Performance Evaluation of Database Design Approaches for Object Relational Data Management

by

Md. Abdul Lahel Shafey

A thesis submitted to the Institute of Information and Communication Technology (IICT) in the partial fulfillment of the requirements for the degree of MASTER OF SCIENCE IN ENGINEERING IN INFORMATION AND COMMUNICATION TECHNOLOGY

Institute of Information and Communication Technology
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY, DHAKA
August 2008
The thesis “Performance Evaluation of Database Design Approaches for Object Relational Data Management”, submitted by Md. Abdul Lahel Shafey, Roll No: MP0231011, Session: 2002-2003 to the Institute of Information and Communication Technology, Bangladesh University of Engineering and Technology, Dhaka, has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Master of Science in Engineering (Information and Communication Technology) and approved as to its style and contents for the examination held on the 14th August, 2008.

**Board of Examiners**

1. Dr. Abu Sayed Md. Latiful Hoque  
   Associate Professor,  
   Department of Computer Science & Engineering  
   BUET, Dhaka-1000  
   Chairman (Supervisor)

2. Dr. S. M. Lutful Kabir  
   Professor and Director,  
   Institute of Information and Communication Technology,  
   BUET, Dhaka-1000  
   Member (Ex-officio)

3. Md. Saiful Islam  
   Assistant Professor,  
   Institute of Information and Communication Technology,  
   BUET, Dhaka-1000  
   Member

4. Dr. Mohammad Zahidur Rahman  
   Associate Professor,  
   Department of Computer Science and Engineering,  
   Jahangirnagar University, Dhaka  
   Member (External)
Candidate’s Declaration

It is hereby declared that the work presented in this thesis or any part of this thesis has not been submitted elsewhere for the award of any degree or diploma, does not contain any unlawful statements and does not infringe any existing copyright.

Signature

(Md. Abdul Lahel Shafey)
Contents

Chapter 1 Introduction............................................................................................................. 1
  1.1 Information System Development.................................................................................. 2
  1.2 Overview of the Database Design Approaches.............................................................. 3
  1.3 Objective ......................................................................................................................... 4
  1.4 Organization of the thesis............................................................................................... 4

Chapter 2 Literature Review .................................................................................................. 5
  2.1 Data Model....................................................................................................................... 5
     2.1.1 Relational Model........................................................................................................ 5
     2.1.2 Object-oriented model................................................................................................ 7
     2.1.3 Object relational model............................................................................................. 8
     2.1.4 Semi-Structure data model....................................................................................... 9
     2.1.5 Associative data model.............................................................................................. 10
     2.1.6 Entity-Attribute-Value model.................................................................................. 10
     2.1.7 Hierarchical data model.......................................................................................... 10
     2.1.8 Network data model................................................................................................. 11
  2.2 Database design for Relational Model............................................................................ 11
     2.2.1 The Entity-Relationship Model............................................................................... 11
     2.2.2 Normalization........................................................................................................... 12
     2.2.3 Problems with Normalization................................................................................ 13
     2.2.4 De-normalization as per required.......................................................................... 14
     2.2.5 Spatial data modeling using relational database design approaches.................. 16
     2.2.6 Relational model for spatial data............................................................................ 17
  2.3 EAV database: conversion of Relational/Universal model to EAV................................. 18
  2.4 Binary Model: conversion of Relational/Universal model to Binary model.................... 19
  2.5 Object Oriented Model.................................................................................................... 19
     2.5.1 Object Structure........................................................................................................ 20
     2.5.2 Object Class............................................................................................................. 20
     2.5.3 Inheritance............................................................................................................... 20
     2.5.4 Object Identity......................................................................................................... 21
2.5.5 Spatial data modeling using Object-oriented approaches .................................................. 22
2.6 Object Relational Model ........................................................................................................ 23
2.7 Spatial Data Modeling ........................................................................................................... 29
  2.7.1 Spatial Data types (SDT) ................................................................................................. 32
  2.7.2 Spatial data modeling requirements ................................................................................ 33
  2.7.3 An extended entity-relational model for Geographic applications ...................... 34
  2.7.4 Spatial pictogram enhanced conceptual data models .............................................. 35
  2.7.5 Geographic Information System Entity Relationship Model (GISER) ............. 37
  2.7.6 Geometry Object Model (OpenGIS® Simple Features Specification for SQL) .... 38

Chapter 3 Design Approaches for Relational up to Object Relational Database........... 42
  3.1 Consideration of database design approaches ................................................................. 42
  3.2 Design Approach 1: ER-Relational-Normalized-Denormalized approach .............. 44
    3.2.1 Step 01: Selection of entity ..................................................................................... 44
    3.2.2 Step 02: Selection of attributes .............................................................................. 45
    3.2.3 Step 03: Relationship identification ...................................................................... 47
    3.2.4 Step 04: Design ERD ............................................................................................ 47
    3.2.5 Step 05: Convert ERD into table ........................................................................... 50
    3.2.6 Step-06: Normalization ....................................................................................... 53
    3.2.7 Step 07: Database tuning by changing parameters ............................................ 54
    3.2.8 Step 08: Fully normalized database ................................................................... 54
    3.2.9 Step 09: De-normalized as required ................................................................... 55
    3.2.10 Step 10: Schema Tunes ...................................................................................... 58
  3.3 Design Approaches 2: ER-Relational-Universal approach ............................................. 60
  3.4 Design Approaches 3: ER-Relational-Binary Approach ................................................... 62
  3.5 Design Approaches 4: Object-oriented to Object Relational Database .................... 65
    3.5.1 Selection of Objects ............................................................................................. 65
    3.5.2 Identify of objects attributes ................................................................................. 67
    3.5.3 Relationship identification ................................................................................... 67
    3.5.4 O-R Database design for LIS .............................................................................. 70
  3.6 Design Approaches 5: Nested relational model ............................................................... 72

Chapter 4 Results and Discussion ......................................................................................... 74
  4.1 Performance analysis of Relational, Universal and Nested-relational model ............ 74
  4.2 Normalization and space gain .......................................................................................... 82
  4.3 Performance Tuning and De-normalization as per required ........................................ 86
    4.3.1 Find the tables for de-normalization ................................................................... 86
  4.4 Performance analysis between Entity-Attribute-Value (EAV), Universal and Binary
      database design ............................................................................................................. 97
  4.5 Performance gain using partitioning ............................................................................... 104
  4.6 Performance tuning changing database parameters ...................................................... 113
    4.6.1 Tuning Share Pool Size ....................................................................................... 114
    4.6.2 Tuning Buffer Cache ............................................................................................ 116
  4.7 Space calculation in the nested relational model ........................................................... 118
  4.8 Conceptual Data Modeling Using Spatial Pictogram Extension (ER Model) .......... 121
## List of Figures

<table>
<thead>
<tr>
<th>Fig. no.</th>
<th>Title</th>
<th>Page NO.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1:</td>
<td>Information system development</td>
<td>2</td>
</tr>
<tr>
<td>Figure 1.2:</td>
<td>Design Approaches</td>
<td>3</td>
</tr>
<tr>
<td>Figure 1.3:</td>
<td>Database design spectrum</td>
<td>3</td>
</tr>
<tr>
<td>Figure 2.1:</td>
<td>Design approaches for relational model</td>
<td>5</td>
</tr>
<tr>
<td>Figure 2.2:</td>
<td>Relational design approaches</td>
<td>6</td>
</tr>
<tr>
<td>Figure 2.3:</td>
<td>Universal design approaches</td>
<td>6</td>
</tr>
<tr>
<td>Figure 2.4:</td>
<td>Binary design approaches</td>
<td>7</td>
</tr>
<tr>
<td>Figure 2.5:</td>
<td>Design approaches for Object Oriented model</td>
<td>7</td>
</tr>
<tr>
<td>Figure 2.6:</td>
<td>Object Oriented design model</td>
<td>8</td>
</tr>
<tr>
<td>Figure 2.7:</td>
<td>Design approaches for Object Relational Model</td>
<td>8</td>
</tr>
<tr>
<td>Figure 2.8:</td>
<td>Nested design approaches</td>
<td>9</td>
</tr>
<tr>
<td>Figure 2.9:</td>
<td>Collapsing tables for de-normalization</td>
<td>14</td>
</tr>
<tr>
<td>Figure 2.10:</td>
<td>Mapping a polygon in relational database tables</td>
<td>16</td>
</tr>
<tr>
<td>Figure 2.11:</td>
<td>Relational schema for spatial data</td>
<td>17</td>
</tr>
<tr>
<td>Figure 2.12:</td>
<td>Universal model example</td>
<td>18</td>
</tr>
<tr>
<td>Figure 2.14:</td>
<td>Binary model example</td>
<td>19</td>
</tr>
<tr>
<td>Figure 2.15:</td>
<td>Special hierarchy</td>
<td>21</td>
</tr>
<tr>
<td>Figure 2.16:</td>
<td>Class hierarchy</td>
<td>21</td>
</tr>
<tr>
<td>Figure 2.17:</td>
<td>Vector class representation</td>
<td>22</td>
</tr>
<tr>
<td>Figure 2.18:</td>
<td>Nested relational model</td>
<td>23</td>
</tr>
<tr>
<td>Figure 2.19:</td>
<td>SQL scripts to create nested relation</td>
<td>25</td>
</tr>
<tr>
<td>Figure 2.20:</td>
<td>Inheritance in the database objects</td>
<td>26</td>
</tr>
<tr>
<td>Figure 2.21:</td>
<td>SQL scripts for inheritance</td>
<td>26</td>
</tr>
<tr>
<td>Figure 2.22:</td>
<td>SQL scripts for multiple inheritance</td>
<td>27</td>
</tr>
<tr>
<td>Figure 2.23:</td>
<td>nesting and un-nesting entity set</td>
<td>29</td>
</tr>
<tr>
<td>Figure 2.24:</td>
<td>Objects in space and their attribute value</td>
<td>31</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>2.25</td>
<td>Three basic abstraction line, point, and region</td>
<td>32</td>
</tr>
<tr>
<td>2.26</td>
<td>Partitions and Networks</td>
<td>32</td>
</tr>
<tr>
<td>2.27</td>
<td>Modeling entities position in space</td>
<td>34</td>
</tr>
<tr>
<td>2.28</td>
<td>Spatial pictogram examples</td>
<td>36</td>
</tr>
<tr>
<td>2.29</td>
<td>the GISER model</td>
<td>37</td>
</tr>
<tr>
<td>2.30</td>
<td>Geometry class hierarchy</td>
<td>38</td>
</tr>
<tr>
<td>2.31</td>
<td>Schema for feature tables under SQL92</td>
<td>39</td>
</tr>
<tr>
<td>3.1</td>
<td>Design Approaches</td>
<td>43</td>
</tr>
<tr>
<td>3.2</td>
<td>Relational database design using ER-Normalized-Denormalized approach</td>
<td>44</td>
</tr>
<tr>
<td>3.3</td>
<td>Fixed column design</td>
<td>48</td>
</tr>
<tr>
<td>3.4</td>
<td>Many to many relation design</td>
<td>49</td>
</tr>
<tr>
<td>3.5</td>
<td>Nested relational design</td>
<td>50</td>
</tr>
<tr>
<td>3.6</td>
<td>SQL scripts for a Nested schema</td>
<td>50</td>
</tr>
<tr>
<td>3.7</td>
<td>Sales-Produce schema</td>
<td>51</td>
</tr>
<tr>
<td>3.8</td>
<td>State of life</td>
<td>51</td>
</tr>
<tr>
<td>3.9</td>
<td>Temporal relational schema</td>
<td>52</td>
</tr>
<tr>
<td>3.10</td>
<td>SQL scripts to convert time stamp data</td>
<td>52</td>
</tr>
<tr>
<td>3.11</td>
<td>Normalization scale</td>
<td>53</td>
</tr>
<tr>
<td>3.12</td>
<td>Mapping among different levels in a database system</td>
<td>53</td>
</tr>
<tr>
<td>3.13</td>
<td>Algorithms for filling frequency matrix</td>
<td>57</td>
</tr>
<tr>
<td>3.14</td>
<td>Customer- Product conceptual schema</td>
<td>58</td>
</tr>
<tr>
<td>3.15</td>
<td>SQL scripts for relational schema</td>
<td>58</td>
</tr>
<tr>
<td>3.16</td>
<td>Partitioning of Customer- Product conceptual schema</td>
<td>59</td>
</tr>
<tr>
<td>3.17</td>
<td>Database design universal design</td>
<td>60</td>
</tr>
<tr>
<td>3.18</td>
<td>ER diagram of Land Information System (LIS)</td>
<td>60</td>
</tr>
<tr>
<td>3.19</td>
<td>Universal schema of Land Information System (LIS)</td>
<td>61</td>
</tr>
<tr>
<td>3.20</td>
<td>Database design universal design</td>
<td>62</td>
</tr>
<tr>
<td>3.21</td>
<td>Design approaches for object relational database</td>
<td>65</td>
</tr>
<tr>
<td>3.22</td>
<td>Association rules</td>
<td>68</td>
</tr>
<tr>
<td>3.23</td>
<td>Land use design using association</td>
<td>68</td>
</tr>
<tr>
<td>3.24</td>
<td>Association and generalization</td>
<td>69</td>
</tr>
<tr>
<td>3.25</td>
<td>Composition</td>
<td>70</td>
</tr>
<tr>
<td>3.26</td>
<td>UML diagram for CLMS database</td>
<td>70</td>
</tr>
<tr>
<td>3.27</td>
<td>Nested relation design</td>
<td>72</td>
</tr>
</tbody>
</table>
Figure 3.28: Nested presentation of collection ................................................................. 72
Figure 3.29: Nested presentation of spatial data ................................................................. 73
Figure 4.1: Fixed Column Design (Design Option -1) ......................................................... 74
Figure 4.2: Convert ERD to universal relational schema ...................................................... 75
Figure 4.3: Many-many relational design (Design option -2) .............................................. 75
Figure 4.4: Convert ERD to Relational schema ................................................................. 75
Figure 4.5: Nested relational design (Design option -3) ..................................................... 76
Figure 4.6: Convert ERD to Nested Relational ................................................................. 76
Figure 4.7: Database size vs. record (nos) ............................................................................ 77
Figure 4.8: Response time for point query ............................................................................ 78
Figure 4.9: SQL scripts for point query ................................................................................. 78
Figure 4.10: Response time for aggregate query ................................................................. 79
Figure 4.11: SQL scripts for aggregate query ....................................................................... 80
Figure 4.12: Response time analysis for range query .......................................................... 81
Figure 4.13: SQL scripts for range query .............................................................................. 82
Figure 4.21: Entity relationship diagram ............................................................................ 86
Figure 4.22: De-nominalization applied in the CLMS ......................................................... 94
Figure 4.23: Response time for point query ........................................................................ 94
Figure 4.24: Response time for Aggregate query .............................................................. 95
Figure 4.25: Response time for range query ...................................................................... 96
Figure 4.26: Universal presentation of document entity ....................................................... 97
Figure 4.27: Binary representation of entity set ................................................................... 98
Figure 4.28: Space requirements for Binary, EAV and Universal database ....................... 98
Figure 4.29: Response time for select query in universal, EAV and binary database .......... 99
Figure 4.30: Response time for Aggregate with five fields ............................................... 100
Figure 4.31: Response time for Aggregate with ten fields ............................................... 101
Figure 4.32: Response time for Aggregate with fifteen fields .......................................... 101
Figure 4.33: Response time for range query with five fields ............................................ 102
Figure 4.34: Response time for Range query with fifteen fields ....................................... 103
Figure 4.35: Conceptual schema of customer sales ............................................................. 104
Figure 4.36: SQL scripts to convert ERD to relational schema .......................................... 104
Figure 4.37: Partition the logical schema ............................................................................ 105
Figure 4.38: SQL scripts to convert in single partition ...................................................... 105
Figure 4.39: Thematic representation of partitioned tables .............................................. 106
Figure 4.40: Storage requirements of partition tables .................................................. 107
Figure 4.41: Response time for point query ................................................................. 108
Figure 4.42: SQL scripts for point query ................................................................. 109
Figure 4.43: Response time for full table scan .......................................................... 110
Figure 4.44: Response time for aggregate query ........................................................ 111
Figure 4.45: SQL Scripts for aggregate query ........................................................... 112
Figure 4.46: Range query for response time .............................................................. 113
Figure 4.47: System global area .................................................................................. 115
Figure 4.48: Shared pool size with respect to parse time ............................................. 116
Figure 4.49: Oracle9i Buffer Cache Advisory takes the guess work out of database memory configuration ................................................................. 118
Figure 4.50: Nested representation of CLMS database .............................................. 119
Figure 4.51: Thematic presentation nested relational design ...................................... 119
Figure 4.52: Nested table size in respect to number of records .................................. 120
Figure 4.53: ER-Diagram for LIS Database ............................................................... 121
Figure 4.54: ER diagram for CLMS database with pictogram ................................... 122
Figure 4.55: Relational presentation of spatial data ..................................................... 124
Figure 4.56: Spatial polygon data storage requirements ............................................ 128
Figure 4.57: Spatial line data storage requirements .................................................... 130
Figure 4.58: Spatial point data storage requirements .................................................. 131
# List of Tables

<table>
<thead>
<tr>
<th>Table No.</th>
<th>Title</th>
<th>Page no.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 3.1</td>
<td>Frequency of SQL statements in an application</td>
<td>55</td>
</tr>
<tr>
<td>Table 3.2</td>
<td>Objects table for LIS</td>
<td>66</td>
</tr>
<tr>
<td>Table 4.1</td>
<td>Database size vs record number</td>
<td>76</td>
</tr>
<tr>
<td>Table 4.4</td>
<td>Response time for range query</td>
<td>81</td>
</tr>
<tr>
<td>Table 4.5</td>
<td>Database size with varied number of records</td>
<td>85</td>
</tr>
<tr>
<td>Table 4.6</td>
<td>Query set, frequency and join tables</td>
<td>87</td>
</tr>
<tr>
<td>Table 4.7</td>
<td>List of join tables with table weight</td>
<td>89</td>
</tr>
<tr>
<td>Table 4.8</td>
<td>Candidates of denormalized Tables</td>
<td>91</td>
</tr>
<tr>
<td>Table 4.9</td>
<td>Frequency table applying Appriori algorithms</td>
<td>91</td>
</tr>
<tr>
<td>Table 4.10</td>
<td>Response time for point query</td>
<td>94</td>
</tr>
<tr>
<td>Table 4.11</td>
<td>Select Query (point Query) response time</td>
<td>99</td>
</tr>
<tr>
<td>Table 4.12</td>
<td>Aggregate Query response time</td>
<td>100</td>
</tr>
<tr>
<td>Table 4.13</td>
<td>Range Query response time</td>
<td>102</td>
</tr>
<tr>
<td>Table 4.14</td>
<td>Size of un-partition and partition database</td>
<td>106</td>
</tr>
<tr>
<td>Table 4.15</td>
<td>Response time for point query</td>
<td>107</td>
</tr>
<tr>
<td>Table 4.16</td>
<td>Query response time for full table scan</td>
<td>109</td>
</tr>
<tr>
<td>Table 4.17</td>
<td>Query response time for aggregate query</td>
<td>110</td>
</tr>
<tr>
<td>Table 4.18</td>
<td>Response time for Range query</td>
<td>112</td>
</tr>
<tr>
<td>Table 4.19</td>
<td>Shared pool size setting</td>
<td>116</td>
</tr>
<tr>
<td>Table 4.20</td>
<td>Nested table record width</td>
<td>120</td>
</tr>
<tr>
<td>Table 4.21</td>
<td>Size of the entity sets for different number of records</td>
<td>120</td>
</tr>
<tr>
<td>Table 4.22</td>
<td>Average record size for oplots table</td>
<td>127</td>
</tr>
<tr>
<td>Table 4.23</td>
<td>Estimated size for Plots table</td>
<td>127</td>
</tr>
<tr>
<td>Table 4.24</td>
<td>Actual Size from DB for Plots table</td>
<td>128</td>
</tr>
<tr>
<td>Table 4.25</td>
<td>Average record size for oRoads table is as follows table:</td>
<td>128</td>
</tr>
<tr>
<td>Table 4.26</td>
<td>Estimated size for oRoads table</td>
<td>129</td>
</tr>
<tr>
<td>Table 4.27</td>
<td>Actual size for oRoads table</td>
<td>129</td>
</tr>
</tbody>
</table>
Table 4.28: Record size for points table (oPoints) .................................................. 130
Table 4.29: Estimated size for points table ................................................................. 130
Table 4.30: Actual size of the table oPoints from database ..................................... 131
Table A.1: The list of SQL statements used in the Land Information System (LIS) .... 138
### List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1NF</td>
<td>First Normal Form</td>
</tr>
<tr>
<td>2NF</td>
<td>Second Normal Form</td>
</tr>
<tr>
<td>3NF</td>
<td>Third Normal Form</td>
</tr>
<tr>
<td>4NF</td>
<td>Forth Normal Form</td>
</tr>
<tr>
<td>BCNF</td>
<td>Boyce-Code Normal Form</td>
</tr>
<tr>
<td>CLMS</td>
<td>Computerization Land Management System</td>
</tr>
<tr>
<td>CODASYL</td>
<td>Conference on Data Systems Languages</td>
</tr>
<tr>
<td>COM</td>
<td>Component Object Model</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DB</td>
<td>Database</td>
</tr>
<tr>
<td>DBMS</td>
<td>Database management System</td>
</tr>
<tr>
<td>EAV</td>
<td>Entity Attribute Value</td>
</tr>
<tr>
<td>EER</td>
<td>Extended Entity-Relationship</td>
</tr>
<tr>
<td>ER</td>
<td>Entity-Relationship</td>
</tr>
<tr>
<td>ERD</td>
<td>Entity Relationship Diagram</td>
</tr>
<tr>
<td>GID</td>
<td>Geometry Identifier</td>
</tr>
<tr>
<td>GIS</td>
<td>Geographic Information Services</td>
</tr>
<tr>
<td>GISER</td>
<td>Geographic Information System Entity Relationship Model</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
</tr>
<tr>
<td>IBM</td>
<td>International Business Machine</td>
</tr>
<tr>
<td>IMS</td>
<td>Information Management System</td>
</tr>
<tr>
<td>MVD</td>
<td>Multi-Value Dependencies</td>
</tr>
<tr>
<td>ODBC</td>
<td>Open Database Connectivity</td>
</tr>
<tr>
<td>ODE</td>
<td>Object Database and Environment</td>
</tr>
<tr>
<td>OID</td>
<td>Object Identifier</td>
</tr>
<tr>
<td>Acronym</td>
<td>Term</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>OLE</td>
<td>Object Link Embedded</td>
</tr>
<tr>
<td>OO</td>
<td>Object Oriented</td>
</tr>
<tr>
<td>OODBMS</td>
<td>Object Oriented Database Management System</td>
</tr>
<tr>
<td>OR</td>
<td>Object Relational</td>
</tr>
<tr>
<td>ORDBMS</td>
<td>Object Relational Database Management System</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>RDBMS</td>
<td>Relational Database Management System</td>
</tr>
<tr>
<td>SDT</td>
<td>Spatial Data Types</td>
</tr>
<tr>
<td>SQL</td>
<td>Structure Query Language</td>
</tr>
<tr>
<td>SRID</td>
<td>Spatial Reference System ID</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>XML</td>
<td>Extended Markup Language</td>
</tr>
<tr>
<td>XSQL</td>
<td>Extended Structure Query Language</td>
</tr>
</tbody>
</table>
Acknowledgement

I want to express my cordial gratitude to my supervisor, Dr. Abu Sayed Md. Latiful Hoque, Associate Professor, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka. He gives me the opportunity to work in the field of database and make new idea in the database design strategies. The proper guideline of my supervisor help me to complete this research. His extra ordinary knowledge on database help me to find path in the vast area of database.

I also like to acknowledge to my mother, my wife Fahmida Hoque for her great support, my little children Nashita and Noshin for their inspiration, other family members and friends for their wishes for completing the research work.
Abstract

The output of the conceptual database design may be a relational database, universal relational database, binary relational database, object database or object relational database. There are also a number of different ways to achieve these different database solutions. The performance of the database in terms of storage and query processing is greatly influenced by the database design strategies or approaches. It is difficult to find a single strategy to achieve good performance in all types of the database applications.

This thesis presents a number of approaches for the design of relational and object-relational databases. We have designed a number of test cases to evaluate the performance of the design approaches and to evaluate the performance of the database.

On the basis of different data models, analytical evaluation is performed for all the design approaches. Experimental evaluation for all different approaches has been performed by using both synthetic and real data sets with different characteristics. The main components that we have considered for different approaches are denormalization, nested relations and object orientation. We have developed an algorithm for denormalization, based on SQL query frequency in different modules in an application. We have applied this algorithm in real as well as synthetic data sets and found effective in schema denormalization. We have evaluated universal, binary, object and object related schema for different applications cases and found that relation schema offers moderate performance in terms of both storage and response time.
Chapter 1
Introduction

Database management system (DBMS) have become an acceptable tool for storing data in a form that is concurrently accessible for multiple users, prevent loss of data, and provide security for access. The relational data model by Codd [1] is the basis of Relational Database Management System (RDBMS). By using a database management system, all task of low-level data management are removed from programmer’s responsibility. In the perspective of application developer data are described by their logical properties, not the physical structure in which they are stored. However the complexity of data requirements increases day by day, the design approaches become an important role to map complex real life data modeling to the objects in the database.

From early 1980’s, RDBMS has been dominating in the field of high performance database processing. However RDBMS failed to support database features like complex data structure as spatial, multimedia and semi-structured data [2]. To incorporate complex structure to the database, Object-Oriented Database Management System (OODBMS) has been evolved [3]. So in 1990’s, a number of OODBMS like ObjectStore [4], ODE [5], ORION [6], SHORE [7] have been developed. None of these systems could show equivalent performance like RDBMS for structured data. Object-Relational Database Management System (ORDBMS) [3] has been developed to make a bridge between relational and object-oriented database systems. Currently all commercial popular DBMS like Oracle 10g [8], SQL Server 2005 [9], PostGRESQL [10] are Object Relational DBMS [11].

The management of complex type spatial data requires both relational and object-oriented features because of its complexity [12]. There are alternative design options, e.g., a relational system with limited extension of objects, an object-based system, or a typed object-relational system. The research question is to explore an efficient design methodology to develop for object relational data management systems with
diversified problem domain. Very few work is found in the literature [12] that evaluates the performance of design approaches in this type of system.

1.1 Information System Development

![Figure 1.1: Information system development](image)

Information system development consists of system analysis, system development and database design. In the Figure-1.1 shows the stages of Information system development. Every part of the Information system development is straight forward. Our focus is on the database design strategies because systems overall performance, efficiency, storage depends on the database design approaches.
1.2 Overview of the Database Design Approaches

There are a number of design approaches for relational and object-relational databases. In the Figure 1.2, we have considered four widely used conceptual approaches, ER [13], Extended ER[13], Object oriented[4] and Spatial ER[14]. The destination database design is Relational[1], Binary Relational[3], Universal[15], Object database[4] and Object-Relational[4].

![Figure 1.2: Design Approaches](image)

The spectrum of the database is Relational to Object Relational.

![Figure 1.3: Database design spectrum](image)

The relationship between the design approach and the target database is shown in Figure 1.2 and the spectrum of database is given in the Figure 1.3. There are many
paths as shown in Figure 1.2 from the data analysis the target database design. The research question is which approach has to be used for an enterprise database design and which destination to terminate.

1.3 Objective

The Objectives of this research are:

- To define algorithms of design approaches for relational and object relational database.
- To apply these approaches to the development of relational and object relational database systems.
- To evaluate the performance of the design approach in terms of storage and query processing.
- To device a number of guidelines to select an approach for development of information systems with different scalability.

1.4 Organization of the thesis

In chapter 2, survey of the research in the database design approaches, data modeling, database design architecture is presented.

Chapter 3 presents algorithms for a number of design approaches for relational and object relational database. The details description of the steps of database design approaches in relational, object relational, universal, binary, object oriented design are discussed.

Chapter 4 gives the detailed analysis of experimental results and discussion. The performance of the design approaches have been analyzed in terms of query response time and space requirement.

Chapter 5 presents the conclusion and suggestions for future work.
Chapter 2

Literature Review

2.1 Data Model

2.1.1 Relational Model

Relational Database Management System (RDBMS) is based on the relational model developed by Codd [I]. A relational database allows the definition of data structures, storage and retrieval operations and integrity constraints. In such a database the data and relations between them are organized in tables. A table is a collection of records and each record in a table contains the same fields. Certain fields may be designated as keys, which means that searches for specific values of that field will use indexing to speed them up. Where fields in two different tables take values from the same set, a join operation can be performed to select related records in the two tables by matching values in those fields.

![Figure 2.1: Design approaches for relational model](image)

There are three design approaches can be considered for relational data model, in the Figure-2.1 describes the design approaches for relational model.
Relation Design approaches

The output of this design approach is normalized relational database.

