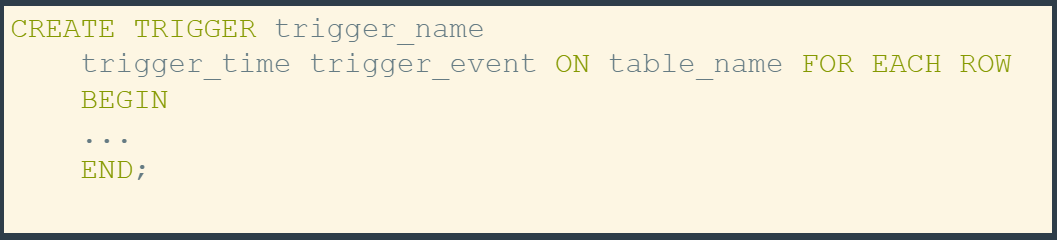
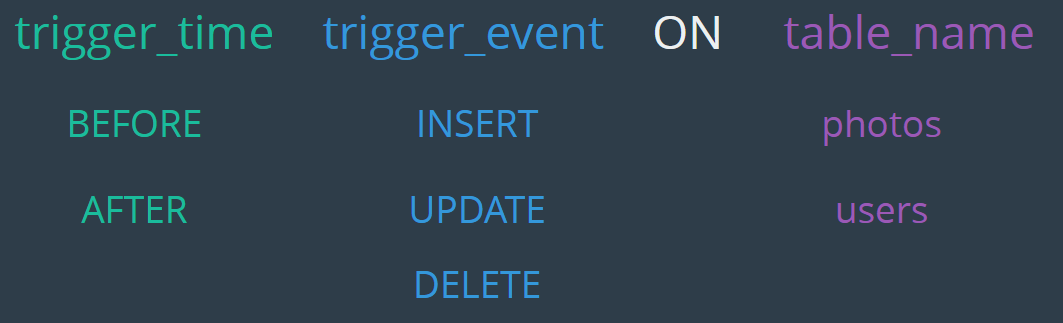
Section slides: <http://webdev.slides.com/coltsteele/mysql-113?token=uW3AQYWv>

# Introduction to Database Triggers

* **Database triggers** are SQL statements that are *automatically run* when a specific table is changes
  + Very useful in particular situations, but by no means central to MySQL
* The syntax for triggers



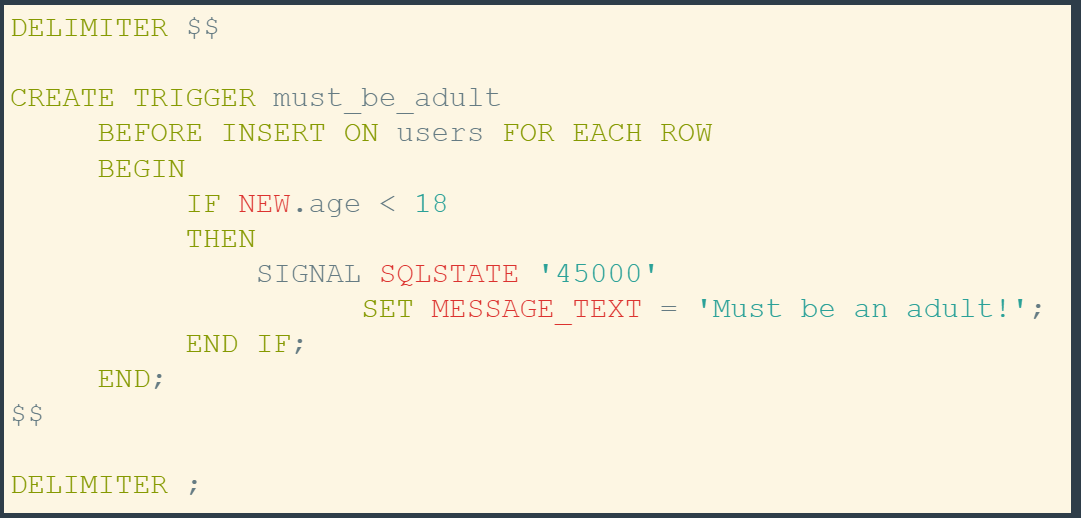
* + trigger\_time describes *when* a trigger is executed, and can either be BEFORE or AFTER the trigger\_event
  + trigger\_event describes the nature of the event that triggers, and can be INSERT, UPDATE, or DELETE
  + table\_name is just the name of the table
  + Example: We can run some code BEFORE we INSERT into a table called *photos*
  + Whatever code is sitting between BEGIN and END will be executed upon the trigger



