Erik Ratcliffe, CS-340, Spring 2015

Final Project - Discrete Components Database

My database tracks an inventory for discrete electronic components. As an amateur electronics enthusiast, I have accumulated a "scrappy" collection of components that spans everything from simple solid-core wires all the way up to ARM Cortex M4 microcontrollers. However, like many others that participate in amateur electronics, I do not have a vast storage facility for organizing the components, nor do I (currently) have any way to know at a glance if I am running low on a part. Having a database that can track my inventory of the parts is long overdue. It will not help with physical organization of the components, but it will solve the "do I have parts for this project" problem nicely.

Detailed Description

The most important table in the database is the Component table. It is where the rubber meets the road: all part names, part numbers values and measures, quantities, etc. are tracked there. The only bits that are not tracked by the Component table are in the two many-to-many tables: Component_Category and Supplier_Component, which track Categories and Suppliers, respectively. More on those tables later.

The **Component** table tracks the component name ('name', unique, no default value, can not be NULL), pins ('pins', default 0, can be null), part number ('partno', default NULL), value ('value', default NULL), and quantity in the inventory ('quantity', no default, can not be NULL), and includes foreign key relationships with the Measure ('MeasureID', default NULL), Package ('PackageID', default NULL), and Vendor ('VendorID', default NULL) tables. If any records in this table are updated or deleted, upon deletion their related records in Component_Category and Supplier_Component tables are deleted and upon update the updates cascade to Component_Category and Supplier_Component.

Components in the Component table do not have any unique identifiers. This is because many vendors can sell their own version of a common component (examples: 555 timers, 74HC595 shift registers) with the only differentiator being the vendor name. These similarly named components do not need to be identified directly by their vendors because they work identically or else they would not be able to use their name (i.e. all 555 timers work the exact same way whether they are manufactured by Texas Instruments, NXP, Maxim, or any other vendor). So, the presence of the part in the inventory is the only bit that is truly important.

The **Measure** table tracks the measure name ('name', unique, no default, can not be NULL) of a component. Examples of measures: "ohms" for resistors, "volts" for power supplies. An auto-incremented ID is assigned to each row in Measure and serves as its primary key. Each Component can have only one Measure. If any of the records in this table are updated or deleted, upon deletion their related record in Component changes to NULL and upon update the update cascades to Component.

The **Package** table tracks the package name ('name', unique, no default, can not be NULL) of a component. Examples of packages: "DIP" for dual inline pins, "QFN" for quad flat no-leads. An auto-incremented ID is assigned to each row in Package and serves as its primary key. Each Component can have only one Package. If any of the records in this table are updated or deleted, upon deletion their related record in Component changes to NULL and upon update the update cascades to Component.

The **Vendor** table tracks the Vendor name ('name', unique, no default, can not be NULL)) as well as contact information of a component manufacturer. Examples of vendors: Atmel, Texas Instruments, NXP. An auto-incremented ID is assigned to each row in Vendor and serves as its primary key. Address and contact information for each Vendor in the table includes three general address fields ('address1', 'address2', and 'address3', all default NULL), a locality ('locality', default NULL), a region ('region', default NULL), a postal code ('postalcode', default NULL), a web address ('web', default NULL), a phone number ('phone', default NULL), and a foreign key relationship with the Country table ('CountryID', default NULL). Effort was put into making these fields generic to any locality as some vendors are not located in the United States and hence may not adhere to US address schemes. (In the US, address1-address3 are the street address, locality is the city, region is the state, and postal code is the zip code.) Each Component can have only one Vendor. If any of the records in this table are updated or deleted, upon deletion their related record in Component changes to NULL and upon update the update cascades to Component.

The **Country** table tracks the country names ('name', unique, no default, can not be NULL) of component Vendors and Suppliers. Examples of countries: China, United States, Japan. An auto-incremented ID is assigned to each row in Country and serves as its primary key. Each Vendor can have only one Country. If any of the records in this table are updated or deleted, upon deletion their related record in Vendor changes to NULL and upon update the update cascades to Vendor.

