1 INTRODUCTION

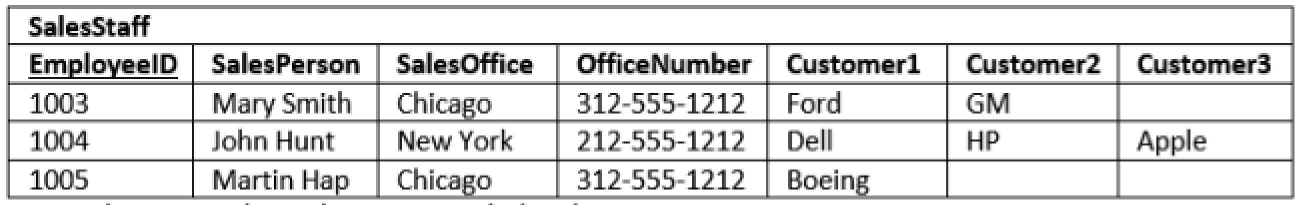
Database normalization is process used to organize a database into tables and columns. The idea is that a table should be about a specific topic and that only those columns which support that topic are included. For example, a spreadsheet containing information about sales people and customers serves several purposes:

- Identify sales people in your organization
- List all customers your company calls upon to sell product
- Identify which sales people call on specific customers.

By limiting a table to one purpose you reduce the number of duplicate data that is contained within your database, which helps eliminate some issues stemming from database modifications. To assist in achieving these objectives, some rules for database table organization have been developed. The stages of organization are called normal forms; there are three normal forms most databases adhere to using. As tables satisfy each successive normalization form, they become less prone to database modification anomalies and more focused toward a sole purpose or topic. Before we move on be sure you understand the definition of a database table.

2 REASONS FOR NORMALIZATION

There are three main reasons to normalize a database. The first is to minimize duplicate data, the second is to minimize or avoid data modification issues, and the third is to simplify queries. As we go through the various states of normalization we'll discuss how each form addresses these issues, but to start, let's look at some data which hasn't been normalized and discuss some potential pitfalls. Once these are understood, I think you'll better appreciate the reason to normalize the data. Consider the following table:



SalesStaff

<u>EmployeeID</u>	SalesPerson	SalesOffice	OfficeNumber	Customer1	Customer2	Customer3
1003	Mary Smith	Chicago	312-555-1212	Ford	GM	
1004	John Hunt	New York	212-555-1212	Dell	HP	Apple
1005	Martin Hap	Chicago	312-555-1212	Boeing		

Note: The primary key columns are underlined

The first thing to notice is this table serves many purposes including:

- Identifying the organization's salespeople
- Listing the sales offices and phone numbers
- Associating a salesperson with a sales office
- Showing each salesperson's customers

As a DBA, this raises a red flag. In general, you would like to see tables that have one purpose. Having the table serve many purposes introduces many of the challenges; namely, data duplication, data update issues, and increased effort to query data.

2.1 Data Duplication and Modification Anomalies

Notice that for each SalesPerson we have listed both the SalesOffice and OfficeNumber. This information is duplicated for each SalesPerson. Duplicated information presents two problems:

It increases storage and decrease performance.

It becomes more difficult to maintain data changes.

For example

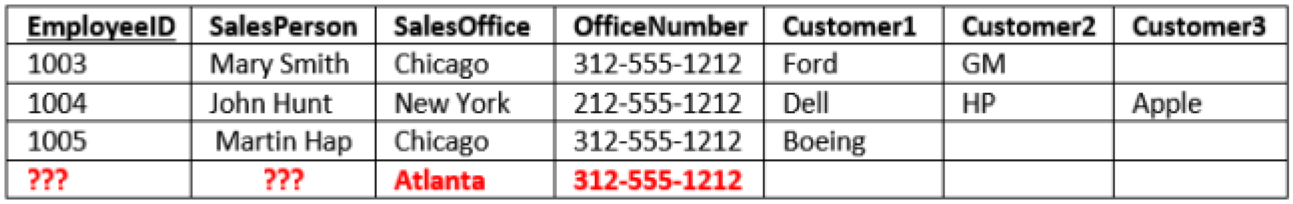
Consider if we move the Chicago office to Evanston, IL. To properly reflect this in our table, we need to update the entries for all the SalesPersons currently in Chicago. Our table is a small example, but you can see if it were larger, that potentially this could involve hundreds of updates.

