Normal Forms

Dr. Katharina Schneider & Dr. Tony Garnock-Jones

BCS1510

EPD150 MSM Conference Hall

30 April 2025

Part II

Design Theory for Relational Databases

Outline of Part II

1. Functional Dependencies & Keys
2. **Today:** Normal Forms
3. Query Tuning
4. Transactions; Beyond the Relational Model

Learning Objectives

Today we focus on *normal forms* as a critical step to good **relational schema design**.

1. Understand Normal Forms:
    - first, second, **third (3NF)**, **Boyce-Codd Normal Form (BCNF)**
    - definitions, purposes, data integrity issues
2. Analyze and classify Functional Dependencies
3. Understand the *lossless join* and *dependency preservation* properties, and when they hold
4. Be able to *refine* a Schema by *decomposition* or *synthesis*
    - bringing it into a Normal Form that you choose

Normal Forms 📈

First Normal Form (1NF)

Definition 1 (Connolly, Begg): A relation **R** is in **1NF** if the **intersection** of each row and column contains *one and only one value*.

× Not in 1NF:

Follower	Followee
Xylo	Lorin
Zara	Lorin
Mystar	Lorin; Zara
Lorin	Mystar; Zara

First Normal Form (1NF)

Definition 1 (Connolly, Begg): A relation **R** is in **1NF** if the **intersection** of each row and column contains *one and only one value*.

✓ In 1NF:

Follower	**Followee**
Xylo	Lorin
Zara	Lorin
Mystar	Lorin
Mystar	Zara
Lorin	Mystar
Lorin	Zara

First Normal Form (1NF)

Definition 2 (Date): A relation **R** is in 1NF if

- There's no *top-to-bottom* ordering to the rows.
- There's no *left-to-right* ordering to the columns.
- There are no *duplicate rows*.
- Every row-and-column **intersection** contains *exactly one value* from the applicable domain (and nothing else).
- All columns are regular, i.e. rows have no hidden components such as row IDs, object IDs, or hidden timestamps.

First Normal Form (1NF)

A variation on `Productsales`.

× Not in 1NF:

customer_id	city	model	maker	quantity	day	paid
9999999999	Limerick	1007, 3007	C, H	1, 3	2013-12-20, 2013-12-20	459.00, 360.00
9876543210	Ennis	1007	C	3	2013-12-19	1530
1122334455	Dublin	2010	G	1	2013-12-19	2300

First Normal Form (1NF)

A variation on `Productsales`.

✓ In 1NF:

customer_id	city	model	maker	quantity	day	paid
9999999999	Limerick	1007	C	1	2013-12-20	459.00
9999999999	Limerick	3007	H	3	2013-12-20	360.00
9876543210	Ennis	1007	C	3	2013-12-19	1530.00
1122334455	Dublin	2010	G	1	2013-12-19	2300.00

1NF: Why?

It allows query and manipulation using *first-order logic*:

> The adoption of a relational model of data, as described above, permits the development of a universal data sublanguage based on an applied predicate calculus.
>
> A first-order predicate calculus suffices if the collection of relations is in [first] normal form.
>
> — E.F. Codd, "A Relational Model of Data for Large Shared Data Banks," 1970.

Aside: Codd's relational model

E. F. Codd, *"A Relational Model of Data for Large Shared Data Banks,"* Comm. ACM, vol. 13, no. 6, Jun. 1970, doi: 10.1145/362384.362685.

- Classic paper; freely available
- Readable; short; worth a look

Information Retrieval P. BAXENDALE, Editor

A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on *n*-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

KEY WORDS AND PHRASES: data bank, data base, data structure, data organization, hierarchies of data, networks of data, relations, derivability, redundancy, consistency, composition, join, retrieval language, predicate calculus, security, data integrity
CR CATEGORIES: 3.70, 3.73, 3.75, 4.20, 4.22, 4.29

1. Relational Model and Normal Form

1.1. INTRODUCTION

This paper is concerned with the application of elementary relation theory to systems which provide shared access to large banks of formatted data. Except for a paper by Childs [1], the principal application of relations to data systems has been to deductive question-answering systems. Levein and Maron [2] provide numerous references to work in this area.

In contrast, the problems treated here are those of *data independence*—the independence of application programs and terminal activities from growth in data types and changes in data representation—and certain kinds of *data inconsistency* which are expected to become troublesome even in nondeductive systems.

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for noninferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various parts of this paper. Implementations of systems to support the relational model are not discussed.

1.2. DATA DEPENDENCIES IN PRESENT SYSTEMS

The provision of data description tables in recently developed information systems represents a major advance toward the goal of data independence [5, 6, 7]. Such tables facilitate changing certain characteristics of the data representation stored in a data bank. However, the variety of data representation characteristics which can be changed *without logically impairing some application programs* is still quite limited. Further, the model of data with which users interact is still cluttered with representational properties, particularly in regard to the representation of collections of data (as opposed to individual items). Three of the principal kinds of data dependencies which still need to be removed are: ordering dependence, indexing dependence, and access path dependence. In some systems these dependencies are not clearly separable from one another.

1.2.1. *Ordering Dependence.* Elements of data in a data bank may be stored in a variety of ways, some involving no concern for ordering, some permitting each element to participate in one ordering only, others permitting each element to participate in several orderings. Let us consider those existing systems which either require or permit data elements to be stored in at least one total ordering which is closely associated with the hardware-determined ordering of addresses. For example, the records of a file concerning parts might be stored in ascending order by part serial number. Such systems normally permit application programs to assume that the order of presentation of records from such a file is identical to (or is a subordering of) the

Volume 13 / Number 6 / June, 1970 **Communications of the ACM** 377

Keys of Productsales 🔑

Today's `Productsales` includes a column `routingkey`.

- *model* → *maker type*
- *customer_id* → *firstname lastname city address email*
- *city address* → *routingkey*
- *customer_id model day* → *quantity paid type_of_payment*

Based on these Functional Dependencies (FDs), the only key of `Productsales` has to be

{*customer_id*, *model*, *day*}

Analysis of the FDs

The key is {*customer_id, model, day*}. The FDs are:

- *model → maker type*
- *customer_id → firstname lastname city address email*
- *city address → routingkey*
- *customer_id model day → quantity paid type_of_payment*

Non-trivial partial dependencies are where the RHS has a non-key[*] attribute which depends on *part*, but not all, of a key

---

[*] A *non-key* attribute is an attribute that does not appear in *any* key of **R**; a.k.a “nonprime”.

