Challenges in database design with Microsoft Access

Jerzy Letkowski
Western New England University

ABSTRACT

Design, development and explorations of databases are popular topics covered in introductory courses taught at business schools. Microsoft Access is the most popular software used in those courses. Despite quite high complexity of Access, it is considered to be one of the most friendly database programs for beginners. A typical Access textbook teaches students first how to design and build databases. Next the students are exposed to forms, reports and queries. Advanced courses also delve into applications of modules and macros. In many database design situations, separation of a logical data model from its physical implementation is necessary. Nontechnical, subject-matter, experts and end-users can better understand the logical data model and thus they can help improve the model's structure. This paper focuses on the database design phase. It shows a database design cases implemented in MySQL Workbench and in Microsoft Access. It also attempts to expose some difficulties and anomalies that may be encountered when using exclusively Microsoft Access. Alternative logical-design solutions are also presented.
Keywords: database, design, data model, table, relationship, query, UML.

DATABASE DESIGN PROCESS

A typical database design process involves requirement analysis, conceptual (high level) design, and logical design—all leading to a physical design (Elmasri, at al., 2004, p. 51). The final logical data model should be specific enough in order to be mapped onto a physical database. As shown, for example, in (Letkowski, 2014), a data model—developed as an Enhanced Entity Relationship Diagram (EERD) in MySQL Workbench—can be automatically transformed into a physical design (database schema). Such a model includes a complete set of entities, attributes, key, relationships, cardinality and participation constraints that are necessary to accurately define a physical database (Silberschatz, at al., 2001, p. 27-37).

Building a data model (EERD), using systems like MySQL Workbench, depends more on the subject-matter (domain) expertise than on the database skills. A subject-matter expert, who understands simple concepts of an entity (or class) and relationship can decidedly contribute to development of the data model. The following example shows a simulated scenario[1] of the database design process with two participants: a domain expert (a sales manager, "Mark") and database expert (a database administrator, "Debbie").

Mark: "I have been managing my sales for quite some time, using a phone and spreadsheet based ordering system. Recently, I have read some interesting articles about how my job can be improved by switching to a Web based system."

Debbie: "You understand that a Web system requires a multiuser and concurrent access to your data! Thus you can't use your spreadsheet program. You would have to move to a database solution."

Mark: "Yes, I have read about that too. Can you help me with designing a database? I'd like to see, for example, how basic information about customers, orders and products can be stored in the database."

Debbie: "Absolutely! Could you briefly explain how your system works? At this stage, let's us focus on capturing and modeling information about orders."

Mark: "It is quite simple. We maintain a list of customers and products. Our customers place orders from time to time. We make sure that each order belongs to one and only one customer. The orders include some of our products—mainly small pieces of furniture (desks, chairs, shelves, etc.). Each product may be part of many orders."

Debbie: "I get it! Would you agree that your basic data entity sets (or object sets) are: Customer, Order and Product?

Mark: "Indeed they are! As a matter of fact they are already maintained in our current system as spreadsheet lists. Our customers are mainly businesses so we store 'name', 'addresses', contact information ('phone', 'fax', and 'email'). Our orders includes 'order date' and reference to the customers who placed them ('cid'). Finally, our products are characterized using such properties as 'description', 'finish', 'unit price', and 'quantity on hand'. "

Debbie: "Great! You can also say that they (entity sets) are organized as tables. If so, they will be organized in a database in almost the same way—as database tables. We have to make sure that all rows in the tables are unique. To this end, for each table, we will create an identifier that, as you probably know, is referred to as a Primary Key (PK)."

Mark: "I thought it would be much harder. In fact, we use integers to identify particular rows in the spreadsheet tables so we could easily look up relevant data, for example, when generating invoices. Our identifiers are: 'cid' (customer ID), 'poid' (Order ID), and 'pid' (Product ID)."

[1] This scenario is loosely based on the Pine Valley Furniture case presented in (Hoffer, at al., 2005 p. 63-70).

Debbie: "Outstanding! Indeed, a spreadsheet lookup procedure comes very close to what we refer to in a database as a relationship. Later, in the database, you will be able to generate invoices by just using relationships, speaking of which, you have already mentioned the most important ones. Let me formalize them, using already established entity sets, as:

<Customer(1) – *places* – Order(1..n)>

<Order(0..n) – *includes* – Product(1..n)> "

Mark: "I kind of get it but not entirely.

Debbie: "The first relationships is of type 'one-to-many'. As you mentioned, a customer may place one or many orders (1..n). In the same time, each order belongs to one (1) and only one customer. The second relationship is of type 'many-to-many'. One order may include many products (1..n) and one product may be part of zero or many orders (0..n). These entities and relationships plus the attributes you mentioned above are all we need to construct a data model—the so called Entity Relationship Diagram (ERD). I will show you how to do it, using the 2014 version of MySQL Workbench[2]. The first step is to define the basic entities: Customer, Order and Product. Start MySQL workbench, create a new EERD model and, using the Table tool add the three entities, providing their names and attributes. Make sure that each entity has its own PK (Primary Key). Figure 1 shows the resulting diagram. Notice that, contrary to a spreadsheet implementation, each of the attributes must have appropriate type. The generic types (INT, FLOAT, DATETIME) are obvious. VARCHAR and CHAR stand for text types. The former speciffies the maximum size (capacity) and the latter—the exact size. Next, the relationships between the entities must be defined. They are a reflection of business. The Customer - Order relationship is created, using a Non-Identifying, One-To-Many relationship (1:n) by and connecting entity Order with entity Customer. A new attribute is added automatically to entity Order: Customer_cid (Figure 2)."

Mark: "Why is the Non-Identifying Relationship used here? What does it mean? "

Debbie: "Both the entities (Customer and Order) are already identified by their primary keys (cid, and poid). The additional field, Customer_cid, added to entity Order does not uniquely identify any order. It simply points to the customers that 'own' the orders."

Mark: "It makes sense. What about the new attribute, Customer_cid? It looks like a copy of the primary key 'cid' in entity (table) Customer. In our spreadsheet implementation we use a similar approach in order to be able to lookup customer information in our invoices."

