**Database Design 1. Design Overview & Shipment DB**

A Database System is made up of a Database Management System (DBMS) and a database (of data values). So DBMS is the database software e.g. MySQL.

In Database Systems (DBS) Design, the fundamental design concepts are called

* dependencies and
* normal forms

These two concepts are used in a design process called normalisation. DB design in general, and particularly these concepts, can be difficult to understand for some students.

In response to this, we will introduce the notion of design using aspects of what called ‘logical modelling’ with the objective of making ‘common’ sense of the more formal and technical normalisation process. Some students understand the more logical abstract approach, others the formal data processing approach of normalisation.

Rather than starting with confusing terms and concepts first, we will start by informally examining problems of organising data into databases/tables and then try to work backwards to show that some formal design system(rules, terms, procedure etc) is required to develop a standard approach to database design.

However, there is no magic formula that can be applied to a design. It is by nature a problem to be solved. Since complex applications always entail a mixture of design elements, the biggest problem for the designer is how to break the large complex problem up into smaller modules that can be solved by using simpler design building blocks and experience.

Many problem solving strategies involve ‘pattern recognition’. In this course we will try to develop your ability to recognise similarity (a pattern) between previous design and a new problem i.e. you build up design experience that helps you recognise aspects of a new problem. All designs are similar and share many logical aspects. You should see this as a pattern that builds up as we deal with each new design problem. However, not all students will make good designers.

The course notes for each design are not given in advance. The notes are designed to develop your problem solving capabilities. You will be given a series of problems to solve; at each stage we will only introduce a new design concept in response to problems we encounter on the way. This prevents students looking ahead and avoiding engaging their problem solving skills. So be careful to 1. attend each class and 2. organise/order your notes correctly.

List of designs (and handouts)

1. Shipment (database used in C.J. Date Database Systems text book)
2. College Marks
3. Web Sales exercise
4. Line item design problem
5. Dream Home Estate Agency (from Connolly & Begg recommended text book)
6. Modelling
7. Photocopy of the normalisation chapter in the recommended text book.

**Shipments Database**

Databases are essentially a computerised version of a filing cabinet for paper records/folders. We will start by examining a paper card/file format or paper filing system. See Appendix for examples of paper forms that are used in CIT.

Imagine an office with a stack of paper receipts; or a folder full of insurance claim forms, or a filing cabinet full of college applications etc.

Supplier Name Address

S1 Smith London Card No 1 in the list

Shipping list

PartNo Name Qty

* P1 Nut 200 Note: only one entry here

Supplier Name Address

S2 Jones Paris Card No 2 in the list

Shipping list

PartNo Name Qty

* + P3 Screw 400 Note: two entries here
  + P5 Wheel 100

S3 Blake Paris Card No 3 in the list

* P3 Screw 200
* P4 Screw 500
* P5 Wheel 350 Note: Three entries

Use the CIT paper forms (Appendix) and visualise hundreds or millions of these records. You should realise that a person has designed the paper form to be that structure. We will be designing the database to store the data instead of the paper.

The first aspect of paper forms fact that each card has a variable number of entries in it is an issue for the design.

**Data like this is called a repeating group.** We can think of it logically as a single element (a list), but in fact it can be broken down into smaller subcomponents (the items on the list).

Repeating groups cause problems for design because you do not know in advance the number of items on any given list.

1. Relational databases only allow one data structure called a table. **A table has some important properties**. The fixed grid structure
   1. has a uniform number of columns i.e. each row has a fixed number of columns.
   2. the intersection of a row and a column can only have one data value. i.e. NO REPEATING GROUPS

Using variable length strings (text data types) to store a list is not allowed in relational DBMS e.g. ‘P3, P4, P8’. One character string representing many shipments is too complex to store and search efficiently; so RDBMS ban them as a general design rule.

Note: the alternative is to use a programming language to read and process the text string (in real time) and extract the different items in the list (string processing functions). This may be acceptable for small amounts of data, but in large data systems this slows the processing of each row (record) significantly. These 2 rules essentially **simplify(& speed) the I/O disk read** of a set number of bytes per record.

**Good design promotes scalability. How?**

Logically, a table can have any number of rows. Note, there will be some limit due to the physical memory in the computer. For example, a USB memory stick might store e.g. 4 Gigabytes. So, the number of rows may be variable depending on the physical storage, but the logical design is the same. We call this **design scalable** i.e. the same design can handle a small DB of 100 records just as it can handle a large DB of a million records.

