11. Normalization

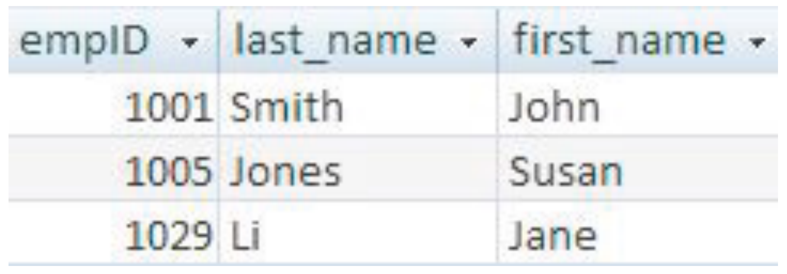
- We saw how to represent the main parts of the data model in a relational database.
  - Each class is represented by a table.
  - Each attribute is represented by a field with a datatype and possible constraints.
  - Each object becomes a row in a table.
  - For each table, we determine a primary key, which is a field(s) that uniquely identifies each row.
  - We use the primary key field(s) to represent relationships between classes by way of foreign keys.
- There may be some classes in our model (or tables in our database) that might still cause us problems.
- Normalization is a formal way of checking the fields to ensure they are in the right table or to see if perhaps we might need restructured or additional tables to help keep our data accurate.
- We will first look at why it is critical that all the attributes are in the right table and how normalization helps us make sure they are.

Today's Lecture

- Update Anomalies
- Functional Dependencies
- Normal Forms
- Exercises
- Summary

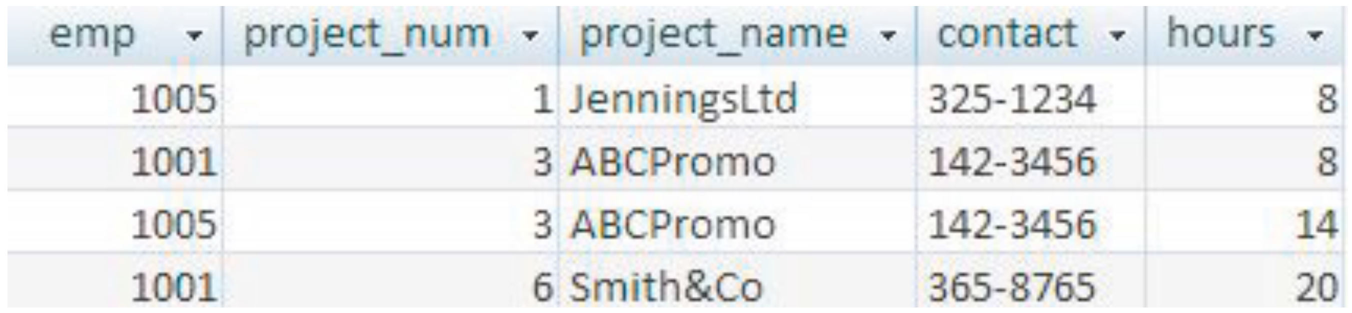
Update Anomalies

- Let's have a look at a simple example where having the attributes in the wrong table can cause a number of problems in maintaining data.



empID	last_name	first_name
1001	Smith	John
1005	Jones	Susan
1029	Li	Jane

Employee



emp	project_num	project_name	contact	hours
1005	1	JenningsLtd	325-1234	8
1001	3	ABCPromo	142-3456	8
1005	3	ABCPromo	142-3456	14
1001	6	Smith&Co	365-8765	20

Assignment

- There are two tables about employees and some small projects to which they have been assigned.
- Can you see a problem lurking in the Assignment table?
- We have repeated information about a project.
- The number, name, and contact can be repeated several times in this table if there is more than one employee working on the project.

Insertion Problems

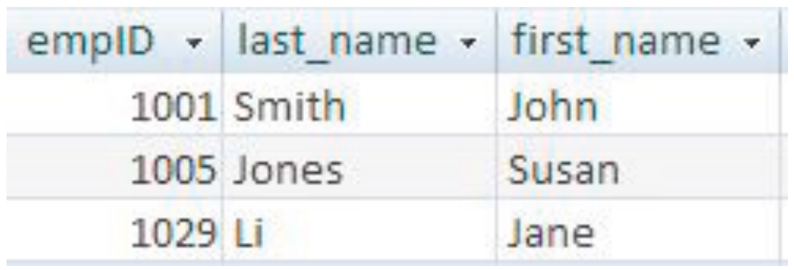
- It is necessary to have a primary key for every table in our database.
- What is a possible primary key for the Assignment table?
- The pair of values for **emp** and **proj_num** is a suitable primary key.
- We have a problem. If we want to keep information about a particular project but there is no employee yet working on it.
- We have no value for **emp**, which is one of the fields making up our primary key.

Deletion Problems

- We will remove that row from the Assignment table.
- What is a possible side effect of deleting this row?
- By deleting information about employee 1001’s involvement in a project, we have inappropriately lost information about the project.

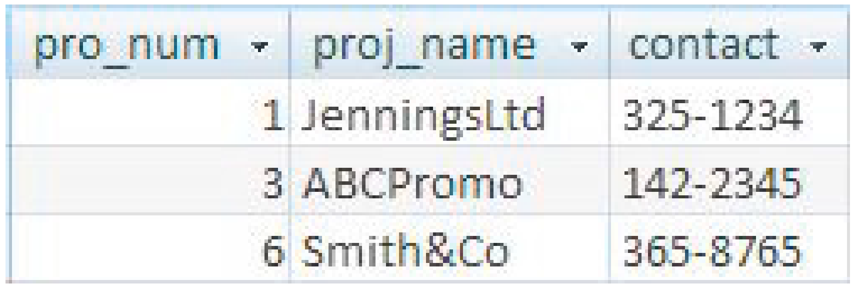
Dealing With Update Anomalies

- We have seen three different updating problems with the Assignment table:
  - possible inconsistent data when repeated information is modified,
  - problems inserting new records because part of the primary key may be empty, and
  - accidental loss of information as a by-product of a deletion.