Analysis of the FDs

The key is {*customer_id, model, day*}. The FDs are:

- *model → maker type*
- *customer_id → firstname lastname city address email*
- *city address → routingkey*
- *customer_id model day → quantity paid type_of_payment*

Non-trivial transitive dependencies are indicated where a RHS depends on an attribute that is *not* part of a key:

> *X → A* in **R** is *transitive* if ∃ attributes *Z* in **R** neither a key nor a subset of any key of **R**, and both X → Z and Z → A hold.

Analysis of the FDs

The key is {*customer_id, model, day*}. The FDs are:

- *model → maker type*
- *customer_id → firstname lastname city address email*
- *city address → routingkey*
- *customer_id model day → quantity paid type_of_payment*

∴ *customer_id → routingkey*

Analysis of the FDs

Non-trivial **partial** and **transitive** dependencies are an indication for update and deletion anomalies.

- **Update anomaly**: When we update a single occurrence of a fact in the table, other occurrences of the same fact in the same table remain unchanged.
- **Deletion anomaly**: When we delete a fact from the table, we also delete another *unrelated* fact from the table.

Second Normal Form (2NF)

R is in 2NF if it is in 1NF and no non-key* attribute of **R** is partially functionally dependent on any key of **R**.

A relation **R** in 2NF *may* contain non-trivial transitive FDs.

Reminder:

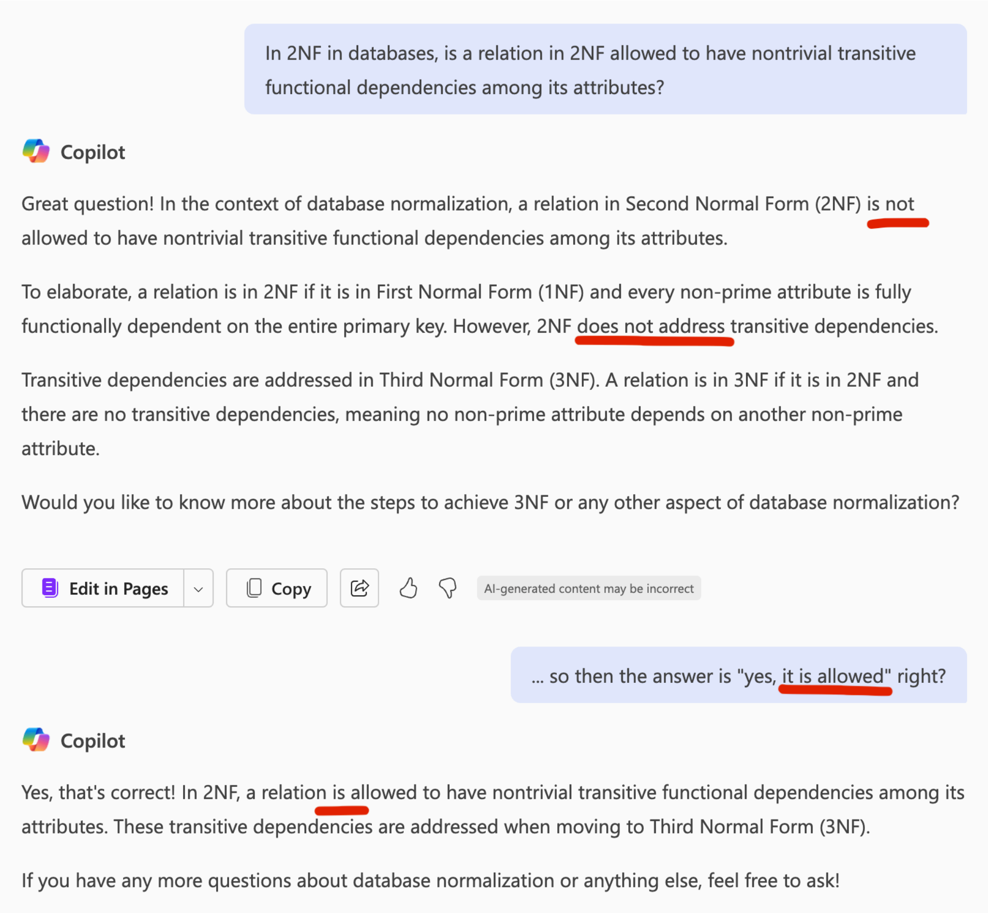
> $X \rightarrow A$ in **R** is *transitive* if ∃ attributes Z in **R** neither a key nor a subset of any key of **R**, and both $X \rightarrow Z$ and $Z \rightarrow A$ hold.

* A *non-key* attribute is an attribute that does not appear in *any* key of **R**; a.k.a "nonprime".

Aside: LLMs are “Bullshit”

> In philosophy and psychology of cognition, the term “bullshit” is sometimes used to specifically refer to statements produced without particular concern for truth, clarity, or meaning, distinguishing “bullshit” from a deliberate, manipulative lie intended to subvert the truth. — Wikipedia, *Bullshit*, 29 April 2025, citing Pennycook *et al.* 2023, *v.i.*

> ... Here we focus on pseudo-profound bullshit, which consists of seemingly impressive assertions that are presented as true and meaningful but are actually vacuous.
> — Pennycook *et al.*, "On the reception and detection of pseudo-profound bullshit", Judgement and Decision Making 10(6), 2023



In 2NF in databases, is a relation in 2NF allowed to have nontrivial transitive functional dependencies among its attributes?

Copilot

Great question! In the context of database normalization, a relation in Second Normal Form (2NF) is not allowed to have nontrivial transitive functional dependencies among its attributes.

To elaborate, a relation is in 2NF if it is in First Normal Form (1NF) and every non-prime attribute is fully functionally dependent on the entire primary key. However, 2NF does not address transitive dependencies.

Transitive dependencies are addressed in Third Normal Form (3NF). A relation is in 3NF if it is in 2NF and there are no transitive dependencies, meaning no non-prime attribute depends on another non-prime attribute.

