12. More on Keys and Constraints

Fall 2018

- We looked at how to represent relationships between classes with primary and foreign keys and then applied the ideas of normalization to ensure the attributes were in the right tables.
- We take another look at some of these ideas and think about some alternative possibilities.
- We take a closer look at primary keys and how to choose them.
- We also take a look at how we can maintain referential integrity when data is being constantly updated.

Today's Lecture

- Choosing a Primary Key
- Unique Constraints
- Using Constraints Instead of Category Classes
- Deleting Referenced Records
- Exercises
- Summary

Choosing a Primary Key

- The key fields will have unique values and so can be used to identify a particular row in a table.
- The primary key is also used to set up relationships between rows in different tables by way of a foreign key.
- Introducing a customer number or using some sort of automatically generated ID number can ensure that we have a field that is unique for every row.

More About ID Numbers

- A generated ID number does not solve all of our problems.
- If we have two rows in our table that are identical in every respect except for their ID, we are going to be in real trouble.
- Are they the same person or are they different people?
- We always expect that a business will be able to find our customer number for us from information that differentiates us from everyone else.
- There will (or should) always be a possible key made up of some combination of the data kept about a customer?
- One of the main reasons that ID numbers are necessary in many cases is that while there might always be some information that distinguishes one customer from another, it is likely that some of those values are constantly changing.

- While ID numbers are essential, there are still problems with them.
- One problem arises when a person is assigned two ID numbers for the same organization.
- At various times the number of patients associated with New Zealand hospitals has been about 25% more than the total population!
- There is not much that can be done about these problems other than to have very careful procedures at data entry times.
- The process cannot be automated though, because sometimes two different people will have identical names and even birth dates.

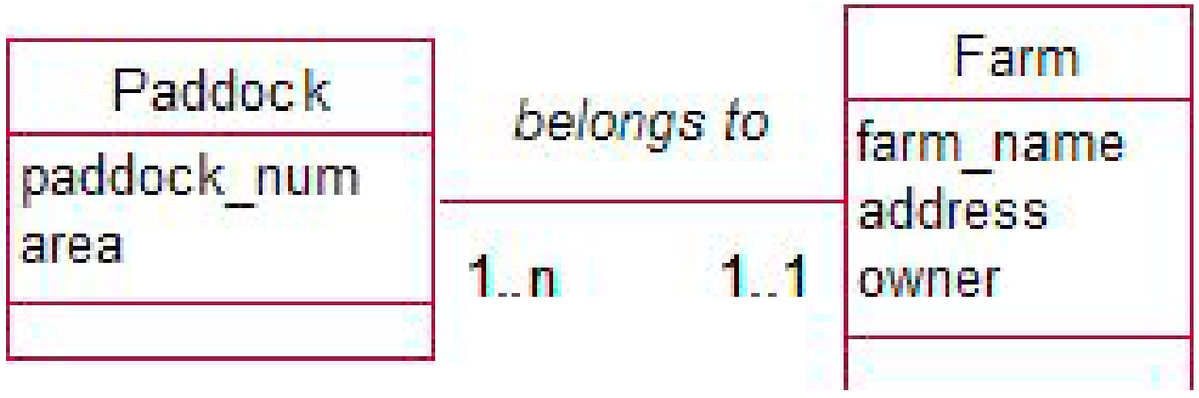
Candidate Keys

A candidate key is a key where no subset of the fields is also a key.

```
Customer(customerID, customer_name, address, phone,
         birth_date, tax_number)
```

- We have two candidate keys: customerID and tax_number.
- What are the considerations for choosing a primary key from among two or more candidates?

An ID Number or a Concatenated Key?



- What will be a suitable primary key for the table representing the Farm class?
- An ID number seems the safest bet.
- What about paddocks?

Farm(<u>farmID</u>, farm_name, address, owner)

Paddock(<u>paddock_num</u>, area, farm)

- Is the paddock number a unique number over all paddocks, or is it just unique within a farm?