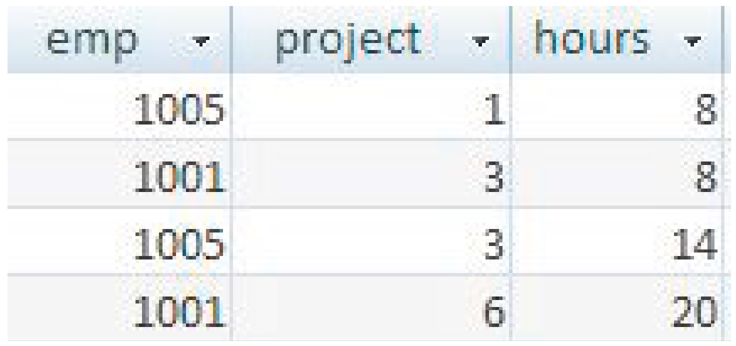
empID	last_name	first_name
1001	Smith	John
1005	Jones	Susan
1029	Li	Jane

Employee



pro_num	proj_name	contact
1	JenningsLtd	325-1234
3	ABCPromo	142-2345
6	Smith&Co	365-8765

Project



emp	project	hours
1005	1	8
1001	3	8
1005	3	14
1001	6	20

Assignment

- How can you be sure you haven't missed anything?
- This is where the formal definition of a normalized table helps.

Functional Dependencies

- Normalization helps us to determine whether our tables are structured in such a way as to avoid the update anomalies.
- Central to the definition of normalization is the idea of a *functional dependency*.
- Functional dependencies are a way of describing the interdependence of attributes or fields in tables.
- With a definition of functional dependencies,
  - we can provide a more formal definition of a primary key,
  - explain what is meant by a normalized table, and
  - discuss the different forms of normalization.

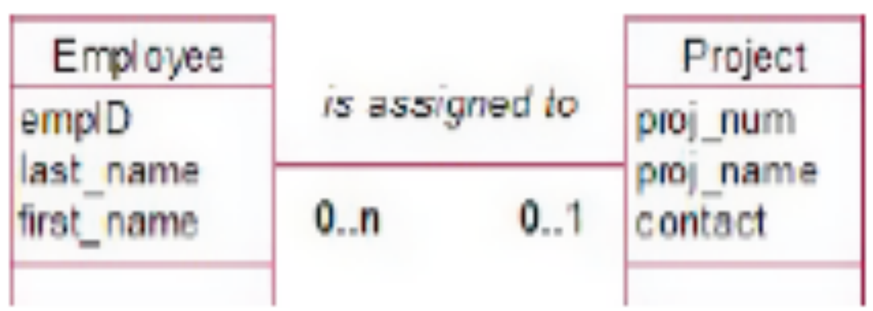
Definition of a Functional Dependency

"If I know the value for this attribute(s), I can uniquely tell you the value of some other attribute(s)."

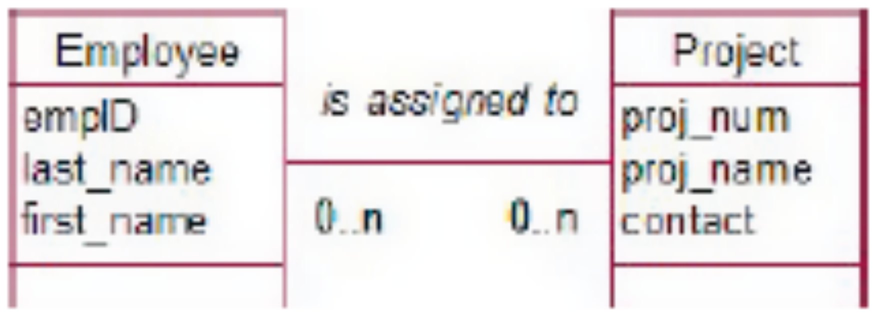
If I know the value of an employee's ID number, I can tell you his last name with certainty.

Employee's ID number ***functionally determines*** *employee's last name.*

`empID → last_name`



a) Employee is assigned to at most one project



b) Employee could be assigned to many projects

Functional Dependencies and Primary Keys

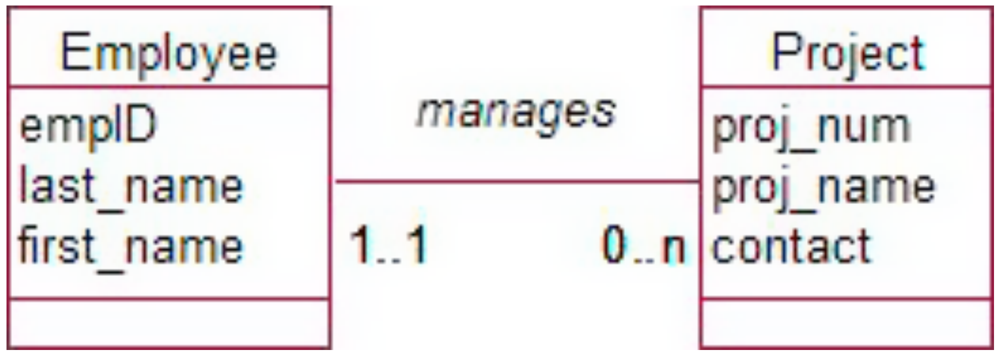
- If we know the values of the key fields of a table, we can find a unique row in the table.
- Once we have that row, then we know the value of all the other fields in that row.
- If I know **`empID`**, I can find a unique row in the Employee table and so be able to determine the **`last_name`** and **`first_name`**. Or, in terms of functional dependencies:

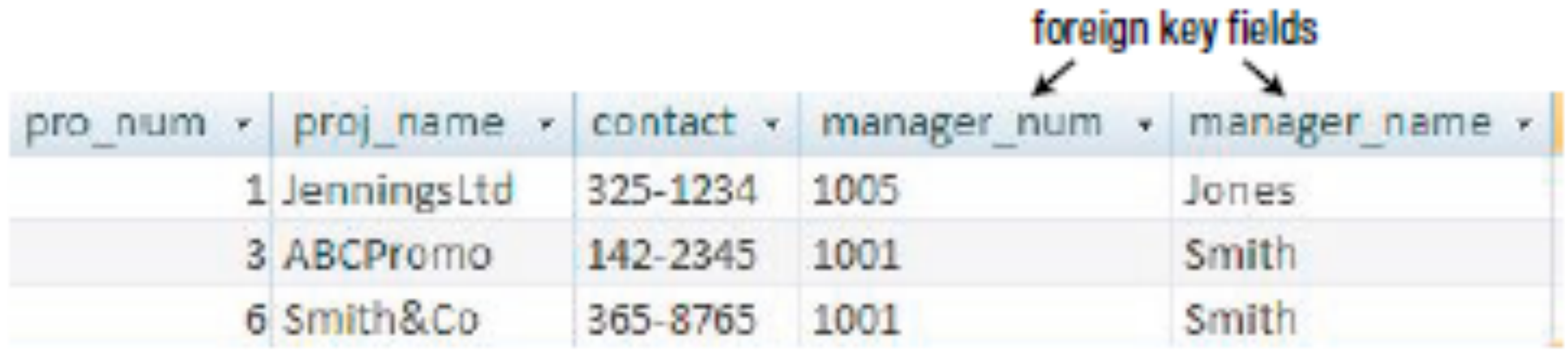
`empID → last_name, first_name`

The key fields functionally determine all the other fields in the table.

