BINF 8211/6211
Design and Implementation of Bioinformatics Databases
Lecture #13
Normalization

Dr. D. Andrew Carr
Dept. Bioinformatics and Genomics UNCC
Spring 2016

Bio-Python

- Use already developed tools for data manipulation
- Can input from python straight into the database
  - http://biopython.org/wiki/Converting_sequence_files
- Already community error checked.
- Other discussion:
- https://www.biostars.org/p/10353/

- **from** Bio **import** SeqIO
- input_handle = open("cor6_6.gb", "rU")
- output_handle = open("cor6_6.fasta", "w“)
- sequences = SeqIO.parse(input_handle, "genbank")
- count = SeqIO.write(sequences, output_handle, "fasta“)
- output_handle.close()
- input_handle.close()
- **print** "Coverted %i records" % count

Example Python Code Convert .sff to .fastq

The following converts a file and then loads the sequences into python using biopython.

```
## tools to import

from Bio import SeqIO
### ----------------------------------------------------------------

###Enter the directory
dir1 = " Your directory here"

### to convert 454 data to fastq
count = SeqIO.convert(dir1+"F3P5W4J04_CR_Pcoffeae.sff", "sff", dir1+"F3P5W4J04_CR_Pcoffeae.fastq", "fastq")
##
###F3P5W4J04_CR_Pcoffeae.fastq
fl1_fastq = "F3P5W4J04_CR_Pcoffeae.fastq"
fl1 = dir1+fl1_fastq
##To manipulate sequences in BioPython
tmpfl = open(fl1, "rU")
sequences = list(SeqIO.parse(tmpfl,"fastq"))
tmpfl.close()

print len(sequences), "loaded into python."
```

Example Bio Python Code Load Fastq into list

The following converts a file and then loads the sequences into python using biopython.

```
## tools to import

from Bio import SeqIO
### ----------------------------------------------------------------

###Enter the directory
dir1 = " Your directory here"

### to convert 454 data to fastq
count = SeqIO.convert(dir1+"F3P5W4J04_CR_Pcoffeae.sff", "sff", dir1+"F3P5W4J04_CR_Pcoffeae.fastq", "fastq")
##
###F3P5W4J04_CR_Pcoffeae.fastq
fl1_fastq = "F3P5W4J04_CR_Pcoffeae.fastq"
fl1 = dir1+fl1_fastq
##To manipulate sequences in BioPython
tmpfl = open(fl1, "rU")
sequences = list(SeqIO.parse(tmpfl,"fastq"))
tmpfl.close()

print len(sequences), "loaded into python."
```

Homework 4

- Load your data into a database:
  - NON-TRIVIAL
  - Create tables to load the data that you collected
    - Nucleotide sequence data
    - Protein data
      - Sequence data (translation of the DNA data)
      - Structure data
    - Pathway data
    - Table to hold data lineage

  - You will decide the attributes needed and their data types.
    - Use the file definitions as guidance
      - FASTQ, PDB ….
  - Link these tables so that they can be queried
    - Can I pull the structure information for a given DNA sequence.

  - 1 Meta data table regarding a topic from the following
  - Due March 1st at 8:00

- Read Chapter 5 in Rob and Coronel ed. 7, or any chapter in a standard text book with a title approximating ‘Normalization of Database Tables’.

Normalizing Table Structures

Web sites explaining Normalization

- http://databases.about.com/od/specificproducts/a/normalization.htm
- http://www.agiledata.org/essays/dataNormalization.html Walks through an extended example
- http://www.dbnormalization.com/ Has several examples
- And of course http://en.wikipedia.org/wiki/Database_normalization

- Here is a wiki book on Data management in Bioinformatics that discusses some of what the lectures have covered:
- http://en.wikibooks.org/wiki/Data_Management_in_Bioinformatics
- There is a discussion as to how database design decisions like referential integrity enforcement affect applications here (that might motivate you to take your precautions early and often):
- http://stackoverflow.com/questions/621884/database-development-mistakes-made-by-application-developers

Normalization is the process of evaluating and correcting table structures to *minimize* data redundancies.

Reduce the chance of inconsistencies.

		vvvvv
		vvvvv
	vvvvv	

Attributes are assigned to tables based on determination.

