Executive Report  
SQL - Inventory Database Design

short line

Big Money Inc.

TEL:867-5309

EMAIL:[bigmoney@hotmail.com](mailto:bigmoney@hotmail.com)

MEMBERS: David Thomsen // Micah Kezar

# Introduction

The purpose of this project was to design a three-part database with the goal of combining and managing the parts, vendors, and products within New England Bikes. The insert codes were given to input the data into the tables so we were able to reverse engineer the database using these insert codes as well as writing the code to create the three databases.

# Executive Summary

The goal of this assignment was to create a three part database for New England Bikes and to create a script that creates these databases and links them so they can be queried together. This would have been much harder to do without the insert code, however with this, we were able to figure out what tables to make, where to put them, and their relationship with each other. The processes for reverse engineering them were all very similar so in the sections below, the parts database will have the full description for what we did, then the other sections will have just the differences listed. Additionally, they will be split by page for easier organization.

# 

# Parts Database

There are many different parts to each database that was created. We were given a spreadsheet of the type of data that is meant to be entered into the database, as well as an insert code for the specific database. This insert code held almost every piece of data that was necessary, and the data filled in the blanks.

## Code Extraction - insert\_parts\_data.sql

use Parts;

delete from PartVendor;

delete from PartHistory;

delete from PartProduct;

delete from PartInventory;

delete from Part;

 This section of the code at the very beginning is used to wipe all of the tables within the database so this was an obvious start. These are the names of the database as well as all of the tables that will be made.

Next we had to figure out which data was going to be put into each table which was easy to figure out with the data tables given as well as the code.

INSERT INTO Part VALUES('BF2001','bike frame - cruising bike');

INSERT INTO Part VALUES('F0235','bike frame - mountain');

Here are the input codes for specifically the part table. There are two separate values in each row being entered which are **part\_id** and a **description**.

We had to do this for each table. Part History was next in the code.

INSERT INTO PartHistory VALUES('BR0021','2024-02-09',120);

INSERT INTO PartHistory VALUES('BR9164','2024-03-02',120);

Similar to before, values here are being inserted into the PartHistory table, however, there are three values being inserted, a **part\_id,** a **date**, and a **quantity.** This table is necessary to link other tables as well as keep a running total of parts.

Next is the PartInventory table.

INSERT INTO PartInventory VALUES('BF2001',12,10);

INSERT INTO PartInventory VALUES('F0235',34,20);

Same as in PartHistory, there are three values that are being inserted into the table; the **part\_id**, **current\_inventory,** and **min\_inventory.** This is similar to part history, except this shows what is currently available.

Next is the PartProduct Table.

INSERT INTO PartProduct VALUES('BF2001',705);

INSERT INTO PartProduct VALUES('F0235',23);

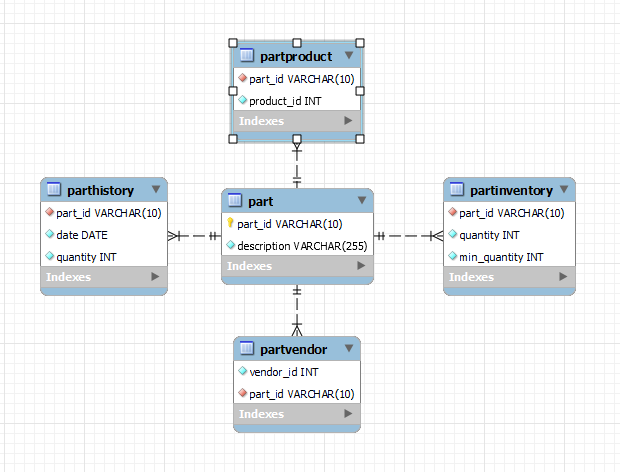
This table only has two values, **part\_id**  and **product\_id.** This is used later on to link parts and products. Similar to the next table which is the PartVendor Table.

INSERT INTO PartVendor VALUES(101,'BF2001');

INSERT INTO PartVendor VALUES(202,'F0235');

This table only has two values, **part\_id**  and **vendor\_id.** This is used later on to link parts and vendors.

## ERD



This combines all of the previous values that were seen in the insert code, and relates them using an ERD. As seen in the ERD, the main table linking everything is the part table with the part\_id. This will be used to link the other databases later for queries.