<table>
<thead>
<tr>
<th>CUS ID</th>
<th>Last Name</th>
<th>First Name</th>
<th>Order no</th>
<th>Date</th>
<th>CUS ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>XX</td>
<td>XX</td>
<td>1001</td>
<td>XX</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>XY</td>
<td>XX</td>
<td>1122</td>
<td>XY</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>ZZ</td>
<td>ZX</td>
<td>2323</td>
<td>XY</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>AA</td>
<td>BB</td>
<td>2334</td>
<td>AA</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>XX</td>
<td>XX</td>
<td>1235</td>
<td>XX</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 2.2: Relational design approaches

Advantage:
- high database performance
- well structured
- easily understandable
- suitable for small to very large database

Disadvantage:
- Support only primitive data types
- Require joining for query output
- Produce more table in the database

Universal Design Approaches

The flat file or universal database is one of the oldest database design approaches

<table>
<thead>
<tr>
<th>CUS ID</th>
<th>Last Name</th>
<th>First Name</th>
<th>Order no</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>XX</td>
<td>XX</td>
<td>1001</td>
<td>XX</td>
</tr>
<tr>
<td>2</td>
<td>XY</td>
<td>XX</td>
<td>1122</td>
<td>XY</td>
</tr>
<tr>
<td>3</td>
<td>ZZ</td>
<td>ZX</td>
<td>2323</td>
<td>XY</td>
</tr>
<tr>
<td>4</td>
<td>AA</td>
<td>BB</td>
<td>2334</td>
<td>AA</td>
</tr>
<tr>
<td>5</td>
<td>XX</td>
<td>XX</td>
<td>1235</td>
<td>XX</td>
</tr>
</tbody>
</table>

Figure 2.3: Universal design approaches

Advantage:
- Less joining in the query processing
- Easy to understand

Disadvantage:
- Data redundancy and inconsistent
- Require high volume storage

Binary Design approaches
Storage model based on the vertical fragmentation of entity set.

<table>
<thead>
<tr>
<th>ID</th>
<th>CUS ID</th>
<th>ID</th>
<th>Last Name</th>
<th>ID</th>
<th>First Name</th>
<th>ID</th>
<th>Order no</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>1</td>
<td>1001</td>
<td>XX</td>
<td>1001</td>
<td>XX</td>
<td>1001</td>
<td>1001</td>
</tr>
<tr>
<td>1002</td>
<td>2</td>
<td>1002</td>
<td>XY</td>
<td>1003</td>
<td>XX</td>
<td>1002</td>
<td>1122</td>
</tr>
<tr>
<td>1003</td>
<td>3</td>
<td>1003</td>
<td>ZZ</td>
<td>1003</td>
<td>2X</td>
<td>1003</td>
<td>2323</td>
</tr>
<tr>
<td>1004</td>
<td>4</td>
<td>1004</td>
<td>AA</td>
<td>1004</td>
<td>BB</td>
<td>1004</td>
<td>2334</td>
</tr>
<tr>
<td>1005</td>
<td>5</td>
<td>1005</td>
<td>XX</td>
<td>1005</td>
<td>1235</td>
<td>1005</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2.4: Binary design approaches**

Advantage:
- Less storage required
- Quick output for small query

Disadvantage:
- Require more joining for query output
- Complex query for large output

### 2.1.2 Object-oriented model

Object DBMSs add database functionality to object programming languages [4]. They bring much more than persistent storage of programming language objects. Object DBMSs extend the semantics of the C++, Smalltalk and Java object programming languages to provide full-featured database programming capability, while retaining native language compatibility. A major benefit of this approach is the unification of the application and database development into a seamless data model and language environment. As a result, applications require less code, use more natural data modeling, and code bases are easier to maintain. Object developers can write complete database applications with a modest amount of additional effort.

**Figure 2.5: Design approaches for Object Oriented model**

The design approaches for Object oriented model is described in the Figure-2.5.
**Object-oriented Design**

To support the complex data type and Object Oriented concepts in DBMS a number of OO-DBMS projects is started from early 1980’s. A number of DMS has come like ODE [6], ORION [7], SHORE [8] in 1990’s.

![Object Oriented design model](image)

**Figure 2.6: Object Oriented design model**

**Advantage:**
- More object based
- Support complex data type
- Able to create user data type

**Disadvantage**
- Difficult to mapping in database
- All query type not support by SQL

### 2.1.3 Object relational model

Object/relational database management systems (ORDBMSs) add new object storage capabilities to the relational systems at the core of modern information systems [4]. These new facilities integrate management of traditional fielded data, complex objects such as time-series and geospatial data and diverse binary media such as audio, video, images, and applets. By encapsulating methods with data structures, an ORDBMS server can execute complex analytical and data manipulation operations to search and transform multimedia and other complex objects.

![Object Relational Model](image)

**Figure 2.7: Design approaches for Object Relational Model**
The design approaches for object relational model is shown in the above Figure-2.7, Object relational approaches and nested relational approaches for object relational model.

**Nested relational model**

The nested relational model is a database model that can be relational column shown in Figure 2.8

![Figure 2.8: Nested design approaches](image)

**Advantage:**
- More powerful use of object types
- The value of attribute of one tuple can be the entire relation

**Disadvantage:**
- Require more storage
- Complex query for DML operation

### 2.1.4 Semi-Structure data model

In semi-structured data model [19], the information that is normally associated with a schema is contained within the data, which is sometimes called "self-describing". In such database there is no clear separation between the data and the schema, and the degree to which it is structured depends on the application. In some forms of semi-structured data there is no separate schema, in others it exists but only places loose constraints on the data.

Semi-structured data has recently emerged as an important topic of study for a variety of reasons. First, there are data sources such as the Web, which we would like to treat as databases but which cannot be constrained by a schema. Second, it may be desirable to have an extremely flexible format for data exchange between disparate databases. Third, even when dealing with structured data, it may be helpful to view it as semi-structured for the purposes of browsing.
2.1.5 Associative data model

The associative model[3] of data is an alternative data model for database systems. In the associative model, everything which has "discrete independent existence" is modeled as an entity, and relationships between them are modeled as associations. The granularity at which data is represented is similar to schemes presented by Chen [11].

2.1.6 Entity-Attribute-Value model

Entity-attribute-value model (EAV)[3], also known as object-attribute-value model and open schema is a data model that is used in circumstances where the number of attributes (properties, parameters) that can be used to describe a thing (an "entity" or "object") is potentially very vast, but the number that will actually apply to a given entity is relatively modest. The are number of benefits in this data model:

- Flexibility. There are no arbitrary limits on the number of attributes per entity. The number of parameters can grow as the database evolves, without schema redesign.
- Space-efficient storage for highly sparse data: One need not reserve space for attributes whose values are null.
- A simple physical data format with partially self-describing data. Maps naturally to interchange formats like XML.

2.1.7 Hierarchical data model

The hierarchical data[16] model organizes data in a tree structure. There is a hierarchy of parent and child data segments. Each parent can have many children but each child only has one parent. All attributes of a specific record are listed under an entity type. In a database, an entity type is the equivalent of a table; each individual record is represented as a row and an attribute as a column. Entity types are related to each other using 1: N mapping, also known as one-to-many relationships. Hierarchical DBMSs were popular from the late 1960s, with the introduction of IBM's Information Management System (IMS)[17] DBMS, through the 1970s.
2.1.8 Network data model

The popularity of the network data model[18] coincided with the popularity of the hierarchical data model. Some data were more naturally modeled with more than one parent per child. So, the network model permitted the modeling of many-to-many relationships in data. The network model allows each record to have multiple parent and child records. In 1969, the Conference on Data Systems Languages (CODASYL) established the first specification of the network database model. This was followed by a second publication in 1971, which became the basis for most implementations. Subsequent work continued into the early 1980s, culminating in an ISO specification, but this had little influence on products.

2.2 Database design for Relational Model

As described earlier relational model was formally introduced by Codd [1] in 1970. This database management system has the capability to recombine data elements to form different relations resulting in a great flexibility of data usage. Croswell[12] describes: A database structure commonly used in GIS in which data is stored based on two dimensional tables where multiple relationships between data elements can be defined and established in an ad-hoc manner.

2.2.1 The Entity-Relationship Model

The Entity-Relationship (ER) model was proposed by Peter Chen in 1976[11], as a way to unify the network and relational database views. Simply stated the ER model is a conceptual data model that views the real world as entities and relationships. A basic component of the model is the Entity-Relationship diagram which is used to visually represents data objects.

Basic Constructs of E-R Modeling

- Entities: Entities are the principal data object about which information is to be collected. Entities are usually recognizable concepts, either concrete or abstract, such as person, places, things, or events which have relevance to the database.
• Relationships: A Relationship represents an association between two or more entities.

• Attributes: Attributes describe the entity of which they are associated. A particular instance of an attribute is a value.

• Degree of a Relationship: The degree of a relationship is the number of entities associated with the relationship. The n-ary relationship is the general form for degree n. Special cases are the binary, and ternary, where the degree is 2, and 3, respectively.

• Connectivity: The connectivity of a relationship describes the mapping of associated entity instances in the relationship. The values of connectivity are "one" or "many".

• Cardinality: The cardinality of a relationship is the actual number of related occurrences for each of the two entities. The basic types of connectivity for relations are: one-to-one, one-to-many, and many-to-many.

2.2.2 Normalization

Normalization is a process which produces a database with minimal redundancy suggested by Codd[1], 1970. This follows a well-defined set of steps. The first stage is to identify what needs to be stored in the database. If this already has some grouping involved, that grouping should be maintained in the first instance and given a unique identifier if one does not already exist.

The first step in the normalization methodology is to remove any repeating data from the initial groupings described by Kendall[20], 1999, for example, if fields describing a plot number and a “landuse” were grouped together, the landuse fields should be removed as a separate grouping. This is because a plot may own many landuse. The landuse data is therefore repeating.

A link between the two is maintained by storing the unique identifier (key) of the original entity in the extracted group of fields which become a new entity. In the example, the unique owner identifier (owner id) would be stored in the new plot_owner table. If the plot identifier (plot id) were not unique over the whole problem domain (something which could be caused by owners using their own plot
identifiers), the key of the plot_owner table would consist of both the owner id and plot id. This structure is called a composite key. The owner id field serves as the link or relation between the two tables. Failure to do this step properly results in major inefficiencies in the database including, in this case, duplication of all the grower information. If this step is done correctly, most of the problems of relational database design are solved.

The next two steps are checks to make sure attributes are attached to the appropriate entities. There is also a check to make sure data which can be calculated is not stored. The final result is a set of relations which is in Third Normal Form. Normalization provides numerous benefits to a database. Some of the major benefits include the following:

- Greater overall database organization
- Reduction of redundant data
- Data consistency within the database
- A much more flexible database design
- A better handle on database security

2.2.3 Problems with Normalization

Normalization is a well-developed methodology, which if followed, produces a schema design which can be directly converted into tables in RDBMS such as Access and Oracle. However it is very easy to make a mistake in the process and develop too many tables, or assign attributes to the wrong table. Users also find the process difficult to understand, particularly when tables are created who's only purpose is to avoid many to many links in the design. Normalization may leads to reduced database performance. The acceptance of reduced performance requires the knowledge that when a query or transaction request is sent to the database, there are factors involved, such as CPU usage, memory usage, and input/output (I/O). A normalized database requires much more CPU, memory, and I/O to process transactions and database queries than does a de-normalized database. A normalized database must locate the requested tables and then join the data from the tables to either get the requested information or to process the desired data.
2.2.4 De-normalization as per required

Denormalization can be described as a process for reducing the degree of normalization with the aim of improving query processing performance. One of the main purposes of denormalization is to reduce the number of physical tables that must be accessed to retrieve the desired data by reducing the number of joins needed to derive a query answer [21].

The objective of normalization is to organize data into stable structures, and thereby minimize update anomalies and maximize data accessibility. It can be used as a supplemental tool to provide an additional check on the stability and integrity of an Entity Relationship Diagram (ERD) and produce naturally normalized relational schemas. However, converting the conceptual entity-relationship model into database tables does not guarantee the best performance. Furthermore, normalization can create retrieval inefficiencies where a comparatively small amount of information is being sought and retrieved from the database. Normalization results in many benefits, there is at least one major drawback – poor system performance [21].

The primary goals of de-normalization are to improve query performance and to present the end user with less complex data. This leads to reduce join between tables in database which ultimate improve performance because joining is a very expensive database operation. But de-normalization have some drawback, such as data duplication, more complex data-integrity rules, update anomalies, and increased difficulty in expressing the type of access. De-normalization usually speeds up retrieval, but it can slow data modification process.

One of the most common and secure de-normalization techniques is the collapsing of

![Figure 2.9: Collapsing tables for de-normalization](image-url)
One-to-One relationships. This situation occurs when for each row of entity A, there is only one related row in entity B. While the key attributes for the entities may or may not be the same, their equal participation in a relationship indicates that they can be treated as a single unit. For example, if users frequently need to see COL1, COL2, and COL3 at the same time and the data from the two tables is in a One-to-One relationship, the solution is to collapse the two tables into one as shown in Figure 2.9. There are several nice advantages of this technique in the form of reduced number of foreign keys on tables, reduced number of indexes (since most indexes are created based on primary/foreign keys), reduced storage space, and reduced amount of time for data modification. Moreover, combining the attributes does not change the business view but does decrease access time by having fewer physical objects and reducing overhead. In general, collapsing tables in One-to-One relationship has fewer drawbacks than others. A Many-to-Many relationship can also be a candidate for the table collapsing. The typical Many-to-Many relationship is represented in the physical database structure by three tables: one table for each of two primary entities and another table for cross-referencing them. A cross-reference or intersection between the two entities in many instances also represents a business entity. These three tables can be merged into two if one of the entities has little data apart from its primary key (i.e. there are not many functional dependencies with the primary key). Such an entity could be merged into the cross-reference table by duplicating the attribute data. There is of course a drawback to this approach. Because data redundancy may interfere with updates, update anomalies may occur when the merged entity has instances that do not have any entries in the cross-reference table. Collapsing the tables in both One-to-One and One-to-Many eliminates the join, but the net result is that there is a significant loss at the abstract level because there is no conceptual separation of the data. In general, collapsing tables in Many-to-Many relationship has a significant number of problems compared to other de-normalization approaches. As de-normalization process improve query processing but increase data duplicacy and data inconsistency, we should follow some guideline during de-normalization process, 1) Identify relations and their candidates, 2) Identify the transaction/module relations and 3) identify the access paths required by each transactions. De-normalizing databases is a critical issue because of the
important trade-offs between system performance, ease of use, and data integrity. Thus, a database designer should not blindly demoralize data without good reason, but should balance the level of normalization with performance considerations in concert with the application and system needs.

2.2.5 Spatial data modeling using relational database design approaches

Relational database system is higher facility to external data management, give flexibility to access GIS and other data. The better data query relates tables using joining or programming. Limited GIS functionalities are applied to external data. In the traditional database system support simple-data types, e.g., number, char, string, time, date. Modeling spatial data types can be done but it is tedious. Example: Modeling of a polygon using numbers requires three new tables: Polygon, Edge, Point. A simple polygon is a polyline with identical first and last point. A simple square polygon is represented as 16 rows across 3 tables, simple spatial operators like area() require joining tables which is tedious and computationally inefficient. In the Figure 2.10 shown Mapping a polygon in relational database.

![Figure 2.10: Mapping a polygon in relational database tables](image-url)
2.2.6 Relational model for spatial data

For represent the Spatial data in a relational database model the Oracle spatial uses four database tables. This database structure is modeled on the first of three OpenGIS[22] Features for SQL Implementation options, namely, using numeric SQL types for geometry storage. The relational model uses a table with a predefined set of columns of type NUMBER and one or more rows for each geometry instance.

In the relational model have following characteristics:

- Each geometry is required to be uniquely identified by a geometry identifier (GID) associating it with the other attributes of the feature.
- A complex geometry such as a polygon with holes would be stored as a sequence of polygon elements. All sub elements of a multi-element polygon are wholly contained within the outermost element.
- One must directly access to the index tables.

Database Structures for the Relational Implementation: The four tables, used to store and index geometries, are collectively referred to as a layer.

Table 1–1 <layername>_sdolayer Table

<table>
<thead>
<tr>
<th>$\text{sdo_ordcnt}$</th>
<th>$\text{sdo_level}$</th>
<th>$\text{sdo_numtiles}$</th>
<th>$\text{sdo_maxlevel}^1$</th>
<th>$\text{sdo_coordsys}^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
<td>&lt;varchar&gt;</td>
</tr>
</tbody>
</table>

1 $\text{sdo\_maxlevel}$ is an optional column. 2 $\text{sdo\_coordsys}$ is an optional column.

Table 1–2 <layername>_sdodim table or view

<table>
<thead>
<tr>
<th>$\text{sdo_dimnum}$</th>
<th>$\text{sdo_lb}$</th>
<th>$\text{sdo_ub}$</th>
<th>$\text{sdo_tolerance}$</th>
<th>$\text{sdo_dimname}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
<td>&lt;varchar&gt;</td>
</tr>
</tbody>
</table>

Table 1–3 <layername>_sdogeom table or view

<table>
<thead>
<tr>
<th>$\text{sdo_gid}$</th>
<th>$\text{sdo_eseq}$</th>
<th>$\text{sdo_etype}$</th>
<th>$\text{sdo_seq}$</th>
<th>$\text{sdo_x1}$</th>
<th>$\text{sdo_y1}$</th>
<th>...</th>
<th>$\text{sdo_xn}$</th>
<th>$\text{sdo_yn}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
<td></td>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
</tr>
</tbody>
</table>

Table 1–4 <layername>_sdoindex table

<table>
<thead>
<tr>
<th>$\text{sdo_gid}$</th>
<th>$\text{sdo_code}$</th>
<th>$\text{sdo_maxcode}^1$</th>
<th>$\text{sdo_groupcode}^2$</th>
<th>$\text{sdo_meta}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;number&gt;</td>
<td>&lt;raw&gt;</td>
<td>&lt;raw&gt;</td>
<td>&lt;raw&gt;</td>
<td>&lt;raw&gt;</td>
</tr>
</tbody>
</table>

Figure 2.11: Relational schema for spatial data
2.3 EAV database: conversion of Relational/Universal model to EAV

The Entity Value Attribute (EAV) model is open schema model. The database Relational or Universal schema can be presented in the EAV by single table which have three attributes: Entity: Name of the entity set. Attribute: Name of the attribute of an entity set. Value: the actual value of the attribute in the entity set.

EAV represent the row wise data rather than column wise, to convert relational/universal relational database schema to EAV model, we have to convert all the attributes value in row wise and all attribute name in the attribute column, the value of attributes are in the value column. For example If we consider a Meta data system of documents then the following schema for relational/universal model:

![Diagram of Universal model example](image)

**Figure 2.12: Universal model example**

In this design schema the keywords for a documents are stored in the attributes name keyword1 to keyword999, there are 999 fields, docid is the number of the documents, to convert it into EAV model we have to design a table with three attributes, the design schema is as following Figure:

![Diagram of EAV model](image)

**Figure 2.13: EAV model**

The keywords are stored in the database EAV model in a single table, there are no NULL value for documents table and keywords are stored in the value column.
2.4 Binary Model: conversion of Relational/Universal model to Binary model

The Binary relational model consider every tables are binary tables. Every single attributes of any relational schema are converted into the binary tables, binary tables consist of two attributes: Key and value. The primary key of a relational table is in the key column and value of an attribute in the value column. If we consider the above example then following schema is for binary relational model:

![Binary Model Example](image)

Figure 2.14: Binary model example

2.5 Object Oriented Model

A data model is a logic organization of the real world objects (entities), constraints on them, and the relationships among objects. A Database (DB) language is a concrete syntax for a data model. A DB system implements a data model. A core object-oriented data model consists of the following basic object-oriented concepts:

- Object and object identifier: Any real world entity is uniformly modeled as an object (associated with a unique id: used to pinpoint an object to retrieve).

- Attributes and methods: every object has a state (the set of values for the attributes of the object) and a behavior (the set of methods - program code - which operate on the state of the object). The state and behavior encapsulated in an object are accessed or invoked from outside the object only through explicit message passing.

- Class: a means of grouping all the objects which share the same set of attributes and methods. An object must belong to only one class as an instance of that class (instance-of relationship). A class is similar to an
abstract data type. A class may also be primitive (no attributes), e.g., integer, string, Boolean.

- Class hierarchy and inheritance: derive a new class (subclass) from an existing class (superclass). The subclass inherits all the attributes and methods of the existing class and may have additional attributes and methods. single inheritance (class hierarchy) vs. multiple inheritance (class lattice).

2.5.1 Object Structure

The object-oriented paradigm is based on encapsulating code and data into a single unit. conceptually, all interactions between an object and the rest of the system are via messages. Thus, the interface between an object and the rest of the system is defined by a set of allowed messages. In general, an object has associated with it:

- A set of variables that contain the data for the object. The value of each variable is itself an object.
- A set of messages to which the object responds.
- A set of methods, each of which is a body of code to implement each message; a method returns a value as the response to the message.

2.5.2 Object Class

Usually, there are many similar objects in a Database. By “similar”, it means that they respond to the same messages, use the same methods, and have variables of the same name and type. We can group similar objects to form a class. Each such object is called an instance of its class. E.g., in a bank DB, customers, accounts and loans are classes.

2.5.3 Inheritance

An object-oriented database schema typically requires a large number of classes. Often, however, several classes are similar. For example, bank employees are similar to customers. In order to allow the direct representation of similarities among classes, we need to place classes in a specialization hierarchy. In the Figure 2.15 Special hierarchy
E.g., Figure is a specialization hierarchy for the ER model. The concept of a class hierarchy is similar to that of specialization in the ER model. The corresponding class hierarchy is shown in Figure 2.13:

![Class hierarchy diagram]

2.5.4 Object Identity

Object identity: An object retains its identity even if some or all of the values of variables or definitions of methods change over time. This concept of object identity is necessary in applications but does not apply to tuples of a relational database. Object identity is a stronger notion of identity than typically found in programming languages or in data models not based on object orientation.

Several forms of identity:

- **value**: A data value is used for identity (e.g., the primary key of a tuple in a relational database).
• **name:** A user-supplied name is used for identity (e.g., file name in a file system).

• **built-in:** A notion of identity is built-into the data model or programming languages, and no user-supplied identifier is required (e.g., in OO systems).

Object identity is typically implemented via a unique, system-generated OID. The value of the OID is not visible to the external user, but is used internally by the system to identify each object uniquely and to create and manage inter-object references. There are many situations where having the system generate identifiers automatically is a benefit, since it frees humans from performing that task. However, this ability should be used with care. System-generated identifiers are usually specific to the system, and have to be translated if data are moved to a different database system. System-generated identifiers may be redundant if the entities being modeled already have unique identifiers external to the system, e.g., SIN#.

### 2.5.5 Spatial data modeling using Object-oriented approaches

In object-oriented approaches, data structures and methods are combined in the definition of classes. A hierarchy of classes is provided as a general tool for the design of spatial applications. A spatial data structure is incorporated in the model as the data structure of a class. Spatial operations are incorporated as methods of classes. Application-specific classes, with spatial capabilities, are defined as subclasses of the hierarchy provided by the system. In the Figure 2.17 represent Vector class representation.

![Figure 2.17: Vector class representation](image)

One of these approaches is the restriction to spatial data management of the spatio-temporal approach [23]. One hierarchy of classes supports the representation and management of 2-d and 3-d spatial data. Elementary features are vector or raster
objects. Vector classes described in the Figure-2.14 represent points in (Figure (a)), simple polylines (Figure (b)) and polygons (Figure (c)). A GeoObject combines a set of non-spatial properties with a collection of at least one feature. User classes with spatial functionality are defined as subclasses of GeoObject. SpatialObject is a feature or a GeoObject. Functions and predicates for the manipulation of spatial data are defined as methods of classes in this hierarchy. Operations Touch, Overlap and Cross enable obtaining those parts of the spatial intersection of two objects for which a relevant predicate evaluates to true. Their functionality is similar to that of Spatial Intersection. The boundary of a point is the empty set. The boundary of a polyline is the set of its end-points. The boundary of a polygon is a set of polylines. Further spatial functionality includes Buffer.

2.6 Object Relational Model

The object-relational data model extends the relational model by providing rich type system including object orientation and add constructs to relational query languages to deal with added (complex) data types. Such extensions attempt to preserve the relational foundations in particular, the declarative access to data, while extending the modeling power.

**Nested Relations**

INF requires that all attributes have atomic (indivisible) domains. The nested relational model is an extension of the relational model in which domains may be either atomic or relation-valued. This allows a complex object to be represented by a single tuple of a nested relation, one-to-one correspondence between data items and objects. Suppose the information to be stored consists of (i) document title, (ii) author list (set of authors), (iii) date (day, month, and year), and (iv) keyword list (list of key words). Example: A non-INF document relation, doc.

<table>
<thead>
<tr>
<th>title</th>
<th>author_list</th>
<th>date</th>
<th>keyword_list</th>
</tr>
</thead>
<tbody>
<tr>
<td>salesplan</td>
<td>{Smith, Jones}</td>
<td>1 April 89</td>
<td>{profit, strategy}</td>
</tr>
<tr>
<td>stat. report</td>
<td>{Jones, Frick}</td>
<td>17 July 94</td>
<td>{profit, personnel}</td>
</tr>
</tbody>
</table>

Figure 2.18: Nested relational model
The doc relation can be represented in 1NF, doc', but awkward. If we assume the following multi-value dependencies (MVDs) hold:

title -> author
title -> keyword
title -> day month year

we can decompose the relation into 4NF using the schemes:
(t title, author)
(t title, keyword)
(t title, day, month, year)

But the non-1NF representation may be an easier-to-understand model (closer to user's view). The 4NF design would require users to include joins in their queries, thereby complicating interaction with the system. We could define a view, but we lose the one-to-one correspondence between tuples and documents.

*Spatial data modeling using nested-relational model*

Nested-relational approaches are the *Constraint-Based Models* proposed in Grumbach [24] and Kuper [25]. At a conceptual level of abstraction, a spatial object is represented by a (possibly infinite) relation with attributes that are interpreted as the dimensions of an n-d space. At a lower level of abstraction, however, such a relation is represented by a finite set of constraints. Spatial union, difference and intersection are achieved by the relational operations *Union*, Except and *Intersect*.

Operation *Unionnest* applies the relational operation *Union* to all the relations of a relation valued attribute, provided that these relations belong to tuples whose values for another set of attributes match. The behavior of operation *Internest* is similar to that of *Unionnest*, except that *Intersection* is now applied instead of *Union*. Further functionality in both of these approaches includes the Boundary of surfaces and spatial *Complementation*.

*Complex Types and Object Orientation*

The extensions include nested relations, nested records, inheritance, references to objects, and other object-oriented features. The presentation is based on a draft of SQL-3, with the incorporation of features from XSQL and Illustra.
Structured and collection types

Inheritance

Reference Types

**Structured and collection types**

Define a relation doc with complex attributes: sets and structured attributes. Create type MyString char varying, the following Figure 2.19 SQL scripts to create nested relation.

```
Create type MyDate
   (day integer, month char(10), year char(10))
Create type Document
   (name MyString, author-list setof(MyString), date MyDate,
    keyword-list setof(MyString))
create table doc of type Document
```

**Figure 2.19: SQL scripts to create nested relation**

It allows type definition recorded in the schema stored in the database. One can also create table directly, without creating an intermediate type for the table. Complex type systems usually support other collection types, such as arrays and multisets, author-array MyString[10] presents an ordered list of authors, print-runs multiset(integer), presents the number of copies in each printing run.

**Inheritance**

Inheritance can be at the level of types or at the level of tables. Inheritance of types is described in the following Figure 2.20
**Create type Person** (name MyString, social-security integer)

**Create type Student** (degree MyString, department MyString)
under **Person**

**Create type Teacher** (salary integer, department MyString)
under **Person**

**Figure 2.20:** Inheritance in the database objects

*Multiple inheritance*
Since name and social-security are inherited from a common source Person, there is no conflict by inheriting them from Student as well as Teacher. However, department is defined separately in Student and Teacher, and one can rename them to avoid a conflict.

create type TeachingAssistant
under Student with (department as student-dept),
under Teacher with (department as teacher-dept)

**Inheritance of tables:**
To avoid creation of too many subtypes, one approach in the context of database systems is to allow an object to have multiple types without having a most specific type. Object-relational systems can model such a feature by using inheritance at the level of tables, rather than types, and allowing an entity to exist in more than one table at once. The Figure 2.21 describe SQL scripts for inheritance.

create table people (name MyString, social-security integer)
create table students (degree MyString, department MyString)
under **people**
create table teachers (salary integer, department MyString)
under **people**

**Figure 2.21:** SQL scripts for inheritance
Multiple inheritance is possible with tables. A teaching-assistant can simply belong to the table students as well as to the table teacher. However, if we want, we can create a table for teaching-assistant entities as Figure 2.2:

```
Create type TeachingAssistant
under Student with (department as student-dept),
under Teacher with (department as student-dept)
```

**Figure 2.22:** SQL scripts for multiple inheritance

Based on the consistency requirements for subtables, if an entity is present in the teaching-assistant table, it is also present in the teachers and in the students table. Inheritance makes schema definition natural, ensures referential and cardinality constraints, enables the use of functions defined for supertypes on object belonging to subtypes, and allows the orderly extension of a database system to incorporate new types.

**Reference Types**

An attribute of a type can be a reference to an object of a specific type, e.g., `ref(Person)`.

Reference to tuples: Tuples of a table can also have references to them. E.g., `ref(people)`. It can be implemented using the primary key or tuple-id. SQL-3 uses identity (for tuples) and oid for objects.

**Querying with Complex Types**

**Relation-Valued Attributes**

Our extended SQL allows an expression evaluating to a relation to appear anywhere that a relation name may appear. The ability to use sub-expressions freely makes it possible to take advantage of the structure of nested relations.

