Application Description

TODO – CHANGE NAMES OF TABLES AS THEY CLASHED WITH KEYWORDS

Assert that ram slots is more than 0

We can also model partially build systems (assumption)

For this project I decided to a model computer system. Each computer system must consist of a cpu, cpu cooler, motherboard, ram, power supply, case, storage and optionally a gpu. The database describes systems that have been built and how the components relate to each other.

I’ve decided to represent each component as an entity and how that components interact with each other e.g. connect, as relationships.

This database could be used by a boutique pc systems builder such as origin pc or by a large scale systems builder such as dell or apple.

The following tables are what I have used reflect the real world semantics and attributes of a pc system in my database.

**The CPU table contains 7 attributes:**

serial\_number – this is a unique identifier for that exact cpu.

motherboard\_sn – this is the motherboard that this cpu is linked to.

name – the model cpu it is e.g. i5 3470k.

socket – the type of socket the cpu uses, this must match the socket of the motherboard as different sockets have different pin layouts.

tdp – the thermal design power of the cpu (how much power it dissipates in the form of heat)

core\_count – how many processing cores the cpu has.

smt – simultaneous multi-threading, can the cpu appear to have more cores than it actually has and as a result process more threads simultaneously.

**The CPU cooler table contains 6 attributes:**

serial\_number – this is a unique identifier for that exact cpu cooler.

motherboard\_sn – this is the motherboard that this cpu cooler is linked to.

name – the model cpu cooler it is e.g. corsair h100i

socket – the type of socket the cpu cooler mounts to, this must match the socket of the motherboard and the cpu the cooler cools.

fan\_rpm – the speed that the cooling fan rotates at ( rpm – revolutions per minute).

noise\_level – how loud the cooler is.

**The motherboard table contains 6 attributes:**

serial\_number – this is a unique identifier for that exact motherboard.

name – the model motherboard it is e.g. asus prime x370 pro.

socket – the type of cpu socket the motherboard has e.g. LGA1150

ram\_type – the type of ram that the motherboard accepts e.g. ddr4, ddr3 etc

ram\_slots – the number of ram slots a motherboard has

form\_factor – the type and size of the motherboard, e.g. atx, m-atx etc.

**The gpu table contains 6 attributes**

serial\_number – this is a unique identifier for that exact gpu.

motherboard\_sn – this is the motherboard that this gpu is linked to.

name – the model gpu it is e.g. nvidia geforce gtx 980ti

core\_clockk – the speed in mhz at which the graphics card operates e.g. 2000mhz

length – the length of the gpu, this is important when mounting the gpu in a case.

memory – the amount of vram (video ram) the gpu has, more vram is desirable for various simulation tasks.

**The ram table contains 6 attributes**

serial\_number – this is a unique identifier for that exact cpu cooler.

motherboard\_sn – this is the motherboard that this cpu cooler is linked to.

name – the model cpu cooler it is e.g. corsair h100i

number\_of\_modules – how many sticks of ram the kit contains.

type – the type of ram e.g. ddr3

capacity - the amount of space in gb

**The storage table contains 6 attributes**

serial\_number – this is a unique identifier for that exact storage device.

name – the model of storage device it is e.g. Samsung 850 evo

case\_sn – the case this storage device is mounted in

capacity – the amount of space in gb that the device has e.g. 500gb

interface – the type of interface it uses to connect to the rest of the pc e.g. sata, m.2 etc

form\_factor – the size and form of the drive e.g. 2.5inch, 3.5inch, pci-e.

**The case table contains 5 attributes**

serial\_number – this is a unique identifier for that exact case.

name – the model of case it is e.g. fractal define s.

storage\_bays – the type of storage bays the case supports e.g. 2.5inch, 3.5 inch.

psu\_sn – the power supply that is mounted in this case.

gpu\_space – the amount of space in mm that is available to mount graphics cards.

**The psu table contains 6 attributes**

serial\_number – this is a unique identifier for that exact power supply.

name – the model of power supply it is e.g. corsair cs600m.

wattage – how many watts of power the power supply can deliver e.g. 600 watts

modular – whether or not the cables an be removed from the power supply.