# 

# Products Database

Similar to above, the same methods will be used for the Products Database, just with less explanation as it is essentially the same process as written in the previous database.

## Code Extraction

use products;

delete from ProductInventory where pid>0;

delete from ProductPriceHistory where pid>0;

delete from PartProduct;

delete from Product where pid>0;

Similar to before, the insert code wipes the tables at the beginning which provides us with the name of the database, as well as the names of all tables being used.

## Product

insert into Product values(896,'The Road Warrior','This is our finest all-purpose bike. Great for all activites',450);

insert into Product values(125,'Racing Bike','Our premier racing bike',1900);

This table has four inserted values: **Product\_id, product\_name, product\_desc,** and **product\_price**

## ProductInventory

insert into ProductInventory values(896,10,'2019-03-1',15);

insert into ProductInventory values(125,18,'2019-03-1',25);

This table has four inserted values: **Product\_id, product\_quantity, last\_update,** and **product\_min\_quantity.**

## ProductPriceHistory

insert into ProductPriceHistory values(705,'2018-03-1',2000);

insert into ProductPriceHistory values(705,'2018-01-1',1900);

This table has three inserted values: **Product\_id, product\_date, and product\_price**

## PartProduct

INSERT INTO PartProduct VALUES('BF2001',705);

INSERT INTO PartProduct VALUES('F0235',23);

This table has two values that are used in join commands later to query the parts and products. **Part\_id and Product\_id**

## 

## ERD

## 

Similar to before, building the table was pretty straight forward as we had all of the tables and values within the tables, so our only job was combining it into a database. There is only one database left to reverse engineer, which is the Vendors database.

# Vendors Database

Similar to above, the same methods will be used for the Vendors Database, just with less explanation as it is essentially the same process as written in the previous database.

## Code Extraction

use vendors;

delete from salespersoncontact where vspid>0;

delete from address where vcid>0;

delete from salesperson where vspid>0;

delete from company where vcid>0;

delete from contact where vspcontactid>0;

DB name: **vendors**

Tables: **salespersoncontact, address, salesperson, company, contact**

## Contact

insert into contact values(11,"Phone");

insert into contact values(22,"E-Mail");

This table has two values that are used in join commands later: **Part\_id and Product\_id**

## Company

insert into company values(1,"Acme");

insert into company values(2,"Speedster");

insert into company values(3,'Widget');

This table has two values that are used in join commands later: **Vendor\_id and Vendor\_name**

## Address

insert into address values(1, "45 East West Street","Manchester","NH","02335");

insert into address values(2, "1 Main Street", "Burlington","VT", "05401");

insert into address values(3, "786 Widets Lane","Hershey","PA", "25874");

This table has two values that are used in join commands later: **Vendorcontact\_id, street, city, state,** and **zip**

## Salesperson

insert into salesperson values(1,"Sally", "Saw",1);

insert into salesperson values(2,"Steve", "Perez",1);

This table has two values that are used in join commands later: **VendorSalesPerson\_id sp\_fn, sp\_ln, vendorContact\_id**

## 

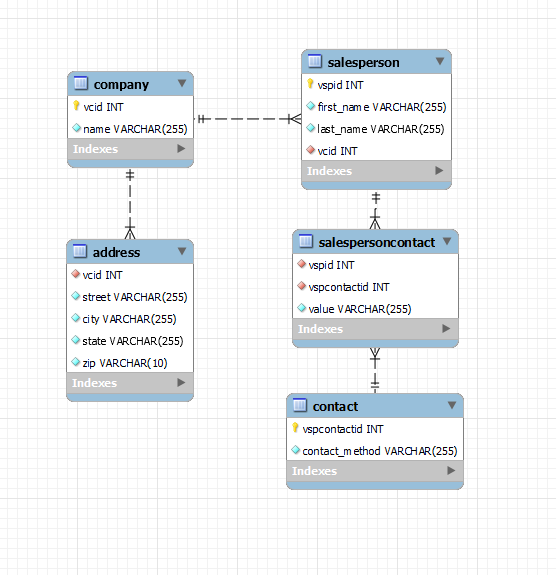
## Salespersoncontact

insert into salespersoncontact values(4,22,'ejones@speedster.com');

insert into salespersoncontact values(1,22,'ss@acme.com');

This table has three values that are used in join commands later: **VendorSalesPerson\_id, VendorContact\_id**

## ERD:



Again, building the table was pretty straight forward as we had all of the tables and values within the tables, so our only job was combining it into a database. The only tricky part within this step is ensuring that the primary keys and foreign keys don't cause conflict later.