Debbie: "Indeed it acts like a copy. Such an attribute is called a Foreign Key (FK). Its role is to make sure that each order 'knows' its owner. One can also say that each order matches (Harkins, 2004) its customer. In addition, the database system should ensure that each value of this attribute is one of the values of the related primary key, 'cid' defined in table Customer. This kind of validation is referred to as Referential Integrity. It is also important to note that this new key, Customer_cid, is strictly connected to the relationship between entities Order and Customer. Removing the key will also remove the relationship (connection line) and vice versa." It is very important to understand that a foreign key only exists in the context of a relationship. It is a property of an entity (table) whose instances (records) depend on or are spawn by an instance of the related entity. With respect to the Customer – Order relationship, a customer places (creates) one or more orders. Each order is 'signed' by one customer. This signature is represented by the foreign key."

[2] A complete (detail) procedure for developing a data model (EERD), using MySQL Workbench is shown in (Letkowski, 2014).

Mark: "OK, I got it! The PK-FK pairs reflect particular relationship instances which arise from business operations (rules). I could say that the role of the Primary Key is to maintain the Entity Integrity and the role of the Foreign Key is to ensure the Referential Integrity."
Debbie: "Very important points! Unlike spreadsheets, databases help us maintain data integrity. Out next step is to take care of the second relationship (Order – Product). This time we will use a Many-to-Many Relationship tool ('n:m'). There is only one version of this tool: the Identifying Relationship. When two entities are connect with this tool, a new entity is spawned as shown in Figure 3. This is a so called associative entity. It may not exist on its own. Both the entities, Order and Product, are needed for this new entity to exist. Its business role is to 'know' which product is part of which order. It is frequently described as an order detail, invoice line, or order line. We will use the latter (OrderLine) to name this new entity."
Mark: "This all sound reasonable. I can see that we are almost there. There is one extra piece of information missing. We should not forget to add the 'order quantity', or just 'quantity' to this new entity."
Debbie: "Excellent! This is why domain experts should always participate in database design processes. Like nobody else, they know the business rules and can anticipate what information will be needed in order to perform all sorts of business operations and decisions. To wrap up our design, let us rename the new entity and new attributes. I would also recommend to rename entity Order to PurchaseOrder. Some systems, like MySQL, do not accept user defined names that coincide with reserved word (e.g. Order). Figure 4 shows the final version of the model. "
Mark: "So now, how do we convert this model into a 'real' database?"
Debbie: "With MySQL, we can automatically generate the database structure or what it is known in the database 'world' as the database schema. This is why the model, we have just created, is called an 'Enhanced' Entity Relationship Diagram[3]."

DESIGNING A DATABASE IN ACCESS USING SQL

Using the data model, like the one shown in Figure 4, a database expert should be able to develop manually appropriate SQL - CREATE TABLE statements that would create the database schema. MySQL Workbench can generate such statements automatically from the EERD model[4]. The resulting statements are fully compatible the MySQL database system:

```
CREATE TABLE IF NOT EXISTS `Customer` (
  `cid` INT NOT NULL,
  `name` VARCHAR(100) NOT NULL,
  `address` VARCHAR(100) NULL DEFAULT NULL,
  `city` VARCHAR(40) NULL DEFAULT NULL,
  `state` CHAR(2) NULL DEFAULT NULL,
  `zip` VARCHAR(20) NULL DEFAULT NULL,
  PRIMARY KEY (`cid`));
ENGINE = InnoDB;
CREATE TABLE IF NOT EXISTS `Product` (
  `pid` INT NOT NULL,
  `description` VARCHAR(200) NULL DEFAULT NULL,
  `finish` VARCHAR(50) NULL DEFAULT NULL,
  `unitPrice` FLOAT NULL DEFAULT NULL,
```

[3] The EERD examples, included in this paper, show the logical data model implemented in UML. Since many database developers also develop applications, using Object-Oriented language it is convenient for them to express the model, using the Object-Oriented design language—UML (Naiburg, Maksimchuk, 2001).
[4] It can be done by means of menu options (commands): Database + Forward Engineer (Letkowski 2014).

```
  `onHand` INT NULL,
  PRIMARY KEY (`pid`))
ENGINE = InnoDB;
CREATE TABLE IF NOT EXISTS `PurchaseOrder` (
  `poid` INT NOT NULL,
  `poDate` DATETIME NULL DEFAULT NULL,
  `cid` INT NOT NULL,
  PRIMARY KEY (`poid`),
  CONSTRAINT `fk_Order_Customer`
    FOREIGN KEY (`cid`)
    REFERENCES `Customer` (`cid`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `OrderLine` (
  `pid` INT NOT NULL,
  `poid` INT NOT NULL,
  `quantity` INT NULL,
  PRIMARY KEY (`pid`, `poid`),
  CONSTRAINT `fk_Product_has_Order_Product1`
    FOREIGN KEY (`pid`)
    REFERENCES `Product` (`pid`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Product_has_Order_Order1`
    FOREIGN KEY (`poid`)
    REFERENCES `PurchaseOrder` (`poid`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Microsoft Access can execute such statements but it does not accept the following tokens: 'IF NOT EXISTS', 'ENGINE = InnoDB', 'DEFAULT NULL', 'ON DELETE NO ACTION', and 'ON UPDATE NO ACTION'. After removing these token in a text editor, the remaining SQL code can be executed in the SQL (Data Definition) panel of the Query environment in Access. Each of the create-table statements must be executed separately. Figure 5 shows execution the last of the following SQL statements:

```
CREATE TABLE `Customer` (
  `cid` INT NOT NULL,
  `name` VARCHAR(100) NOT NULL,
  `address` VARCHAR(100) NULL,
  `city` VARCHAR(40) NULL,
  `state` CHAR(2) NULL,
  `zip` VARCHAR(20) NULL,
  PRIMARY KEY (`cid`));
CREATE TABLE `Product` (
  `pid` INT NOT NULL,
  `description` VARCHAR(200) NULL,
  `finish` VARCHAR(50) NULL,
  `unitPrice` FLOAT NULL,
  `onHand` INT NULL,
  PRIMARY KEY (`pid`)
);
CREATE TABLE `PurchaseOrder` (
```

