Detailed Data Modelling: Attribute Collection and Normalisation of Data

The objective of detailed data modelling is to develop a detailed data structure that:

- has stability, minimum redundancy, and is flexible to allow for future change
- can be used as the basis for physical file and database design
- reflects the actual data requirements of the system

To expand the conceptual data model, we need to identify and describe the details of the entities and relationships:

- Attributes (data elements) of entities and relationships are identified
- The attributes should be independent of implementation technology
- Each attribute should represent a single “fact”
- An organisation-wide perspective should be adopted to ensure minimum redundancy and inconsistency and to facilitate data sharing
Detailed Data Modelling

Detailed data modelling aims to identify and describe attributes, convert ER models to relations, and to normalise the relations to ensure that they are well structured.

Techniques:
- attribute collection
- convert ER models to relations
- normalisation
- convert to data structure diagram
- Data Dictionary

Attributes

An attribute is a named property or characteristic of an entity that is of interest to the organisation.

Use an initial capital letter followed by lower case letters in naming attributes.
- eg Date, First Name, Suburb, Account No,

Each attribute name must be unique and distinctive.

Identifying Attributes

There are three main sources of attributes:
- data to support essential user functions
- data to support current operations
- data to measure performance against objectives
Identifying Attributes

Data to support all essential user activities identified as system requirements:
- Involve all potential users
- Include future data requirements of users
  eg. The inventory control function needs:

  | ITEM | Item Number | Item Description | Item Location | Item Quantity-on-hand | Item Re-order Quantity |

Data to support current operations:
- examine all forms, documents, reports and files (computerised and manual) used in the current system
- check with users for accurate definitions of attributes
- ensure all attributes identified are still required in the new system

Identifying Attributes

Data to measure business objectives and performance:
- ensure that data necessary to measure performance against objectives is identified
  eg. Objective: stock should be held in the warehouse for no more than seven days

  | Requires | Date Of Delivery | Date Of Despatch for each Item |
Documenting Attributes

- All attributes should be defined and described in the Data Dictionary
- Information should include:
  - name
  - description
  - format, precision, example values
  - domain of values, range of values
  - synonyms
  - method of derivation
  - validation constraints
  - entities which this attribute describes

Attribute Names

- Attribute names should be clear, unambiguous, and meaningful
- Each attribute name should be unique
  - Price, Retail Price, Cost Price for Item
  - Date, Delivery Date of Purchase Order
  - Name, First Name, Last Name of Member
  - Quantity-on-hand, Quantity Sold, Qty Ordered

Potential problem areas are:

- Synonyms
  - some attributes may be synonyms of others
  - e.g. retail price, sale price

- Homonyms
  - some attributes may be homonyms of others
  - e.g. price (for retail price)
  - price (for wholesale price)

- Derived attributes
  - some attributes may be derived from others
  - e.g. total salary for department
Attribute Definition

- Each attribute should convey a single fact for ease of processing and querying

  e.g. Pay Code

  A  normal hourly
  B  normal salaried
  C  overtime hourly
  D  overtime salaried

  Pay Factor
  1  normal
  2  overtime

  Job Category
  X  hourly
  Y  salaried

- Avoid embedding extra information in ranges of values
  e.g. Invoice Number
  0000-1499  north-east region
  1500-2999  south-east region
  3000-4499  central region

- Avoid embedding extra information in identifiers of attributes
  e.g. Part Number (9 digits) consists of:
  Part Type Code  (1)
  Factory Code    (2)
  Part Seq Number (4)
  Year Of Manuf   (2)
Normalisation

• Normalisation is a process for converting complex data structures into simple, stable data structures in the form of relations.
• Data models consisting of normalised relations:
  • are robust and stable
  • have minimum redundancy
  • are flexible
  • are technology-independent (logical)

Normalisation

• Normalisation ensures that each attribute is attached to the appropriate relation
  • each attribute is contained in the relation which represents the real world system object or concept that the attribute describes or is a property of
  e.g. the attribute Student Name should be in the relation STUDENT which represents the real world object “student” of interest to a student records system

Normalisation

• Normalisation was originally developed as part of relational database theory by E.F. Codd (1970)
• Normalisation is accomplished in stages, each of which corresponds to a “normal form”
• Originally, Codd defined first, second and third normal forms. Third normal form is adequate for most business applications.
• Later extensions include Boyce-Codd, 4th, 5th and domain-key normal forms.
The Relational Database Model

- The Relational Database Model represents data in the form of tables or relations
- Important concepts are:
  - relation
  - primary key
  - foreign key
  - functional dependency