1. The student should now attempt to transform the ‘paper system’ from page one into a grid/table structure? This is database design, i.e. designing a set of tables to store data.

**The designer can use any number of tables** as required to store the data **correctly** (NB according to the properties/rules).

Table(s) MUST not grow across columns, only down the rows. In other words, when you add new data to a database, you can only add a new row to one of your tables. Adding columns changes the design, and therefore every search (program) and Graphic User Interface(GUI) form based on the old design would have to be reprogrammed each time you add columns. So, in Relational DBMS adding columns is only allowed in exceptional circumstances e.g. add new data like email address.

When a table you organise (design) adheres to certain rules, it is in a normal form (e.g. 1NF means first Normal Form). So, when we take data and structure it according to rules, we call this **normalising the data**.

Hint: If you are confused try use one table to store all the data. It is important to note that a single table storing all the attributes is a design; a simple one, but still a design.

Excel can only handle one list; if we use only one table, effectively Excel and Access would be very similar design wise.

Examining data to see how it might be organised into tables can be thought of as ‘top down’ design i.e. start with one large organisation of data and break it down into smaller elements.

1. In class examine different designs/attempts from the students and discuss problems (usually students will not adhere to the fixed grid structure).
2. Design approach: Concentrate on a one table solution. i.e. list all attributes in the paper form. We will add in another attribute for part, part weight. See Appendix 1(p12)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Sno | Name | City | Pno | Name | Weight | Qty |
| S1 | Smith | London | P1 | Nut | 12 | 200 |
| S2 | Jones | Paris | P3 | Screw | 17 | 400 |
| S2 | Jones | Paris | P5 | Cam | 12 | 100 |
| S3 | Blake | Paris | P3 | Screw | 17 | 200 |
| S3 | Blake | Paris | P4 | Screw | 14 | 500 |

Take some time to analyse/examine this solution for possible problems:

Hint: patterns of repeated data in a table can cause problems? How? Note in this case, the same data is repeated, but it is not a repeating group as in a list on paper.

Bad design suffers from

* Waste of space, resulting in
* Increase search time and data transfer time from the disk to processing memory,
* **Processing anomalies**? A processing anomaly is an undesirable side effect of an Insert, Update or Delete operation. The easiest anomaly to understand is Update where the table contains replication of data. If data is replicated i.e. is in more than one place, then a user might update one instance of the data but not the other, thereby leaving the database in conflict; or confused. E.g. S3 Blake Paris; S3 Blake Cork. In database systems we say that the database becomes inconsistent. Which is correct address, Paris or Cork?

Before we can discuss the Insert and Delete anomalies, we need to add one more property to our list that defines an acceptable table design.

iv) **each row in a table must be unique**.

In other words, no two rows can be the same. This is a fundamental rule of the Relational database model.

At first this property might seem odd considering that some small amounts of repetition seem unavoidable. However, when you think about it, there can be no real need for having two rows with the exact same data in each column. Some of the data can be repeated, but not all of it.

In addition, if we look at it from a system point of view, when the DBMS is searching the long term storage (disk) for information it would a great advantage to know that once it finds a row, that it is guaranteed that there is no other version (copy) somewhere else on the disk. That is, **it can stop searching once it finds it.**

If we didn’t have this property, the DBMS would have to assume that there might be copy somewhere and it would therefore have to search the entire data file (on disk) from start to finish every time it’s looking for a record. This would be like going to a filing cabinet and having to look at every single entry, every time…. very slow and inefficient (i.e. requiring lots of work). So, we adopt this new property (unique rows).

If rows are unique then it follows that we must be able to use the data in each row to distinguish it from any other row. We call this important data, key data.

**Definition: A key is the minimum number of attributes that distinguishes one row of a table from another**.

Most simple tables (objects) have a single attribute key. However **composite key** are also possible in situations where more than one attribute is required to distinguish each row. E.g. surname may not be enough; so try (surname, firstname) combination. This possibility may not be enough etc.