```
  `poid` INT NOT NULL,
  `poDate` DATETIME NULL,
  `cid` INT NOT NULL,
  PRIMARY KEY (`poid`),
  CONSTRAINT `fk_Order_Customer`
    FOREIGN KEY (`cid`) REFERENCES `Customer` (`cid`));
CREATE TABLE `OrderLine` (
  `pid` INT NOT NULL,
  `poid` INT NOT NULL,
  `quantity` INT NULL,
  PRIMARY KEY (`pid`, `poid`),
  CONSTRAINT `fk_Product_has_Order_Product1`
    FOREIGN KEY (`pid`) REFERENCES `Product` (`pid`),
  CONSTRAINT `fk_Product_has_Order_Order1`
    FOREIGN KEY (`poid`) REFERENCES `PurchaseOrder` (`poid`));
```

The resulting Relationships diagram (Figure 6) is similar to the original data model (Figure 4). Access shows all cardinality constraints but without the participation constraints. For example, according to the original model the participation of entity OrderLine in the OrderLine – Product relationship is optional. This detail is not revealed in Access. This relationship is shown in Access as **<OrderLine(∞) – Product(1)>** whereas in the original model it is defined as **<OrderLine(0..n) – *gets* – Product(1)>.** Here, symbols '∞'and 'n' both stand for 'many'.

Another interesting aspect of the Relationships diagram in Access is that the relationships are not firmly coupled with their foreign keys. For example, contrary to the MySQL Workbench model, removing a relationship does not remove the foreign key standing behind the relationship.

Adding records to Access tables via SQL is somewhat challenging. Only one SQL-Insert statement can be executed at a time in the SQL–Query panel. A macro command is needed to run multiple SQL-Insert statements. The following VBA code will execute sequentially all SQL statements contained in a plain-text document, `sql.txt`, stored in directory `C:\db`:

```
Sub BatchSQL()
  Dim fileRef As Integer
  Dim txtSqlStatements As String
  Dim sqlStatmentList As Variant
  Dim sqlStatement As Variant

  fileRef = FreeFile()
  Open "C:\db\sql.txt" For Input As #fileRef
  txtSqlStatements = Input(LOF(fileRef), #fileRef)
  Close fileRef
  sqlStatmentList = Split(txtSqlStatements, ";")

  On Error Resume Next
  For Each sqlStatement In sqlStatmentList
     CurrentDb.Execute sqlStatement
  Next
End Sub
```

Having all the tables populated with data, Access is now ready for applications. The phone based ordering system can benefit from more friendly and reliable input forms. Relevant information can be quickly retrieved using queries. Finally, the management can benefit from periodically generated reports. However, these problems go beyond the scope of this paper. They are exhaustively addressed, for example, in (Adamski, at al., 2011) and (Poatsy, at al., 2014).

DESIGNING A DATABASE IN ACCESS USING THE GUI

Teaching database concepts and applications in business curricula is typically driven by tutorial based instructions provided by popular textbooks like (Adamski, at al., 2011) or (Poatsy, at al., 2014), utilizing mainly the GUI environment of Microsoft Access.

Students learn first how to build database tables. They are not exposed to design issues extensively. Typically, they are brought directly into physical databases, learning technical aspect of navigating in a maze of menus, toolbars, dialog panels, and other Windows widgets. For example, (Adamski, at al., 2011, p. AC1-AC48) starts with Tutorial 1, "Creating a Database". The students are introduced to basic database terminology, including notion of 'field', 'table', 'database', 'primary key', 'foreign key', and 'relationship'. They also learn in this tutorial how to develop a simple 'query', 'form', and 'report'. The concept of a relationship is explained by means of the primary key and the foreign key. A similar approach is shown in (Poatsy, at al., 2014, p. 83-140). The students first learn how to use an existing database by performing simple sorting and filtering operations, next they are introduced to relational database issues where relationships are explained again, using the primary and foreign keys.

It is beyond the scope of this paper to show all details of the database development process performed with the Graphical User- Interface (GUI) in Access. (Adamski, at al., 2011) and (Poatsy, at al., 2014) do it thoroughly. Additional instructive materials can also be retrieved directly from Microsoft (MS Access – Guide, 2014), (MS Access – Relationship, 2014). However it is important to notice that the Access-GUI based process is not entirely consistent with the common system development process. This process, as shown in the previous section (DESIGNING A DATABASE IN ACCESS USING SQL), adheres to the principles and structure of the system development life cycle (SDLC) involving a sequence of tasks such as (Hoffer, at al., p.46):

1. Identification
2. Initiation and Planning
3. Analysis
4. Logical Design
5. Physical Design
6. Implementation
7. Maintenance

In Access, steps 4 and 5 are intertwined or somewhat reversed. Given a fully developed logical data model for a relational database should include both the entity and referential integrity features, as shown, for example, in (Figure 4), such a model can't be developed in Access without first building a physical database. More over this model must (up front) include all necessary foreign keys. Figure 7 shows a 'Design View' for table `PurchaseOrder`. The `pid` and `poid` fields play two roles. They define both the primary key of the table and the foreign keys, linking this table to tables `Product` and `PurchaseOrder`, respectively. At this stage, this table definition is equivalent to the following SQL statement:

```
CREATE TABLE `OrderLine` (
  `pid` INT NOT NULL,
  `poid` INT NOT NULL,
  `quantity` INT NULL,
  PRIMARY KEY (`pid`, `poid`)
);
```

Moreover, the foreign keys, `pid` and `poid`, kind of exist but they are not formally acknowledged or marked up as such. By contrast, the MySQL Workbench design (Figure 4) shows the `PurchaseOrder` entity with all its bells and whistles, identifying explicitly the relationships and producing a physical table that is fully compliant with both the entity and referential integrity rules:

```
CREATE TABLE `OrderLine` (
  `pid` INT NOT NULL,
  `poid` INT NOT NULL,
  `quantity` INT NULL,
  PRIMARY KEY (`pid`, `poid`),
  CONSTRAINT `fk_Product_has_Order_Product1`
    FOREIGN KEY (`pid`) REFERENCES `Product` (`pid`),
  CONSTRAINT `fk_Product_has_Order_Order1`
    FOREIGN KEY (`poid`) REFERENCES `PurchaseOrder` (`poid`)
);
```

In order to bring this and other tables to the same status in Access, a 'Relationships' diagram must be completed. Figure 8 shows an initial diagram with all the database [physical] tables included. For the foreign keys to become the first class citizens, they must be linked with their respective primary keys. Figure 9 shows the final phase for completing the logical and physical design. The primary key, `pid`, of the `Product` table is dragged-and-dropped onto (linked to) the foreign key, `pid`, of the `OrderLine` table and the 'Enforce Referential Integrity' option is checked. All other relationships are defined in a similar way. The database is ready for application development (queries, forms, reports, etc.).

CONCLUSIONS

The entire GUI based database design process leads to the same result as the MySQL Workbench one (Figures 6 and 9). However, it is important to note a subtle difference between these two ways. The EER diagram tool of MySQL Workbench uses relationships to generate foreign keys. It also is capable of generating associative entities when many-to-many relationships are being resolved. The resulting model is fully compliant with the entity and referential integrity constraints. On the other hand, the GUI tools in Access require a physical database that already satisfies the entity integrity constraints. Finally, by developing the Relationships diagram, the database becomes 100% relational.

Between Microsoft Access and MySQL Workbench, the latter seems to be better suited for database design consistent with the System/Software Development Life Cycle (SDLC) process (Wikipadia - SPD 2015). Moreover, developing an ERD with MySQL Workbench has additional advantages. The ERD can be expressed in many notations, including UML (Wikipadia - SPD 2015) which arguably has the most capable tools to fully support the SDLC (Naiburg, Maksimchuk, 2015, p. 10). Figure 10 shows an ERD developed in UML, sing one of the initial versions of Rational Rose. What is very significant about this ERD is that it does not show any foreign keys. It is as pure a design as it gets. Obviously the foreign keys are buried behind the relationships. The following SQL script, generated by Rational Rose, defines a proper database scheme (including the foreign keys):

