Identifying Symmetric and Transitive Binary Relations

Khalid Al-Barrak, Mohammed Alawairdhi, Kailash Kumar

Abstract: ***In real world, the binary relations play a vital role in capturing relation between pair of objects. This paper primarily focuses on identifying the candidate symmetric and candidate transitive binary relations in the relational database. In this paper, an algorithm is devised to identify symmetric and transitive binary relations for a specific pattern.***

Keywords: ***Binary Relations, Symmetric Relations, transitive Relations, Heuristic Data Modeling, Data Analysis.***

I. INTRODUCTION

A binary relation is used to identify similar distinguished relationships between pairs of objects. A pair (or tuple) in such relation states that one object is related with another in a unidirectional manner. Depending on the semantics of the relationship, a binary relation can have properties that allow deducing further relations between objects without having to explicitly state them. Some of the common properties include transitivity and symmetry [1][2]. These properties are imperative because they can have implications on the manner we interpret and process tuples in these relations.

In information systems, these properties are often handled through user, application logic, or database layer. While, the database layer is the most reasonable place to handle these properties but current database standards do not provide straight and well-designed methods to address them. Additionally, database modeling representations, such as Entity-Relationship (ER), do not provide the constructs to model binary relations that are characterized as transitive or symmetric relations [3][4][5]. Failing to address these properties properly can result in storing duplicate information, which in turn, can lead to data inconsistency. Ontology languages such as OWL provide the constructs for interpreting binary relations as symmetric or transitive [6].

A binary relation *r* on domain *D* ($r \subseteq D \times D$) is symmetric if:

$\forall a, b \in D, (a, b) \in r \Rightarrow (b, a) \in r.$

A binary relation *r* on domain *D* ($r \subseteq D \times D$) is transitive if:

$\forall a, b, c \in D, ((a, b) \in r) \wedge ((b, c) \in r) \Rightarrow (a, c) \in r.$

There are plenty of examples in real world where binary relations can be symmetric and/or transitive. Some of them are shown below in Table 1.

Binary Relation	*Type of the Binary Relation*	*Cardinality of the Binary Relation*
Married-to (Person)	Symmetric & Non-Transitive	One-to-One (1:1)
Next (Queued items)	Transitive & Non-Symmetric	One-to-One (1:1)
Manages (Staff)	Transitive & Non-Symmetric	One-to-Many (1:M)
Knows (Person), Follows (Twitter)	Non-Symmetric & Non-Transitive	Many-to-Many (N:M)
Borders (Territory)	Symmetric & Non-Transitive	Many-to-Many (N:M)
Composed-Of (Product), Dependency (Tasks)	Non-Symmetric & Transitive	Many-to-Many (N:M)
Siblings (Person), Live-with (Person)	Symmetric & Transitive	Many-to-Many (N:M)

II. MOTIVATION

Identifying candidate symmetric and/or transitive binary relations in the relational database can help in inferring facts that are closely stated in the knowledge-base. These inferred facts can be vital to solving business problems or identifying business opportunities.

III. ASSUMPTIONS

Data consistency and storage cost are affected by storing redundant information such as that implied by symmetric or transitive binary relations; e.g. storing (*b*,*a*) given (*a*,*b*) in a symmetric binary relation or (*a*,*c*) given (*a*,*b*) and (*b*,*c*) in a transitive binary relation. Storing such data can lead to data inconsistency, which is caused by deleting one tuple and not the other. To overcome this problem, database modelers and developers rely on both modeling and development techniques to allow users to retrieve tuples even when they are not explicitly stored in the database. These techniques range from using views and stored procedures in the database tier to developing business logic in the application tier. However, since analyzing programming logic to discover symmetry and transitivity is infeasible due to the wide spectrum of programming languages and paradigms in use nowadays, this reseach paper has focused on retrieving and analyzing data in databases, which can be accomplished using a standardized language (i.e. SQL).

Identifying *candidate symmetric* and *candidate transitive* binary relations expect the source database to conform to the following two conventions:

Revised Manuscript Received on March 02, 2019.
Dr. Khalid Al-Barrak, George Mason University, United States
Dr. Mohammed Alawairdhi, Saudi Electronic University, KSA
Dr. Kailash Kumar, Saudi Electronic University, KSA

1. The database has been created using common design patterns for symmetric and/or transitive binary relations [7][8][9], and
2. The database does not store tuples that are implied by symmetry or transitivity.

Following these conventions not only help in reducing storage cost, but also avoid data inconsistency.