# Creation Script

Using the previous ERDs, we were able to craft a creation script to build the three databases that are necessary. This part was not too difficult, but called for much troubleshooting in regards to syntax and formatting.

CREATE DATABASE IF NOT EXISTS Parts;

USE Parts;

CREATE TABLE IF NOT EXISTS Part (

part\_id VARCHAR(10) PRIMARY KEY,

description VARCHAR(255) NOT NULL

);

CREATE TABLE IF NOT EXISTS PartHistory (

part\_id VARCHAR(10) NOT NULL,

date DATE NOT NULL,

quantity INT NOT NULL,

FOREIGN KEY (part\_id) REFERENCES Part(part\_id)

);

CREATE TABLE IF NOT EXISTS PartInventory (

part\_id VARCHAR(10) NOT NULL,

quantity INT NOT NULL,

min\_quantity INT NOT NULL,

FOREIGN KEY (part\_id) REFERENCES Part(part\_id)

);

CREATE TABLE IF NOT EXISTS PartProduct (

part\_id VARCHAR(10) NOT NULL,

product\_id INT NOT NULL,

FOREIGN KEY (part\_id) REFERENCES Part(part\_id)

);

CREATE TABLE IF NOT EXISTS PartVendor (

vendor\_id INT NOT NULL,

part\_id VARCHAR(10) NOT NULL,

FOREIGN KEY (part\_id) REFERENCES Part(part\_id)

);

CREATE DATABASE IF NOT EXISTS Products;

USE Products;

CREATE TABLE IF NOT EXISTS Product (

pid INT PRIMARY KEY,

name VARCHAR(255) NOT NULL,

description VARCHAR(255) NOT NULL,

price DECIMAL(10, 2) NOT NULL

);

CREATE TABLE IF NOT EXISTS ProductInventory (

pid INT NOT NULL,

quantity INT NOT NULL,

last\_update DATE NOT NULL,

min\_quantity INT NOT NULL,

FOREIGN KEY (pid) REFERENCES Product(pid)

);

CREATE TABLE IF NOT EXISTS ProductPriceHistory (

pid INT NOT NULL,

date DATE NOT NULL,

price DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (pid) REFERENCES Product(pid)

);

CREATE TABLE IF NOT EXISTS PartProduct (

part\_id VARCHAR(10) NOT NULL,

product\_id INT NOT NULL,

FOREIGN KEY (product\_id) REFERENCES Product(pid)

);

CREATE DATABASE IF NOT EXISTS Vendors;

USE Vendors;

CREATE TABLE IF NOT EXISTS Contact (

vspcontactid INT PRIMARY KEY,

contact\_method VARCHAR(255) NOT NULL

);

CREATE TABLE IF NOT EXISTS Company (

vcid INT PRIMARY KEY,

name VARCHAR(255) NOT NULL

);

CREATE TABLE IF NOT EXISTS Address (

vcid INT NOT NULL,

street VARCHAR(255) NOT NULL,

city VARCHAR(255) NOT NULL,

state VARCHAR(255) NOT NULL,

zip VARCHAR(10) NOT NULL,

FOREIGN KEY (vcid) REFERENCES Company(vcid)

);

CREATE TABLE IF NOT EXISTS Salesperson (

vspid INT PRIMARY KEY,

first\_name VARCHAR(255) NOT NULL,

last\_name VARCHAR(255) NOT NULL,

vcid INT NOT NULL,

FOREIGN KEY (vcid) REFERENCES Company(vcid)

);

CREATE TABLE IF NOT EXISTS SalespersonContact (

vspid INT NOT NULL,

vspcontactid INT NOT NULL,

value VARCHAR(255) NOT NULL,

FOREIGN KEY (vspid) REFERENCES Salesperson(vspid),

FOREIGN KEY (vspcontactid) REFERENCES Contact(vspcontactid)

);



This code was built in a pretty straightforward method, as we started with the database (USE products), and slowly worked through creation of each table. One issue we ran into was the lack of the NN tag, which was a simple addition within the script. The basic form for each section of the script is as follows:

Use DATABASE

CREATE TABLE IF NOT EXISTS (Table Name)

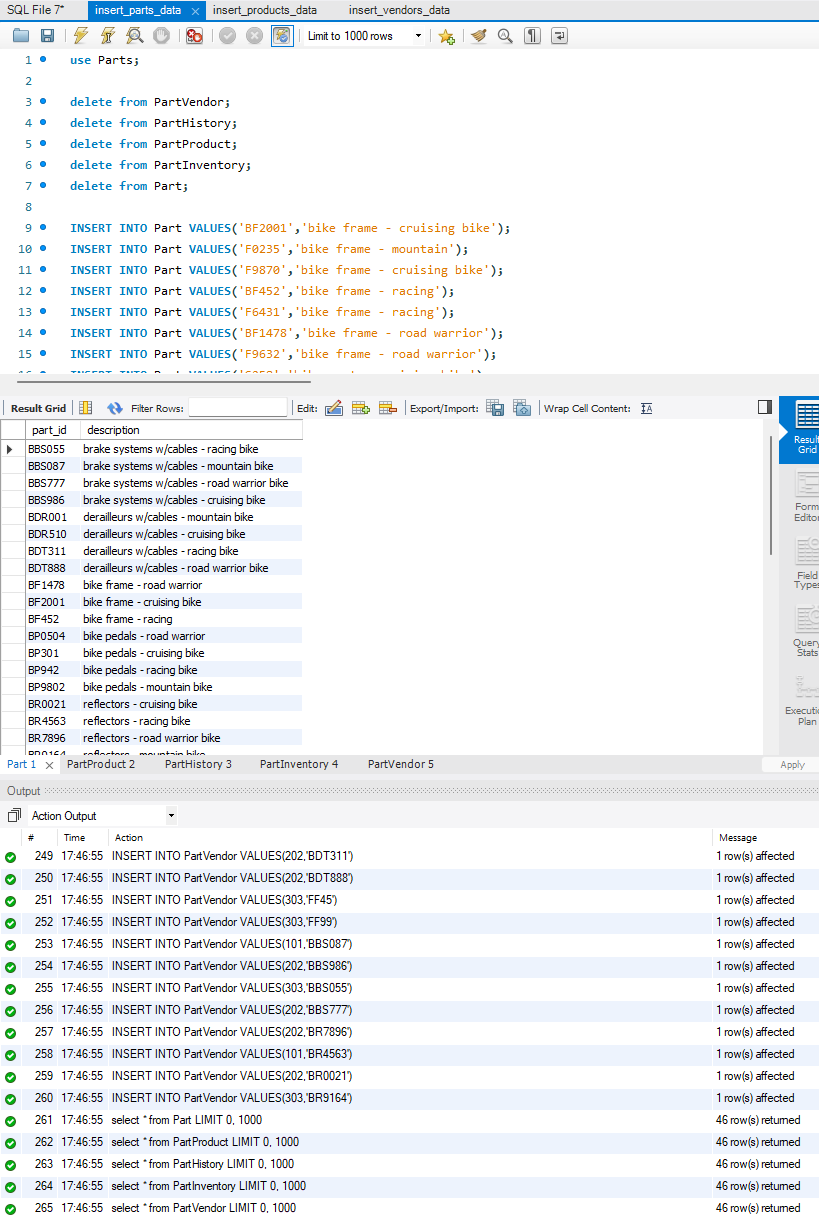
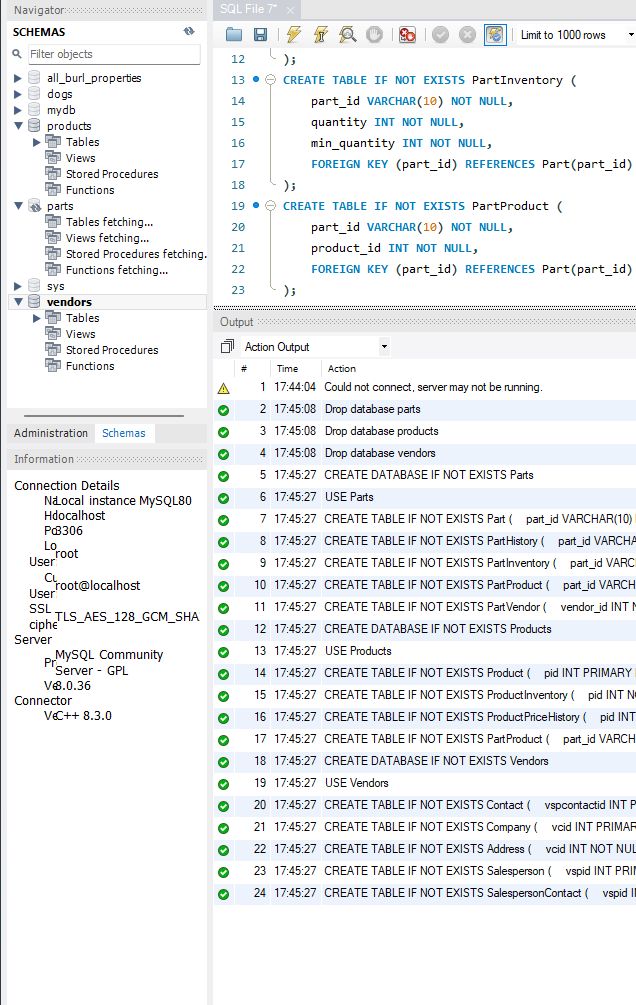
'Table Value #1' 'Datatype' 'Tags'

