Database Normalization Complete

JasonM (jasonm@accessvba.com)
Last Updated: 28 July 2004

Table Of Contents

Appendixes

1

INTRODUCTION

1.1 What Is Normalization?

Normalization, in the general sense, means to conform or reduce to a norm or standard[MW04]. For our purposes here, this means ensuring the data in the database meets a *normal form*. There are several of these normal forms, the most important (or "classical") are in the main body of this paper, with additional normal forms in Appendix E.

A table is *normalized* if it is in *first normal form* (1NF), which we discuss later. Although common to speak of tables that are not in a higher normal form -- such as 3NF -- as "unnormalized", this is strictly incorrect.

It is also common to speak of the entire database, or groups of tables being "normalized", but again, strictly speaking, it is the individual tables that are normalized. There are two points to expand on here:

1. If every table is in the same normal form, it is acceptable to say the database is in that form.
2. Some normal forms require considering a group of tables, rather than a single table.

An important point to make about the normalization process is that it is basically a formalization of common sense. Often the example tables will simply "look wrong"; the purpose of the normal forms is to help us formally identify the problem and determine how to fix it.

1.2 Why Should I Normalize?

[Cod71-2] gives six objectives of normalization:

1. To make it feasible to tabulate any relation in the database.
2. To obtain a powerful retrieval capability by means of a simple collection of operators
3. To free the collection of relations from undesirable insertion, update, and deletion dependencies
4. To reduce the need for restructuring relations when new data is introduced (increasing application lifespan)
5. To make the model for informative to users
6. To make the database neutral to query statistics

[Vin98-2] gives three goals of database design that normalization addresses:

1. Elimination of key-based update anomalies
2. Elimination of redundancy
3. Minimization of storage

Update anomalies occur when data is not changed in all places. This is the general type of anomaly, often broken down into three sub-types:

1. Insertion: Storing new data
2. Deletion: Removing existing data
3. Update: Changing existing data

The first two goals, elimination of anomalies and elimination of redundancy are closely related. When redundant data exists, this means multiple places must be updated, one for each occurrence of the data. It is by eliminating redundancy that we simplify update operations on the table. The last goal, minimization of storage, follows the first two. By eliminating redundancy, we reduce the amount of storage required to maintain the same information.

It is important to note that *duplicate* data is not the same a *redundant* data. For our purposes, redundant data is that data we can remove without a loss of information. In contrast, duplicate data is required to retain information – a simple example being key fields which two tables share in a relationship. In each table, the same values will be duplicated, but this is necessary to retain information and join the tables.

Finally, in [Cod74] the objectives are restated and outlined:

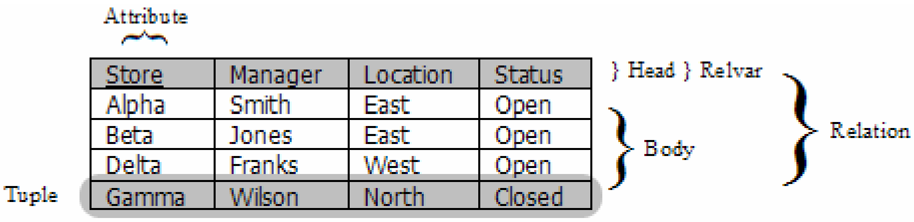
1. Provide a high degree of data independence
2. Provide a simple view of the data
3. Simplify to job of the database administrator
4. Introduce a theoretical foundation for database management
5. Merge fact retrieval and file management
6. Allow sets of data to be treated as operands, rather than being processed one element at a time

1.3 Terminology

We must introduce some terminology to help us understand normalization. The purpose of this section is to illustrate some basic concepts while introducing them. Some of the terms receive further treatment later, and Appendix C of this paper also offers a glossary.

1.3.1 Basic Terms

We begin by examining a sample table and providing relational terminology for each aspect of that table. Consider the following table in Fig. 1.1, called R:



Store	Manager	Location	Status
Alpha	Smith	East	Open
Beta	Jones	East	Open
Delta	Franks	West	Open
Gamma	Wilson	North	Closed

Fig 1.1 – Main Table Example

The table column headers represent a *relation variable*, or *relvar* [Dat02]. We notate this relvar like so:

R {Store, Manager, Location, Status}

This relvar has a *predicate*, a truth statement about its meaning, this relvar says:

"A store has a name, a manager, a location, and a status."

This is sometimes called the *intension*, or meaning of the table.

Rough Equivalents	
Attribute	Column
Cardinality	# of Rows
Degree	# of Columns
Domain	Column Type
Relation	Table
Relvar	Table Headers
Tuple	Row
Tuple Value	Cell

Fig 1.2

This relvar has 4 *attributes*, which correspond to the columns of the table. The table itself is a *relation value*, or simply *relation*; an instance of the relvar. If a row was inserted or deleted, it would be a different relation, but the same relvar. There is therefore a subtle, but sometimes important, difference between a relation and a relvar.

This is sometimes called the *extension*, or instantaneous value, of the table.

A relation consists of a *heading* and a *body*. The *heading* corresponds to the column names, and the body to the rows of the table, the actual data. We will discuss this in more detail when covering first normal form.

This *relation* has 4 *tuples*, each one corresponding to a row of the table, so we can say this relation has a *cardinality* of 4. Each tuple is a different *proposition* about the predicate of the relvar. For example, the first tuple asserts the following proposition is true: "The store named Alpha has the manager named Smith, is in the East location and has a status of Open."

Each tuple (and therefore the relation and relvar) has a *degree* of 4. That is, each tuple contains 4 *tuple values*, one for each attribute of the relvar. Each tuple value has a value from the *domain* of the corresponding attribute. That is, there is an acceptable set of values for the Store attribute, which may or may not be the same acceptable set of values for the Manager attribute. In this case it is not, as Store has a domain of store numbers, and Manager a domain of manager names.

In this paper, we assume the domains can be specified precisely. In a real-world *database management system* (DBMS), we may have to settle for a higher level of abstraction, such as saying Store has a domain of integers, and Manager a domain of character strings. Again, we discuss this further in our coverage of first normal form.

Finally, assume the following business rules are in effect:

1. A store has only one manager, and each manager manages only one store.
2. A store has only one location (but multiple stores can have the same location).
3. Each store in a location has the same status (but multiple locations can have the same status).

Knowing these rules, we can determine the keys of R. And keys are the subject of the next section.

1.3.2 Keys

A *key*, generally speaking, is some set of attributes that can uniquely identify a tuple. That is, no two tuples can share the same key – although they may share the same set of attribute, the combined values of those attributes will be unique for each tuple. (This does not exactly include *foreign keys*, but they are not important in our discussion.)

We will discuss four types of keys: *superkeys*, *candidate keys*, *primary keys* and *alternate keys*.

1.3.2.1 Superkeys

A *superkey* is a set of attributes that can be used to uniquely identify each tuple. Here are the superkeys of R:

{Store, Manager, Location, Status}
{Store, Manager, Location}
{Store, Manager}
Store
{Manager, Location, Status}
{Manager, Location}
Manager

Every relation must have at least one superkey, made up of all attributes. If this were not so, the relation would have duplicate tuples, which is not allowed, as we will see later. Most relations have several superkeys.

Generally, we are not interested in superkeys except that we use them to select candidate keys. However, there are some normal forms that are directly interested in superkeys.

1.3.2.2 Candidate keys

A *candidate key* contains only those attributes required to uniquely identify each tuple. That is, it is a superkey with no proper subset which is itself a superkey, it is irreducible. These are the candidate keys of R:

Store
Manager

Compare these candidate keys to the superkey {Store, Manager}. This superkey has two proper subsets, Store and Manager, both of which are superkeys. Because we do not need every attribute in {Store, Manager} to uniquely identify a tuple, {Store, Manager} is **not** a candidate key.

Just as we saw for superkeys, every relation must have at least one candidate key. Most relations have multiple candidate keys, but fewer candidate keys than superkeys.

An attribute contained in a candidate key is a *prime attribute*. An attribute not in any candidate key is a *nonprime attribute*. The prime attributes in R are Store and Manager, and the nonprime attributes are Location and Status.

Candidate keys are by far the most important type of key we will be concerned with in normalization.

1.3.2.3 Primary keys

We select a *primary key* from the candidate keys. In this case, we select Store and signify this by underlining the attribute name.

Every relation must have one and only one primary key. It is possible that a relation has only one superkey, only one candidate key, and only one primary key, and all three are the same, but this is not usually the case.

Although primary keys are very important in many areas, in normalization we are usually only interested in them because they are also candidate keys, and we often would like to have a relation that has only one candidate key, which therefore must be the primary key.

1.3.2.4 Alternate keys

The remaining candidate keys that we did not select as a primary key are the *alternate keys*. For the purposes of normalization, we are only interested in alternate keys because they are also candidate keys, and we would like to have a relation that doesn't have any alternate keys left after selecting a primary key.

Superkeys	Candidate keys	Primary key	Alternate key
Must have one or more	Must have one or more	Must have one and only one	May have zero or more
Rarely important in normalization	Very important in normalization	Rarely important in normalization	Rarely important in normalization
Selected from attributes	Selected from superkeys	Selected from candidate keys	"Leftover" from candidate keys

Fig 1.3

Just a few points to make before moving on:

1. The presence in a candidate key, **not** primary key determines whether and attribute is prime or nonprime. The similarity in names between *prime attribute* and *primary key* can be misleading.
2. In this paper, *key* without qualification means "a candidate key with one or more attributes".
3. A key having only one attribute is *simple*. A key having more than one attribute is *compound*. We can talk about a *compound primary key* as being a primary key consisting of two or more attributes.

1.3.3 Functional Dependencies

A *functional dependency* (FD) exists when one attribute value has one and only one value of a different attribute associated with it in the relation. For example, in Fig 1.1, the value of Location is determined by Store, that is if you know Store, you know Location.

Following the notation used in [Cod71], we use Store → Location to express this dependency. You can read this notation several ways, according to what you are trying to express or personal preference:

"Store determines Location"
"Location is *functionally dependent* on Store"
"Store is a *determinant* for Location"

If we want to show that there is **not** a FD between two attributes we use Location ↛ Store.

To generalize this we say A → B, where A and B are some attributes in the relation. Note that A and B could be sets of attributes, rather that single attributes. If we want to specifically show there is **not** a FD between two attributes, we can say A ↛ B. If we want to show A → B **and** B → A, we can say A ↔ B. Here are some examples from Fig 1.1:

Store → Location (if you know the value of Store, you also know the value of Location)
Location ↛ Store (however, if you know Location, you do not know Store)
Store ↔ Manager (if you know Store, you know Manager, and vice versa)

A FD always exists if B is a subset of A, and in that case it is a *trivial functional dependency*. To expand on this just a bit, this means if A and B are single attributes, then B **is** A, so in this case A → B is the same as saying A → A, or in Fig 1.1, Store → Store. This is obviously true, and we are usually not interested in trivial functional dependencies.

1.3.3.1 Nontrivial Functional Dependencies

We use R to illustrate. Consider the FD Store → Location. This means "Store determines Location", or if you know Store, you also know Location. By looking at Fig 1.1, if you know Store is "Alpha", then you also know Location is "East". If you know Store is "Beta", then you also know Location is "East". There is a one-to-one correspondence, *in a certain direction*, between Store values and Location values.

Now, consider the reverse: Location ↛ Store. If you know Location is "East", you **do not** know Store. It could be "Alpha" or it could be "Beta". There is **not** a one-to-one correspondence, *in a certain direction*, between Location values and Store values.

However, consider that Store → Manager and Manager → Store are both true. In this case, there is a one-to-one correspondence in *both* directions. We can notate this, then, as Store ↔ Manager. (This particular notation is not common – usually both FDs are expressed separately – but we cover it here for completeness.)

Throughout this paper we use f to mean the *set of all functional dependencies in the relation*. Here, then, is the set of all functional dependencies in R (the f of R), numbered for reference:

f_1. Store → Manager
f_2. Manager → Store
f_3. Store → Location
f_4. Location → Status

Notice that f is a more formal statement of the business rules. This has 2 interesting implications:

1. You can **not** properly normalize a table without knowing the underlying business rules.
2. You can **not** simply look at a table and determine what normal form it is (or isn't) in.

Now, not only does a FD describe a business rule, but by doing so, it also applies a *constraint* to the relation. That means no tuple can be inserted into the table which would make any FD in f invalid, or violate the constraint that a FD implies. Constraints are very important in the normalization process.

For example, consider f_3: Store → Location. This tells us that if you know Store, you also know Location. It means that any given store has one and only one Location. Keep this in mind, and consider what would happen if we tried to insert this tuple *t* into R: *t* (Alpha, Smith, West, Open)

If we allowed *t* to be inserted, the constraint f_3 requires would be violated. That is, store Alpha would have two locations and Store → Location would no longer hold. (There are other problems as well, such as the primary key would be duplicated.)

1.3.3.2 Irreducible Functional Dependencies

Consider f_2: Store → Location. Now let us add an attribute to the left hand side, Manager. (This is the *augmentation*, see Appendix B.) We then have a new FD:

f_5: {Store, Manager} → Location.

This new FD is true, but note that Manager is **not** required for the FD to hold. We could *reduce* the left hand side by removing SName and the FD would still be true. Therefore, {Store, Manager} → Location is **not** an *irreducible functional dependency*.

Closure

Recall that we said f is the set of all FDs in R. This isn't exactly true, because we can infer additional FDs based on the ones we are given. We really mean that f contains **a** set of FDs, from which we **can** infer all FDs, this "true" set of all FDs is the *closure* of f , and is notated f^+.

What we originally give as f is actually the *minimal closure*, the fewest FDs required to infer f^+.

However, we may then infer additional FDs not in that minimal closure.

Compare to: Store → Location, which **is** irreducible – with only one attribute on the left hand side, it must be! Location is *irreducibly dependent* on Store.

We are following the terminology in [Dat04] here. Many references call an irreducible FD a *full functional dependence*, and would call Location *fully functionally dependent* on Store.

2NF seeks to eliminate non-irreducible FDs.

1.3.3.3 Transitive Functional Dependencies

Let us now consider two FDs in f:

f_3: Store → Location
f_4: Location → Status

We can read the above as "Store determines Location, and Location determines Status". Again, we can infer a new FD:

f_6: Store → Status. (This is *transitivity*, again see Appendix B.)

This means if you know the value of Store, you know the value of Status. The FD Store → Status is a *transitive functional dependency*, because some attribute exists that is in "the middle". Yet another way is to say that if <a, b> and <a, c> are in R, then so is the <b, c>.

We often shorten the term transitive functional dependency to *transitive dependency*, abbreviate as TD, and notate as A (B, C). For example, we notate the Store → Location → Status TD like so: Store (Location, Status)

If a FD is not a TD, it is a *direct functional dependency*, often called a *nontransitive dependency*.

3NF seeks to eliminate TDs.

1.3.3.3.1 Strict Transitive Functional Dependencies

If A → B and B → C, but C ↛ B and B ↛ A, then A → C is a *strict transitive dependency*. Loosely speaking it is "one-way". In our example, the TD Store (Location, Status) is a strict transitive dependency.