paddock_num	area	farm
336	30	18
337	23	18
345	25	17
346	25	17
347	35	17

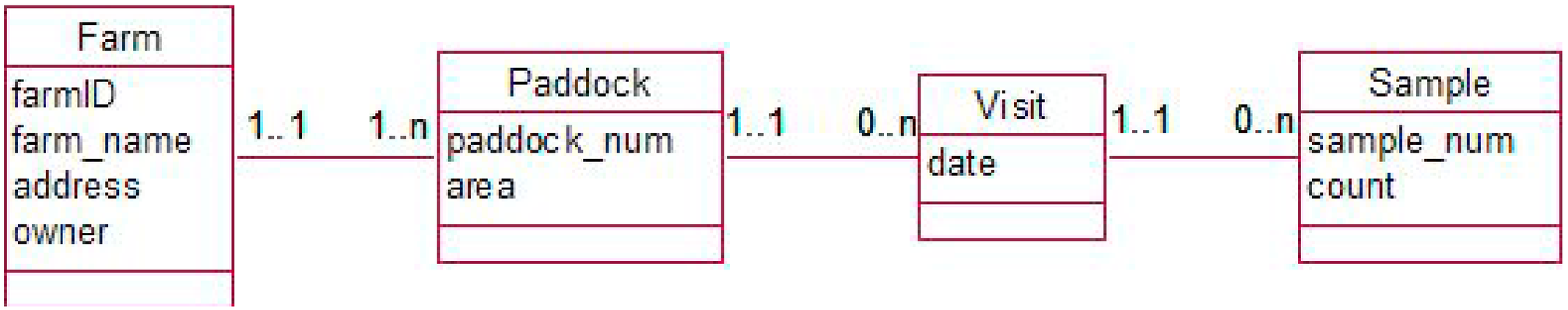
Primary key paddock_num

farm	paddock_num	area
18	1	30
18	2	23
17	2	25
17	3	25
17	4	35

Primary key farm and paddock_num

- This relationship between farm and paddock (a 1-Many with a compulsory 1 end) is sometimes referred to as an *ownership* relation.
- The paddock *must* have an associated farm, or looking at it the other way around, the farm *owns* the paddock.

- When we get a long line of 1-Many ownership relationships, the issue of the size of the foreign key becomes more pressing.



Farm(<u>farmID</u>, farm_name, address, owner)

Paddock(<u>farmID</u>, <u>paddock_num</u>, area), with **farmID** being a foreign key referring to **Farm**

Visit(<u>date</u>, <u>farm</u>, <u>paddock</u>), with (**farm**, **paddock**) being a foreign key referring to **Paddock**

Sample(<u>date</u>, <u>farm</u>, <u>paddock</u>, <u>sample_num</u>, count), with (**date**, **farm**, **paddock**) being a foreign key referring to **Visit**

Farm(<u>farmID</u>, farm_name, address, owner)

Paddock(<u>farmID</u>, <u>paddock_num</u>, area), with **farmID** being a foreign key referring to **Farm**

Visit(<u>date</u>, <u>farm</u>, <u>paddock</u>), with (**farm**, **paddock**) being a foreign key referring to **Paddock**

Sample(<u>date</u>, <u>farm</u>, <u>paddock</u>, <u>sample_num</u>, count), with (**date**, **farm**, **paddock**) being a foreign key referring to **Visit**

- We are assuming paddocks are numbered from 1 within each farm and samples are numbered from 1 within each visit.
- The **Visit** table doesn't need to have an ID because the combination **(<u>date</u>, <u>farm</u>, <u>paddock</u>)** is unique for this problem.
- The **Sample** table is now looking quite cumbersome because the foreign key referring to the **Visit** table is a combination of three fields. This table is going to have the most rows eventually.
- There could also be a size consideration.

- Had we used the alternative of a single key for **`Paddock`**?
- The foreign keys in the **`Visit`** and **`Sample`** tables would be a little smaller, but at the expense of having less-intuitive identifications for paddocks.
- What other options do we have? Introducing a **`visitID`** makes some sense.

`Farm(`<u>**`farmID`**</u>**`, farm_name, address, owner)`**

`Paddock(`<u>**`farmID`**</u>**`,`** <u>**`paddock_num`**</u>**`, area)`**, with **`farmID`** being a foreign key referring to **`Farm`**

`Visit(`<u>**`visitID`**</u>**`, date, farm, paddock)`**, with (**`farm`**, **`paddock`**) being a foreign key referring to **`Paddock`**

`Sample(`<u>**`visit`**</u>**`,`** <u>**`sample_num`**</u>**`, count)`**, with (**`date`**, **`farm`**, **`paddock`**) being a foreign key referring to **`Visit`**

- All our introduced ID numbers therefore have some meaning, and the **`Sample`** table is considerably smaller than in the previous design.