You may have a situation where you have **more than one key available to you** e.g. for a student record we may have attributes CAO# and CIT\_Student#; both are called **candidate keys.** Choose one and it is the **Primary Key**

**Why minimum number of attributes?**

Because we search tables using access mechanisms such as indexes. Indexes must be as small as possible to be stored and searched efficiently. Think about an index in a text book. It is small, to look up quickly, and then access the correct page of the main data book.

Think of it this way, what if you define the primary key as all the columns in the table? It works, as it will identify each row. However, now your index is the same size as your book. It will take as long to search the index as the book itself. Useless!

Ex: What is the Key of the first design single table solution? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Now think about the Insert and delete anomalies in light of these new design concepts called keys (particularly composite keys). We shall return to these in the next design application (College marks) in Design 2 handout.

**Introduction to database design using Modelling.**

5. Now we will discuss an alternative to normalising data. We’ll call it, logical design using modelling as an alternative approach. Modelling is design that uses an abstract view of the system, where the designer analyses the meaning of the application to create a collection of separate objects as the starting design. The Designer may then refine to eliminate some objects e.g. by merging.

Logical design focuses on the meaning of the application to design tables. It doesn’t require actual data to formulate a design. It examines the specification to make its design decisions. It starts at a high abstract level to generate a starting design and then refines that. It can then be translated into to structures to hold the data.

So, we have two types of design. **Database normalisation and Modelling.**

As a (very) general rule of thumb,

**Normalisation is very effective if the system to be computerised exists already i.e. has data already stored in paper forms and the analyst can see redundancy and patterns in the data.**

**Modelling is effective for new build applications.**

**Note, both can be used together to tackle a design ‘from both angles’.**

6. Complicate the problem by adding in more columns. In class, we use smaller database applications for simplicity, but in the real world, there are usually much more descriptors for each object to deal with.

**In database systems terminology, columns represent descriptors or attributes for each object e.g. A part has a colour attribute, A supplier has a phone number, fax, bank account, email attribute etc. We will use the terms attribute and column interchangeably.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Sno | Name | City | Phone\_No | Pno | Name | Colour | Qty |
| S1 | Smith | London | 2343234 | P1 | Nut | Red | 200 |
| S2 | Jones | Paris | 2412343 | P3 | Screw | Blue | 400 |
| S2 | Jones | Paris | 2412343 | P5 | Cam | Blue | 100 |
| S3 | Blake | Paris | 2224344 | P3 | Screw | Blue | 200 |
| S3 | Blake | Paris | 2224344 | P4 | Screw | Red | 500 |

1. In Class Exercise: Students should write a short description of the application (succinct). One/two line max. What is the paper card storing data about?
2. Analyse the text of the description and identify nouns and verbs from different students. **Nouns can be seen to be variables**. These are identifiable objects (Person, Product, Vehicle etc). They exist on their own (independently of other data); they also have attributes that describe them (name, price, model etc). What are the variables in this application?
3. What is Qty in this application? Does it exist on its own? If not, then it must be an attribute that describes something else.

There can also be **associations or relationships between different objects**. **Verbs in a specification usually indicate an association between objects**. E.g. Person drives car; Patient attends Doctor; Customer rents Video, Student loans Book. An association can be described by an attribute e.g. qty, date etc.

Identify the objects and associations and examine if you can use a table for each one?

Use modelling (nouns, verbs) as major entities/objects. These are the main variables of the system. They have attributes that usually identify each instance of the variable and they have other descriptors or attributes that describe features of the object e.g. name, address etc, with an instance of an object being Smith, Cork. Compare how the repeated data usual corresponds to logical objects.

10. In class exercise: How could the single table solution (from 4 above) be broken up into smaller ones to eradicate the problems identified? Or put another way, how can the single table solution be broken up to better reflect (or model) the individual real world objects that are contained in it? Hint: use keys as a link between tables.

Possibly examine and discuss attempts by students.

11. At this stage we need to introduce the formal normalisation DB design concepts of functional dependency (FD) and normal forms(NF).