This is because Store → Location → Status, but Status ↛ Location and Location ↛ Store. The transitivity is "one-way", and therefore strict.

Optimal 3NF is concerned with strict transitive dependencies.

1.3.3.3.2 Transitive Dependencies Contain Functional Dependencies

Interestingly, a FD is just a special case of a TD[Mit83], and the following two expressions are equivalent:

1. (FD) Store → Manager
2. (TD) Store (Manager, Manager)

In our transitive dependency notation, we are simply saying: "Store determines Manager and Manager determines Manager", the second half of which is a trivial FD, so we are left with "Store determines Manager", which the exact same thing the functional dependency notation expresses. Thus the two notations are equivalent, and it is shown that a TD can express a FD. Not every TD is a FD, but every FD **is** a special case of a TD.

Let us clarify these two seemingly conflicting issues:

1. Every FD is a special case of a TD.
2. Some FDs are direct (nontransitive) dependencies.

Simply stated, we don't generally consider those FDs that are a special case of TDs to **actually be** TDs for the purposes of 3NF (which is where the concept first becomes important).

FD is a TD

We mention this relationship between TDs and FDs, because many normal forms are based on finding a more general type of dependency and then handling the more general case. It is good, then, to become acquainted with the idea of a dependency that generalizes another dependency.

However, long after 3NF was introduced it was noticed that all FDs could be considered a type of TDs. (There are additional dependencies we introduce later; each one generalizing other dependencies, this concept is not limited to FDs and TDs.)

It may help to think of transitive dependencies in the form of A (B, B) as being *trivial transitive dependencies*, and then saying we aren't interested in those, except as "normal" functional dependencies.

1.4 Summary

In this introduction we have briefly discussed both what normalization is and why it is desirable. We introduced basic relational concepts in comparison to the familiar table, defined several types of keys, and explained functional dependencies in some detail.

The foundation is therefore in place for the next section, Basic Normal Forms.

2

BASIC NORMAL FORMS

2.1 Zero Normal Form (0NF)

Zero Normal Form isn’t a real normal form at all, but we can use the concept to express the idea that some data is not in 1NF. Occasionally this is called UNF, for UnNormalized Form.

This sort of data is also sometimes called an “unnormalized relation”, but while this conveys an idea, it’s a misleading one. This is because such data is not actually a relation at all [Pra91], since relational theory doesn’t deal with such data. It’s a bit confusing then to call something a relation when it is not!

So then, if data not in 1NF is not a relation, then it follows: *all relations are in 1NF* [Dat03].

And this is what “normalizing” data really means: to represent the data as a relation. Once in 1NF, data is normalized. We do not want to stop at 1NF, but we do want to start there…

2.2 First Normal Form (1NF)

First Normal Form (1NF)

A **table** is in 1NF if:

It is a direct and faithful representation of a relation.

We take this definition of 1NF from [Dat03], and since it differs from some common descriptions of 1NF, we want to examine it carefully.

A major point to make is that a table is a *representation* of a relation, not actually a relation (a picture of something is not actually that something, after all). Therefore, there are differences between a table and a relation, and the point of 1NF is to ensure that the table represents a relation (which by definition is already in 1NF) as faithfully as possible. It is the confusion between a table and a relation (or the representation with the actual), that causes many people great difficulty.

Here are 5 conditions [Dat03] gives which a table must satisfy:

1. The rows are not ordered.
2. The columns are not ordered.
3. There are no duplicate rows.
4. Every row/column intersection contains exactly one domain value and nothing else.
5. No irregular columns.

We now consider each of these points in turn:

1. The rows are not ordered.
This is often confusing, because the user may want the "first" or "last" or "top 10" or similar results. It also clear that a *table* has a first row, followed by a second row, and so forth. However, a *relation* is unordered, even if the table rows representing the tuples have some order imposed on them.

2. The columns are not ordered.
This is the same as the first point. Reading left to right; a *table* clearly has a first column, followed by a second column, and so forth. Again though, in a *relation*, the attributes are unordered, even if the table columns have some order imposed on them.

3. There are no duplicate rows.
Recall that each tuple in a relation is a *proposition*, or a truth statement. To paraphrase Codd: If something is true, saying it twice doesn't make it truer! A *table* may have duplicate rows, but a *relation* never contains duplicate tuples.

4. Every row/column intersection contains exactly one domain value and nothing else.
This is a major point, and we must consider it in some detail.

Often, "data must be atomic", "no repeating groups", or similar phrases are used when defining 1NF.

Consider for a moment what "atomic" may mean. Let us say the table cell contains an individual's last name, a character string. Clearly, this is not "atomic" in the sense of being indivisible, because we can use various functions to take the left-most or right-most characters.

Now, consider a date/time value. We can extract the month, year, day, hour, minute, and second subparts from the value (and perhaps even more); again, clearly not "atomic" in the sense of being indivisible.

Do we then need one table column for each letter of the last name? Do we need six columns to store a date/time value? Clearly, "data must be atomic" is **not** acceptable, and we need a better concept than "atomic" to guide us.

It is better then to say that a "row/column intersection" (or cell) contains a *domain value*. For example, if the attribute that the column represents is "Location", and the domain is "store locations" then each cell in that column contains a store location.

It is true that a particular DBMS may not allow us to so precisely define the domain, and that we may have to settle for a higher level of abstraction (such as attribute "Location" being of type STRING, rather than of type STORELOCATION), but this is failing on the part of the DBMS, not relational theory. This failing, by the way, is one reason why it is often necessary to validate user entries in code or other places *external* to the relation before allowing that data to be added.

Furthermore, a row/column intersection should contain *exactly one* domain value. This has two effects:

1. Prevents one kind of "repeating groups", multiple values in one table cell.
2. Prohibits Nulls in tables.

The second point may seem strange, after all SQL certainly supports Nulls, but accept for now that Nulls are not allowed in tables[1]. If a NULL appears, or could appear, something **must** change to eliminate the NULL or possible NULL.

---

[1] The author hopes to address this in a future appendix to this paper.

Consider now what "repeating groups" may mean. In most references on normalization, this is treated in one of two ways, both of which are said to violate 1NF (for the same "no repeating groups" reason):

A. Multiple values in one cell

Store	Goods
Alpha	Food, Books

B. Multiple columns containing same information

Store	Good1	Good2
Alpha	Food	Books

Fig 2.1

Example A in Fig 2.1 is clearly not in 1NF, because it contains more than one domain value. Example B, though, is a different matter. Each row/column intersection in Example B contains one and only one domain value. Is Example B in 1NF, then?

The (possibly) surprising answer is: **maybe**. Recall that in 1.3.3.1 we said:

> "You can **not** simply look at a table and determine what normal form it is (or isn't) in."

This is an example. Suppose that every store sold exactly two goods. In this case, Example B is in 1NF (even if it may be a poor design choice). However, if some stores had two goods and others less or more, a Null would be required; and 1NF would be violated.

In both cases, we **likely** want to have the same representation:

Store	Good
Alpha	Food
Alpha	Books

Fig 2.2

For Example A, some change is **required** by the rules of 1NF. For Example B, the change may not be required by the rules of 1NF, in this particular case, but it is very probably desirable for other reasons.

Similar to the "atomic" problem, we see that "no repeating groups" is not an acceptable definition for 1NF. So, while a *table* may contain multiple values (or a NULL) is a cell, a *relation* always contains exactly one domain value for each attribute in each tuple[2].

5. No irregular columns

This is not important in our discussion on normalization, but to briefly address the point it prohibits "hidden" columns, table meta-data and other things not present in a relation. That is, a *table* may contain meta-data, but all data in a *relation* is contained in its tuples.

The key to understanding 1NF is realizing the differences between a table and a relation, and making the table represent a relation a directly and faithfully as possible. [Dat03] and [Pas04] contain a more detailed treatment of the concept if you want to pursue additional material.

[Pas04-2] has a nice summary: "… [T]here is no such thing as an 'unnormalized relvar'. We do *not* mean that *any* database *table* is always in 1NF, but rather that tables *designed to faithfully represent relvars* are." [Emphasis in original.]

For the rest of the normal forms, we will consider normalizing a relvar (or set of relvars), and not the table itself. We will use tables to represent the relations that result, but the fact that the table must be in 1NF should be understood.

---

[2] This does differ from the original concept given in [Cod71-2] where "each item is a simple number or character string."

2.3 Second Normal Form (2NF)

Second Normal Form (2NF)

A **relvar** is in 2NF if:

Every nonprime attribute is irreducibly dependent on each candidate key.

Let us recall R from Fig 1.1:

Store	Manager	Location	Status
Alpha	Smith	East	Open
Beta	Jones	East	Open
Delta	Franks	West	Open
Gamma	Wilson	North	Closed

Fig 1.1 – Revisited for 2NF

Also, remember we have identified two candidate keys for R: Store and Manager.

Let us check our table against the 2NF rule. We need to determine if each nonprime attribute is irreducibly dependent on each candidate key.

f of R
f_1: Store → Manager
f_2: Manager → Store
f_3: Store → Location
f_4: Location → Status

Fig 2.4

1. Store → Location

   True, directly given in f_3.
2. Store → Status

   True (but transitive), derived from f_3 and f_4
3. Manager → Location

   True (but transitive), derived from f_2 and f_3
4. Manager → Status

   True (but transitive), derived from f_2 and f_3 and f_4

There are a lot of TDs, but 2NF is not concerned with them. Each nonprime attribute is irreducibly dependent on each candidate key, and so R is in 2NF. (This may be a bit of surprise to those who accept a simplified 2NF definition – we deal with this simplified definition later.)

To illustrate a table in 1NF, but not 2NF, we will therefore need a new example table, call it T:

Store	Location	Item	Ordered
Alpha	East	Widget	10
Alpha	East	Gear	20
Beta	East	Widget	10
Delta	West	Gear	20

Fig 2.5

The f of T:

f_1: Store → Location

f_2: {Store, Item} → Ordered

Here we have one candidate key, a composite candidate key: {Store, Item}. Let us examine the nonprime attributes against this candidate key:

1. {Store, Item} → Location

   True, but not irreducibly so, since Store → Location holds.
2. {Store, Item} → Ordered

   True, and irreducibly so.

So, now we have a 2NF violation. In order to solve the problem, we *decompose* T by taking *projections* over some set of attributes in T. Hopefully common sense will indicate which columns to include in each projection, which usually corresponds to one for each FD in f.

In this case we will take two projects, one as T_1 {Store, Location}, and one as T_2 {Store, Item, Ordered}. Let us do so and examine the results:

T_1	
Store	Location
Alpha	East
Beta	East
Delta	West

T_2		
Store	**Item**	Ordered
Alpha	Widget	10
Alpha	Gear	20
Beta	Widget	10

Delta	Gear	20

Fig 2.6 – A 2NF decomposition of T

Each relation is now in 2NF[3]. We can verify this by checking each relation:

T_1: One candidate key (Store) and one nonprime attribute (Location). Location is irreducibly dependent on Store. Thus, R_1 meets 2NF.

T_2: One candidate key ({Store, Item}), and one nonprime attribute (Ordered). Ordered is irreducibly dependent on {Store, Item}. Thus, R_2 meets 2NF.

Notice that we have *duplicate* data – Store is repeated in each relation. This is acceptable, and is not the same as *redundant* data, which is one thing we want to avoid. Also, by eliminating the redundant data we now only need update Location in one place if it is changed. Also notice that joining these two tables on Store will give us the original data – this means our decomposition is a *lossless decomposition*, because it preserves all the information in the original relation. We always require a lossless decomposition when decomposing a relation.

As another illustration, consider this table of ordering data:

Item	ItemCost	Ordered	TotalCost
Widget	$3.00	10	$30.00
Gear	$3.00	12	$36.00
Paper	$3.00	10	$30.00

Fig 2.5

Here is the$f$:

f_1: Item → ItemCost
f_2: Item → Ordered
f_3: {ItemCost, Ordered} → TotalCost

There is one candidate key, Item. This means one prime attribute, Item, and three nonprime attributes: ItemCost, Ordered, and TotalCost.

It is clear from the FDs in f that ItemCost and Ordered are both irreducibly dependent on the candidate key. It is also equally clear that TotalCost is not. So, we have a 2NF violation. Storing calculated fields is a common normalization problem. In this case, TotalCost need not be stored at all, since it can be calculated from ItemCost and Ordered when needed.

Earlier, we mention briefly a simplified 2NF definition, this is one such definition:

Second Normal Form (2NF) ("Simplified")

A **table** is in 2NF if:

1. It is in 1NF and
2. Every non-key attribute is irreducibly dependent on the primary key.

We have already sufficiently addressed the 1NF qualification, and why it need not appear, but notice the slight difference in the second qualification.

If a relation has only one candidate key, the above simplification is acceptable. This is because with only one candidate key, the second qualification is equivalent to our 2NF definition. However, as we saw here, if a relation has multiple candidate keys, this simplification is misleading, because it would require decomposition. Later we will see that such simplified discussion can confuse things when dealing with higher normal forms. Because we are striving for accuracy, we mention this common simplified definition only to reject it.

2.3.1 Optimal Second Normal Form

Optimal Second Normal Form (Optimal 2NF)

A **collection of relvars** is in Optimal 2NF if:

Every relvar in the collection is in 2NF
The collection contains the fewest possible relvars to meet the above requirement.

Here is our first normal form that deals with a collection of relvars, rather than an individual relvar. Fortunately, it is a very gentle introduction to such normal forms. [Cod71] introduced Optimal 2NF.

This simply means that if there are multiple possible decompositions for a relvar into 2NF; choose the one resulting in the fewest relvars. Our T_1, T_2 example is in optimal 2NF.

---

[3] Actually, we are quite a bit past 2NF in this solution. Our goal is not to move from 1NF to 2NF, it is to satisfy the normal form we notice the relation violating. If the solution then puts us at a higher normal form, that is more than acceptable.

Note that there could be more that one Optimal 2NF representation for a collection of relations [Dat99].

2.4 Third Normal Form (3NF)

Third Normal Form (3NF)

A **relvar** is in 3NF if:
It is in 2NF, and
Every nonprime attribute is directly dependent on each candidate key

Let us recall R from Fig 1.1 and our analysis of the FDs from 2.3:

Store	Manager	Location	Status
Alpha	Smith	East	Open
Beta	Jones	East	Open
Delta	Franks	West	Open
Gamma	Wilson	North	Closed

Fig 1.1 – Revisited for 3NF

Again, remember we have identified two candidate keys for R: Store and Manager.

Let us check our table against the 3NF rule. We need to determine if each nonprime attribute is directly dependent on each candidate key.

1. Store → Location
True, directly given in f_3.
2. Store → Status
A TD derived from f_3 and f_4
3. Manager → Location
A TD derived from f_2 and f_3
4. Manager → Status
A TD derived from f_2 and f_3 and f_4

f of R
f_1: Store → Manager
f_2: Manager → Store
f_3: Store → Location
f_4: Location → Status

Fig 2.4 – Revisited for 3NF

Each nonprime attribute is **not** directly dependent on each candidate key, so the table is **not** in 3NF.

