The NetBeans E-commerce Tutorial - Designing the Data Model

This tutorial unit focuses on data modeling, or the process of creating a conceptual model of your storage system by identifying and defining the entities that your system requires, and their relationships to one another. The data model should contain all the logical and physical design parameters required to generate a script using the Data Definition Language (DDL), which can then be used to create a database.[1]

In this unit, you work primarily with MySQL Workbench, a graphical tool that enables you to create data models, reverse-engineer SQL scripts into visual representations, forward-engineer data models into database schemata, and synchronize models with a running MySQL database server.

You begin by creating an entity-relationship diagram to represent the data model for the `AffableBean` application. When you have completed identifying and defining all entities and the relationships that bind them, you use Workbench to forward-engineer and run a DDL script that converts the data model into a database schema. Finally, you connect to the new schema from the NetBeans IDE.

You can view a live demo of the application that you build in this tutorial: NetBeans E-commerce Tutorial Demo Application.

Software or Resource	Version Required
NetBeans IDE	Java bundle, 6.8 or 6.9
Java Development Kit (JDK)	version 6
MySQL database server	version 5.1
MySQL Workbench	version 5.1 or 5.2

Notes:

- The NetBeans IDE requires the Java Development Kit (JDK) to run properly. If you do not have any of the resources listed above, the JDK should be the first item that you download and install.

- The NetBeans IDE Java Bundle includes Java Web and EE technologies, which are required for the application you build in this tutorial.

- You can download the complete DDL script that MySQL Workbench generates from the entity-relationship diagram you create in this tutorial: affablebean_schema_creation.sql.

Identifying Entities for the Data Model

In the real world, you may not have the luxury of designing the data model for your application. For example, your task may be to develop an application on top of an existing database system. Provided you do not have a data model to base your application on, creating one should be one of the first design steps you take before embarking on development. Creating a data model involves identifying the objects, or *entities*, required by your system and defining the relationships between them.

To begin identifying the entities we need for the data model, re-examine the use-case presented in Designing the Application. Search for commonly-occurring nouns. For example:

Use-Case

Customer visits the welcome page and selects a product **category**. **Customer** browses **products** within the selected category page, then adds a **product** to his or her **shopping cart**. **Customer** continues shopping and selects a different **category**. **Customer** adds several **products** from this **category** to **shopping cart**. **Customer** selects 'view cart' option and updates quantities for cart **products** in the cart page. **Customer** verifies shopping cart contents and proceeds to checkout. In the checkout page, **customer** views the cost of the **order** and other information, fills in personal data, then submits his or her details. The **order** is processed and **customer** is taken to a confirmation page. The confirmation page provides a unique reference number for tracking the customer **order**, as well as a summary of the **order**.

The text highlighted above in **bold** indicates the candidates that we can consider for the data model. Upon closer inspection, you may deduce that the shopping cart does not need to be included, since the data it provides (i.e., products and their quantities) is equally offered by a customer order once it is processed. In fact, as will be demonstrated in Unit 8, Managing Sessions, the shopping cart merely serves as a mechanism that retains a user session temporarily while the customer shops online. We can therefore settle on the following list:

- **customer**
- **category**
- **product**
- **order**

With these four entities, we can begin constructing an entity-relationship diagram (ERD).

Note: In this tutorial, we create a database schema from the ERD, then use the IDE's EclipseLink support to generate JPA entity classes from the existing database. (EclipseLink and the Java Persistence API (JPA) are covered in Unit 7, Adding Entity Classes and Session Beans.) This approach is described as *bottom up* development. An equally viable alternative is the *top down* approach.

- **Top down:** In *top down* development, you start with an existing Java implementation of the domain model, and have complete freedom with respect to the design of the database schema. You must create mapping metadata (i.e., annotations used in JPA entity classes), and can optionally use a persistence tool to automatically generate the schema.

- **Bottom up:** *Bottom up* development begins with an existing database schema. In this case, the easiest way to proceed is to use forward-engineering tools to extract metadata from the schema and generate annotated Java source code (JPA entity classes).

For more information on top down and bottom up design strategies, see Data modeling: Modeling methodologies [Wikipedia].

Creating an Entity-Relationship Diagram

Start by running MySQL Workbench. In this exercise, you use Workbench to design an entity-relationship diagram for the `AffableBean` application.

Note: The following instructions work for MySQL Workbench versions 5.1 *and* 5.2. The images used in this tutorial are taken from version 5.2. There are slight differences in the graphical interface between versions, however the functionality remains consistent. Because version 5.2 incorporates a query editor (previously MySQL Query Browser), as well as a server administration interface (previously MySQL Administrator), you are presented with the Home screen when opening the application (shown below).
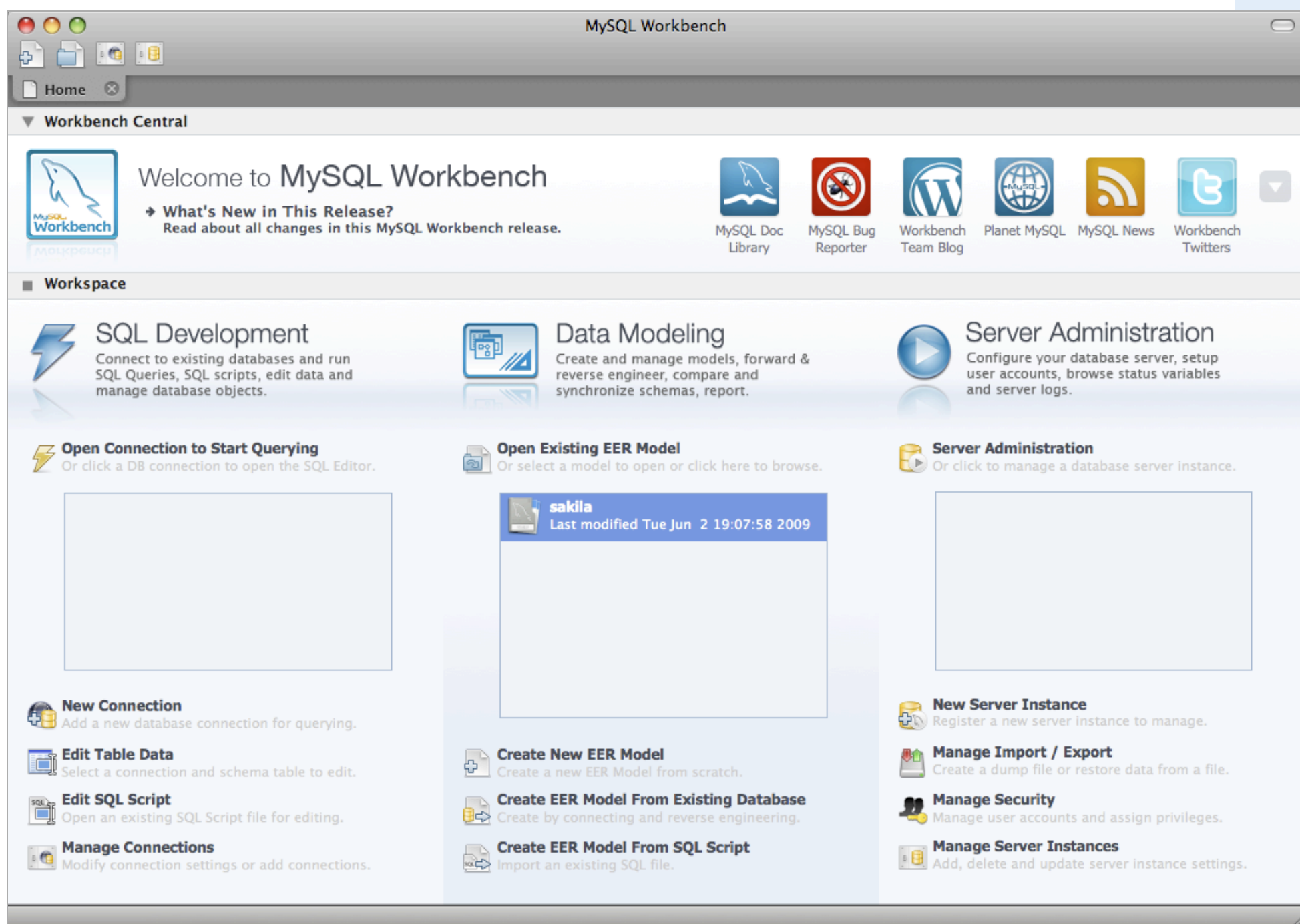
If you are working in Workbench 5.2, click **Create New EER Model** beneath the Data Modeling heading in the Home screen.

- Creating the `affablebean` Schema

- Creating Entities

- Adding Entity Properties

- Identifying Relationships

Creating the `affablebean` Schema

1. In the default interface, begin by creating a new schema which will be used with the `AffableBean` application. Click the plus (+) icon located to the right of the **Physical Schemata** heading.

   A new panel opens in the bottom region of the interface, enabling you to specify settings for the new schema.
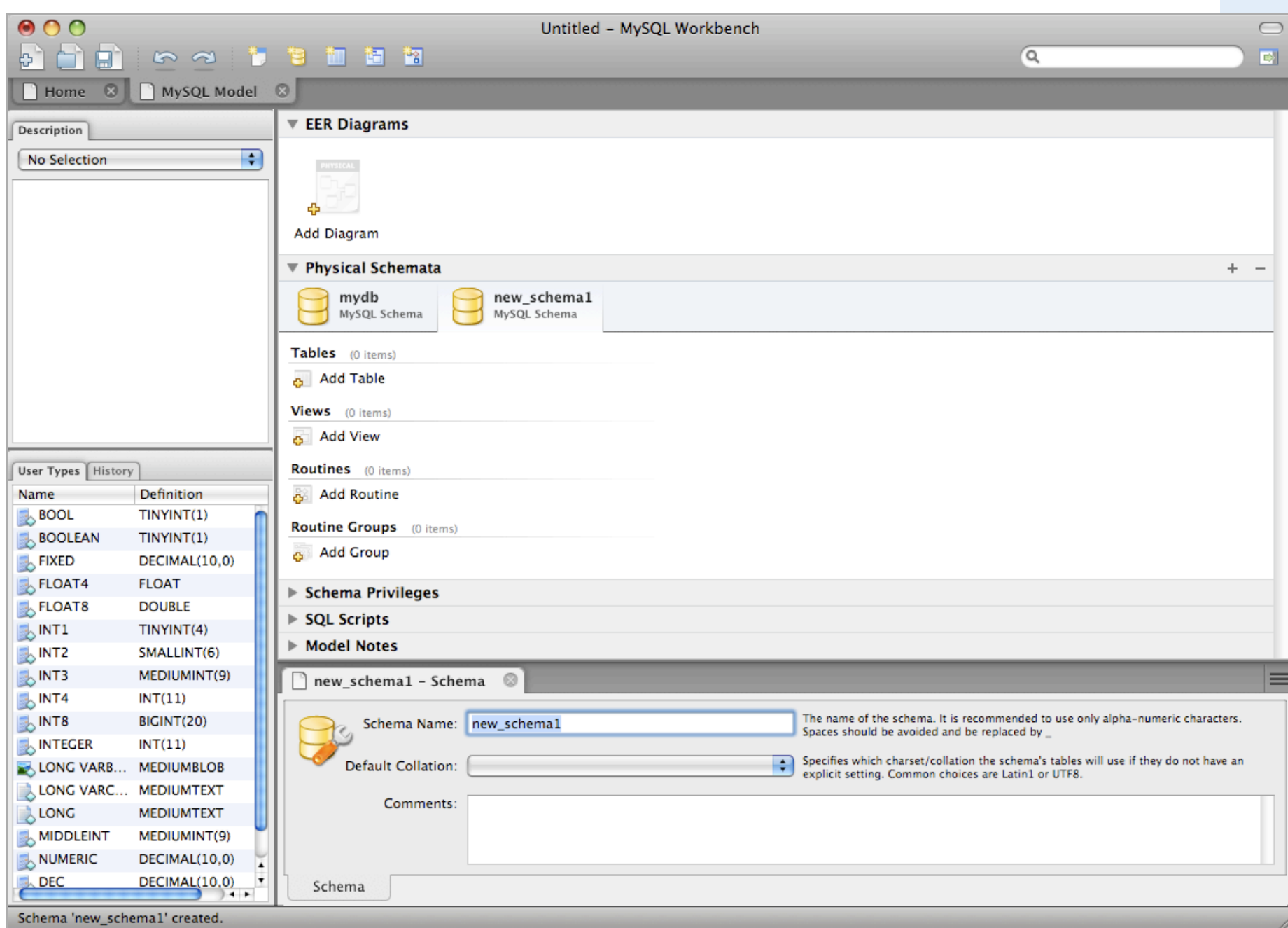
2. Enter the following settings for the new schema:

   - **Schema Name:** `affablebean`

   - **Default Collation:** `utf8 - utf8_unicode_ci`

   - **Comments:** `Schema used with the AffableBean application`
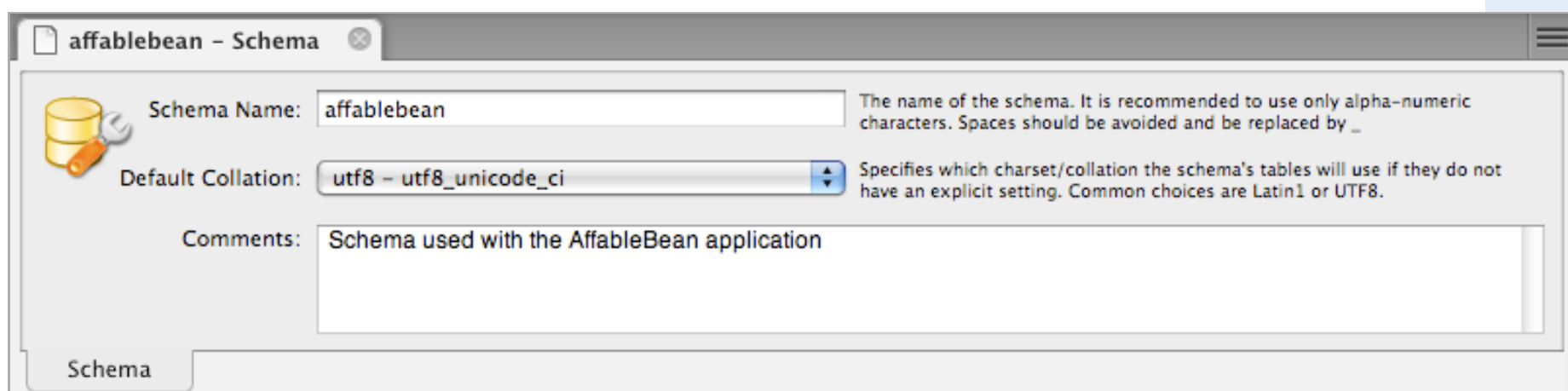
The new schema is created, and becomes listed under the Catalog tab in the right region of the Workbench interface.

TIP For an explanation of character sets and collations, see the MySQL Server Manual: 9.1.1. Character Sets and Collations in General.

Creating Entities

Start by creating a new entity-relationship diagram in MySQL Workbench. You can drag-and-drop entity tables onto the canvas.

1. Under the EER Diagrams heading in WorkBench, double-click the Add Diagram ( ) icon. A new EER Diagram opens displaying an empty canvas.

   TIP 'EER' stands for Enhanced Entity-Relationship.
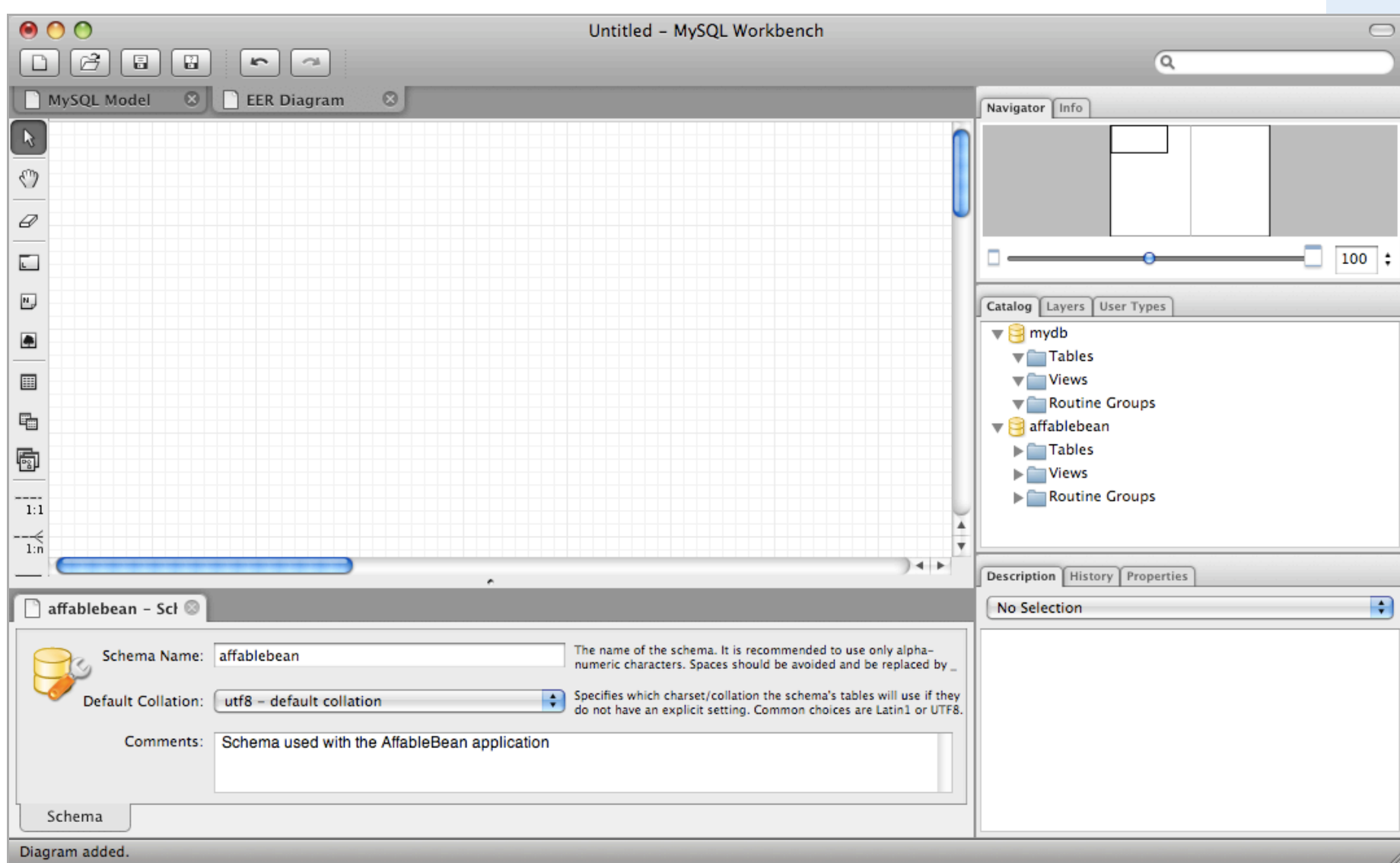
2. Click the New Table (▦) icon located in the left margin, then hover your mouse onto the canvas and click again. A new table displays on the canvas.
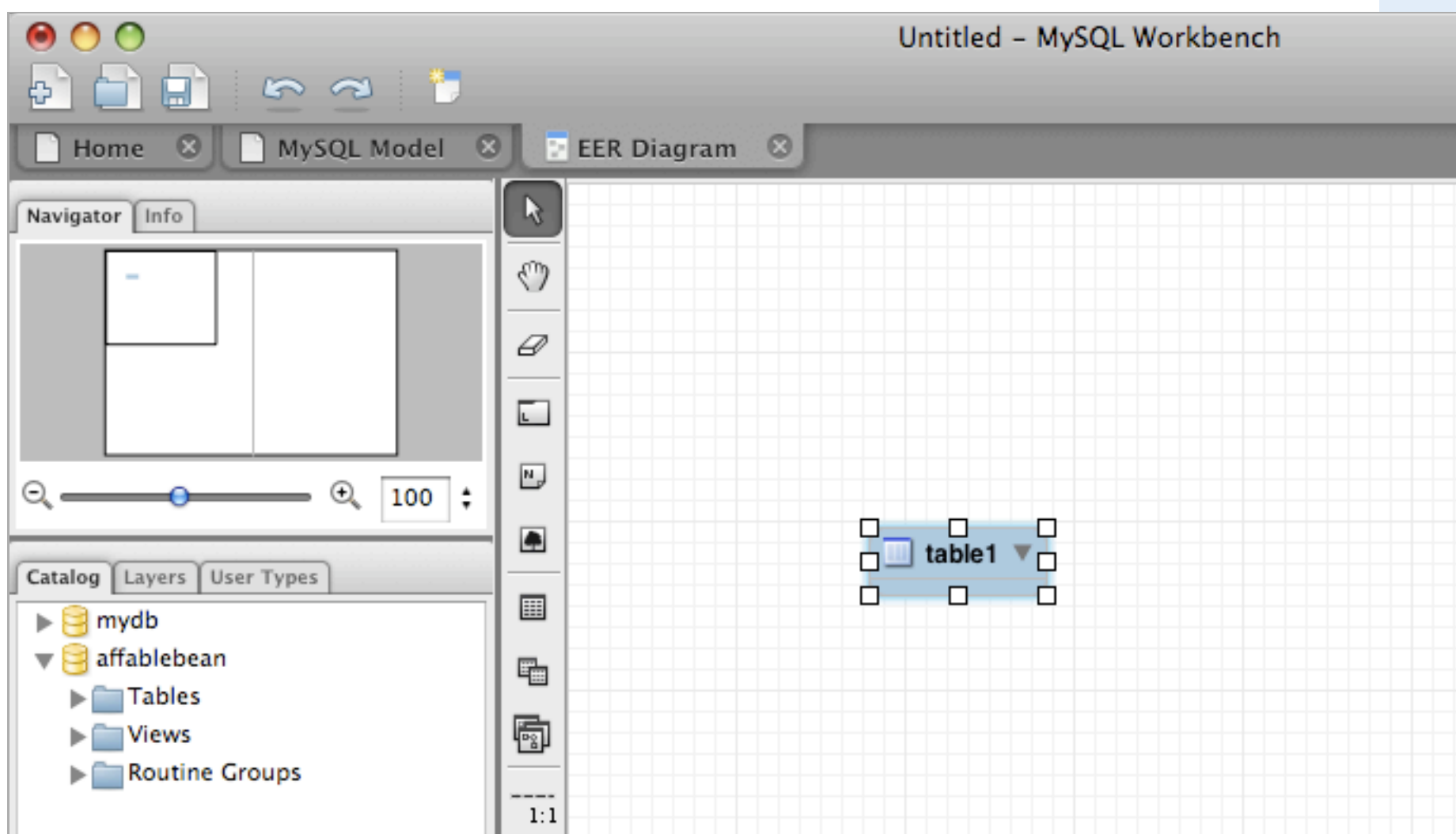

3. Double-click the table. The Table editor opens in the bottom region of the interface, allowing you to configure settings for the table.

> **Note:** The terms 'table' and 'entity' are nearly synonymous in this tutorial unit. From the point of view of a database schema, you are creating tables. From a data modeling perspective, you are creating entities. Likewise, the columns that you later create for each table correspond to entity *properties*.

