2. The Logical Data Model

As part of Requirements Analysis and Structuring, the data analyst builds a conceptual data model, which includes ER diagrams and process models. The next task is logical data modelling, which determines how the data will be stored and accessed logically. The interface, screens, reports and dialogs are designed along with the data structures that will hold the information required by the organisation. The final stage covered in the next section, compares or integrates the conceptual and logical data models to ensure the final draft is robust and complete.

2.1. The Relational Database Model

At this point, a particular logical database model must be selected. The main types of database that could be used nowadays are the relational model, and the object model.

Of these two, the relational model for database design is the most widely accepted and used model underlying commercial database systems. Originally introduced in the early 1970s, it succeeded the
Network and Hierarchical models, two early database configurations, and has been widely successful because of its sound mathematical basis and flexibility

The relational model follows the rules proposed by Dr. E.F. Codd of IBM, in his 1970 paper: "The Relational Model of Data for Large Shared Databanks" by Dr E.F.Codd, Communications of the ACM June 1970 pp 377-387

In his paper Codd used the mathematics of relational algebra to show how data could be systematically stored in two-dimensional tables. This paper defined the relational model very clearly. It was followed by large amounts of research, in IBM and elsewhere, to create relational databases that were fast enough and reliable enough to handle realistic workloads. This produced the systems that we use now, which can handle fast and effective work in large scale online multi-user systems.

The relational model is less coupled to the underlying physical representation of the data, as it relies on common and predictable attribute values (domain integrity). Because the domain of an attribute is known and enforced, information can be linked by relating attribute values together. Where common attribute values exist, relations can be “joined” by looking for records with matching attribute values.

<u>**The Basic Features of the Relational Model**</u>

Relational databases organise data into structures called relations. A relation is a set of attributes, each of which may take on a range of values (domain). The idea of a relation is a mathematical one, and strongly related to set theory.

The relational model handles data in two diemsional tables. These are known by a variety of terms, depending on where in the subject literature you look:

- Relational theory: relations are made up of tuples and attributes.
- Developers: tables are made up of records and fields.
- Users: tables are made up of rows and columns.

The method used to organise a set of attributes into a set of well structured relations is called *normalisation*. Normalising a set of attributes converts them into one of several, defined normal forms.

Thus one of the key processes for building a logical data model for a relational database is to create normalised relations.

2.2. Normalisation

Normalisation is carried out through a series of numbered or specially named stages. A normal form is a state which can be defined and checked by looking for dependencies between attributes. Each normal form has a definition which is mathematically based, but most easily checked for by testing for anomalies, as we will discuss further on.

2.2.1. Keys

A relation is not allowed to have duplicate tuples. This rule implies that each tuple of a relation (or record of a table) can be uniquely identified by some subset of its attribute values. The subset of attributes designated the *primary key* is to be used, first and foremost, to uniquely identify unique tuples. Note that there may be more than one combination of attributes that are unique for a tuple: these combinations are called *candidate keys*. The best primary key is:

1. unique
2. minimal
3. stable
4. relevant

If constructed from more than one attribute, the key is a *composite* key. The relational model defines another type of key: the *foreign* key. A relation has a foreign key to another relation if it has an attribute, or set of attributes, that form the primary key of another relation. Foreign keys are the mechanism by which relationships are represented in the relational model. The link between the primary key and foreign key is what allows the DBMS to locate and draw out the required information from the database.

Think! A relation, P, is related to another relation, Q. Why must the foreign key in Q be a unique primary key in P?

2.2.2. First Normal Form

First normal form is the most simple to test and correct for. A relation is in first normal form if all tuples consist of single-valued attributes. That is, a tuple cannot have a variable number of values for an attribute; each attribute must represent a single fact. A formal definition states

A relation is in first normal form if all attributes are single-valued.

Note that the definition of single-valued is ambiguous in some cases: for example, is it valid to have an attribute value that contains a list of strings? First normal form precludes relational systems from dealing with variable length tuples.

Example of First Normal Form

Order(<u>OrdID</u>, CustID, Date, CustName, {ProdID, Prod, Price, Qty})

The relation above has a variably repeating set of attributes {ProdID, Prod, Price, Qty}. To convert the relation above into first normal form, it is necessary to remove the repeating group into another relation. To ensure we can reconstruct the original relation, we repeat the attribute OrdID as a foreign key to provide a relational link.

Order(<u>OrdID</u>, CustID, Date, CustName)   OrdLine(<u>*OrdID*, ProdID</u>, Prod, Price, Qty)

Derived values may also be removed at this level of normalisation. Removal of derived values is not formally required until third normal form, but setting them aside at an early stage simplifies the normalisation process.