Again, the solution is decomposition by taking projections on attributes in R:

R_1	
Store	Manager
Alpha	Smith
Beta	Jones
Delta	Franks
Gamma	Wilson

R_2	
Store	Location
Alpha	East
Beta	East
Delta	West
Gamma	North

R_3	
Location	Status
East	Open
West	Open
North	Closed

Fig 2.6 – A 3NF decomposition of R

Note that joining these tables on Store and Location will give use the original data, so we have the desired *lossless decomposition* and our original R can be restored. We have “split” each TD, preventing it from existing in our resulting tables.

Again, notice how our results have one relation per FD in the original relation. R_1 corresponds to f_1 (andf_2), R_2 corresponds to f_3, and R_3 to f_4. Each resulting relation is in 3NF.

Like 2NF, sometimes the 3NF definition is sometimes simplified:

Third Normal Form (3NF) (“Simplified”)

A **table** is in 3NF if:
1. It is in 2NF and
2. Every non-key attribute is nontransitively dependent on the primary key.

And, as discussed for the simplified 2NF definition, this is acceptable **if** there is only one candidate key.

2.4.1 Optimal 3NF

Optimal Third Normal Form (Optimal 3NF)

A **collection of relvars** is in Optimal 3NF if:

Every relvar in the collection is in 3NF

No relation in C3 has a pair of attributes A and C where C is strictly transitively dependent on A in C2.

The collection contains the fewest possible relvars to meet the above requirements.

Where C2 is a collection of relvars in Optimal 2NF and C3 is a collection of relvars in 3NF derived from C2.

The definition of Optimal 3NF (from [Cod71]) is much more difficult than the actual concept.

Consider this alternate decomposition of R:

R_1	
Store	Manager
Alpha	Smith
Beta	Jones
Delta	Franks
Gamma	Wilson

R_2	
Store	Location
Alpha	East
Beta	East
Delta	West
Gamma	North

R_3	
Store	Status
Alpha	Open
Beta	Open
Delta	Open
Gamma	Closed

Fig 2.7 – An alternate 3NF decomposition of R

We now compare this alternate decomposition to the Optimal 3NF definition:

1. Every relvar in the collection is in 3NF.

True (with the understanding of what we introduce in the next point).

2. No relation in C3 has a pair of attributes A and C where C is strictly transitively dependent on A in C2.

False. This is the problem, so let us examine it closely. C2 is a collection of relvars in Optimal 2NF (in this case a collection of one relvar, R). C3 contains 3 relvars, R_1, R_2, and R_3.

The problem is with R_3. It contains a pair of attributes, Store and Status, where Status is strictly transitively dependent on Store in R.

In R, the original 2NF relvar, we had:

f_3: Store → Location

f_4: Location → Status

Giving us the TD Store (Location, Status), which is strict because Status $\nrightarrow$ Location and Location $\nrightarrow$ Store.

Consider the effect of having Store and Status (the A and C in the definition) in R_3; we have lost the fact entirely that Location → Status. f_4 is not represented anywhere in C3, and therefore can not be enforced (at the relvar or relation level). There is nothing preventing us changing the Status of Alpha in R3 to Closed, which would violate our business rule that all stores in the same location share the same status.

This idea is *dependency preservation*, meaning that dependencies in the original relvar exist in the relvars that remain after decomposition. This is a very important concept, which along with lossless decomposition means that we retain the same *information* – data and dependencies – when decomposing.

Compare, then, the alternate decomposition in Fig 2.7, with the original 3NF decomposition in Fig 2.6, which is in Optimal 3NF.

3. The collection contains the fewest possible relvars to meet the above requirements.

In Fig 2.7, this is irrelevant, because we did not meet the second requirement. However, in Fig 2.7, this is true.

As with Optimal 2NF, there could be more than one Optimal 3NF representation of the data.

2.5 Summary

In this section, we have introduced 1-3NF, giving definitions and examples for each normal form.

There are two interesting points to make about 3NF, before moving on to the higher normal forms:

- Achieving 3NF is always possible.[Ber76]
- If you are in 3NF with a simple key, then the relation is in PJ/NF as well.[Dat92]

So, if you have a simple key – then 3NF is "far enough", because you are actually in PJ/NF, a much more stringent normal form. Another way to think about it is with a simple key and in 3NF, then:

Every nonprime attribute is dependent on the key, the whole key, and nothing but the key.

However, as we will see in the section on higher forms, composite keys – keys with having two or more attributes – may introduce considerations that 3NF does not address.

We also want to note that although we have tried to use the same (or a similar) relation throughout this section, we will not do so in the upcoming section. There are two reasons for this:

> 1. We want to clearly illustrate the specific problem that each higher normal form addresses. This is hopefully made simpler by constructing a relation that has the problem, rather than trying to "back-track" one example from PJ/NF to 1NF, ensuring it illustrates a violation at each step along the way.
>
> 2. More importantly, it is misleading to think that the normalization process starts with one large mass of unnormalized data and then proceeds to 1NF, then to 2NF, then to 3NF, and so on. The normal forms provide a good test or set of rules that a table can be checked against to ensure the designer has not overlooked some aspect of normalization.

Keep in mind that the normal forms are a formalization of "common sense", and most experienced developers will have a "first cut" of tables that is already in 3NF (or higher) without consciously stepping through each lower normal form.

3

HIGHER NORMAL FORMS

3.1 About the Higher Normal Forms

There are particular relations that can be in 3NF, yet still contain redundancy and therefore present the risk of anomalies.

There have been various normal forms proposed that exceed (or modify) 3NF. In this section, we discuss the three commonly accepted: Boyce-Codd Normal Form, Fourth Normal Form, and Projection-Join Normal Form.

You only need concern yourself with the higher normal forms when you have composite keys.

That's the good news. The bad news is if you do have composite keys, you'll need to consider each of the higher normal forms in turn, and they can be more difficult to understand than the basic normal forms.

3.2 Boyce-Codd Normal Form

Boyce-Codd Normal Form (BCNF)

A **relvar** is in BCNF if:
Every determinant is a candidate key.

The first thing we want to note about BCNF is that we don't need to specify that the relation is already in 2NF or 3NF. This is contrary to expectation, because we usually think about "progressing" through the normal forms, first achieving a lower normal form, and then further refining those results to achieve the next normal form.

We can get away with this because BCNF rule encompasses the rules for 2NF and 3NF[Wer93].

Because of the second point, we will discuss two aspects of BCNF. The first aspect we want to discuss is using BCNF as a "target" normal form to replace 2NF and 3NF rules. The second aspect is considering BCNF in its own right, as a stricter normal form than 3NF.

It is also important to note that while the requirements for 2NF and 3NF were based on non-prime attribute dependencies, BCNF requirements are based on determinants, which may be prime or non-prime attributes.

If you have a relation with a simple key, obviously there can't be (nontrivial) dependencies within that key. All other attributes must be non-prime attributes, so all dependencies must involve a non-prime attribute, and therefore the 2NF and 3NF rules can handle those dependencies.

So then, it becomes clear why we can say BCNF (and higher) are only important when dealing with relations having a composite key: because that is the only situation where a relation can have dependencies not involving a non-prime attribute.

3.2.1 BCNF is 2NF and 3NF

Let's spend just a moment to revisit some previous examples to illustrate that the BCNF rule encompasses the previous rules for 2NF and 3NF. This time, instead of applying the 2NF or 3NF rules, we will simply apply the single BCNF rule:

2NF

Store	Location	Item	Ordered
Alpha	East	Widget	10
Alpha	East	Gear	20
Beta	East	Widget	10
Delta	West	Gear	20

Fig 2.5 – Revisited for BCNF

We have two determinants, Store and {Store, Item}, and Store alone, although a determinant alone for Location, is **not** a candidate key. Taking the same projections:

T_1	
Store	Location
Alpha	East
Beta	East
Delta	West

T_2		
Store	**Item**	Ordered
Alpha	Widget	10
Alpha	Gear	20
Beta	Widget	10
Delta	Gear	20

Fig 2.6 – Revisited for BCNF

Examining each resulting relation in turn:

	Determinant	Candidate Key	BCNF?
T_1	Store	Store	Yes
T_2	{Store, Item}	{Store, Item}	Yes

Fig 3.1 – Verifying BCNF

So we see that BCNF encompasses 2NF.

3NF

Let us recall R from Fig 1.1 and our analysis of the FDs from 2.3:

Store	Manager	Location	Status
Alpha	Smith	East	Open
Beta	Jones	East	Open
Delta	Franks	West	Open
Gamma	Wilson	North	Closed

Fig 1.1 – Revisited for BCNF

We already know this table is in 2NF, but not 3NF. We want to analyze it for BCNF.

f of R
f_1: Store → Manager
f_2: Manager → Store
f_3: Store → Location
f_4: Location → Status

Fig 2.4 – Revisited for BCNF

Looking over the f of R, we see three determinants:

Store
Manager
Location

However, as we well know by now, R has only two candidate keys:

Store
Manager

Since every determinant is **not** a candidate key (Location is a determinant, but not a candidate key), R violates BCNF. To decompose, we take projections over attributes in R, giving us the same result as our original 3NF decomposition:

R_1	
Store	Manager
Alpha	Smith
Beta	Jones

R_2	
Store	Location
Alpha	East
Beta	East

R_3	
Location	Status
East	Open
West	Open

Delta	Franks
Gamma	Wilson

Delta	West
Gamma	North

North	Closed

Fig 2.6 – Revisited for BCNF

Examining each resulting relation in turn:

	Determinant(s)	Candidate Key(s)	BCNF?
R_1	Store, Manager	Store, Manager	Yes
R_2	Store	Store	Yes
R_3	Location	Location	Yes

Fig 3.2 – Verifying BCNF

So we see that BCNF encompasses 3NF.

3.2.2 BCNF is more than 3NF

There's another aspect of BCNF, where we look at it not as a form that includes 2NF and 3NF, but as a separate normal form in its own right. After all, BCNF is an improvement or stricter normal form, not an equivalent of 3NF.

This aspect of BCNF is important with a relation has two or more candidate keys, and those candidate keys share one or more of the same attributes.

To illustrate this, we need a relation that is in 3NF, but **not** is BCNF. Call this relation B, and notice it is a slight variation of our second 2NF example:

Store	Manager	Item	Ordered
Alpha	Smith	Widget	10
Alpha	Smith	Gear	20
Beta	Jones	Widget	10
Delta	Franks	Gear	20

Fig 3.3 – A variation on Fig 2.5

The f of B:

f_1: Store → Manager
f_2: Manager → Store
f_3: {Store, Item} → Ordered

We have simply replaced Location with Manager, yet the table is now in 3NF. This change yields **two** candidate keys (with Location there was only one candidate key): {Store, Item} and {Manager, Item}. This means that Ordered is the only nonprime attribute in B. And, Ordered, the only non-prime attribute, is both fully functionally dependent on each candidate key (2NF), and non-transitively dependent on each candidate key (3NF)[4]. However, every determinant is not a candidate key (neither Store nor Manager alone is a candidate key), so B does violate BCNF.

We decompose like so:

B_1	
Store	Manager
Alpha	Smith
Beta	Jones
Delta	Franks

B_2		
Store	Item	Ordered
Alpha	Widget	10
Alpha	Gear	20
Beta	Widget	10
Delta	Gear	20

Fig 3.4 – A BCNF decomposition of B

Compare Fig 3.4 to Fig 2.6. The results are very similar, but the reasoning is different.

3.2.3 BCNF is "Almost All You Need To Know"

BCNF prevents the anomalies that Codd originally identified, and was considered an improved or corrected definition for 3NF [Hea71] [Cod74]. We have also shown that BCNF covers 2NF and 3NF requirements.

This raises two questions:

1. In practice, can we ignore 2NF and 3NF, and simply use BCNF?
2. Do we ever need go beyond BCNF?

Let us address the first question first: *can we ignore 2NF and 3NF, and simply use BCNF?*

---

[4] Note that this relation violates the simplified 2NF or 3NF rules mentioned earlier. This is one reason why those simplified rules can be confusing – they do not apply properly to multiple candidate keys.

Most of the time, **yes**. However, there is one important distinction between BCNF and 2NF and 3NF that prevents us from saying "In all cases, yes". That distinction is that although all relvars can be put in 2NF and 3NF with lossless decompositions **and** preserve dependencies, there are certain relvars that **can not** be put in BCNF and preserve dependencies [Bee79]. We illustrate with a new table, called Z:

Street	City	ZipCode
Main	Springfield	20010

Fig 3.5 – A problematic relation

The f of Z:

f_1: {Street, City} → ZipCode.
f_2: ZipCode → City.

Z has one candidate key: {Street, City}, and one nonprime attribute, ZipCode, which is directly dependent on that candidate key. Z is in 3NF. However, Z has two determinates, {Street, City} and ZipCode, only one of which is a candidate key. Z is **not** in BCNF. We can generalize this problem as giving the f as: AB → C, C → B.

We *could* decompose like so:

Street	ZipCode
Main	20010

City	ZipCode
Springfield	20010

Fig 3.4 – A possible BCNF decomposition of Z

Consider the decomposition in the light of Optimal 3NF – the problem is very similar. We have lost f_1 – the fact that ZipCode is functionally dependent on {Street, City} is no longer reflected, and we have not preserved all dependencies originally in Z. To illustrate, consider that there is nothing preventing the following:

Street	ZipCode
Main	20010
Main	20020

City	ZipCode
Springfield	20010
Springfield	20020

Fig 3.5 – A problem

Streets can have multiple zip codes (in different cities), and a city can have multiple zip codes, so there is no problem with either relation *at the relation level*. However, examine the join:

Street	City	ZipCode
Main	Springfield	20010
Main	Springfield	20020

Fig 3.6 – The join

f_1 no longer holds; in the join {Street, City} $\nrightarrow$ ZipCode. From BCNF and upward, it may not be possible to maintain dependency preservation. Generally, the advice is to continue normalizing, even if some dependencies are lost, but there are differing opinions. Because of this, some relvars may be consciously left in 3NF in order to preserve dependencies, even when a higher normal form could be achieved.

Understanding this, BCNF does handle all problems with functional dependencies, which brings us to our second question: *do we ever need to go beyond BCNF?*

BCNF meets the definition given in [Pas04-2]: "A fully normalized table satisfies *only* FDs implied by the key, the whole key, and nothing but the key." One might then think that BCNF is indeed an acceptable stopping point; however, there are other, more general dependencies than functional dependencies, and we need higher normal forms to address those more general dependencies.

([Pas04-2] later generalizes the earlier definition: "A fully normalized table satisfies *only* dependencies implied by the key, the whole key, and nothing but the key". With this more general definition, we cover the coming higher normal forms.)

3.3 Fourth Normal Form

Fourth Normal Form (4NF)

A **relvar** is in 4NF if:

There are no independent non-trivial multivalued dependencies.