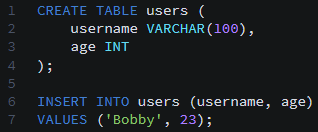
* What would you do with these triggers?
  + **Data validations** – you can enforce certain things on your data
    - You can accomplish this by running code before an INSERT statement is run
    - Example: You can prevent someone from being added to your table if they are below a certain age
    - This is not the greatest way to do this. You could also do this within your application or client-side code, which is oftentimes much easier because you can program those restrictions before you ever execute any SQL queries
  + Manipulating other tables relative to the “trigger table”
    - Example: If you have following relationships in Instagram, it might be helpful to know if someone *unfollows* something. This data could be useful for certain companies
      * By using a trigger, we can create a new row once a “follow” relationship is deleted

# Writing Our First Trigger Part 1

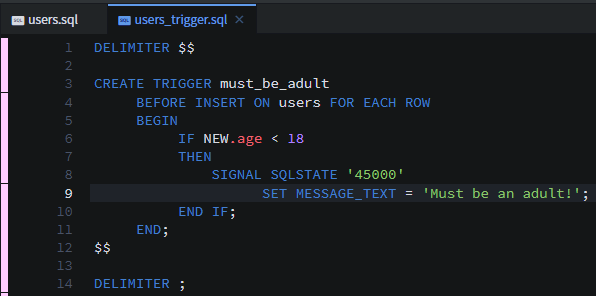
* Slides: <http://webdev.slides.com/coltsteele/mysql-113?token=uW3AQYWv#/4>
* Here we’ll run the code first, then ask questions later. This is a simple validation trigger. What it does is prevent a user from being created if their age is under 18



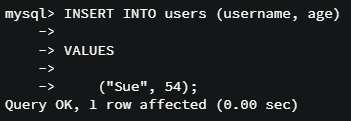
* Start by creating a new database called **trigger\_demo**
* Then within a SQL file, we’ll create a table called *users* and insert one user named Bobby

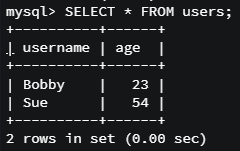


* Now let’s make the trigger, which will check if the new user is less than 18. If they are, we’ll throw an error and not allow the user to be inserted into the table. We use the code above in a new SQL file and then just source the file



* Let’s test it out!
  + Inserting a user who is over 18 works just fine

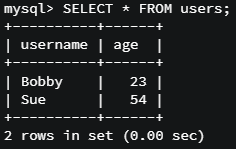




* + But inserting a user who is under 18 does not work, and that user is not added to the table