The schema for a relation pdoc.

```
create table pdoc (name MyString,
    author-list setof(ref(people)),
    date MyDate,
    keyword-list setof(MyString))
```
To find all documents having a keyword "database":

\[
\text{select name from pdoc where "database" in keyword-list}
\]

To find pairs of the form "doc-name, author-name" for each document and each author of the document:

\[
\text{select B.name, Y.name from pdoc as B, B.author-list as Y}
\]

Aggregate functions can be applied to any relation-valued expression.

\[
\text{select name, count(author-list) from pdoc}
\]

**Path Expressions**

An expression of the form "student.advisor.name" is called a path-expression. References can be used to hide join operations and thus the use of references simplifies the query considerably.

Example:

\[
\text{create table phd-students (advisor (ref(people))}
\]

\[
\text{under people}
\]

\[
\text{select phd-student.advisor.name}
\]

\[
\text{from phd-students}
\]

In general, attributes used in a path expression can be a collection, such as a set or a multiset. e.g., to get names of all authors of documents, we have

\[
\text{select Y.name from pdoc.author-list as Y}
\]

**Nesting and Unnesting**

The transformation of a nested relation into 1NF is called un-nesting. Example: To complete unnest the doc relation, we have

\[
\text{select name, A as author, date.day, date.month, date.year,}
\]

\[
K \text{ as keyword}
\]

\[
\text{from doc as B, B.author-list as A, B.keyword-list as K}
\]

The reverse operation of transformation of a 1NF relation into a nested relation is called nesting. Example. To nest the relation flat-doc on the attribute keyword, we have

\[
\text{select title, author, (day, month, year) as date,}
\]

\[
\text{set(keyword) as keyword-list from flat-doc group by title, author, date}
\]
This will generate the following Table 2.23.

<table>
<thead>
<tr>
<th>title</th>
<th>author_list</th>
<th>date</th>
<th>keyword_list</th>
</tr>
</thead>
<tbody>
<tr>
<td>salesplan</td>
<td>Smith</td>
<td>1 April 89</td>
<td>{profit, strategy}</td>
</tr>
<tr>
<td>salesplan</td>
<td>Jones</td>
<td>1 April 89</td>
<td>{profit, strategy}</td>
</tr>
<tr>
<td>status report</td>
<td>Jones</td>
<td>17 July 94</td>
<td>{profit, personnel}</td>
</tr>
<tr>
<td>status report</td>
<td>Frick</td>
<td>17 July 94</td>
<td>{profit, personnel}</td>
</tr>
</tbody>
</table>

**Figure 2.23:** nesting and un-nesting entity set

To convert flat-doc back to the nested table doc, we have:

```sql
select title, set(author) as author-list, (day, month, year) as date, set(keyword)
   as keyword-list
from flat-doc group by title, date
```

**Creation of Complex Values and Objects**
Create and updates tuples with complex types (tuple-valued and set-valued)

```sql
insert into doc values ("salesplan", set("Smith", "Jones"), (1, "April", 89),
   set("profit", "strategy")
```

We can also use complex values in queries: anywhere in a query where a set is expected, we can enumerate a set.

```sql
select name, date from doc where name in set ("salesplan", "opportunities",
   "risks")
```

To create new objects, we can use constructor functions. The constructor function for an object type $T$ is $T()$; when it is invoked it creates a new un-initialized object of type $T$, fills in its oid field, and returns the object. The fields of the object must then be initialized.

### 2.7 Spatial Data Modeling

Shashi Sheker and Others [26] describes the basic concepts of spatial data modeling, they describe as a Geographic Information System provides data abstraction by hiding details of data storage that many users need not know what happened behind. A data model is the main tool for providing this abstraction A data model is a set of concepts that can be used to describe the semantics of a GIS dataset By the semantics of a GIS dataset some one mean the data types, relationships, and constraints that
should hold for the data. Data models can be categorized by the types of concepts they provide to structure data. High-level or conceptual data models provide concepts that are close to the way many users perceive data, whereas low-level or physical data models describe the details of how data should be stored.

A popular conceptual data model, namely the entity relationship (ER) model, uses concepts such as entities, attributes and relationships [13]. An entity represents a concept or a real-world object such as a city that is stored in the database. An attribute represents some property of interest that further describes an entity for example city’s name. A relationship among two or more entities represents an interaction among the entities. An entity type usually has an attribute whose values are distinct for each instance. Such an attribute is called a key attribute. Some entity types may not have any key attributes of their own; they are called weak entity types. A weak entity is identified by its relation to a specific identity type called an identifying owner, and by some of their shared attribute values. The relationship of a weak entity to its identifying owner is the identifying relationship. ER diagrams are used to graphically represent instances of ER models.

The Enhanced-ER (EER) model enhances an ER model to include the concepts of subclass and superclass and the related concepts of specialization and generalization. In many cases, an entity type has numerous additional sub-groupings of its entities that are meaningful and need to be represented explicitly because of their significance to the database application. For example, the entities that are members of the ROAD entity type may be further grouped into FREEWAY, HIGHWAY, and so on. The entity-type ROAD is the superclass for the subclasses FREEWAY, etc. Also a subclass could be viewed as the specialization of the superclass or a superclass could be viewed as the generalization of the subclasses. A subclass inherits all the attributes of a superclass. This characteristic is called class inheritance.

In an Object-Oriented (OO) data model each instance of an entity is modeled as an object which includes a behavior description. It provides object encapsulation. Objects are comprised of both methods and data operations. Methods could also be viewed as procedure-valued attributes. This model also provides the notion of class and Inheritance.
Ralf Hartmut Güting [27] describes in his famous paper “An Introduction to Spatial Database system” that the main application driving research in spatial database systems are GIS. Hence here consider some modeling needs in this area which are typical also for other applications. Examples are given for two dimensional space, but almost everywhere, extension to the three- or more-dimensional case is possible. There are two important alternative views of what needs to be represented:

1. **Objects in space**: Distinct entities arranged in space each of which has its own geometric description. This is called entity-oriented / feature-based view vector data, spatial database systems. In the Figure 2.24 shows the Objects in space and their attributes value.

2. **Space**: Describe space itself, that is, say something about every point in space. space-oriented / position-based view , Raster data in the image database.

Reconcile both views to some extent by offering concepts for modeling i) single objects, and ii) spatially related collections of objects. For modeling single objects, the fundamental abstractions are point, line, and region. A point represents (the geometric aspect of) an object for which only its location in space, but not its extent, is relevant. For example, a city may be modeled as a point in a model describing a large geographic area (a large scale map). A line (in this context always to be understood as meaning a curve in space, usually represented by a polyline, a sequence of line segments) is the basic abstraction for facilities for moving through space, or connections in space (roads, rivers, cables for phone, electricity, etc.). A region is the abstraction for something having an extent in 2d-space, e.g. a country, a lake, or a national park. A region may have holes and may also consist of several disjoint pieces. The Figure 2.25 shows the three basic abstractions for single objects.
The two most important instances of spatially related collections of objects are partitions (of plane) and networks. In the Figure 2.26 shows the Partitions and Networks. A partition can be viewed as a set of region objects that are required to be disjoint. The adjacency relationship is of particular interest, that is, there exist often pairs of region objects with a common boundary. Partitions can be used to represent thematic maps. A network can be viewed as a graph embedded into the plane, consisting of a set of point objects, forming its nodes, and a set of line objects describing the geometry of the edges. Networks are ubiquitous in geography, for example, highways, rivers, public transport, or power supply lines.

2.7.1 Spatial Data types (SDT)
Spatial data types are special data types needed to model geometry and to suitably represent geometric data in database systems. Examples: point, line, region; partitions (maps), graphs (networks). Spatial data types provide a fundamental abstraction for modeling the geometric structure of objects in space, their relationships, properties and operations. Their definition is to a large degree responsible for a successful design of spatial data models and the performance of spatial database systems and exerts a great influence on the expressive power of spatial query languages. This is true regardless of whether a DBMS uses a relational, complex object, object-oriented, or some other kind of data model. Hence, the
definition and implementation of spatial data types is probably the most fundamental issue in the development of spatial DBMS.

2.7.2 Spatial data modeling requirements

As described by Jose R. Rios Viquera, Nikos A. Lorentzos, Nieves R. [28] spatial model should satisfy the following properties:

Spatial Data Types: It should support the point, line and surface types, since in daily practice people are familiar with the use of these objects.

Data Structures: They should be simple. As opposed to the First Normal Form (1NF) relational model, for example, it is noticed that a nested model, though more powerful, is more difficult to both implement and use. Similarly, it is penalizing for the user to process two distinct data structures.

Spatial Operations: They should apply to structures containing any type of spatial data. Two examples: It is practical to (i) apply Overlay to lines, and (ii) apply an operation to two spatial objects of a different type, such as to compute the intersection of a surface with a line. Finally, pieces of data should not be discarded from the result of an operation.

Ralf Hartmann Gütting described [27] described the requirements for spatial database architecture as follows:

- representations for the data types of a spatial algebra,
- procedures for the atomic operations,
- spatial index structures,
- access operations for spatial indices,
- filter and refine techniques,
- spatial join algorithms,
- cost functions for all these operations,
- statistics for estimating selectivity of spatial selection and spatial join,
- extensions of the optimizer to map queries into the specialized query processing methods,
- spatial data types and operations within data definition and query language,
- user interface extensions to handle graphical representation and input of SDT values.
2.7.3 An extended entity-relational model for Geographic applications

A conceptual model of geographies application was proposed by Hadzliacos and Tryfona [29] which is called Geo-ER model. Geographic Entity-Relationship model (Geo-ER) is based on standard ER model, and on the extended ER models which is integrated concept of aggregation and grouping. Geo-ER includes special entity set and relationships to express the semantics of space, geographic entities' position, entities' space-depanding attributes and spatial relationships. Two new constructs are added to express the spatial dimension of complex geographic entity sets: spatial aggregation and spatial grouping.

Entity sets whose position in space matters are called geographic (depicted by bold-

![Diagram](image)

**Figure 2.27: Modeling entities position in space**

rectangles). In the Figure 2.27 modeling entities position in space. Since ER uses entity sets and attributes thereof, they model position as a special entity set with fixed meaning: POSITIONS is a function on all and only on geographic objects and returns for each geographic object a part of space. In order to represent shape, the special entity sets 0-Dimensional, 1-Dimensional, and 2-Dimensional are introduced and are called geometric types. These sets are related to POSITIONS by an ISA hierarchy. 0-Dimensional, 1-Dimensional, and 2-Dimensional are (ISA) shapes of entity. The position of a complex geographic entity can be any combination of points, lines and
regions in space, shape (as well as position itself) is determined by the higher dimension of geometric Figures that constitute entity set's position (dimensionality). As shown in the following Figures shows entity's positions in the space.

### 2.7.4 Spatial pictogram enhanced conceptual data models

Shashi Shekhar, Ranga Raju, Sanjay Chawla [30] presents an extension to Entity-Relationships (ER) diagrams using pictograms for entities and as well as relationships. Conventional ER diagrams have limitations for conceptual spatial data modeling, first difficulty lies with geometric attributes which are continuous, and the second difficulty lies with spatial relationships. In addition the logical data model gets cluttered with redundant tables representing materialization M: N spatial relationships. In the spatial data have many topological relationships between spatial entities, in the traditional ER model that is not possible to map directly.

A pictogram is generally represented as a miniature version (icon) of an object inserted inside of a box. These pictograms are used to extend ER diagrams by inserting them at appropriate places in the entity boxes. A pictogram can be of a basic shape, a user-defined shape, or any other possible shape (see Figure 2.28-A).

**Shape:** Shape is the basic graphical element of a pictogram, which represents the geometric types in the spatial data model. It can be a basic shape, a multi-shape, a derived shape, or an alternate shape. Most objects have simple (basic) shapes (see Figure 2.28-B).

**Basic Shape:** In a vector model the basic elements are point, line and polygon. In a forestry example, the user may want to represent a facility as a point (0-D), a river or road network as lines (1-D) and forest areas as polygons (2-D) (see Figure 2.28-D and 2.28-A).
Multi-Shape: To deal with objects which cannot be represented by the basic shapes, we have defined a set of aggregate shapes. Cardinality is used to quantify multi-shapes. For example, a river network is represented as "a line pictogram with its cardinality n". Similarly, features that cannot be depicted at a certain scale will have cardinality (Figure-2.25-E).

Derived Shape: If the shape of an object is derived from the shapes of other objects, its pictogram is italicized. For example, some one can derives a forest boundary (polygon) from its "forest type" boundaries (polygon), or a country boundary from the constituent state boundaries (Figure 2.28-G).

Alternate Shape: Alternate shapes can be used for the same object depending on certain conditions; for example, objects of size less than x units are represented as points while those greater than x units are represented as polygons. Alternate shapes are represented as a concatenation of possible pictograms. Similarly, multiple shapes are needed to represent objects at deferent scales; for example, at higher scales lakes may be represented as points, and at lower scales as polygons (Figure 2.28-H).

Any Possible Shape: A combination of shapes is represented by a wild card * symbol inside a box, implying that any geometry is possible (Figure. 2.28-E).

User-Defined Shape: Apart from the basic shapes of point, line and polygon, user defined shapes are possible. User defined shapes are represented by an exclamation symbol (!) inside a box (Figure.2.28-F).
2.7.5 Geographic Information System Entity Relationship Model (GISER)

Shashi Sheker, Mark Coyle, Brajesh Goyal, Duen-Ren Liu and Shyamsundar Sarker [31] describes a model for spatial data named GISER, it used enhanced entity relationship diagrams along with dashed lines for continuous fields and relationships. The GISER model is based on 4 major concepts: Space/Time, Features, Coverages and Spatial Objects. Space and time represent boundless multi dimensional extents in which geographic phenomena and events can occur and have relative position and direction. Features represent geographic phenomena such as rivers, vegetation, cities, etc. GISER models features as continuous fields varying over space and time, but features as such cannot be stored in a GIS. These must go through the process of discretization in order to compute coverages which are then stored in a GIS. A feature may have multiple coverages based on multiple discretization with varying resolution, accuracy and sources. A coverage consists of a set of spatial objects. A spatial object occupies a subset of space and time. It has geometric and/or topological properties. A geographic object consists of a spatial object and a non-spatial object. The spatial object represents the location and spatial extent of the river at some point in time. The non-spatial object represents other attributes such as its name.

![Figure 2.29: the GISER model](image-url)
2.7.6 Geometry Object Model (OpenGIS® Simple Features Specification for SQL)

Simple geospatial feature collections will conceptually be stored as tables with geometry valued columns in a Relational DBMS (RDBMS), each feature will be stored as a row in a table. The non-spatial attributes of features will be mapped onto columns whose types are drawn from the set of standard ODBC/SQL92 data types. The spatial attributes of features will be mapped onto columns whose SQL data types are based on the underlying concept of additional geometric data types for SQL. A table whose rows represent Open GIS features shall be referred to as a feature table. Such a table shall contain one or more geometry valued columns. Feature table implementations are described for two target SQL environments: SQL92 and SQL92 with Geometry Types [22].

In the SQL92 environment, a geometry-valued column is implemented as a Foreign Key reference into a geometry table. A geometry value is stored using one or more rows in the geometry table. The geometry table may be implemented using either standard SQL numeric

![Geometry class hierarchy](image)

**Figure 2.30: Geometry class hierarchy**

types or SQL binary types, schemas for both alternatives are described. The term SQL92 with Geometry Types is used to refer to a SQL92 environment that has been extended with a set of Geometry Types. In this environment a geometry-valued
column is implemented as a column whose SQL type is drawn from the set of Geometry Types.

A SQL92 implementation of OpenGIS simple geospatial feature collections defines a schema for storage of feature table, geometry and spatial reference system information. The SQL92 implementation does not define SQL functions for access, maintenance, or indexing of geometry, as these functions cannot be uniformly implemented across database systems using the SQL92 standard. The Figure below describes the database schema necessary to support the OpenGIS simple feature data model. A feature table or view corresponds to an OpenGIS feature class. Each feature view contains some number of features represented as rows in the view. Each feature contains some number of geometric attribute values represented as columns in the feature view. Each geometric column in a feature view is associated with a particular geometric view or table that contains geometry instances in a single spatial reference system. The correspondence between the feature instances and the geometry instances shall be accomplished through a foreign key that is stored in the geometry column of the feature table. This foreign key references the GID primary key of the geometry table.

Figure 2.31: Schema for feature tables under SQL92
Depending upon the type of storage specified by the geometry metadata, Geometry instances shall be stored as either arrays of coordinate values or as binary values using an OpenGIS defined Well-Known Binary Representation for Geometry. In the former case, SQL numeric types are used for the coordinates and client side functions are needed to build OpenGIS geometry objects from the retrieved coordinate numeric values. In the latter case clients may feed the retrieved well-known binary representation directly into the Geometry factory of the client side computing environment (e.g., an OLE/COM or CORBA geometry factory) or choose to access the individual coordinate values by unpacking the well-known representation.

- The GEOMETRY_COLUMNS table describes the available feature tables and their Geometry properties.
- The SPATIAL_REF_SYS table describes the coordinate system and transformations for Geometry.
- The feature table stores a collection of features. A feature table’s columns represent feature attributes, while rows represent individual features. The Geometry of a feature is one of its feature attributes; while logically a geometric data type, a Geometry Column is implemented as a foreign key to a geometry table.
- The geometry table stores geometric objects, and may be implemented using either standard SQL numeric types or SQL binary types.

**Identification of feature tables and geometry columns**

Feature tables and Geometry columns are identified through the GEOMETRY_COLUMNS table. Each Geometry Column in the database has an entry in the GEOMETRY_COLUMNS table. The data stored for each geometry column consists of the following:

a) the identity of the feature table of which this Geometry Column is a member;
b) the name of the Geometry Column;
c) the spatial reference system ID (SRID) for the Geometry Column;
d) the type of Geometry for the Geometry column;
e) the coordinate dimension for the Geometry Column;

f) the identity of the geometry table that stores geometric objects for this Geometry column;

g) the information necessary to navigate the geometry table in the case of normalized geometry storage.
Chapter 3
Design Approaches for Relational up to Object Relational Database

The database design is a complex process. The complexity arises mainly because the identification of relationships among individual components and their representation for maintaining correct functionality are highly involved. The degree of complexity increases if there are many to many relationships among individual components. This chapter describes the details of database design strategies, measure the performance of database in terms space utilization and query response time.

3.1 Consideration of database design approaches

We have considered three main group of conceptual design approach: Entity-Relationship (ER) and Extended Entity-Relationship model (EER), Object-oriented and Spatial E-R model. The output of ER model is the Relational database, Binary relational database or Universal Relational database. These are shown in Figure 3.1. Relational, Binary relational or Universal relational database are transformable to each other. Relational database can be further transformed into fully normalized relational database by applying normalization. If we go depth first, we can form a design approach namely ER->Relational-> Normalized -> Denormalized approach.

The target of the second approach is Binary relational model. This is ER->Relational->Binary approach. The third approach is ER-Relational-Universal approach.

Spatial data has both spatial and non-spatial component. So the output of the spatial ER model may belong to relational, binary, universal (non-spatial components) and object or object-relational (spatial components). So we have ER- EER/Spatial ER-
object relational approach. EAV database can be evolved from universal or relational database.

Figure 3.1: Design Approaches
3.2 Design Approach 1: ER-Relational-Normalized-Denormalized approach

Figure 3.2 shows the flow chart of the relational database design using ER- >Normalization ->Denormalization approach.

![Diagram of relational database design](image)

Figure 3.2: Relational database design using ER-Normalized-Denormalized approach

3.2.1 Step 01: Selection of entity

To begin the modeling process, we need to first isolate the entities required for the business process. An entity is the basic division of a database. In the logical design, entities are representative of the tables that will be present when the database development process moves into the physical design phase. Each entity exists as a separate individual data collection, unique from all the other entities.
An entity is a thing that can be distinctly identified as described by Chen [13] in 1976. In database design, entities are the “things” about which the database stores information. Each entity must have multiple occurrences or instances. Entity instance has specific values for the entities attributes. Each instance of entity must be uniquely identifiable from other instances of the same entity. They can be qualified as one of three basic types: kernel entities, associative entities, and characteristic entities [13]. These entity types are described further in the following list:

- A kernel entity exists on its own; it doesn’t define or provide descriptive information for other entities. An example of a kernel entity would be a plot listing in a land management system. The information contained in each kernel entity of a table represents the heart of the database model.

- Associative entities are needed to allow multiple kernel entities to be tied together. In the land management system, a plot entity would be needed to tie an owner kernel entity to the plots they have purchased. This same plot entity could be tied to another kernel entity, such as landuse.

- A characteristic entity provides additional information for a respective kernel or associative entity. Information contained in characteristic entities can be updated independently of the related entity.

For selection of entity we should consider the following points:

- We should examine the business requirements definition or statement

- Identify those aspects of a business about which data must be known when given source documentation from a business

- All the rules like business rules, process rules, programmatic rules etc.

- Documentation (forms and reports) used by a business

3.2.2 Step 02: Selection of attributes

Identifying or selection of the attributes is the next step in ensuring a successful data modeling process. In defining attributes we are setting out to define entity composition. Each entity will have descriptive elements that pertain solely to that
element. An attribute is a descriptive element or property of an entity. Fields will represent the attributes when the logical design progresses to the physical design stage. There are several different kinds of attributes: simple or atomic, composite, multi-valued, and derived. The properties of an attribute are its name, description, format, and length, in addition to its atomicity. Some attributes may be considered unique identifiers for an entity.

- **Simple** or **atomic** attributes cannot be further broken down or subdivided, hence the notion "atomic." One can examine the domain of values of an attribute to elicit whether an attribute is simple or not. An example of a simple or atomic attribute would be Social Security number, where a person would be expected to have only one, undivided Social Security number.

- A **composite** attribute, sometimes called a group attribute, is an attribute formed by combining or aggregating related attributes. The names chosen for composite attributes should be descriptive and general. The concept of "name" is adequate for a general description, but it may be desirable to be more specific about the parts of this attribute. Most data processing applications divide the name into component parts. Name, then, is called a composite attribute or an aggregate because it is usually composed of a first name, a last name, and a middle initial sub-attributes.

- Another type of non-simple attribute that has to be managed is called a **multi-valued** attribute. The multi-valued attribute, as the name implies, may take on more than one value for a given occurrence of an entity. For example, the attribute landuse could easily be multi-valued if a plot have more than one landuse.

- **Derived** attributes are attributes that are the attributes which can be calculated from other data in the database. An example of a derived attribute would be an age that could be calculated once a Owner's birthdate is entered.

- **Key:** The sense of a database is to store data for retrieval. An attribute that may be used to find a particular entity occurrence is called a **key**. As we model our database with the ER models, we may find that some attributes naturally seem to be keys. If an attribute can be thought of as a unique
identifier for an entity, it is called a candidate key. When a candidate key is chosen to be the unique identifier, it becomes the primary key for the entity.

3.2.3 Step 03: Relationship identification

A relationship is an association among entities. Different relationship degrees exist. The degree of a relationship refers to the number of entities involved in the relationship. Although others exist, it is often sufficient to understand the meaning of three relationship degrees, unary, binary and ternary. A unary relationship (also called a recursive relationship) is a relationship involving a single entity. A relationship between two entities is called a binary relationship. When three entities are involved in a relationship, a ternary relationship exists. Relationships that involve more than three entities are referred to as n-ary relationships, where n is the number of entities involved in the relationship.

Cardinality: Cardinality is a rough measure of the number of entities (one or more) that will be related to another entity (or entities)

<table>
<thead>
<tr>
<th>Many to One</th>
<th>Many to Many</th>
<th>One to One</th>
</tr>
</thead>
<tbody>
<tr>
<td>(M to 1 or M:1)</td>
<td>(M to M or M:M)</td>
<td>(1 to 1 or 1:1)</td>
</tr>
<tr>
<td>• Has a degree (cardinality) of one or more in one direction &amp; a degree of one and only one in the other direction. • Are very common. • M:1 relationships that are mandatory in both directions are very rare.</td>
<td>• Has a degree of one or more in both directions. • Are very common. • Are usually optional in both directions, although usually a M:M relationship is optional in one direction.</td>
<td>• Has a degree of one and only one in both directions. • Are rare. • 1:1 relationships that are mandatory in both directions is very rare. • Entities which seem to have a 1:1 relationship may really be the same entity.</td>
</tr>
</tbody>
</table>

3.2.4 Step 04: Design ERD

This the vital steps in the database design in this approaches which can be described in the following process

a. Use of entity set vs attributes

b. Use of entity set vs relationship set
c. Binary vs n-ary relationship
d. Placement of relationship attributes

a. **Use of entity set vs attributes**

In the conventional database design, the use of entity set and attributes is easy to identify. Consider the student entity set, the attributes of student is `student id, name, level, term` etc. If we consider "hobby" is an attribute of a student entity set, we can assign single value for hobby attribute. If "hobby" has many different verities in that case there are two options:

**Option-01**

Consider one attribute for one hobby. If we consider ten different names for hobby for each name, we can consider one attribute and the design is shown as the Figure below:

![Figure 3.3: Fixed column design](image)

The design in the Figure 3.3 has a number of flaws:

**Schema Evaluation:** In this design, the database supports only ten fixed values of hobby. If a new hobby is to be added, it requires to change the database schema. This is simple for small database but difficult for a very large database.

**Null value:** Most of the students will have very few numbers of hobbies; the rest of the attributes will contain null values. These null values will create problem in indexing, searching and many aggregate operation.

**Space:** The database size will be unnecessary high because of these null values.

Let \( n_t \) = total number of attributes

\( n_p \) = average number of values present for each student.

Average number of null value for each tuple = \( n_t - n_p \)

\( N \) = cardinality of the student entity set.

\( S_n \) = space for defining null in the database
Space wastage =

\[ S_p = \frac{S_n (n_t - n_p)}{n_p \sum_{i=1}^{N} \left( \sum_{j=1}^{n_p} S_{ij} \right)} \]

**Option-2:**

Consider hobby as an entity of a many to many relationship between the entity set student and hobby. The design is as Figure 3.4

![Figure 3.4: Many to many relation design](image)

This design requires at least three tables: one for student, one for hobby and other for student-hobby relation. The number of table has been increased three times as the design option-1. This design also have some flaws:

i) Managing tables: This design increased the database overhead. For each table it requires dictionary entities. Physical database space overhead for each table.

ii) Management overhead: The more number of tables, the more in the management complexity in the database. There are more insert, delete, update etc.

iii) Integrity constraint: Enforcing integrity constraints on multiple tables is costly operation. This increase design complexity in the database.

iv) Query complexity: The more the number of tables, the probability of join of tables increases to support quiries. The effect of the joining in the table is the increase in average response time.

**Option-3**

Object-oriented nested relational model: Entity- relational model can be extended to support nested relational model as shown in Figure 3.5
Entity is represented by a rectangle, we have used rounded rectangle to represent nested relational attribute. This is because nested relational is neither a pure entity or an atomic attribute. The conversion of the ERD for the nested relational model into an object-relational structure is shown in Figure 3.6.

![Figure 3.5: Nested relational design](image)

```sql
CREATE TYPE Hobby AS OBJECT
(
    Type NUMBER,
    Hobby VARCHAR2(30)
);

CREATE TABLE Student
(
    Student_id NUMBER(32),
    Name VARCHAR2(30),
    Hobbys Hobby multiset
)
```

**Figure 3.6: SQL scripts for a Nested schema**

In this approach the design complexities has been minimized.

### 3.2.5 Step 05: Convert ERD into table

The output of the conceptual database design in any methodology is a set of relations in either relational or object-relational tables. There are many alternatives of conversion of ERD into tables and database performance on this conversion.
Conversion of 1: Many relationship:

**Option-1:** No Null value is allowed, this option requires three tables to represent the information representing the rule of the enterprise into the database. As for example, a salesman can sell many products and a product is sold by a particular salesman. The ERD representing the above enterprise rule is given in the Figure 3.7:

![Figure 3.7: Sales-Products schema](image)

The rule also says that there may be some salesman not assigning any product to sell yet and also some product that has not been started sell yet, so no sales man has been assigned.

**Relational information**

Three tables:
- Salesman (salesman_id, name)
- Product (product_id, name)
- Salesman product (salesman_id, product_id, date of sale)

**Temporal ERD**

Existing ERD schema do not support temporal component of the database. Consider the entity student in the Figure 3.8. A student is a state of a person in the whole life cycle. The state of the person is:

![Figure 3.8: State of life](image)
Considering all the states, we can aggregate all the states of the entity set a person entity set which is temporal in nature. We can represent a temporal entity set by the rectangle having in the corner.

Conversion of temporal component of ERD in general, no table require for the temporal relationship. In the Figure 3.9, there is 1: Many relationship between person and person-temporal. The conversion of the ERD of Figure 3.9 will form the following relation-schema:

\[
\text{Person} \ (\text{Personid}, \ldots) \quad \text{Person_temporal} (\text{Personid}, \text{From}, \text{To} \ldots)
\]

If we use object-relational database, the transformation the temporal ERD is Figure 3.9 will form a single object-relational table as shown in Figure 3.10

```
Create Type Timestamp
  (From Date, To Date);
Create Type Person
  (Person_id Varchar(10),
   State TimeStamp multiset
  );
Create table person under Person;
```

In this design, same query processing requires un-nesting of Object Relational (OR) table that incurs some cost. There are tradeoffs between the relational transformation and Object relational transformation.