Think! Why might having multivalued attributes cause problems in retrieving or using data?

____

____

____

2.2.3. Second Normal Form

Second and third normal forms deal with functional dependencies within a relation. It is therefore important to gain a good understanding of functional dependence before attempting to apply the second and third normal forms to a set of attributes.

Functional Dependency

Normalisation relies on the concept of organising data into *functionally dependent* units. By ensuring all attributes that are inter-related are grouped together, we can localise the effect of any changes to an attribute value. The functionally dependent groups become the relations of the database.

Interdependence between relations is managed by allowing controlled repetition of attributes. The appearance of an attribute or attributes in more than one relation allows us to connect those relations together when necessary. The repetition is constrained by only allowing duplication of attributes that appear as primary keys.

An attribute A is functionally dependent on another attribute B, if A's value is uniquely determined by the value of B. That is, for each value of B, there is only one corresponding value of A. The following table illustrates a simple functional dependency:

ID (B)	**Name (A)**
123	Ella
234	John
345	Nina
456	John
567	Duke

Think! In the table above, Name is functionally dependent on ID. Is ID functionally dependent on Name? Why?

____

____

____

A more complex example, with multiple dependencies, is shown below:

OrdID	**CustID**	**CustName**	**LineNo**	**ProdID**	**ProdName**	**ProdDesc**
137	111	Joseph	1	37	2B4	Maths Book
248	222	Benjamin	1	41	1B5	Lined Notebook
359	111	Joseph	1	83	2E4	Address Book
460	333	Dan	1	17	1A3	Scrapbook
460	333	Dan	2	37	2B4	Maths Book
571	444	Reuben	1	41	1B5	Lined Notebook

In this example, the following dependencies exist:

OrdID -> CustID
CustID -> CustName
OrdID,LineNo -> ProdID
ProdID -> ProdName, ProdDesc

A relation in normal form has the property that

"All attribute values in a tuple are functionally dependent on the primary key"

This means that there is only one value that each attribute takes on for a given primary key. That value does not change unless the value of the primary key changes. The relation below is in normal form:

CustID	CustName	CustPh	CustType
333	Dan	456 6564	Retail
444	Reuben	478 8784	Wholesale

In this table, CustID -> CustName, CustPh, CustType. Changes to non-key attributes do not affect other relations. Changes to the primary key usually do affect other relations.

Full functional dependency is an extension to the concept of functional dependency, and applies to relations with composite (multi-attribute) keys.

An attribute A is fully functionally dependent on a set of attributes B, C if for every unique combination of B, C there is only one possible value of A. The following table gives a simple illustration:

Course	**Tutor**	**Studio**
Piano	Michael	4
Piano	Phil	5
Drums	Phil	4
Guitar	Jeff	2

In this example it appears Course, Tutor -> Studio. The value of Studio depends on both of the other two attributes. And example where full functional dependency does not exist is shown below:

Course	**Tutor**	**Studio**
Piano	Michael	4
Music Theory	Michael	4
Piano	Phil	2
Drums	Phil	2
Guitar	Phil	2

In this example it is reasonable to conclude that Tutor determines the location (Studio) rather than a combination of the Course and Tutor.

Think! Under what circumstances is an attribute functionally dependent on the key, but not fully functionally dependent on the key?

Preserving Data Integrity by Avoiding Anomalies

The purpose of normalisation is to structure the data in such a way that addition, deletion or modification of data will not cause anomalies. Examples of these anomalies will be given as we progress through the normalisation process, but a basic explanation of what is meant by each of these types of anomaly follows:

Addition (Insertion) Anomaly: where the data organisation prevents the addition of a new tuple/record

Deletion Anomaly: where the deletion of a tuple/record causes relevant data to be lost

Update (Modification) Anomaly: where changes to an attribute must be carried out in several places, making data integrity dependent on the operator's ability to locate all duplicates

Note that these anomalies occur when constraints are in place that are not required by business rules.

A formal definition of second normal form is

A relation is in second normal form if it is in first normal form, and all attributes are fully functionally dependent on the key.

Second normal form ensures that all attributes within a relation are fully functionally dependent on the primary key. If the primary key is atomic, any relation in first normal form will also be in second normal form. If the primary key is composite, each attribute should be tested for *partial dependence* on the key. If a partial dependency exists, there will be uncontrolled redundancy in the relation, and it will be subject to update anomalies.