Sometimes, database designers purposely opt for data redundancy in order to achieve better performance. The rules given in this paper does not support such cases for identifying candidate symmetric and candidate transitive relations.

IV. METHODOLOGY

This section identifies *candidate symmetric* and *candidate transitive* binary relations using heuristic data modeling and data analysis.

Definition 1: Given a binary relation *r* on domain *D* and property *P*, we say that *r* is *minimal* w.r.t. *P* if the following holds:

$(\neg \exists t \in r)(t \in (r - \{t\})_P^+)$, where $(r - \{t\})_P^+$ is the *P*-closure of *r* without tuple *t*.

This definition states that the binary relation r is said to be minimal if we can not find a tuple t in relation r when the P-closure (e.g., symmetric or transitive closure) of r without t will yield t. In other words, a relation r is considered minimal w.r.t property P if r does not include any tuple that is implied by P (e.g. symmetry and transitivity).

Definition 2: Pattern 1 (P1): Given a relation schema R2, a relation instance r2 over R2, and integrity constraints pk2 as the primary key of R2 (i.e. pk2 = pkey(R2)) and fk2 as a foreign key in R2 with reference to pk2 (i.e. fk2 $\in$ fkey(R2) and refpk(fk2) = pk2). Let Dpk2 be the projection of pk2 values (i.e. Dpk2 = π pk2(r2)), r1 be a binary relation on Dpk2 (i.e. r1Dpk2 $\times$ Dpk2), and R1 be the schema of r1 (i.e. R1={(A1a:Dpk2), (A1b:Dpk2)}). We say r1 conforms to Pattern 1 (P1 in short) if the following holds:

$-$r1 = π pk2, fk2 (σis_not_null (fk2) (r2)).

We note Pattern 1 as a structure P1 = (R2, r2, R1, r1, IC1), where IC1= (pk2, fk2) is a tuple with integrity constraints relevant to P1. Examples of P1 binary relations include Married-to (Person), Next (Queued Items), and Manage (Staff). This pattern is used with 1:1 or M:1 binary relations. Appendix B contains samples of these relations (both schemas and instances).

Definition 3: Pattern 2 (P2): Given relation schemas R1 and R2, relation instances r1 over R1 and r2 over R2, and integrity constraints pk1 as the primary key of R1 (i.e. pk1 = pkey(R1)), pk2 as the primary key of R2 (i.e. pk2 = pkey(R2)), and fk1 and fk2 as foreign keys in R1 with reference to pk2 (i.e. fk1, fk2 $\in$ fkey(R1) and refpk(fk1) = refpk(fk2) = pk2).

Let Dpk2 be the projection of pk2 values (i.e. Dpk2 = π pk2(r2)), and A1a and A1b be sets of attributes corresponding to fk1 and fk2 respectively (i.e. A1a = fk1, A1b = fk2). We say r1 is a binary relation (on Dpk2) that conforms to Pattern 2 (P2 in short) if the following holds:

i) ({fk1 , fk2} = fkey(R1)) $\wedge$ (fk1 $\cup$ fk2 = attrib(R1)), and
ii) fk1 $\cup$ fk2 = pk1.

We note Pattern 2 as a structure: P2 = (R2, r2, R1, r1, IC2), where IC2= (pk1, pk2, fk1, fk2) is a tuple with integrity constraints relevant to P2. Examples of P2 binary relations include Follows (Twitter), Boarders (Territory), Composed-of (Products), and Siblings (Person). This pattern is used mostly with N:M binary relations but can also be used as an alternative to P1 when property 'ii' in Definition 4.5 is adjusted. Appendix B contains samples of such relations.

Definition 4: Pattern 3 (P3): Given relation schemas R2 and R3, relation instances r2 over R2 and r3 over R3, and integrity constraints pk2 as the primary key of R2 (i.e. pk2 = pkey(R2)), pk3 as the primary key of R3 (i.e. pk3 = pkey(R3)), and fk2 as a foreign key in R2 with reference to pk3 (i.e. fk2 $\in$ fkey(R2) and refpk(fk2) = pk3). Let Dp k 2 and Dp k 3 be the projections of pk2 and pk3 values respectively, r1 be a binary relation from Dp k 2 to Dp k 3 (i.e. r1 $\subseteq$ Dp k 2 $\times$ Dp k 3), and R1 be the schema of r1 (i.e. R1= {(A1a:Dpk2), (A1b:Dpk3)}). We say r1 conforms to Pattern 3 (P3 in short) if the following holds:

i) fk2 $\cap$ pk2 = $\emptyset$,
ii) attrib(R3) – *pk3* = $\emptyset$, and
iii) r1 = π pk2, fk2 (σis_not_null (fk2) (r2)).