**Definition: A table is said to be in first normal form (1NF) if it adheres to 3 basic properties of tables**, 1. Fixed columns(grid) structure;2. No repeating groups 3.Key

**To handle different design situations we will add more properties to ensure that our table designs do not suffer any processing anomalies; as we add a new property we will define a higher normal form.**

**So, 1 NF adheres to basic rules, 2 NF extra rule, 3NF another extra rule.**

**Definition: A dependency is where the value that appears in one attribute is determined by another attribute (or is dependent on another). In text books these are called Functional Dependencies (FD). If B is dependent on A, we write**

**A 🡪 B NB: where A can be composite.**

In class discuss why if the values are the same then it is **not always true that the inverse is true i.e. B-> A.** Write a note on this here

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Note: the formal definitions used in databases systems seem too ‘mathematical’ and some students find them confusing. This is why we link the DB normalisation process to the more ‘common sense’ logical modelling whenever possible. Also we have seen an example of design so the concept of FD should be easier to understand if you can relate it to what you have seem and thought about in the shipment design.

**In DB design (normalisation) the strategy is to isolate related groups of FDs in different tables.**

12. Students should now list the dependencies in the shipment design application.

Ex: What is the Key of the first design? \_\_\_\_\_\_ Is there a FD?\_\_\_\_\_\_\_\_\_\_\_\_

Note: the attributes that describe an object are functionally determined by the key of that object or association. So the list of dependencies should match the attribute groups discovered by using modelling; the student should start to realise the link between the two design methodologies. That is, they should generate the same design.

Use modelling and/or data design to create more than one table to store the data in a more efficient way i.e. in a way that solves the problems identified earlier. How can different tables be linked?

You should realise that both design methodologies arrive at a similar final design by using a number of tables to store the data in a correct way, but one that is scaleable for any number of records i.e. the design works whether it has to handle 10, 10000, or 100000 records. Note a computer system for a small Cork company, a company handling Munster, nationwide or even international.

2nd Normal Form?

|  |  |  |
| --- | --- | --- |
| Sno | Pno | Qty |
| S1 | P1 | 200 |
| S2 | P3 | 400 |
| S2 | P5 | 100 |
| S3 | P3 | 200 |
| S3 | P4 | 500 |

Exercise : write out the remaining normalised tables with data allocated.

Note: From modelling we might have thought that this application has three major elements: simple objects (Supplier, Part) and an association (Shipment).

In addition, we might also have identified that each of these elements had attributes that described them i.e. Supplier has one key attribute Sno and other descriptor attributes( name, address, phone\_no etc) and Part has Pno (key) and descriptors (name, colour, weight) and finally Shipment (qty, date etc).

As we have already noted, these attributes are described in DB normalisation as functional dependencies (FD’s). Our initial one table design is not very good because it mixes these elements together in one table when in fact they should (could) be isolated in their own tables. **In normalisation terms, it mixed dependencies**; In modelling terms it mixed objects and associations all in one table.

**Note: FDs corresponds with the objects and the attributes that describe an object. All attributes of an object are functionally dependent on the object identifier (i.e. the key). A FD just describes a pattern of data that is repeated.**

Ex: What is the key of the table representing the association? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Notice that this key is also the key for our (starting) one table solution.

**Important observation: if a top down DB design mixes objects and associations (which it must do if it is one large table for all the data attributes), the association is always the more complex element in the mix and therefore it usually dictates the key of the (single) large table.**

Final normalised design for the shipments database

Supplier Part

|  |  |  |
| --- | --- | --- |
| Pno | Name | Colour |
| P1 | Nut | Red |
| P2 | Bolt | Green |
| P3 | Screw | Blue |
| P4 | Screw | Red |
| P5 | Cam | Blue |

|  |  |  |  |
| --- | --- | --- | --- |
| Sno | Name | City | Phone\_No |
| S1 | Smith | London | 2343234 |
| S2 | Jones | Paris | 2412343 |
| S3 | Blake | Paris | 2224344 |
| S4 | Clark | London | 5224344 |
| S5 | Adams | Athens | 4532111 |

Shipment

|  |  |  |
| --- | --- | --- |
| Sno | Pno | Qty |
| S1 | P1 | 200 |
| S2 | P3 | 400 |
| S2 | P5 | 100 |
| S3 | P3 | 200 |

In our example, we have a simple object Supplier, it has a key attribute Sno; Part is a simple object and has key Pno but the association between the objects requires the keys of the participants i.e. a compound key comprising of both the keys of the participants (Sno,Pno). Therefore, in the single table solution (Sno,Pno) composite is the key. So, if you use your modelling skills to identify an association, then identify an identifier for it, you can use that as the key of the large table used as the point for normalisation.

Note: This is called **a non loss decomposition** because all the information that was in the original table is now in the new set of tables.