4NF was introduced to handle multivalued dependencies[Fag77]. You sometimes hear 4NF dismissed as so rare to be of theoretical interest only. However, a study of 40 large real-world databases found "that data violating 4NF occurred at least once in nine organizational databases constituting over twenty percent of the databases in [the] study."[Wu92]

Even if the following examples seem unrealistic and a bit contrived, keep in mind 4NF violations are a real-world problem.

We begin by considering the following table T (based on Fagin's original example in [Fag77]):

Employee	Salary	Year	Child
Smith	$9K	2000	Thomas
Smith	$9K	2000	Andrea
Smith	$10K	2001	Thomas
Smith	$10K	2001	Andrea

Fig 3.5 – 4NF Violation

Now, there is only one candidate key for **T**, and that is all attributes. Therefore 1-3NF requirements are met. There are no standard functional dependencies, so no determinants, so the above is in BCNF[5]. *Still, it is obvious the table is "wrong".* 4NF helps us formalize exactly what is wrong.

Consider the fact that an Employee has a certain set of children. There is therefore *some* type of dependency between the fields Employee and Child. However, since more than one Child value can depend on the same Employee value, it is not a functional dependency. We can *not* say that **Employee → Child**, because one Employee could have multiple corresponding Child values.

This dependency is a *multivalued dependency* (MVD); one attribute (Employee) determines a specific *set* of values for another attribute (Child)[6]. We notate like so:

Employee →→ {Child}

The same applies for the Salary and Year. An Employee has a specific set of values for these two attributes:

Employee →→ {Salary, Year}[7]

MVDs always go in pairs like this, so it is common to represent them in one notation, like so:

Employee →→ {Child} | {Salary, Year}

We generalize the above notation, and note that some references use curly braces on the right and others do not:

A →→ B | C

A MVD is *trivial* if only that multivalued dependency is in the table. These two MVDs are *independent*, because Child does not affect (Salary, Year) and vice versa. So, we have a table with the two *independent non-trivial multivalued dependencies*, and are thus in violation of 4NF. Note that having two independent trivial MVDs is one relation is not possible.

Just as we decomposed to isolate FDs in the resulting relations, we take a similar approach with MVDs. We need to separate each MVD into a different relation to satisfy 4NF.

Knowing this, we can decompose as follows:

Employee	Child
Smith	Thomas
Smith	Andrea

Employee	Salary	Year
Smith	$9K	2000
Smith	$10K	2001

Fig 3.6 – A 4NF Decomposition of Fig 3.5

Now in each relation there is only one MVD, so that MVD is trivial and both resulting relations are in 4NF.

Let's consider a slightly different example:

Employee	SSN	Child
Smith	123-45-6789	Thomas
Smith	123-45-6789	Andrea

Fig 3.7 – Another 4NF Violation

Here are the dependencies in Fig 3.7:

Employee → SSN
Employee →→ {Child}

In this case we have two dependencies, a FD and a MVD. Is the table is 4NF[8]? No, because a FD is a type of MVD[9], we actually have two independent MVDs (and therefore non-trivial), and so the table violates 4NF.

---

[5] This is a specific example of the general case of saying a table that is "all key" is in BCNF. No proper subset of attributes can form a candidate key.

[6] Note that this means a FD is a special case of a MVD, where the "multi" is equal to one.

[7] This does not mean that Employee →→ {Salary} and Employee →→ {Year} are true. Take the latter; this would mean $\text{YEAR}_{\text{Employee, Salary, Child}}$ is dependent only on Employee, which it is not. [Fag77] states Year and Salary are a "cluster".

[8] It is not in BCNF either, but that's not what we are trying to illustrate here.

[9] We could restate Employee → SSN as Employee →→ {SSN}

Decomposition is simple (and obvious):

Employee	SSN
Smith	123-45-6789

Employee	Child
Smith	Thomas
Smith	Andrea

Fig 3.7 – A 4NF Decomposition of Fig 3.6

One final 4NF illustration, this one more commonly encountered[10]:

Employee	Project	Activity
Smith	Quality Assurance	Support
Smith	User Training	Support
Smith	Quality Assurance	Debug
Smith	User Training	Debug
Jones	Quality Assurance	Support
Jones	User Training	Support
Jones	Quality Assurance	Debug
Jones	UserTraining	Debug

Fig 3.8 – A final 4NF Violation

Assume the business rules:

1. An Employee can be assigned to any Project
2. An Employee can be assigned to any Activity
3. No matter what Project an Employee is assigned to, he is assigned the same Activity.
4. A Project can be assigned to multiple Employees.
5. An Activity can be assigned to multiple Employees.

It is business rule #3 which makes the two MVDs, **Employee →→ Project | Activity** *independent* – it doesn't matter which Project the Employee is assigned, he will have the same Activity within that Project as any other Projects he may be assigned. Understanding this point is crucial to understanding 4NF (and PJ/NF, as we will see).

Problems with the table in Fig 3.8? There is the obvious redundancy, and several opportunities for update anomalies:

An Employee can't be assigned a Project without having an Activity.
An Employee can't be assigned an Activity without having a Project[11].
An Employee only assigned to one Project can't be removed without deleting all assigned Activities.
An Employee only assigned to one Activity can't be removed without deleting all assigned Projects.

Decomposition is obvious enough, each MVD to a relation:

Employee	Project
Smith	Quality Assurance
Smith	User Training
Jones	Quality Assurance
Jones	User Training

Employee	Activity
Smith	Support
Smith	Debug
Jones	Support
Jones	Debug

Fig 3.9 – A 4NF Decomposition of Fig 3.8

There are some other interesting things Fagin proved when introducing 4NF:

- If a relation is in 4NF it is in BCNF.
- All relations can be put in 4NF (but, not necessarily with dependency preservation).

---

[10] This particular example is adopted from [Pas04-2]

[11] Remember, no NULLS allowed!

3.4 Projection – Join Normal Form

Projection-Join Normal Form (PJ/NF)

A **relvar** is in PJ/NF if:
Every nontrivial join dependency is the result of a key

3.4.1 Join Dependencies

Just as 4NF was introduced to deal with multivalued dependencies, which are a generalized form of functional dependencies – PJ/NF is introduced to deal with join dependencies (JDs), which are in turn a generalized form of multivalued dependencies.

First, consider the following simple process:

1

Store	SName	Employee
13	Mint Shop	Jones
13	Mint Shop	Frank

R{Store, SName, Employee}
Store → SName
Store →→ {Employee}

This is our original relation.

2

Store	SName
13	Mint Shop

Store	Employee
13	Jones
13	Frank

Take the projections R_1 and R_2:
R_1 = {Store, SName}
R_2 = {Store, Employee}

Here we take some projections.

3

Store	SName	Employee
13	Mint Shop	Jones
13	Mint Shop	Frank

$R_1 \bowtie R_2$

And finally, we join.

Fig 3.10 – A Join Dependency Illustration

In the above we had a relation R, from which we derived the set of projections {R_1, R_2}. Notice then the join of those projections resulted in R, the original relation. Because of this, we can say that R obeys the JD ∗{R_1, R2}. If the join did not result in the original relation we could say ∗{R_1, R2} does not hold for R.

The JD, like all dependencies, specifies a constraint. In the case of the JD *{R_1, R_2}, this constraint means every instance of R should have a lossless decomposition into R_1 and R_2. [Orl92] Specifically, we can expand the attribute names of R_1 and R_2 and say that R has the JD: *{(Store, SName), (Store, Employee)}.

3.4.1.1 Relating the Dependencies

When [Fag71] introduced 4NF, he showed:
A →→ B | C holds for R, iff R is the join of the projections R_1 (A, B) and R_2 (A, C)

Restating this, it clearly shows that a MVD is a special case of a JD, meaning the JD generalizes the MVD, just as the MVD generalizes the FD:
A →→ B | C ≡ *{AB, AC}

Consider the dependencies in R:
Store → SName
Store →→ Employee

Since a FD is a special case of a MVD, we can rewrite the above using only MVDs:
Store →→ SName | Employee

And, following with our proof from [Fag71], we can rewrite the above using a JD:
***{(Store, SName), (Store, Employee)}**

3.4.1.2 Relating the Normal Forms

[Fag79] provided new alternative definitions for BCNF and 4NF which help illustrate the relationship between these forms[12]:

---

[12] Note the similar effect alternative definitions have in the EKNF discussion in Appendix E in relating several normal forms.

Let R be a relation, and let Δ be the set of key dependencies of R. R is in the stated normal form if:

BCNF

$\Delta \models \sigma$ for each FD in R

4NF

$\Delta \models \sigma$ for each MVD in R

PJ/NF

$\Delta \models \sigma$ for each JD in R

In this definition, *key dependencies* are equivalent to candidate key. [Fag79] also notes that the definitions work if superkeys are considered instead of candidate keys.

3.4.2 PJ/NF Illustrated

To illustrate PJ/NF, consider this example from [Ken83]:

Agent	Company	Product
Smith	Ford	Car
Smith	Ford	Truck
Jones	GM	Car

Fig 3.11 – A PJ/NF Violation

Here, assume the following business rules:

1. An agent represents many companies and sells many products.
2. A company makes many products, and each product is produced by many companies.
3. If an agent represents a company that makes a product the agent sells, then the agent sells that product for that company.

There is a MVD in the above[13], corresponding to rule 1:

Agent →→ Product | Company

Rule 2 could be represented with the following:

Company →→ {Product}
Product →→ {Company}

However, consider rule 3 carefully. This constraint can neither be represented as a functional dependency *nor as a multivalued dependency*. However, it **can** be represented as a join dependency. This is an example of a JD that is **not** a MVD.

Remember that MVDs are a special case of JDs. So, we can rewrite the MVDs as JDs, and include rule 3:

∗{(Agent, Product), (Agent, Company)}
∗{Agent, Company}
∗{Company, Product}

Let us take those projections:

Agent	Company
Smith	Ford
Jones	GM

Company	Product
Ford	Car
Ford	Truck
GM	Car
GM	Truck

Agent	Product
Smith	Car
Smith	Truck
Jones	Car

Fig 3.12 – A PJ/NF Decomposition of Fig 3.11

If we join the above relations, the result is the original table. So, we can say that the original table obeys the join dependencies we identified. Each of the resulting projections is in PJ/NF. We have normalized our original table using projections and joins.

Let us now consider an **incorrect** attempt at the above – the purpose of this attempt is to illustrate what some call a *cyclic dependency*. Perhaps one might try to analyze the table like so:

Agent →→ {Product}
Company →→ {Product}

This would suggest the following projections:

Agent	Product
Smith	Car

Company	Product
Ford	Car

---

[13] So, it is not in 4NF. However, we analyze for PJ/NF to illustrate several points about PJ/NF.

Smith	Truck
Jones	Car

Ford	Truck
GM	Car
GM	Truck

Fig 3.13 – A PJ/NF Decomposition???

At first glance, this may look acceptable; after all we have "covered" all the columns. However, examine the results of the join:

Agent	Company	Product
Smith	Ford	Car
Smith	Ford	Truck
Smith	GM	Car
Smith	GM	Truck
Jones	GM	Truck
Jones	GM	Car

Fig 3.14 – The incorrect resulting join

Notice the shaded area, indicating rows that did not exist in the original relation. Because our identified dependencies are not sufficiently represented – the Agent and Company relationship is not expressed – we have a *lossy decomposition* – not acceptable.

We can also say that original table does **not** obey the join dependencies ∗{Agent, Product} and ∗{Company, Product}. It's clear that these two dependencies can't accurately represent all the dependencies we laid out in our assumptions. Because each pair of attributes exhibits a dependency on another pair of attributes, there is the *cyclic dependency* we noted earlier.

Consider the original table again, but add the assumption that an agent can not work for two companies that produce the same product. *With this new assumption, the table is in 4NF.*

This is because the assumption affects the MVDs:

Agent →→ {Company} | {Product}

These are no longer independent, because under the new assumption the product an agent carries affects which company the agent may represent[14].

This leaves us with something like[15]:

Agent →→ {Company, Product}
Company →→ {Product}

These MVDs are clearly not independent (the second is a subset of those attributes present in the right hand side of the first), so 4NF is no help here. However, the same problem remains in respect to PJ/NF, and decomposition on the three projections we saw earlier is now required by PJ/NF[16]. This becomes an example of 4NF not being strict enough, so PJ/NF is required to address the problems.

Let us now revisit a previous 4NF example[17]:

Employee	Project	Activity
Smith	Quality Assurance	Debug
Smith	User Training	Support
Jones	Quality Assurance	Support
Smith	Quality Assurance	Support

Fig 3.15 – Another PJ/NF Violation

We restate the business rules, changing #3 slightly:

1. An Employee can be assigned to any Project
2. An Employee can be assigned to any Activity
3. If an Employee is assigned to a Project and Activity, and the Project has that Activity, then Employee has that Activity within that Project.
4. A Project can be assigned to multiple Employees.
5. An Activity can be assigned to multiple Employees.

---

[14] This is a similar situation to the $\text{YEAR}_{\text{Employee, Salary, and Child}}$ example in the 4NF discussion.

[15] We say "something like" because this notation doesn't correctly reflect all the dependencies present.

[16] As we noted earlier, the first version of the table violated 4NF, so we could have attempted decomposition on that basis, rather than a PJ/NF violation.

[17] Again, this example is adopted from [Pas04-2]

As we have already seen, a change in the business rules has normalization implications. Under our new rule #3, Project and Activity assignments are no longer independent, as one affects the other. This subtle distinction makes all the difference in analyzing the normal form violations, and is an example of "you can't tell what normal form you are in unless you know the business rules."

Here is the decomposition:

Employee	Project
Smith	Quality Assurance
Smith	User Training
Jones	Quality Assurance

Employee	Activity
Smith	Debug
Smith	Support
Jones	Support

Project	Activity
Quality Assurance	Debug
Quality Assurance	Support
User Training	Support

Fig 3.16 – A PJ/NF Decomposition of Fig 3.15

Note in this example, the tuple {Smith, User Training, Debug} is not valid, as the User Training Project does not have the Debug Activity. The same tuple would be valid under the earlier rule #3 (in fact, it would be required). The 4NF example can not handle the JD *{(Employee, Project), (Employee, Activity), (Project, Activity)}.

That JD was not in effect under the business rules given in the 4NF discussion, but *based on an examination of the table alone*, without knowledge of the business rules; one could not determine exactly how to decompose the original table.

As a final note here, the study in [Wu92] turned up no PJ/NF violations, suggesting such problems are extremely rare in practice[18].

3.4.2 Is PJ/NF 5NF?

From the introduction of [Fag79], the paper that introduced PJ/NF:

> *We note that PJ/NF could logically be called 'fifth normal form", since it is stronger than fourth normal form. However, we instead choose to call it projection-join normal form for several reasons. First, we wish to emphasize its finality with respect to the projection and join operators. Second, we feel that from now on, it will be desirable to explicitly point out the relationship between normal forms and the allowed relational operators.*

Despite this, PJ/NF was often referred to as 5NF. This didn't matter much until [Mai83] introduced *project join normal form (PJNF)*. The similar names caused some confusion, and to make things worse, many people discovered an error in [Mai83], which led to several proposed "corrected" 5NF definitions.

