Carnegie Mellon University

Normal Forms

Lecture #05

Database Systems
15-445/15-645
Fall 2017

Andy Pavlo
Computer Science Dept.
Carnegie Mellon Univ.

DATABASE DESIGN

How do we design a “good” database?

We want to ensure the integrity of the data.

We also want to get good performance.

TODAY'S AGENDA

Normal Forms

NoSQL Denormalization

NORMAL FORMS

Now that we know how to derive more FDs, we can then:

→ Search for "bad" FDs
→ If there are such, then decompose the table into two tables, repeat for the sub-tables.
→ When done, the database schema is normalized.

NORMAL FORMS

A normal form is a characterization of a decomposition in terms of the properties that satisfies when putting the relations back together.

→ Also called the "universal relation"

Loseless Joins

Dependency Preservation

Redundancy Avoidance

DECOMPOSITION SUMMARY

Lossless Joins

→ Motivation: Avoid information loss.
→ Goal: No noise introduced when reconstituting universal relation via joins.
→ Test: At each decomposition $R=(R_1 \cup R_2)$, check whether $(R_1 \cap R_2) \rightarrow R_1$ or $(R_1 \cap R_2) \rightarrow R2$.

DECOMPOSITION SUMMARY

Dependency Preservation

→ Motivation: Efficient FD assertions.
→ Goal: No global integrity constraints that require joins of more than one table with itself.
→ Test: $R=(R_1 \cup \ldots \cup R_n)$ is dependency preserving if closure of FD's covered by each R_1 = closure of FD's covered by $R=F$.

DECOMPOSITION SUMMARY

Redundancy Avoidance

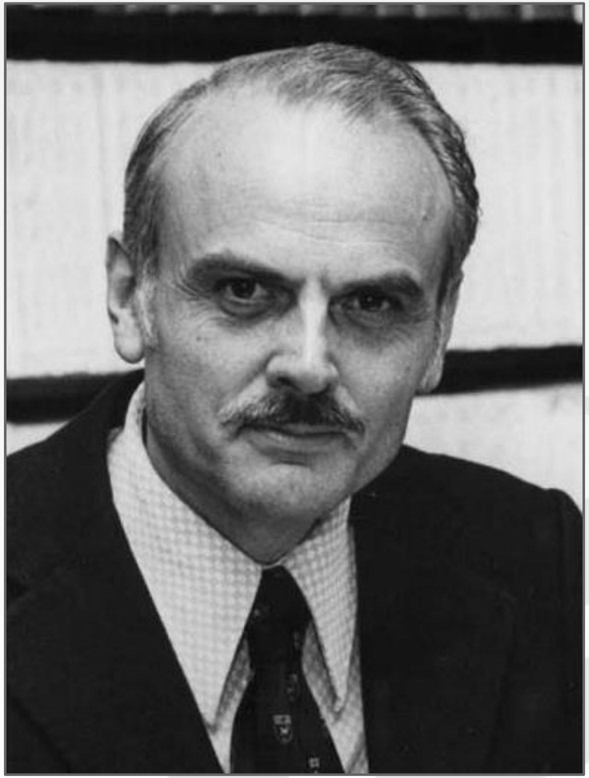
→ Motivation: Avoid update, delete anomalies.
→ Goal: Avoid update anomalies, wasted space.
→ Test: For an $X \rightarrow Y$ covered by R_n, X should be a super key of R_n.

HISTORY

Ted Codd introduced the concept of normalization and the **first normal form** in 1970.

Codd went on to define the **second normal form** and **third normal form** in 1971.

Codd and Raymond Boyce defined the **Boyce-Codd normal form** in 1974



Edgar F. Codd

CARNEGIE MELLON DATABASE GROUP

NORMAL FORMS

1st Normal Form (1NF) → All Tables are Flat

2nd Normal Form (2NF) → "Good Enough"

3rd Normal Form (3NF) → Most Common

Boyce-Codd Normal Form (BCNF) → Most Common

4th & 5th Normal Forms → See textbook

6th Normal Form → Most (normal) people never need this.