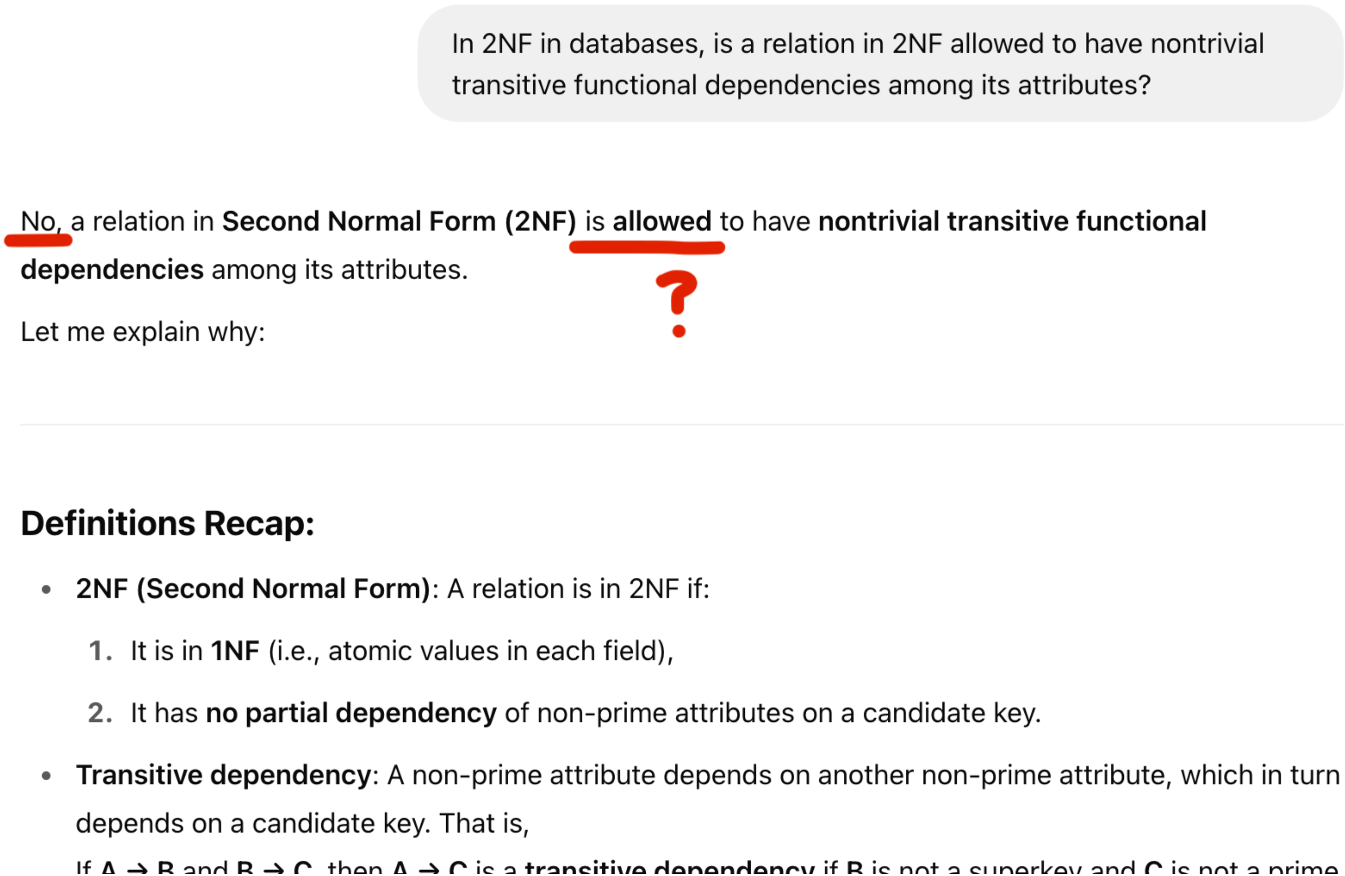
Would you like to know more about the steps to achieve 3NF or any other aspect of database normalization?

... so then the answer is "yes, it is allowed" right?

Copilot

Yes, that's correct! In 2NF, a relation is allowed to have nontrivial transitive functional dependencies among its attributes. These transitive dependencies are addressed when moving to Third Normal Form (3NF).

If you have any more questions about database normalization or anything else, feel free to ask!



In 2NF in databases, is a relation in 2NF allowed to have nontrivial transitive functional dependencies among its attributes?

