL Joins in a Relational Algebraic Notation" International Journal of Computer Applications (0975 – 8887) Volume 104 – No.4, October 2014

[4]Understanding joins. Microsoft TechNet SQL Server. https://technet.microsoft.com/en-us/library/ms190014%28v=sql.105%29.aspx . Accessed 05 November, 2015.

[5]D. DeWitt, D. J., Katz, R. H., Olken, F., Shapiro, L. D., Stonebraker, M. R., & Wood, D. A. (1984). Implementation techniques for main memory database systems (Vol. 14, No. 2, pp. 1-8). ACM.

[6]Kitsuregawa, M., Tanaka, H., & Moto-Oka, T. (1983). Application of hash to data base machine and its architecture. New Generation Computing, 1(1), 63-74.

[7]Mane gold, S., Boncz, P. A., & Kersten, M. L. (2000, September). What happens during a join? Dissecting CPU and memory optimization effects. In Proceedings of the 26th international conference on very large data bases (pp. 339-350). Morgan Kaufmann Publishers Inc.

[8]Syrdal, A. K., & Conkie, A. (2005, September). Perceptually-based data-driven join costs: comparing join types. In INTERSPEECH (Vol. 5, pp. 2813-2816).

[9] Yang, Y., & Singhal, M. (1997). A comprehensive survey of join techniques in relational databases. Computer and Information Science TR, 48.

[10]Swami, A. (1989, June). Optimization of large join queries: combining heuristics and combinatorial techniques. In ACM SIGMOD Record (Vol. 18, No. 2, pp. 367-376). ACM.

[11]Starner, J. W. (2007). Joins on interval data type columns in relational databases. Journal of Computing Sciences in Colleges, 22(4), 235-241.

[12]Pratt, Phillip J (2005), A Guide to SQL, Seventh Edition, Thomson Course Technology, ISBN 978-0-619-21674-0

[13]Ramez Elmasri, Shamkant B. Navathe, Fundamentals of Database Systems, Edition 6, Addison Wesley Pub Co Inc, 2010, ISBN 0136086209, 9780136086208, Page 145 – 164

[14]Ramez Elmasri, Shamkant B. Navathe, Fundamentals of Database Systems, Edition 5, Addison Wesley Pub Co Inc, 2010, ISBN 0136086209, 9780136086208, Page 183 – 184

[15]http://www.oracle-dba-online.com. Accessed 17 December 2015

[16]Gotlieb, L. R. (1975, May). Computing joins of relations. In Proceedings of the 1975 ACM SIGMOD international conference on Management of data (pp. 55-63). ACM.

[17]Cao, Y., Zhou, Y., Chan, C. Y., & Tan, K. L. (2012, March). On optimizing relational self-joins. In Proceedings of the 15th International Conference on Extending Database Technology (pp. 120-131). ACM.

[18] Codd, E.F. (1970). "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM **13** (6): 377–387. doi:10.1145/362384.362685.

[19]http://www.oratechinfo.co.uk.Accessed-25 Jan, 2016.

[20]http://infocenter.sybase.com. Accessed-25 Jan, 2016.

[21]http://dev.mysql.com. Accessed-25 Jan, 2016.

[22]http://www.sqlguides.com. Accessed-25 Jan, 2016.

[23]https://technet.microsoft.com. Accessed-27 Jan, 2016.

[24]http://allthingsoracle.com. Accessed-1 Feb, 2016.

[25]Bernstein, P. A., & Chiu, D. M. W. (1981). Using semi-joins to solve relational queries. Journal of the ACM (JACM), 28(1), 25-40.

[26]Codd, E. F. 1979. Extending the relational database model to capture more meaning. ACM Transactions on Database Systems 4, 4 (Dec.), 397{434.

[27]Xu, Y., & Kostamaa, P. (2009). Efficient outer join data skew handling in parallel dbms. Proceedings of the VLDB Endowment, 2(2), 1390-1396.

[28]Understanding Queries with Semi-Joins and Anti-Joins.http://www.dbspecialists.com/files/presentations/semijoins.html. Accessed-04 FEB, 2016.

[29]http://www.gplivna.eu .Accessed- 11 February 2016.

[30]Morishita, S. (1997). Avoiding Cartesian products for multiple joins. Journal of the ACM (JACM), 44(1), 57-85.

[31]Wang, T. J., & Liu, P. (2011). A Taxonomy of the Join Operations in The REA Data Model. Review of Business Information Systems (RBIS), 8(1), 9-22.

[32]ISO/IEC CD 9075-2:2013(E)**"** Information technology - Database languages - SQL - Part 2: Foundation(SQL/Foundation) Ed 5"

[33]http://www.studytonight.com/. Accessed-04 march, 2016.

[34] ISO/IEC CD 9075-2:2013(E)" Information technology - Database languages - SQL - Part 2: Foundation(SQL/Foundation) Ed 5"

[35] http://allthingsoracle.com. Accessed-1 Feb, 2016.

[36] http://www.oratechinfo.co.uk.Accessed-25 Jan, 2016

[37] Boneset, D. (2013). On the impact of hardware on relational join processing (Doctoral dissertation, Master's thesis, University of Magdeburg).