The **Supplier** table is identical to the Vendor table, but it applies to Suppliers. This table tracks the Supplier

name ('name', unique, no default, can not be NULL)) as well as its contact information. Examples of suppliers: Mouser, Adafruit, eBay. An auto-incremented ID is assigned to each row in Supplier and serves as its primary key. Address and contact information for each Supplier in the table includes three general address fields ('address1', 'address2', and 'address3', all default NULL), a locality ('locality', default NULL), a region ('region', default NULL), a postal code ('postalcode', default NULL), a web address ('web', default NULL), a phone number ('phone', default NULL), and a foreign key relationship with the Country table ('CountryID', default NULL). Effort was put into making these fields generic to any locality as some suppliers are not located in the United States and hence may not adhere to US address schemes. (In the US, address1-address3 are the street address, locality is the city, region is the state, and postal code is the zip code.) Each Component can have multiple Suppliers. If any of the records in this table are updated or deleted, upon deletion their related records in the many-to-many table Supplier_Component are deleted and upon update the update cascades to Supplier_Component.

The **Supplier_Component** table maps the many-to-many relationship between Suppliers and Components. Each record in the table matches a Supplier ID ('supID', default 0, not NULL) and Component ID ('compID, default 0, not NULL). The combination of Supplier and Component IDs makes up the primary key for this table. If either a Component or Supplier (or both) is updated or deleted, upon deletion their related row in Supplier_Component is deleted and upon update the related row is updated with the new ID information.

Backtracking slightly, there is also a **Category** table. The Category table tracks category names ('name', unique, no default, can not be NULL) that can be associated with components. Examples of categories: "AVR" for Atmel AVR microcontrollers, resistors, capacitors. An auto-incremented ID is assigned to each row in Category and serves as its primary key. Each Component can have multiple Categories. If any of the records in this table are updated or deleted, upon deletion their related record in the many=-to-many table Component_Category is deleted and upon update the update cascades to Component_Category.

The **Component_Category** table maps the many-to-many relationship between Categories and Components. Each record in the table matches a Category ID ('catID', default 0, not NULL) and Component ID ('compID, default 0, not NULL). The combination of Category and Component IDs makes up the primary key for this table. If either a Component or Category (or both) is updated or deleted, upon deletion their related row in Component_Category is deleted and upon update the related row is updated with the new ID information.

Queries – Table Creation

```
CREATE TABLE Measure (
    ID int NOT NULL AUTO_INCREMENT,
    name varchar(255) NOT NULL,
    PRIMARY KEY (ID),
    UNIQUE KEY name (name)
) ENGINE=InnoDB;

CREATE TABLE Package (
    ID int NOT NULL AUTO_INCREMENT,
    name varchar(255) NOT NULL,
    PRIMARY KEY (ID),
    UNIQUE KEY name (name)
) ENGINE=InnoDB;

CREATE TABLE Category (
    ID int NOT NULL AUTO_INCREMENT,
    name varchar(255) NOT NULL,
    PRIMARY KEY (ID),
    UNIQUE KEY name (name)
) ENGINE=InnoDB;

CREATE TABLE Country (
    ID int NOT NULL AUTO_INCREMENT,
    name varchar(255) NOT NULL,
    PRIMARY KEY (ID),
    UNIQUE KEY name (name)
) ENGINE=InnoDB;

CREATE TABLE Supplier (
    ID int NOT NULL AUTO_INCREMENT,
    name varchar(255) NOT NULL,
    address1 varchar(255) DEFAULT NULL,
    address2 varchar(255) DEFAULT NULL,
    address3 varchar(255) DEFAULT NULL,
    locality varchar(255) DEFAULT NULL,
    region varchar(255) DEFAULT NULL,
    postalcode varchar(255) DEFAULT NULL,
```

```
    web varchar(255) DEFAULT NULL,
    phone varchar(255) DEFAULT NULL,
    CountryID int,
    PRIMARY KEY (ID),
    CONSTRAINT FOREIGN KEY (CountryID) REFERENCES Country (ID)
      ON DELETE SET NULL
      ON UPDATE CASCADE,
    UNIQUE KEY name (name)
) ENGINE=InnoDB;

CREATE TABLE Vendor (
    ID int NOT NULL AUTO_INCREMENT,
    name varchar(255) NOT NULL,
    address1 varchar(255) DEFAULT NULL,
    address2 varchar(255) DEFAULT NULL,
    address3 varchar(255) DEFAULT NULL,
    locality varchar(255) DEFAULT NULL,
    region varchar(255) DEFAULT NULL,
    postalcode varchar(255) DEFAULT NULL,
    web varchar(255) DEFAULT NULL,
    phone varchar(255) DEFAULT NULL,
    PRIMARY KEY (ID),
    CountryID int,
    CONSTRAINT FOREIGN KEY (CountryID) REFERENCES Country (ID)
      ON DELETE SET NULL
      ON UPDATE CASCADE,
    UNIQUE KEY name (name)
) ENGINE=InnoDB;

CREATE TABLE Component (
    ID int NOT NULL AUTO_INCREMENT,
    name varchar(255) NOT NULL,
    pins int DEFAULT 0,
    partno varchar(255) DEFAULT NULL,
    value varchar(255) DEFAULT NULL,
    quantity int NOT NULL,
    MeasureID int,
    PackageID int,
    VendorID int,
    PRIMARY KEY (ID),
```

```
    CONSTRAINT FOREIGN KEY (MeasureID) REFERENCES Measure (ID)
      ON DELETE SET NULL
      ON UPDATE CASCADE,
    CONSTRAINT FOREIGN KEY (PackageID) REFERENCES Package (ID)
      ON DELETE SET NULL
      ON UPDATE CASCADE,
    CONSTRAINT FOREIGN KEY (VendorID) REFERENCES Vendor (ID)
      ON DELETE SET NULL
      ON UPDATE CASCADE
) ENGINE=InnoDB;

CREATE TABLE Supplier_Component (
    supID int NOT NULL DEFAULT 0,
    compID int NOT NULL DEFAULT 0,
    PRIMARY KEY (supID, compID),
    CONSTRAINT FOREIGN KEY (supID) REFERENCES Supplier (ID)
      ON DELETE CASCADE
      ON UPDATE CASCADE,
    CONSTRAINT FOREIGN KEY (compID) REFERENCES Component (ID)
      ON DELETE CASCADE
      ON UPDATE CASCADE
) ENGINE=InnoDB;

CREATE TABLE Component_Category (
    compID int NOT NULL DEFAULT 0,
    catID int NOT NULL DEFAULT 0,
    PRIMARY KEY (compID, catID),
    CONSTRAINT FOREIGN KEY (compID) REFERENCES Component (ID)
      ON DELETE CASCADE
      ON UPDATE CASCADE,
    CONSTRAINT FOREIGN KEY (catID) REFERENCES Category (ID)
      ON DELETE CASCADE
      ON UPDATE CASCADE
) ENGINE=InnoDB;
```