4. In the Table editor, rename the table to one of the nouns you identified from the use-case above. Optionally add a comment describing the purpose of the table. For example:

- **Name:** `customer`

- **Engine:** `InnoDB`

- **Comments:** `maintains customer details`
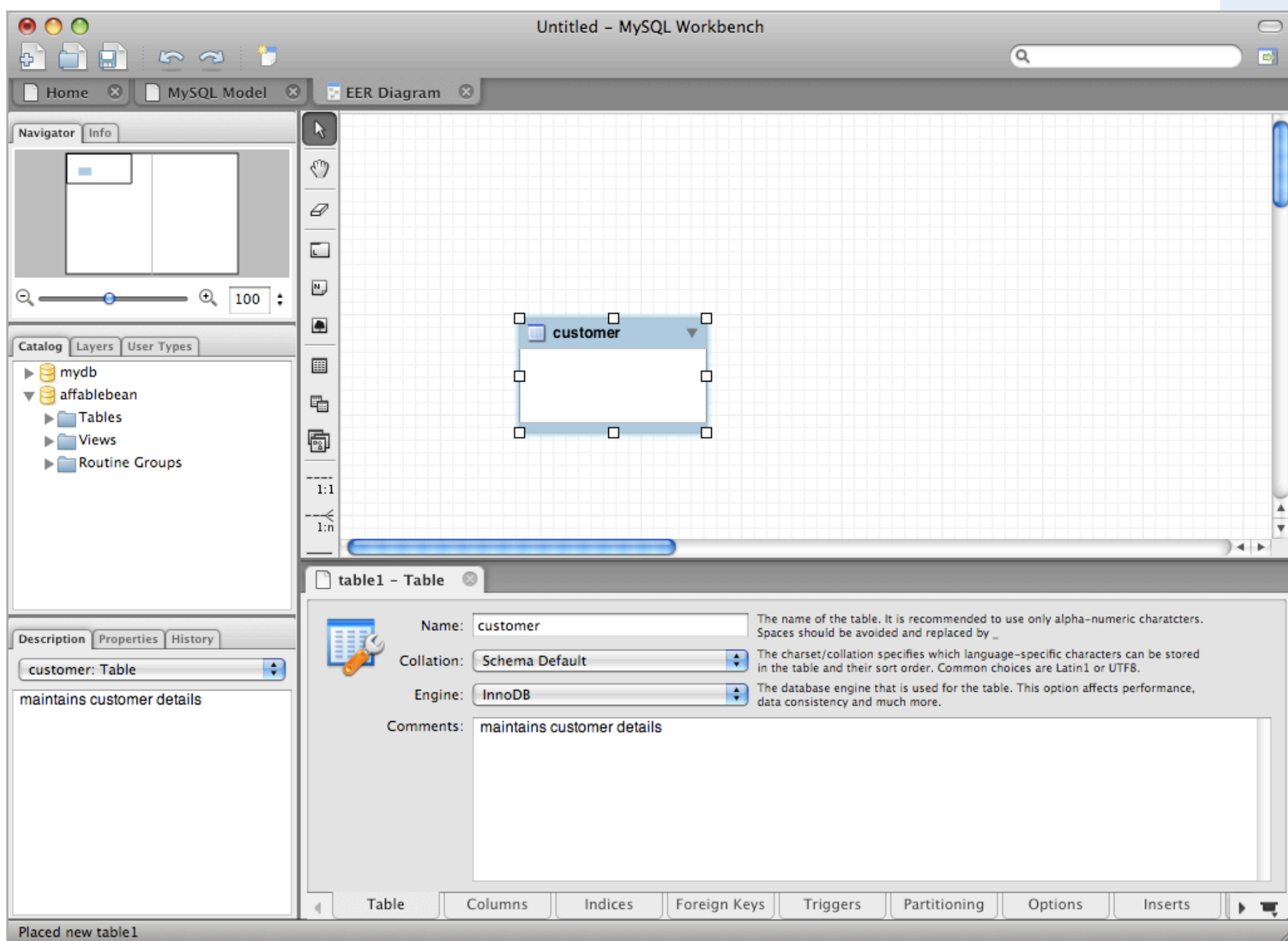
The InnoDB engine provides foreign key support, which is utilized in this tutorial. Later, under Forward-Engineering to the Database, you set the default storage engine used in Workbench to InnoDB.

5. Under the **Catalog** tab in the left region of WorkBench (right region for version 5.1), expand the `affablebean > Tables` node. The **customer** table now displays.
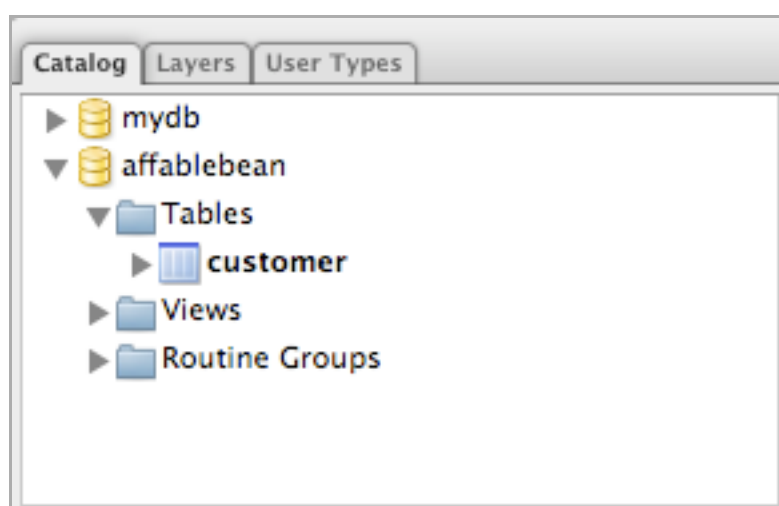
More importantly, note that the new `customer` table is now included in the `affablebean` schema. Because the `affablebean` schema was selected when you created the new EER diagram, any changes you make to the diagram are automatically bound to the schema.

6. Repeat steps 2 - 4 above to add tables to the canvas for the remaining nouns you identified in the use-case above. Before naming your tables however, there is one important consideration which you should take into account. Certain keywords hold special meaning for the SQL dialect used by the MySQL server. Unfortunately, `'order'` is one of them. (For example, `'order'` can be used in an `ORDER BY` statement.) Therefore, instead of naming your table `'order'`, name it `'customer_order'` instead. At this stage, don't worry about arranging the tables on the canvas in any special order.

For a list of reserved words used by the MySQL server, refer to the official manual: 2.2. Reserved Words in MySQL 5.1.
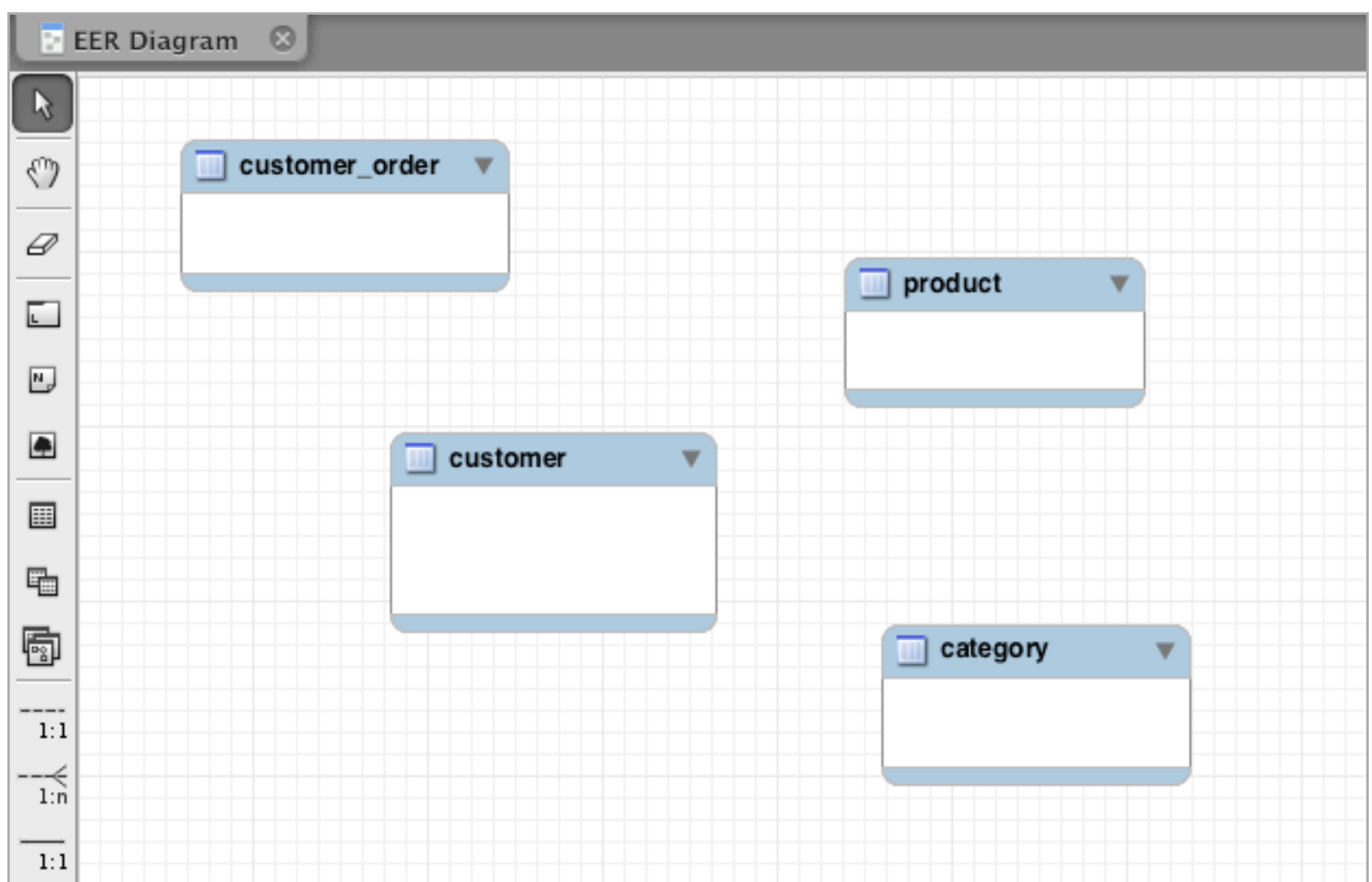
Adding Entity Properties

Now that you've added entities to the canvas, you need to specify their properties. Entity properties correspond to the columns defined in a database table. For example, consider the `customer` entity. In regard to the `AffableBean` application, what aspects of a customer would need to be persisted to the database? These would likely be all of the information gathered in the checkout page's customer details form, as well as some association to the processed order.

When adding properties, you need to determine the most appropriate data type for each property. MySQL supports a number of data types in several categories: numeric types, date and time types, and string (character) types. Refer to the official manual for a summary of data types within each category: 10.1. Data Type Overview. In this tutorial, the data types have been chosen for you. Choosing the appropriate data type plays a significant role in optimizing storage on your database server. For more information see:

- 10.5. Data Type Storage Requirements

- 10.6. Choosing the Right Type for a Column

The following steps describe how you can use MySQL Workbench to add properties to an existing entity in your ERD. As with most of the initial design steps, determining the entity properties would call for careful consideration of the business problem that needs to be solved, and could require hours of analysis as well as numerous consultations with the client.