- There is a distinction between the term *key* rather than *primary key*.
- Is the pair of attributes (**`empID`**, **`last_name`**) a possible key for our Employee table?
- **`last_name`** is redundant.
- To be considered as a primary key, there must be no unnecessary fields.

A primary key has no subset of the fields that is also a key.




foreign key fields

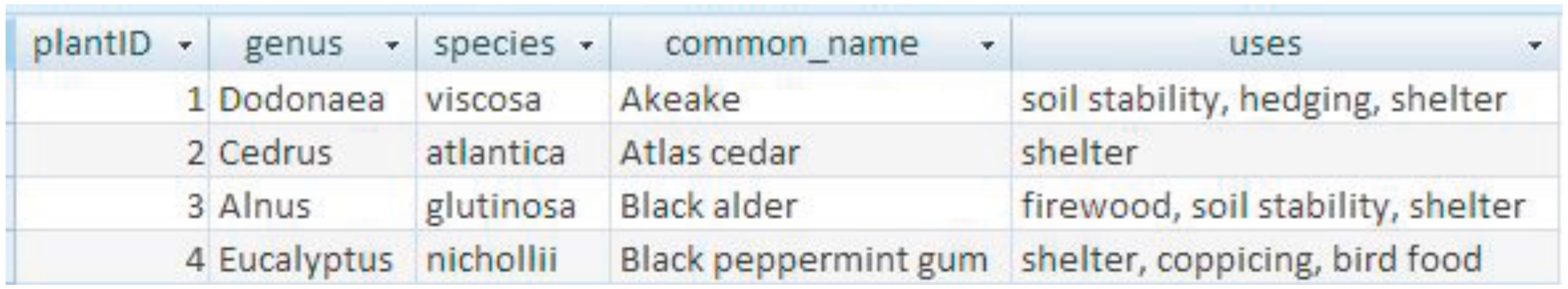
pro_num	proj_name	contact	manager_num	manager_name
1	JenningsLtd	325-1234	1005	Jones
3	ABCPromo	142-2345	1001	Smith
6	Smith&Co	365-8765	1001	Smith

Normal Forms

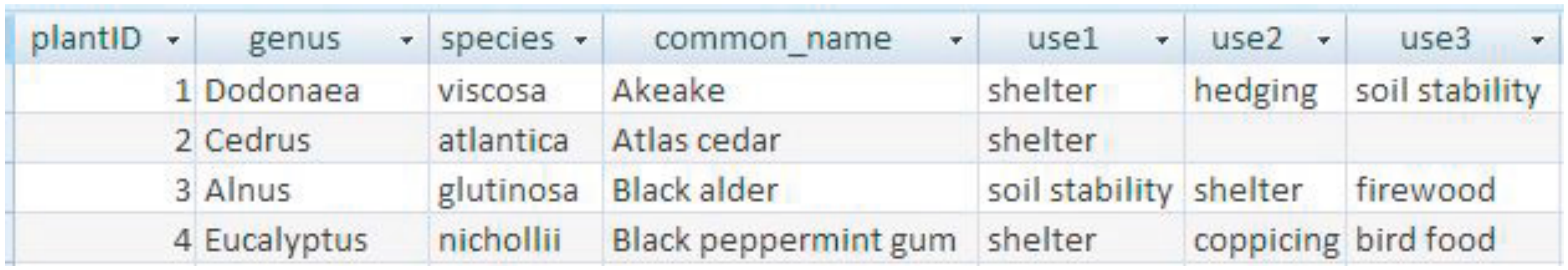
- Tables that are “normalized” will generally avoid the updating problems.
- There are several levels of normalization called *normal forms*, each addressing additional situations where problems may occur.

First Normal Form

- First normal form is the most important, and essentially says that we should not try to cram several pieces of data into a single field.



plantID	genus	species	common_name	uses
1	Dodonaea	viscosa	Akeake	soil stability, hedging, shelter
2	Cedrus	atlantica	Atlas cedar	shelter
3	Alnus	glutinosa	Black alder	firewood, soil stability, shelter
4	Eucalyptus	nichollii	Black peppermint gum	shelter, coppicing, bird food

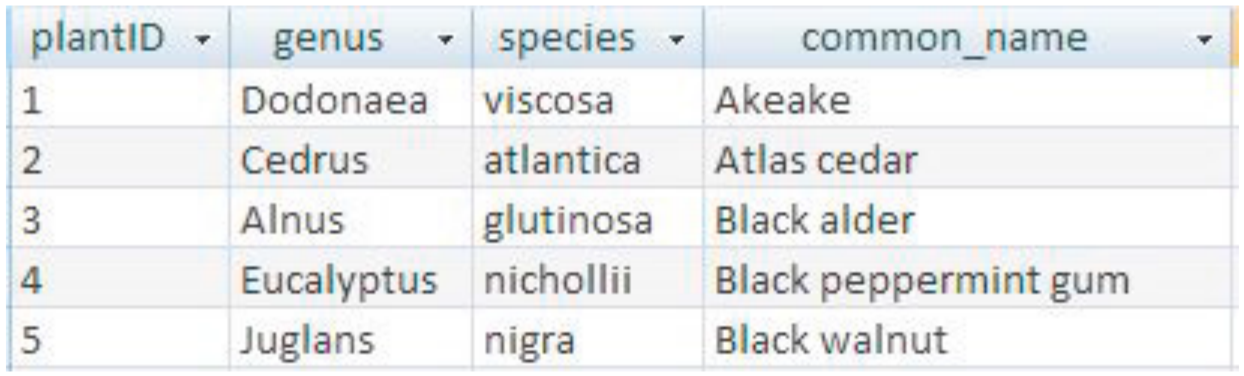


plantID	genus	species	common_name	use1	use2	use3
1	Dodonaea	viscosa	Akeake	shelter	hedging	soil stability
2	Cedrus	atlantica	Atlas cedar	shelter		
3	Alnus	glutinosa	Black alder	soil stability	shelter	firewood
4	Eucalyptus	nichollii	Black peppermint gum	shelter	coppicing	bird food

- `plantID` is a primary key of both tables in the sense that it is different in every row.
- I can't give you any information about a unique use just by knowing the plant's ID. I can only tell you about a collection of uses for each plant.