Queries – General Use

For checking if a name from a table is unique:

```
SELECT name FROM [table name] WHERE name=[name];
```

SELECTing records from the one-to-many tables:

```
SELECT ID, name FROM Category;
SELECT ID, name FROM Country;
SELECT ID, name FROM Measure;
SELECT ID, name FROM Package;
```

I use a generic A alias in the next query because the query in PHP is shared between the Vendor and Supplier tables; same structure, different name and different relation (Vendor = one-to-many, Supplier = many-to-many):

```
SELECT A.ID, A.name, A.address1, A.address2, A.address3,
    A.locality, A.region, A.postalcode, A.web, A.phone, C.name AS countryname
FROM Vendor A;
```

INSERT code for Category, Measure, and Package, is very simple: just a name:

```
INSERT INTO Category (name) VALUES ([name]);
INSERT INTO Country (name) VALUES ([name]);
INSERT INTO Measure (name) VALUES ([name]);
INSERT INTO Package (name) VALUES ([name]);
```

SELECTing from many-to-many tables for Category and Supplier is shared in a common PHP script so the generic A alias is once again employed:

```
SELECT A.ID, A.name, A.address1, A.address2, A.address3,
   A.locality, A.region, A.postalcode, A.web, A.phone, C.name AS countryname
  FROM [table name] A
  INNER JOIN Country C ON C.ID = A.CountryID;
```

For Vendors and Suppliers, my PHP form is mostly text input fields but has one "select" list for country name. The ID associated with the selected country is added directly to the INSERT SQL. I do not do any subqueries to look up the country ID because it's not necessary with the way the select list is built (each option has its country's ID in its value attribute).

```
INSERT INTO Supplier (name, address1, address2, address3,
  locality, region, postalcode, web, phone, CountryID)
VALUES ([name], [1st address line], [2nd address line], [3rd address line],
  [locality], [region], [postal code], [web URL], [phone number],
  [Country ID from the list]);
```