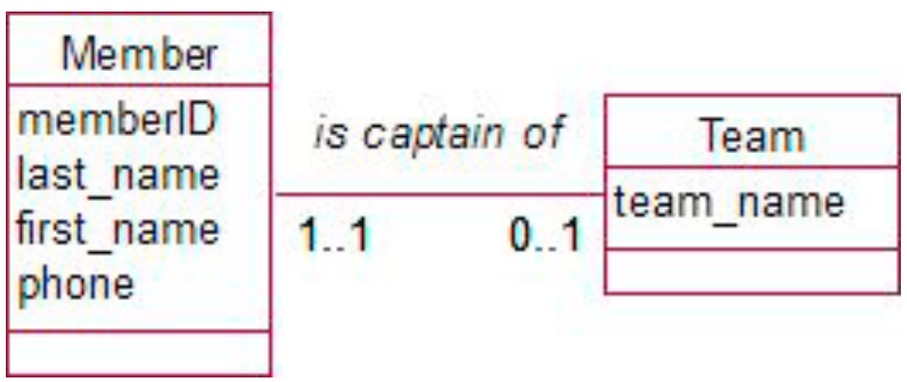
Unique Constraints

- We have two candidate keys: `visitID` and the combination (`date`, `farm`, `paddock`).
- We choose `visitID` as the primary key.
- Do we lose anything by making this choice?
- We still want to maintain the uniqueness of this combination, and we can do this by setting up a unique constraint.

visitID	date	farm	paddock
23	1/03/2011	18	1
24	1/03/2011	18	2
25	1/04/2011	17	3
26	1/04/2011	17	4

```
CREATE TABLE Visit (
visitID INT PRIMARY KEY,
date DATE,
farm INT,
paddock INT ,
FOREIGN KEY (farm, paddock) REFERENCES Paddock
UNIQUE (date, farm, paddock) )
```

- Unique constraints are also a way to enforce a 1-1 relationship between tables.



team_name	captain
SeniorA	203
SeniorB	156
Wed Social	203

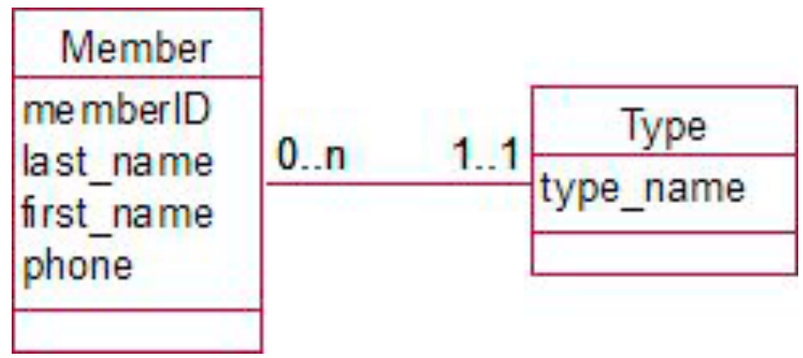
- Each team can have only one captain (because we only have one captain field); however, we have to ensure that each member can captain only one team as required by the 1-1 relationship.

```
CREATE TABLE Team (
team_name VARCHAR(20) PRIMARY KEY,
captain INT UNIQUE FOREIGN KEY REFERENCES Member)
```

Using Constraints Instead of Category Classes

- We are going to have a look at when you might decide not to add additional classes and why.

memberID	last_name	first_name	type
156	Jones	Graeme	senor
187	Green	Chris	Junior
203	Wang	James	Senior



memberID	last_name	first_name	type
156	Jones	Graeme	Senior
187	Green	Chris	Junior
203	Wang	James	Senior

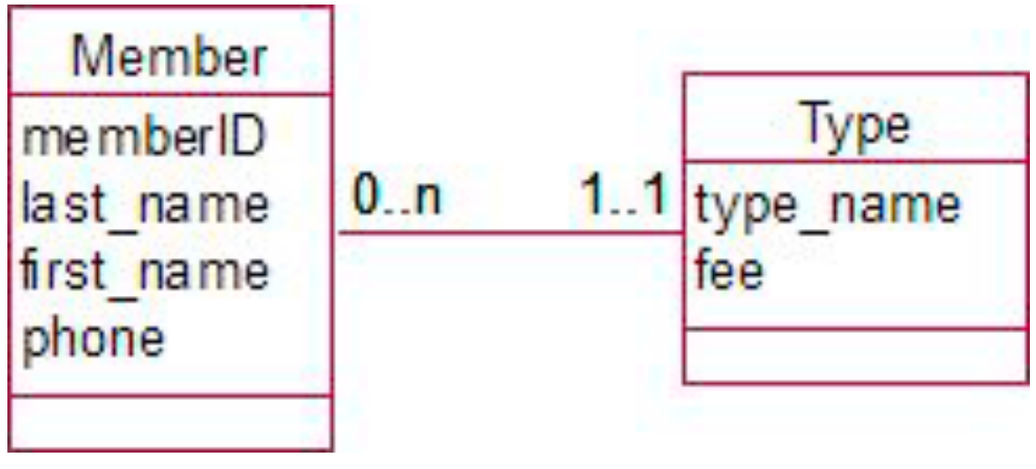
Member

type_name
Senior
Junior
Social

Type

```
CREATE TABLE Member (
memberID INT PRIMARY KEY,
last_name VARCHAR(20),
first_name VARCHAR(20),
type VARCHAR(20) CHECK type IN ('Senior', 'Junior', 'Social'))
```

- Which should we prefer: a table with a constraint on a field or one with a reference to another table?