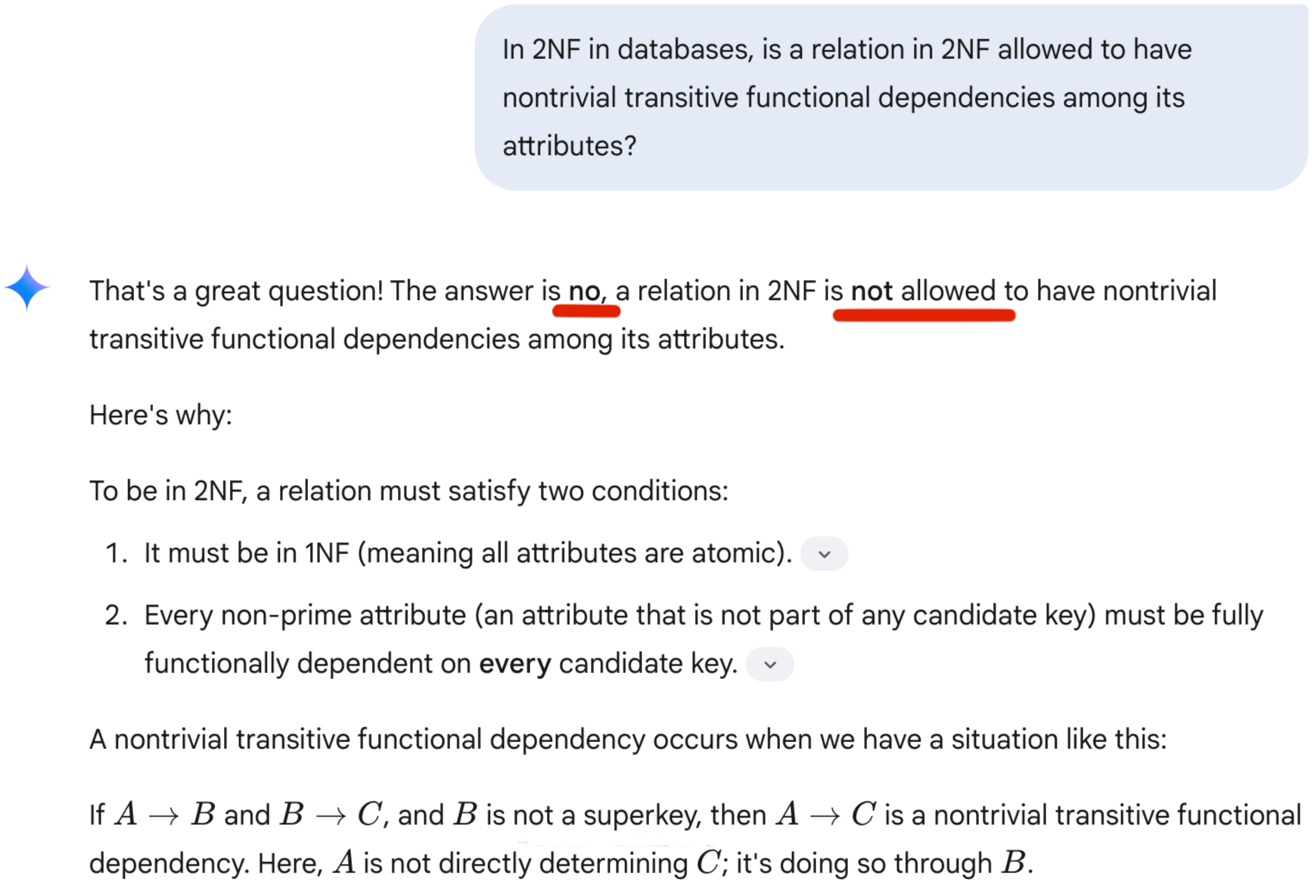
No, a relation in **Second Normal Form (2NF)** is **allowed** to have **nontrivial transitive functional dependencies** among its attributes.

?

Let me explain why:

Definitions Recap:

- **2NF (Second Normal Form)**: A relation is in 2NF if:
  1. It is in **1NF** (i.e., atomic values in each field),
  2. It has **no partial dependency** of non-prime attributes on a candidate key.
- **Transitive dependency**: A non-prime attribute depends on another non-prime attribute, which in turn depends on a candidate key. That is,
  If **A → B** and **B → C**, then **A → C** is a **transitive dependency** if B is not a superkey and **C** is not a prime



In 2NF in databases, is a relation in 2NF allowed to have nontrivial transitive functional dependencies among its attributes?

That's a great question! The answer is **no**, a relation in 2NF is **not** allowed to have nontrivial transitive functional dependencies among its attributes.

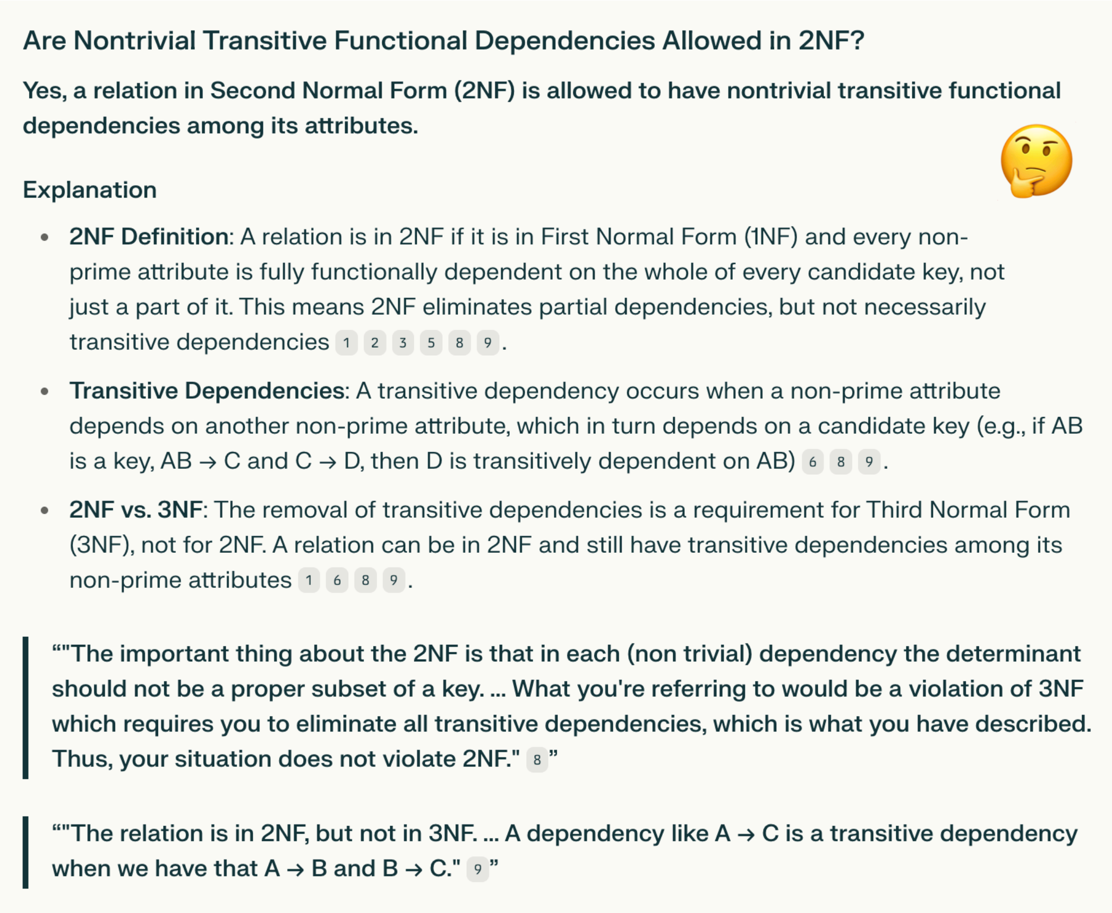
Here's why:

To be in 2NF, a relation must satisfy two conditions:

1. It must be in 1NF (meaning all attributes are atomic).
2. Every non-prime attribute (an attribute that is not part of any candidate key) must be fully functionally dependent on **every** candidate key.