**Rule:** The decision support queries are temporal in nature and OR schema is efficient.

**Rule:** The queries of a particular user group is static in nature and relational schema is efficient.
3.2.6 Step-06: Normalization

Normalization is the process of reducing the size of a relation by removing the repetition of information. The scale of normalization is given in the Figure 3.11.

![Normalization scale](image)

**Figure 3.11: Normalization scale**

The gain in normalizing the database from first normal form (1NF) to second normal form (2NF) and from 2NF to third normal form (3NF) is not the same. At the same time, this is not same for 3NF to BCNF. The higher normal form of the database, the less is the increment in gain. From the above fact, it is important to identify the database schema where to stop the normalization process.

**Tuning**

The architecture of a database consists of three distinct but inter related levels: conceptual, internal and external. The database operates at all three levels and may show poor performance due to deficiency at one or more of these levels. Figure 3.12 shows the different levels.

![Mapping among different levels in a database system](image)

**Figure 3.12: Mapping among different levels in a database system**

In these three levels, the external level is mostly dependent on the client machine and network speed in an n-tire database application. So there is very few option to improve performance in the database level.
The internal level is dependent on the implementation of the Database Management System and here is also limited scope to improve performance through tuning. But this improvement is easy and most database application tunes the database by adjusting parameters in this level.

3.2.7 Step 07: Database tuning by changing parameters

Database performance can be improved with the tune of database parameters. There are number of parameters exists in any standard database. We can concentrate on the parameters of the database which are used to manage memory, disk I/O, storage parameters, SQL parsing memory area. Based on the above factors we will concentrate on the following parameters:

- Library Cache: stores shared SQL and PL/SQL areas.
- Data dictionary cache: keeps information about dictionary objects.
- User Global Area: keeps information about the multithreaded connections.
- Open Cursors: defines the number of cursors referencing private SQL areas allocated to the user’s process.
- Buffer Cache: database server reads data into buffer cache from data file disks.
- Database storage parameters: depends upon these parameters database server extent the database files to store information into the database.
- Packing factor: these parameters are used control the use free space within the all data blocks of a segment.

3.2.8 Step 08: Fully normalized database

A fully normalized database is a database that support a particular normal form. A database supporting third normal form (3NF) is called 3NF database. If a database supports Boyce-Codd normal form or forth normal form it can be called BCNF database or 4NF database. Which normal form we have to use depends on the enforcement of integrity constraint of the database. Most business rules and constraints can be implements in a 3NF database. In exceptional case it may require to transform a database into BCNF or 4NF.
3.2.9 Step 09: De-normalized as required

The process of taking a normalized schema and making it non-normalized is called de-normalization. Normalized schema does not suffer from repetition of information. But a de-normalized schema repetition of information is allowed intentionally to improve query performance. Now the question is when to de-normalize, which part of the schema to de-normalize. In an application, the number of SQL statements are limited and we can analyze the application program to find the statistics of joining the tables according to SQL used.

Table 3.1: Frequency of SQL statements in an application

<table>
<thead>
<tr>
<th>Query</th>
<th>Frequency</th>
<th>Wt</th>
<th>P(Table-1)</th>
<th>P(Table-2)</th>
<th>......</th>
<th>P(Table-m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q₁</td>
<td>f₁</td>
<td>w₁</td>
<td>0</td>
<td>1</td>
<td>0.....0</td>
<td>1</td>
</tr>
<tr>
<td>Q₂</td>
<td>f₂</td>
<td>w₁</td>
<td>1</td>
<td>0</td>
<td>1.....0</td>
<td>0</td>
</tr>
<tr>
<td>......</td>
<td>....</td>
<td>.....</td>
<td>......</td>
<td>.....</td>
<td>......</td>
<td>.....</td>
</tr>
<tr>
<td>Qₙ</td>
<td>fₙ</td>
<td>wₙ</td>
<td>0</td>
<td>1</td>
<td>1.....0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.1 shows the frequency count, weight of different queries and the involvement in the queries for different tables. This information can be obtained by scanning the entire application software. The first column represents individual SQL statements embedded in the application. An SQL statement has been considered unique based on the unique set of tables joining in the query. The second column is the frequency count of each statement (query) through out the application. We have assigned a weighting factor to reflect the importance of the query.

The remaining columns represents the tables in the database. If two tables eg, Table₂ and Tableₙ+3 are involved in query Q₁, then a “1” is placed in these columns. All remaining table columns are “0”. In this way, we can complete the table for Qₙ number of distinct queries. After completion of (m+3) X n frequency table matrix, we can apply the modified data mining algorithm of finding large item sets in a table.

\[
\text{Weight (Table-1)} = \sum_{i=1}^{n} f_i w_i P_{(Table-1)i}
\]
\[ \text{Weight (Table-2)} = \sum_{i=1}^{n} f_i w_i P_{(\text{Table-2})i} \]

\[ \text{..................................} \]

\[ \text{Weight (Table-m)} = \sum_{i=1}^{n} f_i w_i P_{(\text{Table-n})i} \]

\[ \text{Min (support)} = \frac{\text{Weight (Table-1)} + \text{Weight (Table-2)} + \ldots + \text{Weight (Table m)}}{m} \]

**Assigning weight**

We have considered that the higher the frequency of the query in the system, the higher the weight.

\[ \text{Weight} = \log_k (f+1), \quad K \text{ will be either 2 or 10. If the difference between minimum and maximum frequency is using high, } K \text{ will be 10. Otherwise it will be 2.} \]

To find the frequency of queries, the entire software code is needed to be scanned. There are \( m + 3 \) number of columns in the matrix where \( m \) is the number tables involved in the queries. The table information is found directly from the database.

The algorithms shown in the Figure 3.13 is used to complete the frequency matrix.
**Algorithms**

*Algorithms ParseSoftware (Software)*

```
Input: Text file of the software;
Output: Query Frequency Matrix(m,n+3);
m = Total number of the tables in schema;
n = 1;
Do while not eof ( ){
    Qn = findNextQuery( );
    SearchExistingQuery (Qn)
    If found (){
        FrequencyCount (Qn) + ;
    } else{
        n = n +1;
        insertlist (Qn)
    }
```

**Figure 3.13: Algorithms for filling frequency matrix**

**Finding Associativity of Tables**

After completion of the large item sets, Apriori algorithms[32] can be applied to find associativity of two tables. Associativity is measured by the co-occurrence of tables in same query. If Table1, is present in a query and Table2 is also present in the same query, then Table1 and Table2 are associative. The strength of this association is measured by how many times in different queries the tables are present. This is the measure of confidence of the associativity. If the confidence value is the higher than certain predefined value, the two tables are candidate for denormalization. We can see the algorithm for two tables and than three tables and so on until no more association possible with sufficient confidence value. The association of tables can be formally defined as

**Table1 => Table2**

means that if Table2 is present in a query, Table1 must be there in the same query.

Similarly,

**{Table1, Table2} => Table3**

means that if Table3 is present in a query, Table1 and Table2 are also present. In this way, all associativity of tables are found. Now the designer can observe the effect of
denormalization and De-Normalize (Table1, Table2) => Table3) or De-Normalize (Table1 => Table2).

**3.2.10 Step 10: Schema Tunes**

The conceptual level is related to the database application and a logical mapping of the real world into a conceptual database schema. There are many alternative options to represent the conceptual schema for the same information system. The scope of tuning is wide in this level. One optimization in this level is normalization and de-normalization that has already been given in the previous section.

Another optimizations are partitioning large tables or replication of relation in distributed database.

**Partitioning**

The main question in partitioning is when and how to partition the database.

Consider the customer product logical schema as in Figure 3.14

![Customer-Product conceptual schema](image)

**Figure 3.14: Customer-Product conceptual schema**

```sql
Customer_schema
( customer_id, name, division, district, thana/city, houseno, street no)
Product_schema ( product_id, group , title)
Sales (Customer_id, product_id ,date , quantity)
```

**Figure: 3.15: SQL scripts for relational schema**

The corresponding relational schema for the E-R schema is given in Figure 3.15. There are a number of ways of portioning the schema in conceptual level. Figure 3.16 shows the different ways of partitioning.
Figure 3.16(a) represents the portioning of the relation table (sales) only. If the entity tables are not very large and relational table is a transactional table, then this portioning is useful.
Partition can be done for both relation table and entity table as in Figure 3.16(b) and Figure 3.16 (c) if the entity tables are very large.

Figure 3.16: Partitioning of Customer- Product conceptual schema
3.3 Design Approaches 2: ER-Relational-Universal approach

Figure 3.17: Database design universal design

In this approaches the relational table is combined in to a flat table to reduce join operation in the query processing. The disadvantage of this approaches is data redundancy.

To design database in this approach we have to follow step 1 and step 2 in the above relational approach and then convert entity set and relation to a flat table.

The ER schema of the Land Information System (LIS) is given in figure 3.18.

Figure 3.18: ER diagram of Land Information System (LIS)
The universal schema is described in Figure 3.19

| KhatianMain=  (ID,Circle_Code, JLNO,MouzaName, KhatianNO,RNO, Revenue_Taka, Revenue_Pisa, IsMut, Total Area,CaseNo, Jot_No, Approval_Date, InherentType,DagNo,LandUse,Total_plot_area, Part_Dag_Kh, LUCode, LUDesc, Land use area,OwnerSiNO, OwnerName, OwnerGrType, OwnerFname, OwnerAddress, OwnerPart) |

**Figure 3.19:** Universal schema of Land Information System (LIS)

*The spatial data cannot be converted into this model.*
3.4  Design Approaches 3: ER-Relational-Binary Approach

This approaches is opposite of the universal model. From relational approach the table are in fully normalized from every field is converted in to tables contains only key and value. In the binary relational model tables contain the information exactly user need, the query execute after join the tables. The advantage of the binary relational model is minimum redundant information (only Key values are redundant) so require less storages but need join operation for every query.

After followed relational step 1 to step 5 then converted into following binary tables for LIS.

1.  $ncircle\_code=(id,circle\_code)$
2.  $njlno=(id,jlno)$
3.  $nmouzaname=(id,mouzaname)$
4.  $nkhatianno=(id,khatianno)$
5. nmo=( id , rno )
6. nrevenue_taka=( id , revenue_taka )
7. nrevenue_pisa=( id , revenue_pisa )
8. nismut=( id , ismut )
9. nkhserial=( id , khserial )
10. nentrydate=( id , entrydate date )
11. ntotal_area_acre=( id , total_area_acre )
12. ntotal_area_thousand=( id , total_area_thousand )
13. ncaseno=( id , caseno )
14. njot_no=( id , jot_no )
15. napproval_date=( id , approval_date date )
16. nsp_marks=( id , sp_marks )
17. ninherenttype=( id , inherenttype )
18. ndagno=( id , dagno )
19. ndagslno=( id , dagslno )
20. nlandclass=( id , landclass )
21. nlanduse=( id , landuse )
22. ntotal_dag_acre=( id , total_dag_acre )
23. ntotal_dag_thousand=( id , total_dag_thousand )
24. npart_dag_kh=( id , part_dag_kh )
25. nland_dag_acre=( id , land_dag_acre )
26. nland_dag_thousand=( id , land_dag_thousand )
27. nlucode=( id , lucode )
28. nludesc=( id , ludesc )
29. narea_acre=( id , area_acre )
30. narea_thousand=( id , area_thousand )
31. nownerslno=( id , ownerslno )
32. nownername=( id , ownername )
33. nownergrtype=( id , ownergrtype)
34. nownerfname=( id , ownerfname )
35. nowneraddress=( id , owneraddress )
36. nownerpart=( id , ownerpart )
37. nremarks=( id , remarks )

*The spatial data cannot be converted into this model.*
3.5 Design Approaches 4: Object-orientated to Object Relational Database

3.5.1 Selection of Objects

To identify and selection of potential objects is the primary task of object relational design. Many experts suggest searching the requirements and other associated documentation and underline the nouns that may represent potential objects. This could be a monumental task, there may be too many nouns. Use-case modeling provides the solution to this problem by breaking down the entire scope of the system into use cases. To find the potential objects review each use case to find nouns that correspond to business entities or events. We have to make a list of the potential objects, not all the candidates (nouns) on our list useful business objects. By analyzing each candidate of the following points we can determine whether the candidate should stay or be removed from the list:

- Is candidate synonym of another object
- Is candidate outside the scope of the system?
In Object-oriented analysis identified the objects that represent actual data within the business domain. These business domains are called \textit{entity objects}. Entity objects are corresponds to the real life objects and contain information called \textit{attributes} that describes the different instance of the entity. Users interact with system with some other objects which is called \textit{interface} objects. Interface objects translates the user’s input into information that system can understand and use to process the business event. It also translates the system’s information and business event into the appropriate presentation to the user. The objects hold application business rule and logic called \textit{control} objects.

First we have to identify and classify the objects to complete the use case design. We may list out the objects in three separate columns in a table to look at a glance. From our experimental setup the Land Information System (LIS) objects can identified as following Table 3.2:

\begin{table}[h]
\centering
\begin{tabular}{|l|l|l|}
\hline
Interface objects & Control objects & Entity objects \\
\hline
w01: Khatian entry form & Mutation case approved handler & Plot information \\
\hline
w02: Mutation Application form & & Owner name \\
\hline
w03: Land verification reports & & Land use \\
\hline
w09: Plot map display & & Land type \\
\hline
w10: land index reports & & Circle name \\
\hline
\end{tabular}
\caption{Objects table for LIS}
\end{table}

Interface objects column contains a list of objects that users directly interface with screens, windows, card readers, and printers. The only way an actor or user interface with a system is via an interface object. The control objects column contains a list of objects that encapsulate the application logic and business rules. The entity objects
contains a list of business domain objects whose attributes referenced in the use case
design.

3.5.2 Identify of objects attributes
We need to identify what specific pieces of data we want to store about each instant
of a given entity. We call these pieces of data as attributes. When analyzing a system,
we should define those values for an attribute that are legitimate or that make
business sense. The values of each attributes are defined in terms of three properties:
data type, domain, default. The data type for an attributes defined what type of data
can be stored in that attribute. For purpose of system analysis and business
requirements definition, it is useful to declare logical data types for business
attributes. An attribute data type constrains its domain. The domain of an attributes
defines what values the attributes can legitimately take on. Every attribute should
have a logical default value that represents the value of an attribute if its value is not
specified by the user. There exists a need to uniquely identify each instant based on
the data value one-or more attributes. Thus every entity must have a key to identify.

3.5.3 Relationship identification
Conceptually entities and attributes do not exist in isolation. The things they
represent interact with and impact one another to support the business mission. A
relationship is a natural business association that exists between one or more entities.
The relationship may represent an event that links the entities or merely a logical
affinity that exists between the entities objects.

Objects can related to each other through different relationships:
• Association (delegation)
• Generalization (inheritance)
• Realization (interfaces)
• Dependency

Association
Association describes a link, a link being a connection among objects between
classes. Association is shown by a solid line between classes. The following Figure
shows an example:
Association have five properties like name, role, multiplicity, type and direction. A class diagram is used to graphically depict the objects and association. On this diagram we will also include multiplicity, generalization/specialization relationships and aggregation relationships. To identify the association that exists between objects and class we have to know the cross-reference of another object or class. Once association is identified, the multiplicity that governs the association must be identified. A multiplicity in a class specifies the number of instances (Objects) of that class can exists simultaneously. Each association may have two roles: one in each direction, from source to target, roles have multiplicity - how many objects participate in relationship.

Generalization

Generalization/specialization relationships known as classification hierarchies or "is a" relationships, consist of super type classes and sub-type classes. The super type class is specialized in that in contains the common attributes and behaviors of the
hierarchy. The sub type class is specialized is that contains attributes and behaviors unique to the object but it inherits the super type class's attributes and behavior.

Figure 3.24: Association and generalization

Aggregation and Composition

Aggregation is unique type of relationship is which one object “is part of” another object. It is also referred to as a “whole/part relationship” and can be read as “Object A” contains “Object B” and object B is part of Object A. Composition is also a part of relationship but part and whole live and die together.
3.5.4 O-R Database design for LIS

To design in this approaches we have followed steps 1 to steps 3 then we implements the table object as following diagram:

![UML diagram for CLMS database]

Figure 3.26: UML diagram for CLMS database
Object relational model of spatial data

With object relational model of oracle Spatial, the geometric description of a spatial object is stored in a single row, in a single column of object type SDO_GEOMETRY in a user-defined table. Any table that has a column of type SDO_GEOMETRY must have another column, or set of columns, that defines a unique primary key for that table. Tables of this sort are sometimes referred to as spatial tables or spatial geometry tables.

Oracle Spatial defines the object type SDO_GEOMETRY as:

```
Type SDO_GEOMETRY AS OBJECT (sdo_gtype, sdo_srid, sdo_point SDO_POINT_TYPE, sdo_elem_info SDO_ELEM_INFO_ARRAY, sdo_ordinates SDO_ORDINATE_ARRAY);
```

Oracle Spatial also defines the SDO_POINT_TYPE, SDO_ELEM_INFO_ARRAY, and SDO_ORDINATE_ARRAY types, which are used in the SDO_GEOMETRY type definition, as follows:

```
TYPE SDO_POINT_TYPE AS OBJECT (XNUMBER, Y NUMBER, Z NUMBER);
TYPE SDO_ELEM_INFO_ARRAY AS VARRAY (1048576) of NUMBER;
TYPE SDO_ORDINATE_ARRAY AS VARRAY (1048576) of NUMBER;
```

I. Polts GIS data to the Object relation data

plots = (plot_no, jl_no, mauza, thana, lucode, area, geom mdsys.sdo_geometry)

2. Feature GIS data to the Object relation data

features = (id, plot_no, jl_no, lucode, land_use, geom mdsys.sdo_geometry)

3. Road GIS data to Object relational model

roads = (length, ltype, road_name, thana, sub_area, ward, area_name, road_type, geom mdsys.sdo_geometry)
3.6 Design Approaches 5: Nested relational model

![Diagram](image)

**Figure 3.27**: Nested relation design

Figure 3.27 shows the database design using nested relation model. In the classical Entity-Relationship-Model aggregation and composition are modeled through master-detail-relationships. Object relational database management systems provide collection type that can contain multiple elements at a time and thus are suitable to express 1:n relationship directly. Each element or value for a collection has the same substitutable data type. The most popular collection types are nested tables. A nested table is a table that is embedded in another table. It is possible to define a table data type and to use this type as a column type in a table. So, this column contains a table (called

<table>
<thead>
<tr>
<th>column1</th>
<th>column2</th>
<th>C Table</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>c'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c''</td>
</tr>
</tbody>
</table>
nested table) with a collection of values, objects or references. A nested table is a
collection type but it is also a table. Being a table, it can be defined as an object table.
So, the nested table can contain the parts as objects rather than as references. At the
same time the nested table is embedded in a column of another object table (the
whole). The following Figure shows the specification of a composition

```
Create Type Line as OBJECT
(Line_id NUMBER)

Create Type Lista_Lines as Table of Line

Create Type Ploygon as Object
(Polygon_id NUMBER, Lines_number NUMBER, Tine_Lines Lista_Lines)

Create Table T_Polygon OF Polygon
( Primary key (Polygon_Id))
NESTED TABLE Has_Lines STORE AS
Table_Lines
```

Figure 3.29 Nested presentation of spatial data
Chapter 4
Results and Discussion

4.1 Performance analysis of Relational, Universal and Nested-relational model

We have considered the student entity set, the attributes of student is student id, name, and hobby. If we consider “hobby” is an attribute of a student entity set, we can assign a single value for the hobby attribute. If “hobby” has many different varieties in that case there are three options:

a) **Option-1**: Fixed column design.
b) **Option-2**: Using many to many relationship.
c) **Option-3**: Using nested relational model.

**Option-1: Fixed column design, One attribute for a hobby**

![Fixed Column Design (Design Option -1)](image)

*Figure 4.1: Fixed Column Design (Design Option -1)*

This design is referred to as universal or flat table design schema. The conversion of ERD to universal design is shown in following Figure 4.2:
CREATE TABLE tblUniversal (ID NUMBER(32), NAME VARCHAR(30), Adventure CHAR, Boating CHAR, Camping CHAR, Diving CHAR, Fishing CHAR, Flying CHAR, Aquariums CHAR, Beekeeping CHAR, Cat_Lovers CHAR, Horses CHAR, Arts CHAR, Drawing CHAR, Musical CHAR, Instruments CHAR, Writing CHAR, Crafts CHAR, Glassworking CHAR, Needlework CHAR, Embroidery CHAR, Knitting CHAR, Sewing CHAR, Quilting CHAR, Pottery CHAR, Sculpture CHAR, Woodworking CHAR, Antique CHAR, Toys CHAR, Artwork CHAR, Comic CHAR, Books CHAR, Collecting CHAR, Vintage CHAR, Paperbacks CHAR, Recommended CHAR, Card CHAR, Chess CHAR, Poker CHAR, Puzzles CHAR, Crossword CHAR, Cooking CHAR, Gardening CHAR, Magic CHAR, Model CHAR, Building CHAR, Archeology CHAR, Astronomy CHAR, Watching CHAR, Gadgets CHAR, Robots CHAR, Sports CHAR);

Figure 4.2: Convert ERD to universal relational schema

Option-2: Many-many relationship

![Diagram of Many-many relationship](image)

Figure 4.3: Many-many relational design (Design option -2)

This design requires at least three tables: one for student, one for hobby and other for student-hobby relation. This design is referred to as Relational design schema. The corresponding ERD is shown in the following Figure 4.4 for design option-2:

Create Table Student (ID NUMBER(32), Name VARCHAR2(30));
Create Table Hobby (HID NUMBER(2), Hobby VARCHAR2(30));
Create Table Student_Hobby (ID Number(32), HID Number(2));

Figure 4.4: Convert ERD to Relational schema
Option-3: Nested relational model

Figure 4.5 shows the nested relation design for student schema, the hobby attributes are in a nested table.

```
Create type HobbyTyp as Object (HID Number, HobbyVarchar2(30));
Create type nt_Hobby as Table OF HobbyTyp;
Create Table ntPeople
( ID NUMBER(32), Name Varchar2(30), Hobbys nt_Hobby )
NESTED TABLE Hobbys STORE as nt_Hobby_tab;
```

Database size analysis

To measure database size the records are populated from 1 million to 10 million and five samples were taken. The following Table 4.1 shows the snapshot of database size in different state of three design schema:

<table>
<thead>
<tr>
<th>Record Number</th>
<th>Size of the database in MB</th>
<th>Nested</th>
<th>Relational</th>
<th>Universal</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000</td>
<td>5</td>
<td>12</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>200000</td>
<td>9</td>
<td>24</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>500000</td>
<td>21</td>
<td>58</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>1000000</td>
<td>41</td>
<td>114</td>
<td>136</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.5: Nested relational design (Design option-3)

Figure 4.6: Convert ERD to Nested Relational
Figure 4.7: Database size vs. record (nos)

Figure 4.7 shows the database size in respect of number of records in the database. The variation of size is not linear for three schemas. The nested design schema table size is always much smaller than other two database design. The relational is middle and universal one is much higher all the time. In the stating of the sample the difference between three schema is not significant but when the table records is increasing the difference also increase and at ten million records the relational database size almost three times then nested relational model and universal relation much higher then relational schema. The reason behind this is that the universal schema contains many null values for hobby attributes, on the other hand nested relation has more compact structure with no null values. The relational schema is much higher then nested because of it contain more tables for its relation. So in terms of database size nested relational model is much better then other two schema.

Response time analysis:

The response time is measured for three schema of database using three types of queries: point query, aggregate query and range query.

Point Query

The following table shows the response time for point query in different database schema.
Table 4.2: Response time for point query

<table>
<thead>
<tr>
<th>Number of Records</th>
<th>Query Type</th>
<th>Response time in milli second</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Relational</td>
</tr>
<tr>
<td>100000</td>
<td>Point Query</td>
<td>170</td>
</tr>
<tr>
<td>200000</td>
<td>Point Query</td>
<td>230</td>
</tr>
<tr>
<td>500000</td>
<td>Point Query</td>
<td>400</td>
</tr>
<tr>
<td>1000000</td>
<td>Point Query</td>
<td>420</td>
</tr>
</tbody>
</table>

Figure 4.8: Response time for point query

The Figure 4.8 shows the response time with variation for records in the database. Universal design always shows better performance. The nested relational model has higher response time than other two schemas and the difference is increasing with the increasing number of records. This is due to nested relational model having to unnest and join with the internal nested tables. Joining is one of the most costly operations during query processing time. The Figure 4.9 shows the SQL script that were used to measure the response time for three schemas:

```
Query 01: Point_Relational
SELECT p.ID, p.Name, h.Hobby FROM People p,
    People_Hobby p_h, Hobby h WHERE p_ID = p_h.ID and p_h.HID = h.HID and p.ID = 1239;
Query 02: Point_universal
SELECT * FROM tblUni WHERE id = 1239;
Query 03: Point_nested
SELECT * FROM ntPeople WHERE id = 1239;
```

Figure 4.9: SQL scripts for point query
Aggregate query

The Table 4.3 shows the query response time for aggregate query in three databases schema

Table 4.3: Response time for Aggregate Query

<table>
<thead>
<tr>
<th>SL no</th>
<th>Query Type</th>
<th>Record Number</th>
<th>Relational</th>
<th>Universal</th>
<th>Nested</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Aggregate Query</td>
<td>100000</td>
<td>3.75</td>
<td>6.61</td>
<td>10.53</td>
</tr>
<tr>
<td>2</td>
<td>Aggregate Query</td>
<td>200000</td>
<td>3.75</td>
<td>6.61</td>
<td>10.53</td>
</tr>
<tr>
<td>3</td>
<td>Aggregate Query</td>
<td>500000</td>
<td>18.45</td>
<td>4.42</td>
<td>43.84</td>
</tr>
<tr>
<td>4</td>
<td>Aggregate Query</td>
<td>1000000</td>
<td>37.04</td>
<td>7.76</td>
<td>117.4</td>
</tr>
</tbody>
</table>

The Figure 4.10 shows the graphical representation of query response time for aggregate query in three databases schemas. The response time for small number of records is almost same level for all three schema but the difference is increasing with the number of records. The response time for aggregate query in universal database schema shows almost static with varying of the number of records. The variation may occur for higher data range. For the relational database, this is static in initial two samples after that it increases linearly with respect to number of records. For the nested relational, the response time is much higher then the other two databases schemas, and the gap is increasing much for higher number of records. The database server has to perform grouping after the aggregate functions are computed. For the universal database schema, all data are in a single table, so it's takes less time compared to others to compute aggregation operation. In the case of relational...
database schema the database server first do joining operation then do grouping and then aggregate function is computed, that's why, the response time is increasing linearly after certain number of records. For the nested database schema, the un-nest and internal joining operation is completed then compute aggregation function. For this reason it takes more time compared to other design schema.

The Figure 4.11 shows the SQL scripts for aggregate query:

<table>
<thead>
<tr>
<th>Query 01: Aggregate Query for Relational Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select h.Hobby, count(p.ID) as cnt From People p, People_Hobby p_h, Hobby h</td>
</tr>
<tr>
<td>Where p.ID = p_h.ID and p_h.HID = h.HID Group by h.Hobby;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query 02: Aggregate Query for Universal Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT sum(decode(Adventure, 'I', I)) as Adventure, sum(decode(Boating, 'I', I)) as Boating,</td>
</tr>
<tr>
<td>sum(decode(Camping, 'I', I)) as Camping, sum(decode(Diving, 'I', I)) as Diving,</td>
</tr>
<tr>
<td>sum(decode(Fishing, 'I', I)) as Fishing, sum(decode(Flying, 'I', I)) as Flying,</td>
</tr>
<tr>
<td>sum(decode(Aquariums, 'I', I)) as Aquariums, sum(decode(Beekeeping, 'I', I)) as Beekeeping,</td>
</tr>
<tr>
<td>sum(decode(Cat_Lovers, 'I', I)) as Cat_Lovers, sum(decode(Horses, 'I', I)) as Horses,</td>
</tr>
<tr>
<td>sum(decode(Chiefs, 'I', I)) as Chiefs, sum(decode(Glassworking, 'I', I)) as Glassworking,</td>
</tr>
<tr>
<td>sum(decode(Needlework, 'I', I)) as Needlework, sum(decode(Embroidery, 'I', I)) as Embroidery,</td>
</tr>
<tr>
<td>sum(decode(Quilting, 'I', I)) as Quilting, sum(decode(Sewing, 'I', I)) as Sewing,</td>
</tr>
<tr>
<td>sum(decode(Quality, 'I', I)) as Quality, sum(decode(Pottery, 'I', I)) as Pottery,</td>
</tr>
<tr>
<td>sum(decode(Sculpture, 'I', I)) as Sculpture, sum(decode(Woodworking, 'I', I)) as Woodworking,</td>
</tr>
<tr>
<td>sum(decode(Anthique, 'I', I)) as Anthique, sum(decode(Toys, 'I', I)) as Toys,</td>
</tr>
<tr>
<td>sum(decode(Artwork, 'I', I)) as Artwork, sum(decode(Comic, 'I', I)) as Comic,</td>
</tr>
<tr>
<td>sum(decode(Books, 'I', I)) as Books, sum(decode(Collecting, 'I', I)) as Collecting,</td>
</tr>
<tr>
<td>sum(decode(Vintage, 'I', I)) as Vintage, sum(decode(Paperbacks, 'I', I)) as Paperbacks,</td>
</tr>
<tr>
<td>sum(decode(Recommended, 'I', I)) as Recommended, sum(decode(Card, 'I', I)) as Card,</td>
</tr>
<tr>
<td>sum(decode(Chess, 'I', I)) as Chess, sum(decode(Poker, 'I', I)) as Poker, sum(decode(Puzzles, 'I', I)) as Puzzles,</td>
</tr>
<tr>
<td>sum(decode(Crossword, 'I', I)) as Crossword, sum(decode(Cooking, 'I', I)) as Cooking,</td>
</tr>
<tr>
<td>sum(decode(Gardening, 'I', I)) as Gardening, sum(decode(Magic, 'I', I)) as Magic,</td>
</tr>
<tr>
<td>sum(decode(Model, 'I', I)) as Model, sum(decode(Building, 'I', I)) as Building,</td>
</tr>
<tr>
<td>sum(decode(Archeology, 'I', I)) as Archeology, sum(decode(Astronomy, 'I', I)) as Astronomy,</td>
</tr>
<tr>
<td>sum(decode(Watching, 'I', I)) as Watching, sum(decode(Gadgets, 'I', I)) as Gadgets,</td>
</tr>
<tr>
<td>sum(decode(Robots, 'I', I)) as Robots, sum(decode(Sports, 'I', I)) as Sports FROM tblUni;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query 03: Aggregate query for nested relational</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select h.hobby, count(p.ID) as cnt from nPeople p, TABLE(Hobbys) h Group by h.hobby</td>
</tr>
</tbody>
</table>

Figure 4.11: SQL scripts for aggregate query
Range Query

The response time is described in the following Table 4.4. The graphical representation of response time is given in the figure 4.12 and the corresponding SQL script is given in 4.13.