More Restrictive

MORE NORMAL FORMS

Domain-Key Normal Form (1981)

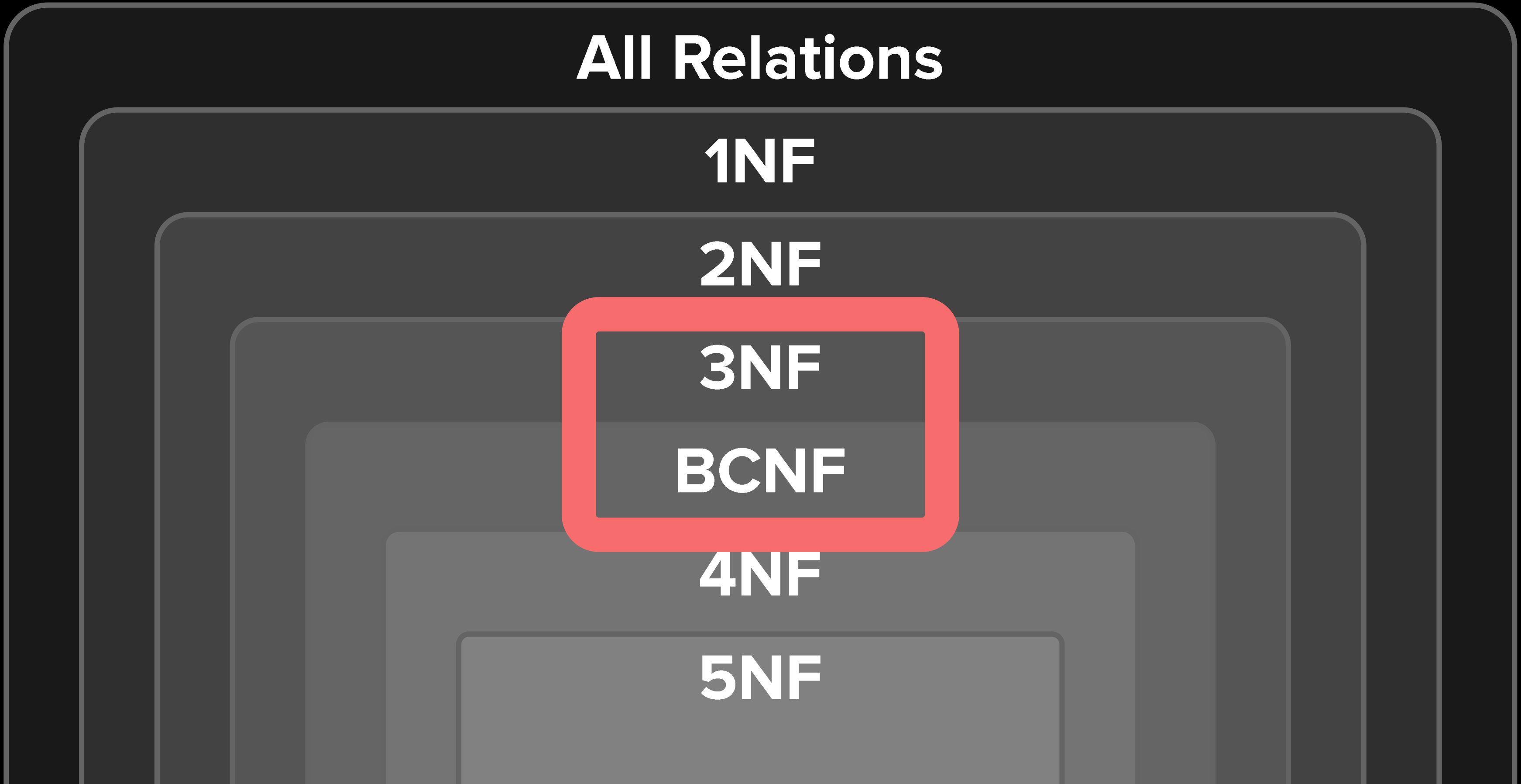
Elementary Key Normal Form (1982)

Inclusion Normal Form (1992)

Key-Complete Normal Form (1998)

Inclusion Dependency Normal Form (2000)

THE UNIVERSE OF RELATIONS



FIRST NORMAL FORM

All types must be atomic.