Relation

- A relation is a named, two-dimensional table of data
- Each relation consists of a set of named columns and an arbitrary number of rows
- Each column corresponds to an attribute of the relation
- Each row corresponds to an instance (or record) that contains values for that instance

Example Relation

- A relation generally corresponds to some real world object or concept of interest to the system (similar to an entity) e.g.:

<table>
<thead>
<tr>
<th>Emp#</th>
<th>Name</th>
<th>Salary</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>1247</td>
<td>Adams</td>
<td>24000</td>
<td>Finance</td>
</tr>
<tr>
<td>1287</td>
<td>Smith</td>
<td>27000</td>
<td>MIS</td>
</tr>
<tr>
<td>1144</td>
<td>Jones</td>
<td>30000</td>
<td>Finance</td>
</tr>
</tbody>
</table>

Employee (Emp#, Name, Salary, Dept)
Properties of Relations

- Relational tables are tables in which:
  - data values are atomic (single-valued)
  - data values in columns are from the same domain
  - each row in the relation is unique
  - the sequence of columns is insignificant
  - the sequence of rows is insignificant

Primary Key

- An attribute or group of attributes which uniquely identifies a row of a relation
  - Employee (Emp#, Name, Salary, Dept)
  - Order-item (Order#, Item#, Qty-ordered)
  - Book (Book#, ISBN#, Call#, Copy#)

- Entity integrity (Relational Database theory) requires that each relation has a non-null primary key

Primary Key

- Where several possible keys are identified, they are known as candidate keys - choose one to be the primary key
  - stability
  - meaning
  - value
  - eg EMPLOYEE (Emp#, Office#, Name)
  - PERSON (TaxFile#, Medicare#, Lic#, SSec#)

  …or assign a unique key
  - BOOK (Book#, ISBN, Call#, Title)
Foreign key

- A foreign key is an attribute in one relation that is also a primary key in another relation.
- The referential integrity constraint (relational database theory) specifies that if an attribute value exists in one relation then it must also exist in a linked relation.
- A foreign key must satisfy referential integrity.

Foreign Key

- In the example below, if a given Dept# exists in an Employee relation then that Dept# must exist in the Department relation.

Employee (Emp#, Name, Salary, Dept#)

Department (Dept#, Dname, Budget)

Functional Dependency

- A functional dependency is a particular relationship between attributes in a relation.
- For any relation R, attribute B is functionally dependent on attribute A if each value of A has only ONE value of B associated with it, i.e., if for every valid instance of A, that value of A uniquely determines the value of B.

A identifies B
A ——— B
Emp # ——— Emp-name
Emp # ——— Salary
Normalisation to Third Normal Form (3NF) is accomplished in 3 steps each corresponding to a basic normal form.

- A normal form is a state of a relation that can be determined by applying simple rules concerning dependencies within that relation.

Each step of the normalisation process is applied to a single relation in sequence so that the relation is converted to 3NF.

Steps in Normalisation

1. First Normal Form (1NF)
   - Unnormalised form
   - Remove repeating groups

2. Second Normal Form (2NF)
   - First Normal Form (1NF)
   - Remove partial dependencies

3. Third Normal Form (3NF)
   - Second Normal Form (2NF)
   - Remove transitive dependencies

Well Structured Relations

- A well structured relation contains a minimum amount of redundancy and allows users to insert, modify, and delete rows in a table without errors or inconsistencies (known as anomalies).

- Three types of anomaly are possible:
  - Insertion
  - Deletion
  - Modification

- 3NF relations are considered to be well structured relations.
First Normal Form 1NF

- A relation is in 1NF if it contains no repeating data: the value of the data at the intersection of each row and column must be single-valued
- Remove any repeating groups of attributes to convert a relation to 1NF (key of the removed group is a composite key)

Order (Order#, Customer#, (Item#, Desc, Qty))

Order-Item (Order#, Item#, Desc, Qty)
Order (Order#, Customer#)

Second Normal Form 2NF

- A relation is in 2NF if it is in 1NF and every non-key attribute is fully functionally dependent on the primary key
- A partial dependency exists if one or more non-key attributes are dependent on only part of a composite primary key
- Remove any partial dependencies to convert a 1NF relation to 2NF
- If the primary key consists of only one attribute or there are no non-key attributes, then a 1NF relation is automatically in 2NF

Second Normal Form 2NF

Remove Partial Dependencies
- a non-key attribute cannot be identified by part of a composite key

Order (Order#, Item#, Desc, Qty-ordered)
Order-Item (Order#, Item#, Qty-ordered)
Item (Item#, Desc)
If a relation has a composite key, it must be checked for dependencies within the key to determine whether they should be retained e.g.