However, most people refer to Fagin's Projection-Join Normal Form as 5NF.

In some formal literature, [Fag79] is projection-join normal form (PJ/NF), and [Mai83] project-join normal form is called 5NF, to help distinguish the two. We adopt this practice throughout this paper, which is why there is no treatment of a "5NF" in the main body.

In Appendix E the following normal forms are discussed which are related to this discussion, all of which address PJ/NF and/or correcting [Mai83]:

- Key Complete Normal Form
- Redundancy Free Normal Form
- Reduced Fifth Normal Form
- Project Join Normal Form (5NF)
- Superkey Normal Form
- Q-5NF
- 5NF (Khodovskii's)

---

[18] [Wu92]: "Data violating 5NF did not occur in any of the databases which may indicate that 5NF is only an academic issue."

APPENDIX A

NOTATION

Set Theory

$\in$	Set membership
$\emptyset$	Null set
$\subset$	Proper subset (not equal)
$\subseteq$	Subset
$\cap$	Intersection
$\cup$	Union
$-$	Difference
$\{A,B\}$	Set containing A and B

Logic

$\Rightarrow$	Logical Implication
$\equiv$	Identical to
$\models$	Logical Consequence -whenever the left hand side holds, so does the right hand side, and there is no possible counterexample to the contrary. Example: $\{A \rightarrow B, B \rightarrow C\} \models A \rightarrow B$

Relational Theory

AB	$A \cup B$ (When A and B are sets of attributes) $\{A, B\}$ (When A and B are single attributes)	
$A \rightarrow B$	Functional dependency	FD
$A(B,C)$	Transitive dependency	TD
$A \rightarrow\rightarrow B$	Multivalued dependency	MVD
$A \rightarrow\rightarrow B \mid C$	Embedded multivalued dependency	EMVD
$*\{R_1,R_2\}$	Join dependency	JD
IN(A, S)	Domain dependency	DD
KEY(K)	Key dependency	KD
$R_i[X] \subseteq R_j[Y]$	Inclusion dependency	IND
$R_i[X] \mid R_j[Y]$	Exclusion dependency	EXD
$\bowtie$	Join	
$\lvert A \rvert$	Cardinality of A	

APPENDIX B

INFERENCE RULES

Functional Dependencies	
Armstrong's three original axioms[Arm03]	*Derived*[Bee77][Zan82]*:*
(FD1) Reflexivity If B ⊆ A, then A → B	**(FD4) Pseudotransitivity** If A → B, and BC → D, then AC → D
(FD2) Augmentation If A → B, then AC → BC	**(FD5) Union** If A → B, and A → C, then A → BC
(FD3) Transitivity If A → B, and B → C, then A → C	**(FD6) Decomposition** If A → B, then A → b, for every b in B

Example:

Consider the set of FDs: {ABC → D, B → A}

We want to know if ABC → D is an elementary functional dependency.

1. B → A = BBC → ABC (FD2) Augmentation
2. BBC → ABC = BC → ABC (FD1) Reflexivity
3. BC → ABC → D (FD3) Transitivity

Since ABC → D and BC → D, then ABC → D contains a non-required attribute (A) on the left hand side and is therefore **not** an elementary functional dependency.

Manager	Store	Week	Promotion
Smith	27	1	Donuts
Smith	27	2	Cola
Jones	13	1	Toothpaste

Here we map our dependencies to the example like so:

- (Manager, Store, Week) → Promotion (ABC → D)
- Store → Manager (B → A)

It is obvious (Store, Week) is sufficient to determine Promotion, and that Manager is extraneous.

The FD axioms and inference rules above are the most commonly used. The rules that follow cover the more general types of dependencies.

Multivalued Dependencies[Bee77]	
(MVD0) Complementation If $B \cap C \subseteq A$, then $A \rightarrow\rightarrow B$ iff $A \rightarrow\rightarrow C$	**(MVD4) Pseudotransitivity** If $A \rightarrow\rightarrow B$, and $BD \rightarrow\rightarrow C$, then $AD \rightarrow\rightarrow C - BD$
(MVD1) Reflexivity If $B \subseteq A$, then $A \rightarrow\rightarrow B$	**(MVD5) Union** If $A \rightarrow\rightarrow B_1$, and $A \rightarrow\rightarrow B_2$, then $A \rightarrow\rightarrow B_1B_2$
(MVD2) Augmentation If $C \subseteq D$, and $A \rightarrow\rightarrow B$, then $AD \rightarrow\rightarrow BC$	**(MVD6) Decomposition** If $A \rightarrow\rightarrow B_1$, and $A \rightarrow\rightarrow B_2$, then *Intersection*[Sil01] $A \rightarrow\rightarrow B_1 \cap B_2$, *Difference*[Sil01] $A \rightarrow\rightarrow B_1 - B_2$, and $A \rightarrow\rightarrow B_2 - B_1$
(MVD3) Transitivity *General Case* If $A \rightarrow\rightarrow B$, and $B \rightarrow\rightarrow C$, then $A \rightarrow\rightarrow B - C$ *Special Case of B and C being disjoint* If $A \rightarrow\rightarrow B$, and $B \rightarrow\rightarrow C$, then $A \rightarrow\rightarrow C$	

Functional Dependencies and Multivalued Dependencies[Bee77]
(FD-MVD1) Replication If $A \rightarrow B$, then $A \rightarrow\rightarrow B$ **(FD-MVD2) Coalescence** If $A \rightarrow\rightarrow C$, and $B \rightarrow\rightarrow C'$, then $A \rightarrow\rightarrow C'$ Where $B \cap C = \varnothing$ **(FD-MVD3)** If $A \rightarrow\rightarrow B$, and $AB \rightarrow C$, then $A \rightarrow C - B$

Inclusion Dependencies	
Given[Cas83][Cas84-2]	*Derived* [Mit83]
(IND1) Reflexivity $R[X] \subseteq R[X]$ if X is a sequence of distinct attributes of R	**(IND4) Substitutivity of Equivalents** If $AB \subseteq CC$, $\sigma \in \Sigma$, then τ where τ is obtained from σ by substituting A for one or more occurrences of B
(IND2) Projection & Permutation If $R[A_1 \ldots A_m] \subseteq S[B_1 \ldots B_m]$, then $R[A_{i_1} \ldots A_{i_k}] \subseteq S[B_{i_1} \cdots B_{i_k}]$ for each sequence $i_1 \ldots i_k$ of distinct integers from {1 … m}	**(IND5) Redundancy** If $U \subseteq V \subset \Sigma$ then $UX \subseteq VY$, where $X \subseteq Y$ follows from $U \subseteq V$ by a single application of IND2
(IND3) Transitivity If $R[X] \subseteq S[Y]$ and $S[Y] \subseteq T[Z]$, then $R[X] \subseteq T[Z]$	

Functional and Inclusion Dependencies[Mit83]
(FD-ID1) Pullback If $UV \subseteq XY$, $X \rightarrow Y$ then $U \rightarrow V$, where $\lvert X \rvert = \lvert U \rvert$ **(FD-ID2) Collection** If $UV \subseteq XY$, $UW \subseteq XZ$, $X \rightarrow Y$ then $UVW \subseteq XYZ$, where $\lvert X \rvert = \lvert U \rvert$ **(FD-ID3) Attribute Induction** If $U \subseteq V$, $V \rightarrow B$, then $UA \subseteq VB$, where A is an attribute that does not appear in Σ

Exclusion Dependencies[Cas83]

(EXD1)
If $R[X] \mid S[Y]$, then $S[Y] \mid R[X]$

(EXD2)
If $R[A_{1_1} \ldots A_{1_m}] \mid S[A_{1_1} \cdots A_{1_m}]$, then $R[A_1 \ldots A_n] \mid S[B_1 \ldots B_n]$

(EXD3)
If $R[X] \mid R[W]$, then $R[Y] \mid S[Z]$
Where $R[X] \mid R[W]$ is vacuous

Inclusion and Exclusion Dependencies[Cas83]

(IND-EXD1)
If $R[X] \mid R[W]$, then $R[Y] \subseteq S[Z]$
Where $R[X] \mid R[W]$ is vacuous

(IND-EXD2)
If $R[X] \subseteq S[Y]$, $T[W] \subseteq U[Z]$, and $S[Y] \mid U[Z]$, then $R[X] \mid T[W]$

Full Join Dependencies[Sci82]

(JD0)

$\varnothing \models \bowtie[X]$ for any set X

(JD1)

$\bowtie[S] \models \bowtie[S, Y]$, if $Y \subseteq S$

(JD2)

$\bowtie[S, Y, Z] \models \bowtie[S, YZ]$

(JD3)

$\{\bowtie[S, Y], \bowtie[R]\} \models \bowtie[S, R]$ if $R = Y$

(JD4)

$\bowtie[S, YA] \models \bowtie[S,Y]$ if $A \notin S$

(JD5)

$\{\bowtie[S, Y], \bowtie[R]\} \models \bowtie[S, Y_1 \cap Y \ldots Y_m \cap Y]$

Where:
$\bowtie[S,Y]$ and $\bowtie[R]$ are full JDs such that $A \subseteq Y$
and A appears in more than one relation scheme and
$R = \{Y_1 \ldots Y_m\}$

Embedded Multivalued Dependencies[Par80]

(EMVD0) Complementation
If $A \rightarrow\rightarrow B|C$, then $A \rightarrow\rightarrow C|B$

(EMVD1) Projection
If $A \rightarrow\rightarrow BC|D$, then $A \rightarrow\rightarrow B|D$

(EMVD2) Root Weighing
If $A \rightarrow\rightarrow BC|D$, then $AB \rightarrow\rightarrow C|D$

(EMVD3) Decomposition
If $A \rightarrow\rightarrow B|CD$, and $AB \rightarrow\rightarrow C|D$, then
$A \rightarrow\rightarrow C|D$

(EMVD4) Intersection
If $A \rightarrow\rightarrow B|C$, and $A \rightarrow\rightarrow D|E$, then
$A \rightarrow\rightarrow B\cap D|B\cap E$
Where $B\cap D= \varnothing$ and $B\cap E=\varnothing$

(EMVD5) Pseudo-transitivity
If $A \rightarrow\rightarrow B|CDE$ and $BC \rightarrow\rightarrow D|AF$, then
$AC \rightarrow\rightarrow D|BF$
Where A,B,C,D,E is disjoint and A,B,C,D,E,F is disjoint

Functional Dependencies and Embedded Multivalued Dependencies[Par80]

(FD-EMVD1)
If $A \rightarrow\rightarrow B|C$ and $B \rightarrow C$, then $A \rightarrow C$

(FD-EMVD2)
If $A \rightarrow\rightarrow B|C$ and $AB \rightarrow D$, then $A \rightarrow\rightarrow BD|C$

Multivalued Dependencies and Embedded Multivalued Dependencies[Par80]
(MVD-EMVD1) Joinability If A →→ C, then AB →→ C and A →→ B|C **(MVD-EMVD2) Union** If AB →→ DE and AC →→ DF and A →→ B|C, then A →→DEF Where F ⊆ B and E ⊆ C

Transitive Dependencies[Par80]	
(TD1) Reflexivity If B ⊆ A, or C ⊆ A then A(B,C)	**(TD5) Transitivity** If A(B,C) and B(C, D) then AD(B,C)
(TD2) Symmetry / Complementation Iff A(C,B), then A(B, C)	**(TD6) Union** If A(B,D), A(C,D) and D(B, CD), then A(BC, D) **(TD6a)** If A(B, CD) and A(C,D), then A(BC, D) **(TD6b)** If A(BE, D), A(CE, D), DE(B,CD), then A(BC,D)
(TD3) Projection If A(BC, D), then A(B,D)	**(TD7) Root Shifting** If AB(AC, BD), then AB(C, ABD) Where A,B,C,D are disjoint
(TD4) Root Augmentation If A(B,C), then AD(B,C)	**(TD8)** If A(B,D), B(C, DX), D(E, BX), X(BE,CD), then A(BC, DE)

APPENDIX C

GLOSSARY

Attribute
Analogous to a column in a table, an attribute is a property of an entity

Attribute, Non-prime
An attribute not in a candidate key of a relation. Sometimes called a *non-key attribute.*

Attribute, Prime
An attribute in a candidate key of a relation. Sometimes called a *key attribute.*

Closure
The set of all functional and multivalued dependencies implied by a given set of functional and multivalued dependences.

If D is the given set, then we notate the closure as D^+.

Closure, Minimal
A set of FDs (G), derived from a set of given FDs (D), where:

$G^+ = D^+$ (We can derive all the same dependencies from G as we can from D)
Every FD in G has only one attribute on the right side
Every FD in G is required
Every FD in G is elementary

Decomposition
Splitting a relation into two or more relations.

Decomposition, Lossless
Decomposition that when joined to form the original relation contains the same data.

Decomposition, Lossless, Minimal
A lossless decomposition is *minimal* if no proper subset of that decomposition is itself as lossless decomposition.

See Also: Dependency, Join, Strong-Reduced

Dependency, Domain
A is an attribute and S is a set of values. The *domain dependency* IN(A, S) indicates every A value must be in set S.

Dependency, Exclusion
The *exclusion dependency* $R_i[X] \mid R_j[Y]$ is valid iff $R_1[X]$ and $R_j[Y]$ are disjoint. Abbreviated EXD.

Dependency, Exclusion, Trivial
If R_i is in a vacuous EXD, then any EXD of the form $R_j [X_j] \mid R_i [X_i]$, where $|X_j| = |X_i|$ is always valid, and therefore *trivial*.

Dependency, Exclusion, Vacuous
The *exclusion dependency* $R_i[X] \mid R_j[Y]$ is *vacuous* iff $R_1 = \emptyset$.

Dependency, Functional
If A determines B, that is for any given value of A there is exactly one corresponding value of B, there is the *functional dependency* we notate as A → B. Abbreviated as FD.

Dependency, Functional, Direct
If A → C, and there is no B such that A → B and B → C, then there is a *direct functional dependency.* This is also called a *non-transitive dependency*.

Dependency, Functional, Elementary
A non-trivial full functional dependency.

Dependency, Functional, Full
If $A \rightarrow B$, and A has two or more attributes, and B is not dependent on any subset of A, this is a *full functional dependency*.

Dependency, Functional, Single-Valued

See Dependency, Functional

Dependency, Functional, Transitive
If $A \rightarrow B$ and $B \rightarrow C$, then there is a *transitive functional dependency* between A and C. This is often shortened to *transitive dependency*.

Dependency, Functional, Transitive, Strict
If $A \rightarrow B$ and $B \rightarrow C$, but $C \rightarrow B$ and $B \rightarrow C$ do not hold, there is a *strict transitive functional dependency* between A and C. This is often shortened to *strict transitive dependency*.

Dependency, Functional, Trivial
If B is a subset of A, then $A \rightarrow B$. This is a *trivial functional dependency*. If A only contains one attribute, this means $A \rightarrow A$, which is also a trivial functional dependency.

Dependency, Inclusion
The *inclusion dependency* $R\ [A_1 \ldots A_m] \subseteq S\ [B_1 \ldots B_m]$ holds for a database if for each tuple in relation R is also in relation S. R and S could be the same relation. Abbreviated as IND.