No repeating groups.

```
loans(bname,assets,cname,loanId,amt)
```

bname	assets	cname	loanId	amt
Pittsburgh	$9M	[Andy, DJ Snake]	L-17	$1000
Pittsburgh	$9M	Obama	L-23	$2000
Los Angeles	$2M	Andy	L-93	$500

FIRST NORMAL FORM

All types must be atomic.

No repeating groups.

```
loans(bname,assets,cname1,cname2,…,loanId,amt)
```

bname	assets	cname1	cname2	cname…	loanId	amt
Pittsburgh	$9M	Andy	DJ Snake		L-17	$1000
Pittsburgh	$9M	Obama			L-23	$2000
Los Angeles	$2M	Andy			L-93	$500

FIRST NORMAL FORM

All types must be atomic.

No repeating groups.

`loans(bname,assets,cname,loanId,amt)`

bname	assets	cname	loanId	amt
Pittsburgh	$9M	Andy	L-17	$1000
Pittsburgh	$9M	Obama	L-23	$2000
Los Angeles	$2M	Andy	L-93	$500
Pittsburgh	$9M	DJ Snake	L-17	$1000

✓ Valid 1NF

SECOND NORMAL FORM

Provided FDs
bname → assets
loanId → amt,bname

1NF and non-key attributes fully depend on the candidate key.

loans(bname,assets,cname,loanId,amt)

bname	assets	cname	loanId	amt
Pittsburgh	$9M	Andy	L-17	$1000
Pittsburgh	$9M	Obama	L-23	$2000
Los Angeles	$2M	Andy	L-93	$500
Pittsburgh	$9M	DJ Snake	L-17	$1000

SECOND NORMAL FORM

Provided FDs

bname → assets

loanId → amt,bname

1NF and non-key attributes fully depend on the candidate key.

R_1(bname,assets,cname,loanId)

bname	assets	cname	loanId
Pittsburgh	$9M	Andy	L-17
Pittsburgh	$9M	Obama	L-23
Los Angeles	$2M	Andy	L-93
Pittsburgh	$9M	DJ Snake	L-17

R_2(loanId,bname,amt)

loanId	bname	amt
L-17	Pittsburgh	$1000
L-23	Pittsburgh	$2000
L-93	Los Angeles	$500

SECOND NORMAL FORM

Provided FDs
bname → assets
loanId → amt,bname

1NF and non-key attributes fully depend on the candidate key.

R_1(bname,assets,cname,loanId)

bname	assets	cname	loanId
Pittsburgh	$9M	Andy	L-17
Pittsburgh	$9M	Obama	L-23
Los Angeles	$2M	Andy	L-93
Pittsburgh	$9M	DJ Snake	L-17

R_2(loanId,bname,amt)

loanId	bname	amt
L-17	Pittsburgh	$1000
L-23	Pittsburgh	$2000
L-93	Los Angeles	$500

SECOND NORMAL FORM

Provided FDs
bname → assets
loanId → amt,bname

1NF and non-key attributes fully depend on the candidate key.

R_1(bname,assets)

bname	assets
Pittsburgh	$9M
Los Angeles	$2M

R_3(bname,cname,loanId)

bname	cname	loanId
Pittsburgh	Andy	L-17
Pittsburgh	Obama	L-23
Los Angeles	Andy	L-93
Pittsburgh	DJ Snake	L-17

R_2(loanId,bname,amt)

loanId	bname	amt
L-17	Pittsburgh	$1000
L-23	Pittsburgh	$2000
L-93	Los Angeles	$500

✓ Valid 2NF

CARNEGIE MELLON DATABASE GROUP

BOYCE-CODD NORMAL FORM

BCNF guarantees no redundancies and no lossless joins (but not DP).