DEPT (Dept#, Dept-name, (Emp#, Emp-name))

DEPT (Dept#, Dept-name), DEPT-EMP (Dept#, Emp#)

But DEPT-EMP (Dept#, Emp#, Emp-name)

Remove Dept# from the primary key and make it a foreign key

EMP (Emp#, Emp-name, Dept#)

---

Partial Dependency Anomalies

### Partial Dependency Anomalies

<table>
<thead>
<tr>
<th>Order#</th>
<th>Item#</th>
<th>Item-desc</th>
<th>Qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>873</td>
<td>nut</td>
<td>2</td>
</tr>
<tr>
<td>28</td>
<td>402</td>
<td>bolt</td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>873</td>
<td>nut</td>
<td>10</td>
</tr>
<tr>
<td>30</td>
<td>495</td>
<td>washer</td>
<td>50</td>
</tr>
</tbody>
</table>

**UPDATE**: change item-desc in many places

**DELETE**: data for last item lost when last order for that item is deleted

**CREATE**: cannot add new item until it is ordered

---

Partial Dependency Anomalies

<table>
<thead>
<tr>
<th>Order#</th>
<th>Item#</th>
<th>Qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>873</td>
<td>2</td>
</tr>
<tr>
<td>28</td>
<td>402</td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>873</td>
<td>10</td>
</tr>
<tr>
<td>30</td>
<td>405</td>
<td>50</td>
</tr>
</tbody>
</table>

**DELETE**: delete last order for item, but item remains

**UPDATE**: add new item at any time

**CREATE**: change item description in one place only

<table>
<thead>
<tr>
<th>Item#</th>
<th>Desc</th>
</tr>
</thead>
<tbody>
<tr>
<td>873</td>
<td>nut</td>
</tr>
<tr>
<td>402</td>
<td>bolt</td>
</tr>
<tr>
<td>495</td>
<td>washer</td>
</tr>
</tbody>
</table>
Third Normal Form 3NF

- A relation is in 3NF if it is in 2NF and no transitive dependencies exist.
- A transitive dependency is a functional dependency between two or more non-key attributes.
- Remove any transitive dependencies to convert a 2NF relation to 3NF.

Transitive Dependency Anomalies

<table>
<thead>
<tr>
<th>Emp#</th>
<th>Emp-name</th>
<th>Dept#</th>
<th>Dname</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Smith</td>
<td>05</td>
<td>MIS</td>
</tr>
<tr>
<td>20</td>
<td>Jones</td>
<td>07</td>
<td>Finance</td>
</tr>
<tr>
<td>25</td>
<td>Smith</td>
<td>07</td>
<td>Finance</td>
</tr>
<tr>
<td>30</td>
<td>Black</td>
<td>08</td>
<td>Sales</td>
</tr>
</tbody>
</table>

UPDATE: change dept name in many places
DELETE: data for dept lost when last employee for that dept is deleted
CREATE: cannot add new dept until an employee is allocated to it.
Transitive Dependency Anomalies

<table>
<thead>
<tr>
<th>Emp#</th>
<th>Ename</th>
<th>Dept#</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Smith</td>
<td>D5</td>
</tr>
<tr>
<td>20</td>
<td>Jones</td>
<td>D7</td>
</tr>
<tr>
<td>25</td>
<td>Smith</td>
<td>D7</td>
</tr>
<tr>
<td>30</td>
<td>Black</td>
<td>D8</td>
</tr>
</tbody>
</table>

- Delete last emp in dept, but dept remains
- Add new dept at any time
- Change dept name in one place

Department

<table>
<thead>
<tr>
<th>Dept#</th>
<th>Dname</th>
</tr>
</thead>
<tbody>
<tr>
<td>D5</td>
<td>MIS</td>
</tr>
<tr>
<td>D7</td>
<td>Finance</td>
</tr>
<tr>
<td>D8</td>
<td>Sales</td>
</tr>
</tbody>
</table>

Normalisation to 3NF

- A relation is normalised if all attributes are fully functionally dependent on the primary key
  - Remove repeating groups
  - Remove partial dependencies
  - Remove transitive dependencies

ER Diagrams and Relations

- Transforming an ER diagram into normalised relations, and then merging all the relations into one final, consolidated set of relations can be accomplished in four steps:
  1. Represent entities as relations
  2. Represent relationships as relations
  3. Normalise each relation
  4. Merge the relations
Entities and Relations

- Each entity in the ER model is transformed into a relation e.g.