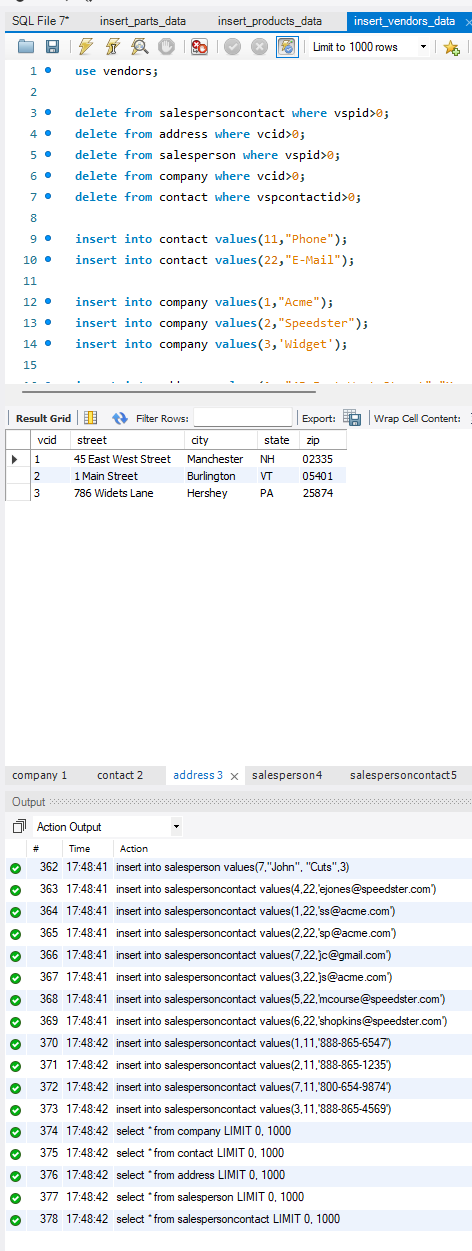
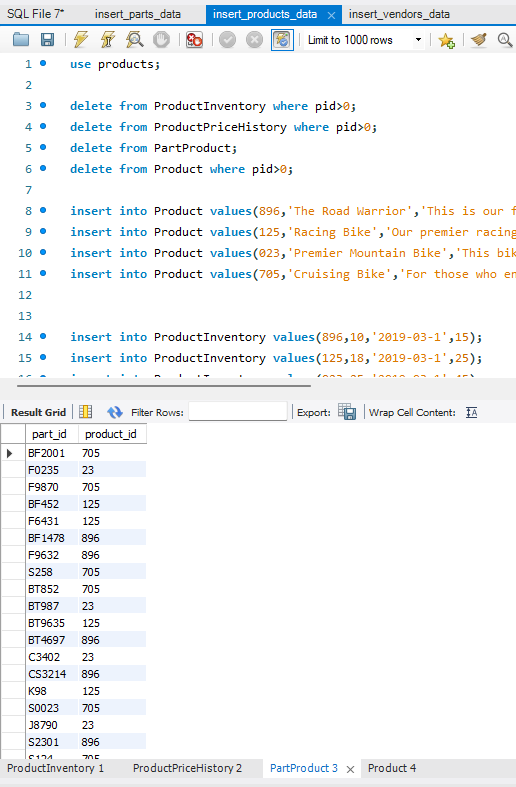
'Table Value #2' 'Datatype'

FOREIGN KEY ('value') REFERENCES Tabele(value),

This was repeated to create and fill in all of the ERDs that we had created, then after much troubleshooting, the insert scripts ran without issues.