We note Pattern 3 as a structure: P3 = (R3, r3, R2, r2, R1, r1, IC3), where IC3= (pk2, pk3, fk2) is a tuple with the integrity constraints relevant to P3. Examples of P3 binary relations include Siblings and Live-with (Person). Note that this pattern is an alternative to P2 for N:M binary relations that are both symmetric and transitive. Moreover, it is worth noting here that Pattern 3 can also be used for a category-like relation (e.g. when *R3* is indexed by a category-name and has no other attributes). While it is uncommon to have a category-like relation without a category-id as its index in addition to a category-name attribute, we acknowledge that such relation when encountered will be identified wrongly as pattern 3 (i.e. a false-positive). Appendix 1 contains a sample of a valid Pattern 3 relation.

This paper has identified the specific patterns described above commonly used to implement real-world symmetric/transitive binary relations in relational database. Once the pattern is detected, it performs data analysis to classify the binary relation as *candidate symmetric* and/or *candidate transitive*.

Methods to Identify Candidate Symmetric and Candidate Transitive Binary Relations

Several structural patterns exist for modeling symmetric and/or transitive binary relations. The use of one or another depends on the cardinality and the design choices made by the database designer. In this paper, DM2ONT has identified three structural patterns that are commonly used to implement real-world symmetric/transitive binary relations in RDB. These patterns were termed Pattern 1, 2 and 3 (or P1, P2 and P3 for short). Once DM2ONT detects these structural/schema patterns, it performs data analysis (if necessary) to classify the binary relations associated with these patterns as *candidate symmetric* and/or *candidate transitive*.

Published By:
Blue Eyes Intelligence Engineering & Sciences Publication

Since data in P3 binary relations do not exhibit any special characteristics, the schemas associated with P3 binary relations are declared as candidate symmetric solely based on the schema definition. For P1 and P2 however, further analysis is required in order to declare the specific pattern as symmetric or transitive binary relation. Figure 1 depicts the *overall* process for determining *candidate symmetry* and *candidate transitivity* for all three patterns.

The following two sections present the algorithms used in DM2ONT for determining if a P1 or P2 binary relation is *candidate symmetric* or *candidate transitive*.

Identifying Candidate Symmetry and Candidate Transitivity for Pattern 1

This section presents Algorithm A1, which addresses candidate symmetry and candidate transitivity for binary relations that conform to Pattern 1 (definitions 4.2).

Candidate Symmetric: We say a binary relation *r1* is *Candidate Symmetric* w.r.t. pattern *Py* if Algorithm Ay(Py, card(r1)) returns isCandSymm=True, where $y \in \{1, 2\}$ and Py is a structure conforming to Pattern *y*.

Candidate Transitive: We say a binary relation *r1* is *Candidate Transitive* w.r.t. pattern *Py* if Algorithm Ay(Py, card(r1)) returns isCandTrans=True, where $y \in \{1, 2\}$ and Py is a structure conforming to Pattern *y*.

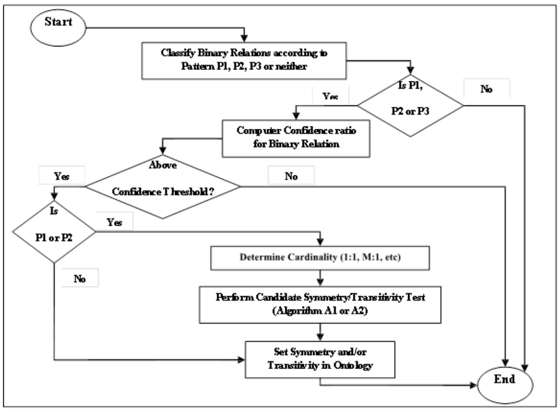

Fig 1: Identifying candidate symmetric and transitive binary relations - Overall process

Algorithm A1 (Pattern 1 – Candidate Symmetric/ Transitive):