1. Double-click the `customer` table heading to bring up the Table editor in WorkBench.

2. In the Table editor click the Columns tab, then click inside the displayed table to edit the first column. Enter the following details:

Column	Datatype	PK (Primary Key)	NN (Not Null)	UN (Unsigned)	AI (Autoincrement)
id	INT	✓	✓	✓	✓
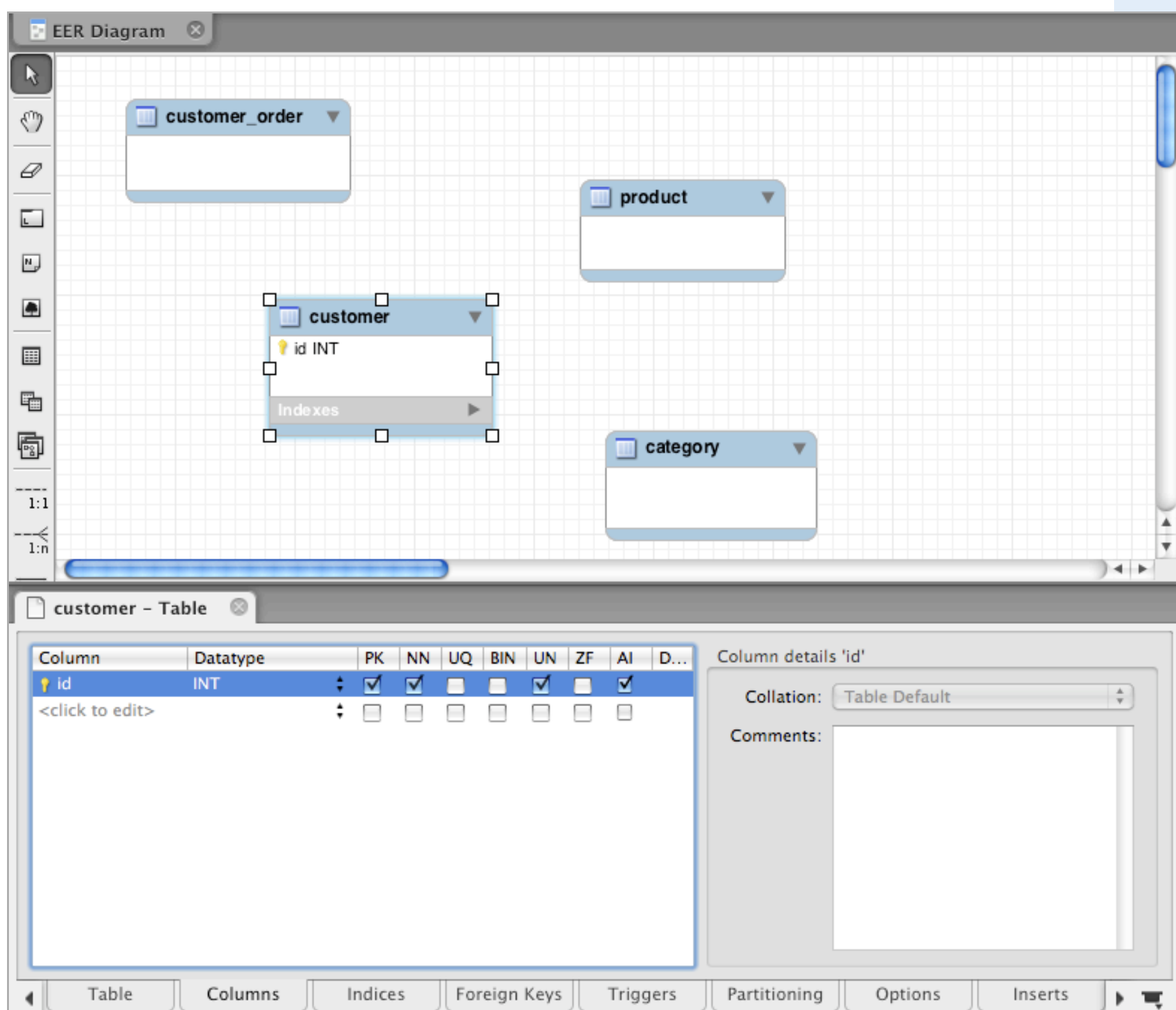
3. Continue working in the `customer` table by adding the following `VARCHAR` columns. These columns should be self-explanatory, and represent data that would need to be captured for the Affable Bean business to process a customer order and send a shipment of groceries to the customer address.

Column	Datatype	NN (Not Null)
name	VARCHAR(45)	✓
email	VARCHAR(45)	✓
phone	VARCHAR(45)	✓
address	VARCHAR(45)	✓
city_region	VARCHAR(2)	✓
cc_number	VARCHAR(19)	✓

TIP For an explanation of the `VARCHAR` data type, see the MySQL Reference Manual: 10.4.1. The CHAR and VARCHAR Types.
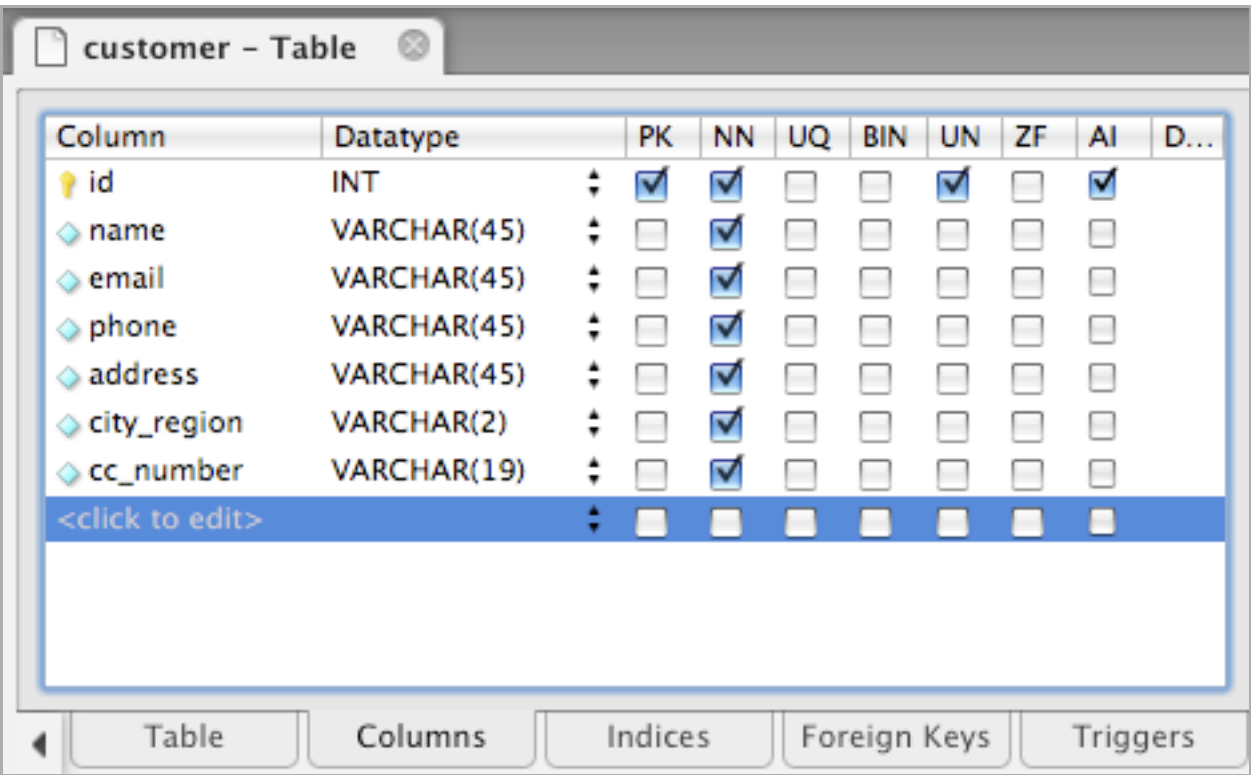
4. With the `customer` table selected on the canvas, choose Arrange > Reset Object Size to resize the table so that all columns are visible on the canvas. Also click the Indexes row so that any table indexes are also visible. (This includes primary and foreign keys, which becomes useful when you begin creating relationships between tables later in the exercise.)

When you finish, the `customer` entity looks as follows.

5. Follow the steps outlined above to create columns for the remaining tables.

category

Column	Datatype	PK	NN	UN	AI
id	TINYINT	✓	✓	✓	✓
name	VARCHAR(45)		✓		

customer_order

| Column	Datatype	PK	NN	UN	AI	Default
id	INT	✓	✓	✓	✓	
amount	DECIMAL(6,2)		✓			
date_created	TIMESTAMP		✓			CURRENT_TIMESTAMP
confirmation_number	INT		✓	✓		

product

Column	Datatype	PK	NN	UN	AI	Default
id	INT	✓	✓	✓	✓	
name	VARCHAR(45)		✓			

price	DECIMAL(5,2)	✓				
description	TINYTEXT					
last_update	TIMESTAMP	✓			CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	

> **TIP** For details on the `TIMESTAMP` data type, see the MySQL Reference Manual: 10.3.1.1. TIMESTAMP Properties.

When you finish, your canvas will look similar to the following.
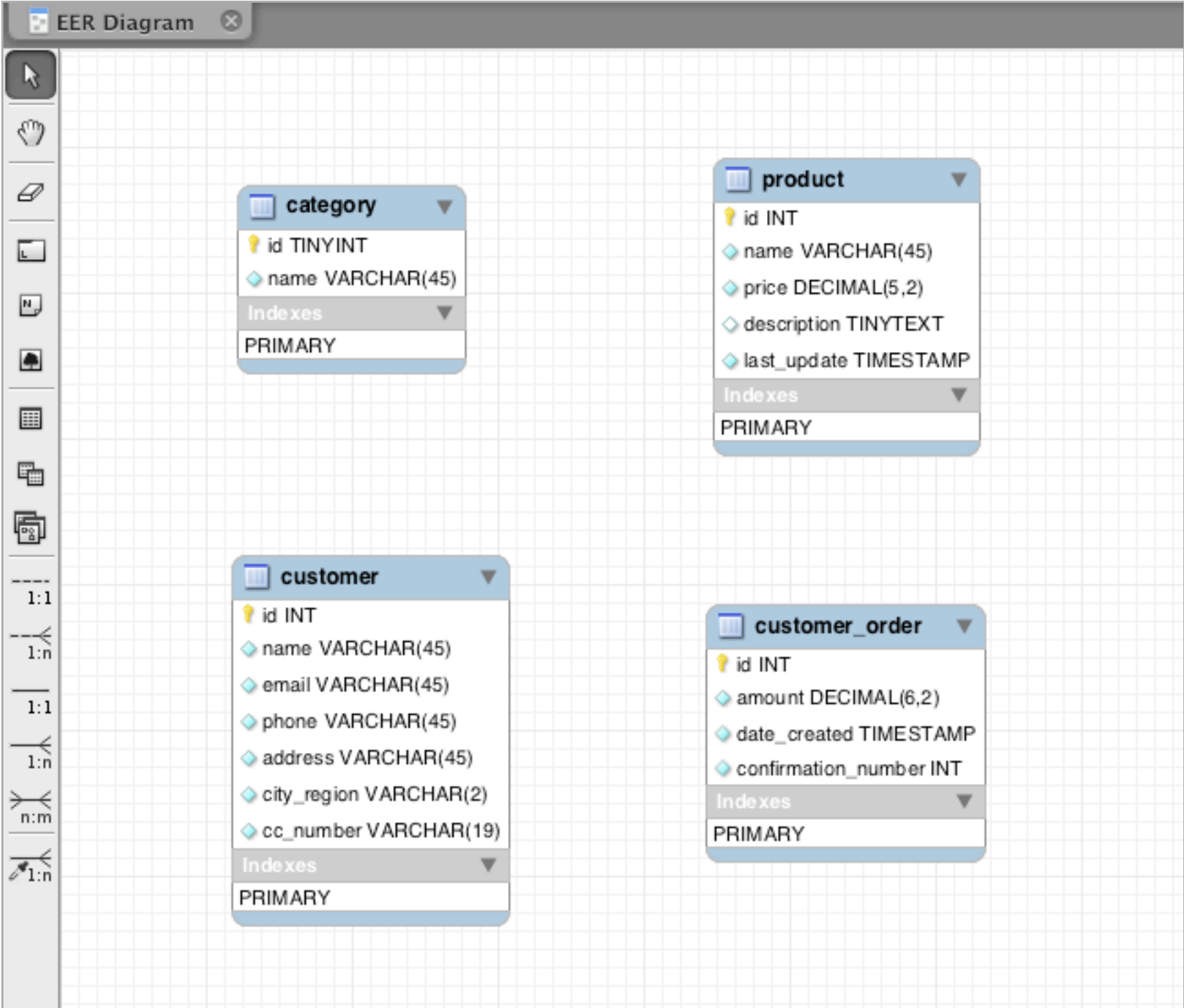
Identifying Relationships

So far, the entity-relationship diagram contains several entities, but lacks any relationships between them. The data model that we are creating must also indicate whether objects are aware of (i.e., contain references to) one another. If one object contains a reference to another object, this is known as a *unidirectional* relationship. Likewise, if both objects refer to each other, this is called a *bidirectional* relationship.