Also, consider what would happen if John Hunt quits. If we remove his entry, then we lose the information for New York.

These situations are modification anomalies. There are three modification anomalies that can occur:

2.2 Insert Anomaly

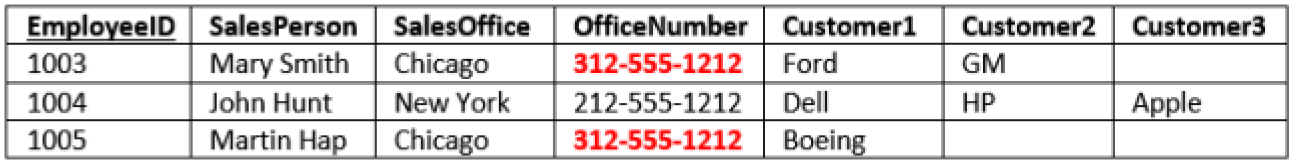
There are facts we cannot record until we know information for the entire row. In our example, we cannot record a new sales office until we also know the sales person. Why? In order to create the record, we need provide a primary key. In our case this is the EmployeeID.



EmployeeID	SalesPerson	SalesOffice	OfficeNumber	Customer1	Customer2	Customer3
1003	Mary Smith	Chicago	312-555-1212	Ford	GM	
1004	John Hunt	New York	212-555-1212	Dell	HP	Apple
1005	Martin Hap	Chicago	312-555-1212	Boeing		
???	**???**	**Atlanta**	**312-555-1212**			

2.3 Update Anomaly

The same information is recorded in multiple rows. For instance, if the office number changes, then there are multiple updates that need to be made. If these updates are not successfully completed across all rows, then an inconsistency occurs.



EmployeeID	SalesPerson	SalesOffice	OfficeNumber	Customer1	Customer2	Customer3
1003	Mary Smith	Chicago	**312-555-1212**	Ford	GM	
1004	John Hunt	New York	212-555-1212	Dell	HP	Apple
1005	Martin Hap	Chicago	**312-555-1212**	Boeing		

2.4 Deletion Anomaly

Deletion of a row can cause more than one set of facts to be removed. For instance, if John Hunt retires, then deleting that row cause use to lose information about the New York office.

EmployeeID	SalesPerson	SalesOffice	OfficeNumber	Customer1	Customer2	Customer3
1003	Mary Smith	Chicago	312-555-1212	Ford	GM	
~~1004~~	~~John Hunt~~	~~New York~~	~~212-555-1212~~	~~Dell~~	~~HP~~	~~Apple~~
1005	Martin Hap	Chicago	312-555-1212	Boeing		

2.5 Search and Sort Issues

The last reason we'll consider is making it easier to search and sort your data. In the SalesStaff table if you want to search for a specific customer such as Ford, you would have to write a query like

```
SELECT SalesOffice
FROM SalesStaff
WHERE Customer1 = 'Ford' OR
      Customer2 = 'Ford' OR
      Customer3 = 'Ford'
```

Clearly if the customer were somehow in one column our query would be simpler. Also, consider if you want to run a query and sort by customer. The way the table is currently defined, this isn't possible, unless you use three separate queries with a UNION. These anomalies can be eliminated or reduced by properly separating the data into different tables, to house the data in tables which serve a single purpose. The process to do this is called normalization, and the various stages you can achieve are called the normal forms.

3 What is Normalization?

- Normalization is a *process* in which we systematically examine relations for *anomalies* and, when detected, remove those anomalies by splitting up the relation into two new, related, relations.
- Normalization is an important part of the database development process: Often during normalization, the database designers get their first real look into how the data are going to interact in the database.
- Finding problems with the database structure at this stage is strongly preferred to finding problems further along in the development process because at this point it is fairly easy to cycle back to the conceptual model (Entity Relationship model) and make changes.
- Normalization can also be thought of as a trade-off between data redundancy and performance. Normalizing a relation reduces data redundancy but introduces the need for joins when all the data is required by an application such as a report query.

Recall, the Relational Model consists of the elements: relations, which are made up of attributes.

- A relation is a set of attributes with values for each attribute such that:
    1. Each attribute (column) value must be a single value only.
    2. All values for a given attribute (column) must be of the same data type.