**Definition of non loss**: W**e can always re-combine these separate tables into the original single table without loss of data (rows) or the addition of any previously unknown rows.**

In fact since our final design has no processing problems we can say it is a ‘better’ representation of the real world.

For database programming this recombination is a very common requirement. And we shall return to this in the database programming section. Throughout this course we will see the consequences or knock on effects that result from the use of normalised tables as the basis of data storage. These include:

* data retrieval i.e. how data is retrieved from many tables (see SQL section later), and
* how can we know that the set of tables we form by breaking up a large single table is correct. (we’ll need definitions and rules for correctness, see design 2 next)
* how data is organised on the disk for optimum efficiency

**Role of the Analyst:**

We have introduced many concepts like keys, identifiers, objects etc. Many of these may have nothing to do with business application itself; the manager of the Shipment application understands supplier and parts etc NOT keys, objects and associations.

The developer understands and documents the business logic using techniques like modelling objects. Understanding the exact rules of the business so that it can be computerised is the work of the analyst. (S)he is responsible for interacting with the external client (owner/manager of business). S(he) must find out e.g. by interviews, questionnaires etc, the meaning of each attribute and its relation to other attributes before design begins. Any problems/errors mean that the computerised solution is compromised and may need full reprogramming.

Aside: Most infamously costly(bad) computer systems stem from poor analysis, followed by incorrect design; and the errors only come to the surface after some programming has been done e.g. after an initial prototype has been developed. Usually by then so much money has been spent on the project that they decide to continue and either redesign the entire system or try program ‘around’ the design problem….usually not a good idea. Note HSE medical project; Air traffic control etc.

You must complete an analysis of a problem/application before you start designing a solution for it. That is, you must work out a definition or an exact description of the problem to be solved before you spend time implementing possible solutions. Many failures in IT systems stem from inadequately defined problems; these problems only come to light after a lot of work has been put in trying to develop a solution. In many cases all of this work is worthless.

So in database terms, don't go implementing tables & GUI forms & reports until you are fairly certain that you know what you are doing.

As a general rule, make sure there is a paper trail for all decisions. Do not be the one to blame, and since the customer is always right, the onus is on you to make sure you get them to 'sign off' on a contract of work based on an accurate description of the system.

Another aspect of good analysis is 'future proofing'. This means that you identify potential problems that may occur with the current system. For example, over time, the business may grow and develop. Will your database be able to respond?

Any/every item (attribute) of data that is to be stored should have a use? (otherwise why store it?)

In some situations, you may decide to store calculated values. This is a trade off in design. If I store the data (e.g. a total field) then I use up storage and increase the amount of data that is processed. However, I save on the cost of performing the calculation every time that total value is required in a query or report i.e. calculate it once, store it, and never need to recalculate it again. However, if I do not store that calculated field, then I save of space, but I must do the recalculation in the queries & reports.

Most clients will not know how to answer a question like 'how many tables'. They can describe the business, not the database design. You cannot ask a client is 4 tables is ok; if they knew the design, you wouldn't be needed.

Can a client tell you where to put an attribute? A client usually can't tell you where to put an attribute (that's the data base designers work i.e. you). They can tell you the meaning of the attribute.

Don't be misled by the names appearing on the paper form; look beyond the names to the meaning. Borrower, Authoriser, Issued by, are really all the one. i.e. employees.

It is dangerous to do design work without clearing the analysis first.

You were also told in class to fill in dummy data. This will test how your design stores the data, if you see replication then that usually indicates problems. So, using dummy data values can test a design.

So a major role of the Analyst is to find every data attribute to be used in the database, know what it’s used for; find is it related to anything else; what controls might be suitable for it (data type, required field, restricted values, security) etc.

Learning objectives of this section: To be able to define or apply the following concepts when appropriate:

1. the basic properties of tables; What are they and why are they used.
2. scaleable design
3. what is a repeating group, give eg, and discuss in terms of design.
4. Why do RDBMS ban repeating groups as a design rule?
5. How are repeating groups handled in other data processing systems?
6. Why do repeating patterns of data (e.g. groups) cause problems for DBMS.
7. Adding columns to a design is allowed only on exceptional circumstances, why?
8. Why are keys made up of the minimum number of attributes needed?
9. the problems associated with bad design, use examples.
10. normalising or normalisation (design from the top down), explain.
11. logical modelling (design from the bottom up), explain
12. how to compare and contrast the two design approaches.
13. When to use normalisation, when to use logical modelling.
14. how both design approached can be used to complement one another
15. be able to understand and use the terms attributes, keys, composite, primary, candidate, foreign key, functional dependency, normal forms
16. Why are functional dependencies one way (generally). A -> B.
17. non loss decomposition, explain.
18. be aware of the influence of normalisation on other aspects of the database system.
19. Role of the analyst.