# Insert Demo





## Queries File

-- Shows all paarts as well as their descriptions and quantity left

SELECT Part.part\_id, Part.description, PartInventory.quantity

FROM Parts.Part

JOIN Parts.PartInventory ON Part.part\_id = PartInventory.part\_id;

-- Retrieve the salespersons associated with a company, including their contact information.

SELECT Salesperson.first\_name, Salesperson.last\_name, Contact.contact\_method, SalespersonContact.value

FROM Vendors.Salesperson

JOIN Vendors.SalespersonContact ON Salesperson.vspid = SalespersonContact.vspid

JOIN Vendors.Contact ON SalespersonContact.vspcontactid = Contact.vspcontactid

WHERE Salesperson.vcid = (SELECT vcid FROM Vendors.Company WHERE name = 'Acme');

-- Get the inventory details of parts associated with a specific product

SELECT Part.part\_id, Part.description, PartInventory.quantity, PartInventory.min\_quantity

FROM Parts.Part

JOIN Parts.PartProduct ON Part.part\_id = PartProduct.part\_id

JOIN Parts.PartInventory ON Part.part\_id = PartInventory.part\_id

WHERE PartProduct.product\_id = (SELECT pid FROM Products.Product WHERE name = 'The Road Warrior');



short dash

# Additional Questions:

**Database Design & Analysis (12 points)**

* Did you design your database/tables using the design principles we have learned from the Churcher book? (ex. is there unnecessary repetition?)
  + Designing the database was not very difficult as all of the databases, tables, and values were given to us, so all that was necessary was putting them in the right places so that they interact properly with each other. In this case, we were able to create a database for each section that can interact with other databases that were created, and this can all be recreated in the table creation and insert scripts.
* Are your tables in third normal form?
  + Yes because there are no partial dependencies between tables and attributes are dependent on the primary key
* Is your design efficient?
  + I believe so as it was taken straight from the code that was given.I believe that it would be considered efficient as the end goal was three coherent and integrated databases and that is what was created.
* Did you implement all of the features required by this document?
  + Yes. The features that are required by the document are that the databases created are in 3NF, they are efficient, and the input scripts work. All of this was done to the best of our ability and everything is working properly with the creation script that we created.

**SQL (6 points)**

* Did your SQL work?
  + Yes sir! We made sure to test this about a thousand times to ensure that even on other systems or environments, the script should work alongside the insert scripts.
* Did you properly use SQL syntax?
  + Yes. If this was not the case, the script wouldn’t run properly and would create a large amount of errors until SQL could decipher it properly.
* Was your SQL of a clean style? (comments, PEP-8, etc.)
  + We did not include comments as we were aiming to keep the script very compact. We described the code in this report as it is pretty straightforward and to avoid syntax errors that we had been running into. Other than a lack of comments directly in the file, the SQL is clean and organized by database, then tables, then values.
* Did it meet the criteria laid out in this document? (Anything missing?)
  + Nope, there is nothing missing as the document outlines that the final report should include the following:
    - A SQL query file that can recreate your database (Have that!)
    - Another SQL query file with sample SQL statements for each of the features that each of your database supports and comments explaining each one (Got that too)
    - A short report (about two pages) that explains how your database data for each model works, which also includes a diagram (ERD) of your data model (As always we went overboard on a report but it's here!)

**Report (2 points)**

* Did your report coherently explain your project?
  + Our report walks the reader through every step we took in order to get to the result that we have come to. It is coherent with snippets of insert code as well as screenshots of our ERDs.
* - Does your report meet the requirements outlined in this document? (don’t forget the diagram!)
  + This report surpasses the two page requirement by a large amount but much of this is due to formatting and inclusion of large blocks of code to add clarity to our descriptions. Again this meets the requirements but also is much more than was necessary.