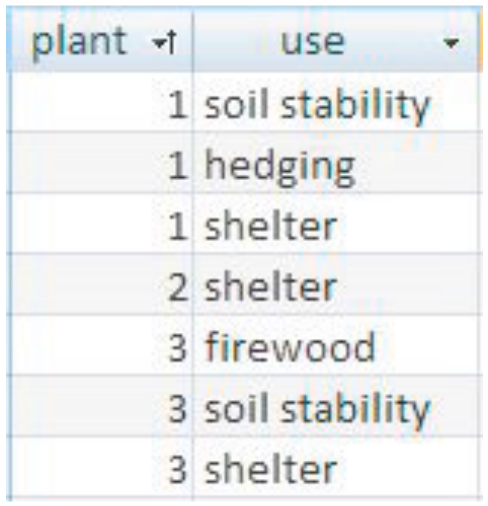
A table is not in first normal form if it is keeping multiple values for a piece of information.

If a table is not in first normal form, remove the multivalued information from the table. Create a new table with that information and the primary key of the original table.



plantID	genus	species	common_name
1	Dodonaea	viscosa	Akeake
2	Cedrus	atlantica	Atlas cedar
3	Alnus	glutinosa	Black alder
4	Eucalyptus	nichollii	Black peppermint gum
5	Juglans	nigra	Black walnut

Plant

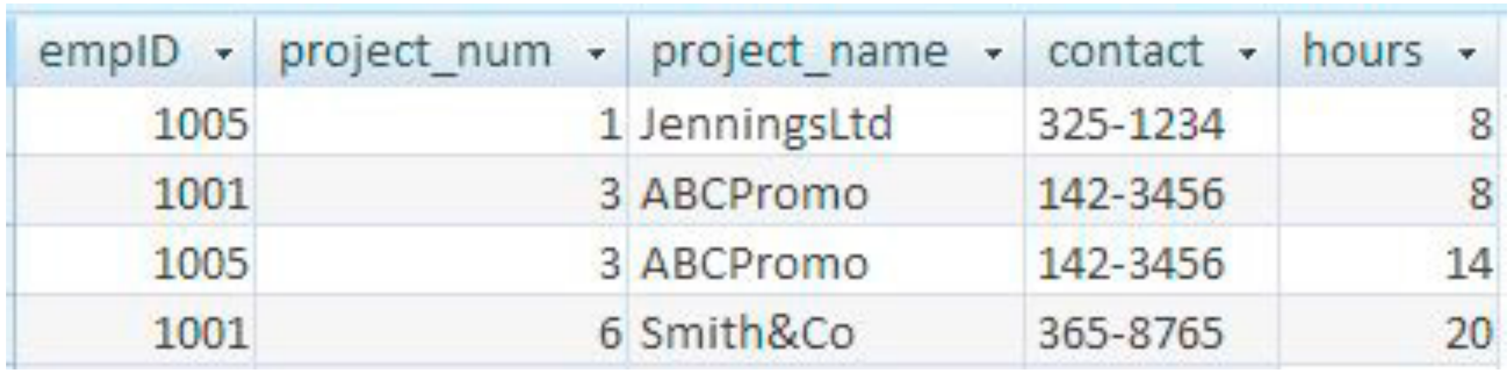


plant	use
1	soil stability
1	hedging
1	shelter
2	shelter
3	firewood
3	soil stability
3	shelter

PlantUse

Second Normal Form

- It is possible for a table in first normal form to still have updating problems.
- It has the information about the names and contacts of projects repeated several times, with the result that eventually the information might become inconsistent.



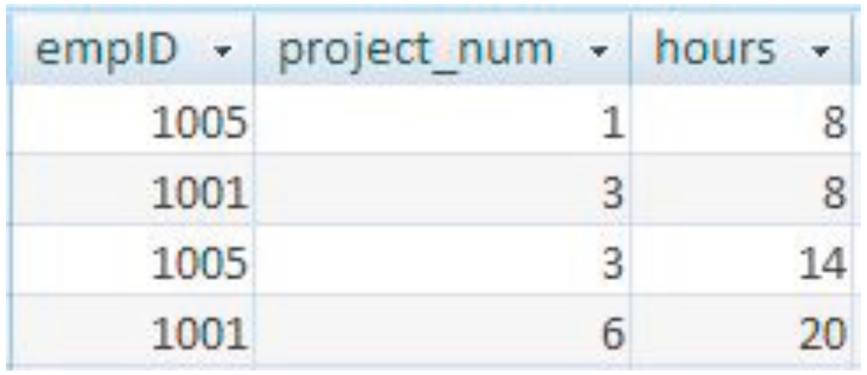
empID	project_num	project_name	contact	hours
1005	1	JenningsLtd	325-1234	8
1001	3	ABCPromo	142-3456	8
1005	3	ABCPromo	142-3456	14
1001	6	Smith&Co	365-8765	20

- The problem here is that while I can figure out the value of all the non-key fields by knowing the primary key.