Deleting Referenced Records

team_name	captain
SeniorA	203
SeniorB	156

Team

memberID	last_name	first_name	type
156	Jones	Graeme	Senior
187	Green	Chris	Junior
203	Wang	James	Senior

Member

- A foreign key ensures that we have referential integrity.
- A foreign key field is not necessarily mandatory, and the captain field may be empty
- We also have the situation of deleting members from the Member table.

Disallow delete: You cannot delete a row that is being referenced. For example, the deletion of member 156 will not be allowed while it is being referenced by the **`Team`** table. If we want to delete member 156, we will first have to remove the reference to him in the **`Team`** table and then delete him from the **`Member`** table.

Nullify delete: If member 156 is deleted, the field that is referencing it, **`captain`**, will be nullified (made empty). This essentially is saying that if a captain of a team leaves the club, that team has no captain—which is probably quite sensible in this situation.

Cascade delete: If a row is deleted, all the rows referencing it will be deleted also (and the rows referencing them, and on and on). In this case, deleting member 156 would mean that the team SeniorB would be deleted. This is clearly not desirable.

```
CREATE TABLE Team (
team_name VARCHAR(20),
captain INT FOREIGN KEY REFERENCES Member ON DELETE NULLIFY )
```

order_num	date	customer	product	quantity
10034	1/Mar/11	1345	809	4
10035	1/Mar/11	1562	975	3
10036	2/Mar/11	1345	996	1

Order

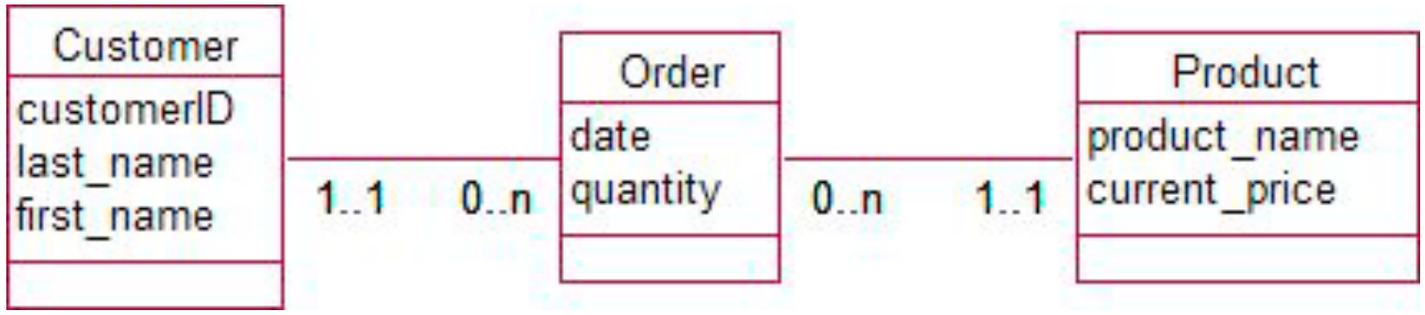
product_num	name	price
809	teddy	10.50
810	doll	15.75
811	cart	23.80

Product

- If we no longer stock product 809
- It is important that we keep discontinued products in the table, but we will want to be able to distinguish them from current products.
- How do we prevent new orders from being entered for discontinued products?
- A *trigger* is a procedure that is fired by a change to a table (e.g., adding or updating a row) and will carry out specified actions.
- Having said all that, we might think that the safest option is to never delete anything. It is possible to set up tables so that no rows can ever be deleted.

Exercise 12-1

The ever-useful "customer orders product" example again—there is always something new to discover. Design the table that will represent the Order class in the figure. Consider constraints, primary and foreign keys, and updating rules.