3. Each attribute (column) name must be unique.
4. The order of attributes (columns) is insignificant
5. No two tuples (rows) in a relation can be identical.
6. The order of the tuples (rows) is insignificant.

- From our discussion of E-R Modeling, we know that an Entity typically corresponds to a relation and that the Entity's attributes become attributes of the relation.
- We also discussed how, depending on the relationships between entities, copies of attributes (the *identifiers*) were placed in related relations as **foreign keys**.

4 FUNCTIONAL DEPENDENCIES

- A *Functional Dependency* describes a relationship between *attributes* within a single relation.
- An attribute is *functionally dependent* on another if we can use the value of one attribute to determine the value of another.
- Example: Employee_Name is functionally dependent on Social_Security_Number because Social_Security_Number can be used to uniquely determine the value of Employee_Name.
- We use the arrow symbol → to indicate a functional dependency.

X → Y is read *X functionally determines Y*

Here are a few more examples:

```
Student_ID → Student_Major
Student_ID, CourseNumber, Semester → Grade
Course_Number, Section → Professor, Classroom, NumberOfStudents
SKU → Compact_Disk_Title, Artist
CarModel, Options, TaxRate → Car_Price
```

- The attributes listed on the left-hand side of the → are called *determinants*.

One can read A → B as, "A determines B". Or more specifically: Given a value for A, we can uniquely determine one value for B.

5 KEYS AND UNIQUENESS

- **Key**: One or more attributes that uniquely identify a tuple (row) in a relation.
- The selection of keys will depend on the application being considered.
- In most cases the key for a relation will already be specified during the conversion from the E-R model to a set of relations.
- Users can also offer some guidance as to what would make an appropriate key.
- Recall that no two relations should have the same values, thus a candidate key would consist of all the attributes in a relation.
- A key functionally determines a tuple (row). So, one functional dependency that can always be written is:
- `The Key → All other attributes`

- Not all *determinants* are *keys*.

6 Normalization Process

- Relations can fall into one or more categories (or classes) called *Normal Forms*
- **Normal Form**: A class of relations free from a certain set of modification anomalies.
- Normal forms are given names such as:
    - First normal form (1NF)
    - Second normal form (2NF)
    - Third normal form (3NF)
    - Boyce-Codd normal form (BCNF)
    - Fourth normal form (4NF)
    - Fifth normal form (5NF)
    - Domain-Key normal form (DK/NF)
- These forms are cumulative. A relation in Third normal form is also in 2NF and 1NF.
- The *Normalization Process* for a given relation consists of:
    1. Specify the *Key* of the relation
    2. Specify the *functional dependencies* of the relation.
       Sample data (tuples) for the relation can assist with this step.
    3. Apply the definition of each normal form (starting with 1NF).
    4. If a relation fails to meet the definition of a normal form, change the relation (most often by splitting the relation into two new relations) until it meets the definition.
    5. Re-test the modified/new relations to ensure they meet the definitions of each normal form.

7 First Normal Form (1NF)

- A relation is in first normal form if it meets the definition of a relation:
    - Each attribute (column) value must be a single value only.
    - All values for a given attribute (column) must be of the same type.
    - Each attribute (column) name must be unique.
    - The order of attributes (columns) is insignificant
    - No two tuples (rows) in a relation can be identical.
    - The order of the tuples (rows) is insignificant.
- If you have a *key* defined for the relation, then you can meet the *unique row* requirement.
- Example relation in 1NF (note that key attributes are underlined):
    - STOCKS (Company, <u>Symbol</u>, Headquarters, <u>Date</u>, Close_Price)

Company	**<u>Symbol</u>**	**Headquarters**	**<u>Date</u>**	**Close Price**
Microsoft	MSFT	Redmond, WA	09/07/2013	23.96
Microsoft	MSFT	Redmond, WA	09/08/2013	23.93
Microsoft	MSFT	Redmond, WA	09/09/2013	24.01
Oracle	ORCL	Redwood Shores, CA	09/07/2013	24.27
Oracle	ORCL	Redwood Shores, CA	09/08/2013	24.14

Oracle	ORCL	Redwood Shores, CA	09/09/2013	24.33

Note that the key (which consists of the Symbol and the Date) can uniquely determine the Company, headquarters and Close Price of the stock. Here was assume that Symbol must be unique but Company, Headquarters, Date and Price are not unique