The definitions I use in this course are the simplest I have found across many different text books. If you use other data sources, you may get varying definitions. So be careful that any source you use is accurate. For example: see section on Functional dependencies & Key constraints etc in

<Http://en.wikipedia.org/wiki/Relational_model>

Aside: Note that when the computer system reads data from the long term storage (disk) into processing memory it must be able to make sense of the binary data stream. In DBS as the number of columns is fixed, the DBS knows exactly how much data to read and allocate to each row.

Appendix 1: The single table (Excel style) is an example of a relational table design with dummy data filled in.

| **Sno** | **sname** | **status** | **city** | **Pno** | **Pname** | **Weight** | **Material** | **Colour** | **Pcity** | **Jno** | **Qty** | **U\_price** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S2 | Jones | 10 | Paris | P3 | Screw | 17 | Brass | Blue | Rome | J1 | 500 | 1.2 |
| S3 | Blake | 30 | Paris | P3 | Screw | 17 | Brass | Blue | Rome | J1 | 1100 | 1 |
| S1 | Smith | 20 | London | P1 | Nut | 12 | Steel | Red | London | J1 | 1500 | 0.25 |
| S5 | Adams | 30 | Athens | P6 | Cog | 19 | Graphite | Red | London | J2 | 1000 | 10 |
| S2 | Jones | 10 | Paris | P5 | Can | 12 | Brass | Blue | Paris | J2 | 700 | 22.5 |
| S3 | Blake | 30 | Paris | P4 | Screw | 14 | Graphite | Red | London | J2 | 1000 | 1.02 |
| S2 | Jones | 10 | Paris | P3 | Screw | 17 | Brass | Blue | Rome | J2 | 250 | 1.4 |
| S5 | Adams | 30 | Athens | P2 | Bolt | 17 | Steel | Green | Paris | J2 | 1300 | 12.5 |
| S4 | Clark | 20 | London | P6 | Cog | 19 | Graphite | Red | London | J3 | 475 | 10 |
| S2 | Jones | 10 | Paris | P3 | Screw | 17 | Brass | Blue | Rome | J3 | 1200 | 1 |
| S5 | Adams | 30 | Athens | P6 | Cog | 19 | Graphite | Red | London | J4 | 150 | 13 |
| S5 | Adams | 30 | Athens | P5 | Can | 12 | Brass | Blue | Paris | J4 | 250 | 25 |
| S5 | Adams | 30 | Athens | P4 | Screw | 14 | Graphite | Red | London | J4 | 600 | 1.05 |
| S2 | Jones | 10 | Paris | P3 | Screw | 17 | Brass | Blue | Rome | J4 | 900 | 1.05 |
| S5 | Adams | 30 | Athens | P3 | Screw | 17 | Brass | Blue | Rome | J4 | 350 | 1.25 |
| S5 | Adams | 30 | Athens | P2 | Bolt | 17 | Steel | Green | Paris | J4 | 1550 | 12.25 |
| S1 | Smith | 20 | London | P1 | Nut | 12 | Steel | Red | London | J4 | 1200 | 0.3 |
| S5 | Adams | 30 | Athens | P1 | Nut | 12 | Steel | Red | London | J4 | 350 | 1.2 |
| S5 | Adams | 30 | Athens | P5 | Can | 12 | Brass | Blue | Paris | J5 | 250 | 25 |
| S2 | Jones | 10 | Paris | P3 | Screw | 17 | Brass | Blue | Rome | J5 | 750 | 1.15 |
| S2 | Jones | 10 | Paris | P3 | Screw | 17 | Brass | Blue | Rome | J6 | 100 | 1.5 |
| S4 | Clark | 20 | London | P6 | Cog | 19 | Graphite | Red | London | J7 | 690 | 9.75 |

Appendix 2 Paper forms in CIT