A relation **R** with FD set **F** is in BCNF if for all non-trivial **X→Y** in **F+**:
→ **X→R** (i.e., **X** is a super key)

BOYCE-CODD NORMAL FORM (EX.1)

Is R in BCNF?

Consider the non-trivial dependencies in F+:

A→B, A→R (A is a super key)

A→C, A→R (A is a super key)

B→C, B↛A (B is not a super key)

Not Valid BCNF

```
R(A,B,C)
F = {A→B, B→C}
```

BOYCE-CODD NORMAL FORM (EX.2)

Is R_1 and R_2 in BCNF?

Step #1 – Test R_1

$A \rightarrow B$, $A \rightarrow R_1$ (A is a super key)

Step #2 – Test R_2

$B \rightarrow C$, $B \rightarrow R_2$ (B is a super key)

$R_1(A,B)$ $R_2(B,C)$

$F = \{A \rightarrow B,\ B \rightarrow C\}$

✓ Valid BCNF

BOYCE-CODD NORMAL FORM

Given a schema R and a set of FDs F, we can always decompose R into $\{R_1, \ldots, R_n\}$ such that

→ $\{R_1, \ldots, R_n\}$ are in BCNF

→ The decompositions are lossless.

But some BCNF decompositions might lose dependencies.

BCNF DECOMPOSITION ALGORITHM

Given a relation **R** and a FD set **F**:

Step #1 – Compute **F+**

Step #2 – `Result ← {R}`

Step #3 – While $R_i \in$ `Result` not in BCNF, do:

→ **(a)** Choose **(X→Y) ∈ F+** such that **(X→Y)** is covered by R_i and $X \nrightarrow R_i$

→ **(b)** Decompose R_i on **(X→Y)**:

```
Ri,1 ← X ∪ Y
Ri,2 ← Ri - Y
Result ← (Result - {Ri}) ∪ {Ri,1, Ri,2}
```

← $R_{i,1}$ includes Y

← $R_{i,2}$ does not include Y

BOYCE-CODD NORMAL FORM (EX.3)

Step #1 – Compute Closure

→ F+ ← { ssn→name,
ssn→city,
ssn→name,city }

R(name,ssn,phone,city)
F = {ssn→name,city}

name	ssn	phone	city
Andy	123-45-6789	555-555-5555	Pittsburgh
Andy	123-45-6789	666-666-6666	Pittsburgh
Lil' Fame	987-65-4321	777-777-7777	Brooklyn
Lil' Fame	987-65-4321	888-888-8888	Brooklyn

BOYCE-CODD NORMAL FORM (EX.3)

Step #3 – R is not in BCNF

- → **3(a):** We choose ssn→name,city as the FD to split on because ssn does not get us the phone (i.e., it is not the super key).
- → **3(b):** Split R based on ssn→name,city such that R_1=(name,ssn,city) and R_2=(ssn,phone)

```
R(name,ssn,phone,city)
F = {ssn→name,city}
```

name	ssn	phone	city
Andy	123-45-6789	555-555-5555	Pittsburgh
Andy	123-45-6789	666-666-6666	Pittsburgh
Lil' Fame	987-65-4321	777-777-7777	Brooklyn
Lil' Fame	987-65-4321	888-888-8888	Brooklyn

BOYCE-CODD NORMAL FORM (EX.3)

Step #3: R is not in BCNF

→ **3(c):** The resulting schema is now $R=\{R_1, R_2\}$

R_1(name, <u>ssn</u>, city)
R_2(<u>ssn</u>, <u>phone</u>)
F = {ssn→name, city}

name	ssn	city
Andy	123-45-6789	Pittsburgh
Lil' Fame	987-65-4321	Brooklyn

ssn	phone#
123-45-6789	555-555-5555
123-45-6789	666-666-6666
987-65-4321	777-777-7777
987-65-4321	888-888-8888

BOYCE-CODD NORMAL FORM (EX.3)

Step #3: Check whether $\{R_1, R_2\}$ are not in BCNF

→ Lossless?