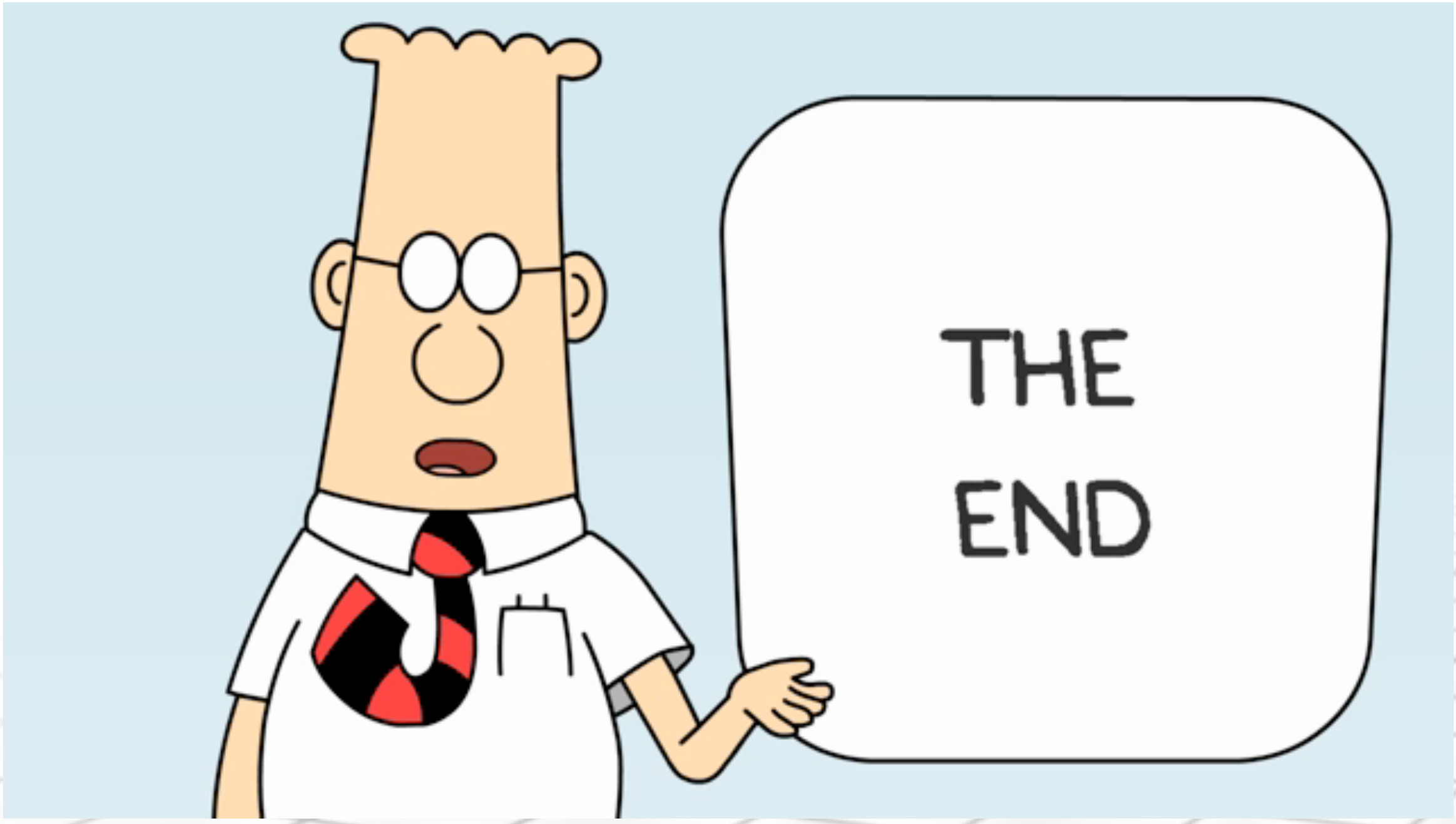


Summary

We have looked at some issues involved with choosing appropriate primary keys and for ensuring referential integrity is maintained when we update the data in our tables. Some of the important points to remember are the following:

- We often need to introduce a generated ID number to ensure we have a field, with stable and unique values, that we can use as a primary key. This is particularly true for people, where identifying information such as names and addresses are likely to change.
- Be aware that mistakes in data entry means it is possible to have a person in your database twice with two different ID numbers. Try to avoid this!
- Where a primary key is made up of several concatenated fields, it is worth considering a generated ID number to reduce the size of the foreign keys referencing the table.

- Where a generated ID has been introduced, constraints should be used to retain the uniqueness of the combinations of fields that have been replaced as a primary key.
- Unique constraints can be used to enforce a 1-1 relationship.
- A constraint on the value of a field may be more appropriate than a relationship to another (very simple) table.
- You have three options when you wish to delete a row that it is being referenced by a foreign key:
  - Disallow the deletion.
  - Make the field referencing the deleted row NULL (“nullify delete”).
  - Remove all rows that reference the deleted row (“cascade delete”).



출처: metachannels.com

Thank you!