A nontrivial transitive functional dependency occurs when we have a situation like this:

If $A \to B$ and $B \to C$, and B is not a superkey, then $A \to C$ is a nontrivial transitive functional dependency. Here, A is not directly determining C; it's doing so through B.



Are Nontrivial Transitive Functional Dependencies Allowed in 2NF?

Yes, a relation in Second Normal Form (2NF) is allowed to have nontrivial transitive functional dependencies among its attributes.

🤔

Explanation

- **2NF Definition**: A relation is in 2NF if it is in First Normal Form (1NF) and every non-prime attribute is fully functionally dependent on the whole of every candidate key, not just a part of it. This means 2NF eliminates partial dependencies, but not necessarily transitive dependencies [1] [2] [3] [5] [8] [9].
- **Transitive Dependencies**: A transitive dependency occurs when a non-prime attribute depends on another non-prime attribute, which in turn depends on a candidate key (e.g., if AB is a key, AB → C and C → D, then D is transitively dependent on AB) [6] [8] [9].
- **2NF vs. 3NF**: The removal of transitive dependencies is a requirement for Third Normal Form (3NF), not for 2NF. A relation can be in 2NF and still have transitive dependencies among its non-prime attributes [1] [6] [8] [9].

> **“"The important thing about the 2NF is that in each (non trivial) dependency the determinant should not be a proper subset of a key. ... What you're referring to would be a violation of 3NF which requires you to eliminate all transitive dependencies, which is what you have described. Thus, your situation does not violate 2NF."** [8] **”**

> **“"The relation is in 2NF, but not in 3NF. ... A dependency like A → C is a transitive dependency when we have that A → B and B → C."** [9] **”**

Anyway! Where were we

If you want to clarify any of this stuff, don't ask an LLM: *read the textbook!* It's just as readable, and is actually correct

Second Normal Form (2NF)

R is in 2NF if it is in 1NF and no non-key attribute of **R** is partially functionally dependent on any key of **R**.

R is in 2NF if it is in 1NF and, for each attribute A of **R**, either

1. A is part of some key of **R**, or
2. for all keys K of **R**, there is no FD $X \rightarrow A$, where $X \subset K$.

Second Normal Form (2NF)

R is in 2NF if it is in 1NF and no non-key attribute of **R** is partially functionally dependent on any key of **R**.

A nontrivial FD $X \rightarrow Y$ is a 2NF **violation** if

- some $B \in Y$ is a non-key attribute **AND**
- X is not a superkey, and some $A \in X$ is part of a key of **R**.

Decomposition of `Productsales` ✂️

Separating it into 3× 2NF tables

Follow the **partial** FDs:

- *model → maker type*
- *customer_id → firstname lastname city address email*
- *city address → routingkey*
- *customer_id model day → quantity paid type_of_payment*

Decomposition of `Productsales` into 3× 2NF tables ✂️

- 1. Products

model	maker	type
1007	C	pc
3007	H	printer
2010	G	laptop

- **key:** {*model*}
- *model* → *maker type*

Decomposition of `Productsales` into 3× 2NF tables ✂️

- 2. Customers

customer_id	**(...)**	**city**	**address**	**routingkey**
9999999999	(...)	Limerick	2 Thomas St.	V94
9876543210	(...)	Galway	101 O'Connell St.	H91
1122334455	(...)	Dublin	1 Jervis St.	D01

- **key:** {*customer_id*}
- *customer_id* → *firstname lastname city address email*; *city address* → *routingkey*
- *city address* → *routingkey* indication for the transitive dependency *customer_id* → *routingkey*

Decomposition of `Productsales` into 3× 2NF tables ✂️

- 3. Sales

customer_id	model	quantity	day	paid	type_of_payment
9999999999	1007	1	2013-12-20	459.00	visa credit
9999999999	3007	2	2013-12-20	360.00	visa credit
9876543210	1007	3	2013-12-19	1530.00	visa debit
1122334455	2010	1	2013-12-19	2300.00	mastercard credit

- **key:** {*customer_id, model, day*}
- *customer_id model day → quantity paid type_of_payment*

Remaining redundancy: transitivity

customer_id	(...)	city	address	routingkey
9999999999	(...)	Limerick	2 Thomas St.	V94
9876543210	(...)	Galway	101 O'Connell St.	H91
1122334455	(...)	Dublin	1 Jervis St.	D01

- *customer_id → firstname lastname city address email*
- *city address → routingkey*

Boyce-Codd Normal Form (BCNF)

BCNF is 2NF, without transitive FDs.

R is in *BCNF* if, for each *nontrivial* FD $X \rightarrow Y$ in **R**, X is a superkey.

- nontrivial: Y not contained in X.
- superkey: a key, or a superset of a key

There is no need to bring a relation first into 2NF and then to BCNF. Decomposition into BCNF gets rid of all partial dependencies **and** all transitive dependencies.