  CUSTOMER

  BECOMES
  Customer (Customer-no, Name, Address, City, State, Postcode, Discount)

Relationships and Relations

1. Binary Relationships (1:N, 1:1)

   CUSTOMER \( \rightarrow \) ORDER

   Customer (Customer-no, Name, Address, City, State, Postcode, Discount)
   Order (Order-no, Order-date, Promised-date, Customer-no)

Relationships and Relations

1. Binary Relationships (1:N, 1:1)

   - For 1:M, add the primary key of the entity on the 'one' side of the relationship as a foreign key in the relation that is on the 'many' side
   - For 1:1 relationship involving entities A and B, choose from
     - add the primary key of A as a foreign key of B
     - add the primary key of B as a foreign key of A
     - both of the above
Where we wish to know the quantity of each product on each order, i.e. this attribute is an attribute of the relationship 'requests':

Order (Order-no, Order-date, Promised-date)
Order Line (Order-no, Product-no, Quantity-ordered)
Product (Product-no, description, (other attributes))

For M:N, first create a relation for each for each of the entity types, and then create a relation for the relationship, with a composite primary key formed from the primary keys of the participating entity types.

EMPLOYEE

ITEM

MANAGES

ITEM

is a component of

EMPLOYEE

REPORTS TO

EMPLOYEE

EMPLOYEE (Emp-id, Name, Birthday, Manager-id)

ITEM (Item-no, Name, Cost)

ITEM-BILL (Item-no, Component-no, Quantity)

(1:N)

(1:N)

(M:N)
4. IS-A Relationship (Class-Subclass or generalisation)

PROPERTY

BEACH PROPERTY

MOUNTAIN PROPERTY

Property (Street-address, City-state-postcode, No-rooms, Typical-rent)
Beach_Property (Street-address, City-state-postcode, Distance-to-beach)
Mountain_Property (Street-address, City-state-postcode, Skiing)

Merging Normalised Relations

- During the normalisation process two or more relations with the same primary key may appear
- The set of 3NF relations must not contain any duplicate data
- Relations with the same primary key should be merged

Normalisation of Relations

- Synonyms
  - Two or more attributes may have different names but the same meaning
  - Either adopt one of the names as a standard or choose a third name
    - STUDENT1 (Student-id, Name, Phone-no)
    - STUDENT2 (VCE-no, Name, Address)
    - STUDENT (Student-id, Name, Address, Phone-no)
Normalisation of Relations

● Homonyms
● Two or more attributes may have the same name but different meanings
● To resolve the conflict, new attribute names need to be created

STUDENT1 (Student-id, Name, Address)
STUDENT2 (Student-id, Name, Phone-no, Address)
STUDENT (Student-id, Name, Phone-no, Campus-address, Permanent-address)

Normalisation of Relations

● Transitive dependencies
When two 3NF relations are merged, transitive dependencies may result

STUDENT1 (Student-id, Major)
STUDENT2 (Student-id, Advisor)
STUDENT (Student-id, Major, Advisor)

BUT

STUDENT1 (Student-id, Major)
MAJOR (Major, Advisor)

Data Structure Diagram DSD

● A set of 3NF relations may be converted to a simple diagrammatic form to begin physical database design

● The conversion is simple:
1. Draw a named rectangle for each relation
2. Draw a relationship line between rectangles linked by foreign keys with a “many” cardinality at the foreign key end of the relationship
CUSTOMER (Cust#, Cname, Phone number)
SALES ORDER (Sord#, Sord-date, Cust#)
SALES ORDER-ITEM (Sord#, Item#, Qty)
ITEM (Item#, Item-desc)

Data Structure Diagram DSD

• Eliminate Redundant Relationships

TOUR
(Tourcode, ...)

DEPARTURE
(Tourcode, depdate, ...)

BOOKING
(Booking#, ..., Tourcode, depdate)

Data Structure Diagram DSD

Summary

• Detailed data modelling involves:
  • collecting detailed attributes for each entity and relationship identified
  • converting ER models to relations
  • normalising the relations
  • merging relations from each user viewpoint
  • converting the normalised and merged relations to create a data structure diagram
Summary

Benjamin-Cummings, MA USA.  
Chapter 16

Whitten, J.L. & Bentley, L.D. and Dittman, K.C.,  
Chapters 7, 12