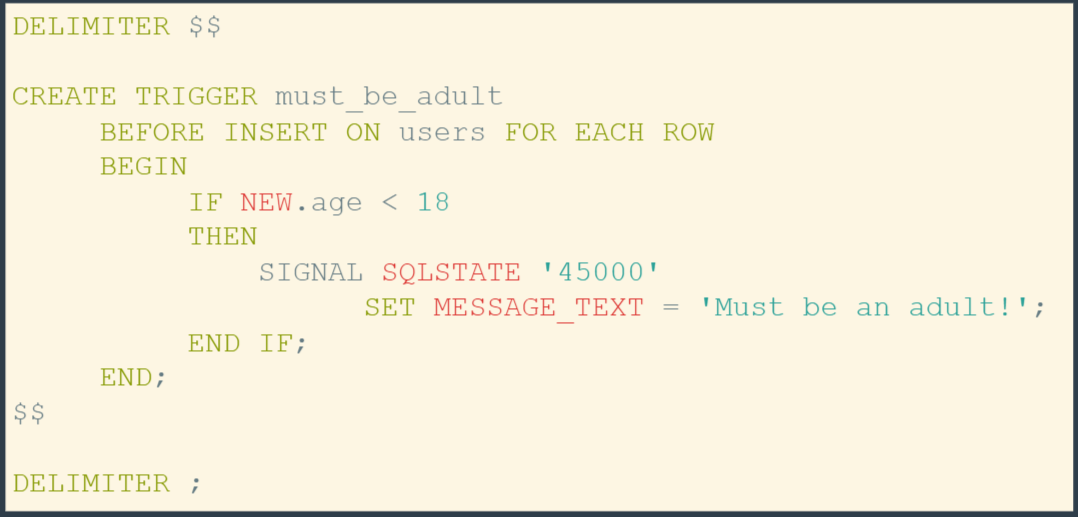




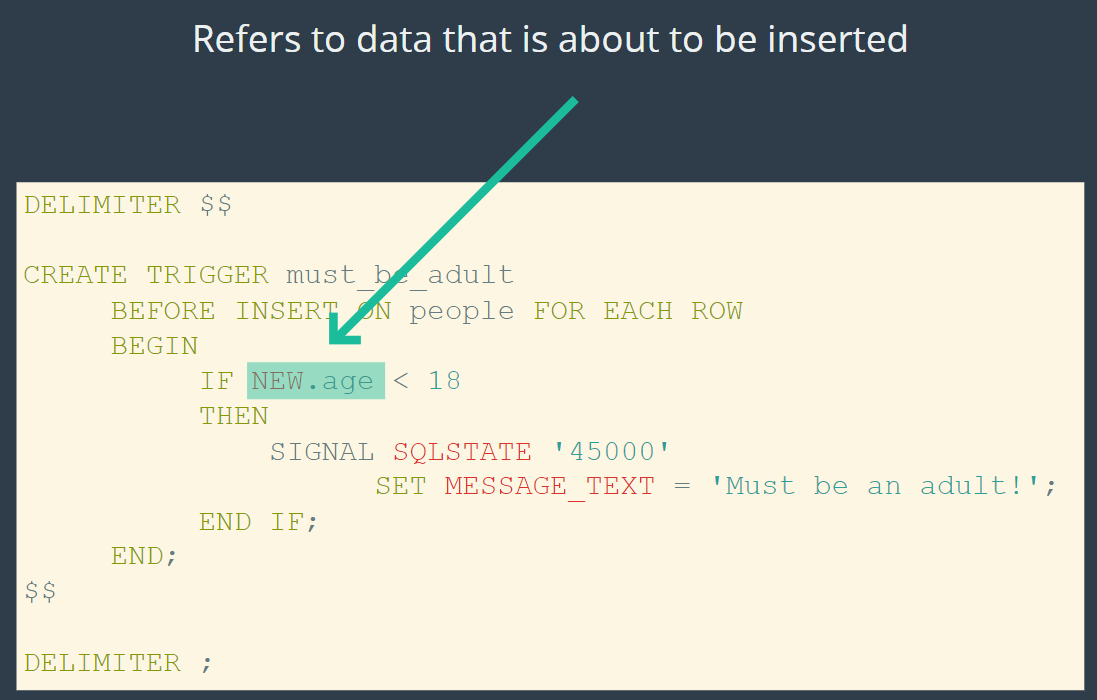
* Great! But there’s still a bunch of syntax in the code that we don’t understand. We’ll talk about it in the next lecture

# Writing Our First Trigger Part 2

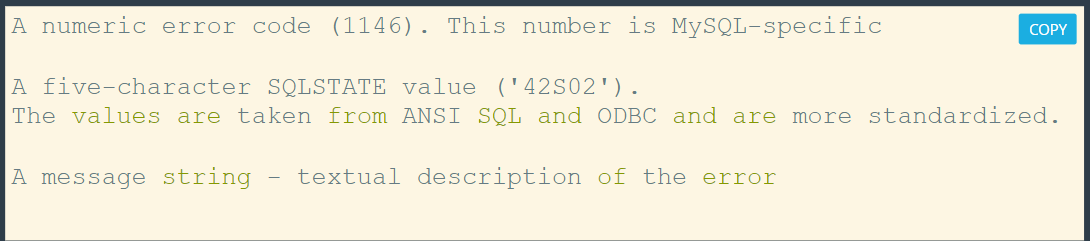
* Slides: <http://webdev.slides.com/coltsteele/mysql-113?token=uW3AQYWv#/4>
* So we ran the code first. Let’s now talk about it