Each succeeding Normal Form (level) is structurally better, BUT, the higher the Form, the more joins tend to be required when formulating queries.

			vvvv	

	vvvvv	

A coordinated strategy to improve a database is to make it conform to the Normal Forms.

In each step we separate a table into subgroups focused on single subjects, with minimal redundancy (FK) and where all attributes depend only on the PK.

There are 5 rules that one uses as tests: 1NF, 2NF, 3NF, BCNF and 4NF. Normalization and ER modeling are used concurrently/ iteratively to produce a good design.

Data Analysis:

To generate output files for analysis programs, summaries and graphs you will likely create a View, which is not normalized, but your base data tables that are loaded should be well normalized.

Determination is the concept that by knowing the value of one attribute you know the value of every other attribute for that instance of that entity.

- The primary key is defined in order to ensure this identification can occur.
- Determination:
  - Attribute B is functionally dependent on attribute A *when each value* in attribute **A** *determines one and only one* value of attribute **B**.
  - Attribute A determines attribute B if all of the rows in the table that agree in value for attribute A also agree in value for attribute B.

The levels of the Normal Forms are as follows

Normal Form		Characteristic
First Normal Form	1NF	Table format; no repeating groups, PK identified
Second Normal Form	2NF	1 NF AND no partial dependencies
Third Normal Form	3NF	2NF AND no transitive dependencies
Boyce-Codd Normal Form	BCNF	Every determinant is a candidate key
Fourth Normal Form	4NF	3NF AND no independent multi-valued dependencies

The Fifth Normal Form and the domain-key normal form have also been defined, but are not generally enforced.

Note: most of the tables and figures are closely adapted from the Rob and Coronel textbook (7th edition). Errors are my own. See Chapter 5.

Data in 1NF means?

All of the attributes are defined, those whose values reflect the same characteristic been given identical names (all copies of redundant data must be identical).

All of the **key** attributes are defined and have no nulls .

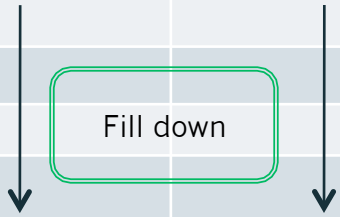
There are no multi-valued attributes .

The data dependencies of attributes have been identified.

Note: if you have followed the basic rules then all relational tables are in 1NF and thus the model/schema is *functional*; this does not imply an *effective* design (for example, there can be more data duplication than required by FK needs).

Regularize/standardize attribute values

Complex_id	Name	Component_id	Component _name	Function	Rate (uM/sec)	Half-life (msec)
U30	Spliceosome	103	U3	mRNA alignment	300	1000
		101	U5	Splice exons	15	400
		105	U7	Splice introns	20	400
		106	U12	RNA degradation	500	100
		102	U4	Junction recognition	35	400
30S	Ribosome	114	L12	50S interface	5	700
		118	16S	mRNA alignment	200	1000
		104	L1	P site	12	1200
		112	L15	E site	24	1200
18L	Lysosome	105	U7	Splice introns	20	400
		104	L1	P site	12	1200
		113	EBB2	Membrane fusion	44	260
		111	ERBP	ER budding	180	320
		106	U12	Degrade RNA	500	100
12P	Peroxisome	107	PFG	Protein folding scaffold	75	500
		115	DSF	Disulfide bond formation	322	90
		101	U5	Exon splicing	15	400
		114	L12	50S junction	5	700
		108	MPF	Membrane pore	68	90
		118	16S	mRNA alignment	200	1000
		112	L15	E site	24	1200



Data dependencies identified → Primary key is initially considered.

- Test: by knowing this value can I correctly predict the field value for every other attribute in that row?

Conversion to the First Normal Form (1NF) involves getting rid of nulls, then identifying repeating groups, standardizing terms, determining the PKs.