References correlate to foreign keys in the database schema. You will note that, as you begin linking tables together, foreign keys are added as new columns in the tables being linked.

Two other pieces of information are also commonly relayed in an ERD: *cardinality* (i.e., multiplicity) and *ordinality* (i.e., optionality). These are discussed below, as you begin adding relationships to entities on the canvas. In order to complete the ERD, you essentially need to create two *one-to-many* relationships, and one *many-to-many* relationship. Details follow.

- Creating One-To-Many Relationships
- Creating Many-To-Many Relationships

Creating One-To-Many Relationships

Examine the four objects currently on the canvas while considering the business problem. You can deduce the following two *one-to-many* relationships:

- A category must contain one or more products

- A customer must have placed one or more orders

Incorporate these two relationships into the ERD. You can download a copy of the MySQL Workbench project that contains the four entities required for the following steps: affablebean.mwb.

1. In the left margin, click the 1:n Non-Identifying Relationship (`1:n`) button. This enables you to create a *one-to-many* relationship.

2. Click the `product` table, then click the `category` table. The first table you click will contain the foreign key reference to the second table. Here, we want the `product` table to contain a reference to `category`. In the image below, you see that a new column, `category_id`, has been added to the `product` table, and that a foreign key index, `fk_product_category` has been added to the table's indexes.
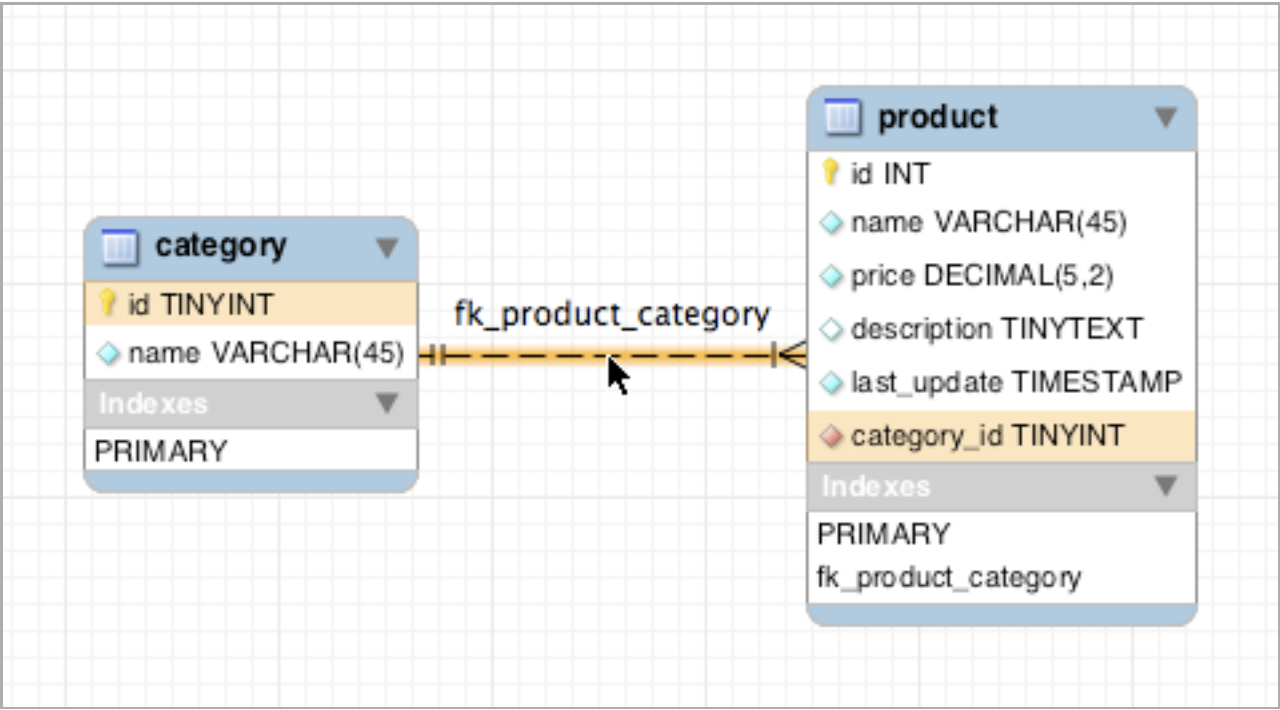
Since foreign keys must be of the same data type as the columns they reference, notice that `category_id` is of type `TINYINT`, similar to the `category` table's primary key.

TIP The entity-relationship diagram in this tutorial uses Crow's Foot notation. You can alter the relationship notation in WorkBench by choosing Model > Relationship Notation.

3. Double-click the relationship (i.e., click the dashed line between the two entities). The Relationship editor opens in the bottom region of the interface.

4. Change the default caption to '`belongs to`'. In other words, "product x belongs to category y." Note that this is a *unidirectional* relationship: A `product` object contains a reference to the category it belongs to, but the related `category` object does not contain any references to the products it contains.

5. Click the Foreign Key tab in the Relationship editor. You see the following display.
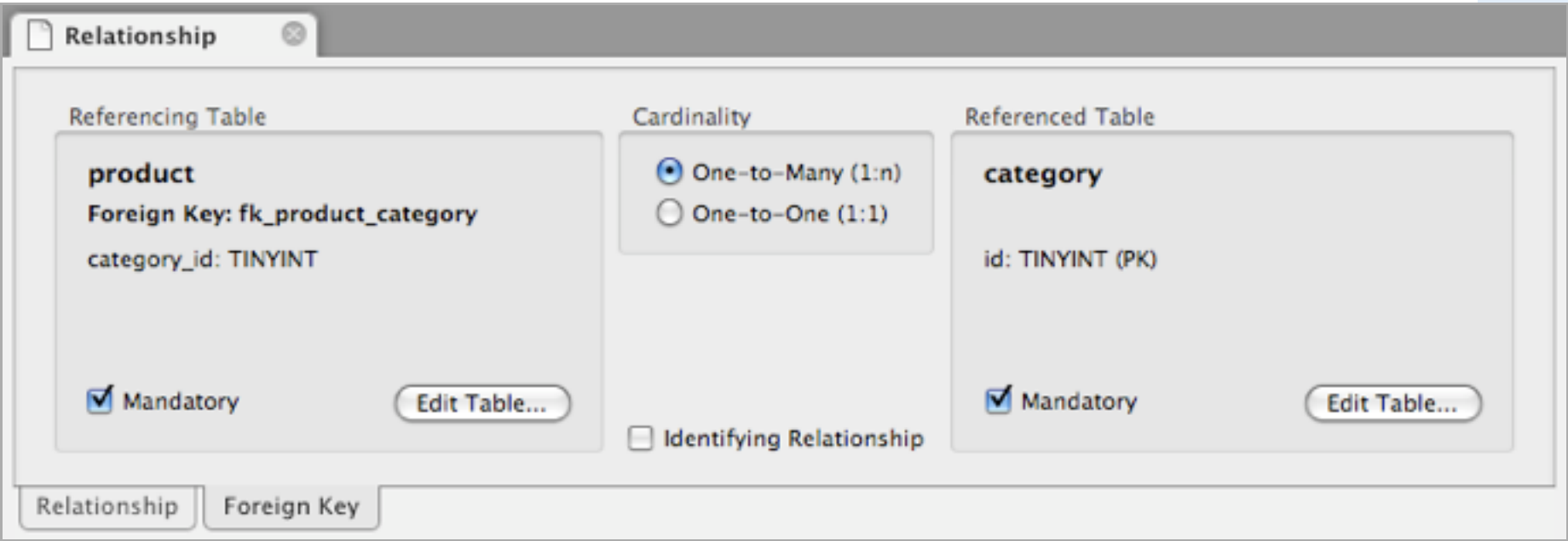
Under the Foreign key tab, you can modify a relationship's:

- **cardinality:** whether the relationship between two objects is *one-to-one* or *one-to-many*.

- **ordinality:** whether a reference between entities must exist in order to maintain the integrity of the model. (Toggle the Mandatory checkbox for either side.)

- **type:** (i.e., *identifying* or *non-identifying*). A non-identifying relationship, such as this one, refers to the fact that the

child object (`product`) can be identified independently of the parent (`category`). An identifying relationship means that the child cannot be uniquely identified without the parent. An example of this is demonstrated later, when you create a many-to-many relationship between the `product` and `order` tables.

6. Click the 1:n Non-Identifying Relationship ( ) button. In the following steps, you create a *one-to-many* relationship between the `customer` and `customer_order` objects.

7. Click the `order` table first (this table will contain the foreign key), then click the `customer` table. A relationship is formed between the two tables.

8. Click the link between the two tables, and in the Relationship editor that displays, change the default caption to `'is placed by'`. The relationship now reads, "customer order x is placed by customer y."
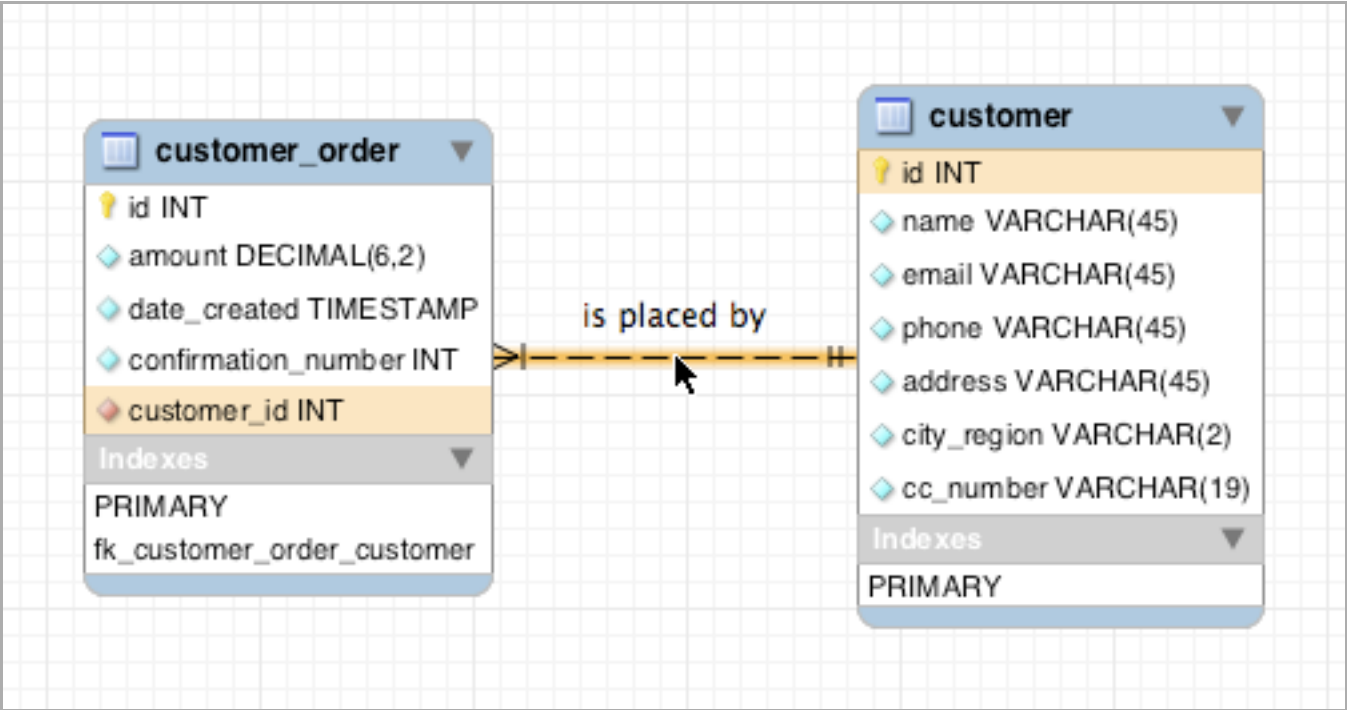

TIP You can click and drag tables on the canvas into whatever position makes the most sense for your model. In the image above, the `order` table has been moved to the left of `customer`.