Example of Second Normal Form

OrdLine(*OrdID*, ProdID, ProdName, Price, Qty)

The relation above will be subject to the following problems when the data is in use:

Update Anomaly	To make a change to a product name, we must update multiple records
Addition Anomaly	To enter information about a product, we must associate it with an order
Deletion Anomaly	If we delete all order lines with a particular product on them, we loose all information about the product

These anomalies arise because ProdName and Price are functionally dependent on only part of the primary key – ProdID. The anomalies will be avoided by moving the partially dependent attributes into a separate relation, again applying controlled redundancy to ensure we can reconstruct the data:

OrdLine(*OrdID, ProdID*, Qty) Product(*ProdID*, ProdName, Price)

It is clear in looking at this example that by applying normalisation, we are progressively moving towards cohesive relations that describe a single entity.

Think! Under what circumstance do you not need to check for second normal form for a relation?

____

2.2.4. Third Normal Form

Third normal form, like second normal form, deals with dependencies. Third normal form tests for interdependence between non-key entities (hence derived values will be specifically excluded if they have not been removed already). The formal definition of third normal form is

> *A relation is in third normal form if it is in second normal form, and there are no functional dependencies between non-key attributes.*

Non-key interdependence is formally called *transitive dependency*. A simple test for transitive dependency is to ask whether we can indirectly change the value of an attribute without altering the primary key. If altering a non-key attribute affects another non-key attribute, a transitive dependency exists.

Example of Third Normal Form

Order(<u>OrdID</u>, CustID, Date, CustName)

The relation above is clearly in second normal form. CustName is functionally dependent on OrdID, since, for any given value of OrdID there is only one possible value for CustName. However, CustName is only dependent on OrdID indirectly, as a result of being dependent on CustID. The relation above is subject to similar anomalies as the earlier example:

<u>Update Anomaly</u>	To make a change to a customer name, we must update multiple records
<u>Insertion Anomaly</u>	To enter information about a customer, we must associate them with an order
<u>Deletion Anomaly</u>	If we delete all orders with a particular customer on them, we loose all information about that customer

To avoid these anomalies, we again split the relation and duplicate the determining attribute

Order(*<u>OrdID</u>, CustID*, Date)   Customer(*<u>CustID</u>*, CustName)

The final set of relations in third normal form is

Order(*<u>OrdID</u>, CustID*, Date)
Customer(*<u>CustID</u>*, CustName)
OrdLine(*<u>OrdID, ProdID</u>*, Qty)
Product(*<u>ProdID</u>*, ProdName, Price)

2.2.5. Further Normal Forms

For single value primary keys, analysing data beyond 3rd normal form is not necessary. There are, however, situations where further analysis is necessary to avoid potential for anomalies. <u>The key objective of normalisation is to organise data so that its integrity is preserved when acted upon by an application</u>. Further normal forms should be applied with care, keeping this objective in mind.

A short description of other normal forms follow. Most are difficult to understand without an example.

Boyce-Codd Normal Form

A group of attributes in 3rd normal form may still suffer from anomalies if it has overlapping candidate keys. This is because an attribute may, while being fully functionally dependent on the whole key, have a particular link to part of the key that needs to be maintained and documented. The formal definition of Boyce-Codd normal form is

> *A relation is in Boyce Codd normal form if the only determinants are candidate keys*

Fourth and fifth normal forms only apply to sets of three or more attributes, all of which are key; otherwise any sets of attributes in BCNF are already in fourth and fifth normal form.

