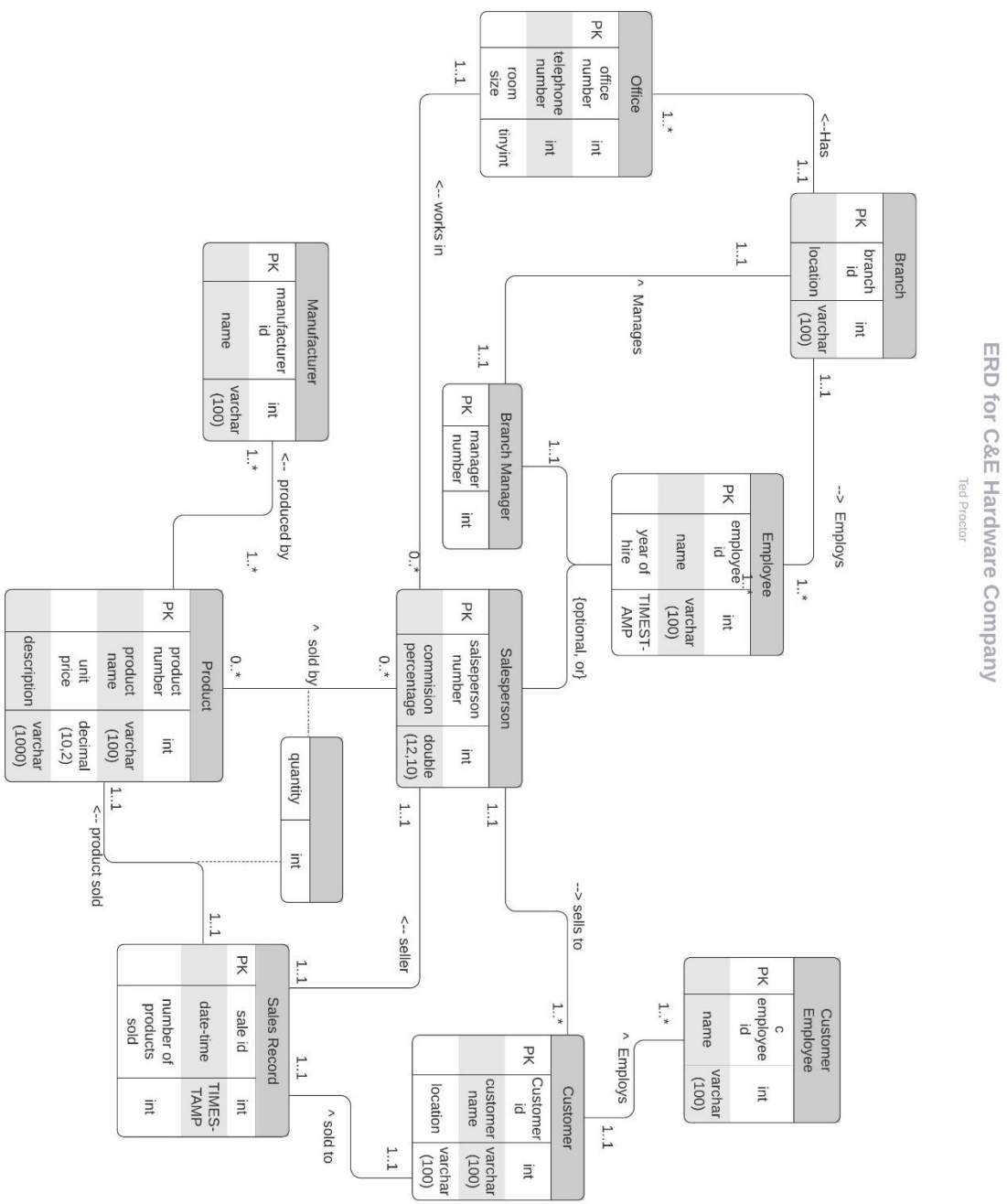
# ECM2419 – Hardware Company Database:



Ted Proctor

Hardware company ERD explained:

The diagram contains a number of tables, each with varying complexity regarding their attributes and relations to other tables or themselves. After a brief study of the specification, one can deduce that the Employee table, or more specifically the Salesperson table (which is a subclass of the employee table), is the principal table in the system, due to the fact that many other tables relate or rely on it. The Employee table has three attributes; 'employee id', 'name' and 'year of hire'. The 'employee id' field was added as the primary key, used to uniquely identify any employee. Although it was not explicitly stated that this field should be added I thought it was a pertinent addition as the only stated unique identifier for any kind of employee was for salesmen, despite other employees existing in the company. This identifier has been given the datatype int, as that has sufficient range to accommodate a very large number of employees, meaning that company growth will not affect the database after time has passed. I also thought the addition of a 'name' field would be beneficial as it allows for a user of the database to more easily identify a subject of the table. The name field has been given the datatype varchar(100), as this allows for a string of characters with length less than 100, which I thought was a reasonable length to include all names. The final attribute native to this table is 'year of hire'. This attribute contains a timestamp from when the employee was first hired, containing both the date and exact time of employment. This attribute was described in the specification under the Salesperson section, but I thought this information would be useful for a number of working roles, so I thought it should be put in the superclass of Salesperson (employee).

The employee table has two subclasses: – Branch Manager and Salesperson. The Branch Manager table acts more as a role identifier, allowing us to differentiate between the manager and the rest of the employees, with a single attribute acting as both the primary key and a unique identifier for each manager in the table, of the name 'manager number' and of the type int. The Salesperson subclass is the most frequently used table in the database as it has many connections to other tables. It possesses two attributes; 'Salesperson number' - which acts as the primary key for the table and is of type int, and 'commission percentage', which is of type double(12,10) (this is a range of values from 0 to 99 with a maximum of ten digits of decimal value). The subclasses have been given the options {optional, or}, as other employees may exist in the system (e.g., administrative staff, janitorial staff, etc) which would not fit into either category, and as far as the specification states I believe that there can be no crossover between the roles, meaning that they must be exclusive. In addition to this, the Branch Manager is part of an inherently recursive relation that moves between branch, employee and itself. This shows the administrative power capabilities of some employees and allows us to see the workplace hierarchy from an inspection of the database. The relationship between Branch Manager and Branch is defined by 'Manages' and it exists as a one-to-one ratio.

Another important table held within the system is the Branch table. This table could be considered the base table as all relations in the database are somewhat dependant on it due to the fact that all data regarding the affairs of the workplace should be stored under an easy generalisation that categorises different work forces and separates them such that they are compartmentalised into self-sufficient divisions. This method of organisation also allows users with high level clearance to easily examine any given branch, and from there look into greater detail at the semantics of labour within the branch itself. The table is possessed of two attributes; 'branch id' and 'location'. 'Branch id' acts as a unique identifier for each branch and the primary key, and is of type int. The location field is used to describe the geographical location of the branch and should store sufficient information to locate the branch, such as city, street, and postcode. I decided on an arbitrary length of 100 for the length of the allowed string describing the location of the branch, of type varchar(). It has a relation 'Employs' to the Employee table, which is of the topology one to many, where the branch employs many employees.

The class Office is directly related to the Branch class, defined by the relation ‘Has’. The Office table is also directly related to another important table – Salesperson subclass of Employee. This relation is defined by [Employee] ‘works in’ [Office] and is a one-to-many relationship, with one office being able to support 0 to many (dependant on the size of the office) salespeople. This mirrors the relationship between Office and Branch, as one branch will be able to support many offices. The office table has three fields: 'office number’, ‘telephone number’ and ‘room size’. All these attributes use some form of int type, with ‘room size’ using the type tinyint, which is a number from 0-255. This decision was made as I thought it was unlikely that an office would be able to support more than 200 individuals, as it would become more of a workspace and no longer an office. This also improves the space efficiency of the database. ‘Office number’ acts as the primary key for the table.