If the inclusion dependency $R_i[X] \subseteq R_j[Y]$ is valid, then $R_i[X]$ is a subset of $R_j[Y]$.

Dependency, Join
If R is the join of its projections $R[X_1] \ldots R[X_n]$, we say it obeys the *join dependency* $*\{X_1 \ldots X_n\}$

Dependency, Join, Reduced

A join dependency $*\{X_1 \ldots X_n\}$ is *reduced* if it is total and $R_i \nsubseteq R_j$ for all i, j, $i \neq j$

Dependency, Join, Strong-Reduced
Let Σ be a set of FDs and JDs. A join dependency $*\{X_1 \ldots X_n\}$ in Σ^+ is *strong-reduced* if it is total and for every component the JD obtained by removing that component is either not in Σ^+ or it is not total.

See Also: Decomposition, Lossless, Minimal

Dependency, Join, Total
A join dependency $*\{X_1 \ldots X_n\}$ is *total* if $R = X_1 \ldots X_n$

Dependency, Join, Trivial
For the projections $R[X_1] \ldots R[X_n]$, a *trivial join dependency* exists in the join dependency $*\{X_1 \ldots X_n\}$ if one of the X_i is R.

Dependency, Key
In a relation R with attributes X, the *key dependency* KEY (K) says that K is a superkey.

Dependency, Multivalued
If A determines a specific set of possible values for B, there is a *multivalued dependency* we notate as $A \rightarrow\rightarrow \{B\}$. Abbreviated as MVD.

In R(X, Y, Z), $X \rightarrow\rightarrow Y$ if Y_{XZ} depends on Y, and is independent of Z: a multivalued dependency.

Dependency, Multivalued, Embedded
An *embedded multivalued dependency* $X \rightarrow\rightarrow Y|Z$ exists when X, Y, and Z are all in the same relation.

Dependency, Multivalued, Essentially
A multivalued dependency $X \rightarrow\rightarrow Y|Z$ is *essentially multivalued* if it is not trivial and neither $X \rightarrow\rightarrow Y$ nor $X \rightarrow\rightarrow Y$ is a functional dependency.

Dependency, Multivalued, Trivial
A relation has a *trivial multivalued dependency* when only those attributes involved in the dependency are in the relation.
In R(X, Y), if $X \rightarrow\rightarrow Y$, a trivial multivalued dependency.
In R(X, Y, Z), if $(X, Y) \rightarrow\rightarrow Z$, a trivial multivalued dependency.

Dependency Preservation
A decomposition that groups the same function dependencies that existed in the original relation is said to observe *dependency preservation*.

Iff
If, and only if

Join
A *join* of two relations on a common attribute links the tuples of the two relations that share that attribute. Unless otherwise noted, *join* means a *natural* or *equal join.*

X_1	
Store	SName
27	Apple House
13	Mint Shop

X_2	
Store	Location
27	East
13	West

Join on Store

$R = X_1 \bowtie X_2$		
Store	SName	Location
27	Apple House	East
13	Mint Shop	West

Key, Candidate
An attribute or set of attributes that uniquely identifies a tuple, and has no attribute not necessary to uniquely identify a tuple. Usually, this is what is meant if only the word *key* is used.

Key, Elementary
A key (X) is said to be an *elementary key*, if for some attribute (A), X → A is an elementary functional dependency

Key, Super
An attribute or set of attributes that uniquely identifies a tuple. Equivalently, a set of attribute that contains a candidate key.

Projection
A *projection* of a table for some attributes X, is the set of tuples consisting only of those attributes in X. If two resulting sets of tuples are the same, they are merged into a single tuple.

Store	SName	Employee
27	Apple House	Smith
27	Apple House	Jones
27	Apple House	Franks

Projection on X, where X{Store, SName}

Store	SName
27	Apple House

Relation
Analogous to a table, a *relation* is a collection of data organized according to certain rules, representing a part of a (or a complete) entity.

Tuple
Analogous to a row of a table, a *tuple* represents a specific occurrence or example of a relation.

APPENDIX D

CHEAT SHEET

1NF

A direct and faithful representation of a relation.

- The rows are not ordered.
- The columns are not ordered.
- There are no duplicate rows.
- Every row/column intersection contains exactly one domain value and nothing else.
- No irregular columns.

Functional Dependencies

2NF

Every non-prime attribute is fully functionally dependent on each candidate key.

- An attribute is non-prime if it is not in a candidate key.
- A candidate key identifies a record and contains only attributes required to do so.
- Fully functionally dependent means on every attribute in the key.

3NF

Every non-prime attribute is directly dependent on each candidate key.

- Directly dependent means A → C, not A → B → C

BCNF

Every determinant is a candidate key.

- A determinant is one value that determines another value (one-to-one).

Multivalued Dependencies

4NF

There are no independent non-trivial multivalued dependencies.

- A multivalued dependency is one value determining a set of other values (one-to-many).
- Independent means there are two or more present.
- Non-trivial means the right hand side is not exactly equal to the left hand side.

Join Dependencies

PJ/NF

Every nontrivial join dependency is the result of a key.

- A join dependency is a set of projections that join to form the original table.
- Non-trivial means that one of the projections is not the same as the original table.

APPENDIX E

OTHER NORMAL FORMS

There are many other normal forms, not covered in the main body of this paper, which address various aspects of database design. We define and briefly discuss these normal forms in this appendix.

TNF	Third Normal Form
EKNF	Elementary Key Normal Form
(3,3)NF	(3,3)NF
LTKNF	Improved Third Normal Form
	Improved Boyce-Codd Normal Form
PSJU/NF	Projection Split Join Union Normal Form
	Overstrong PJ/NF
5NF	Project-Join Normal Form
	Tight Fourth Normal Form
5NFR	Reduced Fifth Normal Form
KCNF	Key Complete Normal Form
RFNF	Redundancy Free Normal Form
Q-5NF	Q-5NF
SNF	Superkey Normal Form
DK/NF	Domain Key Normal Form
IDNF	Inclusion Dependency Normal Form
IN-NF	Inclusion Normal Form
VRFNF	Value Redundancy Free Normal Form
ARFNF	Attribute Redundancy Free Normal Form
RFNF	Redundancy Free Normal Form
IDNF	Inclusion Dependency Normal Form
IFIDNF	Interaction Free Inclusion Dependency Normal Form
ERNF	Entity-Relationship Normal Form
ONF	Object Normal Form
6NF	Sixth Normal Form
ONF	Object-Normal Form
4ONF	Fourth Object-Normal Form
5NF	Fifth Normal Form
5ONF	Fifth Object-Normal Form
6NF	Sixth Normal Form
6ONF	Sixth Object-Normal Form
	“Restriction-union” Normal Form

Notation used in this appendix	
R	A relation
σ	A dependency (type of which will be specified)
J	Set of JDs
I	Set of INDs
E	Set of EXDs
D	A database scheme

“Unacceptable File Operations in a Relational Data Base”
Proc. ACM SIGFIDET Workshop, San Diego, 1971
I.J. HEATH

Third Normal Form (TNF)

R is in TNF if:

- Every determinant is an ID
- No field is dependent on two determinants

An ID is a candidate key where no attribute in that candidate key is dependent on another attribute within the candidate key.

This is equivalent to BCNF, and is considered the first published definition of BCNF, even though it uses the name “Third Normal Form” [Dat04].

This paper is also the source of *Heath’s Theorem* which is given exactly in [Hea71] as:

Theorem: A relation R (A, B, C), where A determines B, is the natural join J of R [A, B] and R [A, C].
(Note: R [A, B] means the projection of R on fields A and B.)

Heath’s Theorem explains why we can have a lossless decomposition for certain relations, although it does not explain why some decompositions are lossy [Dat04]. [Fag77] has a stronger version of Heath’s Theorem which includes multivalued dependencies for 4NF, and cover lossless and lossy decompositions.

Elementary Key Normal Form

“A New Normal Form for the Design of Relational Database Schemata”
ACM Transactions on Database Systems 7(3), September 1982
CARLO ZANIOLO

Restricting our discussion to functional dependencies, we can still be unsatisfied in the area of 3NF – BCNF. 3NF is not strict enough, and BCNF is too strict, in the sense that not all relations can be put in BCNF and retain all functional dependencies. It is also unclear from our main definitions exactly *why* BCNF is a stricter form than 3NF.

The main concept to introduce here is the *elementary functional dependency*, which is simply a non-trivial full functional dependency. It follows from here that an *elementary key* is a key having elementary functional dependency, and that an *elementary key attribute* is an attribute belonging to an elementary key.

Armed with this new concept, we can achieve two interesting things. First, we can introduce a new normal form, *Elementary Key Normal Form*, which “lies between” 3NF and BCNF, and we can give alternative definitions for 3NF and BCNF that make it clear there is an increase in the restrictions we apply to the relations and we move from 3NF to BCNF.

Here are the new definitions, including the new normal form.

R is in the stated normal form iff for every elementary FD of R, say X → A:

3NF

1. X is a key for R, or
2. A is a key attribute for R.

EKNF

1. X is a key for R, or
2. A is a elementary key attribute for R.

BCNF

1. X is a key for R

Note how each form gradually tightens the exception the second rule provides, until it is removed altogether for BCNF. This clearly illustrates BCNF is a stricter form than 3NF.

Because of the above elegance of the new definitions it is also easy to see that:

$$\text{BCNF} \Rightarrow \text{EKNF} \Rightarrow \text{3NF}$$

Finally, it is proved that the algorithm to put relations in 3NF in [Ber76] actually results in relations that are in EKNF. It follows then that all relations can be put in EKNF.

(3, 3)NF

“A Normal Form for Abstract Syntax”
Proc. 4th Conf. Very Large Data Bases, Berlin, 1978
J.M. Smith

(3, 3)NF is closely related to BCNF. First, compare the definitions:

BCNF

R is in BCNF if each nontrivial FD in R is from a key in R.

(3,3)NF

R is in (3,3)NF if for each subset of R, each nontrivial FD in that subset is from a key in R.

The first “3” in (3, 3) NF corresponds to BCNF (BCNF is sometimes called “Boyce-Codd Third Normal Form” or just “Third Normal Form”), in that the *components* of the relation are under consideration.

The second “3” in (3, 3) NF indicates that the *categories* of the attributes are under similar consideration. Since this is where (3, 3) NF differs from BCNF (and indeed differs from almost every other normal form except PSJU/NF and DK/NF), we want to examine this aspect further.

From one of several examples in [Smi78], we consider this table, call it P_ROLE:

Person	**Role**
P1	man
P1	musician
P1	engineer
P2	woman
P2	firefighter

Accept that a person has two *types* of roles: a sex role, of which there is exactly one for each person, and a professional role, of which there can be many for each person.

The above table is in PJ/NF[19], yet there is an obvious problem, namely nothing prevents the insertion of the tuple (P1, woman), which violates our rule that a person has only one sex role.

The decomposition is simple, but it is not as we usually perform decomposition through projections, but rather through a *split*:

Person	SexRole
P1	man
P2	woman

Person	ProfessionalRole
P1	musician
P1	engineer
P2	firefighter

Note that it follows from the definition that (3, 3) NF ⇒ BCNF.

---

[19] It is not, however, in PSJU/NF or DK/NF. In fact, this same example is used in [Fag81] in explaining DK/NF and in [Fag79] for PSJU/NF.

Improved Third Normal Form
Improved Boyce-Codd Normal Form

“An Improved Third Normal Form for Relational Databases”
ACM Transactions on Database Systems 6(2), June 1981
T. LING, F. TOMPA, and T. KAMEDA

Improved 3NF

R_i in a preparatory relation PR is in *improved third normal form* if each nonessential attribute is not restorable in R_i.

Similar to Optimal 3NF, Improved 3NF considers a set of relations.

Consider Example 1 from [Lin81]:
Given a relation R, this is our preparatory relation:
{A, B, C, D, E, F}
Assume these FDs[20]:
{AB → CD, A → E, B → F, EF → C}
And consider this decomposition into a set of 3NF relations, with the primary key underlined:
R_1 : $\underline{\text{AB}}$CD
R_2 : $\underline{\text{A}}$E
R_3 : $\underline{\text{B}}$F
R_4 : $\underline{\text{EF}}$C

The problem is there is no way to ensure the C value in R_1 corresponds to the C value in R_4.

The C in R_1 is not transitively dependent on AB, but it is *superfluous* meaning it is *restorable and nonessential*.

It is *restorable* because its value in R_1 can be derived from the other attributes in PR.
It is *nonessential* because it value is not required to derive the value of any other attributes in PR.

So, while R_4 is in 3NF, it is not in Improved 3NF. It is also shown that if a relation is in Improved 3NF, then it is in 3NF as well.

Improved 3NF is sometimes called LTK Normal Form, after the originators.

Improved BCNF

R_i in a preparatory relation PR is in *improved Boyce-Codd normal form* if no attribute is restorable in R_i.

It is shown that a relation in improved BCNF is also in both BCNF and improved 3NF.

See also Inclusion Normal Form (IN-NF) for an extension of this approach.

---

[20] A possible set of values:
A: Model Number, B: Serial Number, C: Price, D: Color, E: Model Name, F: Year of Manufacture

Overstrong PJ/NF
Projection Split Join Union Normal Form

“Normal Forms and Relational Database Operators”
ACM SIGMOD Conference, 1979
RONALD FAGIN

In the same paper that introduced PJ/NF (commonly called 5NF), Fagin touched on two variations of PJ/NF. We briefly describe them here.

Compare PJ/NF to overstrong PJ/NF, both definitions from [Fag79]

PJ/NF

R is in PJ/NF if $\mathbf{K} \models \sigma$.

Where σ is each JD.

Thus, every JD is the result of keys.

Overstrong PJ/NF

R is in overstrong PJ/NF if, there is a key dependency $K \rightarrow X$ of R such that $K \rightarrow X \models \sigma$.

Where σ is each JD.

Thus, every JD is the result of a key.

Fagin provides proof the two definitions are not equivalent, and that overstrong PJ/NF is “too demanding”.

Projection Split Join Union Normal Form (PSJU/NF)

R is in PSJU/NF if:

- It is in PJ/NF, and
- There is no way to split R into R_1 and R_2 where the set of dependencies that hold in R_1 is not the same as the set of dependencies that hold in R_2

Since PJ/NF is framed in terms of the two operations projection and join, Fagin suggests this normal form based on four operations.

The split operation is described as the inverse of the union, and is notated with Ψ.

[Fag79] gives the above as a “possible definition”, noting it requires modifications. The “major defect” identified is what restrictions the split operation should have; Fagin raises some points but leaves the details as an “interesting research problem”.

PSJU/NF is closely related to (3, 3) NF[Smi78], discussed earlier in this appendix.

Tight Fourth Normal Form
Project-Join Normal Form

The Theory of Relational Databases
Computer Science Press, 1983
D. MAIER

Tight 4NF

R is in *tight fourth normal form* if:

1. R is in 4NF and
2. R can be obtained by a series of *tight decompositions.*

A decomposition (R, S) is a *tight decomposition* if there is no other decomposition (R', S') such that $R' \cap S'$ is properly contained in $R \cap S$. That is, the overlap of R and S is minimal.