Input: P1 (R2, r2, R1, r1, IC), card (r1)
Output: isCandSymm (Boolean), isCandTrans (Boolean)
Begin-Steps
//R1 (from structure P1) has two sets of attributes: A1a and A1b
Let isCandSymm = isCandTrans = false
If (card = = '1:1') Then
Let result_set1 = πA_{1a} (r1) $\cap \pi A_{1b}$ (r1)
If (result_set1 = = $\varnothing$) Then
isCandSymm = true
End_If
End_If
If (isCandSymm = = false) Then
Let isAcyclic = isTrivial = true
Let A1a_set = πA_{1a} (r1)
Let ei = GetElement(1) (A1a_set)
While (ei $\neq$ null) and (isAcyclic) Do
Let tc_set = πA_{1b} (σA_{1a} = e i (r1))
Let ej = GetElement(1) (tc_set)
While (ej $\neq$ null) and (isAcyclic) Do
Let c = GetFirstElement (2) (π A (σA = e j (r1)))
If ((c $\neq$ null) and ((c $\in$ tc_set) OR (c = = ei))) Then
isAcyclic = false
Else If (c $\neq$ null) Then
tc_set = tc_set $\cup$ {c}
isTrivial = false
End_If_Else
ej = GetElement(1) (tc_set)
End_While (ej $\neq$ null…)
A1a_set = A1a_set – tc_set
ei = GetElement(1) (A1a_set)
End_While (ei $\neq$ null…)
If (isAcyclic) and (isTrivial = = false) Then
isCandTrans = true
End_If
End_If // (isCandSymm = = false)
End-Steps

----

(1) GetElement (): A function that takes a set as an input and returns an element that has not been processed or null otherwise. It marks returned element as processed.

(2) GetFirstElement (): A function that returns the first element in the given set or null if the set is empty.

Identifying Candidate Symmetry and Candidate Transitivity for Pattern 2

This section presents Algorithm A2, which addresses candidate symmetry and candidate transitivity for binary relations that conform to Pattern 2 (***Definitions 3***).

Algorithm A2 (Pattern 2 – Candidate Symmetric & Transitive):

Input: P2 (R2, r2, R1, r1, IC2), card (r1)
Output: isCandSymm (Boolean), isCandTrans (Boolean)
Begin-Steps
//R1 (from Pattern 2) has two sets of attributes: A1a and A1b
Let isCandSymm = isCandTrans = false
Let result_set = r1 $\bowtie$ ((r1.A1a = tx.A1b) and (r1.A1b = tx.A1a)) ρ_{tx} (r1)
If (result_set = = $\varnothing$) Then
isCandSymm = true
Let isTransMin = isAcyclic = isTrivial = true
isAcyclic = CheckAcyclic(1) (r1)
If (isAcyclic) Then
Let A1a_set = πA_{1a} (r1)
Let ei = GetElement(2) (A1a_set)
While (ei $\neq$ null) and (isTransMin) Do
Let processed_tuples = σA_{1a} = ei (r1)
Let ei_direct = tc_set = πA_{1b} (processed_tuples)

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

```
        While ( (isTransMin) and ( (∃ (b,c)
        ∈ r1) (b ∈ tc_set ∩ A1a_set) and
          ((b, c) ∉ processed_tuples) ) Do
      processed_tuples = processed_tuples
      ∪ {(b, c)}
      If (c ∈ ei_direct) Then
        isTransMin = false
      Else
        tc_set = tc_set ∪ {c}
        isTrivial = false
      End_If_Else
        End_While //((isTransMin) and …)
        ei = GetElement(2) (A1a_set)
     End_While (ei ≠ null and …)
     If (isTransMin) and (isTrivial = =
     false) Then
        isCandTrans = true
     End_If
   End_If //(isAcyclic)
 End_If //( result_set = = ∅ )
        End-Steps
```

(1) CheckAcyclic (): A function that returns true if the given binary relation is acyclic and false otherwise.

(2) GetElement (): A function that takes a set as an input and returns an element that has not been processed or null otherwise. It marks returned element as processed.

V. RESULTS

The algorithms are implemented on sample database provided by IBM DB2. It was implemented using Java and JDBC. The portability was achieved by Java, which elimitates the need to rebuild the code when running it in different platforms. JDBC interface was used instead of API clients provided by DBMS vendors. The results are shown at the end of the paper in Appendix 1.

VI. SUMMARY

Binary relations exist in various real-life scenarios. Some of these relations exhibit characteristics such as symmetry and/or transitivity. With DBMS(s) lacking the explicit support for symmetric and transitive binary relations, database designers typically rely on data modeling patterns to capture them. Using these patterns, such databases can avoid common modeling pitfalls associated with data inconsistency and storage overage.