The next of the purlieus about the Salesmen subclass that we should examine are the tables that provide information for the products. There exists two tables here, Product and Manufacturer. The Product table is involved in three relations, the first of which being [Product] ‘produced by’ [Manufacturer]. This relation is of the cardinality many-to-many, with at least one of each instance being needed in the relation. This is because one manufacturer may produce many different types of products, and one product may be produced by many manufacturers, but to be contained with the system a product has to be present, meaning any related manufacturer has too as well. The next relation we are to examine is the relationship between Product and Salesperson, defined by ‘sold by’. This is another many to many relationship, with the lower bound being 0 instances of the other. This is due to the fact that each product in the system might not be sold by every Salesperson, nor might each Salesperson sell every product. There is a possibility that a Salesperson doesn’t sell any products (although he might not keep his job for too long) or that a product is not sold by any Salesperson (deadstock). This relation is also possessive of a relation attribute ‘quantity’ that is of type int and shared with ‘product sold’. This attribute is used to track the number of products purchased or consumed in a transaction, and is needed for the Sales Record. The final relationship that the Product table is involved in is defined as [Sales Record] ‘product sold’ [Product], and has a cardinality of one to one. This relationship is very simple as for each unique record of a product being sold the product has to be recognised, hence the relationship. This relation also uses the quantity relation attribute. The Product table has four attributes; ‘product number’, ‘product name’, ‘unit price’ and ‘description’. ‘Product number’ acts as the primary key and is of the type int. Although ‘product name’ was not explicitly stated in the specification, I thought it was a germane addition as it is a much more user-friendly identifier for a product, as opposed to an arbitrary number, and it is of the type varchar with a limit of 100 characters. ‘Unit price’ is of decimal type, bound to two decimal places, with 10 figures to account for the more significant digits. This is reflective of the semantics of money, and is sufficiently large to accommodate any reasonable value for a ‘unit’ of product. The final attribute we are to inspect is ’description’. This is another non-specified attribute that I added, however, much like the ‘product name’ attribute, I thought it would be beneficial for the user to have such data on standby, as opposed to perhaps having to generate this data on request, or sourcing it from an external source. This attribute is a varchar with length 1000, as I thought this was a sufficient amount to provide a brief but detailed description. The manufacturer table only has two attributes; ‘manufacturer id’ and ‘name’. The ‘manufacturer id’ is the primary key for the table, and is of type int. ‘Name’ acts as a user-friendly unique identifier for each manufacturer, and is of type varchar(100).

The next set of tables we are to explore are Customer and Customer Employee. The Customer table has a number of relationships with other tables, one of which being its connection to the Customer

Employee table. The nature of this relation is defined by [Customer] ‘Employs’ [Customer Employee], and is a one to many relationship, as one Customer (store) could employ numerous workers. The minimum bound for the many side of the relationship is one as a store needs at least one employee to function. The Customer table is also linked to the Salesperson table by the relationship [Salesperson] ‘sells to’ [Customer]. The cardinality of this relationship is one to many, as one Customer should only deal with a single Salesperson, but a Salesperson may sell to numerous different Customers. The final relation this table is involved in is the relation [Salesperson] ’sold to’ [Customer]. This relation is of cardinality one to one, and is very similar to the relationship ‘product sold’ between the Sales Record and Product table, as it merely acts as an identifier for a useful piece of information for a previous sale. The customer table is possessive of three fields, containing a field ‘customer id’, which acts as the primary key for the table and is of type int, ‘customer name’ which is a varchar of length 100 and is a user-friendly identifier for the Customer, and ‘location’. ‘Location’ contains a string (type varchar(100)) which contains an adequate description of the Customers location (e.g. postcode, street address, floor, etc). The Customer Employee table has two attributes; ‘c employee id’ and ‘name’. ‘C employee id’ is an int type and is the primary key of the table. ‘Name’ acts as a user-friendly identifier for each Customer Employee.

The final table in the system is the Sales Record table. This table has a number of relationships with other classes (although most of them have been covered), including ‘sold to’, ‘product sold’ and ‘seller’. As the first two entries in the list have been described elsewhere, I shall only examine the final attribute listed – ‘seller’. This relation exists between Salesperson and Sales Record as [Sales Record] ‘seller’ [Salesperson], and is of the cardinality one to one.