The Component table on the webpage shows all information in the Component table except for the Vendor. This bit of information is nice to have but it is not essential for a main view of an inventory system.

```
SELECT CM.id, CM.name, CM.pins, CM.partno, CM.value,
  CM.quantity, P.name, M.name
FROM Component CM
LEFT JOIN Package P ON P.ID = CM.PackageID
LEFT JOIN Measure M ON M.ID = CM.MeasureID;
```

If the user clicks on a Component's "Edit" link (in the "Actions" table column) or chooses the "Add Component" link in the site navigation, (s)he will be directed to a page where all Component data as well as Vendor and Supplier associations can be edited/added. To build this page, I start with the attributes in the Components table:

```
SELECT name, pins, partno, value, quantity, MeasureID, PackageID, VendorID
FROM Component
WHERE ID=[Component ID];
```

From there, I grab all Categories that are associated with the Component. These will be used later to highlight the chosen Categories in the edit form's multi-select Category and Supplier lists:

```
SELECT ID FROM Category C
INNER JOIN Component_Category CC ON CC.catID = C.ID
WHERE CC.compID=[Component ID];
```

```
SELECT ID FROM Supplier S
INNER JOIN Supplier_Component SC ON SC.supID = S.ID
WHERE SC.compID=[Component ID];
```

As the HTML form in the PHP "edit" script is built, attributes from one-to-many tables Measure, Package, and Vendor, as well as records from Category and Supplier tables, are pulled from the database and are used to build select lists:

```
SELECT DISTINCT id, name FROM [table name] ORDER BY name;
```

If the user clicks on a "Delete" button (in the "Actions" column in any table view) the item associated with the Delete button will be removed and the table updated:

```
DELETE FROM [table name] WHERE id=[ID of item to delete];
```

INSERT code for Components is slightly more complex. I start off by INSERTing the values that can go straight into the table with no fuss (values for Measure, Package, and Vendor are chosen by the user from "select" lists. The IDs for these attributes are in each form element's value attribute):

```
INSERT INTO Component (name, pins, partno, value, quantity,
  MeasureID, PackageID, VendorID)
VALUES ([name], [number of pins], [part number], [value], [quantity],
  [Measure ID from the list], [Package ID from the list],
  [Vendor ID from the list]);
```

With the bulk of the Component data INSERTed, I grab the ID of the Component and use it to add entries to the "Component-Category" and "Supplier_Component" tables, both of which are many-to-many:

```
INSERT INTO Component_Category (compId, catId)
VALUES ([Component ID],[Category ID]);

INSERT INTO Supplier_Component (compId, supId)
VALUES ([Component ID],[Category ID]);
```

These INSERT statements are run for each selection in the multi-select list associated with Category and Supplier in the form. Each form element returns an array, which I walk through to add the records.

When editing a Component, the form is pre-filled with existing values from the database. If nothing changed for a field, the value for the attribute is updated with an identical value; otherwise, it changes to reflect what the user entered in the form:

```
UPDATE Component
SET name = [name], pins = [number of pins], partno = [part number],
  value = [value], quantity = [quantity],
  MeasureID = [Measure ID from the list],
  PackageID = [Package ID from the list],
  VendorID = [Vendor ID from the list]
WHERE ID=[Component ID];
```

Updating many-to-many records for Categories and Suppliers requires a few extra steps. Rather than UPDATE the records in each table, the cleanest method is to DELETE the old records and create new ones that match the full sets of Categories and Suppliers the user selected from the lists:

```
DELETE FROM Component_Category WHERE compId=[Component ID];

DELETE FROM Supplier_Component WHERE compId=[Component ID];
```

Then, for each Category in the array returned by the form:

```
INSERT INTO Component_Category (compId, catId)
VALUES ([Component ID],[Category ID from the list]);
```

Then, for each Supplier in the array returned by the form:

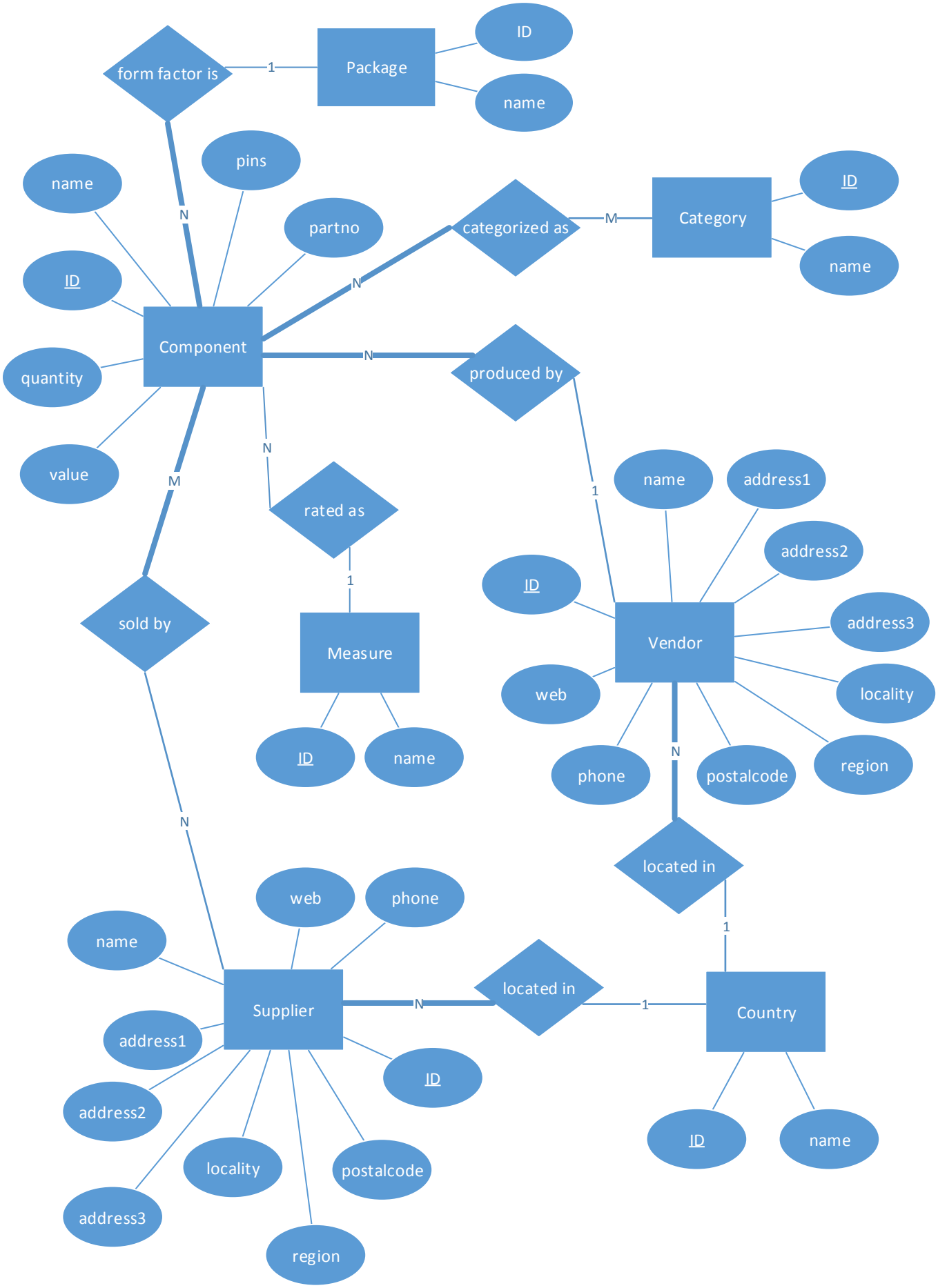
```
INSERT INTO Supplier_Component (compId, supId)
VALUES ([Component ID],[Supplier ID from the list]);
```