Complex_id	Name	Component_id	Component _name	Function	Rate (uM/sec)	Half-life (msec)
U30	Spliceosome	103	U3	mRNA alignment	300	1000
U30	Spliceosome	101	U5	Splice exons	15	400
U30	Spliceosome	105	U7	Splice introns	20	400
U30	Spliceosome	106	U12	RNA degradation	500	100
U30	Spliceosome	102	U4	Junction recognition	35	400
30S	RibosomalSubunit	114	S12	50S interface	5	700
30S	RibosomalSubunit	118	16S	mRNA alignment	200	1000
30S	RibosomalSubunit	104	S1	P site	12	1200
30S	RibosomalSubunit	112	S15	E site	24	1200
18L	Lysosome	105	U7	Splice introns	20	400
18L	Lysosome	104	L1	P site	12	1200
18L	Lysosome	113	EBB2	Membrane fusion	44	260
18L	Lysosome	111	ERBP	ER budding	180	320
18L	Lysosome	106	U12	RNA degradation	500	100
12P	Peroxisome	107	PFG	Protein folding scaffold	75	500
12P	Peroxisome	115	DSF	Disulfide bond formation	322	90
12P	Peroxisome	101	U5	Splice exons	15	400
12P	Peroxisome	114	L12	50S interface	5	700
12P	Peroxisome	108	MPF	Membrane pore	68	90
12P	Peroxisome	118	16S	mRNA alignment	200	1000
12P	Peroxisome	112	L15	E site	24	1200

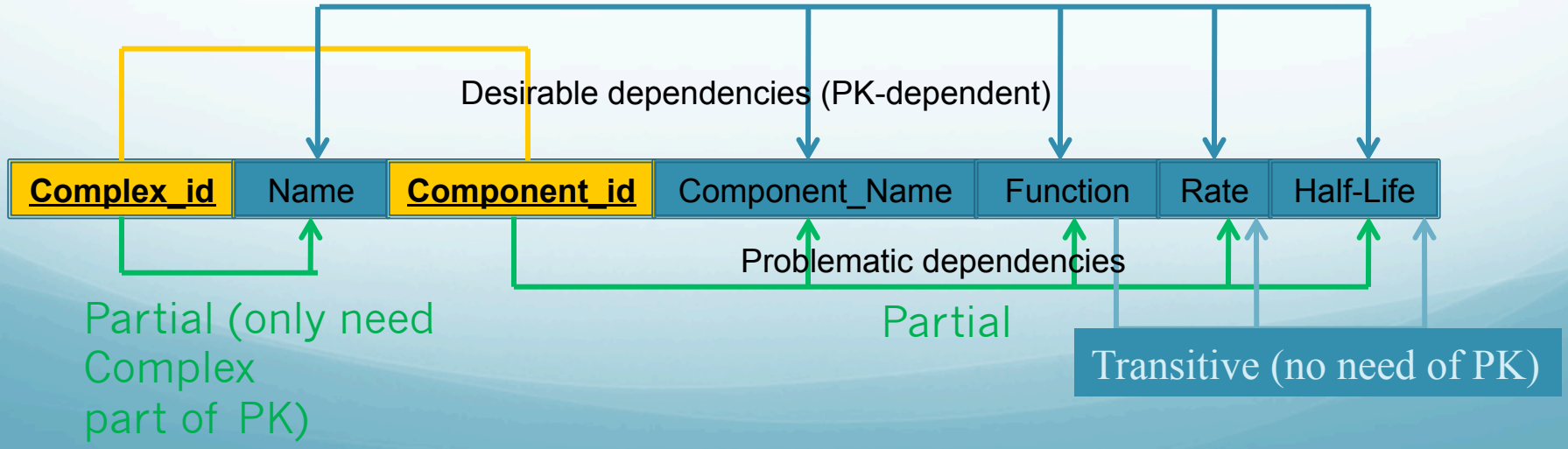
Complex_id	Name	Component_id	Component _name	Function	Rate (uM/sec)	Half-life (msec)
U30	Spliceosome	103	U3	mRNA alignment	300	1000
U30	Spliceosome	101	U5	Splice exons	15	400
U30	Spliceosome	105	U7	Splice introns	20	400
U30	Spliceosome	106	U12	RNA degradation	500	100
U30	Spliceosome	102	U4	Junction recognition	35	400

The Complex_id → Name
(Read: when I know the complex_id I know the complex name)