Creating Many-To-Many Relationships

Many-to-many relationships occur when both sides of a relationship can have numerous references to related objects. For example, imagine the Affable Bean business offered products that could be listed under multiple categories, such as cherry ice cream, sausage rolls, or avocado soufflé. The data model would have to account for this by including a *many-to-many* relationship between `product` and `category`, since a category contains multiple products, and a product can belong to multiple categories.

In order to implement a *many-to-many* relationship in a database, it is necessary to break the relationship down into two *one-to-many* relationships. In doing so, a third table is created containing the primary keys of the two original tables. The `product -category` relationship described above might look as follows in the data model.
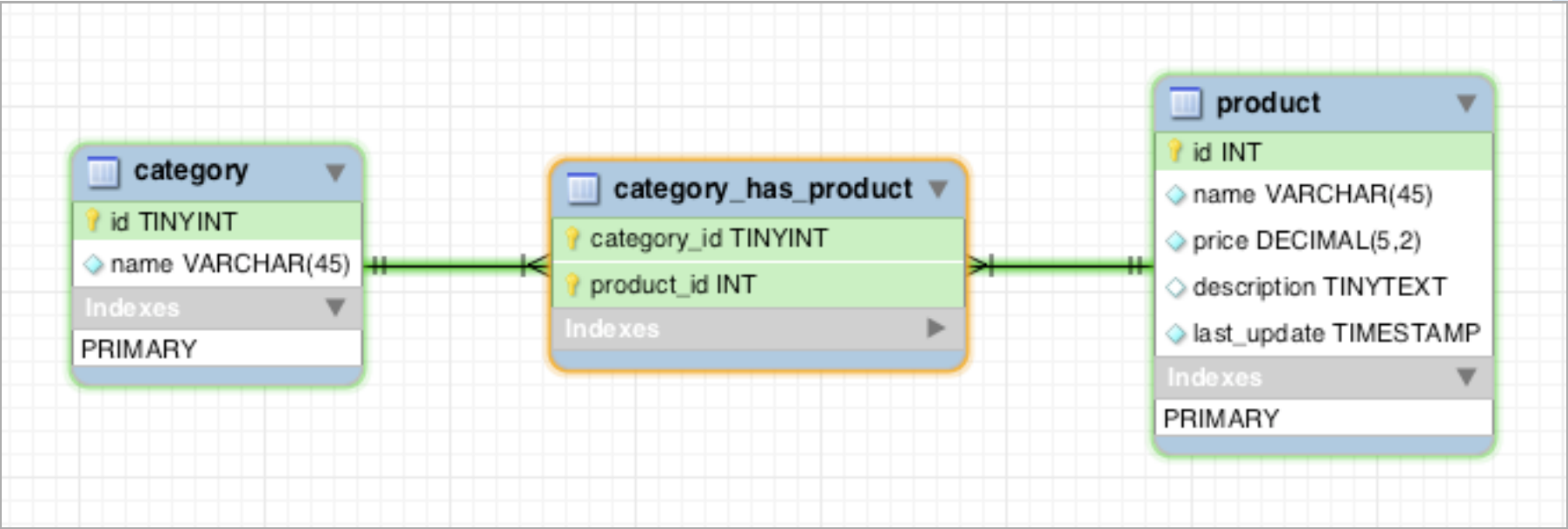

Now, consider how the application will persist customer orders. The `customer_order` entity already contains necessary properties, such as the date it is created, its confirmation number, amount, and a reference to the customer who placed it. However, there currently is no indication of the products contained in the order, nor their quantities. You can resolve this by creating a *many-to-many* relationship between `customer_order` and `product`. This way, to determine which products are contained in a given order, the application's business logic can query the new table that arises from the many-to-many relationship, and search for all records that match an `order_id`. Because customers can specify quantities for products in their

shopping carts, we can also add a `quantity` column to the table.

1. In the left margin, click the n:m Identifying Relationship (⋈ `n:m`) button. This enables you to create a *many-to-many* relationship.

2. Click the `customer_order` table, then click the `product` table. A new table appears, named `customer_order_has_product`.

   TIP Recall that an *identifying relationship* means that the child cannot be uniquely identified without the parent. Identifying relationships are indicated on the Workbench canvas by a solid line linking two tables. Here, the `customer_order_has_product` table forms an identifying relationship with its two parent tables, `customer_order` and `product`. A record contained in the `customer_order_has_product` table requires references from both tables in order to exist.

3. Arrange the tables according to the following image. The *many-to-many* relationship is highlighted below.
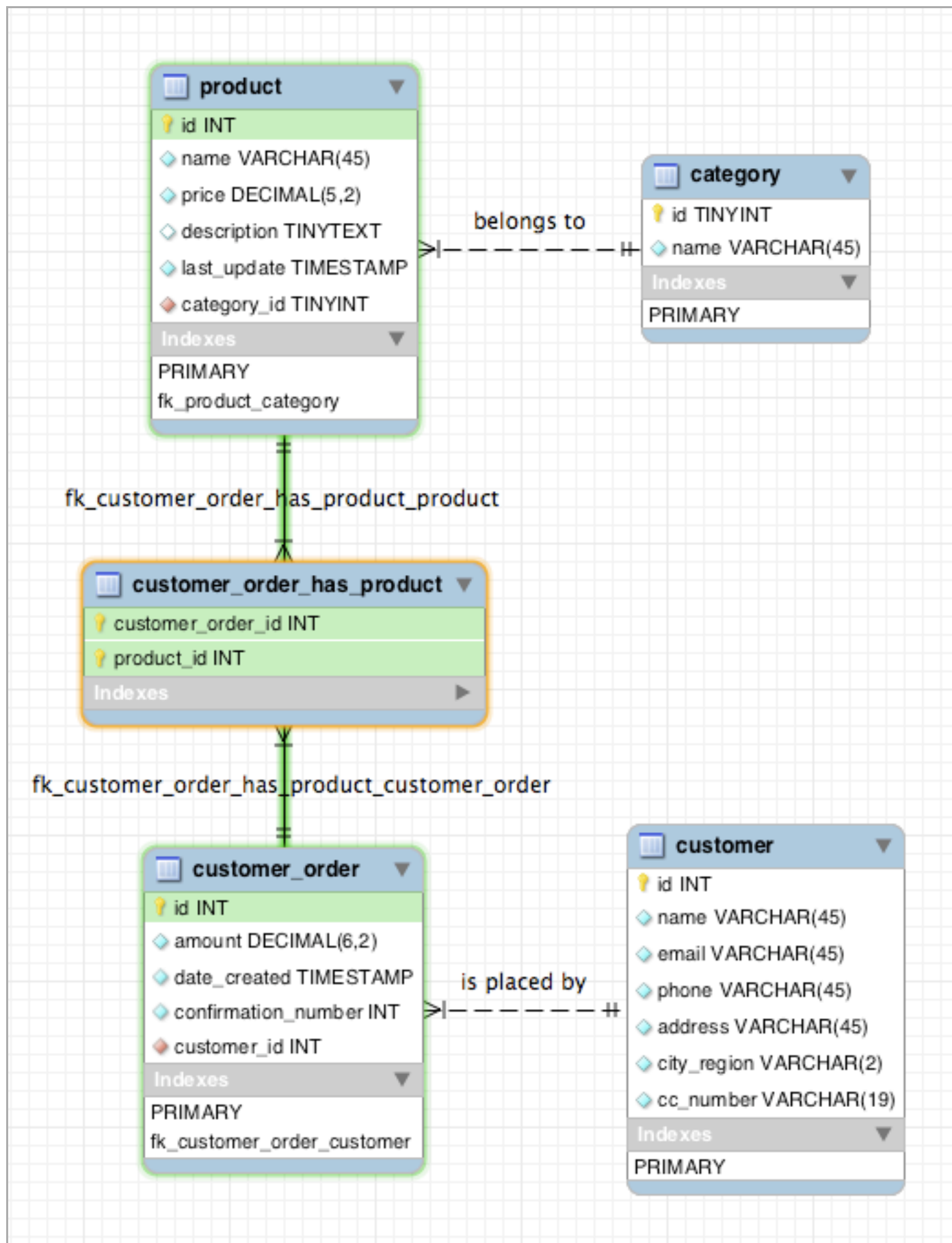

The new `customer_order_has_product` table contains two foreign keys, `fk_customer_order_has_product_customer_order` and `fk_customer_order_has_product_product`, which reference the primary keys of the `customer_order` and `product` tables, respectively. These two foreign keys form a composite primary key for the `customer_order_has_product` table.

4. Change the name of the new `customer_order_has_product` table to 'ordered_product'. Double-click the `customer_order_has_product` table to open the Table editor. Enter `ordered_product` into the Name field.

5. Rename the foreign key indexes to correspond to the new table name. In the `ordered_product`'s Table editor, click

the Foreign Keys tab. Then, click into both foreign key entries and replace 'customer_order_has_product' with 'ordered_product'. When you finish, the two entries should read:

- fk_**ordered_product**_customer_order

- fk_**ordered_product**_product
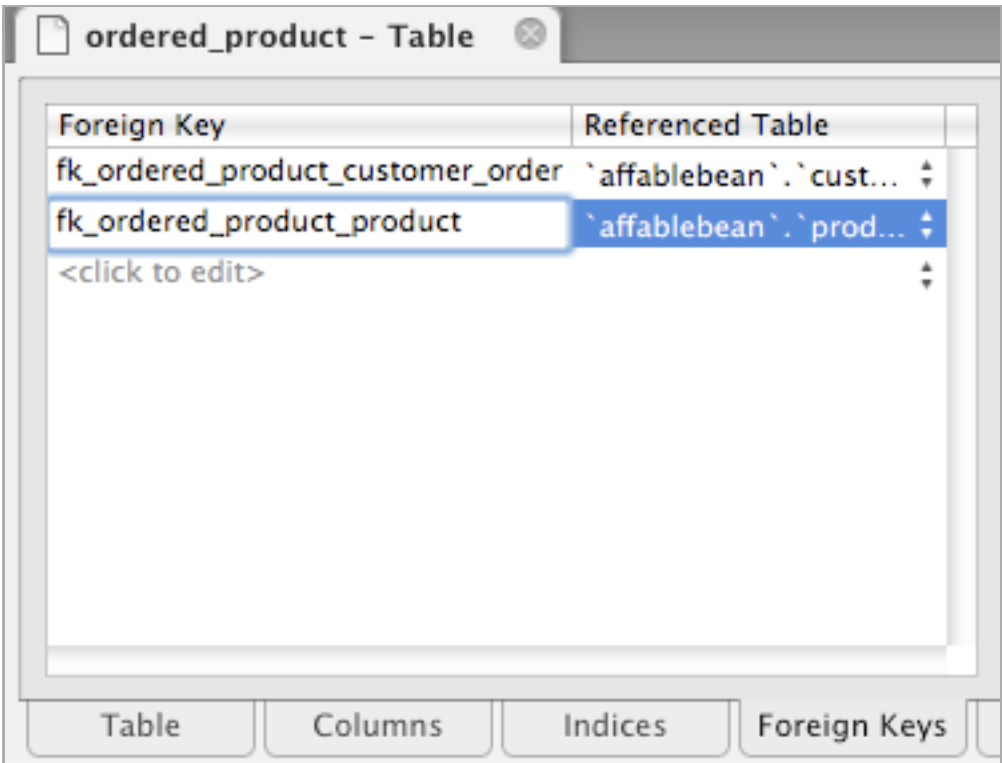
6. Double-click the lines between the two objects and delete the default captions in the Relationship editor.