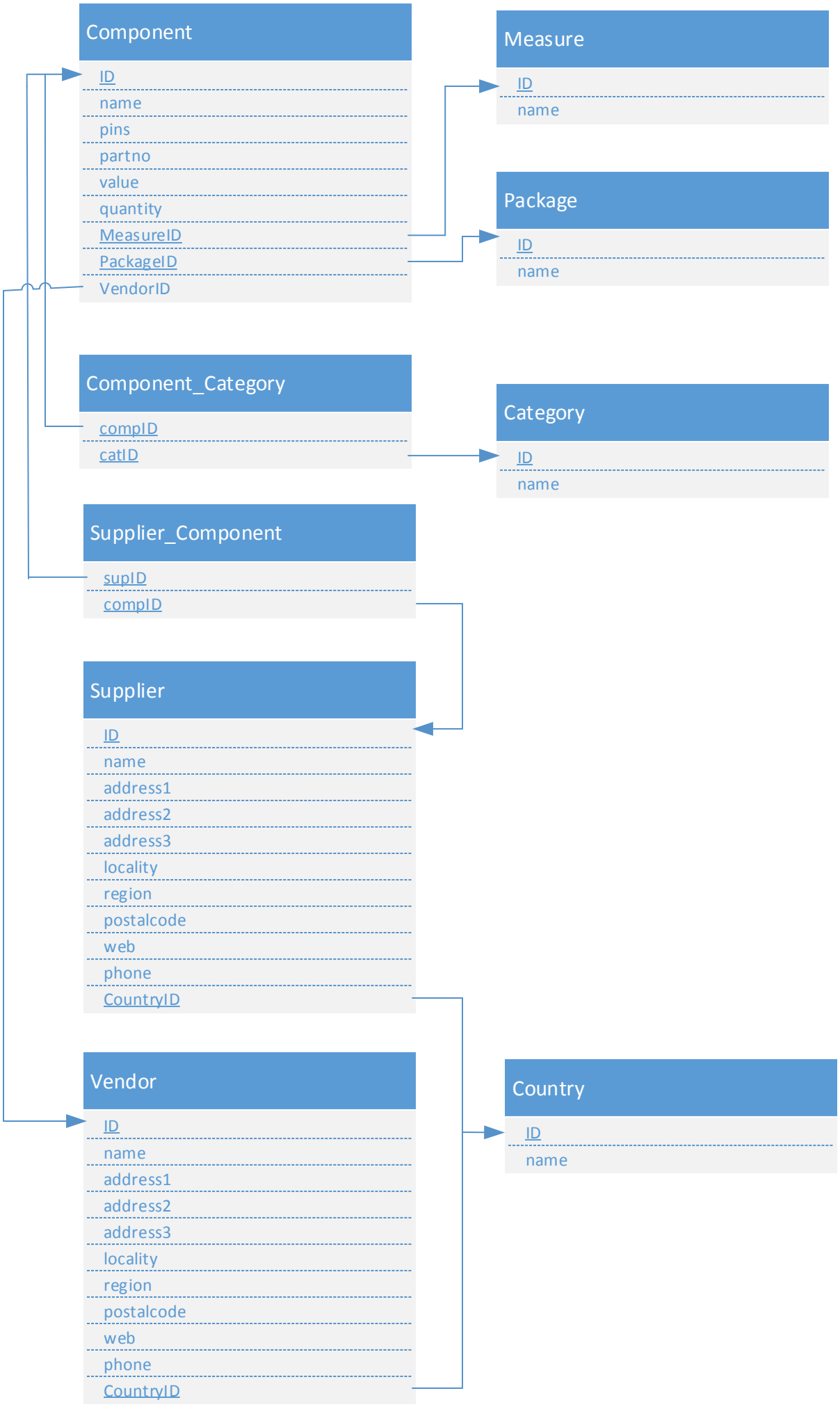
UPDATEing Measure, Package, Country, and Component records is fairly straightforward:

```
UPDATE [table name] SET name = [name] WHERE ID=[table item ID];
```

UPDATEing Vendor and Supplier records is the same, but for different tables:

```
UPDATE [table name] SET name = [name], address1 = [1st address line],
  address2 = [2nd address line], address3 = [3rd address line],
  locality = [locality], region = [region], postalCode = [postal code],
  web = [web URL], phone = [phone number],
  CountryID = [Country ID from the list]
WHERE ID=[table item ID];
```


Package
ID
name
form factor is
1
N
pins
name
partno
ID
Component
quantity
value
categorized as
M
N
Category
ID
name
N
produced by
1
name
address1
address2
ID
Vendor
address3
web
locality
phone
postalcode
region
N
M
N
rated as
1
Measure
ID
name
sold by
N
located in
1
web
phone
name
Supplier
address1
address2
address3
locality
region
postalcode
ID
N
located in
1
Country
ID
name


Component
ID
name
pins
partno
value
quantity
MeasureID
PackageID
VendorID
Measure
ID
name
Package
ID
name
Component_Category
compID
catID
Category
ID
name
Supplier_Component
supID
compID
Supplier
ID
name
address1
address2
address3
locality
region
postalcode
web
phone
CountryID
Vendor
ID
name
address1
address2
address3
locality
region
postalcode
web
phone
CountryID
Country
ID
name