→ Anomalies?

R_1(name,ssn,city)

R_2(ssn,phone)

F = {ssn→name,city}

name	ssn	city
Andy	123-45-6789	Pittsburgh
Lil' Fame	987-65-4321	Brooklyn

ssn	phone#
123-45-6789	555-555-5555
123-45-6789	666-666-6666
987-65-4321	777-777-7777
987-65-4321	888-888-8888

✓ Valid BCNF

A PROBLEM WITH BCNF

```
R(item,comp,category)
F = {item→comp, comp,category→item}
```

Super Key: `(item,category)`

A PROBLEM WITH BCNF

R_1(item,comp) R_2(item,category)
F = {item→comp, comp,category→item}

item	comp
Basketball	Pavlo Inc.
Baseball Bat	Pavlo Inc.

item	category
Basketball	Sports Equipment
Baseball Bat	Sports Equipment

We keep item→comp but we lose comp,category→item

At this point we don't have any problems:
→ We're in BCNF and all local FDs are satisfied.

A PROBLEM WITH BCNF

R_1(item,comp) R_2(item,category)

F = {item→comp, comp,category→item}

item	comp
Basketball	Pavlo Inc.
Baseball Bat	Pavlo Inc.

⋈

item	category
Basketball	Sports Equipment
Baseball Bat	Sports Equipment

item	comp	category
Basketball	Pavlo Inc.	Sports Equipment
Baseball Bat	Pavlo Inc.	Sports Equipment

Violates (comp,product→item)

A PROBLEM WITH BCNF

We started with a relation R and its dependency set FD.

We decomposed R into BCNF relations $\{R_1, \ldots, R_n\}$ with their own $\{FD_1, \ldots, FD_n\}$.

We can reconstruct R from $\{R_1, \ldots, R_n\}$.

But we cannot reconstruct FD from $\{FD_1, \ldots, FD_n\}$.

THIRD NORMAL FORM

3NF preserves dependencies but may have some anomalies.

A relation **R** with FD set **F** is in 3NF if for every **X→Y** in **F+**:

- → **X→Y** is trivial, or
- → **X** is a super key, or
- → **Y** is part of a candidate key

3NF DECOMPOSITION ALGORITHM

Given a relation **R** and a FD set **F**:

Step #1: Compute **Fc**

Step #2: **Result ← ∅**

Step #3: For **(X→Y)** ∈ **Fc**, add a relation $R_i(X,Y)$ to **Result**

Step #4: If **Result** is not lossless, add a relation with an appropriate key.

3NF EXAMPLE

Step #1: Compute canonical cover

→ Fc ← {A→B, B→C}

R(A,B,C)

F = {A→B, B→C}

A	B	C
A1	B_A1	C_B_A1
A2	B_A2	C_B_A2
A3	B_A3	C_B_A3
A2	B_A2	C_B_A2

3NF EXAMPLE

Step #3: Split R based on its FDs

→ R_1(A,B) because A→B

→ R_2(B,C) because B→C

R(A,B,C)

F = {A→B, B→C}

A	B	C
A1	B_A1	C_B_A1
A2	B_A2	C_B_A2
A3	B_A3	C_B_A3
A2	B_A2	C_B_A2

3NF EXAMPLE

Step #3: Split R based on its FDs

→ R_1(A,B) because A→B

→ R_2(B,C) because B→C

R_1(A,B) R_2(B,C)

F = {A→B, B→C}

A	B
A1	B_A1
A2	B_A2
A3	B_A2
A2	B_A2

B	C
B_A1	C_B_A1
B_A2	C_B_A2
B_A3	C_B_A3
B_A2	C_B_A2

3NF EXAMPLE

Step #4: Check whether $\{R_1, R_2\}$ is lossless.