```
CREATE TABLE T_Customer(
    cid int() NOT NULL,
    name VARCHAR(100),
    address VARCHAR(100),
    city VARCHAR(40),
    state VARCHAR(20),
    zip VARCHAR(20),
    PRIMARY KEY(cid))
CREATE TABLE T_PurchaseOrder(
    poid int() NOT NULL,
    poDate DateTime(),
    cid int() NOT NULL,
    FOREIGN KEY (cid) REFERENCES T_Customer,
    PRIMARY KEY(poid))
CREATE TABLE T_Product(
    pid int() NOT NULL,
    description VARCHAR(200),
    finish VARCHAR(50),
    unitPrice float(),
    onHand int(),
    PRIMARY KEY(pid))
CREATE TABLE T_OrderLine(
    quantity int(),
    poid int() NOT NULL,
    FOREIGN KEY (poid) REFERENCES T_PurchaseOrder,
    pid int() NOT NULL,
    FOREIGN KEY (pid) REFERENCES T_Product,
    OrderLineId NUMBER(5),
    PRIMARY KEY(OrderLineId))
```

No matter which design tools is used, an intriguing question remains. Should business students learn how to design databases according to the SDLC process? The current approach seems to be some sort of prototyping—a method of the Rapid Application Development (RAD) approach. The SDLC process is more methodical and better structured and, generally, prototyping requires a higher level of database expertise (Hoffer, at al., 2004, p.48). One would suppose that the SDLC process should be given more consideration especially in business curricula for at least two reasons. First, typically, at the learning time, the students are beginners. Second, after graduation they are more likely to become domain rather than database experts and thus they may have more opportunities to participate in database design activities rather than in database application development duties.

REFERENCES

Adamski, J. J., Finnegan. K. T. (2011). *New Perspectives on Microsoft Access 2010, Brief.* Boston: Course Technology - Cengage Learning.

Elmasri, R., Navathe, S. B. (2004). *Fundamentals of Database Systems*. Boston: Pearson / Addison Wesley.

Harkins, S. (2004). *Define relationships between database tables.* TechRepublic. Retrieved from: http://www.techrepublic.com/article/define-relationships-between-database-tables/

Hoffer, J. A, Prescott, M. B., McFadden, F. R. (2005). *Modern Database Management*, Seventh Edition. Upper Saddle River, New Jersey: Pearson - Prentice Hall.

Letkowski, J. (2014). *Doing database design with MySQL.* Journal of Technology Research. ISSN Online: 1941-3416 (http://www.aabri.com/jtr.html) Volume 6.

MS Access - Relationship (2014). *Create, edit or delete a relationship*. Retrieved from: https://support.office.com/en-in/article/Create-edit-or-delete-a-relationship-dfa453a7-0b6d-4c34-a128-fdebc7e686af

MS Access - Guide (2014). *Guide to table relationships*. Retrieved from: https://support.office.com/en-in/article/Guide-to-table-relationships-30446197-4fbe-457b-b992-2f6fb812b58f

Naiburg, E.J., Maksimchuk, R. J. (2001). *UML for Database Design*. Boston: Addison Wesley.

Poatsy, M. N., Krebs, C., Cameron, E., Williams, J., Grauer, R.T. (2014). *Exploring: Microsoft Access 2013, Comprehensive*. Boston: Pearson Education, Inc.

Silberschatz, A., Korth, H. F., Sudarshan S. (2001), *Database System Concepts*. Boston: Osborne/McGraw-Hill.

Wikipadia - SPD (2015). Software development process. Retrieved from: http://en.wikipedia.org/wiki/Software_development_process

Wikipadia - UML (2015). Unified Modeling Language. Retrieved from: http://en.wikipedia.org/wiki/Unified_Modeling_Language

APPENDIX

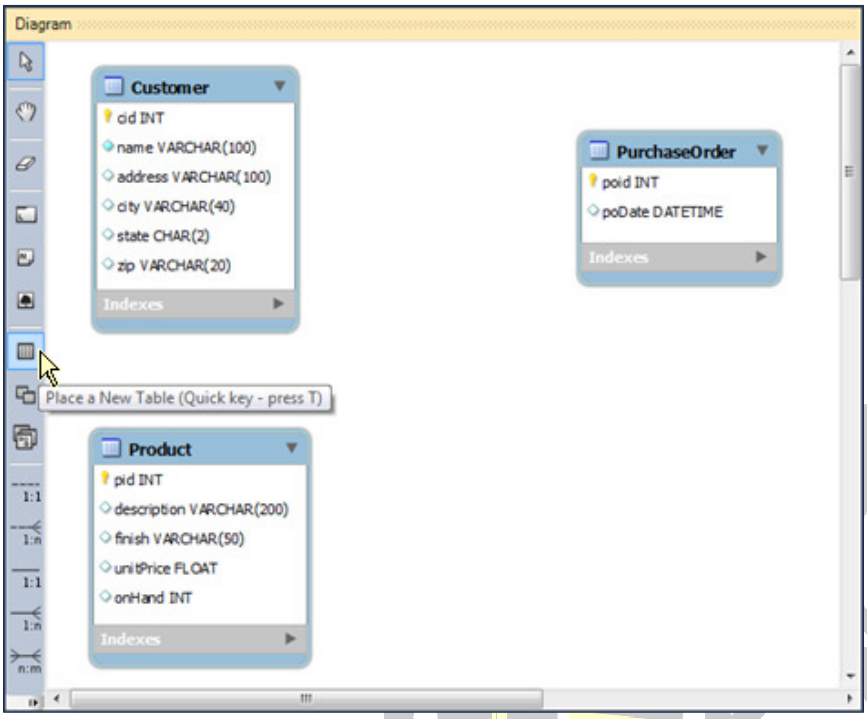

Figure 1. The basic components (entities) of the data model. A new table is created using the New Table tool.