Since, ontology languages (e.g. OWL) provide the grammar to annotate binary relations as symmetric and/or transitive, and given the business value for semi-automating the generation of explicit ontology models, we investigated in this research methods to identify binary relations in relational databases that are likely to be symmetric or transitive. Identifying such relations required detecting certain structural patterns, and in some cases analyzing data instances. Similar to other data analysis methods in DM2ONT, the identification methods here take into account the number of data instances supporting the finding and process only those that pass a confidence threshold.

REFERENCES:

1. Dey, Debabrata et al. 1999. Improving database design through the analysis of relationships. ACM Transactions on Database Systems (TODS), Volume 24 Issue 4, Dec. 1999.
2. Simsion, Graeme C. & Witt, Graham C. 2005. Data Modeling Essentials, 3rd edition. Morgan Kaufmann Publishers; Books24x7 version.
3. Date, Chris J. 1995. Introduction to Database Systems, 6th edition. Addison-Wesley
4. Ramakrishnan, Raghu and Gehrke, Johannes. 2003. Database Management Systems, 3ed edition. McGraw-Hill.
5. Ross, Kenneth A and Stoyanovich, Julia. 2004. Symmetric Relations and Cardinality-Bounded Multisets in Database Systems. The 30th VLDB Conference, Toronto, Canada, 2004.
6. Blaha, Michael. 2010. Patterns of Data Modeling. CRC Press.
7. Bohring, Hannes and Auer, Sören 2005. Mapping XML to OWL Ontologies. Leipziger Informatik-Tage, volume 72 of LNI (2005), pp. 147-156
8. Date, Chris J. 2003. On Various Types of Relations. http://www.dbdebunk.com/page/page/622108.htm (accessed on Dec 5th, 2011)
9. Dey, Debabrata et al. 1999. Improving database design through the analysis of relationships. ACM Transactions on Database Systems (TODS), Volume 24 Issue 4, Dec. 1999.
10. Albarrak, Khalid M. and Sibley, Edgar H. 2009. Translating Relational & Object- Relational Database Models into OWL Models. Proceedings of the IEEE International Conference on Information Reuse and Integration, IRI 2009, 10-12 August 2009, Las Vegas, Nevada, USA. pp. 336-341
11. Albarrak, Khalid M. and Sibley, Edgar H. 2010. An Extensible Framework for Generating Ontology Models from Data Models, International Transactions on System Science and Applications (ITSSA), Vol. 6, No. 2/3, August 2010,. pp. 97-112.
12. Albarrak, Khalid M. and Sibley, Edgar H. 2011. A survey of methods that transform data models into Ontology models. Proceedings of the IEEE International Conference on Information Reuse and Integration, IRI 2011, 3-5 August 2011, Las Vegas, Nevada, USA. pp. 58-65
13. Albarrak, Khalid M. and Sibley, Edgar H. 2012. Methodology to Measure Expressivity between Ontology Models. To appear in <TBD> 2012.
14. Albarrak, Khalid M., Mohammed Alawairdhi, Kumar Kailash. Identifying Candidate Symmetric and Candidate Transitive Binary Relations. International Journal of Emerging Technology and Advanced Engineering ISSN 2250-2459, Volume 8, Special Issue 4, April 2018.

AUTHOR’S BIOGRAPHY

Dr. Khalid Al-Barrak an Information Technology leader with over 18 years of professional experience in various IT lead roles across the software supply-chain. These lead roles range from software engineering, quality assurance, technical enablement, solution architecture, software sales and services, to IT and Analytics consultancy and management. Al-Barrak is the VP of Corporate Analytics and Data, Saudi Telecom Company. Previously, Channels

Technical Manager, North America, Southeast and Federal, IBM. He holds a B.Sc. in computer information systems from king Saud University, MSc. degree in computer science from California State University, Chico, and a PhD in Information Technology from George Mason University.
Al-Barrak has many publications in international journals and conferences in database, ontology and data modelling.

Dr. Mohammed Alawairdhi an associate professor in the Computer Science department and the Vice rector for Graduate Studies and Scientific Research, Saudi Electronic University. He participated in establishing the Saudi Electronic University in 2012. Dr. Alawairdhi is the head of the University Scientific Council and a member of the higher consultancy committee. Previously, he was the dean of Computing and Informatics College. Prior to joining SEU, he was vice dean of graduate studies and research, College of Computer and Information Sciences, Al-Imam Muhammad bin Saud Islamic University. He holds a B.Sc. in computer information systems from king Saud University, MSc. degree in computer science from California State University, Chico, and a PhD in computer science from De Montfort University. Dr. Alawairdhi has many publications in international journals and conferences in software engineering, blended learning and ubiquitous computing.