Nope!

Add R_3 based on join attribute $A \rightarrow C$

$R_1(A,B)$ $R_2(B,C)$
$F = \{A \rightarrow B,\ B \rightarrow C\}$

A	B
A1	B_A1
A2	B_A2
A3	B_A3
A2	B_A2

⋈

B	C
B_A1	C_B_A1
B_A2	C_B_A2
B_A3	C_B_A3
B_A2	C_B_A2

A	B	C
A1	B_A1	C_B_A1
A2	B_A2	C_B_A2
A2	B_A2	C_B_A2
A3	B_A3	C_B_A3
A2	B_A2	C_B_A2
A2	B_A2	C_B_A2

3NF EXAMPLE

Step #4: Check whether $\{R_1, R_2\}$ is lossless.

Nope!

Add R_3 based on join attribute $A \rightarrow C$

$R_1(A,B)$ $R_2(B,C)$ $R_3(A,C)$
$F = \{A \rightarrow B,\ B \rightarrow C\}$

A	B
A1	B_A1
A2	B_A2
A3	B_A3
A2	B_A2

⋈

B	C
B_A1	C_B_A1
B_A2	C_B_A2
B_A3	C_B_A3
B_A2	C_B_A2

A	C
A1	C_B_A1
A2	C_B_A2
A3	C_B_A3

✓ Valid 3NF

BCNF VS. 3NF

BCNF:

→ No anomalies, but may lose some FDS.
→ In practice, this is what you want.

3NF:

→ Keeps all FDs, but may have some anomalies.
→ You usually get this when you convert an ER diagram to SQL.

CONFESSION

The normal forms is usually not how people design databases.

Instead, people usually think in terms of object-oriented programming.

THE RISE OF NOSQL

Prior to the early 2000s, few people needed a high-performance DBMS.

Key tenants of the NoSQL movement:

→ Joins are slow, so we will denormalize tables.
→ Transactions are slow and we need to be on-line 24/7, so let's drop ACID.

DOCUMENT DATABASES

Document Model = JSON / XML

MongoDB supports basic server-side joins. They instead promote "pre-joined" collections by embedding related documents inside of each other.

mongoDB

RAVENDB

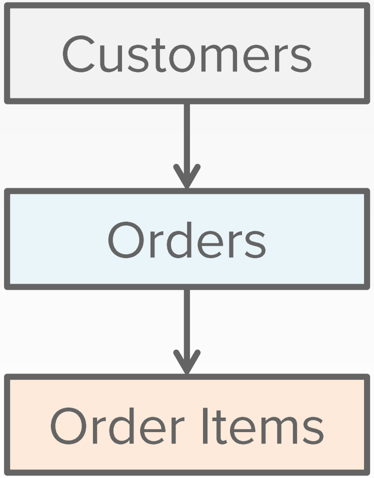
MarkLogic™

Couchbase

RethinkDB

BCNF EXAMPLE

A customer has orders and each order has order items.