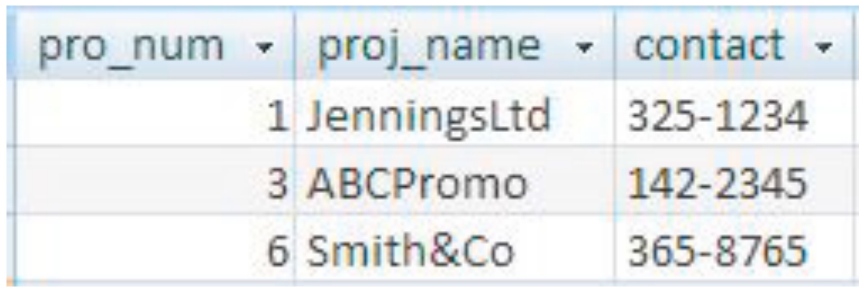
A table is in second normal form if it is in first normal form AND we need ALL the fields in the key to determine the values of the non-key fields.

If a table is not in second normal form, remove those non-key fields that are not dependent on the whole of the primary key. Create another table with these fields and the part of the primary key on which they do depend.

- This splitting up of an unnormalized table is often referred to as *decomposition*.



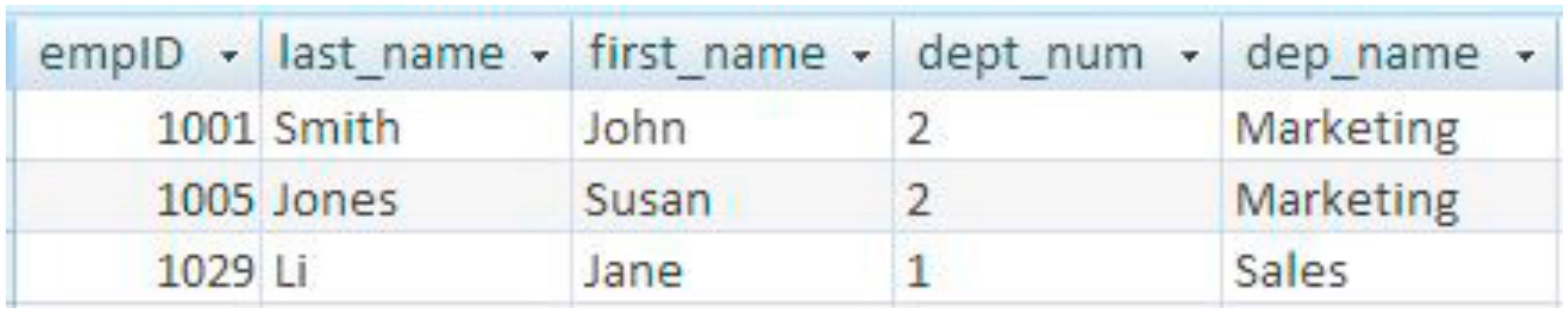
empID	project_num	hours
1005	1	8
1001	3	8
1005	3	14
1001	6	20



pro_num	proj_name	contact
1	JenningsLtd	325-1234
3	ABCPromo	142-2345
6	Smith&Co	365-8765

Third Normal Form

- Tables in second normal form can still cause us problems.

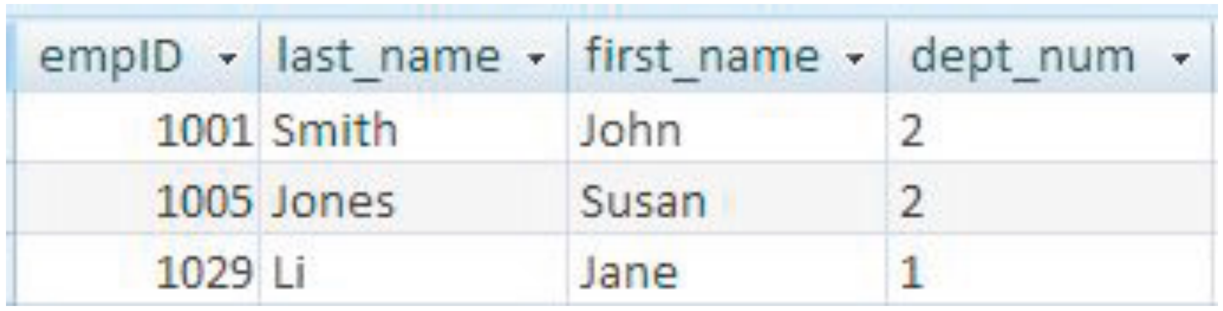


empID	last_name	first_name	dept_num	dep_name
1001	Smith	John	2	Marketing
1005	Jones	Susan	2	Marketing
1029	Li	Jane	1	Sales

  - What is the primary key for the **Employee** table?
  - If an employee works for only one department,
  - Is the table in first normal form?
  - Is the table in second normal form?
  - The information about the department name is repeated on several rows and is liable to become inconsistent.
  - The situation in this table is that the name of the department is determined by more than one field.

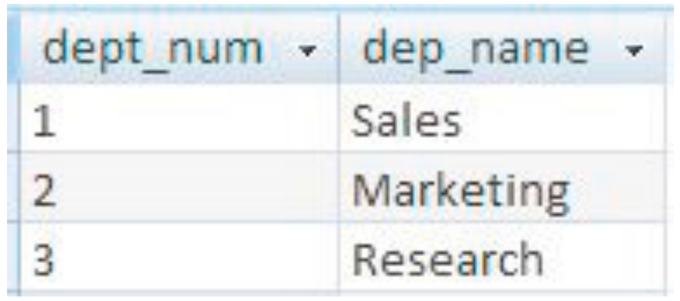
A table is in third normal form if it is in second normal form AND no non-key fields depend on a field(s) that is not the primary key.

If a table is not in third normal form, remove the non-key fields that are dependent on a field(s) that is not the primary key. Create another table with this field(s) and the field on which it does depend.



empID	last_name	first_name	dept_num
1001	Smith	John	2
1005	Jones	Susan	2
1029	Li	Jane	1

Employee



dept_num	dep_name
1	Sales
2	Marketing
3	Research

Department

Boyce-Codd Normal Form

- It is a slightly stronger statement for some tables where there is more than one possible combination of fields that could be used as a primary key.

A table is in Boyce-Codd normal form if every determinant could be a primary key.

- The value of a particular field determines the value of another field. We say that **proj_num** is a determinant.
- In any table where this is the case, then **proj_num** must be able to be the primary key.
- Boyce-Codd normal form is a statement that includes third normal form.