Table 4.4: Response time for range query

<table>
<thead>
<tr>
<th>ID NO</th>
<th>Query Type</th>
<th>Record Number</th>
<th>Universal</th>
<th>Relational</th>
<th>Nested</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Range Query</td>
<td>100000</td>
<td>0.79</td>
<td>1.75</td>
<td>2.78</td>
</tr>
<tr>
<td>2</td>
<td>Range Query</td>
<td>200000</td>
<td>0.79</td>
<td>1.75</td>
<td>2.78</td>
</tr>
<tr>
<td>3</td>
<td>Range Query</td>
<td>500000</td>
<td>2.09</td>
<td>4.42</td>
<td>12.3</td>
</tr>
<tr>
<td>4</td>
<td>Range Query</td>
<td>1000000</td>
<td>3.76</td>
<td>15.56</td>
<td>27.51</td>
</tr>
</tbody>
</table>

Range Query: Response time vs. Number of records

Figure 4.12: Response time analysis for range query
4.2 Normalization and space gain

**Experimental Setup**

We have consider the Books entity set, the attributes of Book is CallNO, Authername, AuthorAddress, PublisherName, Publisher_address, price, ISBN, Edition, BookTitle etc. If we consider “CallNO” is the primary key of the entity set and the every attributes are atomic then we get 1NF of this entity set as following Figure:

1NF

![1NF from of entity Book](Image)

The following sql script was used to convert 1NF entity set to the table.
Create table BOOK1N (CalINo Char (10), BookTitle Varchar2 (100), PublisherName Varchar2(100), PublisherAddress Varchar2(255), Price Number(4), ISBN Char(10), Edition Varchar2(5), AuthorName Varchar2(100), AuthorAddress Varchar2(255));

Figure 4.15 SQL script to convert 1NF Book entity

In the above entity set Author name depends upon Callno and Author Address depends upon the Author Name which is partial dependency. To resolve this dependency we have spited the entity set into two different entity set as shown in the following Figure:

2NF

Book 2NF (Callno, PublishersName, Publisher_address, price, ISBN, Edition,BookTitle)

Author 2NF (Callno, AuthorName, AuthorAddress)

Figure 4.16 2NF form of entity Book

The following sql scripts is used to convert 2NF entity set

Create table BOOK2N
(
CallNo Char (10), BookTitle Varchar2 (100),
PublisherName Varchar2(100), PublisherAddress Varchar2(255),
Price Number(4), ISBN Varchar2(10),
Edition Varchar2(5)
);
Create table Author2N
(
CallNo Char(10), AuthorName Varchar2(100),
AuthorAddress Varchar2(255)
);

Figure 4.17: SQL Scripts to convert the 2NF form

In the above Figure there is also transitive dependency exist, Publishers_address depends on the attribute PublishersName. So we have to split the above Book2N in to
two entity set one is Book3NF contains attribute callno, publishername, price, ISBN, Edition. BookTitle and other entity set is Publisher3NF contain PublisherName and PublisherAddress.

The entity sets are as following Figures:

**3NF**

- **Book 3NF (Callno, PublisherName, price, ISBN, Edition, BookTitle)**
- **Author 3NF (Callno, AuthorName, AuthorAddress)**
- **Publisher 3NF (PublisherName, Publisher_address)**

**Figure 4.18:** 3NF form of entity Book

The Following SQL scripts was used to convert the 3NF form into tables

```sql
Create table BOOK3N
(
    CallNo Char (10), BookTitle Varchar2 (100),
    Price Number (4), ISBN Char(10),
    Edition Varchar2(5), PublisherName Varchar2(100)
);
Create table Author3N
(
    Callno char(10), AuthorName Varchar2(100),
    AuthorAddress Varchar2(255)
);
Create Table Publisher3N
(
    PublisherName Varchar2(100), Publisher_Address Varchar2(255)
);
```

**Figure 4.19:** SQL scripts to convert 3NF schema
Database storage analysis

Data is populating using a customized program and insert data in the three normal forms, we get Table 4.5 and bar chart in Figure 4.20 to describe the situation.

Table 4.5: Database size with varied number of records

<table>
<thead>
<tr>
<th>SL NO</th>
<th>Number of records</th>
<th>Table Size in MB</th>
<th>1NF</th>
<th>2ndNF</th>
<th>3rdNF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10000</td>
<td></td>
<td>15</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>100000</td>
<td></td>
<td>120</td>
<td>57</td>
<td>56</td>
</tr>
<tr>
<td>3</td>
<td>200000</td>
<td></td>
<td>229</td>
<td>110</td>
<td>82</td>
</tr>
<tr>
<td>4</td>
<td>500000</td>
<td></td>
<td>493</td>
<td>272</td>
<td>264</td>
</tr>
</tbody>
</table>

Figure 4.20: database size vs. number of records

The normalization process reduces the size of the relation by removing the repetition from entity set. The space gain from first normal form (1NF) to second normal form (2NF) and 2NF to third normal form (3NF) is not linear. The database size increases with the increase of record in the database. The rate of database size increase is not linear. Another thing is that space gain from 1NF to 2NF is not same as from 2NF to 3NF.
4.3 Performance Tuning and De-normalization as per required

We have considered a real life example, Computerization Land Information System (LIS) for this experiment. This is a project taken by Ministry of Land in 2006. The main objective to develop this system is to provide update land related records to citizen.

4.3.1 Find the tables for de-normalization

![Entity Relationship Diagram]

Figure 4.21: Entity relationship diagram

Figure 4.21 shows the entity relationship diagram of the LIS database. First, we have to identify the tables where we have to apply de-normalization process to the database. To the schema where we have to apply de-normalization, we have analyzed the whole software and identified the SQL statements. We have considered those SQL statements which have join in more than one tables. We have fill up the following frequency table (Table 4.6) set the values to “1” which tables are joined in a query and other values are “0”.

```
Figure 4.21: Entity relationship diagram
```
Table 4.6: Query set, frequency and join tables

| SL NO | Query NO | Table NO | Frequency | Wt | tblLook up Circ le | tblLook up Dist rict | tblLook up Than a | tblLook up Mouza | tblLook up TaxTyp e | tblLook up Land use | tblKhat ian | tblKh Dag | tblDa g Owner | tblDa g KhL use | tblAp plica nt | tblCas eG Info | tblDa g LUse | tblDa g Info | tblMu tKhD ag | tblTr ansL andD etails |
|-------|----------|----------|-----------|----|-------------------|---------------------|-------------------|------------------|-------------------|-------------------|------------|----------|---------------|----------------|----------------|----------------|--------------|-------------|-----------|-------------|--------------------------|
| 1     | Q1       | 5        | 2.58      | 1  | 1                 | 1                   | 1                 | 1                | 1                 | 0                 | 0          | 0        | 0             | 0              | 0             | 1             | 0           | 0           | 0           | 0                        |
| 2     | Q2       | 1        | 1.00      | 0  | 0                 | 0                   | 0                 | 0                | 0                 | 1                 | 1          | 1        | 0             | 0              | 0             | 0             | 0           | 0           | 0           | 0                        |
| 3     | Q3       | 3        | 2.00      | 1  | 1                 | 1                   | 1                 | 1                | 0                 | 0                 | 0          | 0        | 0             | 0              | 0             | 0             | 1           | 0           | 0           | 0                        |
| 4     | Q4       | 2        | 1.58      | 0  | 0                 | 0                   | 0                 | 0                | 1                 | 1                 | 1          | 1        | 0             | 0              | 0             | 0             | 0           | 1           | 0           | 0                        |
| 5     | Q5       | 6        | 2.81      | 1  | 1                 | 1                   | 1                 | 1                | 0                 | 0                 | 0          | 0        | 0             | 0              | 0             | 0             | 0           | 0           | 1           | 0                        |
| 6     | Q6       | 3        | 2.00      | 0  | 0                 | 0                   | 0                 | 0                | 1                 | 0                 | 0          | 0        | 0             | 0              | 0             | 0             | 1           | 0           | 0           | 0                        |
| 7     | Q7       | 2        | 1.58      | 0  | 0                 | 0                   | 0                 | 0                | 1                 | 0                 | 1          | 0        | 0             | 0              | 0             | 0             | 1           | 0           | 0           | 0                        |
| 8     | Q8       | 1        | 1.00      | 0  | 0                 | 0                   | 0                 | 0                | 0                 | 0                 | 0          | 1        | 1             | 0              | 0             | 0             | 0           | 0           | 0           | 0                        |
| 9     | Q9       | 1        | 1.00      | 0  | 0                 | 0                   | 0                 | 0                | 1                 | 1                 | 0          | 0        | 0             | 0              | 0             | 0             | 0           | 0           | 0           | 0                        |
| 10    | Q10      | 1        | 1.00      | 0  | 0                 | 0                   | 0                 | 0                | 1                 | 1                 | 0          | 0        | 0             | 0              | 0             | 0             | 0           | 1           | 1           | 0                        |
| 11    | Q11      | 1        | 1.00      | 0  | 0                 | 0                   | 0                 | 0                | 1                 | 0                 | 0          | 0        | 0             | 0              | 0             | 0             | 0           | 0           | 0           | 0                        |
| 12    | Q12      | 2        | 1.58      | 1  | 1                 | 1                   | 1                 | 0                | 1                 | 1                 | 1          | 1        | 0             | 0              | 0             | 0             | 0           | 0           | 0           | 0                        |
| 13    | Q13      | 2        | 1.58      | 1  | 1                 | 1                   | 1                 | 1                | 0                 | 0                 | 0          | 0        | 0             | 0              | 0             | 0             | 0           | 0           | 0           | 0                        |
| 14    | Q14      | 10       | 3.46      | 0  | 0                 | 0                   | 0                 | 1                | 1                 | 0                 | 0          | 0        | 0             | 0              | 0             | 0             | 0           | 0           | 0           | 0                        |
| 15    | Q15      | 3        | 2.00      | 1  | 1                 | 1                   | 1                 | 1                | 0                 | 0                 | 0          | 0        | 0             | 0              | 0             | 0             | 0           | 0           | 0           | 0                        |
| 16    | Q16      | 3        | 2.00      | 1  | 1                 | 1                   | 1                 | 1                | 0                 | 0                 | 0          | 0        | 0             | 0              | 0             | 0             | 0           | 0           | 0           | 0                        |
| 17    | Q17      | 2        | 1.58      | 1  | 1                 | 1                   | 1                 | 0                | 0                 | 0                 | 0          | 0        | 0             | 0              | 0             | 0             | 0           | 0           | 0           | 0                        |
| 18    | Q18      | 1        | 1.00      | 0  | 0                 | 0                   | 0                 | 1                | 1                 | 0                 | 0          | 0        | 0             | 0              | 0             | 0             | 0           | 0           | 0           | 0                        |
| 19    | Q19      | 3        | 2.00      | 1  | 1                 | 1                   | 1                 | 0                | 0                 | 0                 | 0          | 0        | 0             | 0              | 0             | 0             | 0           | 0           | 0           | 0                        |
| 20    | Q20      | 1        | 1.00      | 0  | 0                 | 0                   | 0                 | 1                | 1                 | 1                 | 0          | 0        | 0             | 0              | 0             | 0             | 0           | 1           | 0           | 0                        |
| 21    | Q21      | 3        | 2.00      | 1  | 1                 | 1                   | 1                 | 0                | 1                 | 0                 | 0          | 0        | 0             | 0              | 0             | 0             | 0           | 0           | 0           | 0                        |
| 22    | Q22      | 1        | 1.00      | 0  | 0                 | 0                   | 0                 | 0                | 0                 | 0                 | 0          | 0        | 0             | 0              | 0             | 0             | 0           | 0           | 0           | 0                        |
| SL NO | Query | Frequency | Wt  | tblL ook up Circ le | tblL ook up Dist rict | tblLo okup Than a | tblLo okup Mouz a | tblL ook up TaxTyp e | tblL ookup Land use | tblKhat ian | tblK | tblK owner | tblDa gKhL use | tblAp plicant | tblCas eG Info | tblDa g LUse | tblDa g Info | tblMu tkhD ag | tblTr ansL andD etails |
|-------|-------|-----------|-----|---------------------|-----------------------|-------------------|-------------------|---------------------|---------------------|-------------|------|-----------|--------------|---------------|---------------|----------------|----------------|----------------|--------------|------------------|
| 23    | Q23   | 4         | 2.32| 1                   | 1                     | 1                 | 1                 | 0                   | 1                   | 1            | 0    | 1         | 0             | 0             | 0             | 0             | 0             | 0             | 0               |
| 24    | Q24   | 1         | 1.00| 0                   | 0                     | 0                 | 0                 | 0                   | 1                   | 0            | 1    | 1         | 0             | 0             | 0             | 0             | 0             | 0             | 0               |
| 25    | Q25   | 2         | 1.58| 1                   | 1                     | 1                 | 1                 | 0                   | 0                   | 0            | 0    | 0         | 0             | 0             | 0             | 1             | 0             | 0             | 0               |
| 26    | Q26   | 1         | 1.00| 0                   | 0                     | 0                 | 0                 | 0                   | 1                   | 0            | 1    | 1         | 0             | 0             | 0             | 0             | 1             | 0             | 0               |
| 27    | Q27   | 1         | 1.00| 0                   | 0                     | 0                 | 0                 | 0                   | 1                   | 1            | 0    | 1         | 1             | 0             | 0             | 0             | 1             | 0             | 0               |
| 28    | Q28   | 2         | 1.58| 0                   | 0                     | 0                 | 0                 | 0                   | 1                   | 1            | 1    | 0         | 0             | 0             | 0             | 0             | 0             | 0             | 0               |
| 29    | Q29   | 1         | 1.00| 0                   | 0                     | 0                 | 0                 | 0                   | 1                   | 1            | 1    | 0         | 0             | 0             | 0             | 0             | 0             | 1             | 0               |
| 30    | Q30   | 1         | 1.00| 0                   | 0                     | 0                 | 0                 | 1                   | 1                   | 1            | 0    | 1         | 1             | 0             | 0             | 0             | 0             | 0             | 0               |
| 31    | Q31   | 2         | 1.58| 0                   | 0                     | 0                 | 0                 | 0                   | 1                   | 1            | 1    | 0         | 0             | 0             | 0             | 0             | 0             | 0             | 0               |
| 32    | Q32   | 3         | 2.00| 1                   | 1                     | 1                 | 1                 | 0                   | 1                   | 0            | 0    | 1         | 0             | 0             | 0             | 1             | 0             | 0             | 0               |
| 33    | Q33   | 1         | 1.00| 0                   | 0                     | 0                 | 0                 | 0                   | 1                   | 1            | 0    | 0         | 0             | 0             | 0             | 0             | 0             | 0             | 0               |
| 34    | Q34   | 2         | 1.58| 0                   | 0                     | 0                 | 0                 | 0                   | 1                   | 1            | 1    | 0         | 0             | 0             | 0             | 0             | 0             | 0             | 0               |
| 35    | Q35   | 1         | 1.00| 1                   | 0                     | 0                 | 1                 | 0                   | 1                   | 1            | 1    | 0         | 0             | 0             | 0             | 0             | 0             | 0             | 0               |
| 36    | Q36   | 1         | 1.00| 0                   | 0                     | 0                 | 0                 | 0                   | 1                   | 1            | 1    | 0         | 0             | 0             | 0             | 0             | 0             | 0             | 0               |
| 37    | Q37   | 2         | 1.58| 1                   | 0                     | 0                 | 0                 | 1                   | 1                   | 1            | 0    | 0         | 0             | 0             | 0             | 0             | 0             | 0             | 0               |
| 38    | Q38   | 1         | 1.00| 0                   | 0                     | 0                 | 0                 | 1                   | 1                   | 1            | 0    | 1         | 0             | 0             | 0             | 0             | 0             | 0             | 0               |
| 39    | Q39   | 4         | 2.32| 0                   | 0                     | 0                 | 0                 | 0                   | 1                   | 1            | 0    | 0         | 0             | 0             | 0             | 0             | 0             | 0             | 0               |
| 40    | Q40   | 2         | 1.00| 1                   | 1                     | 1                 | 1                 | 0                   | 1                   | 1            | 1    | 0         | 0             | 0             | 0             | 0             | 0             | 0             | 0               |
| 41    | Q41   | 3         | 2.00| 1                   | 1                     | 1                 | 1                 | 0                   | 0                   | 0            | 0    | 0         | 0             | 0             | 1             | 1             | 0             | 0             | 0               |
| 42    | Q42   | 3         | 2.00| 0                   | 0                     | 0                 | 0                 | 1                   | 1                   | 0            | 0    | 1         | 0             | 0             | 0             | 0             | 0             | 0             | 0               |
| 43    | Q43   | 3         | 2.00| 1                   | 1                     | 1                 | 1                 | 0                   | 1                   | 0            | 0    | 0         | 0             | 0             | 0             | 0             | 0             | 0             | 0               |

We have calculate the frequency ($f_i$) of a query as the number of times the query is used in the LIS application. The weight of the frequency is calculated using Weight ($w_i$) = $\log_2 (f_i+1)$. The weight of a tables can be calculated using following formula
Weight (Table-n) = \sum_{i=1}^{n} f_i w_i P_{(Table-n)} , The detail of the analysis is given in section 3.2.9.

Table 4.7: List of join tables with table weight

| SL NO | Query | \( f_i \) | \( W_j \) | tblLoo kup Circle | tblLoo kup Distric t | tblLoo kup Than a | tblLoo kup Mouz a | tblLoo kup TaxTy pe | tblLoo kup Landu se | tblK hatian | tblT hDa g | tblKh Owne r | tblDa gKhL use | tblCase GLInfo | tblD ag LUse | tblD ag Info | tblMut KhDag | tblTr ansL andD etails |
|-------|-------|----------|---------|-----------------|-------------------|-----------------|-----------------|-----------------|-----------------|-------------|---------|------------|----------------|-------------|--------------|-------------|--------------|-------------|-------------------|
| 1     | Q1    | 5         | 2.6     | 12.9           | 12.9              | 12.9            | 12.9            | 12.9            | 0.0             | 0.0         | 12.9     | 0.0         | 0.0            | 0.0          | 0.0          | 0.0          | 0.0          | 0.0          |                   |
| 2     | Q2    | 1         | 1.0     | 0.0            | 0.0               | 0.0             | 0.0             | 0.0             | 1.0             | 1.0         | 1.0      | 0.0         | 0.0            | 0.0          | 0.0          | 0.0          | 0.0          | 0.0          |                   |
| 3     | Q3    | 3         | 2.0     | 6.0            | 6.0               | 6.0             | 6.0             | 0.0             | 0.0             | 0.0         | 12.9     | 0.0         | 0.0            | 0.0          | 0.0          | 0.0          | 0.0          | 0.0          |                   |
| 4     | Q4    | 2         | 1.6     | 0.0            | 0.0               | 0.0             | 0.0             | 0.0             | 3.2             | 3.2         | 3.2      | 0.0         | 0.0            | 0.0          | 0.0          | 0.0          | 0.0          | 0.0          |                   |
| 5     | Q5    | 6         | 2.8     | 16.8           | 16.8              | 16.8            | 0.0             | 0.0             | 0.0             | 0.0         | 0.0      | 0.0         | 0.0            | 0.0          | 0.0          | 0.0          | 0.0          | 0.0          |                   |
| 6     | Q6    | 3         | 2.0     | 0.0            | 0.0               | 0.0             | 0.0             | 6.0             | 0.0             | 0.0         | 0.0      | 0.0         | 0.0            | 0.0          | 0.0          | 0.0          | 0.0          | 0.0          |                   |
| 7     | Q7    | 2         | 1.6     | 0.0            | 0.0               | 0.0             | 0.0             | 3.2             | 0.0             | 0.0         | 0.0      | 0.0         | 0.0            | 0.0          | 0.0          | 0.0          | 0.0          | 0.0          |                   |
| 8     | Q8    | 1         | 1.0     | 0.0            | 0.0               | 0.0             | 0.0             | 0.0             | 0.0             | 0.0         | 0.0      | 0.0         | 0.0            | 0.0          | 0.0          | 0.0          | 0.0          | 0.0          |                   |
| 9     | Q9    | 1         | 1.0     | 0.0            | 0.0               | 0.0             | 0.0             | 0.0             | 0.0             | 0.0         | 0.0      | 0.0         | 0.0            | 0.0          | 0.0          | 0.0          | 0.0          | 0.0          |                   |
| 10    | Q10   | 1         | 1.0     | 0.0            | 0.0               | 0.0             | 0.0             | 0.0             | 0.0             | 0.0         | 0.0      | 0.0         | 0.0            | 0.0          | 0.0          | 0.0          | 0.0          | 0.0          |                   |
| 11    | Q11   | 1         | 1.0     | 0.0            | 0.0               | 0.0             | 0.0             | 0.0             | 0.0             | 0.0         | 0.0      | 0.0         | 0.0            | 0.0          | 0.0          | 0.0          | 0.0          | 0.0          |                   |
| 12    | Q12   | 2         | 1.6     | 3.2            | 3.2               | 3.2             | 3.2             | 0.0             | 0.0             | 3.2         | 3.2      | 3.2         | 0.0            | 0.0          | 0.0          | 0.0          | 0.0          | 0.0          |                   |
| 13    | Q13   | 2         | 1.6     | 3.2            | 3.2               | 3.2             | 3.2             | 0.0             | 0.0             | 3.2         | 3.2      | 3.2         | 0.0            | 0.0          | 0.0          | 0.0          | 0.0          | 0.0          |                   |
| 14    | Q14   | 10        | 3.5     | 0.0            | 0.0               | 0.0             | 0.0             | 34.6            | 34.6            | 0.0         | 0.0      | 0.0         | 0.0            | 0.0          | 0.0          | 0.0          | 0.0          | 0.0          |                   |
| 15    | Q15   | 3         | 2.0     | 6.0            | 6.0               | 6.0             | 6.0             | 0.0             | 0.0             | 0.0         | 6.0      | 0.0         | 0.0            | 0.0          | 0.0          | 0.0          | 6.0          | 0.0          |                   |
| 16    | Q16   | 3         | 2.0     | 6.0            | 6.0               | 6.0             | 6.0             | 0.0             | 0.0             | 0.0         | 6.0      | 0.0         | 0.0            | 0.0          | 0.0          | 0.0          | 6.0          | 0.0          |                   |
| 17    | Q17   | 2         | 1.6     | 3.2            | 3.2               | 3.2             | 3.2             | 0.0             | 0.0             | 0.0         | 0.0      | 0.0         | 0.0            | 0.0          | 0.0          | 0.0          | 0.0          | 0.0          |                   |
| 18    | Q18   | 1         | 1.0     | 0.0            | 0.0               | 0.0             | 0.0             | 1.0             | 1.0             | 0.0         | 1.0      | 0.0         | 0.0            | 0.0          | 0.0          | 0.0          | 0.0          | 0.0          |                   |
| 19    | Q19   | 3         | 2.0     | 6.0            | 6.0               | 6.0             | 6.0             | 0.0             | 0.0             | 0.0         | 0.0      | 0.0         | 0.0            | 0.0          | 0.0          | 0.0          | 0.0          | 0.0          |                   |
| 20    | Q20   | 1         | 1.0     | 0.0            | 0.0               | 0.0             | 0.0             | 1.0             | 1.0             | 0.0         | 1.0      | 0.0         | 0.0            | 0.0          | 0.0          | 0.0          | 0.0          | 0.0          |                   |
| 21    | Q21   | 3         | 2.0     | 6.0            | 6.0               | 6.0             | 0.0             | 6.0             | 0.0             | 0.0         | 0.0      | 0.0         | 0.0            | 0.0          | 0.0          | 0.0          | 0.0          | 0.0          |                   |
| 22    | Q22   | 1         | 1.0     | 0.0            | 0.0               | 0.0             | 0.0             | 0.0             | 0.0             | 0.0         | 0.0      | 0.0         | 0.0            | 0.0          | 0.0          | 0.0          | 0.0          | 0.0          |                   |
Based on the analysis given in the section 3.2.9, we have computed the min support. The value of Min (support) = 56.0.

So the following tables are candidates for de-normalization:

<table>
<thead>
<tr>
<th>SL NO</th>
<th>Query</th>
<th>( f_i )</th>
<th>( W_i )</th>
<th>tblLoo kup Circle</th>
<th>tblLoo kup Distric t</th>
<th>tblLoo kup Mouz a TaxTy pe</th>
<th>tblLoo kup Landu se</th>
<th>tblK hani</th>
<th>tblK hDa g</th>
<th>tblKh Owne r</th>
<th>tblDa gKhl use</th>
<th>tblCase GlInfo</th>
<th>tblDa g LUse</th>
<th>tblDa g Info</th>
<th>tblMut KhDag</th>
<th>tblTr ansL andD etails</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>Q23</td>
<td>4</td>
<td>2.3</td>
<td>9.3</td>
<td>9.3</td>
<td>9.3</td>
<td>0.0</td>
<td>9.3</td>
<td>9.3</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>24</td>
<td>Q24</td>
<td>1</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>25</td>
<td>Q25</td>
<td>2</td>
<td>1.6</td>
<td>3.2</td>
<td>3.2</td>
<td>3.2</td>
<td>0.0</td>
<td>0.0</td>
<td>3.2</td>
<td>3.2</td>
<td>3.2</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>26</td>
<td>Q26</td>
<td>1</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>3.2</td>
<td>3.2</td>
<td>3.2</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>27</td>
<td>Q27</td>
<td>1</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>3.2</td>
<td>3.2</td>
<td>3.2</td>
<td>3.2</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>28</td>
<td>Q28</td>
<td>2</td>
<td>1.6</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>3.2</td>
<td>3.2</td>
<td>3.2</td>
<td>3.2</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>29</td>
<td>Q29</td>
<td>1</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>30</td>
<td>Q30</td>
<td>1</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>3.2</td>
<td>0.0</td>
<td>0.0</td>
<td>3.2</td>
</tr>
<tr>
<td>31</td>
<td>Q31</td>
<td>2</td>
<td>1.6</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>32</td>
<td>Q32</td>
<td>3</td>
<td>2.0</td>
<td>6.0</td>
<td>6.0</td>
<td>6.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>33</td>
<td>Q33</td>
<td>1</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>34</td>
<td>Q34</td>
<td>2</td>
<td>1.6</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>3.2</td>
<td>3.2</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>35</td>
<td>Q35</td>
<td>1</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>36</td>
<td>Q36</td>
<td>1</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>37</td>
<td>Q37</td>
<td>2</td>
<td>1.6</td>
<td>3.2</td>
<td>3.2</td>
<td>3.2</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>38</td>
<td>Q38</td>
<td>1</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>39</td>
<td>Q39</td>
<td>4</td>
<td>2.3</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>9.3</td>
<td>0.0</td>
<td>9.3</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>9.3</td>
<td>0.0</td>
</tr>
<tr>
<td>40</td>
<td>Q40</td>
<td>2</td>
<td>1.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>0.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>41</td>
<td>Q41</td>
<td>3</td>
<td>2.0</td>
<td>6.0</td>
<td>6.0</td>
<td>6.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>6.0</td>
<td>6.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>42</td>
<td>Q42</td>
<td>3</td>
<td>2.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>6.0</td>
<td>6.0</td>
<td>0.0</td>
<td>0.0</td>
<td>6.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>43</td>
<td>Q43</td>
<td>3</td>
<td>2.0</td>
<td>6.0</td>
<td>6.0</td>
<td>6.0</td>
<td>0.0</td>
<td>6.0</td>
<td>0.0</td>
<td>0.0</td>
<td>6.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

| Total  | 106.9 | 101.7 | 101.7 | 103.7 | 60.5 | 60.5 | 49.2 | 40.3 | 31.8 | 30.5 | 25.5 | 71.0 | 12.0 | 29.8 | 14.3 |
Table 4.8: Candidates of denormalized Tables

<table>
<thead>
<tr>
<th>SL. No</th>
<th>Weight</th>
<th>Table name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>106.9</td>
<td>tblLookupCircle</td>
</tr>
<tr>
<td>2</td>
<td>101.7</td>
<td>tblLookupDistrict</td>
</tr>
<tr>
<td>3</td>
<td>101.7</td>
<td>tblLookupThana</td>
</tr>
<tr>
<td>4</td>
<td>103.7</td>
<td>tblLookupMouza</td>
</tr>
<tr>
<td>5</td>
<td>60.5</td>
<td>tblLookupTaxType</td>
</tr>
<tr>
<td>6</td>
<td>60.5</td>
<td>tblLookupLanduse</td>
</tr>
<tr>
<td>7</td>
<td>71.0</td>
<td>tblDagLuse</td>
</tr>
</tbody>
</table>

After finding the candidates of de-normalized tables (Table 4.8) we have applied the Apriori algorithm to find the large item sets. First we considered the tables that have weight have more than minimum support. Table 4.9 shows different iteration to find the association among tables.