7. Create a `quantity` column in the `ordered_product` table. To do so, click the Columns tab in the `ordered_product`'s Table editor. Enter the following details.

Column	Datatype	NN (Not Null)	UN (Unsigned)	Default
quantity	SMALLINT	✓	✓	1
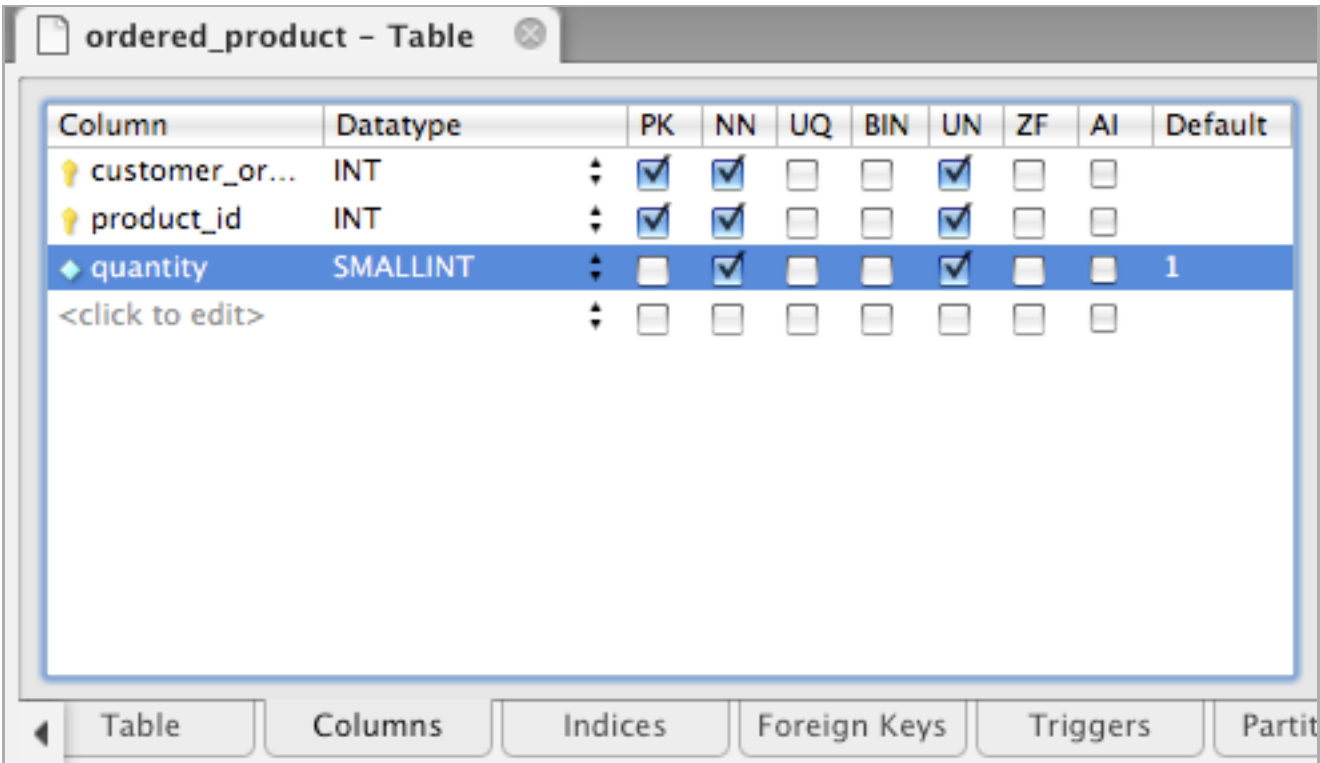
You have now completed the ERD (entity-relationship diagram). This diagram represents the data model for the `AffableBean` application. As will later be demonstrated, the JPA entity classes that you create will be derived from the entities existing in the data model.

Data Model for the
AffableBean Application
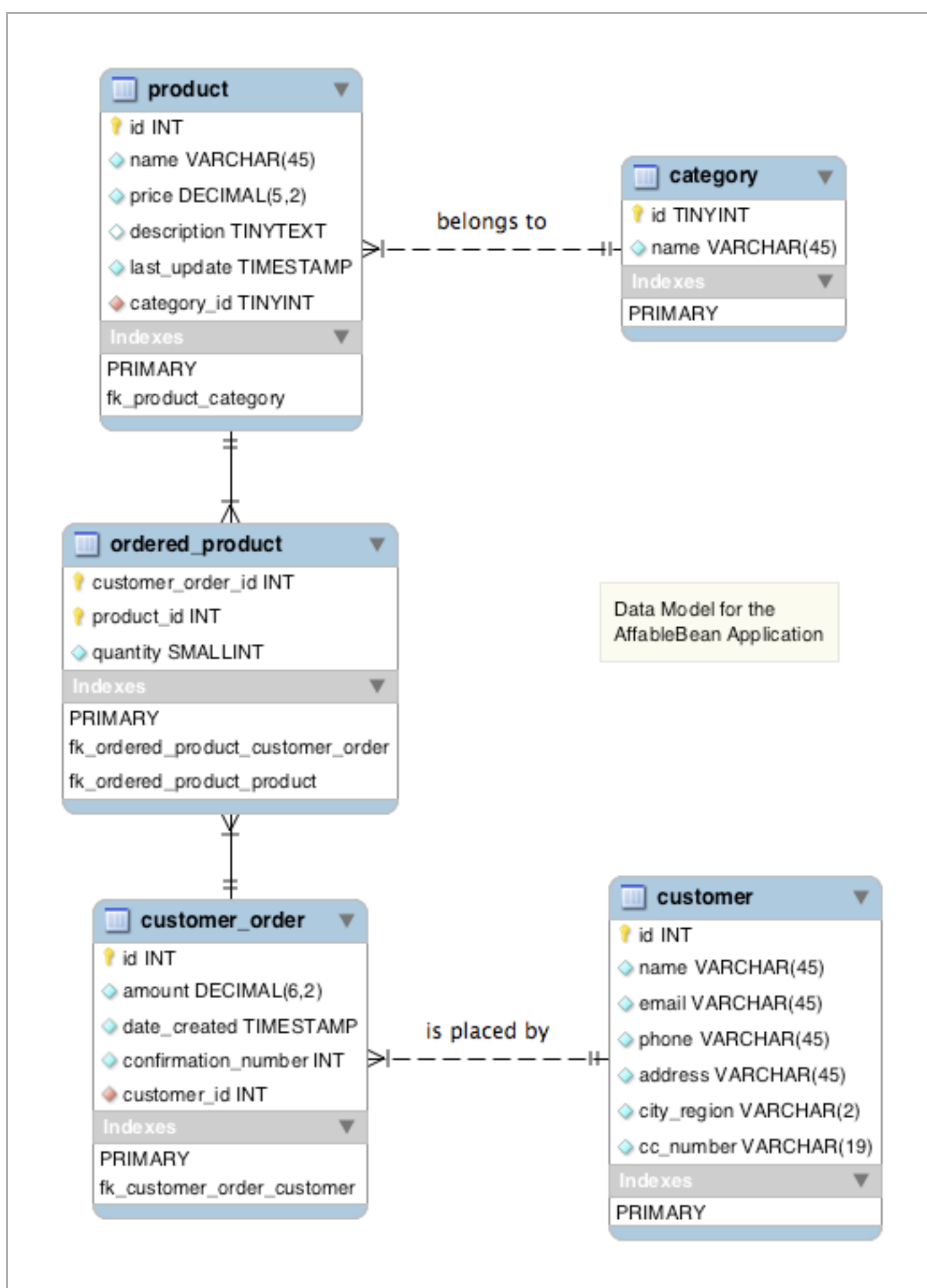
TIP Choose View > Toggle Grid to disable the canvas grid. You can also create notes for your diagram using the New Text Object ( ) button in the left margin.

Forward-Engineering to the Database

To incorporate the data model you created into the MySQL database, you can employ WorkBench to forward-engineer the diagram into an SQL script (more precisely, a DDL script) to generate the schema. The wizard that you use also enables you to immediately run the script on your database server.

Important: Make sure your MySQL database server is running. Steps describing how to setup and run the database are provided in Setting up the Development Environment: Communicating with the Database Server.

1. Set the default storage engine used in Workbench to InnoDB. Choose Tools > Options (MySQLWorkbench > Preferences on Mac) to open the Workbench Preferences window. Click the MySQL tab, then select InnoDB as the default storage engine.
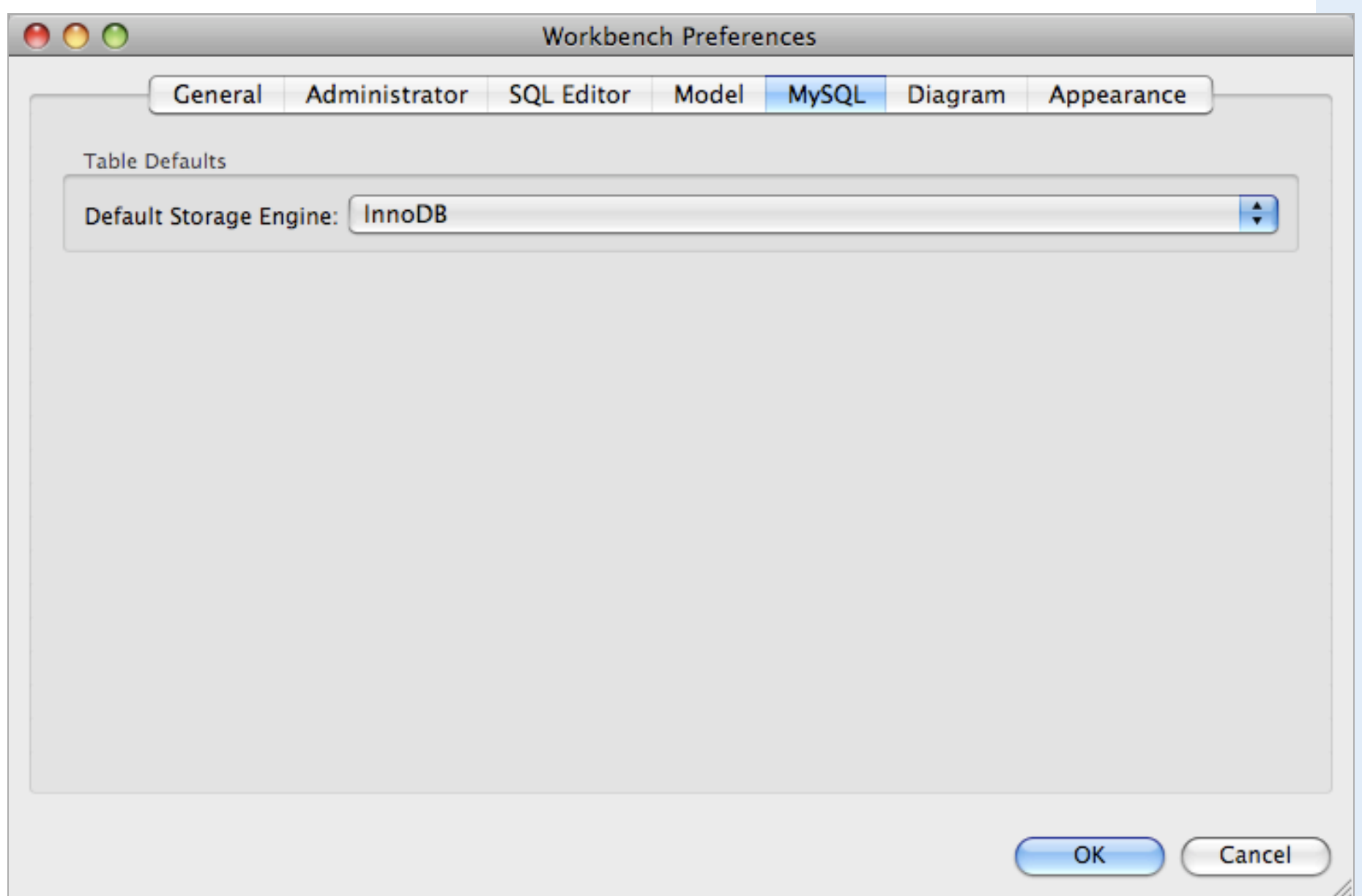
The InnoDB engine provides foreign key support, which is utilized in this tutorial.