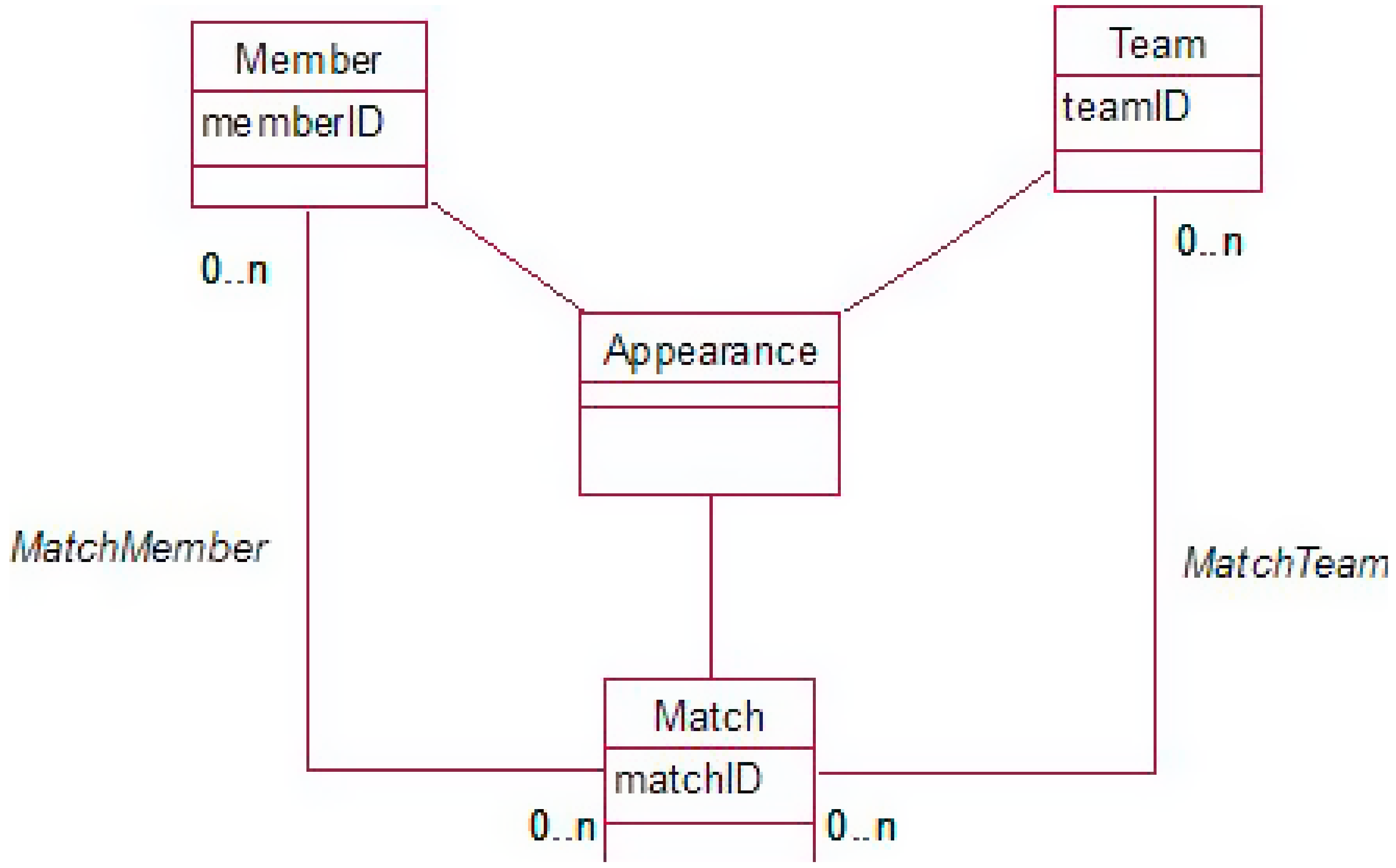
A table is based on
the key,
the whole key,
and nothing but the key.

Data Models or Functional Dependencies

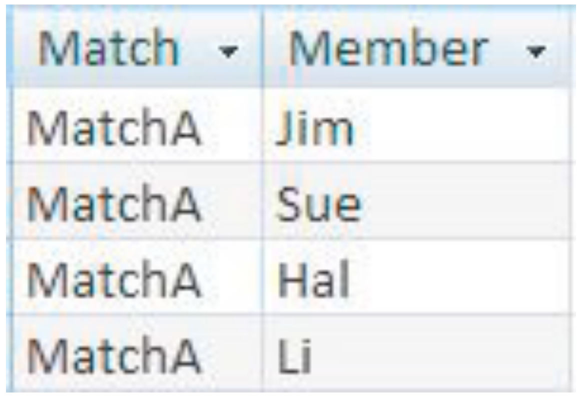
- What are the differences between the two approaches?
- The most essential thing is to understand the scope of the problem and the intricacies of the relationships between pieces of data. A detailed understanding requires us to ask very specific questions about the project.
- We can represent the answers with either part of a data model or by writing down a functional dependency.
- From a data modeling perspective, “Can an employee be associated with more than one project?”
- From a functional dependency perspective, “If I know the employee’s ID number, can I know a unique project with which she is associated?”

Additional Considerations

- Fourth and fifth normal forms deal with tables for which there are multi-valued dependencies.

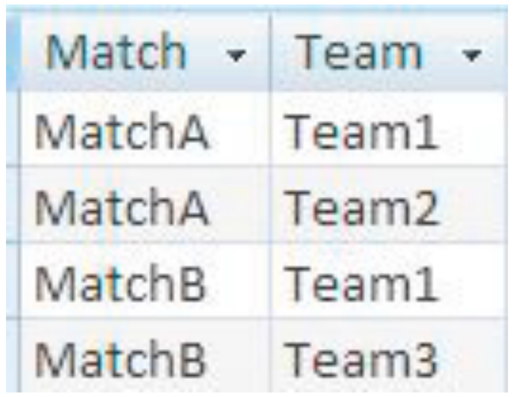


- We would need tables for each of the classes **`Member`**, **`Team`**, **`Match`**, and **`Appearance`**.
- We would also need two additional tables to represent the Many-Many relationships between **`Match`** and **`Team`**, and between **`Match`** and **`Member`**.