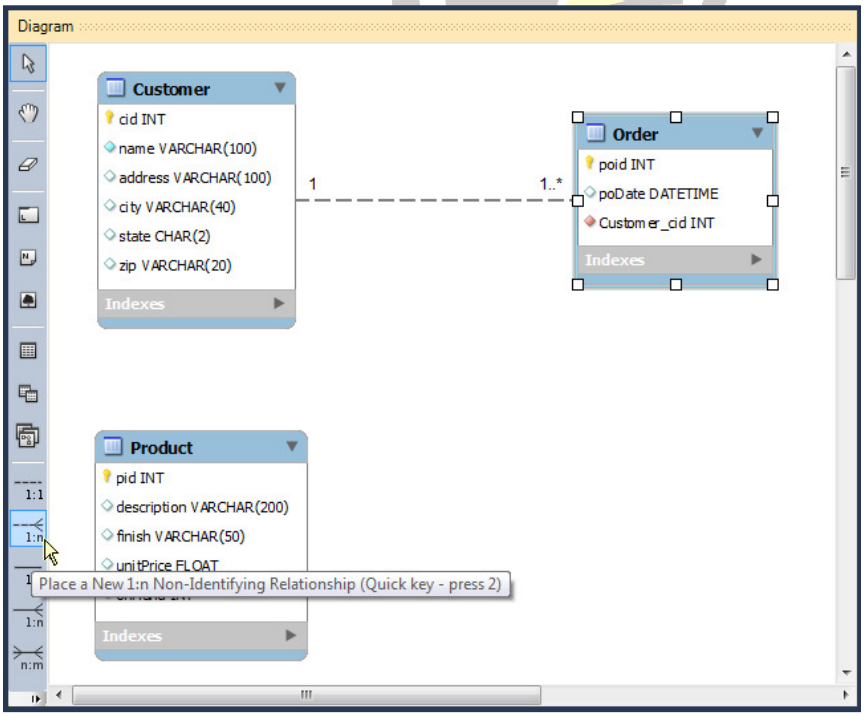

Figure 2. Connecting entity (table) Order with Customer. A foreign key, Customer_cid, is added automatically to table Order.

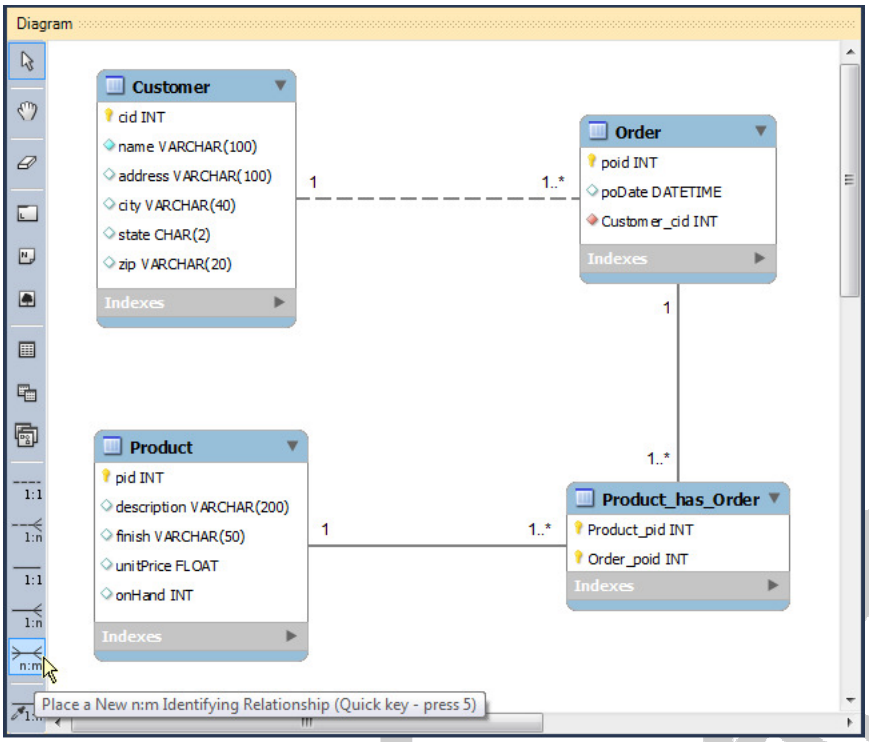

Figure 3. Resolving a Many-to-Many relationship. The 'n:m' tool connects entities Product and Order.

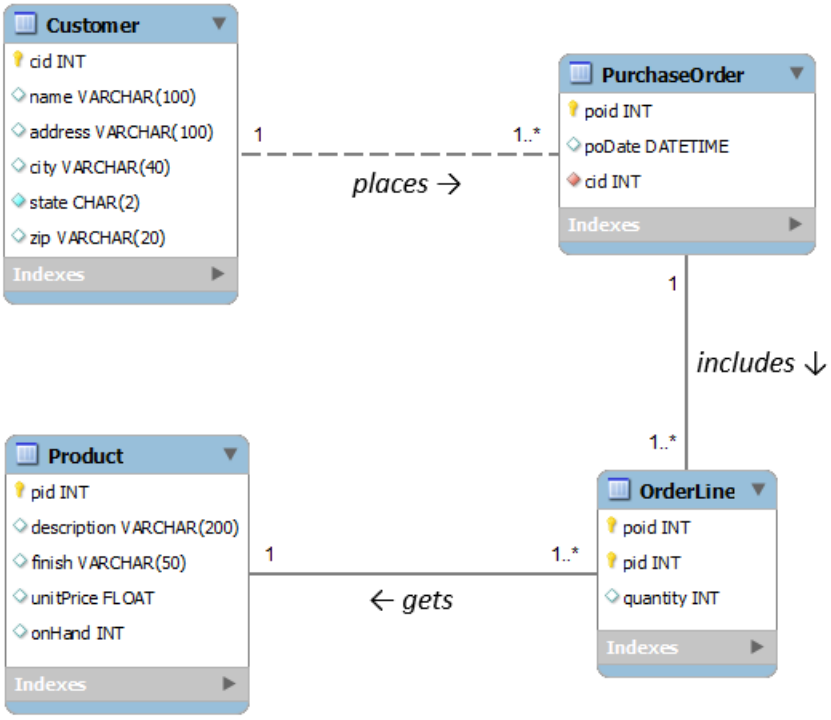

Figure 4. A data model (EERD) of a product ordering system.

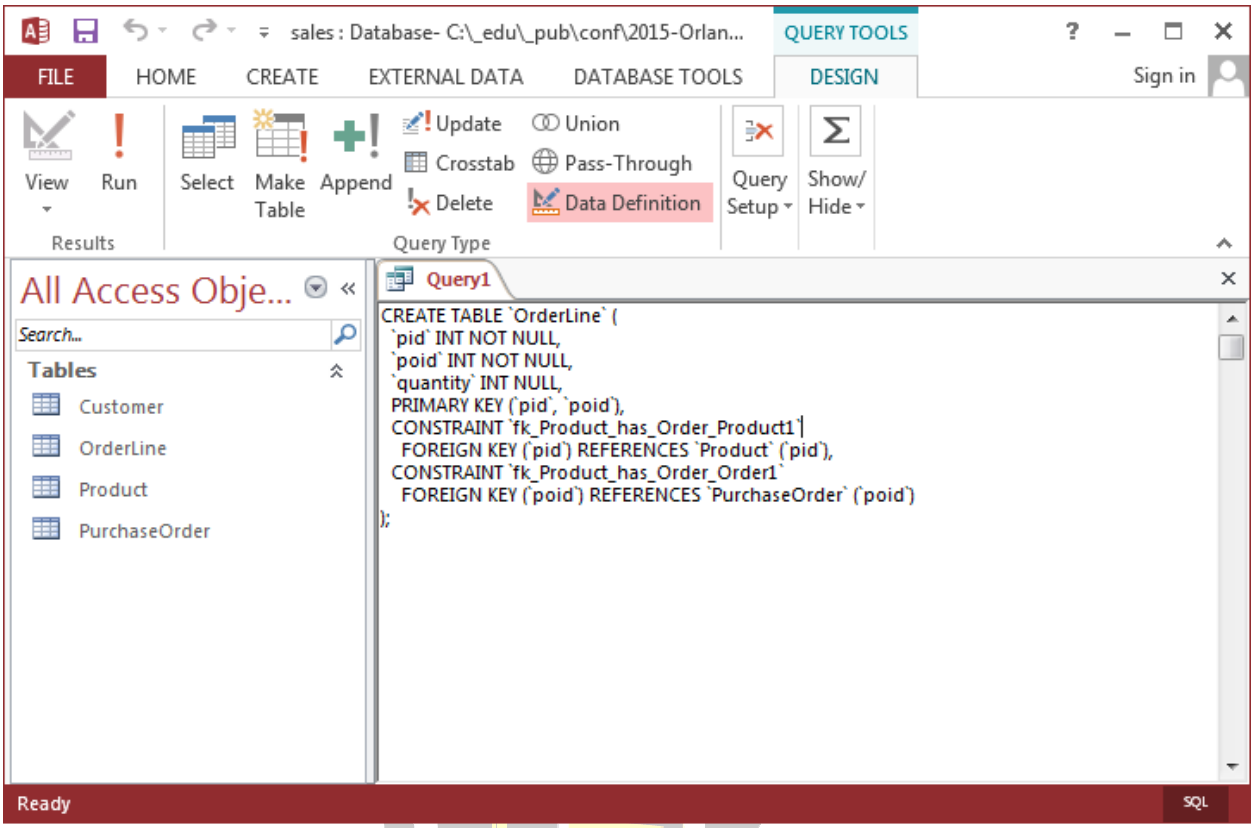

Figure 5. Executing SQL- Create Table statement in a Data Definition panel, in Access.

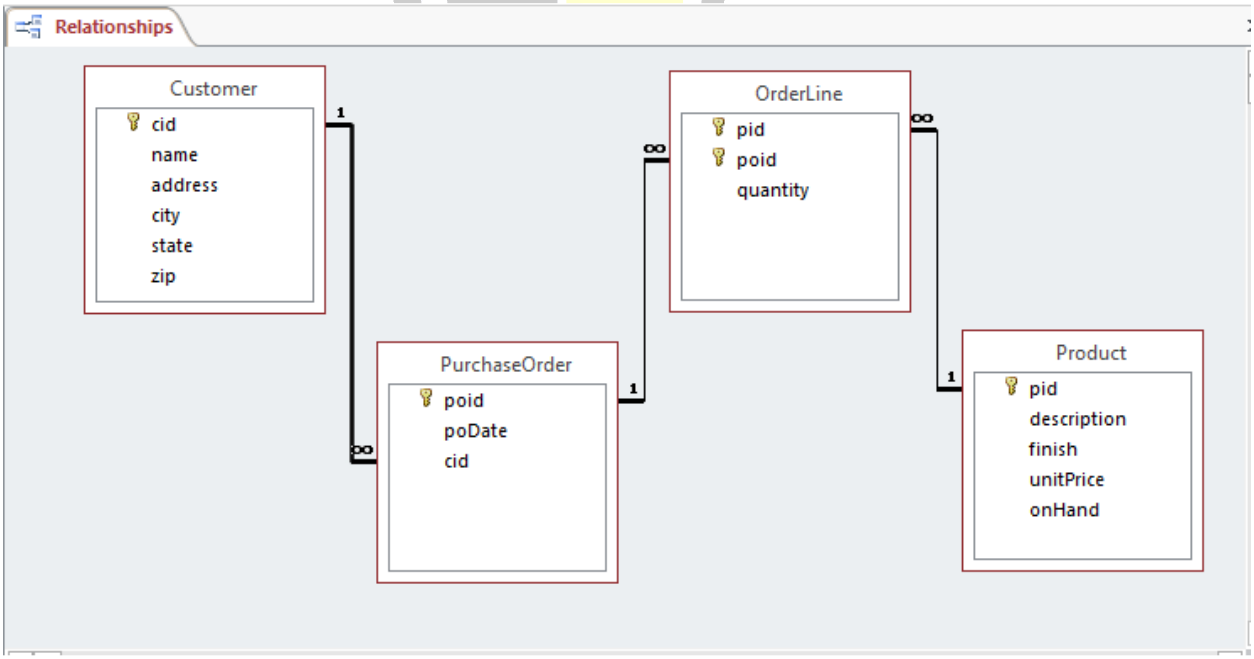

Figure 6. The Relationship diagram of the database in Access.

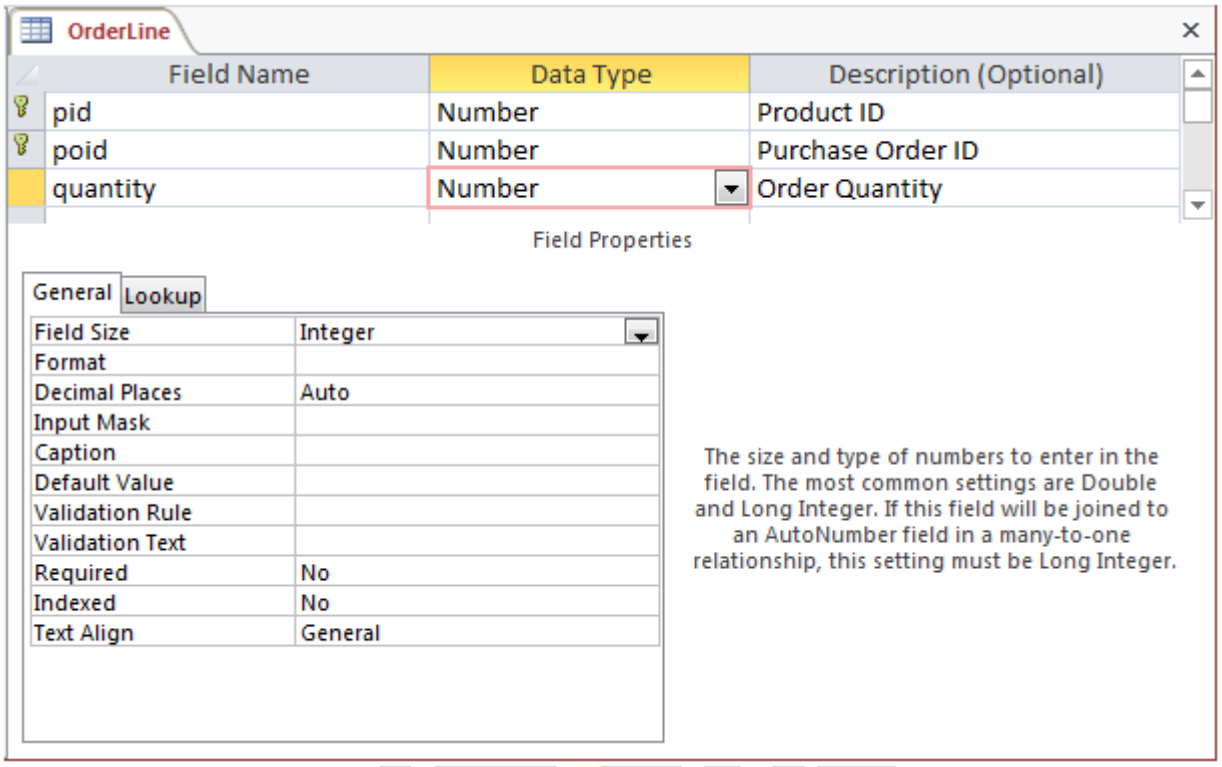

Figure 7. Defining table PurchaseOrder in Access.

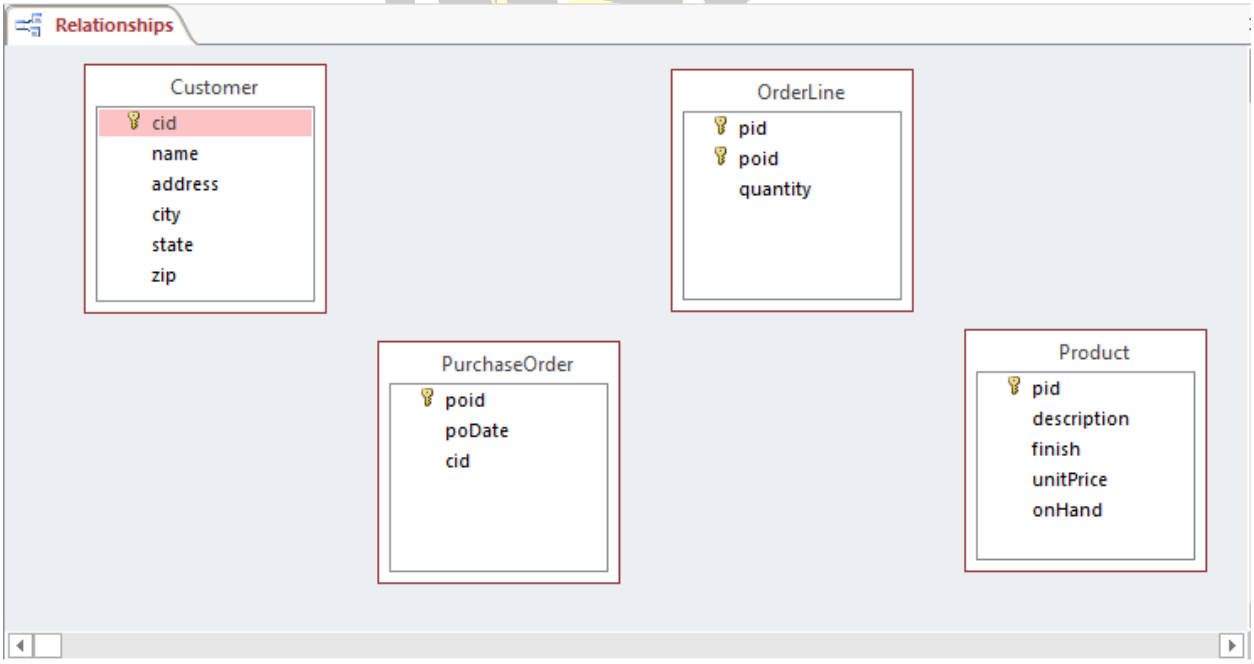

Figure 8. The initial phase of developing a Relationships diagram in Access (a collection of unlinked tables).

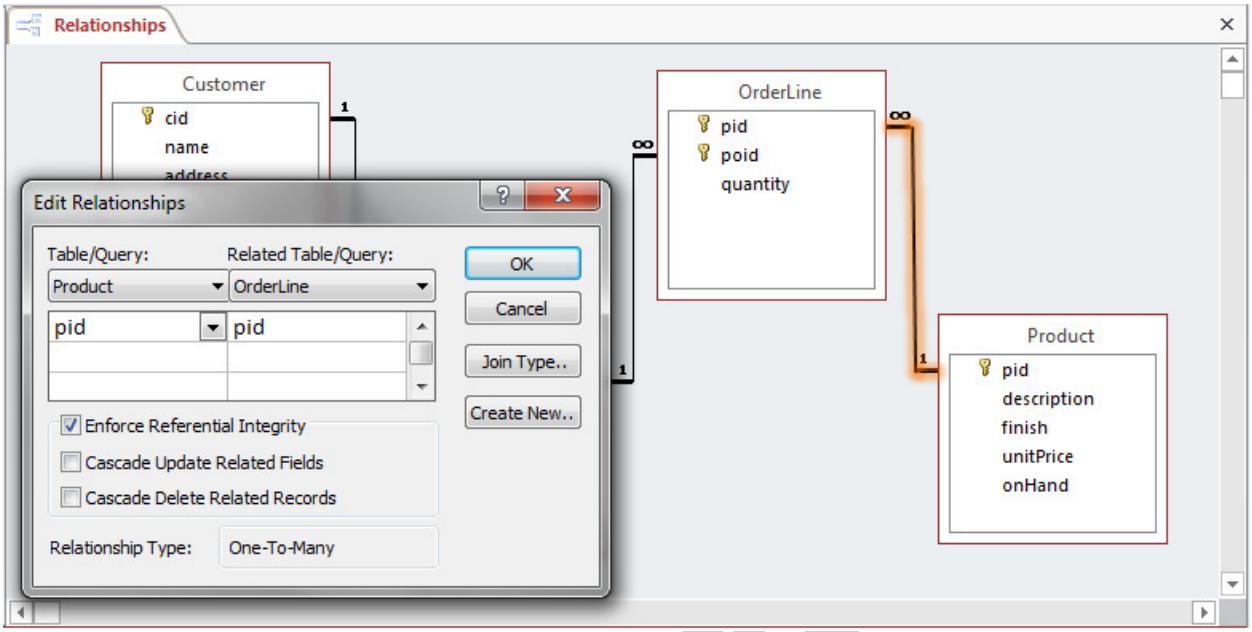

Figure 9. Defining a relationship between database entities (tables) and enforcing the referential integrity in Access.

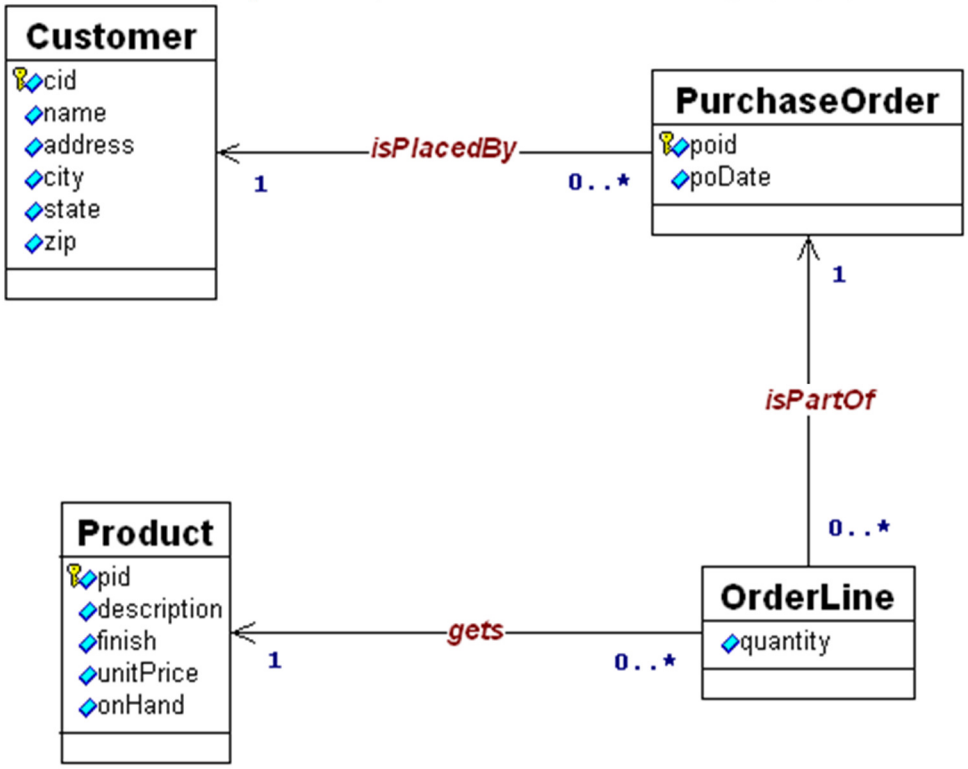

Figure 10 An ERD developed in UML, using Rational Rose.