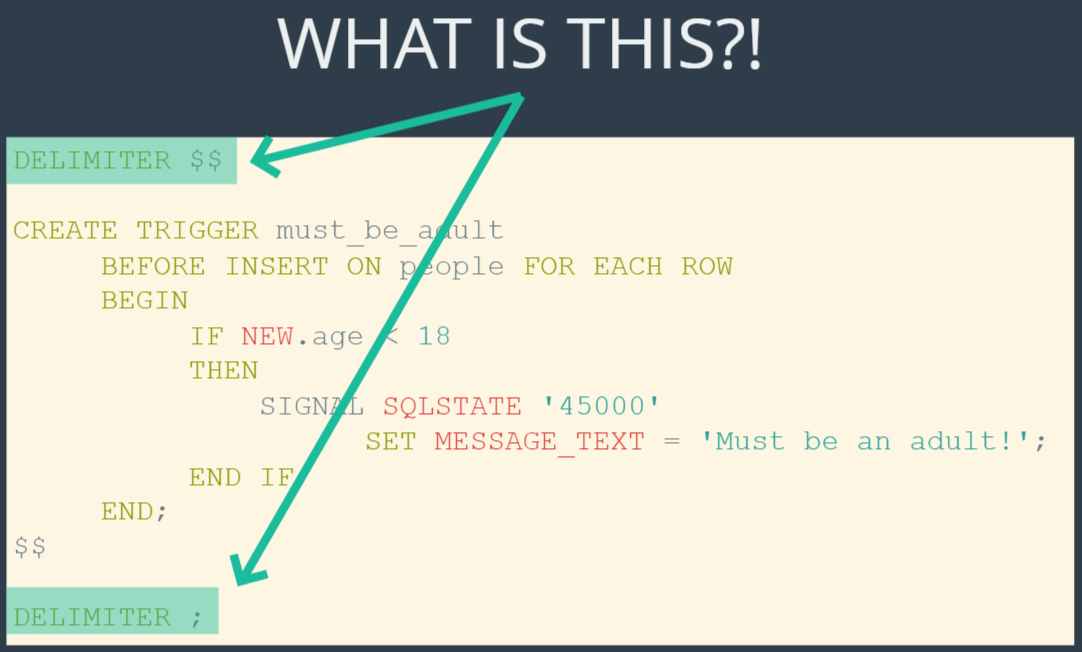
* **CREATE TRIGGER** is where you name the trigger, so that you can delete it by its label
  + Generally helpful to give it a meaningful name
* The next line essentially says “We want to execute this code BEFORE something is INSERTed into the *users* table”
* FOR EACH ROW is default syntax. Don’t worry much about it
* The BEGIN and END denotes the code that will run every time right before something is inserted into *users*
* The NEW.age phrase is a placeholder that refers to the data that’s about to be inserted. Basically NEW is an alias for the new user that’s being evaluated. We then look at the “age” property/value of that user and determine if it’s less than 18
  + OLD is also a placeholder that gives us access to information that is deleted



* The SQLSTATE bit is a little confusing, and this slide summarizes the components of SQL errors



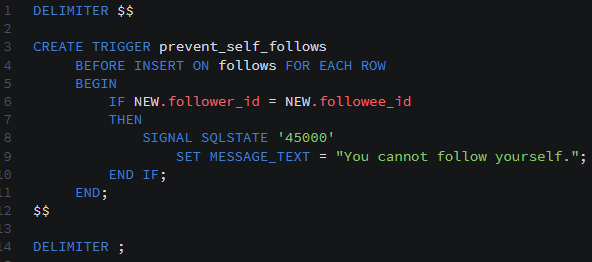
* For our purposes, we will only use SQLSTATE 45000, which is a wildcard generic error state that represents an unhandled, user-defined exception
  + In this example, we signal the SQLSTATE of 45000 and then set a custom message MESSAGE\_TEXT (required syntax) and then write out any message we want
* What’s the DELIMITER stuff?