2. Click OK to exit the Preferences window.

3. Choose Database > Forward Engineer from the main menu.

4. In the first panel of the Forward Engineer to Database wizard, select 'DROP Objects Before Each CREATE Object', and 'Generate DROP SCHEMA'.
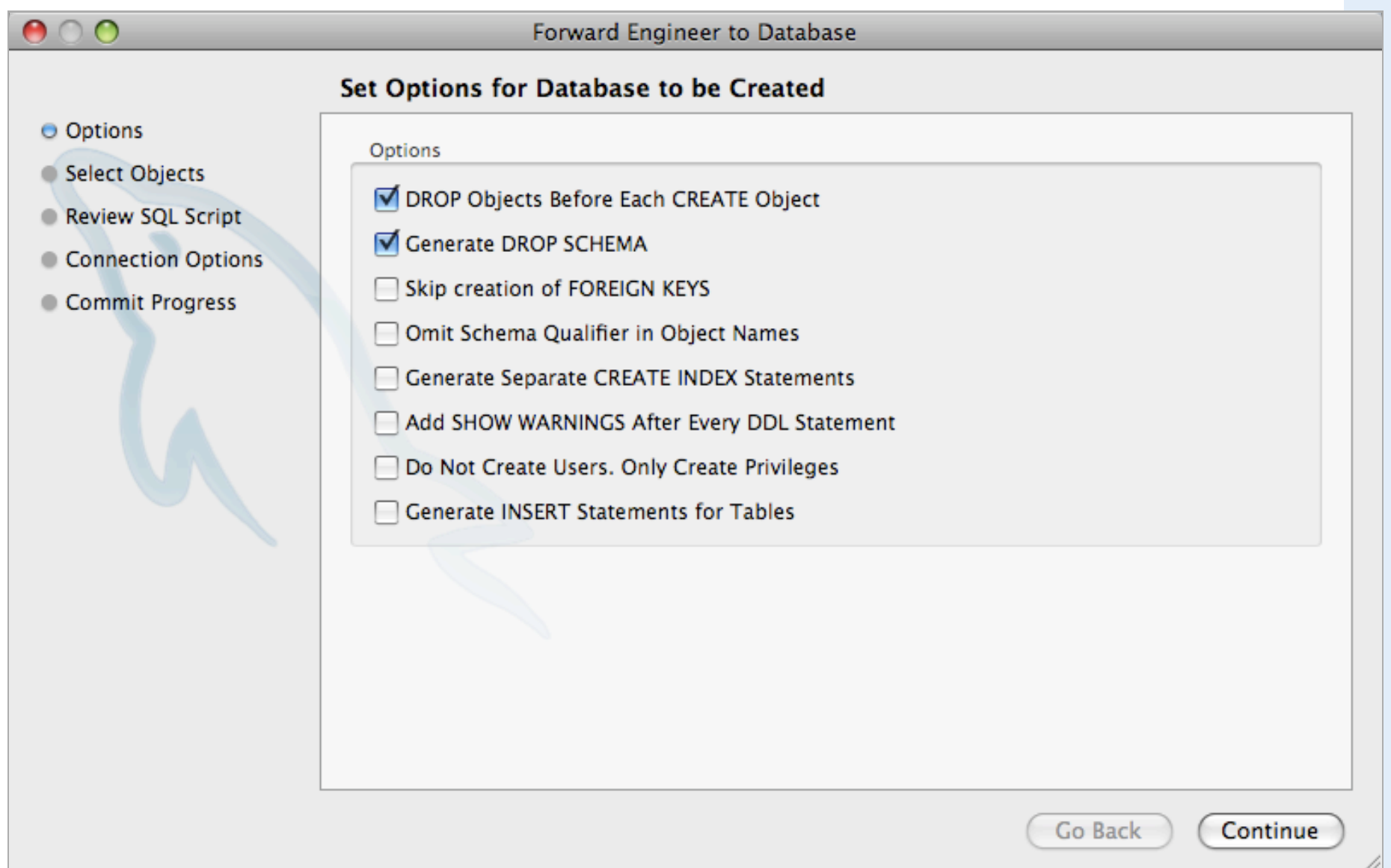
These DROP options are convenient for prototyping - if you need to make changes to the schema or schema tables, the script will first delete (i.e., *drop*) these items before recreating them. (If you attempt to create items on the MySQL server that already exist, the server will flag an error.)

5. Click Continue. In Select Objects to Forward Engineer panel, note that the Export MySQL Table Objects option is selected by default. Click the Show Filter button and note that all five tables in the affablebean schema are included.

6. Click Continue. In the Review SQL Script panel, you can examine the SQL script that has been generated based on the data model. Optionally, click Save to File to save the script to a location on your computer.

> **Note:** In examining the script, you may notice that the following variables are set at the top of the file:

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL';
```

For an explanation of what these variables are, and their purpose in the script, see the official Workbench manual: Chapter 11. MySQL Workbench FAQ.

7. Click Continue. In the Connection Options panel, set the parameters for connecting to the running MySQL server.

- **Hostname:** `127.0.0.1` (*or* `localhost`)

- **Port:** `3306`

- **Username:** `root`

- **Password:** `nbuser`

(The parameters you set should correspond to those from Setting up the Development Environment: Communicating with the Database Server.)

8. Click Execute. In the final panel of the wizard, you receive confirmation that the wizard was able to connect to and execute the script successfully.

9. Click Close to exit the wizard.

The `affablebean` schema is now created and exists on your MySQL server. In the next step, you connect to the schema, or *database*, from the IDE. At this stage you may ask, "What's the difference between a schema and a database?" In fact, the MySQL command `CREATE SCHEMA` is a synonym for `CREATE DATABASE`. (See 12.1.10. CREATE DATABASE Syntax.) Think of a schema as a blueprint that defines the contents of the database, including tables, relationships, views, etc. A database implements the schema by containing data in a way that adheres to the structure of the schema. This is similar to the object-oriented world of Java classes and objects. A class defines an object. When a program runs however, objects (i.e., class instances) are created, managed, and eventually destroyed as the program runs its course.

Connecting to the Database from the IDE

Now that the `affablebean` schema exists on your MySQL server, ensure that you can view the tables you created in the ERD from the IDE's Services window.

Important: Make sure that you have followed the steps outlined in Setting up the Development Environment: Communicating with the Database Server. This heading describes how to run the MySQL database server, register it with the IDE, create a database instance, and form a connection to the instance from the IDE.

1. In the IDE, open the Services window (Ctrl-5; ⌘-5 on Mac) and locate the database connection node ( ) for the `affablebean` database instance you created in the previous tutorial unit.

2. Refresh the connection to the `affablebean` database. To do so, right-click the connection node and choose Refresh.

3. Expand the Tables node. You can now see the five tables defined by the schema.

4. Expand any of the table nodes. Each table contains the columns and indexes that you created when working in MySQL Workbench.
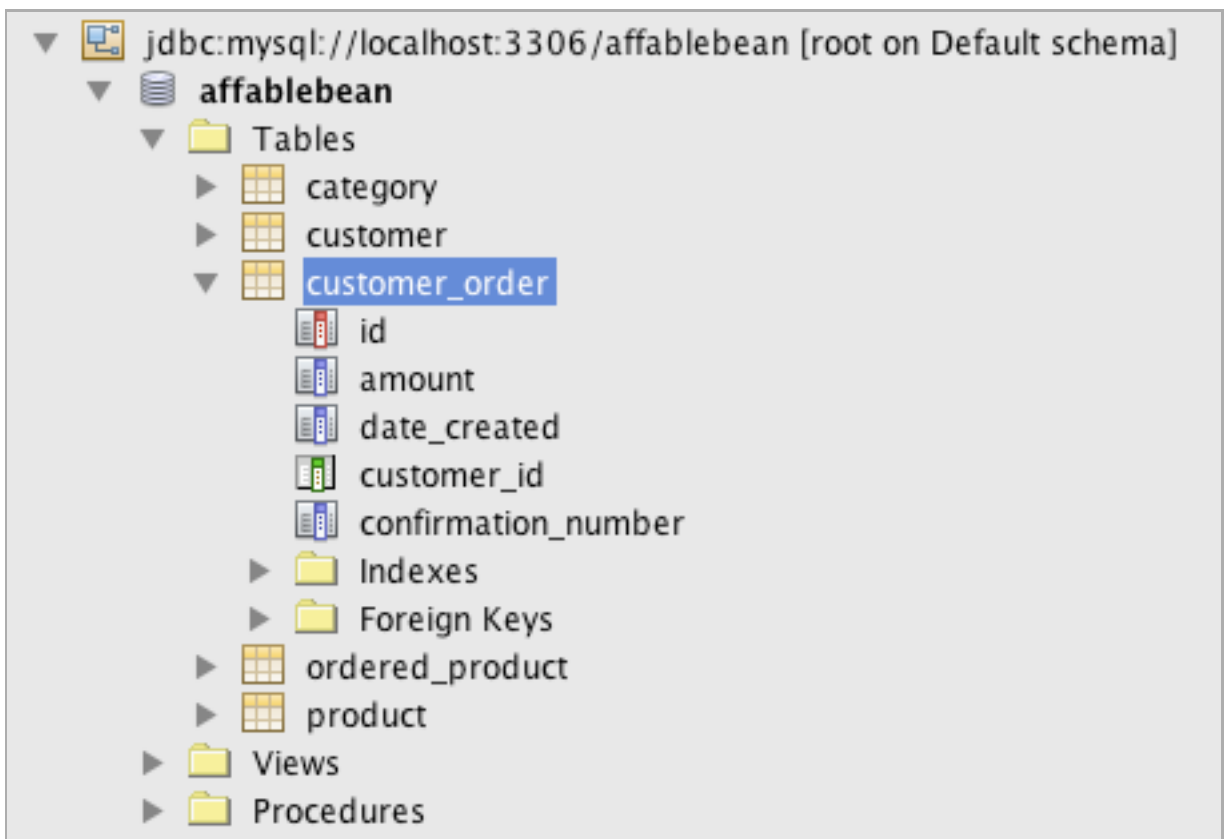
The IDE is now connected to a database that uses the schema you created for the `AffableBean` application. From the IDE, you can now view any table data you create in the database, as well as directly modify, add and delete data. You will explore some of these options later, in Connecting the Application to the Database, after you've added sample data to the database.

Send Us Your Feedback

See Also

NetBeans Resources

- MySQL and NetBeans IDE

- Connecting to a MySQL Database

- Creating a Simple Web Application Using a MySQL Database

- Screencast: Database Support in NetBeans IDE

MySQL & Data Modeling Resources

- MySQL Workbench Blog

- MySQL Workbench Forum

- The MySQL Community Librarian

- MySQL Workbench Reference Manual

- MySQL 5.1 Reference Manual

- InnoDB [Wikipedia]

- Database Model [Wikipedia]

- Data Modeling [Wikipedia]

References

1. ^ Data Definition Language (DDL) is a subset of the SQL language and includes statements such as `CREATE TABLE`, `DROP`, and `ALTER`. Other subsets include Data Manipulation Language (DML), and Data Control Language (DCL). For more information, see Data Definition Language [Wikipedia].