Ted Proctor

# ECM2419 – Data Normalization coursework

## Original data:

As one can see when first examining the given dataset, it is in a disorganized and hard to read mess which is in dire need of normalization. The given data has numerous fields with a range of values, examples of such being ‘Product Number’, ‘Product Name’, ‘Unit Price’ and ‘Quantity’. These can make the database hard to read due to the inconsistency in returned queries, making it look jumbled and poorly organised. Repeating groups are also present in the database, and they cause issues when records must be updated, as one record might have been neglected in the update and therefore a lack of continuity in the dataset might appear. This occurs when instances of a product number (e.g. 6722) or another similar attribute have the same value repeated numerous times in one column, and create a large amount of redundant data. This decreases the efficiency of the system and will result in the company incurring higher cost for maintenance and running cost of the server.

## First normal form:

The first normal form requires that there are no repeating groups, or more formally ‘no attribute domain has relations as elements’ (Wikipedia article on the first normal form). To do this, we need to split any rows of attributes with repeating fields into a number of composite rows, until no repeating groups remain. This leads to the issue of there being no unique identifier for each row. We could use quantity, since so far all entries have been unique, but this would lead to a number of complications down the line, so I decided it was a pertinent decision to create a new attribute ‘transaction Number’. This field is a unique identifier for each row and thus acts as the primary key of the table. Another two fundamental products of the first normal form are the invariance of attribute domains and the idiosyncrasy of attribute names. This means one field cannot host numerous element ‘types’ (such as int, varchar(length), etc), and must have a unique name. Fortunately, the dataset we have been given already obeys those laws and no changes have to be made. The final rule of the first normal form that must be obeyed is that the order in which data is stored doesn’t matter. This issue was fully rectified by the introduction of the transaction number field, although it would not have had a very large impact on the dataset before. These operations should remove any deletion anomalies, as the data should be in a form that has no incorrect values at this stage. Once all of those rules have been adhered to, we get the following table:

| Transaction Number | Salesperson Number | Product Number | Salesperson Name | Commission Percentage | Year of Hire | Department Number | Manager Name | Product Name | Unit Price | Quantity |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 37 | 6722 | Baker | 10 | 2005 | 73 | Scott | Pliers | 11.50 | 688 |
| 1 | 37 | 4013 | Baker | 10 | 2005 | 73 | Scott | Saw | 26.25 | 170 |
| 2 | 37 | 9440 | Baker | 10 | 2005 | 73 | Scott | Hammer | 17.50 | 473 |
| 3 | 86 | 6386 | Adams | 15 | 2001 | 59 | Lopez | Wrench | 12.95 | 1745 |
| 4 | 86 | 9440 | Adams | 15 | 2001 | 59 | Lopez | Hammer | 17.50 | 2529 |
| 5 | 86 | 1765 | Adams | 15 | 2001 | 59 | Lopez | Drill | 32.99 | 1962 |
| 6 | 86 | 4013 | Adams | 15 | 2001 | 59 | Lopez | Saw | 26.25 | 3071 |
| 7 | 14 | 1765 | Johnson | 10 | 2008 | 73 | Scott | Drill | 32.99 | 809 |
| 8 | 14 | 6722 | Johnson | 10 | 2008 | 73 | Scott | Pliers | 11.50 | 734 |
| 9 | 61 | 6386 | Davies | 20 | 2001 | 73 | Scott | Wrench | 12.95 | 3729 |
| 10 | 61 | 1765 | Davies | 20 | 2001 | 73 | Scott | Drill | 32.99 | 3110 |
| 11 | 61 | 6722 | Davies | 20 | 2001 | 73 | Scott | Pliers | 11.50 | 2738 |

Ted Proctor

Second normal form:

The second normal form requires the data to be fully functionally dependent. This means that a relation must not contain any partial dependency and that any attribute must not be dependent on a subset of another attribute. This results in tables with every non-primary-key attribute being fully functionally dependent on the primary key. In our data it means two new tables must be formed 'Salesperson table', which contains all the details regarding salespeople, and 'Product table', which contains the details regarding product information, and massively reduces the redundancy of the table as many repeating values are eliminated. This decision was made as a number of attributes were functionally dependent on 'Salesperson Number' and 'Product Number', which resulted in repeated values that could be reduced. The 'Salesperson table' has the attributes 'Salesperson Number', 'Salesperson name', 'Commission percentage', 'Year Of Hire', 'Department Number' and 'Manager'. Each of these values were functionally dependent on 'Salesperson Number', and due to its uniqueness, it can be used as the primary key in a new table. The 'Product table' has the attributes 'Product Number', 'Product Name' and 'Unit Price', with name and price being functionally dependent on 'Product Number'. This should result in the removal of all insertion anomalies as individual tables will be updated, meaning no blank fields will be left (for example, if a salesperson made no sales). Now that all partial dependencies have been removed, the tables should be in second normal form as follows:

Salesperson table

| Salesperson Number | Salesperson Name | Commission Percentage | Year of Hire | Department Number | Manager Name |
|---|---|---|---|---|---|
| 37 | Baker | 10 | 2005 | 73 | Scott |
| 86 | Adams | 15 | 2001 | 59 | Lopez |
| 14 | Johnson | 10 | 2008 | 73 | Scott |
| 61 | Davies | 20 | 2001 | 73 | Scott |

Product table

| Product Number | Product Name | Unit Price |
|---|---|---|
| 6722 | Pliers | 11.50 |
| 4013 | Saw | 26.25 |
| 9440 | Hammer | 17.50 |
| 6386 | Wrench | 12.95 |
| 1765 | Drill | 32.99 |

Transaction table

| Transaction Number | Salesperson number | Product Number | Quantity |
|---|---|---|---|
| 0 | 37 | 6722 | 688 |
| 1 | 37 | 4013 | 170 |
| 2 | 37 | 9440 | 473 |
| 3 | 86 | 6386 | 1745 |
| 4 | 86 | 9440 | 2529 |
| 5 | 86 | 1765 | 1962 |
| 6 | 86 | 4013 | 3071 |
| 7 | 14 | 1765 | 809 |
| 8 | 14 | 6722 | 734 |
| 9 | 61 | 6386 | 3729 |
| 10 | 61 | 1765 | 3110 |
| 11 | 61 | 6722 | 2738 |

Ted Proctor

Third normal form:

This is the process of removing all transitive dependencies from the tables. Fortunately, there only exists one transitive dependency, which is 'Manager' depends on 'Department Number'. To resolve this issue, we create another table 'Manager table' with 'Department Number' as the primary key and 'Manager' existing as a candidate key. This eliminates all modification anomalies, as there should now be no repeated values in the tables. This is the final form we encounter and is equivalent to a lossless compression technique, as it results in a massive decrease in size of table(s) without losing any data. The subsequent tables should look like this:

Salesperson table

| Salesperson Number | Salesperson Name | Commission Percentage | Year of Hire | Department Number |
|---|---|---|---|---|
| 37 | Baker | 10 | 2005 | 73 |
| 86 | Adams | 15 | 2001 | 59 |
| 14 | Johnson | 10 | 2008 | 73 |
| 61 | Davies | 20 | 2001 | 73 |

Manager table

| Department Number | Manager Name |
|---|---|
| 73 | Scott |
| 59 | Lopez |

Product table

| Product Number | Product Name | Unit Price |
|---|---|---|
| 6722 | Pliers | 11.50 |
| 4013 | Saw | 26.25 |
| 9440 | Hammer | 17.50 |
| 6386 | Wrench | 12.95 |
| 1765 | Drill | 32.99 |

Transaction table

| Transaction Number | Salesperson Number | Product Number | Quantity |
|---|---|---|---|
| 0 | 37 | 6722 | 688 |
| 1 | 37 | 4013 | 170 |
| 2 | 37 | 9440 | 473 |
| 3 | 86 | 6386 | 1745 |
| 4 | 86 | 9440 | 2529 |
| 5 | 86 | 1765 | 1962 |
| 6 | 86 | 4013 | 3071 |
| 7 | 14 | 1765 | 809 |
| 8 | 14 | 6722 | 734 |
| 9 | 61 | 6386 | 3729 |
| 10 | 61 | 1765 | 3110 |
| 11 | 61 | 6722 | 2738 |

Assumptions:

A number of assumptions were made when developing this normalized database. They are as follows:

- There exists only one manager per department.
- The data in each row correlates to its other members.
- All tools of the same type are of equal value.