* + The DELIMITER statement protects the semicolons within the enclosed code from indicating an end to the SQL statements, thus allowing SQL to continue to progress through the code when semicolons are encountered
  + Here, we have temporarily changed the indicator for the end of a SQL statement from a semicolon to two dollar signs $$. Thus, $$ marks the actual end of our code
  + Then at the end of the code block, we change the delimited back to semicolon
* The instructor essentially never writes database triggers from scratch – he copies them and then modifies them as needed

# A Database Trigger to Prevent Self-Following on Instagram

* We return to our Instagram clone and create a trigger that prevents a person from following themselves in Instagram
  + Again, this is probably better handled on the client site, if this were an app with external code. But for us, this is really the only way to get it done
* Here’s what we want to do – before inserting a new follow, check whether **follower\_id** is the same as **followee\_id**. If it is, do not perform the insertion and throw an error
* Here’s our code for this trigger:



* + Now let’s test it out!



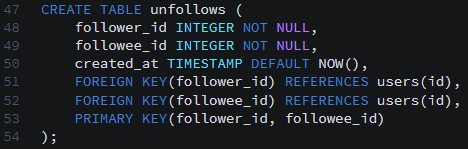


* + And note that we can still create normal follows as usual:

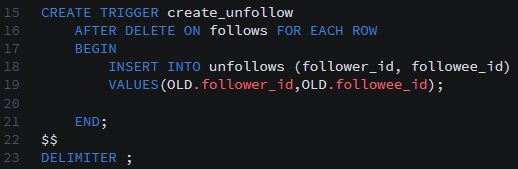


# Creating Logger Triggers

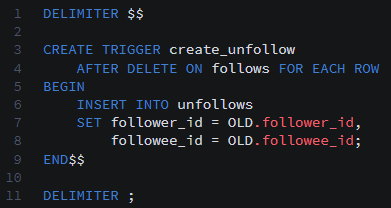
* Slides: <http://webdev.slides.com/coltsteele/mysql-113?token=uW3AQYWv#/14>
* In this example, we’ll be creating new data based on an action
  + After a follow is deleted, we’ll insert a new row into a new table called *unfollows* that records when this unfollowing happens, who is doing the unfollowing, and who is being unfollowed
* Within our Instagram app, we want to keep track of when someone unfollows someone else
  + Maybe there’s a pattern of people unfollowing certain companies or users
* We start by creating an “unfollows” table



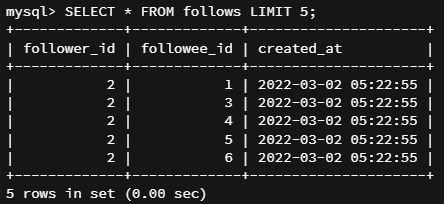
* Now we build the **unfollows trigger**



* + What this will do is AFTER a delete happens on the *follows* table, we’ll insert the OLD data in to *unfollows* table
  + This is another syntax that will also work and utilizes the SET function
    - All we’re difference is that we’re using equal signs to set the data instead of the VALUES syntax

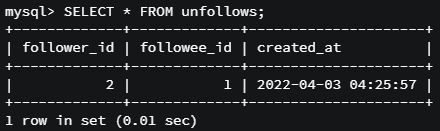


* To test this, we’ll remove a follow and see what happens. First, we delete the follow of person 2 on person 1

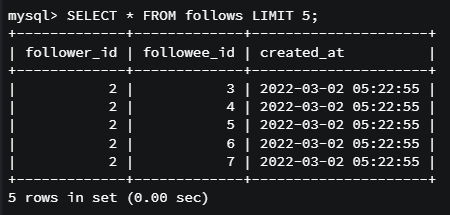




* + Next we check our *unfollows* table to make sure the unfollow was documented



* + And we see that’s it’s gone from the *follows* table



# Managing Triggers and a Warning

* Slides: <http://webdev.slides.com/coltsteele/mysql-113?token=uW3AQYWv#/16>
* Trigger management involves showing which triggers are active, and how to delete them
* Showing triggers is as easy as running this code



* + What you get back is kind of ugly to look at, but it nevertheless shows us all of the triggers being used in that particular database
* To remove a trigger, use this syntax:



* As a final word of warning, *triggers can make debugging difficult!*
  + You have no way to identify when things are happening due to a trigger – they are stealthy in that sense
  + Some people also tend to chain triggers together, and if you don’t know about them or forget about them, it can make applications difficult to debug
  + Use triggers with care – be conscientious about using them, and try to find ways NOT to use them if possible