Compare project-join normal form (5NF) with projection-join normal form (PJ/NF):

R is in the given normal form if for every JD $*\{R_1, R2 \ldots R_p\}$ implied by $\{\boldsymbol{f}, \mathbf{J}\}$:

Project-Join Normal Form (5NF)

1. The JD is trivial or
2. Every R_i is a superkey for R

Projection-Join Normal Form (PJ/NF)

1. $*\{R_1,R_2 \ldots R_p\}$ is implied by the key FDs of R

The definition of 5NF then is that every component of every nontrivial JD is a key.

[Vin97] points out several problems with this requirement see the discussion of Reduced Fifth Normal Form (5NFR). [Kho02] notes the same problem, and proposes a different solution.

Reduced Fifth Normal Form

"A Corrected 5NF Definition for Relational Database Design"
Theoretical Computer Science 185, 1997
MILLIST W. VINCENT

In this paper, some deficiencies are identified with the 5NF definition given in [Mai83]:

- It does not generalize 4NF.
- It requires every attribute to be a superkey.

The second in particular is problematic, because it is a requirement that is "virtually impossible to achieve in practice and is clearly not what was intended in the introduction of 5NF" [Vin97].

To correct these deficiencies, 5NFR is introduced.

Compare the definitions, as given in [Vin97].

R is in the given normal form with respect to **{*f*, J}** if:

Project-Join Normal Form (5NF)

For every $\sigma \in \{f, J\}^+$, every component of σ is a superkey.
Where σ is a nontrivial total JD

Reduced 5NF (5NFR)

For every $\sigma \in \{f, J\}^+$, every component of σ is a superkey.
Where σ is a nontrivial strong-reduced total JD

A JD $*\{R_1, \ldots R_p\}$ in Σ^+ is *strong-reduced* if it is total and for every component R_i the JD obtained by removing R_i from $*\{R_1, \ldots R_p\}$ is either not in Σ^+ or it is not total.

One of the examples (Example 3.3) given in [Vin97] is:

$R = \{A, B, C\}$ and $\Sigma = \{A \rightarrow BC\}$. This means Σ implies the JD $*\{AB, AC, BC\}$. The problem here is the component BC is not a superkey. BC is not needed in that the JD with it removed, $*\{AB, AC\}$ is also in Σ^+.

Thus $*\{AB, AC, BC\}$ is **not** a strong-reduced join dependency.

[Vin97] then shows that the 5NFR definition:

- Generalizes and implies 4NF
- Is weaker than both 5NF and PJ/NF

5NFR meets the design goal of minimizing storage.

Key Complete Normal Form
Redundancy Free Normal Form

"Redundancy Elimination and a New Normal Form for Relational Database Design"
Semantics in Databases, Springer-Verlag LNCS 1358, 1998
MILLIST W. VINCENT

In addressing the design goal of elimination of redundancy, [Vin98] introduces KCNF. On the way to doing so, RFNF is introduced.

The two are equivalent, but KCNF is a *syntactic* normal form (like BCNF and 4NF), where RFNF is a *semantic* normal form. The argument for this approach is given:

> The difference between the two types of normal forms is that semantic normal forms directly specify the set of relations with the desired design properties, whereas the syntactic normal forms only do this indirectly….We believe that semantic normal forms, by making explicit the aims of database design, are the appropriate starting point….

Redundancy Free Normal Form (RFNF)

R is in RFNF with respect to $\{ \boldsymbol{f}, \mathbf{J} \}$ if:

1. There does not exist a relation in SAT ($\{\boldsymbol{f}, \mathbf{J}\}$) that is redundant.

The set of all relations which satisfy $\{\boldsymbol{f}, \mathbf{J}\}$ constraints is: SAT ($\{\boldsymbol{f}, \mathbf{J}\}$). A relation in this set is *r*.
t is a tuple in *r*.

t [A] is *redundant* if:
For every replacement of t [A] by a value A' such that $t\,[A] \neq A'$ and resulting in a new relation r', then $r' \in$ SAT ($\{\boldsymbol{f}, \mathbf{J}\}$).

There is an alternative definition of RFNF in [Lev00].

Key Complete Normal Form (KCNF)

R is in KCNF with respect to $\{ \boldsymbol{f}, \mathbf{J} \}$ if:

1. The left hand side of every nontrivial FD in $\{ \boldsymbol{f}, \mathbf{J} \}$ is a superkey, and
2. Every JD in $\{\boldsymbol{f}, \mathbf{J}\}$ is key complete.

A JD $*\{R_1, \ldots, R_n\} \in \{\boldsymbol{f}, \mathbf{J}\}^+$ is *key complete* if the union of its components which are superkeys is equal to R

[Vin98] also demonstrates that: PJ/NF ⇒ 5NFR ⇒ KCNF

It may be helpful to summarize these various fifth normal forms:

Form	Strictness	Aspect	Introduced in:
KCNF	Least	Redundancy Elimination	1998
5NFR	.	Storage Minimization	1997
PJ/NF	.	No key-based update anomalies	1979
5NF	Most	Storage Minimization	1983

Not only are the forms increasingly strict as the table shows, but [Vin98-2] also shows:
5NF ⇒ PJ/NF ⇒ 5NFR ⇒ KCNF

Q-5NF

“Understanding the Fifth Normal Form (5NF)”
Australian Computer Science Communications 14(1), 1992
M. ORLOWSKA and Y. ZHANG

Q-5NF is mentioned in [Vin97], where it is described as being shown equivalent to PJ/NF in the paper that introduces it.

Q-5NF

R is in Q-5NF iff:

For any JD of R JD, *{R_1, R_2, …, R_m) the intersection graph is *superkey connected*.

Superkey Normal Form

“Minimal Lossless Decompositions and Some Normal Forms between 4NF and PJ/NF”
Information Systems 23(7), 1988
RAGNAR NORMANN

Superkey Normal Form is closely related to 5NFR. This means [Nor98] and [Vin97] cover much of the same ground. [Nor78] even opens with the same example we saw in 5NFR, noting that it meets PJ/NF, but not 5NF.

Superkey Normal Form (SNF)

R is in SNF if:

1. Every component of every minimal lossless decomposition of R is a superkey in R.

A lossless decomposition of R is said to be *minimal* if no proper subset of that decomposition is also a lossless decomposition of R. The means a minimal lossless decomposition is equivalent to [Vin97] strong-reduced JD. This in turn means SNF is equivalent to 5NFR.

Superkey Normal Form k (SNF_k)

R is in SNF_k if:

1. All minimal lossless decompositions with at most k components consist of superkeys only.

[Nor98] then shows:

1. $SNF_1 = 1NF$
2. $SNF_2 = 4NF$
3. $SNF_{k+1} \subset SNF_k$
4. $SNF \subset SNF_k$
5. $5NF \subset SNF$

Finally, [Nor98] gives an example of a relation in SNK_3 which is not in 5NF. This is what is referred to in the title as a form “between 4NF and PJ/NF”[21].

---

[21] [Nor98] is using “PJ/NF” to refer to Maier’s project-join normal form, which we refer to as 5NF.

Domain Key Normal Form

"A Normal Form for Relational Databases that is based on Domains and Keys"
ACM Transactions on Database Systems 6(3), September 1981
RONALD FAGIN

A relation R is in DK/NF if every constraint in R can be inferred by the domain and key dependencies. Restated, a relation in DK/NF if, by enforcing the DDs and KDs, every schema constraint is also enforced [Fag81].

The problem with DK/NF is two-fold: one, it is not possible to put every relation in DK/NF, and secondly, it's not possible to know exactly when a relation can be put in DK/NF. This means DK/NF is more of a goal to strive for than actually achieve.

The main reason for this problem is that a relation may have constraints that DDs and KDs simply can not address. A common example of this type of constraint is requiring that a table contain *x* number of records – a cardinality constraint.

That being said, [Fag81] has some interesting re-definitions of the common higher normal forms, and proves that the new definitions are equivalent to the original, provided no domain is too small.

Let Γ be the set of DDs and KDs of R. R is in the stated normal form if:

BCNF'

$\Gamma \models \sigma$

Where σ is each FD

4NF'

$\Gamma \models \sigma$

Where σ is each MVD

PJ/NF'

$\Gamma \models \sigma$

Where σ is each JD

DK/NF

$\Gamma \models \sigma$

Where σ is every constraint

It is further proved that:

$$\text{DK/NF} \Rightarrow \text{PJ/NF'} \Rightarrow \text{4NF'} \Rightarrow \text{BCNF'}$$

Finally, [Fag81] suggests a generalization of DK/NF, called $\mathscr{C}$ normal form, where $\mathscr{C}$ is a certain class of constraints.

A relation is in $\mathscr{C}$ normal form, if the set of constraints of the relation is the set of logical consequences from the constraints in class $\mathscr{C}$.

So, for example, if the class $\mathscr{C}$ is the class of all DDs and KDs then $\mathscr{C}$ normal form is DK/NF.

Inclusion Dependency Normal Form

“Inclusion Dependencies in Database Design”
Proc. Int’l Conf. Data Engineering, Los Angeles, 1986
HEIKKI MANNILA and KARI-JOUKO RÄIHÄ

Inclusion Dependency Normal Form (IDNF)

D is in IDNF if:

1. Every R in **D** is in 3NF
2. Every IND is key-based
3. The set of INDs is noncircular

An inclusion dependency $R[X] \subseteq S[Y]$ is *key-based* if Y is a key of S.

A set of INDs is *circular* if:

1. There exists a relation R with distinct attributes X and Y such that $R[X]\subseteq R[Y]$ or
2. There exist relations $R_1 \dots R_n$, where $R_1[X_1]\subseteq R_2[Y_2], \dots R_n[X_n]\subseteq R_1[Y_1]$

3NF instead of BCNF was chosen as a requirement because, according to [Man86]: “it is not always possible to obtain schemes in BCNF if dependency preservation and lossless joins are also desired.”

[Man86] identifies 3 situations where INDs are used, with the general form $R[X] \subseteq S[Y]$:

1. Abstraction
RegisteredCars [Model] $\subseteq$ CarTypes [Model]
Here, each registered car is an instance of a car type.

2. Specialization
SportsCars [Model] $\subseteq$ CarTypes [Model]
Here, sports cars contain additional information not relevant to other car types.

3. Existency Constraints
Yachts [Owner] $\subseteq$ RegisteredCars [Owner]
Here the implication is yachts can only be owned by those owners owning cars.

In the first two, Y is a key of S, but in examples of the third type Y may or may not be a key.

[Man86] makes the argument that existency constraints should not influence the design process, and therefore restrict IDNF to considering key-based INDs.

This argument is questioned in [Lin92], where Inclusion Normal Form (IN-NF) is introduced. [Lin92] also suggests that IDNF does not address the relational design goals of minimality and avoiding update anomalies.

Noncircular INDs are required because if the set of INDs was circular, no inserts could be made without violating one or more INDs.

A slightly different definition of IDNF is given in [Lev00].

Inclusion Normal Form

“Logical Database Design with Inclusion Dependencies”
Proc. Int’l Conf. Data Engineering, Tempe, 1992
TOK WANG LING and CHENG HIAN GOH

Inclusion Normal Form (IN-NF)

D is in IN-NF if:

1. There are no *weakly superfluous attributes* in any of the relation schemes.

D has the universal relation scheme R_0 and set of FDs Σ is one where:

1. The set of FDs implied by all keys forms a minimal cover for Σ
2. No 2 relation schemes have equivalent keys
3. $R_1 \ldots R_n$ form a nonloss decomposition of R_0

A *weakly superfluous attribute* is both *weakly restorable* and *weakly nonessential.*

Let $D = \{R_1 \ldots R_n\}$ be a preparatory database scheme. Let B be some attribute in R_i. Let Φ be the set of INDs in D. Let G'_i be a minimal closure for R_i. Let “derived” in the next two definitions mean “derived from $G_i(B) \cup \Phi$ using inference rules”

An attribute B is *weakly restorable* if there exists a key K of R_i, not containing B, such that $K \rightarrow B$ can be derived. If an attribute is restorable it is weakly restorable.

An attribute B is *weakly nonessential* if there exists another key K’ of R_I, not containing B, such that $K \rightarrow K'$ can be derived. If an attribute is nonessential it is weakly nonessential.

Consider an example from [Lin92] with this modeling:

R_1 (<u>Employee</u>, EName, Office)
R_2 (<u>Office</u>, <u>Phone</u>, Dept)
R_3 (<u>Manager</u>, MName, MPhone, <u>Dept</u>)

FDs	INDs
Employee → EName, Office	$R_3[\text{Manager, MName}] \subseteq R_1[\text{Employee, EName}]$
Office → Phone, Dept	$R_3[\text{Manager, Dept}] \subseteq (R_1 \bowtie R_2)[\text{Employee, Dept}]$
Phone → Office	R3[Manager, MPhone] $\subseteq (R_1 \bowtie R_2)$[Employee, Phone]
Manager → MName, MPhone, Dept	
Dept→ Manager	

The first IND means every manager is an employee. The second IND means if an employee is a manager of a dept, they must belong to that dept. The third follows from the first.

Each relation in the example is in 5NF and Improved 3NF[22], yet there is redundancy. MName and MPhone can both be removed from R_3 because they are both weakly superfluous. After this removal, IN-NF is satisfied.

It is shown in [Lin92] that:

IN-NF $\Rightarrow$ Improved 3NF
IN-NF $\Rightarrow$ BCNF

---

[22] See [Lin92] for details. R_3.Dept is restorable but **not** weakly nonessential, and R_3.MName is weakly nonessential, but **not** nonessential, etc.

Value Redundancy Free Normal Form
Attribute Redundancy Free Normal Form
Redundancy Free Normal Form
Inclusion Dependency Normal Form
Interaction Free Inclusion Dependency Normal Form

“Justification for Inclusion Dependency Normal Form”
IEEE Transactions on Knowledge and Engineering 12(2), March/April 2000
MARK LEVENE and MILLIST W. VINCENT

Let Σ be the combined sets of $\{\boldsymbol{f}, \mathbf{I}\}$

Value Redundancy Free Normal Form (VRFNF)

D is in VRFNF if there does not exist a database *d* over **R** and an occurrence of a value t [A] that is redundant in *d* with respect to f.

Attribute Redundancy Free Normal Form (ARFNR)

D is in ARFNF if there does not exist an attribute A in a relation schema $R \in \mathbf{D}$ which is redundant with respect to Σ.

Redundancy Free Normal Form (RFNF)

D is in RFNF if it is in VRFNF and ARFNF

Inclusion Dependency Normal Form (IDNF)

D is in IDNF if:

- **D** is in BCNF with respect to f and
- **I** is noncircular and key-based

Note that this differs slightly from the IDNF definition given in [Man86].

Interaction Free Inclusion Dependency Normal Form (IFIDNF)

D is said to be in IFIDNF if:

- **D** is in BCNF with respect to f and
- All INDs in **I** are either key-based or express pair-wise consistency
- f and **I** do not interact.