8 Second Normal Form (2NF)

- A relation is in second normal form (2NF) if all its non-key attributes are dependent on all the *key*.
- Another way to say this: A relation is in second normal form if it is free from partial-key dependencies
- Relations that have a single attribute for a key are automatically in 2NF.
- This is one reason why we often use artificial identifiers (non-composite keys) as keys.
- In the example below, Close Price is dependent on Company, Date
- The following example relation *is not* in 2NF:
- `STOCKS (Company, Symbol, Headquarters, Date, Close_Price)`

Company	**Symbol**	**Headquarters**	**Date**	**Close Price**
Microsoft	MSFT	Redmond, WA	09/07/2013	23.96
Microsoft	MSFT	Redmond, WA	09/08/2013	23.93
Microsoft	MSFT	Redmond, WA	09/09/2013	24.01
Oracle	ORCL	Redwood Shores, CA	09/07/2013	24.27
Oracle	ORCL	Redwood Shores, CA	09/08/2013	24.14
Oracle	ORCL	Redwood Shores, CA	09/09/2013	24.33

- To start the normalization process, list the functional dependencies (FD):
  - `FD1: Symbol, Date → Company, Headquarters, Close Price`
  - `FD2: Symbol → Company, Headquarters`
- Consider that Symbol, Date → Close Price.

So, we might use Symbol, Date as our key.

- However, we also see that: Symbol → Headquarters
- This violates the rule for 2NF in that a *part of our key* determines a non-key attribute.
- Another name for this is a *Partial key dependency*. Symbol is only a "part" of the key and it determines a non-key attribute.
- Also, consider the insertion and deletion anomalies.
- **One Solution:** Split this up into two new relations:
  - `COMPANY (Company, Symbol, Headquarters)`
  - `STOCK_PRICES (Symbol, Date, Close_Price)`
- At this point we have two new relations in our relational model. The original "STOCKS" relation we started with is removed from the model.
- Sample data and functional dependencies for the two new relations:
- COMPANY Relation:

Company	<u>Symbol</u>	Headquarters
Microsoft	MSFT	Redmond, WA
Oracle	ORCL	Redwood Shores, CA

`FD1: Symbol → Company, Headquarters`

STOCK_PRICES relation:

<u>Symbol</u>	<u>Date</u>	Close Price
MSFT	09/07/2013	23.96
MSFT	09/08/2013	23.93
MSFT	09/09/2013	24.01
ORCL	09/07/2013	24.27
ORCL	09/08/2013	24.14
ORCL	09/09/2013	24.33

`FD1: Symbol, Date → Close Price`

- In checking these new relations, we can confirm that they meet the definition of 1NF (each one has well defined unique keys) and 2NF (no partial key dependencies).

9 Third Normal Form (3NF)

- A relation is in third normal form (3NF) if it is in second normal form and it contains no *transitive dependencies*.
- Consider relation R containing attributes A, B and C. R(A, B, C)
- If A → B and B → C then A → C
- **Transitive Dependency**: Three attributes with the above dependencies.
- Example: At CUNY:
  - `Course_Code → Course_Number, Section`
  - `Course_Number, Section → Classroom, Professor`

- Consider one of the new relations we created in the STOCKS example for 2nd normal form:

Company	<u>Symbol</u>	Headquarters
Microsoft	MSFT	Redmond, WA
Oracle	ORCL	Redwood Shores, CA

- The functional dependencies we can see are:
  - `FD1: Symbol →   Company`
  - `FD2: Company → Headquarters`
  - `so therefore:`
  - `Symbol → Headquarters`

- This is a transitive dependency.
- What happens if we remove Oracle?

We lose information about 2 different facts.

- The solution again is to split this relation up into two new relations:
  - `STOCK_SYMBOLS(Company, Symbol)`
  - `COMPANY_HEADQUARTERS(Company, Headquarters)`
- This gives us the following sample data and FD for the new relations

Company	Symbol
Microsoft	MSFT
Oracle	ORCL

```
FD1: Symbol → Company
```

Company	**Headquarters**
Microsoft	Redmond, WA
Oracle	Redwood Shores, CA

```
FD1:  Company →  Headquarters
```

- Again, each of these new relations should be checked to ensure they meet the definition of 1NF, 2NF and now 3NF.