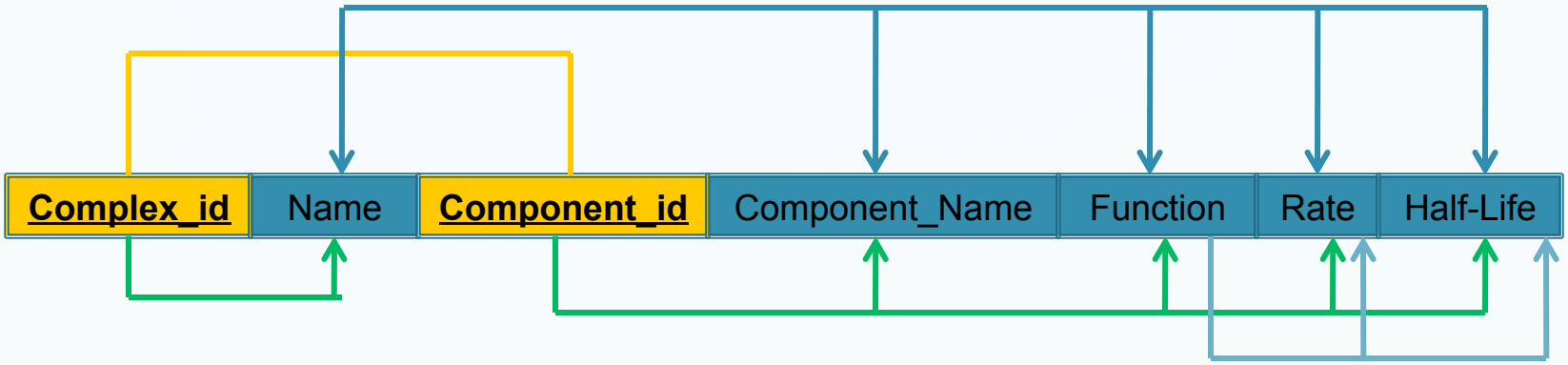
The Component_id → Component_Name, Function, Rate and Half-life
= key (underline and bold, different color)

Thus, Complex_id + Component_id → PK

But Function alone → Rate and Half-Life



Converting the diagram to text definitions:



1NF (Complex_id, Component_id, Name, Component_Name, Function, Rate, Half-life**)**

Partial dependencies:

Complex_id → Name

Component_id → Component_Name, Function, Rate, Half-life

Transitive Dependencies:

Function → Rate, Half-life

In moving up to 2NF you eliminate the partial and transitive dependencies you've identified, adding new tables needed to show the relationship.

Partial dependencies are?

Transitive dependencies are?

Pick one:

1. A non-key attribute value can be determined from other non-key attributes.
2. Not completely defined by anything in the table
3. One non-primary attribute on another non-primary attribute
4. Only require part of a PK to be completely defined

The strategy to convert to 2NF is to separate the dependencies into new entities.

The strategy is to separate PK components:
Complex_id
Component_id
To relate Complex_id x Component_id (Link) and in this case rename as: Subunit

These become revised/new tables:

COMPLEX	
PK	**Complex_id**
	Name

SUBUNIT	
PK, FK1	**Complex_id**
PK, FK2	**Component_id**
	Stoichiometry

COMPONENT	
PK	**Component_id**
	Component_Name
	Function
	Rate
	Half-life

Repeat the diagramming exercise for each new entity to make sure the *partial* dependencies have been eliminated.

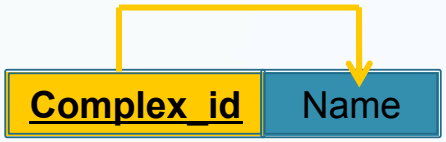


Table Name: COMPLEX

COMPLEX(**Complex_id**, Name)

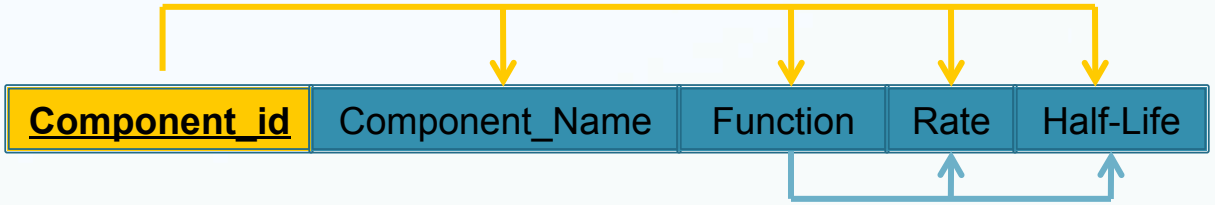


Table name: COMPONENT

COMPONENT(**Component_id**, Component_Name, Function, Rate, Half-life)

We still have the Transitive Dependency (Function → Rate, Half-life)

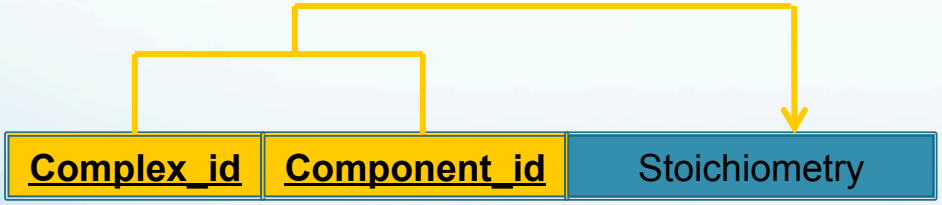


Table Name: SUBUNIT

SUBUNIT(**Complex_id**, **Component_id**, Stoichiometry)

A table is in second normal form (2NF) when it is in 1NF AND it has no *partial* dependencies.

A *partial dependency* only occurs when a PK is composite (includes more than one attribute when it is not a linking table); any table with a single attribute as a PK is in 2NF.

Note that transitive dependencies are still possible in this form.

Conversion to the 3NF means eliminating any *transitive* dependencies.

A determinant is an attribute whose value determines the values of other attributes in the row.

The first task is to identify the determinant that is going to be the PK for a new table

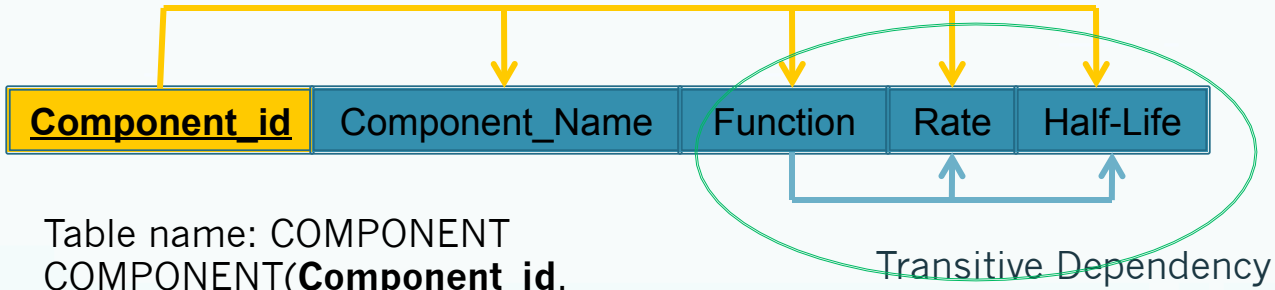


Table name: COMPONENT
COMPONENT(**Component_id**, Component_Name, Function, Rate, Half-life)

Transitive Dependency
(Function → Rate, Half-life)

Table name: COMPONENT
COMPONENT(**Component_id**, Component_Name)

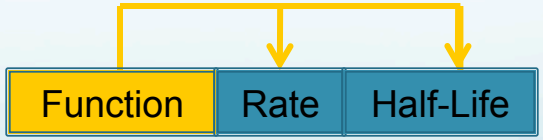


Table name: FUNCTION
FUNCTION(**EC_id**, Rate, Half-Life)

A table in in 3NF when it is in 2NF AND there are no transitive dependencies.

First thing you notice: all of the tables are smaller.
This does not ensure a *good* design, only that there is minimal redundancy.

Second look: are the attributes simple? Are the names still clear?

Table name: COMPONENT
COMPONENT(**Component_id**, Component_Name)

Table name: FUNCTION
FUNCTION(**EC_id**, Rate, Half-Life)

Table Name: SUBUNIT
SUBUNIT(**Complex_id**, **Component_id**, Stoichiometry)

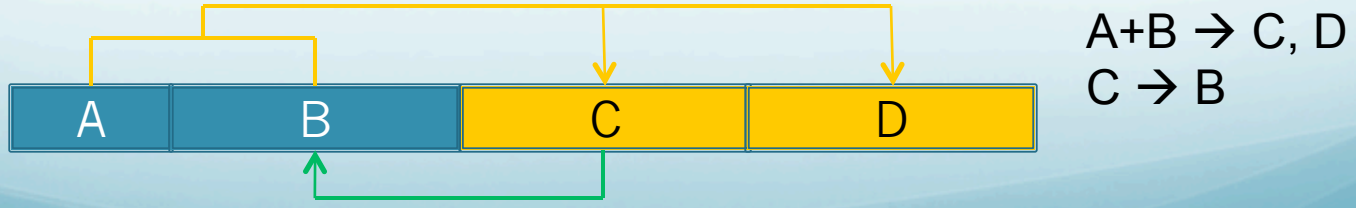
Table Name: COMPLEX
COMPLEX(**Complex_id**, Name)

A higher level of control occurs when tables are put in the Boyce-Codd Normal form (BCNF).

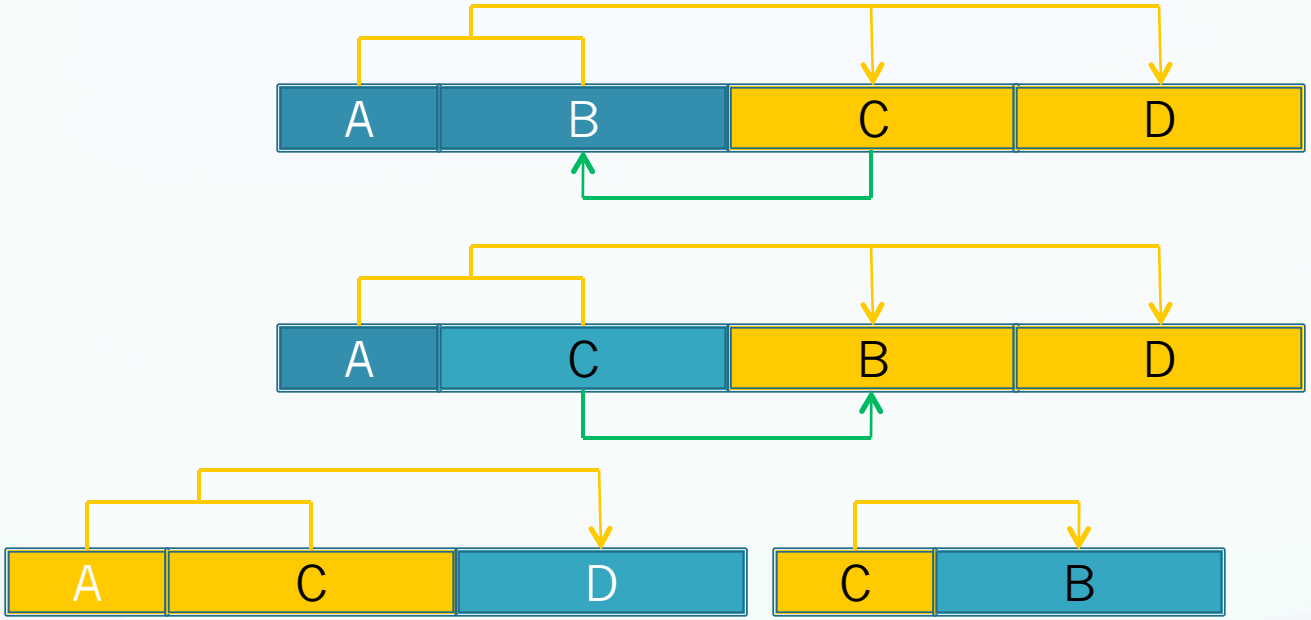
In BCNF every determinant in a table is a candidate key.

Reminder: candidate keys are appropriate as choices for PKs.

Sometimes a non-key attribute is a determinant of a key attribute (transitive means a non-key attribute determines another *non-key* attribute) – this does not violate the 3NF rule which only considers the key attributes.



If you change the key and then decompose the partial dependency the resulting tables will be in 3NF.



In the final grouping the tables are 3NF and also BCNF.
When a table contains only one candidate key, 3NF = BCNF

Patient_No	Name	Appt_No	Appt_Time	MD
10021	Anson	1	0900	Kinson
10032	Peters	1	0900	Buckle
14921	McBride	2	1000	Kinson
17332	Braxton	1	1300	Buckle
20003	Walsh	2	1400	Kinson

Functional analysis
TABLE(Patient_No, Name, Appt_No, Appt_Time, MD)
Determinants:

Patient_No → Name
Patient_No, Appt_No → Appt_Time, MD
Appt_Time → Appt_No

For keys we could use

TABLE(**Patient_No**, Name, **Appt_No**, Appt_Time, MD)
TABLE(**Patient_No**, Name, Appt_No, **Appt_Time**, MD)

Patient_No	Name	Appt_No	Appt_Time	MD
10021	Anson	1	0900	Kinson
10032	Peters	1	0900	Buckle
14921	McBride	2	1000	Kinson
17332	Braxton	1	1300	Buckle
20003	Walsh	2	1400	Kinson

Functional analysis

TABLE(**Patient_No**, Name, **Appt_No**, Appt_Time, MD)

From Patient_ No → Name (dependency of Name on just part of the key)

2NF

TABLE1 (**Patient_No**, **Appt_No**, Time, MD)

TABLE2 (**Patient_No**, Name)

There aren't any transitive dependencies (one non-key determines another)

Patient_No	Name	Appt_No	Appt_Time	MD
10021	Anson	1	0900	Kinson
10032	Peters	1	0900	Buckle
14921	McBride	2	1000	Kinson
17332	Braxton	1	1300	Buckle
20003	Walsh	2	1400	Kinson

Boyce-Codd NF – is every determinant a candidate key? Or, is there a non-key determinant of a key?

2NF

TABLE1 (**Patient_No**, **Appt_No**, Time, MD)

TABLE2 (**Patient_No**, Name)

In TABLE 1, neither Time nor MD is a determinant of the Appt_No or the MD, so not in BCNF.

In TABLE2, the value of Name is not a determinant of Patient_No (you could have multiple individuals named, Peters, for example) so Patient_No is a candidate key, and it is the only determinant so every determinant is a candidate key and this is in BCNF.

Patient_No	Name	Appt_No	Appt_Time	MD
10021	Anson	1	0900	Kinson
10032	Peters	1	0900	Buckle
14921	McBride	2	1000	Kinson
17332	Braxton	1	1300	Buckle
20003	Walsh	2	1400	Kinson

Functional dependence

TABLE1 (**Patient_No**, **Appt_No**, Time, MD)

In TABLE 1, Patient_No, Appt_No → Time, MD

Time → Appt_No

but Time does not determine the other attributes so this is not in BCNF.

This is where we swap around attributes, make Time the candidate key instead of Appt_No

TABLE1 (**Patient_No**, **Time**, MD)
TABLE2 (**Patient_No**, Name)
TABLE3 (**Time**, ApptNo)

Patient_No	Name	Appt_No	Appt_Time	MD
10021	Anson	1	0900	Kinson
10032	Peters	1	0900	Buckle
14921	McBride	2	1000	Kinson
17332	Braxton	1	1300	Buckle
20003	Walsh	2	1400	Kinson

Functional analysis second option

TABLE(**Patient_No**, Name, Appt_No, **Appt_Time**, MD)

Patient_No, Appt_No → Time, MD

Time → Appt_No

Decomposing for 4NF arises when a candidate key must be *the entire row* of attribute values in order to be unique, or when composite keys make it difficult to determine whether a set of values will be unique.

Consider in our working example the attributes rate and half-life. It is possible for distinct enzymes to have the same rate values and one enzyme can have different rates and half-life depending on its environment. The purpose to having the values is to give the rate and half life in the context of the complex.

A table is in fourth normal form when it is in 3NF *and* has no more than two attributes being combined as the PK value.

To avoid the problem:
Attributes are dependent on the PK and mutually independent.
No row uses a composite key as the PK.

In practice, many databases are only taken to 2NF (this is because larger tables minimize the need for joins).

Disadvantages:
Data updates are not efficient.

Indexing becomes a problem – it can be quite impractical to build indices for all of the attribute combinations you want.

There are no simple strategies for creating views (virtual tables).

Application programs cannot magically make a bad design transparent to redundancy anomalies and similar problems.

Some ‘rules’ for best practices in data modeling are garnered from experience rather than theory .

Vocabulary: a *natural key* is a **primary key** that uses the natural name of an attribute as the label.

Specifically: single words are useful, but long text or text-numeral combinations, especially if a composite is needed, tend to degrade over time.

For example: a probe on a microarray has a sequence, location, manufacturer and version, none ‘intuitively obvious’. So an assigned number is equally convenient.

What is a key for? Identity or description (semantics)?