Dr. Kailash Kumar presently serving as an assistant professor in the College of Computing and Informatics, Saudi Electronic University, Riyadh, KSA. Previously, he was Dean (Academics) and Assistant Professor in the department of Computer Science & Engineering since July 2010 in Modern Institute of Technology & Research Centre, Alwar, Rajasthan. He is associated with various technical societies of national and international repute. He has also published various research papers at national and international level. He is a Life Member of Computer Society of India (CSI) and Indian Society for Technical Education (ISTE). He is also member of International Association of Computer Science & Information Technology (IACSIT) and International Association of Engineers (IAENG). He is profoundly engrossed in the area of Data Structure & Algorithms, Compiler Construction, DBMS, C and C++.

APPENDIX 1: SYMMETRY/TRANSITIVITY BINARY RELATIONS

This section contains various relations, both relation schemas and relation instances for the different patterns that were introduced in this paper.

Pattern 1:

This pattern is used with one-to-one binary relations that are Symmetric or Transitive, and with one-to-many binary relations that are Transitive:

One to One Symmetric:

Table Schema: Person = (id (PK), name, gender, spouse-id (FK ref Person (id)))

Table Instance:

Id	*Name*	*Gender*	*Spouse-id*
1	John	M	2
2	Jane	F	
3	Riyadh	M	
4	Faisal	M	
5	Tami	F	6
6	Tom	M	

One to One Transitive:

Table Schema: Next-in-queue = (id (PK), name, next-id (FK ref Next-in-queue (id)))

Table Instance:

Id	*Name*	*Next-id*
1	Item 1	2
2	Item 2	3
3	Item 3	

One to Many Transitive:

Table Schema: Employee = (id (PK), name, mgr-id (FK ref Employee (id)))

Table Instance:

Id	*Name*	*Mgr-id*
1	Edgar	
2	Khalid	1
3	Jane	1
4	Faisal K.	2
5	Riyadh K.	2

Pattern 2:

This pattern is mostly used with many-to-many relations that are Symmetric, Transitive, both Symmetric and Transitive, or neither:

Many to Many – Non-Symmetric and Non-Transitive:

Table Schema: Person = (id (PK), name, gender)

Knows = (id1 (FK ref Person (id)), id2 (FK ref Person (id)), PK(id1, id2))

Table Instance:

Person

Id	*Name*
1	Person 1
2	Person 2
3	Person 3
4	Person 4

Knows

Id1	*Id2*
1	2
1	3
2	1
3	4
4	1

Many-to-many - Symmetric and Non-Transitive:

Table Schema: Country = (id (PK), name)

Border-with = (id1 (FK ref Country (id)), id2 (FK ref Country (id)) , PK(id1, id2))

Country

Id	*Name*
1	Jordan
2	Saudi

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

3	Iraq
4	Kuwait

Border-with

Id1	*Id2*
1	2
1	3
2	3
3	4
4	2

Many-to-many – Non-Symmetric and Transitive:

Table Schema: Table Schema: Product = (id (PK), name)

Composed-of = (id1 (FK ref Product (id)), id2 (FK ref Product (id)), PK(id1, id2))

Product

Id	*Name*
	Product 1
2	Product 2
A	Product 3
B	Product 4
C	Product 5
D	Product 6
E	Product 7

Composed-of

Id1	*Id2*
1	A
1	B
2	A
2	C
A	D
B	D
B	E

Many-to-many Symmetric and Transitive:

Table Schema: Person = (id (PK), name)

Sibling = (id1 (FK ref Person (id)), id2 (FK ref Person (id)) , PK(id1, id2))

Person

Id	*Name*
1	Khalid
2	Sami
3	Faris
4	Riyadh
5	Faisal

Sibling

Id1	*Id2*
1	2
2	3
4	5

Pattern 3:

This pattern is used with many-to-many binary relations that are both Symmetric and Transitive. This pattern is considered an alternative to Pattern 2 for binary relations that are *both* Symmetric and Transitive.

Many-to-many Symmetric and Transitive:

Table Schema: Person = (id (PK), name, sibling-set-id (FK ref Sibling-set (set-id)))

Sibling-set = (set-id (PK))

Table Instance:

Person

Id	*Name*	*Sibling-set-id*
1	Khalid	S1
2	Sami	S1
3	Faris	S1
4	Riyadh	S2
5	Faisal	S2

Sibling-set

Set-id
S1
S2

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication