

(Free) Sample 4

Taken from:

E-book 4: Normalization

Knowledge River Ltd



(Free) Sample 4

Welcome to **Knowledge River** – the on-line e-learning store for Computer Science practitioners, academics and students.

Thank you for taking the time to download and examine this free sample – based on one of our popular e-learning products. In this free sample we have tried to give you a flavour of the full document upon which it is based. If you like what you see and would like to purchase the full document then please follow the instructions on our website: www.knowledge-river.co.uk

Whilst every effort has been made to ensure the accuracy of this material and any included software code has been tested carefully, they are only intended for instructional and illustrative purposes and are not guaranteed for any particular application, purpose or function. Knowledge River Ltd does not offer any warranties or representations, nor do we accept any liabilities with respect to any code examples included here.

Please note that Knowledge River provides Computer Science educational material in electronic format – we do not offer an on-line IT consultancy or help-desk function and so we cannot get into detailed correspondence on specific technical issues. However, we would appreciate general feedback and comments on what you think of the material, its layout, its usefulness and how it could be improved etc.

Please email us at: **feedback@knowledge-river.co.uk**

Copyright © Knowledge River Ltd 2006-7

All rights reserved.

Please see the accompanying notes on the Knowledge River Ltd usage policy.

Knowledge River Ltd - Usage Policy (How to use our products)

We hope you enjoy using these e-learning products and get a lot of benefit by doing so. However, these electronic resources do represent a very considerable investment in intellectual effort, time and money on our behalf and naturally we want to protect our intellectual property. These products are meant for *your own personal educational use* and should not be modified, altered, edited, re-sold or transferred to a third party.

Please use and enjoy our work but please respect our efforts too.

The table below summarizes what you can and cannot do with our e-learning products:

Action	Allowed?	Notes
Reading	Yes	Obviously! On-screen or off-screen (see printing).
Printing / Making a Hardcopy	Yes (Limited)	Only single copies for <i>your own personal use</i> . Multiple copies and/or photocopies intended for distribution or sale to third parties is NOT allowed.
Editing / Updating / Extending	No	The e-learning products you purchase from Knowledge River Ltd must NOT be edited or modified in any way. This is how they were intended to be and this is how we want them to stay. Edits are our responsibility.
Copying (Electronic)	Yes (Limited)	You may copy the purchased e-learning product for <i>your own personal use</i> within your own computing equipment and storage devices (eg on a desktop and laptop with a backup on a portable storage device). However, you must NOT make multiple copies of these products with the intention of passing these copies onto third parties (so doing copies for your friends or putting copies on a server for sale or distribution is not allowed). In short, keep one or two copies for <i>your own personal convenience</i> but do not give copies to other people. This rule applies regardless of whether or not you charge money for it.
Transferring / Passing On	No	Each e-learning product purchased is intended for <i>one individual</i> – so please do not pass copies onto third parties – let them purchase their own. This rule applies even if there is no money involved or financial gain for yourself – please do not pass these products around – ever. Knowledge River products are non-transferable.
Re-selling	No	It goes without saying that this is never allowed – taking a copy of our work and then selling it on is simply theft and is immoral - please don't even think about doing this.
Anything Else	No	Anything not already covered is not allowed – please treat our company and products with respect. Enjoy.

The full *E-book 4* contents are:

1. When in the database development lifecycle do we normalize?
2. Do we normalize entities?
3. Do we normalize relations?
4. Do we normalize tables?
5. What is the aim of normalization?
6. What does normalization seek to remove?
7. Where does normalization come from?
8. Some new Relations?
9. What is FD?
10. What is First Normal Form (1NF)?
11. What do *Un-Normalised* (UNF) Relations Look Like?
12. What is Second Normal Form (2NF)?
13. What is Third Normal Form (3NF)?
14. What is Boyce-Codd Normal Form (BCNF)?
15. 'The Key, the Whole Key and Nothing but the Key'
16. Are there any more normal forms?
17. What is Multi-Valued Dependency?
18. What is Fourth Normal Form (4NF)?
19. What is Fifth Normal Form (5NF)?
20. What are *Inference Rules*?
21. What are Update Anomalies?
22. Is it possible to implement a poorly normalized relation?
23. Why would we *de-normalize*?
24. What do we do with a set of normalized relations?

Like this sample, the full document is in PDF format and can be purchased individually at a modest price or you may like to buy several different documents and get *very substantial discounts* – full details on our website. Thank you once again for taking the time to examine this **Knowledge River** educational material – please have a look at our website or drop us an email.

Below are three examples...

When in the database development lifecycle do we normalize?

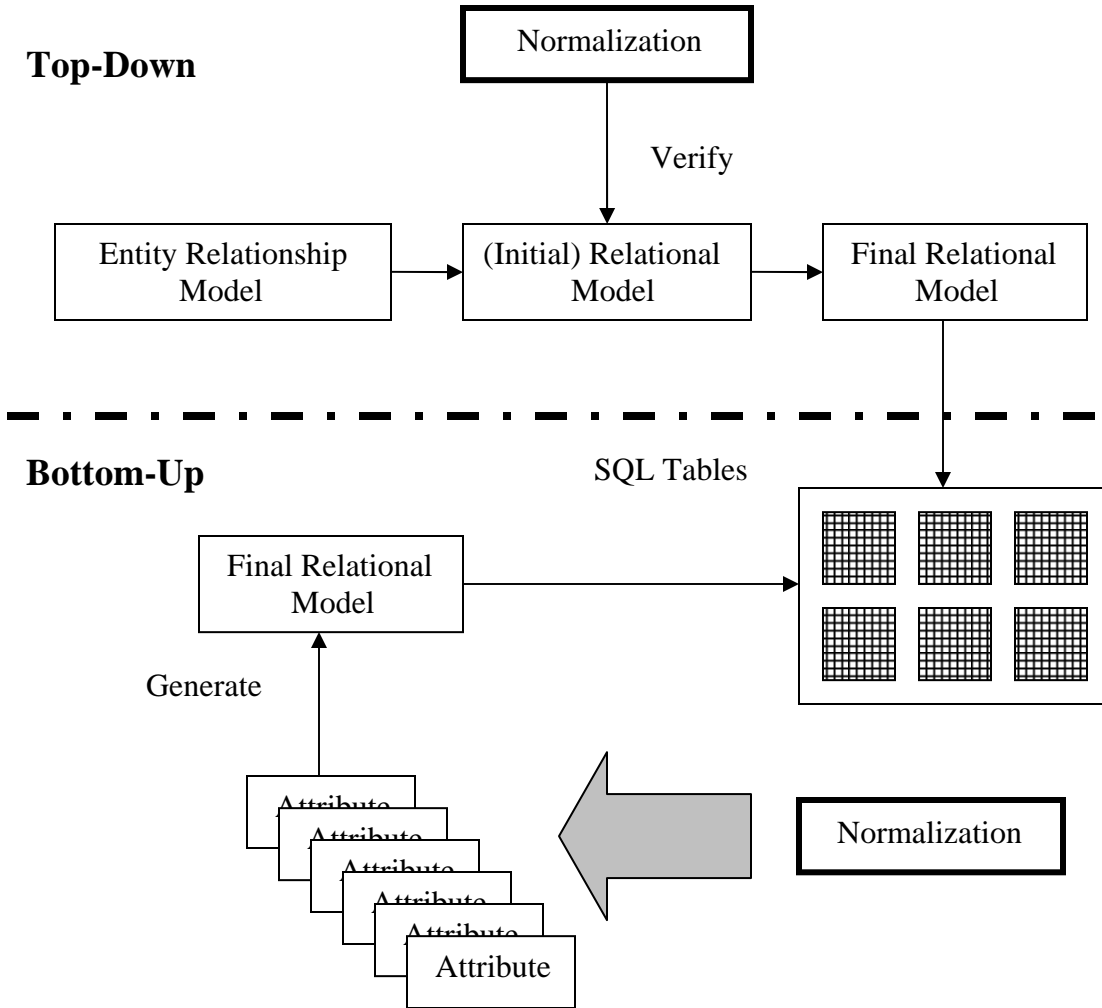
In this series of e-books we have been following the so-called ‘top-down’ methodology of database development: we began by looking at the universe of discourse as a whole and then constructed an entity-relationship model, from which we generated a relational model comprising a set of individual relations. We are now going to examine how to normalize each of these separate relations prior to implementation in SQL. Hence, in this approach normalization comes quite late in the database development lifecycle.

However, there is another methodology called the ‘bottom-up’ approach whereby we examine the data at the individual attribute level and try and isolate inter-attribute relationships using the normalization techniques. Here normalization comes at the beginning of the lifecycle.

Think of the ‘top-down’ model as using normalization as a verification or confirmation technique while in the ‘bottom-up’ model normalization is actually generating the relations (as opposed to checking them afterwards).

In either case, normalization seeks to have relations which are theoretically sound prior to implementation as SQL tables.

The diagram below sets out the big picture.

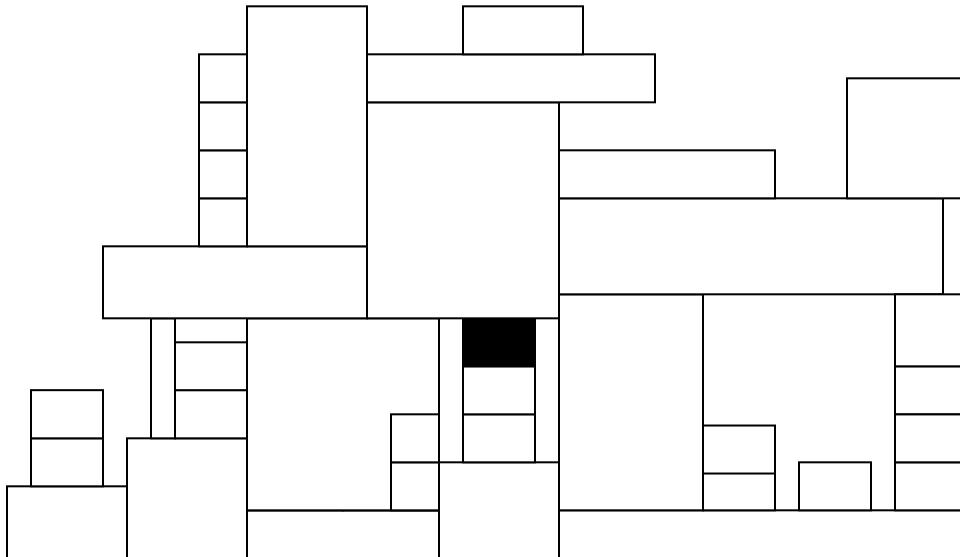


What is the aim of normalization?

Before looking at the technical reasons behind performing normalization it would be beneficial to study the concept of good and bad storage itself (because databases are essentially just storage sites for data). Consider the following well-behaved storage facility and its badly-behaved cousin.

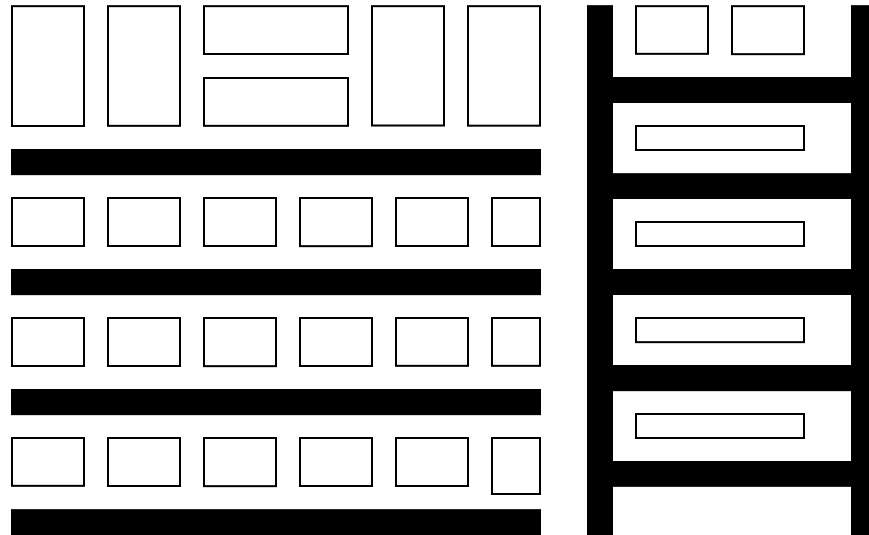
The Badly-Behaved Storage Facility

Here all the items to be stored (boxes and containers of various sizes) are simply stacked up, one on top of the other, with no thought given to organisation or how we can insert, move or extract any given box. For example, to remove the black box below would mean considerable disruption to other boxes and would necessitate a lot of unnecessary work.



The Well-Behaved Storage Facility

By contrast, here we have arranged our boxes and containers in a highly structured and carefully laid-out manner, utilizing shelves. To insert, move or extract any box now is very easy and disruption to other boxes is zero to minimum.



The lessons to be learned...

- Storage alone is not enough – we need well designed, easily maintained storage
- The insertion, movement or removal of one item should not cause disruption
- Deciding what and where an item goes should be done before that item is stored
- Storage facilities are volatile, not static and item movement will be a constant
- This movement or modification should be as quick and efficient as possible
- Neighbouring items should not get caught up in this movement (localize changes)

So how does this relate to databases and normalization?

Well, we can view the relations that make up the database as like storage facilities – but for data, not boxes or containers. Hence, we need to apply the same principles to data storage in relations as we do to physical storage – ease of insertion, extraction, modification and deletion with minimal disruption are the guiding principles. The process of normalization is the tool we use to achieve this optimized data storage.

We can now turn to the more technical issues of normalization – but always bear in mind the two above approaches to storage.

What is Second Normal Form (2NF)?

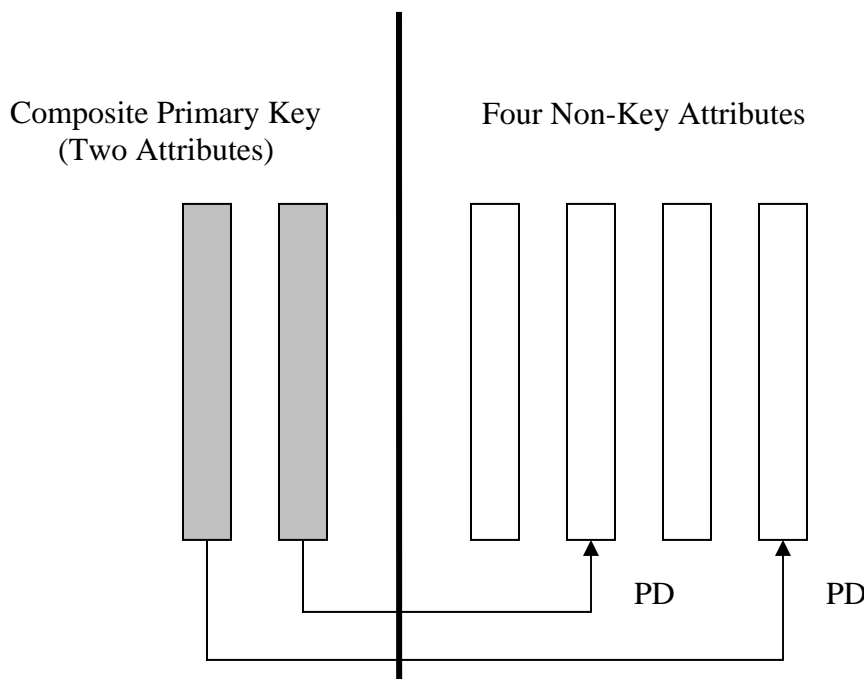
This stage of normalization is founded on two related concepts – full functional dependency (FFD) and partial dependency (PD).

For a relation to be in second normal form it must satisfy the following criteria:

- Be in first normal form (1NF)
- AND
- The primary key consists of a single attribute (it is an atomic primary key)
- OR
- There are no non-primary key attributes (all the attributes are inside the key)
- OR
- Every non-primary key attribute is fully functionally dependent on the *entire* key

So in other words, each non-primary key attribute must depend for its values upon the *whole* primary key (we cannot allow partial dependency where only a sub-set of the primary key determines the non-key attributes). Now, of course, if the primary key only has a single attribute (it is atomic) then the 2NF condition is automatically satisfied (because you cannot have partial dependency with one attribute). Likewise, if all the attributes in a relation are inside the primary key then again the relation is automatically in 2NF.

However, if you do have a composite primary key with two or more attributes and you also have attributes outside the primary key then you may or may not have a partial dependency problem – you need to check. This diagram sums up the situation:



(Free) Sample 4

In this example, non-key attributes 1 and 3 are fine – they are dependent upon the *whole* primary key (both attributes together) but non-key attributes 2 and 4 present a problem because they are dependent only upon *part* of the primary key (as shown with the lines). We say that non-key attributes 1 and 3 are fully functionally dependent (FFD) upon the (whole) primary key whilst non-key attributes 2 and 4 are partially dependent (PD) upon the primary key. Partial dependencies are bad and to be in 2NF a relation must have them removed.

We identify partial dependencies by adopting the following structured approach:

Stage 1: Identify the Primary Key

- Atomic primary keys = one attribute = *automatic* 2NF (move onto 3NF)
- Composite primary keys = two or more attributes = possible partial dependencies
- In this case move onto step 2

Stage 2: Identify all the non-primary key attributes

- If no attributes outside the primary key = *automatic* 2NF (move onto 3NF)
- However, if there are, document them
- In this case move onto step 3

Stage 3: Itemize all functional dependencies for the relation

- Check each NON-primary key attribute by asking yourself:

“Does this attribute depend on ALL the primary key or just PART of the primary key for its values”

- If the attribute depends on the *whole* primary key we have full functional dependency (FFD) – which is good. Move onto the next non-key attribute.
- If the attribute depends only on *part* of the primary key we have partial dependency (PD) – which is bad. This problem needs to be fixed. Move onto stage 4.

Stage 4: Decompose and link the offending relation into two child relations

- Just as in resolving 1NF problems, break the parent relation into two child relations, re-name them and ensure that they can be re-joined by having a common attribute(s) – the link – between them. You should now have two 2NF relations.

(Free) Sample 4

Here is a real example...

Order

Supplier ID	Date	Order#	Supplier Name	Item	Price (£)	Quantity
1	22-Sept	1001	Canine Care	Surgical Forceps	16	12
2	22-Sept	1002	Doggy Style	Rubber Gloves	0.10	1000
3	29-Sept	1003	K94U	Microchips	2.50	150
3	5-Oct	1004	K94U	Worming Tablets	0.50	1000
4	12-Oct	1005	Canine Medical Supplies	Booster Jabs	10	500
4	19-Oct	1006	Canine Medical Supplies	X-Ray Plates	18	50
5	21-Oct	1007	VetLine Vaccinations	Puppy Jabs	12	200
6	02-Nov	1008	Canine Blood Services	Epilepsy Test	9	25

Primary key = (Supplier ID, Date) = composite key – needs further investigation

We have five non-key attributes.

- FD1: Supplier ID, Date ⇒ Order# [FFD]
- FD2: Supplier ID ⇒ Supplier Name [PD]
- FD3: Supplier ID, Date ⇒ Item [FFD]
- FD4: Supplier ID, Date ⇒ Price [FFD]
- FD5: Supplier ID, Date ⇒ Quantity [FFD]

All non-key attributes are fine except Supplier Name which is only dependent upon part of the primary key (Supplier ID). This is a partial dependency and needs to be resolved as follows.

Firstly, we take the determinant (Supplier ID) and the determined (Supplier Name) out into a new child relation – we will call it **Supplier Details**.

Supplier Details

Supplier ID	Supplier Name
1	Canine Care
2	Doggy Style
3	K94U
3	K94U
4	Canine Medical Supplies
4	Canine Medical Supplies
5	VetLine Vaccinations
6	Canine Blood Services

Note the redundancy caused by duplicating rows – which we can remove to get:

(Free) Sample 4

Supplier Details2

Supplier ID	Supplier Name
1	Canine Care
2	Doggy Style
3	K94U
4	Canine Medical Supplies
5	VetLine Vaccinations
6	Canine Blood Services

The remaining attributes are put into another child relation (with the same primary key) but with Supplier Name removed. We will call this relation **Orders 2**.

Order 2

Supplier ID	Date	Order#	Item	Price (£)	Quantity
1	22-Sept	1001	Surgical Forceps	16	12
2	22-Sept	1002	Rubber Gloves	0.10	1000
3	29-Sept	1003	Microchips	2.50	150
3	5-Oct	1004	Worming Tablets	0.50	1000
4	12-Oct	1005	Booster Jabs	10	500
4	19-Oct	1006	X-Ray Plates	18	50
5	21-Oct	1007	Puppy Jabs	12	200
6	02-Nov	1008	Epilepsy Test	9	25

By having Supplier ID in both new relations we retain a link that enables us to re-join the two new tables if necessary (remember the loss-less decomposition rule). We have now created a pair of 2NF relations. The original non-2NF relation is forgotten and never used again. Only the new child relations will be carried forward to the 3NF stage.

Here is a second 2NF example...

Appointment

Client ID	Date	Time	Treatment Code	Units	Vet ID	Total Cost (£)
1	12-Oct	9.30	1	1	11	25
1	19-Oct	11.00	11	1	12	10
2	22-Oct	2.00	2	1	10	18
2	29-Oct	2.30	4	1	5	15
2	01-Nov	3.15	11	1	8	10
4	03-Oct	10.30	9	2	4	150
4	14-Nov	11.15	5	1	3	250
7	05-Nov	2.30	10	1	2	45
9	02-Dec	4.00	8	3	2	75
10	10-Dec	12.00	12	2.5	1	375

(Composite) primary key is (Client ID, Date)

There are five non-key attributes

(Free) Sample 4

FD1: Client ID, Date \Rightarrow Time	[FFD]
FD2: Client ID, Date \Rightarrow Treatment Code	[FFD]
FD3: Client ID, Date \Rightarrow Units (of treatment)	[FFD]
FD4: Client ID, Date \Rightarrow Vet ID	[FDD]
FD5: Client ID, Date \Rightarrow Total Cost	[FFD]

Here there are no partial dependencies and so no need to decompose the relation – it is already in 2NF. This relation can now be moved onto the 3NF stage.