Entity integrity → identity

Data granularity is one way of addressing consistency of representation

Measurement data is a large component of scientific databases – the granularity should include units, and an indication of precision (significant digits).

Rate is uM/sec but half-life is minutes? You can convert, but this takes an additional step, better to store in seconds.
Consistency allows the correct application of business rules.

When a controlled vocabulary is available this is often presented as a drop-down list for selection, to avoid transcription errors.

Function *descriptions* rather than EC numbers might be provided, as surrogate keys.

You might add attributes that allow temporal information to be tracked.

Temporal or time-variant data is common in scientific databases: such attributes are multi-valued.

For example, if every measurement has a date-stamp then you have a composite attribute.

If sequence data always has a QC associated with it, this is a composite attribute. This is handled by creating a new entity.

You might have one date-stamp for a large set of measurements, then the relationship will be 1:M. You might consider storing only differences.

This situation can lead to design traps, such as the 'fan' trap: the creation of two 1:M relationships with new entities that inadvertently creates/ misses a relationship with existing entities.

TEAM	
PK	Team_ID
FK1	Division_ID
	Team_Name

DIVISION	
PK	Div_ID
FK1	Division_Name

PLAYER	
PK	Player_ID
FK1	Division_ID
	Player_Name

The ‘fan’ trap’ above, you can’t figure out what player belongs to what team, while below you can.

DIVISION	
PK	Div_ID
FK1	Division_Name

TEAM	
PK	Team_ID
FK1	Division_ID
	Team_Name

PLAYER	
PK	Player_ID
FK1	Team_ID
	Player_Name

Some design considerations have to do with the *speed* of queries.

The DBMS automatically generates unique numbers if the PK is simply made an incremented value (e.g. read the SQLite Documentation). Then uniqueness does not have to be checked with each addition.

Similarly a date for the record can be automatically generated by the system, and stored to be similarly used.

A composite PK makes FKs more cumbersome as well as other search routines, and if text labels are used may become a rather long string to search.

Attributes of a good PK.

Characteristic	Explanation
Value is unique, not null	This is the heart of a PK requirement
Non-intelligible	An embedded semantic meaning turns out to be prone to mis-typing and mis-interpretation – an alias for the descriptive term is a better choice
Permanent	A side-effect of using a semantically meaningful PK is a change in interpretation over time (updates). Then changes will cascade to PKs and lots of unnecessary fun will ensue.
Single-attribute type	A single-attribute PK is preferred although not always possible. They make FKs easier to implement, and keep linking table PKs shorter.
Numeric	Ensuring uniqueness is readily done with a counter that increments, rather than using a name that must be checked.
Confidentiality	Even with numbers, make sure that the choice does not have privacy implications (like a SSN identifier)

Business Rules
Verify rules with users and then DOCUMENT (hearsay is not admissible….)
Rules identify entities, attributes, relationships and constraints as well as intended use in derived fields and views. Make them clear, precise and simple.
Give the reason for each rule (source, demand, date), the verification test and approval code.

ENTITIES
Represent a single object
Should be in 3NF (or higher)
Should have a well-defined granularity
Have a clearly defined PK that supports the granularity needed

Attributes
Simple and single-valued (‘atomic’)
Defined default values, constraints, synonyms and aliases
Derived attributes are clearly identified and sources are included
Redundant only as needed for transaction accuracy, for history, for FKs

RELATIONSHIPS
Clearly identify all participants
Clearly define participation and cardinality rules

ER Diagram
Validate against expected processes (insert, update, delete)
Evaluate where, when and how when a history is needed
Redundant relationships are present only for clearly defined needs
Overall data redundancy is minimized

Entity Naming Conventions: limit length (DBMS sensitive)
Use short meaningful (within context) nouns
Include common abbreviations, synonyms and aliases as meta-data
Ensure model-internal uniqueness
For composite entities you may link abbreviated entity names

Attribute Naming conventions: limit length
Ensure within-entity uniqueness
Use the entity abbreviation of prefix, as appropriate (unambiguous tie)
Aim to be descriptive, within context
For PK attributes use underscore abbreviation notation (e.g.: _ID)
Be aware of and avoid DBMS-reserved words (DATE is usually one such)
Avoid special characters and spaces

Relationship Naming conventions: limit length
Use verbs that indicate the nature of the relationship (e.g. ‘codes for’)