Table 4.9: Frequency table applying Apriori algorithms

<table>
<thead>
<tr>
<th>Table Association</th>
<th>No of tables</th>
<th>Weight=$\sum$ wt</th>
<th>Confidence</th>
<th>Candidate</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Iteration-1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-&gt;2</td>
<td>2</td>
<td>106</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>1-&gt;3</td>
<td>2</td>
<td>102</td>
<td>96</td>
<td>1</td>
</tr>
<tr>
<td>1-&gt;4</td>
<td>2</td>
<td>102</td>
<td>96</td>
<td>1</td>
</tr>
<tr>
<td>1-&gt;5</td>
<td>2</td>
<td>19</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>1-&gt;6</td>
<td>2</td>
<td>13</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>1-&gt;12</td>
<td>2</td>
<td>13</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>2-&gt;3</td>
<td>2</td>
<td>102</td>
<td>96</td>
<td>1</td>
</tr>
<tr>
<td>2-&gt;4</td>
<td>2</td>
<td>103</td>
<td>97</td>
<td>1</td>
</tr>
<tr>
<td>2-&gt;5</td>
<td>2</td>
<td>19</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>2-&gt;6</td>
<td>2</td>
<td>13</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>2-&gt;12</td>
<td>2</td>
<td>22</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>3-&gt;4</td>
<td>2</td>
<td>102</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>3-&gt;5</td>
<td>2</td>
<td>19</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>3-&gt;6</td>
<td>2</td>
<td>13</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>3-&gt;12</td>
<td>2</td>
<td>22</td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>4-&gt;5</td>
<td>2</td>
<td>19</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>4-&gt;6</td>
<td>2</td>
<td>13</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>4-&gt;12</td>
<td>2</td>
<td>22</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>5-&gt;6</td>
<td>2</td>
<td>55</td>
<td>90</td>
<td>1</td>
</tr>
<tr>
<td>5-&gt;12</td>
<td>2</td>
<td>48</td>
<td>79</td>
<td>1</td>
</tr>
<tr>
<td>6-&gt;12</td>
<td>2</td>
<td>48</td>
<td>79</td>
<td>1</td>
</tr>
<tr>
<td>Table Association</td>
<td>No of tables</td>
<td>Weight=$\Sigma$ wt</td>
<td>Confidence</td>
<td>Candidate</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------</td>
<td>-------------------</td>
<td>------------</td>
<td>-----------</td>
</tr>
<tr>
<td>(1-&gt;2)-&gt;3</td>
<td>3</td>
<td>102</td>
<td>96</td>
<td>1</td>
</tr>
<tr>
<td>(1-&gt;2)-&gt;4</td>
<td>3</td>
<td>103</td>
<td>97</td>
<td>1</td>
</tr>
<tr>
<td>(1-&gt;2)-&gt;5</td>
<td>3</td>
<td>19</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>(1-&gt;2)-&gt;6</td>
<td>3</td>
<td>13</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>(1-&gt;2)-&gt;12</td>
<td>3</td>
<td>22</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>(1-&gt;3)-&gt;4</td>
<td>3</td>
<td>102</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>(1-&gt;3)-&gt;5</td>
<td>3</td>
<td>19</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>(1-&gt;3)-&gt;6</td>
<td>3</td>
<td>13</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>(1-&gt;3)-&gt;12</td>
<td>3</td>
<td>22</td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>(1-&gt;4)-&gt;5</td>
<td>3</td>
<td>19</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>(1-&gt;4)-&gt;6</td>
<td>3</td>
<td>13</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>(1-&gt;4)-&gt;12</td>
<td>3</td>
<td>22</td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>(2-&gt;3)-&gt;4</td>
<td>3</td>
<td>102</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>(2-&gt;3)-&gt;5</td>
<td>3</td>
<td>19</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>(2-&gt;3)-&gt;6</td>
<td>3</td>
<td>13</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>(2-&gt;3)-&gt;12</td>
<td>3</td>
<td>22</td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>(2-&gt;4)-&gt;5</td>
<td>3</td>
<td>19</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>(2-&gt;4)-&gt;6</td>
<td>3</td>
<td>13</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>(2-&gt;4)-&gt;12</td>
<td>3</td>
<td>22</td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>(3-&gt;4)-&gt;5</td>
<td>3</td>
<td>19</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>(3-&gt;4)-&gt;6</td>
<td>3</td>
<td>13</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>(3-&gt;4)-&gt;12</td>
<td>3</td>
<td>22</td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>(5-&gt;6)-&gt;12</td>
<td>3</td>
<td>48</td>
<td>87</td>
<td>1</td>
</tr>
<tr>
<td>((1-&gt;2)-&gt;3)-&gt;4</td>
<td>4</td>
<td>102</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>((1-&gt;2)-&gt;3)-&gt;5</td>
<td>4</td>
<td>19</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>((1-&gt;2)-&gt;3)-&gt;6</td>
<td>4</td>
<td>13</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>((1-&gt;2)-&gt;3)-&gt;12</td>
<td>4</td>
<td>22</td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>((1-&gt;2)-&gt;4)-&gt;5</td>
<td>4</td>
<td>19</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>((1-&gt;2)-&gt;4)-&gt;6</td>
<td>4</td>
<td>13</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>((1-&gt;2)-&gt;4)-&gt;12</td>
<td>4</td>
<td>22</td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>((1-&gt;3)-&gt;4)-&gt;5</td>
<td>4</td>
<td>19</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>((1-&gt;3)-&gt;4)-&gt;6</td>
<td>4</td>
<td>13</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>((1-&gt;3)-&gt;4)-&gt;12</td>
<td>4</td>
<td>22</td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>(((1-&gt;2)-&gt;3)-&gt;4)-&gt;5</td>
<td>5</td>
<td>19</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>(((1-&gt;2)-&gt;3)-&gt;4)-&gt;6</td>
<td>5</td>
<td>13</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>(((1-&gt;2)-&gt;3)-&gt;4)-&gt;12</td>
<td>5</td>
<td>22</td>
<td>22</td>
<td>0</td>
</tr>
</tbody>
</table>

In the iteration-1 we identified those tables which have higher weight value then min support.
We have considered minimum confidence is 50% and minimum support is 56.0. So in iteration-1 we have found ($Table_1 \rightarrow Table_2$), ($Table_1 \rightarrow Table_3$), ($Table_1 \rightarrow Table_4$), ($Table_2 \rightarrow Table_3$), ($Table_2 \rightarrow Table_4$), ($Table_3 \rightarrow Table_4$) as the candidates for de-normalization as well as candidates for next iteration. In iteration-2, we have found ($Table_1, Table_2 \rightarrow Table_3$), ($Table_1, Table_2 \rightarrow Table_4$), ($Table_1, Table_3 \rightarrow Table_4$), ($Table_2, Table_3 \rightarrow Table_4$). These are used for third iteration. In iteration-3, we have found only ($Table_1, Table_2, Table_3 \rightarrow Table_4$). In iteration-4, no tables are found. So we have to stop de-normalization in iteration three in our experiment.

The following Figure shows the de-normalization of the four tables that we identified in the above process.

**Normalized Database**
Query performance:

The response time for point query is described in the following tables for fully normalized database and tuned database schema.

Table 4.9: Response time for point query

<table>
<thead>
<tr>
<th>Number of Records</th>
<th>Query Type</th>
<th>Fully normalized Response time in second</th>
<th>Tuned Database Response time in second</th>
</tr>
</thead>
<tbody>
<tr>
<td>44296</td>
<td>Point Query</td>
<td>3.57</td>
<td>1.95</td>
</tr>
<tr>
<td>221480</td>
<td>Point Query</td>
<td>14.67</td>
<td>10.17</td>
</tr>
<tr>
<td>487256</td>
<td>Point Query</td>
<td>67.6</td>
<td>16.96</td>
</tr>
</tbody>
</table>

Figure 4.23: Response time for point query
The above Figure 4.23 shows the response time gain by de-normalization process. In two databases schema there is slight difference in response time for small number of records, but for larger amount of records, the response time difference is significant. For large number records, expensive joining operations make delay in the normalized database.

![Graph showing response time for Aggregate query](image)

**Figure 4.24:** Response time for Aggregate query

Significant response time gain is also achieved for aggregate query in the tuned database. The grouping in the normalized table is applied after joining of four tables in case of normalized database. On the other hand grouping is applied to the tuned database without joining of four different tables. This is the cause of improvement in the aggregate query in the tuned database.
The response time gain for range query is shown in Figure 4.25. The gain in the range query is not as significant as the other two cases. In range query the joining is not a big factor.
4.4 Performance analysis between Entity-Attribute-Value (EAV), Universal and Binary database design

We have considered a database containing keywords of different documents.

**Option -1: EAV representation**

The entity-attribute-value represent the wider range of data types. In this data model attributes are presented row wise rather than column. The value of the all attributes are described in the value field. The following is the EAV representation of our experimental schema, there are keywords name will be on attribute name and value will be on the value field:

![EAV database schema](image)

**Figure 4.25: EAV database schema**

**Option-2: Universal representation**

There are thousand (1000) keywords for a document; we consider an entity set with thousand fields as shown in the following Figure:

![Universal presentation of document entity](image)

**Figure 4.26: Universal presentation of document entity**

In the universal entity set there are 20% fill by keywords, so there are many NULL values in the entity set.
Option-3: Binary representation

In the binary relation that have two attributes in an entity, ID and value. The keywords are stored in the binary schema in individual entity, there are 1000 tables to store keywords

![Binary representation of entity set](image)

**Figure 4.27:** Binary representation of entity set

**Database space simulation**

In our experiments we have populated data from .1 million to .4 million, the space requirements for the database for binary, EAV and universal is given in the Figure 4.28.

![Size vs number of records](image)

**Figure 4.28:** Space requirements for Binary, EAV and Universal database

The space requirements for universal model is much higher than Entity-Attribute-Value (EAV) and Binary model. In our experiment Universal model contains many null value, there are only 20% value is presented per tuple. EAV model require less storage because all present value are stored in a three columns table, there is no null value. In the binary model the value are stored in binary tables, there are 1000 tables for 1000 number of column, only presented value are stored in the corresponding binary table. For EAV database all data are stored in three columns of a table and no
null value is stored in the EAV table, so it requires less storage compare to others database.

Response Time Analysis
To make the analysis database performance in respect to response time we have executed three types of queries in the all database schema.

Point Query
The response time for point query is described in the Figure 4.29. Universal database have lower response time than other two database. The Binary database requires much time then other database. EAV database performance is the middle of two database system.

Table 4.10: Select Query (point Query) response time

<table>
<thead>
<tr>
<th>Number of Records</th>
<th>Query</th>
<th>Select Columns</th>
<th>Response time in second</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Universal</td>
</tr>
<tr>
<td>100000</td>
<td>Select -5</td>
<td>5</td>
<td>0.45</td>
</tr>
<tr>
<td>200000</td>
<td>Select -5</td>
<td>5</td>
<td>0.39</td>
</tr>
<tr>
<td>300000</td>
<td>Select -5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>400000</td>
<td>Select -5</td>
<td>5</td>
<td>1.61</td>
</tr>
</tbody>
</table>

Figure 4.29: Response time for select query in universal, EAV and binary database
Aggregate Query

Response time for aggregate query is described in Table 4.11. We have tested aggregate query using 5, 10, 15 columns. The SQL script is described in appendix.

Table 4.11: Aggregate Query response time

<table>
<thead>
<tr>
<th>Number of Records</th>
<th>Query</th>
<th>Select Columns</th>
<th>Response time in second</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Universal</td>
</tr>
<tr>
<td>100000</td>
<td>Aggregate - 5</td>
<td>5</td>
<td>0.26</td>
</tr>
<tr>
<td>200000</td>
<td>Aggregate - 5</td>
<td>5</td>
<td>0.53</td>
</tr>
<tr>
<td>300000</td>
<td>Aggregate - 5</td>
<td>5</td>
<td>0.81</td>
</tr>
<tr>
<td>400000</td>
<td>Aggregate - 5</td>
<td>5</td>
<td>1.09</td>
</tr>
<tr>
<td>100000</td>
<td>Aggregate - 10</td>
<td>10</td>
<td>0.2</td>
</tr>
<tr>
<td>200000</td>
<td>Aggregate - 10</td>
<td>10</td>
<td>0.43</td>
</tr>
<tr>
<td>300000</td>
<td>Aggregate - 10</td>
<td>10</td>
<td>0.54</td>
</tr>
<tr>
<td>400000</td>
<td>Aggregate - 10</td>
<td>10</td>
<td>0.65</td>
</tr>
<tr>
<td>100000</td>
<td>Aggregate -15</td>
<td>15</td>
<td>0.45</td>
</tr>
<tr>
<td>200000</td>
<td>Aggregate -15</td>
<td>15</td>
<td>0.81</td>
</tr>
<tr>
<td>300000</td>
<td>Aggregate -15</td>
<td>15</td>
<td>2.03</td>
</tr>
<tr>
<td>400000</td>
<td>Aggregate -15</td>
<td>15</td>
<td>3.25</td>
</tr>
</tbody>
</table>

The response time for aggregate query of five columns is described in the above Figure 4.30. In this experiment we have counted the number of occurrence of keywords in the five column. For binary schema we have to join five tables, for universal table we search five columns for universal table, for EAV table we select
five attributes for attribute column. The response time is much less for universal table. Binary table shows better performance with respect to EAV table. As number of columns increase, performance degrade dramatically for binary database. Figure 4.31 and Figure 4.32 show response time for higher number of columns in this database.

**Figure 4.31: Response time for Aggregate with ten fields**

**Figure 4.32: Response time for Aggregate with fifteen fields**
Range Query
The response time for range query is described in the table 4.12.

Table 4.12: Range Query response time

<table>
<thead>
<tr>
<th>Number of Records</th>
<th>Query</th>
<th>Select Columns</th>
<th>Response time in second</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Universal</td>
</tr>
<tr>
<td>100000</td>
<td>Range -5</td>
<td>5</td>
<td>0.31</td>
</tr>
<tr>
<td>200000</td>
<td>Range -5</td>
<td>5</td>
<td>0.59</td>
</tr>
<tr>
<td>300000</td>
<td>Range -5</td>
<td>5</td>
<td>0.87</td>
</tr>
<tr>
<td>400000</td>
<td>Range -5</td>
<td>5</td>
<td>1.78</td>
</tr>
<tr>
<td>100000</td>
<td>Range -10</td>
<td>10</td>
<td>0.1</td>
</tr>
<tr>
<td>200000</td>
<td>Range -10</td>
<td>10</td>
<td>0.34</td>
</tr>
<tr>
<td>300000</td>
<td>Range -10</td>
<td>10</td>
<td>0.5</td>
</tr>
<tr>
<td>400000</td>
<td>Range -10</td>
<td>10</td>
<td>0.66</td>
</tr>
<tr>
<td>100000</td>
<td>Range -15</td>
<td>15</td>
<td>0.06</td>
</tr>
<tr>
<td>200000</td>
<td>Range -15</td>
<td>15</td>
<td>0.43</td>
</tr>
<tr>
<td>300000</td>
<td>Range -15</td>
<td>10</td>
<td>0.57</td>
</tr>
<tr>
<td>400000</td>
<td>Range -15</td>
<td>5</td>
<td>0.71</td>
</tr>
</tbody>
</table>

The response time comparison for five columns is described in the Figure 4.33. For binary schema we have to join five tables, for universal table we search five columns for universal table, for EAV table we select five attributes for attribute column. The Universal and Binary database shows steady performance and EAV database performance sharply increases for higher number of records.

Figure 4.33: Response time for range query with five fields
The response time comparison for five columns is described in the Figure 4.34. For binary schema we have to join fifteen tables, for universal table we search fifteen columns for universal table, for EAV table we select fifteen attributes for attribute column. In the binary database schema requires high response time because of large number of joining. In the lower number of records binary database schema shows better performance but when the number of records is high then binary database response time requires larger time then EAV database.

![Figure 4.34: Response time for Range query with fifteen fields](image)
4.5 Performance gain using partitioning

We have considered a logical schema as the following Figure

![Conceptual schema of customer sales](image)

**Figure 4.35:** Conceptual schema of customer sales

The customer entity set contains the attributes Customer id, customer name, and Address. The Product entity set contains product id, product title, and group id. The sales is the relational entity set contain both primary key of customer and product, sales date and sales quantity.

The following is the SQL scripts to convert ERD to relational schema

```
1. Create table Customer
   (Customer_id Number(16), Customer_name varchar2(100),
    Customer_address varchar2(255), Regdate Date,
    cusGrp Number);

2. Create table Product
   (Product_id Number(16),
    Title Varchar2(255),
    Groupid Number(2));

3. Create table Sales_u
   (Customer_id Number(16),Product_id Number(16),
    Quantity number, sales_date Date);
```

**Figure 4.36:** SQL scripts to convert ERD to relational schema

We have considered the partitioning in three stages:

a) Single partition: Partition the sales entity set only.

b) Double partition: Partition customer and sales entity set.

c) Triple partition: Partition customer, sales and product entity sets
The following SQL scripts is used to convert the logical schema to relational table in single partition:

```
CREATE TABLE sales
(  customer_id     NUMBER(16),
  product_id      NUMBER(16),
  quantity        NUMBER,
  sales_date      DATE
)
PARTITION BY RANGE (sales_date)
(  PARTITION year_2006 VALUES LESS THAN ('01-JAN-2007')
  TABLESPACE year_2006,
  PARTITION year_2007 VALUES LESS THAN ('01-JAN-2008')
  TABLESPACE year_2007,
  PARTITION year_2008 VALUES LESS THAN (MAXVALUE)
  TABLESPACE year_2008 );
```

Figure 4.38: SQL scripts to convert in single partition

The sales table is horizontal partitioned in based of sales date. Three partitions have created on sales table: "Partition 2006" contain the sales data of year 2006, "Partition 2007" contains the sales data of year 2007, "Partition 2008" contains the sales data of year 2008. There are three tablespaces that contain these three partitions. Different
tablespace is used to improve I/O speed and parallel query processing. In the same way we have partitioned the “Customer” entity and “Product” entity set. For both customer and product partition tables we have taken three different tablespaces. There are nine tablespaces that are used to partitioned these three tables, the following is the thematic representation of partitioned tables (Figure 4.39):

![Thematic representation of partitioned tables](image)

**Figure 4.39: thematic representation of partitioned tables**

**Storage requirements**

To analyze the space of the database we have populated records using a customized program.

**Table 4.13: Size of un-partition and partition database**

<table>
<thead>
<tr>
<th>Record Number</th>
<th>Database size in MB</th>
<th>Un-partition</th>
<th>Single Partition</th>
<th>Double Partition</th>
<th>Triple Partition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000000</td>
<td></td>
<td>63</td>
<td>65</td>
<td>65</td>
<td>66</td>
</tr>
<tr>
<td>2000000</td>
<td></td>
<td>124</td>
<td>125</td>
<td>126</td>
<td>127</td>
</tr>
<tr>
<td>3000000</td>
<td></td>
<td>157</td>
<td>186</td>
<td>187</td>
<td>187</td>
</tr>
<tr>
<td>4000000</td>
<td></td>
<td>195</td>
<td>253</td>
<td>254</td>
<td>251</td>
</tr>
</tbody>
</table>

The record number of sales table is used as a reference for the number of record in this database. The data generator has generated one to four million records for Sales table. As sales is the relation between Customer and product table, fewer records are generated for these tables.
Figure 4.40: Storage requirements of partition tables

Figure 4.40 shows the database size of four options (un-partition and partitions). The increase of size is almost linear with number records. The storage requirements is almost same in all cases for one million records, but for the higher number of records the storage requirements is more for partitioned table. With table partitioning, some automatic clustering of the data with the index will take place automatically, since new table rows will be directed to the partition that contains similar partition values. The space for these index require more for partitioned table. In terms of space requirements the un-partition and partition tables are almost same.

**Query performance**

The response time has measured for three type of partitioned database using three types of queries: point query, aggregate query and range query.

**Point query:**

<table>
<thead>
<tr>
<th>Record Number</th>
<th>Response time in second</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Un-partition</td>
</tr>
<tr>
<td>1000000</td>
<td>1.7</td>
</tr>
<tr>
<td>2000000</td>
<td>5.5</td>
</tr>
<tr>
<td>3000000</td>
<td>5.76</td>
</tr>
<tr>
<td>4000000</td>
<td>109.06</td>
</tr>
</tbody>
</table>
Response time gain is significant for large number of records due to partitioning (Figure 4.41). For all database schema the response time sharply increases after three million number of tuples. This is due to exceeds of buffer cache memory area needs extra I/O operation. In this position single partition performed better than un-partition database schema. The partitioned table can perform well because database optimizer can detect the values within each partition and access only those partitions that are necessary to service the query.

The SQL scripts used for the point query is as shown in Figure 4.4.2:
**Query 1: Un-partitioned**


**Query 2: Single Partition**


**Query 3: Double Partition**


**Query 4: Triple Partition**


**Figure 4.42: SQL scripts for point query**

**Full table scan:**

The query output of the full table scan is as shown in Table 4.15 and and Figure 4.43:

**Table 4.15: Query response time for full table scan**

<table>
<thead>
<tr>
<th>Record Number</th>
<th>Response time in second</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Un-partition</td>
</tr>
<tr>
<td>1000000</td>
<td>1.01</td>
</tr>
<tr>
<td>2000000</td>
<td>2.32</td>
</tr>
<tr>
<td>3000000</td>
<td>3.34</td>
</tr>
<tr>
<td>4000000</td>
<td>45.32</td>
</tr>
</tbody>
</table>
Figure 4.43: Response time for full table scan

The response time behavior of full table scan is almost the same as point query. There is also sharply increase of response time from record number three million to four million. The difference is not significant in the lower number of records and the difference increases for larger number of records.

Aggregate Query:

The response time behavior for aggregate query in the partitioned table is shown in Table 4.16 and Figure 4.44.

Table 4.16: Query response time for aggregate query

<table>
<thead>
<tr>
<th>Record Number</th>
<th>Un-partition</th>
<th>Single Partition</th>
<th>Double Partition</th>
<th>Triple Partition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000000</td>
<td>17.34</td>
<td>9.5</td>
<td>5.28</td>
<td>4.29</td>
</tr>
<tr>
<td>2000000</td>
<td>41.32</td>
<td>17.31</td>
<td>13.82</td>
<td>12.68</td>
</tr>
<tr>
<td>3000000</td>
<td>78.3</td>
<td>38.53</td>
<td>22.64</td>
<td>19.64</td>
</tr>
<tr>
<td>4000000</td>
<td>183.05</td>
<td>98.5</td>
<td>76</td>
<td>66.5</td>
</tr>
</tbody>
</table>
Figure 4.44: Response time for aggregate query

For aggregate query, response time gain is significant for partitioned table. The single partition response time is almost half then the un-partitioned table response time in the all four samples of record count. In the aggregate query, response time depends on the number of records processing and complexity of the aggregate function. As the tables are horizontally partitioned in our experiment, there are less number of records processing compared to un-partitioned tables. The response time gain is not same for all portioned schema. The relation entity set (sales) is partitioned in the single partitioned schema, relation entity set is containing more records then other two entity set and this table joined other two tables, so partitioned of this table give significant gain in response time.

The SQL scripts are used for the aggregate query in our experiment is given below:
Query 01: Un partitioned Aggregate Query

```
SELECT TO_CHAR(s.sales_date,'YYYY'), sum (s.QUANTITY) as sQ
FROM Customer c, SALES_U s, Product p WHERE c.Customer_id = s.Customer_id and s.Product_id = p.Product_id GROUP BY TO_CHAR(s.sales_date,'YYYY')
```

Query 02: Single partitioned Aggregate Query

```
SELECT TO_CHAR(s.sales_date,'YYYY'), sum (s.QUANTITY) as sQ
FROM Customer c, SALES Partition (year_2006) s, Product p WHERE c.Customer_id = s.Customer_id and s.Product_id = p.Product_id GROUP BY TO_CHAR(s.sales_date,'YYYY')
```

Query 03: Double partition Aggregate Query

```
SELECT TO_CHAR(s.sales_date,'YYYY'), sum (s.QUANTITY) as sQ
FROM Customer_P partition (Customer_1000) c, SALES Partition (year_2006) s, Product p WHERE c.Customer_id = s.Customer_id and s.Product_id = p.Product_id GROUP BY TO_CHAR(s.sales_date,'YYYY')
```

Query 04: Triple partition Aggregate Query

```
SELECT TO_CHAR(s.sales_date,'YYYY'), sum (s.QUANTITY) as sQ
FROM Customer_P partition (Customer_1000) c, SALES Partition (year_2006) s, Product_P partition (Product_1000) p WHERE c.Customer_id = s.Customer_id and s.Product_id = p.Product_id GROUP BY TO_CHAR(s.sales_date,'YYYY')
```

Figure 4.45: SQL Scripts for aggregate query

Range Query

The partition of the entity set is based on the range value of an attribute, for sales table in our experiment is based on the sales date, the product and customer table is partitioned based upon the group id. An index is automatically updated in the partition table based on the partitioned attribute value. If a range query is processed based upon this attribute value then the response time gain will be better then unpartitioned table. In our experiments the following results are found for range query in partition tables (Table 4.17 and Figure 4.46)

Table 4.17: Response time for Range query

<table>
<thead>
<tr>
<th>Record Number</th>
<th>Response time in second</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unpartition</td>
</tr>
<tr>
<td>1000000</td>
<td>17.68</td>
</tr>
<tr>
<td>2000000</td>
<td>89.5</td>
</tr>
<tr>
<td>3000000</td>
<td>93.1</td>
</tr>
<tr>
<td>4000000</td>
<td>381.7</td>
</tr>
</tbody>
</table>
The response time changes is almost same for partitioned tables, single partitioned, double partitioned and triple partitioned tables and increases linearly. In respect to un-partitioned table, the response time gain is significant for partitioned table, particularly in case of record number four million, the response time of un-partitioned table is two hundred times then the portioned tables.

The response time gain is not same for un-partition to single partition and single partition to double partition and double partitioned to triple partitioned. There is huge time gain in the single partition table, but with the reference of single partition other partitioned tables gain difference is not significant. So we have to decide carefully which table should be partitioned to get optimum benefits.

4.6 Performance tuning changing database parameters

In our experiment, we have used Oracle 10g as a database server running on a personal computer (PC). The computer processor is 2GHz Intel Core2 Duo, RAM 2 GB, Hard disk SATA.

Before the start of the tuning of database parameters, we have set some measurable tuning goals. The tuning goals are:

- Response time: time to process of a query processing and receive data of a request.
- Database hit percentages: provide a good baseline from which to determine performance increase or decrease after tuning.
- Memory utilization: the proper utilization of memory by server after changing parameters during the query processing time

The recommended tuning steps of database could be following:
- The database design (that is described earlier section)
- The application SQL
- The memory
- Input/Output
- Contention
- Operating system

In early steps, tuning cause more efficient application development and database can perform more efficient. If an application is using a lot of full table scan, this may show up excessive I/O. There is no point in resizing the buffer cache or redistributing disk files if we can rewrite the queries so that they can access only 4 blocks instead of 4,000. so the database parameter changes can improve the overall performance when there is more tune in the application area. The database parameters should be set for the nature of the application run on the server. The parameter for OLTP system may be not same for the parameter for DSS type system.

The following tuning parameters has been tested and changed in the performance tuning:

### 4.6.1 Tuning Share Pool Size

The System Global Area (SGA) is a chunk of memory that is allocated by an Oracle Instance and is shared among Oracle processes, hence the name. It contains all sorts of information about the instance and the database that is needed to operate.
The SGA consists of database buffer cache, redo log buffer, shared pool and large pool. Shared pool is one the main component of SGA, where all SQL, parse SQL, table definition etc stored. Shared pool consist of library cache which stores shared SQL and PL/SQL areas, data dictionary cache, which keeps information about dictionary objects and user global area which keeps information about the multithreaded area. A cache miss on the data dictionary cache or library cache is more expensive then database buffer cache. Tuning shared pool is a priority.

The size of the shared pool is set in the Oracle database with "SHARED_POOL_SIZE". In our experiments we have set this parameter to a minimum value then startup database, monitor the database activity after execute some SQL scripts. We measure the misses in the data dictionary and library cache. The parsed SQL are kept in the library cache, if the requested SQL are found in the cache then no more parsing again by the server and if the server not found in the library cache then only server parse this SQL. The shared pool size is set by "SHARED_POOL_SIZE" parameter in the init.ora initialize parameter file, after setting the parameter file database should be shut down and again startup again. To analyze for setting a suitable shared pool size, se have to setup the Oracle tracing activities. There is some pre-defined tools come with oracle to view the statistics of the database activity. In our experiment we have take UTLBSTAT and UTLESTAT utilities, these are some SQL scripts to gather statistics the database activities. Before starting our experiments we have set the SHARED_POOL_SIZE to a minimum value then we have run the UTLBSTAT scripts, it creates environment to take statistics, then we have run some SQL in our database having 5 million records in
that time. We have monitored the “GETHITRATIO” parameter value, this parameter indicates the parsing time gain and server hit ratio to the cache. We have taken six sample settings of shared pool size data from 40 MB to 120 MB. The following tables we got the experimental data:

Table 4.18: Shared pool size setting

<table>
<thead>
<tr>
<th>Shared Pool Size (MB)</th>
<th>Get hit Ratio (Relative parse time gain)</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>0.9880</td>
</tr>
<tr>
<td>50</td>
<td>0.9940</td>
</tr>
<tr>
<td>60</td>
<td>0.9980</td>
</tr>
<tr>
<td>80</td>
<td>1.0000</td>
</tr>
<tr>
<td>100</td>
<td>1.0000</td>
</tr>
<tr>
<td>120</td>
<td>1.0010</td>
</tr>
</tbody>
</table>

Figure 4.48: Shared pool size with respect to parse time

In the above Figure clearly describes, there is performance increase from 40 MB shared pool size to 80 MB, after that performance is almost static. So, the current setting of 80 MB of shared pool size is optimum parameter value.

4.6.2 Tuning Buffer Cache

The buffer cache holds copies of the data blocks from data files. Because it is a part of System Global Area (SGA), these blocks can be shared by all users. The database server handles memory by block. Block is the smallest unit of data file I/O and that can be allocated in the memory. Oracle block is the multiple of operating system
blocks. The size of the buffer cache can be set using `DB_BLOCK_BUFFERS` parameter in the parameter files. This parameter specifies the number of blocks in the buffer cache. To find the size of the cache in bytes, we have to multiplied `DB_BLOCK_BUFFERS` by `DB_BLOCK_SIZE` (size of a block in bytes).

Because physical I/O takes significant time and increase CPU demand, Database server performance can be improved when the servers find most of the blocks that they need in memory. The statistic that measures the performance of the database buffer cache is the cache-hit ratio. This statistics is the ratio of the number of blocks found in memory to the number of blocks accessed. When the database buffer is too small, the system is slower because it is performing too many I/Os.

**Buffer Cache Advisory**

In the Oracle latest version also includes an advisory to help Database Administrators (DBAs) size the buffer cache optimally. This advisory relies on an internal simulation based on the current workload to predict the cache "miss" rates for various sizes of the buffer cache ranging from 10% to 200% of the current cache size. These predictions are published through a new view `V$DB_CACHE_ADVICE`. This view can be used to determine if the current size of the buffer cache should be increased or decreased to ensure optimal performance for the present workload. By providing a deterministic way to size the buffer cache, Oracle takes the guesswork out of database memory configuration thereby eliminating wastage caused by memory over allocation.

The buffer cache advisory is turned off by default since there is a minor performance overhead associated with the data collection and cache simulation. It can be turned on by setting the parameter `DB_CACHE_ADVICE` to `ON`. Changing the value of this parameter to `READY` stops further data collection but preserves the advisory results in the `V$DB_CACHE_ADVICE` view. The contents of this view are cleared when the advisory is turned off by setting the `DB_CACHE_ADVICE` parameter `OFF`. 
4.7 Space calculation in the nested relational model

The nested relational data model contain complex data structure. The whole entity set can be inserted in the nested tables. A nested table is a table that is embedded in another table. It is possible to define a table data type and to use this type as a column type in a table. So, this column contains a table (called nested table) with a collection of values, objects or references. A nested table is a collection type but it is also a table. For nested relational model we have described a hypothetical analysis in the above Figure, for ease of understand this analysis again described here.

Experimental setup and proof of the above analysis

In our experimental setup we consider three level of nesting tables. "Khatian" is main table that contains "KhatianNO", "JLNO" as primary key, tax information plots information in "Dags" column and owners information in "Owners" relational column. Again "Dags" second level nested table contains ten atomic value column
and one “LandUses” relational valued column. The schema is described below in the Figure 4.50 and Figure 4.51:

**Khatian Table**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circle_Code</td>
<td>(Number)</td>
</tr>
<tr>
<td>JLNO</td>
<td>(Number)</td>
</tr>
<tr>
<td>KhatianNO</td>
<td>(Number)</td>
</tr>
<tr>
<td>RNO</td>
<td>(Number)</td>
</tr>
<tr>
<td>Revenue_Taka</td>
<td>(Number)</td>
</tr>
<tr>
<td>Revenue_Pisa</td>
<td>(Number)</td>
</tr>
<tr>
<td>Remarks</td>
<td>(Varchar2)</td>
</tr>
<tr>
<td>Total_Area_Acre</td>
<td>(Number)</td>
</tr>
<tr>
<td>Total_Area_Thousand</td>
<td>(Number)</td>
</tr>
<tr>
<td>CaseNo</td>
<td>(Varchar2)</td>
</tr>
<tr>
<td>Case_Date</td>
<td>(Date)</td>
</tr>
<tr>
<td>Jet_No</td>
<td>(Varchar2)</td>
</tr>
<tr>
<td>Approval_Date</td>
<td>(Date)</td>
</tr>
<tr>
<td>Sp.Remarks</td>
<td>(Varchar2)</td>
</tr>
<tr>
<td>InherentType</td>
<td>(Varchar2)</td>
</tr>
<tr>
<td>DACS_Nt_Indag_typ</td>
<td>(Nested Table)</td>
</tr>
<tr>
<td>Owners</td>
<td>(Nested Table)</td>
</tr>
</tbody>
</table>

**Khatian-plot Nested Table (DACS), level-1**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DagNo</td>
<td>(Varchar2(50))</td>
</tr>
<tr>
<td>DagSLNO</td>
<td>(Number)</td>
</tr>
<tr>
<td>LandClass</td>
<td>(Varchar2)</td>
</tr>
<tr>
<td>LandUse</td>
<td>(Varchar2)</td>
</tr>
<tr>
<td>Total_Dag_Acre</td>
<td>(Number)</td>
</tr>
<tr>
<td>Total_Dag_Thousand</td>
<td>(Number)</td>
</tr>
<tr>
<td>Part_Dag_Kh</td>
<td>(Number)</td>
</tr>
<tr>
<td>Land_Dag_Acre</td>
<td>(Number)</td>
</tr>
<tr>
<td>Land_Dag_Thousand</td>
<td>(Number)</td>
</tr>
<tr>
<td>Remarks</td>
<td>(Varchar2)</td>
</tr>
<tr>
<td>LandUses</td>
<td>(Varchar2)</td>
</tr>
<tr>
<td>Nt_LandUse_typ</td>
<td>(Nested Table)</td>
</tr>
</tbody>
</table>

**Plot-land use Nested Table (LandUses), level-2**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUCode</td>
<td>(Number)</td>
</tr>
<tr>
<td>LUDesc</td>
<td>(Varchar2)</td>
</tr>
<tr>
<td>Area_Acre</td>
<td>(Number)</td>
</tr>
<tr>
<td>Area_Thousand</td>
<td>(Number)</td>
</tr>
</tbody>
</table>

**Ownership Nested Table (Owners), level-1**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OwnerSLNo</td>
<td>(Number)</td>
</tr>
<tr>
<td>OwnerName</td>
<td>(Varchar2)</td>
</tr>
<tr>
<td>OwnerGrType</td>
<td>(Varchar2)</td>
</tr>
<tr>
<td>OwnerFather</td>
<td>(Varchar2)</td>
</tr>
<tr>
<td>OwnerAddress</td>
<td>(Varchar2)</td>
</tr>
<tr>
<td>OwnerPart</td>
<td>(Number)</td>
</tr>
<tr>
<td>Remarks</td>
<td>(Varchar2)</td>
</tr>
</tbody>
</table>

**Figure 4.50: Nested representation of CLMS database**

In our experiment we take three type of data name: Number, Varchar2, and Date.

**Figure 4-51: Thematic presentation nested relational design**
Table 4.19: Nested table record width

<table>
<thead>
<tr>
<th>Table name</th>
<th>Nested Level</th>
<th>Number column</th>
<th>Varchar column</th>
<th>Date column</th>
<th>Relational column</th>
<th>Size per Record(KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Landuses</td>
<td>3</td>
<td>3</td>
<td>50</td>
<td></td>
<td></td>
<td>0.0332</td>
</tr>
<tr>
<td>Dags</td>
<td>2</td>
<td>6</td>
<td>1150</td>
<td>1</td>
<td></td>
<td>0.1299</td>
</tr>
<tr>
<td>Owners</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>0.1056</td>
</tr>
<tr>
<td>Khatan_obj</td>
<td>1</td>
<td>7</td>
<td>881</td>
<td>2</td>
<td>2</td>
<td>0.2063</td>
</tr>
</tbody>
</table>

From the Table 4.19 per record size is applied in the “Khatian” table and get the following Table 4.20:

Table 4.20: Size of the entity sets for different number of records

- - - First Level (Khatian) - Second Level (Dags) - Second level (Owners) - Third Level (Land Uses) - 

<table>
<thead>
<tr>
<th>Record Count (Khatian)</th>
<th>First Level (Khatian)</th>
<th>Second Level (Dags)</th>
<th>Second level (Owners)</th>
<th>Third Level (Land Uses)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1167</td>
<td>960</td>
<td>384</td>
<td>384</td>
<td>128</td>
</tr>
<tr>
<td>8391</td>
<td>7936</td>
<td>2816</td>
<td>3072</td>
<td>768</td>
</tr>
<tr>
<td>9297</td>
<td>8128</td>
<td>3008</td>
<td>3072</td>
<td>960</td>
</tr>
<tr>
<td>17391</td>
<td>14336</td>
<td>6144</td>
<td>5120</td>
<td>2048</td>
</tr>
<tr>
<td>30430</td>
<td>22528</td>
<td>9216</td>
<td>8192</td>
<td>3072</td>
</tr>
<tr>
<td>35611</td>
<td>26624</td>
<td>10240</td>
<td>10240</td>
<td>3072</td>
</tr>
<tr>
<td>44296</td>
<td>231744</td>
<td>13312</td>
<td>11264</td>
<td>4096</td>
</tr>
</tbody>
</table>

Figure 4:52: Nested table size in respect to number of records
The table size increases almost linearly with respect to number of records in all levels. The first level nested relation (khatian) contains second level (dags and owners) and third level table is land-use table.

4.8 Conceptual Data Modeling Using Spatial Pictogram Extension (ER Model)

The ER model is one of the most popular high level conceptual data models. The spatial semantics is difficult to capture with ER diagrams. The first difficulty lies with geometric attributes are continuous, and the second difficulty lies with spatial relationships. Many extensions have been proposed to extend ER to make the conceptual modeling of spatial applications easier and more intuitive. Data modeling using pictogram is to provide more

![Figure 4.53: ER-Diagram for LIS Database](image-url)
constructs which capture the semantics of spatial applications and at the same time to keep the graphical representation simple. A pictogram is generally represented as a miniature version (icon) of an object inserted inside of a box. These pictograms are used to extend ER diagrams by inserting them at appropriate places in the entity boxes. A pictogram can be of a basic shape, a user defined shape, or any other possible shape. Figure 4.53 shows the E-R diagram of Land Information System (LIS) database. The corresponding design using pictogram is shown in Figure 4.54.

Figure 4.54: ER diagram for CLMS database with pictogram

**Convert ERD to tables**

The main objective to this step is to map the information requirements reflected in the entity relationship model into relational database design. We did map the simple
entities to tables, map attributes to columns and indicate required, unique and NULL attributes. We have identified primary keys and foreign keys from the realtions. From the above entity relationship diagram we get entity set and relation set which are converted into following tables:

1. khatian = (khatian_number, JL_NO, tax, remarks, total land)
2. khatian_owner = (khatian_number, JL_NO, owner_sl_no, owner name, address, owner part)
3. khatian_plot = (khatian_number, JL_NO, plot_no, total plot area, khatian plot area, Govt interest, Naba state)
4. plot_khatian_landuse = (khatian_number, JL_NO, plot_no, land use code land use, land use area)
5. Mutation_case = (khatian_number, JL_NO, case_no, case date, total land)
6. Applicants = (khatian_number, JL_NO, Applicant_sl_no, Applicant_name, address, applicant part)
7. Mauza = (Circle_code, JL_NO, Mouza Name)
8. Land transfer = (Khatian_no, Reference khatian_no, JL_No, plot_no, transfer land area)
9. Dolil = (Khatian_no, JL_No, Dolil Number, inheritance type, sub reg office)

Relational model for spatial data

For represent the Spatial data in a relational database model the Oracle spatial uses four database tables. This database structure is modeled on the first of three OpenGIS Features for SQL Implementation options, namely, using numeric SQL types for geometry storage. The relational model uses a table with a predefined set of columns of type NUMBER and one or more rows for each geometry instance. In the relational model have following characteristics:

Each geometry is required to be uniquely identified by a geometry identifier (GID) associating it with the other attributes of the feature. A complex geometry such as a polygon with holes would be stored as a sequence of polygon elements. All sub
elements of a multi-element polygon are wholly contained within the outermost element. One must directly access to the index tables. Database Structures for the Relational Implementation: The four tables, used to store and index geometries, are collectively referred to as a layer.

**Table 1-1 <layername>_sdoLayer Table**

<table>
<thead>
<tr>
<th>Sdo_ordcnt</th>
<th>Sdo_level</th>
<th>sdo_numtiles</th>
<th>sdo_maxlevel</th>
<th>sdo_coordsys</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
<td>&lt;varchar&gt;</td>
</tr>
</tbody>
</table>

1 sdo_maxlevel is an optional column. 2 sdo_coordsys is an optional column.

**Table 1-2 <layername>_sdoDim table or view**

<table>
<thead>
<tr>
<th>Sdo_dimnum</th>
<th>Sdo_lb</th>
<th>sdo_ub</th>
<th>sdo_tolerance</th>
<th>sdo_dimname</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
<td>&lt;varchar&gt;</td>
</tr>
</tbody>
</table>

**Table 1-3 <layername>_sdoGeom table or view**

<table>
<thead>
<tr>
<th>Sdo_gid</th>
<th>sdo_eseq</th>
<th>Sdo_etype</th>
<th>sdo_seq</th>
<th>sdo_x1</th>
<th>sdo_y1</th>
<th>...</th>
<th>sdo_xn</th>
<th>Sdo_yn</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
<td></td>
<td>&lt;number&gt;</td>
<td>&lt;number&gt;</td>
</tr>
</tbody>
</table>

**Table 1-4 <layername>_sdoIndex table**

<table>
<thead>
<tr>
<th>Sdo_gid</th>
<th>sdo_code</th>
<th>sdo_maxcode</th>
<th>sdo_groupcode</th>
<th>sdo_meta</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;number&gt;</td>
<td>&lt;raw&gt;</td>
<td>&lt;raw&gt;</td>
<td>&lt;raw&gt;</td>
<td>&lt;raw&gt;</td>
</tr>
</tbody>
</table>

Figure 4:55 Relational presentation of spatial data

To convert shape file to relational tables we have used “shp2Sdo” converter tools. From a particular shape file this tool produce five relational tables.

“Plots” GIS file, which consists of plot boundary shape (polygon) of the circle converted into following tables:

1. `plots_rel= (area, perimeter, plot_no, sheet_no, jl_no, mauza, thana, lucode, sdo_gid)`

   contains the attribute information and `sdo_gid` is link fields to the table.
2. plots_re1_sdolayer (sdo_ordcnt, sdo_level)
   The SDO_ORDCNT column is the total number of ordinates per row in the
   PLOTS_REL_SDOGEOM table. That is, the total number of data value
   columns, and not the number of points or coordinates.

3. plots_re1_sdodim = (sdo_dimnum, sdo_lb, sdo_ub,
   sdo_tolerance, sdo_dimname)
   Dimension information of the data layers

4. plots_re1_sdoindex = (sdo_gid, sdo_code) : Index code of the layer

5. plots_re1_sdogeom = (sdo_gid, sdo_eseq, sdo_etype, sdo_seq, sdo_x1, sdo_y1,
   sdo_x2, sdo_y2, sdo_x3, sdo_y3, sdo_x4, sdo_y4, sdo_x5, sdo_y5, sdo_x6,
   sdo_y6, sdo_x7, sdo_y7, sdo_x8, sdo_y8)
   The sdo_gid column is a unique numeric identifier for each geometry in a layer,
   The sdo_eseq column enumerates each element in a geometry, that is, the
   Element Sequence number, The sdo_etype column is the geometric primitive
   type of the element. The valid values are sdo_geom.point_type,
   sdo_geom.linestring_type, or sdo_geom.polygon_type (etype values 1, 2, and 3,
   respectively).

Roads GIS data converted into following tables:
roads = (length, ltype, road_name, thana, sub_area, ward, area_name, road_type,
   sdo_gid)
roads_re1_sdolayer = (sdo_ordcnt, sdo_level)
roads_re1_sdodim = (sdo_dimnum, sdo_lb, sdo_ub, sdo_tolerance, sdo_dimname)
roads_re1_sdoindex = (sdo_gid, sdo_code) : Index code of the layer
roads_re1_sdogeom = (sdo_gid, sdo_eseq, sdo_etype, sdo_seq, sdo_x1, sdo_y1,
   sdo_x2, sdo_y2, sdo_x3, sdo_y3, sdo_x4, sdo_y4, sdo_x5, sdo_y5, sdo_x6, sdo_y6,
   sdo_x7, sdo_y7, sdo_x8, sdo_y8)

Features GIS data converted into following tables:
features = (id, plot_no, jl_no, locode, land_use, sdo_gid)
features_re1_sdolayer = (sdo_ordcnt, sdo_level)
features_re1_sdodim = (sdo_dimnum, sdo_lb, sdo_ub, sdo_tolerance, sdo_dimname)
features_re1_sdoindex = (sdo_gid, sdo_code), Index code of the layer
features_rel_sldgeom = (sdo_gid, sdo_eyeq, sdo_etype, sdo_seq, sdo_x1, sdo_y1, sdo_x2, sdo_y2, sdo_x3, sdo_y3, sdo_x4, sdo_y4, sdo_x5, sdo_y5, sdo_x6, sdo_y6, sdo_x7, sdo_y7, sdo_x8, sdo_y8)

**Database size estimation for object - relational database**

To measure the database size in

Let,

\( W = \) Average attributes width

\( a_s = \) Number of the attributes in the database with string data type

\( a_i = \) Number of the attributes in the database with Integer data type

Integer data type width = 4 bytes.

\( a_f = \) Number of attributes in the database with float type

Float data type width = 16 bytes.

\( a_d = \) Number of attributes in the database with double type

double data type width = 32 bytes.

\( a_{sp} = \) Number of the attribute in the database with spatial data type

\( n = \) Number of records in the table

Size of the average database will be:

\[ S = n \left\{ \left( \sum_{i=1}^{a_s} W_i \right) + \left( 4 \times a_i \right) + \left( 16 \times a_f \right) + \left( 32 \times a_d \right) + \left( \sum_{i=1}^{a_{sp}} W_i \right) \right\} \]

In theoretical the table size estimate as above equation , in the table size calculation is given as below

Geometry data type is a composite data type contains Number data type and other composite data type.

For the Geometry Data type we can find in the following Table 4.21:

**Table 4.21: Data type and data column**

<table>
<thead>
<tr>
<th>Data Column</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDO_GTYPE</td>
<td>NUMBER</td>
</tr>
<tr>
<td>SDO_SRID</td>
<td>NUMBER</td>
</tr>
<tr>
<td>SDO_POINT</td>
<td>SDO_POINT_TYPE</td>
</tr>
<tr>
<td>SDO_ELEM_INFO</td>
<td>SDO_ELEM_INFO_ARRAY</td>
</tr>
<tr>
<td>SDO_ORDINATES</td>
<td>SDO_ORDINATE_ARRAY</td>
</tr>
</tbody>
</table>
In the Geometry data type SDO_POINT_TYPE contain following three columns:

<table>
<thead>
<tr>
<th>Data Column</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>NUMBER</td>
</tr>
<tr>
<td>Y</td>
<td>NUMBER</td>
</tr>
<tr>
<td>Z</td>
<td>NUMBER</td>
</tr>
</tbody>
</table>

* For polygon SDO_POINT_TYPE is null

SDO_ORDINATE_ARRAY and SDO_ELEM_INFO_ARRAY contains the points position of the Vertex, if we consider average number vertex for polygon is 50 then it will contain 100 Numbers. For average size for a Geometry we consider number average data type is 8 then, average size of the Geometry data type for polygon will be \(100 \times 8 \text{ (bytes)} = 800 \text{ bytes per record.}\)

In the spatial relation table we consider three tables *Plots* (ID, PLOT_NO, SHEET_NO, JL_NO, MAUZA, THANNA, LUCODE, AGRE_CODE, AREA, PERIMETER, GEOM)

**Table 4.22** Average record size for oplots table

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Attribute Number</th>
<th>Average Width (bytes)</th>
<th>Per records</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>8</td>
<td>16</td>
<td>128</td>
</tr>
<tr>
<td>Varchar2</td>
<td>2</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>SDO_Geometry</td>
<td>1</td>
<td>800</td>
<td>800</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>11</strong></td>
<td><strong>968</strong></td>
<td></td>
</tr>
</tbody>
</table>

**Table 4.23** Estimated size for Plots table

<table>
<thead>
<tr>
<th>Record Number</th>
<th>Per Record size (bytes)</th>
<th>Theoretical table Size (KB)</th>
<th>Table Size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3221</td>
<td>968</td>
<td>3045</td>
<td>3</td>
</tr>
<tr>
<td>16438</td>
<td>968</td>
<td>15539</td>
<td>15</td>
</tr>
<tr>
<td>21928</td>
<td>968</td>
<td>20729</td>
<td>20</td>
</tr>
<tr>
<td>27614</td>
<td>968</td>
<td>26104</td>
<td>25</td>
</tr>
<tr>
<td>46091</td>
<td>968</td>
<td>43570</td>
<td>43</td>
</tr>
<tr>
<td>52624</td>
<td>968</td>
<td>49746</td>
<td>49</td>
</tr>
</tbody>
</table>
Table 4.24 Actual Size from DB for Plots table

<table>
<thead>
<tr>
<th>Record Number</th>
<th>Size (KB)</th>
<th>Size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3221</td>
<td>2048</td>
<td>2</td>
</tr>
<tr>
<td>16438</td>
<td>12288</td>
<td>12</td>
</tr>
<tr>
<td>21928</td>
<td>15360</td>
<td>15</td>
</tr>
<tr>
<td>27614</td>
<td>19456</td>
<td>19</td>
</tr>
<tr>
<td>46091</td>
<td>31744</td>
<td>31</td>
</tr>
<tr>
<td>52624</td>
<td>35840</td>
<td>35</td>
</tr>
</tbody>
</table>

From the above table, following graph is plotted:

![Graph showing the comparison between theoretical and actual table sizes](image)

**Figure 4:56: Spatial polygon data storage requirements**

Table 02: oRoads (contains the road network of the demra circle, contains line information.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Number</th>
<th>Average Width (bytes)</th>
<th>Per records</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>3</td>
<td>16</td>
<td>48</td>
</tr>
<tr>
<td>Varchar2</td>
<td>5</td>
<td>50</td>
<td>250</td>
</tr>
<tr>
<td>SDO_Geometry</td>
<td>1</td>
<td>800</td>
<td>800</td>
</tr>
<tr>
<td>Total</td>
<td>9</td>
<td>1098</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.25: Average record size for oRoads table is as follows table:
Table 4.26: Estimated size for oRoads table

<table>
<thead>
<tr>
<th>Record Number</th>
<th>Per Record size (bytes)</th>
<th>Theoretical table Size (KB)</th>
<th>Table Size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>141</td>
<td>1098</td>
<td>151</td>
<td>0</td>
</tr>
<tr>
<td>479</td>
<td>1098</td>
<td>514</td>
<td>1</td>
</tr>
<tr>
<td>1585</td>
<td>1098</td>
<td>1700</td>
<td>2</td>
</tr>
<tr>
<td>3543</td>
<td>1098</td>
<td>3799</td>
<td>4</td>
</tr>
<tr>
<td>4124</td>
<td>1098</td>
<td>4422</td>
<td>4</td>
</tr>
<tr>
<td>4352</td>
<td>1098</td>
<td>4667</td>
<td>5</td>
</tr>
<tr>
<td>4415</td>
<td>1098</td>
<td>4734</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 4.27: Actual size for oRaods table

<table>
<thead>
<tr>
<th>Record Number</th>
<th>Size (KB)</th>
<th>Size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>141</td>
<td>128</td>
<td>0.125</td>
</tr>
<tr>
<td>479</td>
<td>256</td>
<td>0.25</td>
</tr>
<tr>
<td>1585</td>
<td>576</td>
<td>0.5625</td>
</tr>
<tr>
<td>3543</td>
<td>2048</td>
<td>2</td>
</tr>
<tr>
<td>4124</td>
<td>2048</td>
<td>2</td>
</tr>
<tr>
<td>4352</td>
<td>2048</td>
<td>2</td>
</tr>
<tr>
<td>4415</td>
<td>2048</td>
<td>2</td>
</tr>
</tbody>
</table>

From the above two table following graph is plotted
Table 03: oPoints (id,jl_no,mauza,thana,land_use, land_use_b, agrecode, agreuse, geom)

Table 4.28: Record size for points table (oPoints)

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Attribute Number</th>
<th>Average Width (bytes)</th>
<th>Per records</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>4</td>
<td>16</td>
<td>64</td>
</tr>
<tr>
<td>Varchar2</td>
<td>5</td>
<td>25</td>
<td>125</td>
</tr>
<tr>
<td>SDO_Geometry</td>
<td>1</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Total</td>
<td>10</td>
<td>237</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.29: Estimated size for points table

<table>
<thead>
<tr>
<th>Record Number</th>
<th>Per Record size (bytes)</th>
<th>Theoretical table Size (KB)</th>
<th>Table Size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>308</td>
<td>237</td>
<td>71</td>
<td>0.07</td>
</tr>
<tr>
<td>433</td>
<td>237</td>
<td>100</td>
<td>0.10</td>
</tr>
<tr>
<td>1111</td>
<td>237</td>
<td>257</td>
<td>0.25</td>
</tr>
<tr>
<td>7217</td>
<td>237</td>
<td>1670</td>
<td>1.63</td>
</tr>
<tr>
<td>8650</td>
<td>237</td>
<td>2002</td>
<td>1.96</td>
</tr>
<tr>
<td>8659</td>
<td>237</td>
<td>2004</td>
<td>1.96</td>
</tr>
<tr>
<td>8736</td>
<td>237</td>
<td>2022</td>
<td>1.97</td>
</tr>
</tbody>
</table>
Table 4.30: Actual size of the table oPoints from database

<table>
<thead>
<tr>
<th>Record Number</th>
<th>Size (KB)</th>
<th>Size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>308</td>
<td>128</td>
<td>0.13</td>
</tr>
<tr>
<td>433</td>
<td>128</td>
<td>0.13</td>
</tr>
<tr>
<td>1111</td>
<td>320</td>
<td>0.31</td>
</tr>
<tr>
<td>7217</td>
<td>2048</td>
<td>2.00</td>
</tr>
<tr>
<td>8650</td>
<td>2048</td>
<td>2.00</td>
</tr>
<tr>
<td>8659</td>
<td>2048</td>
<td>2.00</td>
</tr>
<tr>
<td>8736</td>
<td>2048</td>
<td>2.00</td>
</tr>
</tbody>
</table>

Figure 4.58: Spatial point data storage requirements

The theoretical and analytical databases sizes for spatial data are in the same train.
Chapter 5
Conclusion and Future Research

The design approaches of a database is a complex process. The database design strategies have significant effect on the overall performance of an application. The nature of application is also important factor to select database design option. The database design strategies for Online Transaction Processing (OLTP) system may not suitable for Decision Support System (DSS). On the OLTP system requires high throughput, intensive insert and update, large continuously growing data; on the other hand DSS requires quires on large data, accuracy and availability.

In the relational database design data types are scalar, normalization process remove duplicate tuple from entity set, but create a number of different entity sets, results increase the number of joining in the query processing time. The flat (universal) database design is a good option, because it has less number of joining, but this design requires high space and less integrity constraint.

To support complex data types and to map real life entity object oriented database have evolved. In the object oriented schema, user defined data types are produced, but the SQL are not supported for all types of queries especially when joining requires from different level on the object oriented approaches. To better representation objects in the database, object relational approaches has introduced. The nested relational model can represent relation domain in a column that can handle more relationship in a single table results compact presentation of data, but it suffers more in query processing time due to internal un-nesting operation. The binary relational model is very efficient in the point query for even very large database, but if we require full schema presentation then it need more joining and require more time to response. The following are the findings in our research work:
5.1 General Thesis Findings

- We have evaluated the effect of normalization on database size and found that space gain is not same for all normalization stages. The space gain from first normal form (1NF) to second normal form (2NF) and 2NF to third normal form (3NF) is not linear. Generally, we can say that 60-80% gain is achieved from 1NF to 2NF.