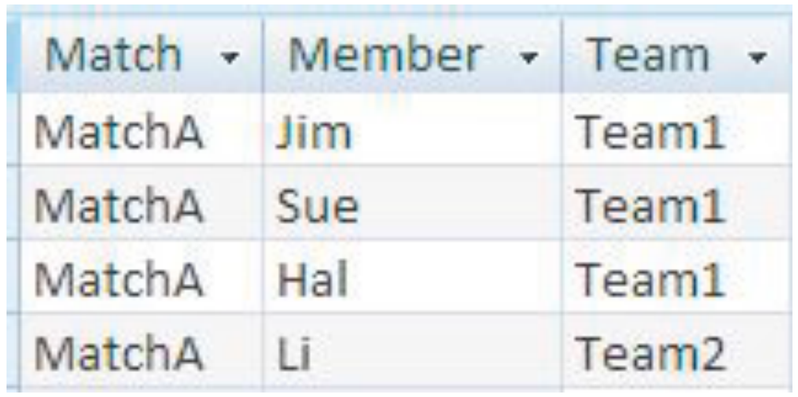
Match	Member
MatchA	Jim
MatchA	Sue
MatchA	Hal
MatchA	Li

MatchMember



Match	Team
MatchA	Team1
MatchA	Team2
MatchB	Team1
MatchB	Team3

MatchTeam



Match	Member	Team
MatchA	Jim	Team1
MatchA	Sue	Team1
MatchA	Hal	Team1
MatchA	Li	Team2

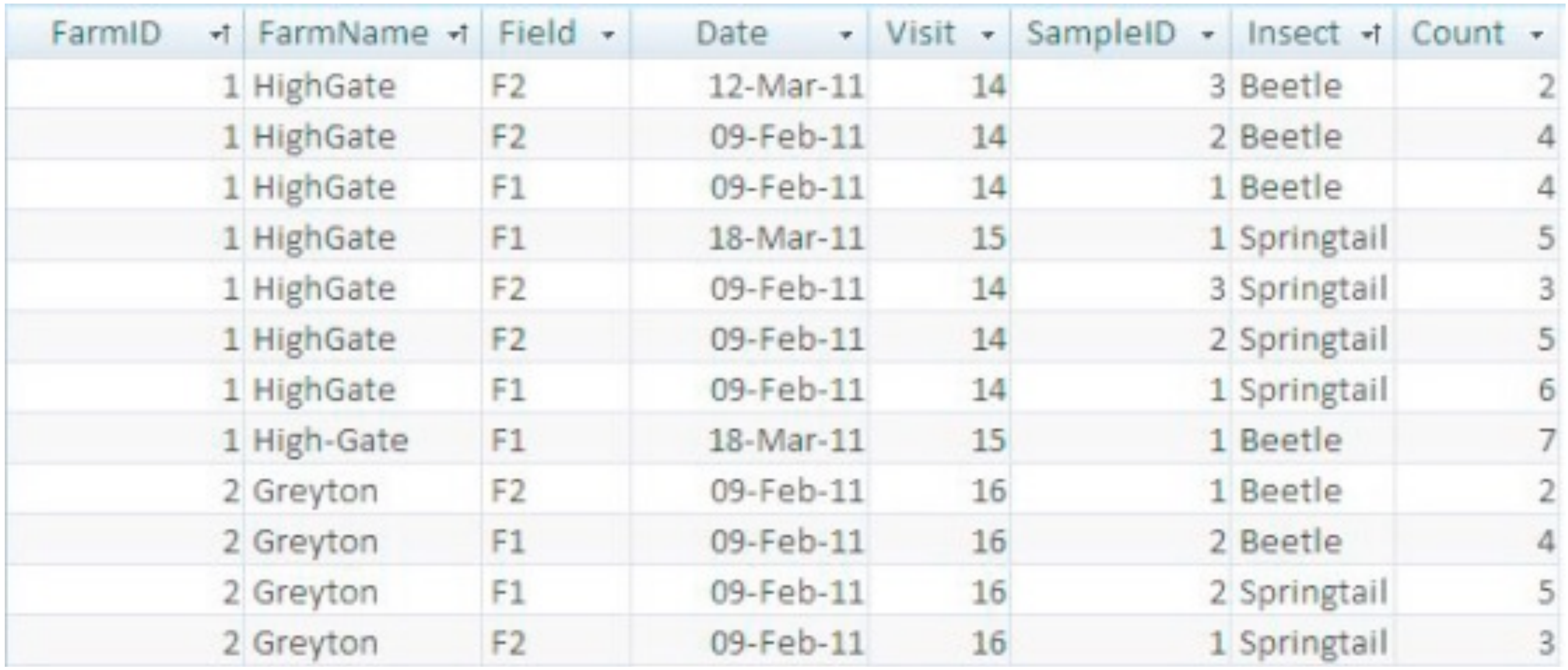
Appearence

- The primary key is made up of all the fields. There are no non-key fields, and there are no functional dependencies. They are all therefore in Boyce-Codd normal form because there are no determinants that are not possible keys

- “Do we need all three tables?”
- There is clearly some repeated information with the data as it stands.
- When information is stored twice, there is always the danger of it becoming inconsistent.
- So what (if anything) do we need to get rid of?
- “Are these two sets of information independent for our problem?”
- We need to know “who played for which team in which match,” the Appearance table is necessary
- “Do we want to know about matches and teams independent of the members involved?”
- When the original draw for the competition is determined? We will probably need to record in our database that Team1 and Team2 are scheduled to play in MatchA.
- We need to record a score. Without a MatchTeam table, where would we store that?
- “Do we want to know about members and matches independent of the teams?”

Exercise 11-1

Example 7-3 (Insect Example) is a good real-life example of unnormalized data. To recap: Farms are visited and a number of samples are taken from different fields. The number of each species (just Springtail and Beetle for now) in each sample is recorded. A version of the data is shown in the figure. Consider the following questions:



FarmID	FarmName	Field	Date	Visit	SampleID	Insect	Count
1	HighGate	F2	12-Mar-11	14	3	Beetle	2
1	HighGate	F2	09-Feb-11	14	2	Beetle	4
1	HighGate	F1	09-Feb-11	14	1	Beetle	4
1	HighGate	F1	18-Mar-11	15	1	Springtail	5
1	HighGate	F2	09-Feb-11	14	3	Springtail	3
1	HighGate	F2	09-Feb-11	14	2	Springtail	5
1	HighGate	F1	09-Feb-11	14	1	Springtail	6
1	High-Gate	F1	18-Mar-11	15	1	Beetle	7
2	Greyton	F2	09-Feb-11	16	1	Beetle	2
2	Greyton	F1	09-Feb-11	16	2	Beetle	4
2	Greyton	F1	09-Feb-11	16	2	Springtail	5
2	Greyton	F2	09-Feb-11	16	1	Springtail	3

a) What are some of the updating problems that could occur with the table in the figure?

b) Which of the following functional dependencies hold for the insect data?