10 Boyce-Codd Normal Form (BCNF)

- A relation is in BCNF if every determinant is a candidate key.
- Recall that not all determinants are keys.
- Those determinants that are keys we initially call *candidate keys*.
- Eventually, we select a single candidate key to be *the key* for the relation.
- Consider the following example:
  - Funds consist of one or more Investment Types.
  - Funds are managed by one or more Managers
  - Investment Types can have one more Managers
  - Managers only manage one type of investment.
- Relation: FUNDS (FundID, InvestmentType, Manager)

FundID	**InvestmentType**	**Manager**
99	Common Stock	Smith
99	Municipal Bonds	Jones
33	Common Stock	Green
22	Growth Stocks	Brown
11	Common Stock	Smith

```
FD1:  FundID, InvestmentType → Manager
FD2:  FundID, Manager        → InvestmentType
FD3:  Manager                → InvestmentType
```

- In this case, the combination FundID and InvestmentType form a *candidate key* because we can use FundID, InvestmentType to uniquely identify a tuple in the relation.

- Similarly, the combination FundID and Manager also form a *candidate key* because we can use FundID, Manager to uniquely identify a tuple.
- Manager by itself is not a candidate key because we cannot use Manager alone to uniquely identify a tuple in the relation.
- Is this relation FUNDS(FundID, InvestmentType, Manager) in 1NF, 2NF or 3NF?

Given we pick FundID, InvestmentType as the *Primary Key:* 1NF for sure.

2NF because all the non-key attributes (Manager) is dependent on all the key.

3NF because there are no transitive dependencies.

- However, consider what happens if we delete the tuple with FundID 22. We lose the fact that Brown manages the InvestmentType “Growth Stocks.”
- Therefore, while FUNDS relation is in 1NF, 2NF and 3NF, it is in BCNF because not all determinants (Manager in FD3) are candidate keys.
- The following are steps to normalize a relation into BCNF:
    1 List all the determinants.
    2 See if each determinant can act as a key (candidate keys).
    3 For any determinant that is *not* a candidate key, create a new relation from the functional dependency. Retain the determinant in the original relation.
- For our example:

FUNDS (FundID, InvestmentType, Manager)

1. The determinants are:

    - `FundID, InvestmentType`
    - `FundID, Manager`
    - `Manager`

2. Which determinants can act as keys?

    - `FundID, InvestmentType` *YES*
    - `FundID, Manager` *YES*
    - `Manager` *NO*

3. Create a new relation from the functional dependency:

MANAGERS(<u>Manager</u>, InvestmentType)

FUND_MANAGERS(<u>FundID</u>, <u>Manager</u>)

In this last step, we have retained the determinant “Manager” in the original relation MANAGERS.

- Each of the new relations should be checked to ensure they meet the definitions of 1NF, 2NF, 3NF and BCNF

11 Normalization Exercises

Here are a bunch of fun normalization exercises you can try! Answers are on the last page.

1. Choose a key and write the dependencies for the following GRADES relation:

```
GRADES (Student_ID, CourseNumber, SemesterNumber, Grade)
```

2. Choose a key and write the dependencies for the PO_LINE_ITEMS relation:
Keep in mind ItemNum is an integer that counts up from 1 for each purchase order.

```
PO_LINE_ITEMS (PO_Number, ItemNum, PartNum, Description, Price, Qty)
```

3. What normal form is the above LINE_ITEMS relation in (before doing any normalizing)?

4. What normal form is the following relation in (key is underlined):

STORE_ITEM (<u>SKU</u>, <u>PromotionID</u>, Vendor, Style, Price)

```
FD1:   SKU, PromotionID → Vendor, Style, Price
FD2:   SKU → Vendor, Style
```

5. Normalize the above STORE_ITEM relation into the next higher normal form:

6. Choose a key and write the dependencies for the following SOFTWARE relation (assume all the vendor's products have the same warranty):

```
SOFTWARE (SoftwareVendor, Product, Release, SystemReq, Price, Warranty)

FD1: SoftwareVendor, Product, Release → SystemReq, Price, Warranty
```

7. Consider the following relation:
ArrivalDate is the date the ship arrives in the given Port

```
   Shipping (ShipName, ShipType, VoyageID, Cargo, Port, ArrivalDate)

   FD1:    ShipName              → ShipType
   FD2:    VoyageID              → ShipName, Cargo
   FD3:    ShipName, ArrivalDate → VoyageID, Port
```