Two relation schemes R and S are *consistent* if I includes the two INDs:
$R[R \cap S] \subseteq S[R \cap S]$ and $S[R \cap S] \subseteq R[R \cap S]$

D is *pair-wise consistent* if every pair of its relation schemas are consistent.

f and **I** do not interact if:

- For all FDs α over **D**, for all subsets $G \subseteq f$, $G \cup \mathbf{I} \models \alpha$ iff $G \models \alpha$, and
- For all INDs β over **D**, for all subsets $J \subseteq \mathbf{I}$, $f \cup \mathbf{I} \models \beta$ iff $J \models \beta$

“Mapping Uninterrupted Schemes into Entity-Relationship Diagrams:
Two Applications to Conceptual Schema Design”
IBM Journal Res. Develop. 28(1), January 1984
MARCO A. CASANOVA and JOSE E. AMARAL de SA

Entity-Relationship Normal Form (ERNF)

A relational scheme is said to be in ERNF iff:

1. Each relation scheme is in BCNF
2. Each IND $R[X] \subseteq S[Y]$ is such that Y is a key of S
3. The relational scheme defines an ER-schema

[Cas84] gives us these definitions:

Let **D** be a set of relation schemes and Σ be a well-formed set of references. Let R be a relation scheme in **D**.

a. R defines an *entity type* in Σ iff Σ contains no reference of the form $R[K] \subset S[L]$, for any S in **D**
b. R defines a *weak entity type* in Σ iff Σ contains a single reference of the form $R [K] \subset S [L]$, for some S in **D**, and K intersects every key of R.
c. R defines a *relationship type* in Σ iff Σ contains a set of references of the form $R[K_1] \subset S_1[L_1] \dots R[K_m] \subset S_m[L_m]$ such that $K_1 \cup \dots \cup K_m$ is a key of R.
d. R defines an *ER-object* iff R defines either an entity type, a weak entity type, or a relationship type.
e. **D** and Σ define an *ER-schema* iff each relation scheme in **D** defines an ER-object.

ERNF is an attempt to address two problems [Cas84]:

1. How to define a relational database schema that can be interpreted at higher level concepts.
2. How to obtain a relational database schema from the description of a conventional file system.

Object Normal Form

“Objects in Relational Database Schemes with Functional, Inclusion, and Exclusion Dependencies”
3^{rd} *Symp. of Math. Fund. of Database and Knowledge Systems*, Rostock, 1991
JOACHIM BISKUP and PRATUL DUBLISH

D is in the stated normal form iff/if for each (X, i) $\in$ LHS (f), <R_i, f_i> is in BCNF, and:

Weak Object Normal Form (Weak ONF)

(iff) if $R_i[Z]$ occurs in a non-trivial EXD in $(f \cup I \cup E)^+$ then Z is not a subsequence of X.

Strong Object Normal Form (Strong ONF)

(iff) X is the unique minimal key of <R_i, f_i> and Ri does not occur in any non-trivial EXD in $(f \cup I \cup E)^+$

With the above definitions, there are no sound and complete procedures to test if a given database scheme is in weak or strong ONF. Because of this, the modified definitions are given, in which [Bis91] proves are testable in polynomial time.

Weak Object Normal Form *(modified)*

(if) if $R_i[Z]$ occurs in a non-trivial EXD in $(f \cup I \cup E)^+$ then Z is not a subsequence of X.

Strong Object Normal Form *(modified)*

(if) X is the unique minimal key of R_i and R_i does not occur in any non-trivial EXD in E.

Sixth Normal Form

Temporal Data and the Relational Model
Morgan Kaufmann, 2002
C.J. DATE, HUGH DARWEN, and NIKOS A. LORENTZOS

Sixth Normal Form (6NF)

R is in 6NF iff
It satisfies no nontrivial JDs.

Although 6NF is always achievable and can be applied to all relational databases, it is in temporal databases where this level of decomposition becomes most important.

Consider the example in [Dat02] (which is also given in [Dat04]):

SSSC_DURING {<u>S#</u>, SName, Status, City, <u>During</u>)

Where S# is a supplier number, SName is a supplier name, Status is some status code, City is the city the supplier is located in, and During a timestamp of the form [dX: dY], where dX indicates a starting day and dY indicates an ending day. This gives the “during” period.

Accept that in a temporal database this is a single value of type INTERVAL and does not violate 1NF.

The above is in PJ/NF, because all JDs are trivial (and the result of a key), yet there are still potential problems.

Specifically, any of the non-key attributes could change independently of the others during the time stamped period. This would lead to multiple update operations.

The decomposition into 6NF is referred to as *vertical decomposition* and is as follows:

S_DURING{<u>S#</u>, <u>During</u>}
S_NAME_DURING{<u>S#</u>, <u>During</u>, SName}
S_STATUS_DURING{<u>S#</u>, <u>During</u>, Status}
S_CITY_DURING{<u>S#</u>, <u>During</u>, City}

Thus, any change in an attribute does not affect the entire tuple from SSSC_During.

It is easy to see that 6NF ⇒ 5NF. However, it is interesting to note that DK/NF ⇒ 6NF does not hold.

Object-Normal Form
Fourth Object-Normal Form
Fifth Normal Form
Fifth Object-Normal Form
Sixth Normal Form
Sixth Object-Normal Form

“On Normalization of Relations in Relational Databases”
Programming and Computer Software 28(1), 2002
V.V. KHODOROVSKII

Object-Normal Form (ONF)

R is in ONF if:
- It is BCNF
- It does not contain subobjects

A subobject is a set of attributes X in a BCNF relation R where:
- $X \subset R$
- X contains more than one attribute
- X is not a superkey of R
- Attributes from X are semantically interrelated and isolated from the attributes of (R – X)
- Attributes from X represent semantic unity of the corresponding part of the application domain

This is not the same ONF as [Bis91].

Fourth Object-Normal Form (4ONF)

R is in 4ONF if:
- It is in 4NF and in ONF

Fifth Normal Form (5NF)

R is in 5NF if:
- For any irredundant JD $*[R_1, R_2, \ldots, R_p]$ in R every R_i is a superkey of R

A JD $*[R_1, R_2, \ldots, R_p]$ of R is irredundant if no proper subset of that JD in turn defines a JD in R. This is “essentially equivalent”[Kho02] to the minimal JD from [Fag79].

[Kho02] claims this is “the correct definition … given for the first time”.

Fifth Object-Normal Form (5ONF)

R is in 5ONF if:
- It is in 5NF and in ONF

Sixth Normal Form (6NF)

R is in 6NF if:
- It is in 5NF
- It does not contain dependencies with weak excess.

The trivial JD J is said to be a dependency with weak excess if:

$\alpha = \frac{n^* - n}{n} \ll 1$, where α is the excess coefficient. n^* is the cardinality of R^* and n is the cardinality of R_0.
R^* is the join of R_1 and R_2, where R_1, R_2, and R_3 are the projections of R_0 into 5NF, where R_0 is in ONF with three subobjects.

This is not the same 6NF as [Dat02].

Sixth Object-Normal Form (6ONF)

R is in 6ONF if:
- It is in 6NF and in ONF

“Restriction-union” Normal Form

An Introduction to Database Systems (8th Edition)
Addison-Wesley, 2004
C.J. DATE

This is not a real normal form, but more of a provoking question posed in [Dat04]. Since the “classic” normal forms all deal with the projection and join operations, could a normal form (or even a series of forms) be worked out around decomposing by restriction, rather than projection?

Date immediately notes that the results would almost certainly be a poor design, but is making the larger point that “classic normalization theory has absolutely nothing to say in answer to such questions.”

(3, 3)NF from [Smi78] and PSJU/NF from [Fag79] are given as possible starting points or sources for ideas.

We can also add that there are several normal forms covered in this appendix that consider operations other than projection.

APPENDIX G

REFERENCES

Arm03 Armstrong, William W., Nakamura Y., and Rudnicki P.
"Armstrong's Axioms"
Journal of Formalized Mathematics, 14, 2003.

Bee77 Beeri, C., Fagin, R. and J. Howard
"A Complete Axiomatization for Functional and Multivalued Dependencies in Database Relations"
Proceedings of the 1977 ACM SIGMOD Conference Ontario, Canada, 1977

Bee79 Beeri, C. and Bernstein P.
"Computational Problems Related to the Design of Normal Form Relational Schemas"
ACM Transactions on Database Systems 4(1), March 1979

Ber76 Berstein, P.A.
"Synthesizing Third Normal Form Relations from functional dependencies"
ACM Transactions on Database Systems 1(4), December 1976

Bis91 Biskup, Joachim and Pratul Dublish
"Objects in Relational Database Schemes with Functional, Inclusion, and Exclusion Dependencies"
3rd Symp. of Math. Fund. of Database and Knowledge Systems, Rostock, 1991

Cas83 Casanova, Marco A. and V.M.P. Vidal
"Towards A Sound View Integration Methodology"
Proc. 2nd ACM SIGACT-SIGMOD Sym. of Prin. of Database Systems, Atlanta, 1983

Cas84 Casanova, Marco A. and Jose E. Amaral de Sa
"Mapping Uninterrupted Schemes into Entity-Relationship Diagrams: Two Applications to Conceptual Schema Design"
IBM Journal Res. Develop. 28(1), January 1984.

Cas84-2 Casanova, Marco A., Ronald Fagin, and Christos H. Papadimitriou
"Inclusion Dependencies and Their Interaction with Functional Dependencies"
Journal of Computer and System Sciences 28(1), February 1984

Cod70 Codd, E.F.
"A Relational Model of Data for Large Shared Data Banks."
Communications of the ACM 13(6), June 1970

Cod71 Codd, E.F.
"Further Normalization of the Data Base Relational Model"
IBM Research Report RJ909, August 1971

Cod71-2 Codd, E.F.
"Normalized Data Base Structure: A Brief Tutorial"
Proc. ACM SIGFIDET Workshop, San Diego, 1971

Cod74 Codd, E.F.
"Recent Investigations in Relational Data Base Systems"
IFIP Congress, 1974

Dat92 Date, C.J. & Fagin, R.
"Simple Conditions for Guaranteeing Higher Normal Forms in Relational Databases"
ACM Transactions on Database Systems, 17(3), September 1992

Dat99 Date, C.J.
"Thirty Years of Relational: The First Three Normal Forms, Part 2"
Intelligent Enterprise 2(6), April 1999

Dat02 Date, C.J., Hugh Darwen, and Nikos A. Lorentzos
Temporal Data and the Relational Model
Morgan Kaufmann, 2002.

Dat03 Date, C.J.
"What First Normal Form Really Means"
Available from: http://www.dbdebunk.com

Dat04 Date, C.J.
An Introduction to Database Systems
8th Edition. Addison-Wesley, 2004

Fag77 Fagin, R.
"Multivalued Dependencies and a New Normal Form for Relational Databases"
ACM Transactions on Database Systems 2(3), September 1977

Fag79 Fagin, R.
"Normal Forms and Relational Database Operators"
ACM SIGMOD Conference, 1979

Fag81 Fagin, R.
"A Normal Form for Relational Databases That Is Based on Domains and Keys"
ACM Transactions on Database Systems 6(3), September 1981

Hea71 Heath, I.J.
"Unacceptable File Operations in a Relational Data Base"
Proc. ACM SIGFIDET Workshop, San Diego, 1971

Ken83 Kent, W.
"A Simple Guide to Five Normal Forms in Relational Database Theory"
Communications of the ACM 26(2), February 1983

Kho02 Khodorovskii, V.V.
"On Normalization of Relations in Relational Databases
Programming and Computer Software 28(1), 2002

Lev00 Levene, Mark and Millist W. Vincent.
"Justification for Inclusion Dependency Normal Form"
IEEE Transactions on Knowledge and Engineering 12(2), Mar/Apr 2000

Lin81 Ling, Tok-Wang, Frank W. Tompa and Tiko Kameda.
"An Improved Third Normal Form for Relational Databases"
ACM Transactions on Database Systems 6(2), June 1981

Lin92 Ling, Tok-Wang and Cheng Hian Goh
"Logical Database Design with Inclusion Dependencies"
Proc. Int'l Conf. Data Engineering, Tempe,1992

Mai83 Maier, D.
The Theory of Relational Databases
Computer Science Press, 1983

Man86 Mannila, Heikki and Kari-Jouko Räihä
"Inclusion Dependencies in Database Design"
Proc. Int'l Conf. Data Engineering, Los Angeles, 1986

Mit83 Mitchell, John C.
"Inference Rules for Functional and Inclusion Dependencies"
Proc. ACM PODS, 1983

MW04 "normalization"
Merriam-Webster Online Dictionary. 2004.
http://www.merriam-webster.com (1 March 2004)

Nor98 Normann, Ragnar.
"Minimal Lossless Decompositions and Some Normal Forms between 4NF and PJ/NF"
Information Systems 23(7), 1998

Orl92 Orlowska, Maria E. and Yanchun Zhang
"Understanding the Fifth Normal Form (5NF)"
Australian Computer Science Communication 14(1), 1992

Par80 Parker, D. Stott, Jr. and Kamran Parsaye-Ghomi
"Inferences Involving Embedded Multivalued Dependencies and Transitive Dependencies"
Proc. ACM SIGMOD Int'l Conf. on Management of Data, Santa Monica, 1980

Pas04 Pascal, Fabian
"What First Normal Means Not"
Available from: http://www.dbdebunk.com

Pas04-2 Pascal, Fabian
"The Costly Illusion: Normalization, Integrity and Performance"
Available from: http://www.dbdebunk.com

Pra91 Pratt, Philip & Joseph Adamski.
Database Systems Management and Design. p. 258
Second Edition. Boyd & Frasier. 1991.

Sci82 Sciore, Edward.
“A Complete Axiomatization of Full Join Dependencies”
Journal of the ACM 29(2), April 1982

Sil01 Silberschatz A., Korth, H.F, and S. Sundershan
Database System Concepts.
4th Edition. Appendix C. 2001

Smi78 Smith, J.M.
“A Normal Form for Abstract Syntax”
Proc. 4th Conf. Very Large Data Bases, Berlin, 1978.

Vin97 Vincent, M.W.
“A Corrected 5NF Definition for Relational Database Design”
Theoretical Computer Science 185, 1997.

Vin98 Vincent, M.W.
“Redundancy Elimination and a New Normal Form for Relational Database Design”
Semantics in Databases, Springer-Verlag LNCS 1358, 1998.

Vin98-2 Vincent, M.W.
“What is the real 5NF?”
http://www.cis.unisa.edu.au/~cismwv/papers (Awaiting publication, 1998)

Wer93 Wertz, Charles.
Relational Database Design.
CRC Press, Inc. 1993. p. 72

Wu92 Wu, M.S.
“The Practical Need for Fourth Normal Form”
ACM SIGCSE Bulletin 24(1), March 1992

Zan82 Zaniolo, Carlo
“A New Normal Form for the Design of Relational Database Schemata”
ACM Transactions on Database Systems 7(3), September 1982