- *FarmID → FarmName?*
- *FarmID → Visit?*
- *Visit → Date?*
- *Date → Visit?*
- *Visit → FarmID?*
- *Sample → Field?*
- *(Sample, VisitID) → Field?*
- *(Sample, Insect) → Count?*
- *(VisitID, Sample, Insect) → Count?*

c) (VisitID, Sample, Insect) is suggested as an appropriate primary key. Can you determine all the other values from knowing the values of these three fields? Would it be a suitable primary key?

d) Using the fields in Part C as a primary key use the normalization rules to decompose the table into a set of tables in third normal form.

Summary

- If we have poorly structured tables in a database, we run the risk of having problems with updating data. These include:

  Modification problems: If information is repeated, it will become inconsistent if not updated everywhere.

  Insertion problems: If we don't have information for each of the primary key fields, we will not be able to enter a record

  Deletion problems: If we delete a record to remove a piece of information, we might as a consequence lose some additional information.

Summary

- By understanding the concepts of functional dependencies, primary keys, and normalization, we can ensure that our tables are structured in such a way as to avoid the update problems described previously.
  - A functional dependency exists between two sets of fields in a table: If field A functionally determines field B, this means that if I know the value for A, I can uniquely tell you a value for B.
  - A primary key is a (minimal) set of field(s) that functionally determines all the other fields in the table.
  - The first three normal forms can be summed up as

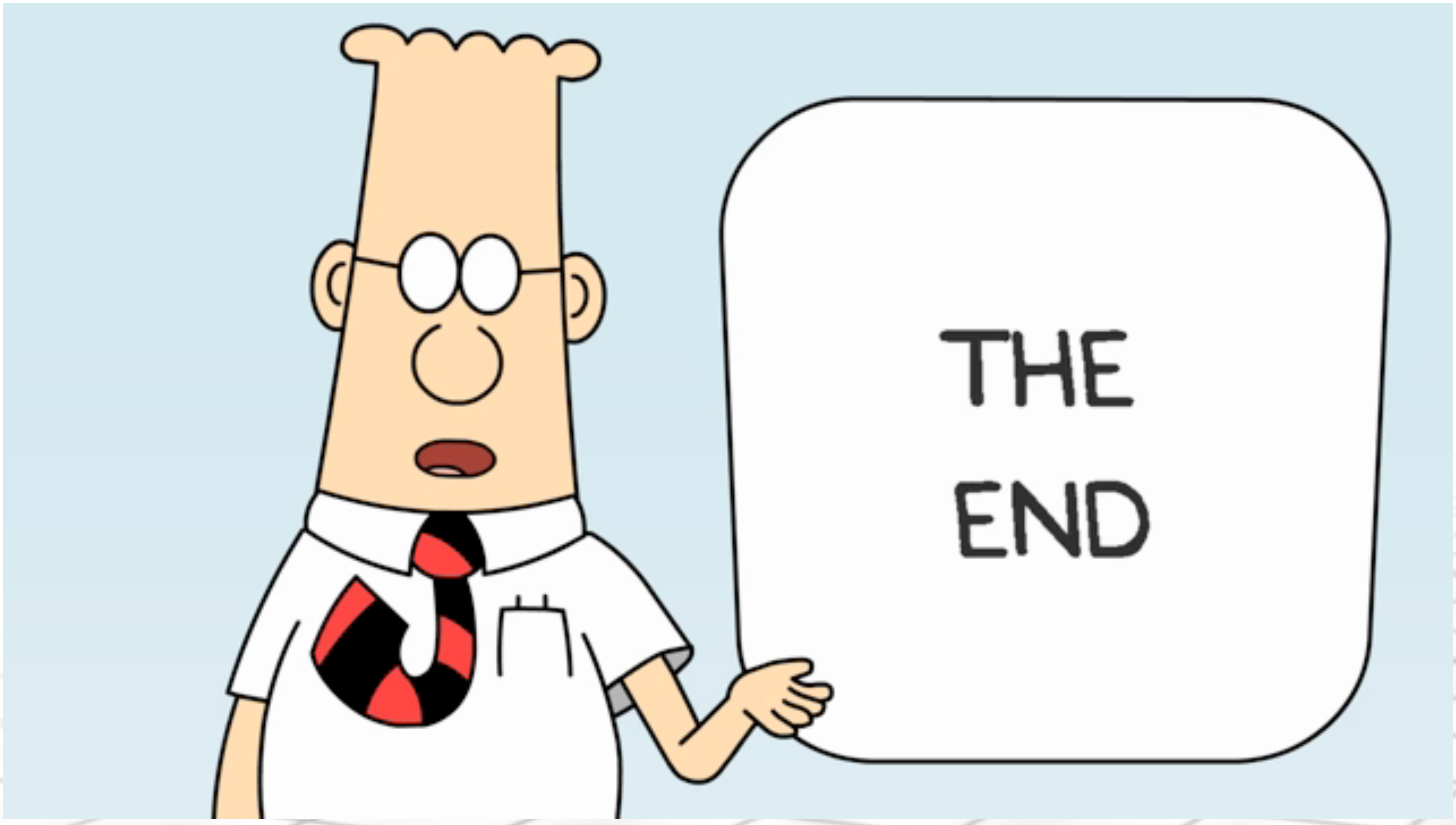
    A table is based on

    the key,

    the whole key,

    and nothing but the key
  - A table in Boyce-Codd normal form is one in which every determinant could be a primary key.
  - Where you have three or more interrelated classes, ask questions about what information, if any, you need to know that involves all three classes and what information involves two classes independent of the third.

- When designing a relational database
    - Create original use cases and a data model.
    - Ask questions about the data model to improve understanding of the problem.
    - Represent the data model with tables, primary keys, and foreign keys.
    - Check that each table is suitably normalized.



출처: metachannels.com

Thank you!