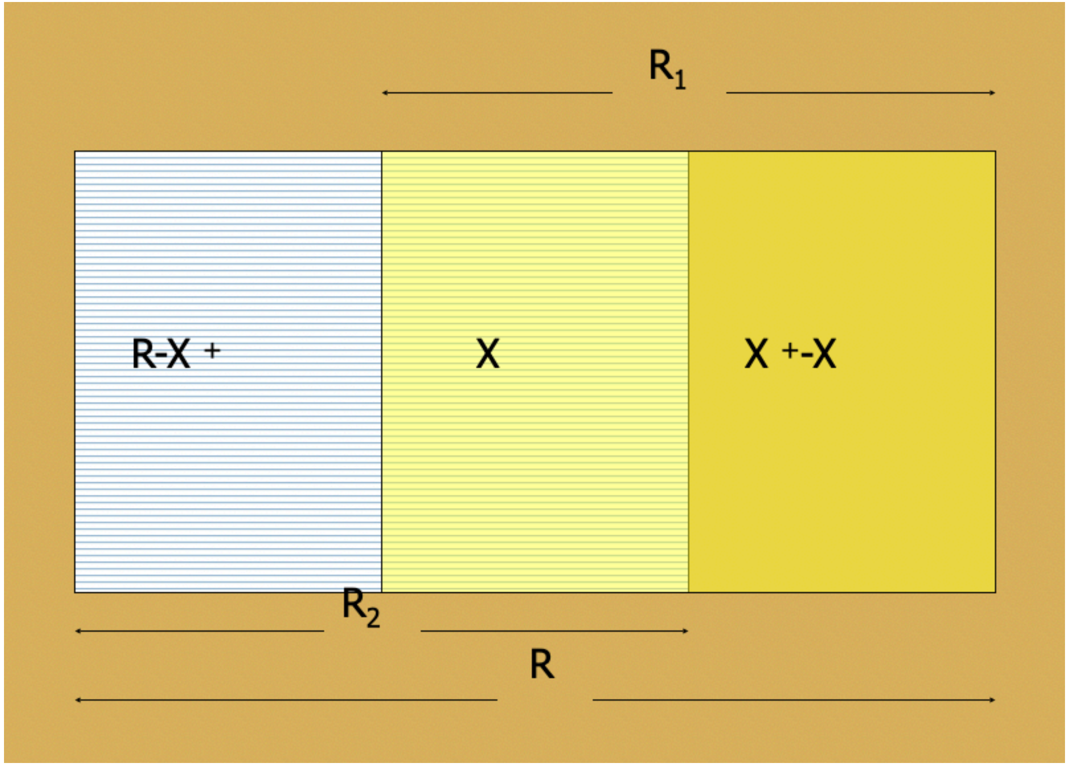
Decomposition into BCNF ✂️

- Step 1: Find some BCNF violation $X \to Y$ in the FDs of **R**.
- Step 2: Compute $X+$. This will not be **all** attributes of **R**! … **Question:** Why?
- Step 3: Replace **R** by $\mathbf{R_1}(X+)$ and $\mathbf{R_2}(X \cup (\mathit{attrs}(\mathbf{R}) - X+))$
  - The schema of $\mathbf{R_1}$ is $X+$
  - The schema of $\mathbf{R_2}$ is the remaining attributes, *and X*
- Step 4: Project all non-transitive FDs of **R** onto $\mathbf{R_1}$, $\mathbf{R_2}$.
- Step 5: Recursively decompose $\mathbf{R_1}$ and $\mathbf{R_2}$.*

---

* You actually have to look at $F+$, the closure of the set F of FDs of **R**, when recursing.

Decomposition into BCNF ✂️



Example: BCNF Decomposition 🛠️

customer_id	(...)	city	address	routingkey
9999999999	(...)	Limerick	2 Thomas St.	V94
9876543210	(...)	Galway	101 O'Connell St.	H91
1122334455	(...)	Dublin	1 Jervis St.	D01

- **key:** {*customer_id*}
- *customer_id → firstname lastname city address email*
- *city address → routingkey*

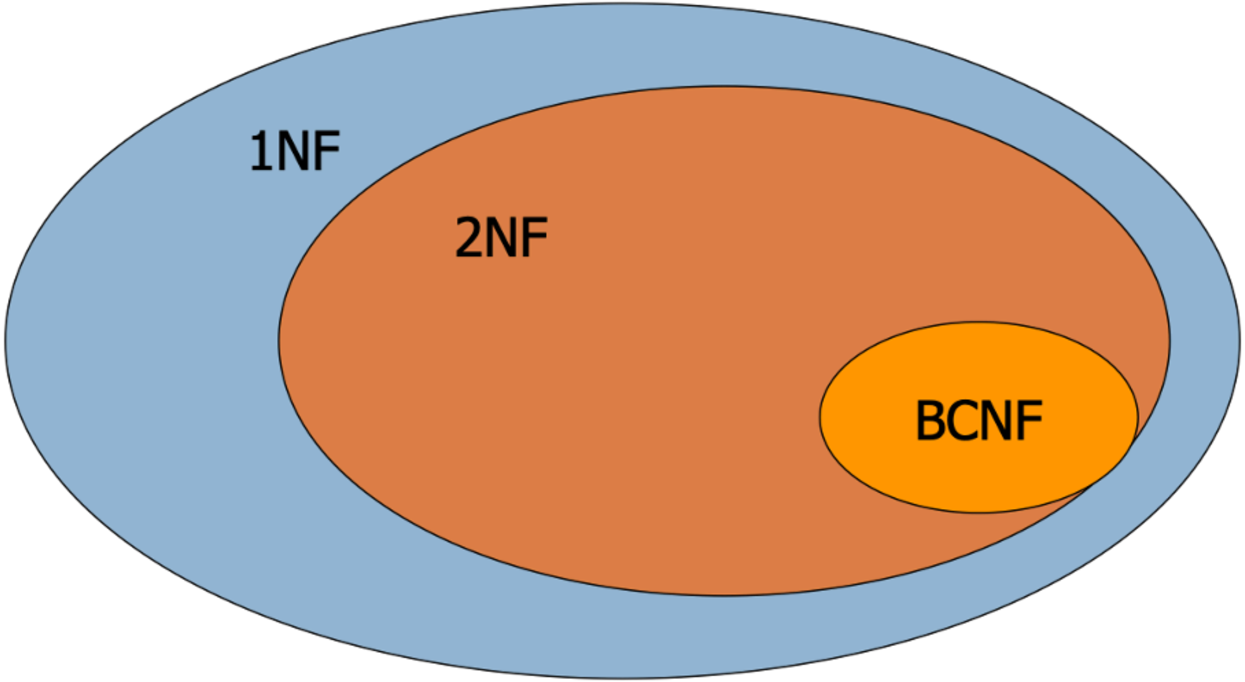
BCNF Limitations

Problem: It is not always possible to project the FDs to the new relations in the decomposition.

Therefore, after decomposition, some FDs may contain attributes which do not occur together in a single table.

- Hard to maintain consistency when that happens
- Would be nice if all FDs could be assigned to a *single* table!