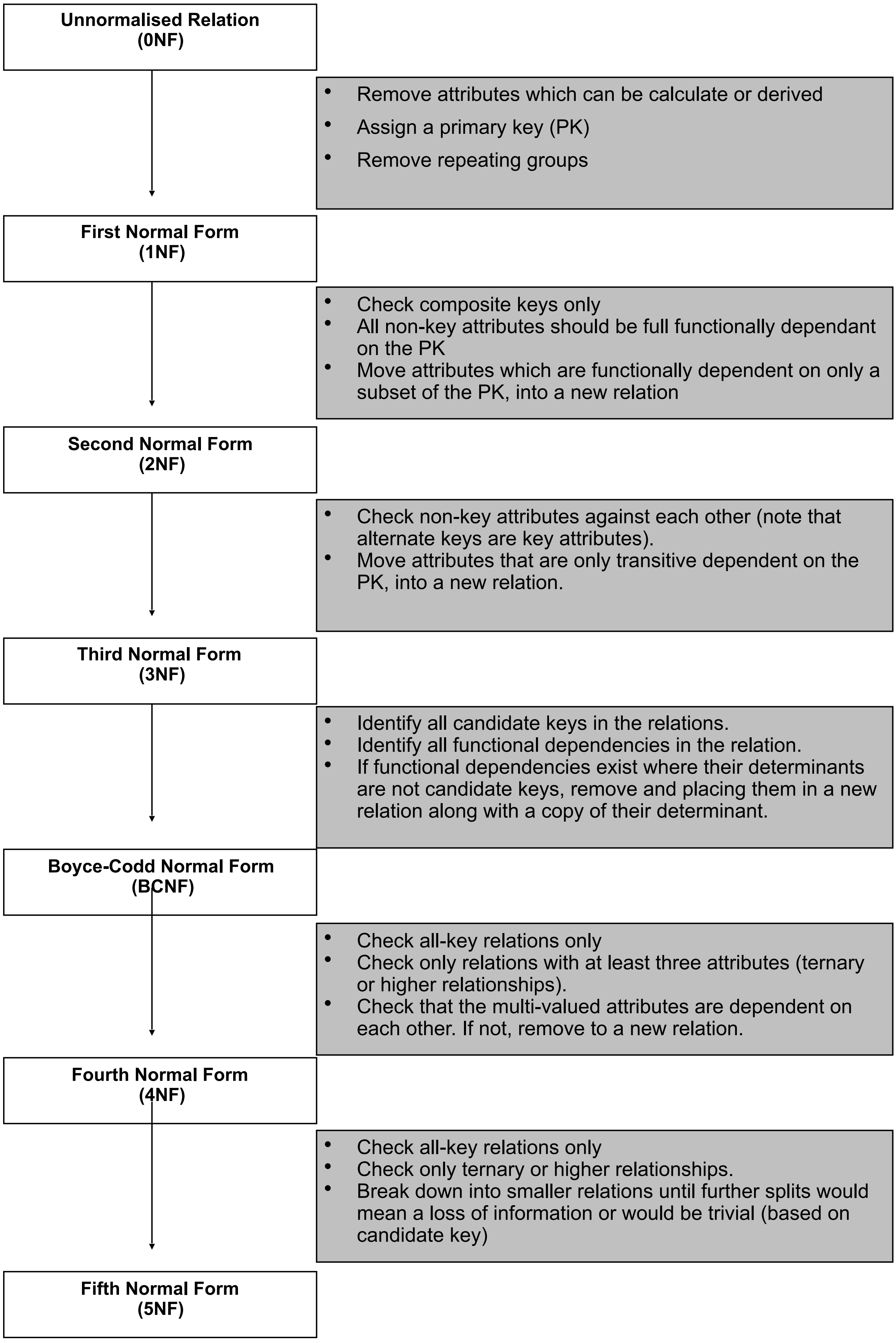
Fourth normal form resolves problems that arise when dealing with relationships of ternary (or higher) degree. Such relationships typically result in associative entities that have multi part keys. If all attributes are key, the attributes are automatically in BCNF, since the only determinant is the key; but there may still be redundant data.

Fifth normal form deals with join dependencies. A join dependency exists when a set of attributes cannot be broken down further without losing information. To check for a join dependency, simply ensure that the original set of attributes can be reconstructed from the smaller ones.

Check!

Having completed this section, can you

- give an overview of the logical data modelling phase?
- explain what insertion, deletion and update anomalies are and provide examples?
- identify functional dependencies in a relation?
- define and test for 1st, 2nd and 3rd normal form, reorganising data to achieve each normalised state?
- define Boyce-Codd normal form?
- complete the normalisation exercises for this section?



Unnormalised Relation (0NF)

- Remove attributes which can be calculate or derived
- Assign a primary key (PK)
- Remove repeating groups

First Normal Form (1NF)

- Check composite keys only
- All non-key attributes should be full functionally dependant on the PK
- Move attributes which are functionally dependent on only a subset of the PK, into a new relation

Second Normal Form (2NF)

- Check non-key attributes against each other (note that alternate keys are key attributes).
- Move attributes that are only transitive dependent on the PK, into a new relation.

Third Normal Form (3NF)

- Identify all candidate keys in the relations.
- Identify all functional dependencies in the relation.
- If functional dependencies exist where their determinants are not candidate keys, remove and placing them in a new relation along with a copy of their determinant.

Boyce-Codd Normal Form (BCNF)

- Check all-key relations only
- Check only relations with at least three attributes (ternary or higher relationships).
- Check that the multi-valued attributes are dependent on each other. If not, remove to a new relation.

Fourth Normal Form (4NF)

- Check all-key relations only
- Check only ternary or higher relationships.
- Break down into smaller relations until further splits would mean a loss of information or would be trivial (based on candidate key)

Fifth Normal Form (5NF)