$$R_1(\texttt{custId},\texttt{name},\ldots)$$

$$\bowtie$$

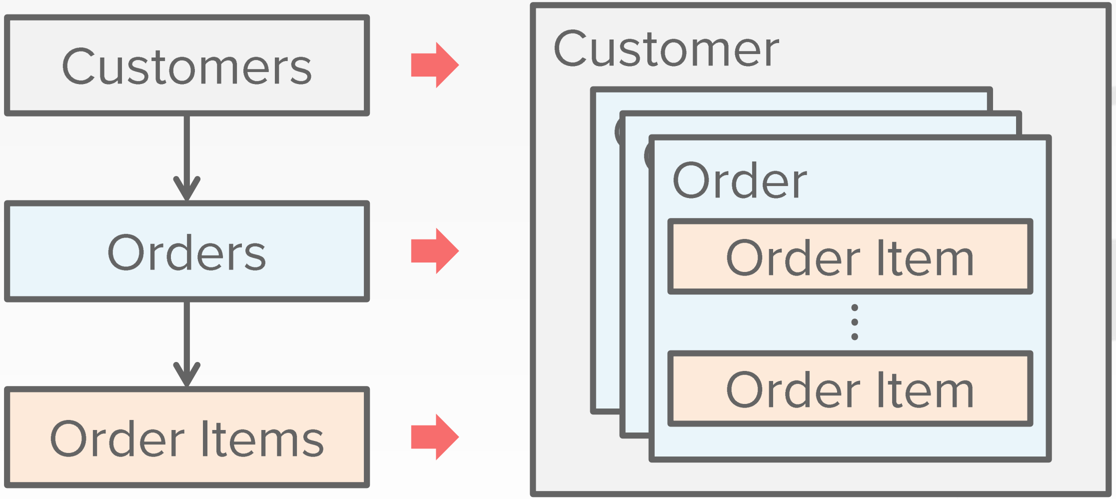
$$R_2(\texttt{orderId},\texttt{custId},\ldots)$$

$$\bowtie$$

$$R_3(\texttt{itemId},\texttt{orderId},\ldots)$$

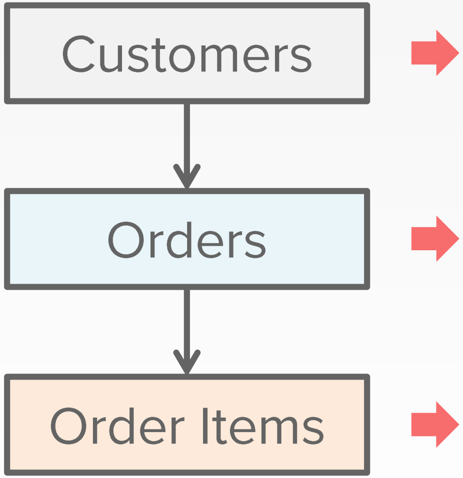
BCNF EXAMPLE

A customer has orders and each order has order items.



BCNF EXAMPLE

A customer has orders and each order has order items.




```
{
  "custId": 1234,
  "custName": "Andy",
  "orders": [
    { "orderId": 9999,
      "orderItems": [
        { "itemId": "XXXX",
          "price":  19.99 },
        { "itemId": "YYYY",
          "price":  29.99 },
      ] }
  ]
}
```

DENORMALIZATION EXAMPLE

No joins is not a by-product of using the document model, but it makes logical denormalization more “natural”.

Violates the separation between a database's logical layer and its physical layer.

PHYSICAL VS. LOGICAL

The relational model also supports "nesting" at the physical storage level.

```
db.customers.find(
  {"orders.orderItems": "XXXX"}
)
```

```
SELECT * FROM customers AS c,
              orders AS o,
              order_items AS oi
WHERE c.custId = o.custId
  AND o.orderId = oi.orderId
  AND oi.itemId = "XXXX"
```

```
{
  "custId": 1234,
  "custName": "Andy",
  "orders": [
    { "orderId": 9999,
      "orderItems": [
        { "itemId": "XXXX",
          "price":  19.99 },
        { "itemId": "YYYY",
          "price":  29.99 },
      ] }
  ]
}
```

PHYSICAL VS. LOGICAL

The relational model also supports "nesting" at the physical storage level.

```
db.customers.find(
  {"orders.orderItems": "XXXX"}
)
```

```
SELECT * FROM customers AS c,
              orders AS o,
              order_items AS oi
WHERE c.custId = o.custId
  AND o.orderId = oi.orderId
  AND oi.itemId = "XXXX"
```

custId	custName	orders
1234	Andy	(nested table below)

custId	orderId	orderItems
1234	9999	(nested table below)

orderId	itemId	price
9999	XXXX	19.99
9999	YYYY	29.99

CARNEGIE MELLON DATABASE GROUP

CONCLUSION

You should know about normal forms. They exist.

There is no magic formula to determine what is the right amount of normalization for an application.

NEXT CLASS

Database Storage Management