1NF ⊃ 2NF ⊃ BCNF



Third Normal Form (3NF)

3NF: Motivation

Certain patterns of FDs cause trouble when we decompose a relation to avoid transitive dependencies.

- $AB \to C$
- $C \to B$

Keys: $\{AB\}$, $\{AC\}$

3NF: Motivation

Example: ZipCodes(street, city, code)

- *street city → code*
- *code → city*

Keys: {*street city*}, {*street code*}

3NF: Motivation

Example: ZipCodes(street, city, code)

- *street city → code*
- *code → city* ← **our chosen BCNF violation**

Keys: {*street city*}, {*street code*}

{*code*}+ = {*code city*}

∴ new relations: Streets(street, code), Cities(code, city)

3NF: Motivation

New BCNF relations: Streets(street, code), Cities(code, city)

Now, step 4: partition FDs across the new relations.

- *street city → code*
- *code → city*

Clearly, the second can go to Cities. But the first...? 🙁
Knowing the street & city no longer allows *direct* determination of the zip code.

3NF: Motivation

Street	Code
Stationsstraat	6221BM
Stationsstraat	6247BJ

Code	City
6221BM	Maastricht
6247BJ	Gronsveld

Street	City	Code
Stationsstraat	Maastricht	6221BM
Stationsstraat	Gronsveld	6247BJ

street city → code ✓ *code → city* ✓ 😐

3NF: Motivation

Street	Code
Stationsstraat	6221BM
Stationsstraat	6247BJ

Code	City
6221BM	Maastricht
6247BJ	Maastricht

Street	City	Code
Stationsstraat	Maastricht	6221BM
Stationsstraat	Maastricht	6247BJ

street city → code × *code → city* ✓ 😱

Third Normal Form (3NF)

R is in *3NF* if, for each *nontrivial* FD $X \rightarrow Y$ in **R**, X is a superkey…

… **OR** every $A \in Y - X$ is contained in some key of **R** ("prime").

A nontrivial FD $X \rightarrow Y$ is a 3NF **violation** if

- X is not a superkey
- **AND** some $A \in Y - X$ is nonprime.

Third Normal Form (3NF)

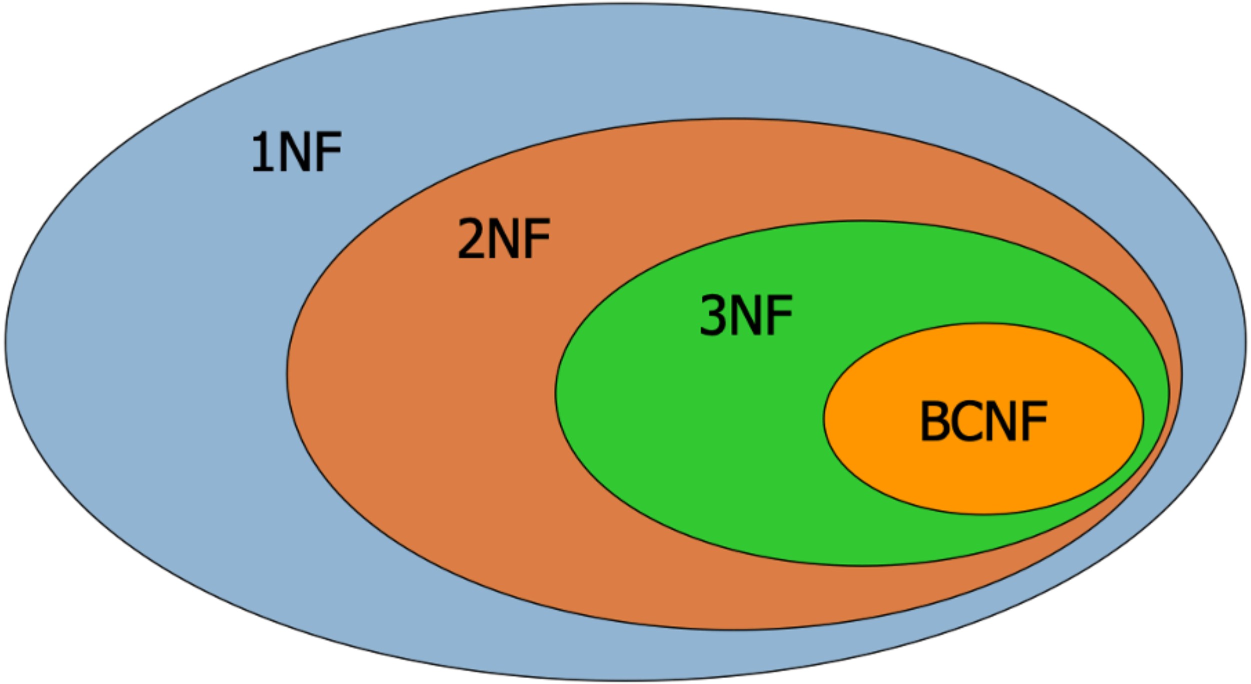
Let's look again at the problematic example:

FD	BCNF violation?	3NF violation?
street city → *code*	no	no
code → *city*	**yes**, {*code*} is not a superkey	**no**, *city* ∈ {*street city*}, which is a key

Keys: {*street city*}, {*street code*}

∴ ZipCodes(street, city, code) is in 3NF!

1NF ⊃ 2NF ⊃ 3NF ⊃ BCNF



What 3NF and BCNF Give You

There are three important properties we want from these decompositions:

1. *Minimal* redundancy: squeeze out every last duplication!
2. Lossless join: can we re-join the decomposed relations to get the original?
3. Dependency preservation: can we preserve and check *all* the FDs we started with, after we decompose?

What 3NF and BCNF Give You

Property	**BCNF**	**3NF**
Minimal redundancy	**yes**[*]	no
Lossless join	**yes**	**yes**
Dependency preservation	no	**yes**

Silberschatz et al. (§7.3.3) give their opinion (and it *is* an *opinion*, not a fact!):

> even if we are not able to get a dependency-preserving BCNF decomposition, it is still preferable to opt for BCNF, since checking functional dependencies other than primary key constraints is difficult in SQL.

Break

3NF Synthesis Algorithm

A different approach: starting from the FDs of **R**, *synthesise* an entirely new, equivalent set of relations.