- Fully normalized design approach produces more tables in the database and in the most cases, more tables requires more join in the query processing. Joining is an expensive operation in database that effects on the query response time. De-normalization as required provides better performance then the fully normalized design. We have developed an algorithm that help the database designer to select which tables are suitable candidate for de-normalization for very large database schema. This algorithm uses the query frequency in the application. The association among tables are found by applying the Aprior algorithm.

- Significant performance gain is possible using partitioning. The horizontal partition of tables in different tablespace increase I/O speed and parallel query processing. In terms of space requirements, the partitioned design schema need little space more then un-partitioned design, even for large database. The response time gain is not same for partitioning of all the tables. Partition in relation table gives more impact than the partition in the original entity sets.

- The database parameter tuning has an upper bound on the overall performance of the applications. We have tuned the shared SQL area, data buffer cache, sort area and get significant improvement in the query response time.

- In the performance analysis of nested, universal and relational design schema we found that nested schema require less space compared to others but it suffers for larger query response time due to internal un-nesting process. The universal design schema have less response time for point, aggregate and
range query, but it requires more space and contain duplicate data. The relational schema is the middle position in terms of space and response time.

- The spatial data can represent in the database using relational and object relational design approaches. Spatial modeling in the traditional database (support only simple data types like number, char, date) is possible but requires more tables and inefficient for computation of any spatial operations. To make efficient support of spatial data in the database, spatial data types are introduced in the object relational database schema. we have examined the spatial data modeling both in relational and object relational database.

5.2 Future Research

The future expansion of this research is to explore the following issues:

- Design approaches can be considered for time series data. We have not considered any time series data in our present research.
- The design approaches and performance evaluation can be done for distributed database system will be considered in future.
- We have performed all the experiments in a single database, Heterogeneous or multidimensional system can be considered for future work.
Bibliography


[26] Shashi Sheker, Mark Coyle, Brajesh Goyal, Duen-Ren Liu, Shyamsundar Sarker: "Experiences with data models in Geographic Information Systems".


[29] Thanasis Hadzilacos, Nectaria Tryfona: "An Extended Entity-Relational model for geographic applications".


## Appendix

Table A.1: The list of SQL statements used in the Land Information System (LIS)

<table>
<thead>
<tr>
<th>SL. No</th>
<th>Query Name</th>
<th>Query Statements</th>
</tr>
</thead>
</table>
| 1      | Q1          | **Query:** Find the Landuse area and total applicable tax according to District name, Circle name and Land use name.  
SQL: SELECT tblLookup_District.DistName, tblLookup_Circle.CircleName, tblLookup_LandUse.LUDesc, tblLookup_LandUse.LUCode,  
sum(tblDag_LUse.Area / 10000) AS DagArea,  
sum((tblDag_LUse.Area + tblDag_LUse.Area_Dec / 10000) *  
tblLookup_TaxType.TaxRate * 100) AS TotalTax  
FROM tblLookup_Mouza INNER JOIN tblLookup_Circle ON  
tblLookup_District ON tblLookup_Thana.DistNo =  
tblLookup_District.DistNO INNER JOIN tblDag_LUse ON  
tblDag_LUse.Circle_Code = tblDag_LUse.Circle_Code AND  
tblDag_LUse.JLNO = tblDag_LUse.JLNO INNER JOIN  
tblDag_LUse ON tblDag_LUse.LUCode =  
tblDag_LUse.BagCode INNER JOIN tblDag_TaxType ON  
tblDag_TaxType.Tax_Code = tblDag_TaxType.Tax_Code Group by  
tblLookup_District.DistName, tblLookup_Circle.CircleName, tblLookup_LandUse.LUDesc, tblLookup_LandUse.LUCode |
| 2      | Q2          | **Query:** Find Khatian No, Dag (Plot) no, Land class, Owner name and Owner type of a particular mouza.  
SQL: SELECT tblKhatian.Circle_Code, tblKhatian.ThanaNO,  
tblKhatian.KhatianNO, tblKhd Dag.DagNo, tblKhDag.LandClass,  
tblKhOwner.OwnerSLNo, tblKhOwner.OwnerName,  
tblKhOwner.OwnerGrType  
FROM tblKhatian INNER JOIN tblKhd Dag ON tblKhatian.Circle_Code =  
tblKhDag.KhatianNO INNER JOIN tblKhOwner ON tblKhatian.Circle_Code =  
tblKhOwner.Circle_Code AND tblKhatian.ThanaNO =  
tblKhOwner.ThanaNO AND tblKhatian.JLNO = tblKhOwner.JLNO AND tblKhatian.KhatianNO = tblKhOwner.KhatianNO |
<table>
<thead>
<tr>
<th>SL. No</th>
<th>Query Name</th>
<th>Query Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Where tblKhatian.JLNO = 3</td>
</tr>
</tbody>
</table>
| 3     | Q3         | **Query:** Find District name, thana name, circle name, mouza name, sa dag(plot) no, government interest on the SA dag for a particular mouza.  
SQL: SELECT tblLookup_District.DistName, tblLookup_Thana.ThanaName, tblLookup_Circle.CircleName, tblLookup_Mouza.MouzaName, tblDag_Info.SADagNO, tblDag_Info.IsGovt  
WHERE tblLookup_Mouza.JLNO = 13 |
| 4     | Q4         | **Query:** Find Khatian, Plot details, ownership info and mutation land area.  
| 5     | Q5         | **Query:** Find all district name, thananame, mouza name and circle name  
SQL: SELECT tblLookup_Mouza.Circle_Code, tblLookup_Mouza.JLNO,
<table>
<thead>
<tr>
<th>SL. No</th>
<th>Query Name</th>
<th>Query Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Q1</td>
<td>No Name</td>
</tr>
<tr>
<td>3</td>
<td>Q3</td>
<td>Query: Find land use code used in all mouzas. SQL: Select distinct tblDag_LUse.LUCode FROM tblDag_LUse, tblLookup_LandUse WHERE tblDag_LUse.LUCode = tblLookup_LandUse.LUCode and tblDag_LUse.JLNO = &quot; &amp; curJLNO</td>
</tr>
<tr>
<td>4</td>
<td>Q4</td>
<td>Query: Find khitian no , ownership information , tax and owners remarks from khitian database. SQL: SELECT tblKhOwner.Circle_Code, tblKhOwner.JLNO, tblKhOwner.KhatianNO, tblKhOwner.OwnerSLNo, ISNULL(OwnerName, &quot;) + CHAR(13) + ISNULL(RTRIM(OwnerGrType), &quot;) + &quot;:&quot; + ISNULL(OwnerFname, &quot;) + CHAR(13) + ISNULL(OwnerAddress, &quot;) AS OwnerName, ROUND(OwnerPart, 4) AS OwnerPart, tblKhOwner.Revenue_Taka as Taka, tblKhOwner.Revenue_Pisa as Pisa, tblKhOwner.Remarks as OwnerRemarks FROM tblKhOwner INNER JOIN tblKhatian ON tblKhOwner.JLNO = tblKhatian.JLNO and tblKhOwner.KhatianNO = tblKhatian.KhatianNO WHERE tblKhOwner.Circle_Code = 1 and tblKhOwner.JLNO = &quot; &amp; cmbMouza.ItemData</td>
</tr>
<tr>
<td>5</td>
<td>Q5</td>
<td>Query: Find owner name, owner’s guardian name, owner’s address, owner’s area part in a khatian. SQL: SELECT o.Circle_Code, o.ThanaNO, o.JLNO, k.KhatianNO, o.OwnerSLNO, o.OwnerName, o.OwnerGrType, o.OwnerFname, o.OwnerAddress, o.OwnerPart, o.Remarks FROM tblApplicant o, tblCaseGInfo k WHERE o.Circle_code = k.Circle_code and o.ThanaNo = k.ThanaNo and o.JLNO = k.JLNO and o.CaseNO = k.CaseNO and not o.OwnerPart is null and k.CaseNO = '1234/1' And k.ThanaNo = 230 And k.JLNO = 17 And Circle_code = 1</td>
</tr>
<tr>
<td>SL. No</td>
<td>Query Name</td>
<td>Query Statements</td>
</tr>
<tr>
<td>-------</td>
<td>------------</td>
<td>------------------</td>
</tr>
</tbody>
</table>
| 9     | Q9         | **Query:** Find Khatian Case no, Khatian Type, dag no, Land area total of a khatian for all mouzas.  

SQL: SELECT DISTINCT tblKhatian.KhSerial, CASE tblKhatian.IsMut WHEN 0 THEN 'খুন' WHEN 1 THEN 'নামজাহাঁ' END AS KHType,  
tblKhatian.DagNo, tblKhatian.RSKhatianNO, tblKhatian.KhatianNO,  
tblKhatian.Total_Area_Acre + tblKhatian.Total_Area_Thousand / 10000 -  
ISNULL(rptTrasL.TArea / 10000, 0) AS KhArea, (tblKhatian.Land_Dag_Acre +  
isnull(tblKhatian.Land_Dag_Thousand, 0) / 10000) AS DagArea  
FROM tblKhatian INNER JOIN tblKhatDag ON tblKhatian.Circle_Code =  
tblKhatian.KhatianNO = tblKhatDag.KhatianNO |
| 10    | Q10        | **Query:** Find the all mutation case no, mutation plot no, SA plot no, reference plot number, khatian type, proposed land transfer area information.  

SQL: SELECT tblMutKhatDag.JLNO, tblMutKhatDag.CaseNO,  
tblMutKhatDag.DagSLNO, tblMutKhatDag.DagNo, tblMutKhatDag.SADagNO,  
tblMutKhatDag.RSKhatianNO, case ISNULL(tblMutKhatDag.RSKhType,0) WHEN 0 then 'খুন ক্ষতিহ্র' WHEN 1 then 'নামজাহাঁ ক্ষতিহ্র' END as KhatianType ,  
tblTransLandDetails.ProposedArea FROM tblMutKhatDag INNER JOIN  
tblTransLandDetails ON tblMutKhatDag.Circle_Code =  
tblTransLandDetails.Circle_Code AND tblMutKhatDag.JLNO = tblTransLandDetails.JLNO AND tblMutKhatDag.CaseNO =  
| 11    | Q11        | **Query:** Find khatina n , mutation case no, case date, revenue record no, tax information, total area under a khatian for a particular mouza.  

SQL: SELECT tblLookup_Circle.Circle_Code, tblKhatian.JLNO,  
tblKhatian.KhatianNO, tblKhatian.Jot_No, tblKhatian.CaseNo,  
tblKhatian.Case_Date as CaseDate,tblKhatian.IsMut,tblKhatian.RNO,  
(tblKhatian.Revenue_Taka + tblKhatian.Revenue_Pisa/100) as Tax,  
tblKhatian.KhSerial, tblKhatian.Total_Area_Acre,  
tblKhatian.Total_Area_Thousand  
FROM tblLookup_Circle INNER JOIN tblKhatian ON  
tblKhatian.Circle_Code = 1"
<table>
<thead>
<tr>
<th>SL. No</th>
<th>Query Name</th>
<th>Query Statements</th>
</tr>
</thead>
</table>
| 12     | Q12        | **Query:** Find circle name, district name, mouza name, khatian no, dag no, land class, land area, owner information like name, address, guardian information for a khatian.  
WHERE tblLookup_Circle.Circle_Code = 1 AND tblKhatian.KhatianNo = 23 |
| 13     | Q13        | **Query:** Find the circle name, district name, mouza name, plot information, land use code, land use area for a khatian.  
SQL: SELECT tblLookup_Circle.CircleName, tblLookup_District.DistName, tblLookup_Thana.ThanaName, tblLookup_Mouza.MouzaName, tblDag_LUse.DagNO, tblDag_LUse.LUCode, tblDag_LUse.LUDesc, tblDag_LUse.Acre, tblDag_LUse.Area_Dec  
FROM tblLookup_Circle INNER JOIN tblLookup_District ON tblLookup_District.DistNo = tblLookup_Circle.Circle_Code  
WHERE tblLookup_Circle.Circle_Code = 1 AND tblLookup_Mouza.ThanaNO = 23 |
<table>
<thead>
<tr>
<th>SL. No</th>
<th>Query Name</th>
<th>Query Statements</th>
</tr>
</thead>
</table>
| 14    | Q14        | **Query:** Find land revenues tax information for a particular mouza.  
**SQL:**  
SELECT DISTINCT tblLookup_TaxType.TaxType, tblLookup_TaxType.Tax_Code  
FROM tblLookup_TaxType INNER JOIN tblLookup_LandUse ON tblLookup_TaxType.Tax_Code = tblLookup_LandUse.TaxCode INNER JOIN tblDag_LUse ON tblLookup_LandUse.LUCode = tblDag_LUse.LUCode  
WHERE tblDag_LUse.JLNO = 1 ORDER BY tblLookup_TaxType.Tax_Code |
| 15    | Q15        | **Query:** Find mutation case information, mouza name, case no, case date, case approval status for a moza and a given date range of case approval date.  
**SQL:**  
SELECT tblCaseGInfo.JLNO, vMouzaInfo.MouzaName, tblCaseGInfo.KhatianNO, tblCaseGInfo.Kot_No, tblCaseGInfo.CaseNo, tblCaseGInfo.Case_Date AS CaseDate, tblCaseGInfo.Approval_Date  
WHERE (tblCaseGInfo.JLNO = 34) AND (tblCaseGInfo.Circle_Code = 1) AND (tblCaseGInfo.ThanaNO = 23)  
AND Approval_Date > '12-10-2005' AND Approval_Date < '20-12-2005' |
**Query:** Find tax and land use information for a particular thana.

**SQL:**
```
SELECT tblLookup_Circle.CircleName, tblLookup_District.DistName, tblLookup_Thana.ThanaName, 
tblLookup_Mouza.MouzaName, tblLookup_TaxType.TaxRate, 
tblLookup_TaxType.TaxType, 
tblLookup_LandUse.LUDesc, tblDagKhLuse.LUCode, tblDagKhLuse.DagNO, 
tblDagKhLuse.Area_Acre, tblDagKhLuse.Area_Thousand 
FROM tblLookup_LandUse INNER JOIN tblLookup_TaxType ON 
tblLookup_LandUse.TaxCode = tblLookup_TaxType.Tax_Code INNER 
JOIN tblLookup_Circle INNER JOIN tblLookup_Thana INNER JOIN 
ON tblLookup_LandUse.LUCode = tblDagKhLuse.LUCode 
WHERE tblLookup_Circle.CircleCode = 1 AND tblLookup_Thana.ThanaNO = 13 AND tblLookup_TaxType.TaxType = 3
```
Query: Find circle code, khatian no, owner name, owner type, land transfer from one khatian to another khatian for a mutation.

**SQL:**
```
```
21 Q21

**Query:** Find circle info, district name, circle name, mouza name, land area of a khatian.

**SQL:**
```
SELECT tblLookup_Circle.Circle_Code, tblLookup_Circle.CircleName, tblLookup_District.DistNO, tblLookup_District.DistName,
       tblLookup_Thana.ThanaNO, tblLookup_Thana.ThanaName, tblLookup_Mouza.JLNO, tblLookup_Mouza.MouzaName,
       tblKhatian.KhatianNO, tblKhatian.Total_Area_Acre,
       tblKhatian.Total_Area_Thousand
FROM tblKhatian INNER JOIN tblLookup_Mouza
   ON tblKhatian.ThanaNO = tblLookup_Mouza.ThanaNO AND
   tblKhatian.JLNO = tblLookup_Mouza.JLNO INNER JOIN
   tblLookup_Circle
   INNER JOIN tblLookup_Thana
   ON tblLookup_Mouza.ThanaNO = tblLookup_Thana.ThanaNO AND
   tblLookup_Mouza.JLNO = tblLookup_Thana.JLNO INNER JOIN
   tblLookup_District
   ON tblLookup_Thana.DistNO = tblLookup_District.DistNO WHERE (tblKhatian.Circle_Code = I) AND
   (tblKhatian.ThanaNO = 12) AND (tblKhatian.KhatianNO = N'1005') AND
   (tblKhatian.JLNO = 1)
```

22 Q22

**Query:** Fill mutation case no, case date, land transfer area, khatian owner name and owner guardian name for a particular mutation case.

**SQL:**
```
SELECT tblCaseGInfo.CaseNo, tblCaseGInfo.Case_Date, tblTransLandDetails.ProposedArca, tblKhOwner.OwnerName, tblKhOwner.OwnerFname
FROM tblTransLandDetails INNER JOIN tblCaseGInfo
   ON tblTransLandDetails.Circle_Code = tblCaseGInfo.Circle_Code AND
   tblTransLandDetails.JLNO = tblCaseGInfo.JLNO AND
   tblTransLandDetails.CaseNO = tblCaseGInfo.CaseNo INNER JOIN
   tblKhOwner
   ON tblCaseGInfo.Circle_Code = tblKhOwner.Circle_Code AND
   tblCaseGInfo.ThanaNO = tblKhOwner.ThanaNO AND
   tblCaseGInfo.JLNO = tblKhOwner.JLNO AND tblCaseGInfo.KhatianNO = tblKhOwner.KhatianNO
WHERE (tblCaseGInfo.CaseNo = '4934/2004')
```

23 Q23

**Query:** Find circle name, JLNO, district name, thana name, mouza name, khatian number, land case area, khatian land area for a particular khatian.

**SQL:**
```
SELECT tblLookup_Mouza.Circle_Code, tblLookup_Mouza.JLNO, tblLookup_District.DistName, tblLookup_Thana.ThanaName,
       tblLookup_Mouza.MouzaName, tblLookup_Circle.CircleName, tblKhatian.KhatianNO, tblKhDag.DagNo, tblKhDag.LandClass,
       tblKhDag.Land_Dag_Acre, tblKhDag.Land_Dag_Thousand
FROM tblKhatian INNER JOIN tblLookup_Mouza
   ON tblKhatian.ThanaNO = tblLookup_Mouza.ThanaNO AND
   tblKhatian.JLNO = tblLookup_Mouza.JLNO INNER JOIN
   tblLookup_Circle
   INNER JOIN tblLookup_Thana
   ON tblLookup_Mouza.ThanaNO = tblLookup_Thana.ThanaNO AND
   tblLookup_Mouza.JLNO = tblLookup_Thana.JLNO INNER JOIN
   tblLookup_District
   ON tblLookup_Thana.DistNO = tblLookup_District.DistNO WHERE (tblKhatian.Circle_Code = I) AND
   (tblKhatian.ThanaNO = 12) AND (tblKhatian.KhatianNO = N'1005') AND
   (tblKhatian.JLNO = 1)
```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
|   |   | Query: Find circle code, khatian number, owner information, owner part, khatian tax info, owner name for a khatian.
|   |   | SQL: SELECT tblKhOwner.Circle_Code, tblKhOwner.JLNO, tblKhOwner.KhatianNO, tblKhOwner.OwnerSLNo, ISNULL(OwnerName, '') + CHAR(13) + ISNULL(RTRIM(OwnerGrType), '') + ':' + ISNULL(OwnerFname, '') + CHAR(13) + ISNULL(OwnerAddress, '') AS OwnerName, ROUND(OwnerPart, 4) AS OwnerPart, tblKhatian.Revenue_Taka as Taka, tblKhatian.Revenue_Pisa as Pisa, tblKhOwner. Remarks as OwnerRemarks
|   |   | From tblKhOwner INNER JOIN tblKhatian ON tblKhOwner.JLNO = tblKhatian.JLNO AND tblKhOwner.KhatianNO = tblKhatian.KhatianNO
|   |   | WHERE tblKhOwner.Circle_Code = 1 AND tblKhOwner.JLNO = cmbMouza.ItemData(cmbMouza.ListIndex) AND tblKhOwner.KhatianNO = '123'
| 25 | Q25 | Query: Find circle code, district name, thana name, mouza name, mutation case number. Case date, mutation land area for a particular mutation case.
**Query:** Find circle, mouza, mutation case date, case number, approval date, land area, inherent type, case land area for a particular case date.

**SQL:**
```
SELECT tblLookup_Mouza.Circle_Code, tblLookup_Mouza.JLNO, tblLookup_Mouza.MouzaName, tblCaseGInfo.CaseNo, tblCaseGInfo.Case_Date, tblCaseGInfo.Approval_Date, tblCaseGInfo.Sp_Remarks, tblCaseGInfo.InherentType, tblCaseGInfo.Total_Area_Acre, tblCaseGInfo.Total_Area_Thousand
```

**Query:** Find mutation case number, plot number, khatian type, land transfer area and reference RS khatian number for a mutation case.

**SQL:**
```
SELECT tblMutKhDag.JLNO, tblMutKhDag.CaseNO, tblMutKhDag.DagSLNO, tblMutKhDag.DagNo, tblMutKhDag.SADagNO, tblMutKhDag.RSKhatianNO, case isnull(tblMutKhDag.RSKhatianNO,0) When 0 then 'ूर्व भिडियाम' When 1 then 'सामान्य बिडियाम' End as KhatianType, tblTransLandDetails.ProposedArea FROM tblMutKhDag INNER JOIN tblTransLandDetails ON tblMutKhDag.Circle_Code = tblTransLandDetails.Circle_Code AND tblMutKhDag.JLNO = tblTransLandDetails.JLNO AND tblMutKhDag.CaseNO = tblTransLandDetails.CaseNO AND tblMutKhDag.DagNo = tblTransLandDetails.DagNO AND tblMutKhDag.RSKhatianNO = tblTransLandDetails.RSKhatianNO WHERE tblMutKhDag.ThanaNO = '34' and tblMutKhDag.JLNO = '12' AND rtrim(ltrim(tblMutKhDag.CaseNo)) = '2345/2005'
```

**Query:** Find land transfer area, owner name, khatian number, owner name, case date, mutation case number for a mutation case.

**SQL:**
```
FROM tblMutKhDag INNER JOIN tblCaseGInfo ON
```

### Analysis
- **Table:** The tables `tblCaseGInfo`, `tblLookup_Mouza`, `tblMutKhDag`, `tblMlItKItDag`, `tblKhatian`, and `tblTransLandDetails` are used to store various pieces of information such as case details, land transfer details, and khatian information.
- **Queries:** The provided queries demonstrate how to retrieve specific data using SQL JOIN operations. The JOINs are used to link the tables based on common fields like `Circle_Code`, `ThanaNO`, and `JLNO` to ensure accurate data retrieval.
- **Data Retrieval:** The queries are designed to find specific information related to circles, mouzas, mutation cases, and land transfers, facilitating detailed analysis and reporting.

### Conclusion
The document appears to be a page from a technical or administrative document, likely related to real estate or property management, given the context and the queries provided. The queries seem to be part of a database retrieval process, possibly for an administrative or legal setting where detailed records of land transfers and property-related information are maintained.
Query: Find owner information, mutation case and mutation land class of particular mutation case.

Query: Find mutation case information, case date, plot number land use and land class for a particular case number.


Query: Find location information, khatian no. Owner name and address for a particular khatian number.

33  Q33

**Query:** Find mutation case number, reference RS khatian number, total land area of the mutation land for a particular reference SA dag number.

**SQL:**
```sql
SELECT tblCaseGInfo.JLNO, tblCaseGInfo.CascNo, tblCaseGInfo.Casc_Date, tblMutKhDag.RSKhatianNO, tblMutKhDag.DagSLNO, tblMutKhDag.SADagNO, tblMutKhDag.Total_Dag_Acre, tblMutKhDag.Total_Dag_Thousand
FROM tblCaseGInfo INNER JOIN tblMutKhDag ON tblCaseGInfo.Circle_Code = tblMutKhDag.Circle_Code AND tblCaseGInfo.ThanaNO = tblMutKhDag.ThanaNO AND tblCaseGInfo.JLNO = tblMutKhDag.JLNO
WHERE (tblCaseGInfo.JLNO = 1) AND (tblCaseGInfo.CascNo = '11386/2004')
```

34  Q34

**Query:** Find a mutation case with owner name, owner address, owner proposed area for a particular mutation case.

**SQL:**
```sql
SELECT tblCaseGInfo.JLNO, tblCaseGInfo.CascNo, tblCaseGInfo.Casc_Date, tblApplicant.OwnerName, tblApplicant.OwnerGrType, tblApplicant.OwnerFname, tblApplicant.OwnerAddress, tblApplicant.OwnerPart
FROM tblCaseGInfo INNER JOIN tblApplicant ON tblCaseGInfo.Circle_Code = tblApplicant.Circle_Code AND tblCaseGInfo.ThanaNO = tblApplicant.ThanaNO AND tblCaseGInfo.JLNO = tblApplicant.JLNO AND tblCaseGInfo.CascNo = tblApplicant.CaseNO
WHERE (tblCaseGInfo.JLNO = 1) AND (tblCaseGInfo.CascNo = '11386/2004')
```

35  Q35

**Query:** Find location information of a dag with land class, total land area for a particular plot of a mouza.

**SQL:**
```sql
WHERE (tblKhDag.DagNo = 'Q33')
```
Query: Find khatian , plot, plot area, owner name, SA plot, reference khatian number, owner name, land transfer area for a particular plot.


Query: Find Khatian, Owner name, Land class, Land area for a particular khatian number.

Query: Find Owner information, mutation case information and mutated land area for a particular mutation case number.


Query: Find Land use, civil case, thana, Khatian, Government interest land for a particular khatian number.

WHERE (tblKhDag.Circle_Code = 1) AND (tblKhDag.KhatianNO = '12')
**Query:** Find Owner information with location info like circle name, mouza name, and also find the dag (plot) information for a Khatian.

**SQL:**
```
SELECT DISTINCT tblLookup_Mouza.Circle_Code, 
tblLookup_Mouza.JLNO, tblLookup_District.DistName, 
tblLookup_Thana.ThanaName, tblLookup_Mouza.MouzaName, 
tblLookup_Circle.CircleName, tblKhatian.KhatianNO, tblKhDag.DagNo, 
tblKhDag.LandClass, tblKhOwner.OwnerSLNo, tblKhOwner.OwnerName, 
tblKhOwner.OwnerFname
FROM tblLookup_Mouza INNER JOIN tblLookup_Circle ON 
JOIN tblLookup_Thana ON tblLookup_Mouza.DistNO = 
tblLookup_Thana.DistNo AND tblLookup_Mouza.ThanaNO = 
tblLookup_Thana.ThanaNO INNER JOIN tblLookup_District ON 
tblLookup_Thana.DistNo = tblLookup_District.DistNO INNER JOIN 
tblKhatian ON tblLookup_Mouza.ThanaNO = tblKhatian.ThanaNO AND 
tblLookup_Mouza.JLNO = tblKhatian.JLNO INNER JOIN tblKhDag ON 
tblKhDag.ThanaNO AND tblKhatian.JLNO = tblKhDag.JLNO AND 
tblKhatian.KhatianNO = tblKhDag.KhatianNO INNER JOIN tblKhOwner ON 
tblKhatian.ThanaNO = tblKhOwner.ThanaNO AND tblKhatian.JLNO = 
tblKhOwner.JLNO AND tblKhatian.KhatianNO = tblKhOwner.KhatianNO
WHERE (tblLookup_Mouza.Circle_Code = 1) AND 
(tblLookup_Mouza.JLNO = 23) AND (tblKhatian.KhatianNO = 'N107')
```

**Query:** Find the mutation case applicant information of a mouza.

**SQL:**
```
SELECT tblLookup_Mouza.Circle_Code, tblLookup_Mouza.JLNO, 
tblLookup_District.DistName, tblLookup_Thana.ThanaName, 
tblLookup_Mouza.MouzaName, tblLookup_Circle.CircleName, 
tblCaseGInfo.CaseNo, tblCaseGInfo.Case_Date, tblApplicant.OwnerSLNo, 
tblApplicant.OwnerName, tblApplicant.OwnerGrType, 
tblApplicant.OwnerFname, tblApplicant.OwnerAddress
FROM tblLookup_Mouza INNER JOIN tblLookup_Circle ON 
JOIN tblLookup_Thana ON tblLookup_Mouza.DistNO = 
tblLookup_Thana.DistNo AND tblLookup_Mouza.ThanaNO = 
tblLookup_Thana.ThanaNO INNER JOIN tblLookup_District ON 
tblLookup_Thana.DistNo = tblLookup_District.DistNO INNER JOIN 
tblCaseGInfo ON tblLookup_Mouza.Circle_Code = tblCaseGInfo.Circle_Code AND tblLookup_Mouza.ThanaNO = tblCaseGInfo.ThanaNO AND 
tblLookup_Mouza.JLNO = tblCaseGInfo.JLNO INNER JOIN 
tblApplicant ON tblCaseGInfo.Circle_Code = tblApplicant.Circle_Code AND 
tblCaseGInfo.ThanaNO = tblApplicant.ThanaNO AND tblCaseGInfo.JLNO = 
tblApplicant.JLNO AND tblCaseGInfo.CaseNo = tblApplicant.CaseNo
```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>WHERE (tblLookup_Mouza.Circle_Code = 1) AND (tblLookup_Mouza.JLNO = 23)</th>
</tr>
</thead>
</table>
| 42 | Q42 | **Query:** Find land use and land tax information  
**SQL:** SELECT tblLookup_LandUse.LUCodc, tblLookup_LandUse.LUDesc, tblLookup_LandUse.TaxCode, tblLookup_TaxType.TaxRate FROM tblLookup_LandUse INNER JOIN tblLookup_TaxType ON tblLookup_LandUse.TaxCodc = tblLookup_TaxType.TaxCode |
| 43 | Q43 | **Query:** Find district information, Land use, Khatian number, mouza name, land use, land use area of th khatian  