(a) Identify the candidate keys.
(b) Normalize to 2NF
(c) Normalize to 3NF
(d) Normalize to BCNF

8. Given the following relation and example data:

PartNumber	Description	Supplier	SupplierAddress	Price

10010	20 TB Disk	Seagate	Cuppertino, CA	$100
10010	20 TB Disk	IBM	Armonk, NY	$90
10220	256 GB RAM card	Kensington	San Mateo, CA	$220
10220	256 GB RAM card	IBM	Armonk, NY	$290
10220	256 GB RAM card	Sun Microsystems	Palo Alto, CA	$310
10440	21" LCD Monitor	IBM	Armonk, NY	$2,100

List the functional dependencies and Normalize this relation into BCNF .

12 Normalization Exercises – Answers

1. Key is Student_ID, CourseNumber, SemesterNumber
   Functional Dependency is: Student_ID, CourseNumber, SemesterNumber → Grade
   Therefore GRADES is in 4NF
2. Key can be: PO_Number, ItemNum
   Functional Dependencies are:
   FD1: PO_Number, ItemNum → PartNum, Description, Price, Qty
   FD2: PartNum → Description, Price
3. First off, LINE_ITEMS could not be in BCNF because not all determinants are keys
   Next, it could not be in 3NF because there is a transitive dependency: PO_Number, ItemNum → PartNum and PartNum → Description
   Therefore, it must be in 2NF. We can check this is true because the key of PO_Number, ItemNum determines all the non-key attributes however, PO_Number by itself and ItemNum by itself cannot determine any other attributes.
4. STORE_ITEM is in 1NF (non-key attribute (vendor) is dependent on only part of the key)
5. STORE_ITEM (SKU, PromotionID, Price) VENDOR_ITEM (SKU, Vendor, Style)
6. Key is SoftwareVendor, Product, Release SoftwareVendor, Product, Release → SystemReq, Price, Warranty SoftwareVendor → Warranty SOFTWARE is in 1NF
7. (a) Key is ShipName and ArrivalDate
   (b) Starting with the initial relation:

```
Shipping (ShipName, ShipType, VoyageID, Cargo, Port, ArrivalDate)
```

We have a partial key dependency as ShipName alone can determine ShipType. So, we normalize to:

```
SHIPS( ShipName, ShipType)
FD1: ShipName → ShipType

VOYAGES (ShipName, VoyageID, Cargo, Port, ArrivalDate)
FD1: ShipName, ArrivalDate → VoyageID, Port
FD2: VoyageID → ShipName, Cargo
```

(c) As above, VOYAGES has a transitive dependency:

```
ShipName, ArrivalDate → VoyageID
VoyageId → Cargo
```

So, normalize VOYAGES into two new relations:

SHIPPORTS (ShipName, VoyageID, Port, ArrivalDate)

FD1: ShipName, ArrivalDate → VoyageID, Port
FD2: VoyageID → ShipName

CARGO (VoyageID, Cargo)
FD1: VoyageId → Cargo

SHIPS(ShipName, ShipType)
FD1: ShipName → ShipType

(d) SHIPPORTS is not in BCNF since it has VoyageID as a determinant but VoyageID is not a candidate key.

SHIPDATES (ShipName, Port, ArrivalDate)
FD1: ShipName, Date → Port

SHIPVOYAGE (VoyageID, ShipName)
FD1: VoyageID → ShipName

CARGO (VoyageID, Cargo)
FD1: VoyageId → Cargo

SHIPS(ShipName, ShipType)
FD1: ShipName → ShipType

8. Functional dependencies are:

- FD1: PartNumber → Description
- FD2: PartNumber, Supplier → Price
- FD3: Supplier → SupplierAddress
- 1NF: Suggest PartNumber, Supplier as the key so we are in 1NF
- 2NF: We have a partial key dependency in that Supplier → SupplierAddress so normalize:

- R1(PartNumber, Description, Supplier, Price)
- FD1: PartNumber → Description
- FD2: PartNumber, Supplier → Price

- R2(Supplier, SupplierAddress)
- FD1: Supplier → SupplierAddress

But we still have a problem with R1 so normalize again:

- R3 (PartNumber, Supplier, Price)
- FD1: PartNumber, Supplier → Price

- R4 (PartNumber, Description PartNumber → Description R2 (Supplier, SupplierAddress)
- FD1: Supplier → SupplierAddress