1. Are there any FDs of **R** that violate 3NF? If not, stop.
2. Compute a *canonical cover* F_c for the FDs.
3. For each FD $X \rightarrow A$ in F_c, add a relation with schema XA.
4. Is any of the XA a superkey for **R**? If not, add another relation with schema any of the keys for **R**.
5. Remove redundant relations (schema subset of another)

3NF Synthesis Algorithm

A different approach: starting from the FDs of **R**, *synthesise* an entirely new, equivalent set of relations.

1. Are there any FDs of **R** that violate 3NF? If not, stop.
2. Compute a *canonical cover* F_c for the FDs.
3. Schemas = $\{ XA \mid X \rightarrow A \in F_c \}$
4. If Schemas ∩ *superkeys*(**R**) = ∅, then add one of *keys*(**R**) to Schemas.
5. Remove $\{ Z \mid Z \in \text{Schemas}, Y \in \text{Schemas}, Z \subseteq Y \}$

Canonical cover

A **minimal basis** F_c for a set of FDs F:

- RHSs are single attributes (split them).
- No FD in F can be removed.
- No attribute can be removed from a LHS of an FD in F.

To **construct** one: first, split the FDs so they all have a single attribute on the RHS; then, repeatedly remove whole FDs and/or LHS attributes if they *don't change the meaning*.

Canonical cover: Redundant FDs

Consider some $X \rightarrow B$ in the partially-finished F_c we're building.

$X \rightarrow B$ can be removed from F_c if

$$F_c - \{X \rightarrow B\} \text{ logically implies } F_c$$

Example. Let $F_c = \{ AB \rightarrow C, AB \rightarrow D, A \rightarrow C \}$. Now, $AB \rightarrow C$ can be removed because the stronger $A \rightarrow C$ can be weakened to recover $AB \rightarrow C$.

Canonical cover: Extraneous LHS attributes

Consider some $X \rightarrow B$ in the partially-finished F_C we're building.

A is **extraneous** in $X \rightarrow B$, and can be removed, if $A \in X$ and

$$F_C \text{ logically implies } (F_C - \{X \rightarrow B\}) \cup \{(X - A) \rightarrow B\}$$

Example. Let $F_C = \{ AB \rightarrow C, A \rightarrow D, D \rightarrow C \}$. Now, B is extraneous in $AB \rightarrow C$, because we can show $A \rightarrow C$ by combining $A \rightarrow D$ and $D \rightarrow C$.

Wait: Why are we doing this again?

1. Trying to synthesize 3NF relations from some FDs
2. We need a *canonical cover*
3. So we need to figure out which FDs and/or attributes are redundant
4. So we need to figure out how one set of FDs can imply another
5. ... and that's why we're about to look at a small *logic*

Logical implication among FDs

Figuring out the FDs **implied by** some set of FDs F.

Apply *Armstrong's axioms*:

- **Reflexivity**. If $Y \subseteq X$, then $X \rightarrow Y$.
- **Augmentation**. If $X \rightarrow Y$, then $ZX \rightarrow ZY$.
- **Transitivity**. If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.

Properties: both *sound* (correct FDs) and *complete* (all FDs).

Logical implication among FDs

Figuring out the FDs **implied by** some set of FDs F.

Use derived rules:

- **Union**. If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$.
- **Decomposition**. If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$.
- **Pseudotransitivity**. If $X \rightarrow Y$ and $QY \rightarrow Z$, then $XQ \rightarrow Z$.

These follow from (are provable using) Armstrong's axioms.

Phew! Back to 3NF

3NF Synthesis Algorithm, again

1. Are there any FDs of **R** that violate 3NF? If not, stop.
2. Compute a *canonical cover* F_c for the FDs.
3. For each FD $X \to A$ in F_c, add a relation with schema XA.
4. Is any of the XA a superkey for **R**? If not, add another relation with schema any of the keys for **R**.
5. Remove redundant relations (schema subset of another)

Example: 3NF Synthesis

R($ABCD$).

- FDs $F = \{A \rightarrow B, A \rightarrow C\}$
- Keys: $\{AD\}$
- Cover $F_C = F$

Thus, decomposition is $\mathbf{R_1}(AB)$ and $\mathbf{R_2}(AC)$ from F_C, plus $\mathbf{R_3}(AD)$ for the key.

Example: 3NF Synthesis

Let's try working through Productsales.

Properties of 3NF Synthesis

- **Preserves dependencies**: each FD from a minimal basis is *directly* represented as a relation, thus preserved *by construction*.
- **Lossless Join**: harder to see 🙂 but can be proved!
- ~~Unique solution~~: No, canonical covers are not necessarily unique for a given set of FDs
- **Produces a 3NF design**: this is the hardest part to show, but again is a theorem. See Silberschatz §7.5.3 for the proof

Higher normal forms: 4NF, 5NF, 6NF, ...

- BCNF ⊃ **4NF**: handles redundancy from *independent sets*
  - goes beyond the kinds of redundancies FDs can be used to eliminate
  - need a new form of specification: a *Multivalued Dependency*
- **5NF**: even *more* general
  - another new form of specification: "join dependency"
  - hard to reason with, hard to use, very rarely seen

If you're interested in these: there's a **freely-available**, online-only chapter of Silberschatz *et al.* called Chapter 28: Advanced Relational Database Design.

See you in the lab 👋🏼

BCNF and *F+*, the closure of the set of FDs

We use *F+*, not *F*, after the first cut in a BCNF decomposition.

Example: Consider **R**(*ABCDE*), with $A \to B$ and $BC \to D$. The key is thus {*ACE*}.

Pick $A \to B$ as the violation to address. Then $\mathbf{R_1}(AB)$ and $\mathbf{R_2}(ACDE)$.

Neither FD contains only attributes from *ACDE*. Are we done? **No.**

There is $AC \to D \in F+$, but $\notin F$:

$$A \to B \wedge BC \to D \Rightarrow AC \to D \text{ (pseudotransitivity)}$$

and $AC \to D$ is a BCNF violation for $\mathbf{R_2}$. So we continue splitting.

Example due to Silberschatz et al. §7.5.1.1; also given is a different technique that only uses *F*.