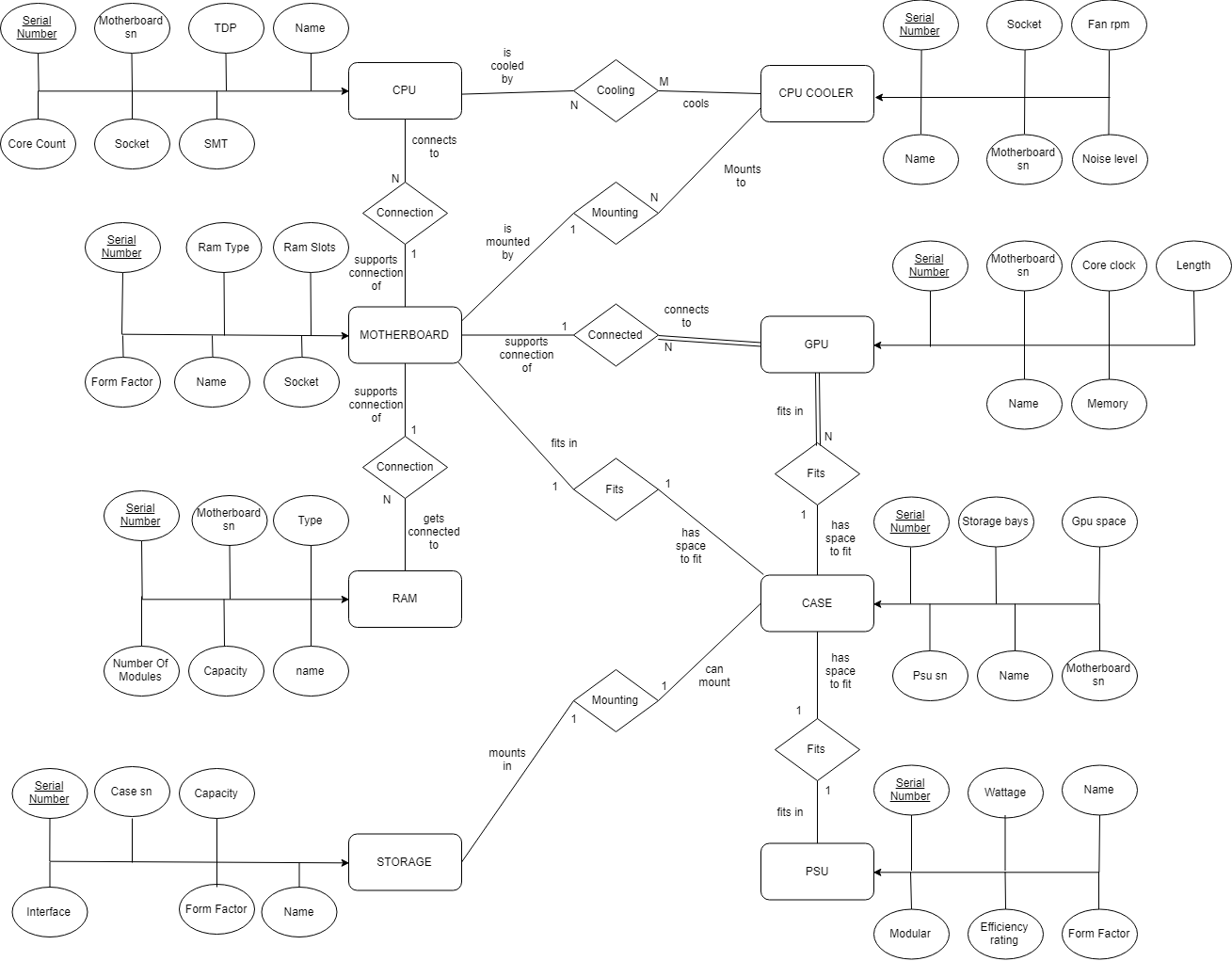
efficiency\_rating – how efficient the power supply is at converting to different voltages e.g. bronze, gold.

form\_factor – the size of the power supply e.g. atx, itx etc.

In my model I make the following assumptions:

* Certain cpus require more than one cooler to cool them. E.g. high performance server hardware
* cases only support one type of power supply.
* Pcs only contain one type of storage.
* Pcs do not require a dedicated gpu as most cpus have onboard graphics.
* Motherboards can have multiple sockets and therefore house multiple cpus.
* Each component has a unique id tied to it, this id uniquely identifies an instance of that component.
* Components cannot be reused across multiple pcs.
* Pcs that have been released to the public have serial numbers that are divisible by 2?

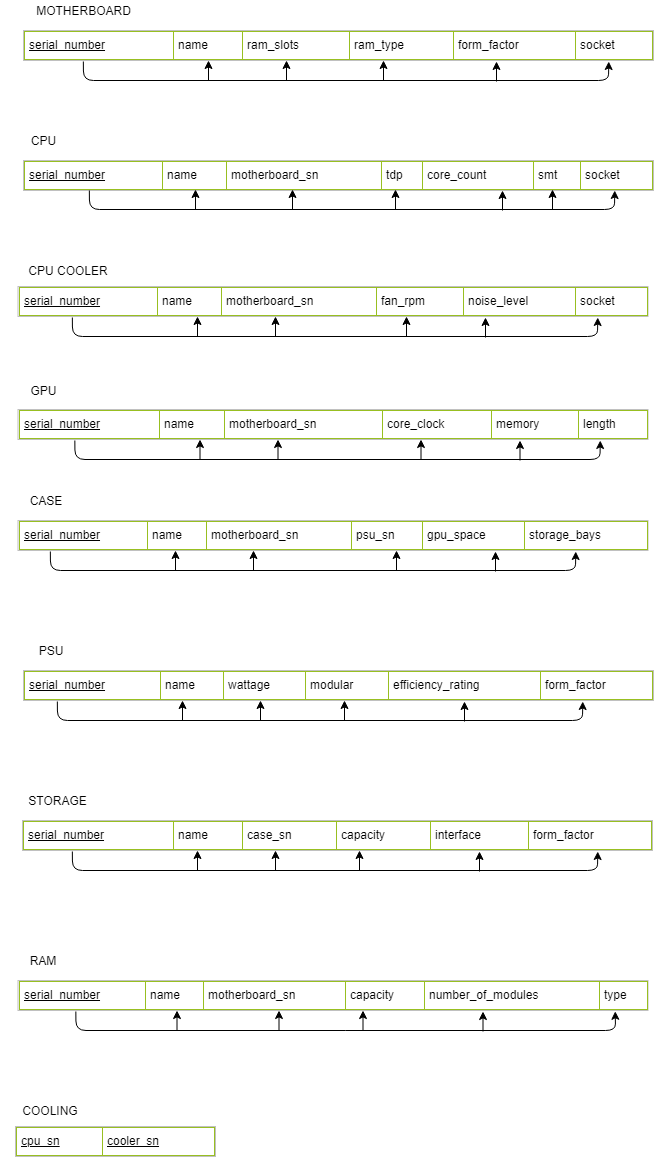
Entity Relationship Diagram



Relational Schema



Functional Dependency Diagram



Normalisation

After creating the first draft of my entity relationship and functional dependency diagrams, I began to see if any normalisation could be carried out. This is key in the database design process to avoid abnormalities when inserting or deleting values in the database tables. It also ensures the tables will be structured in such a way that they cannot contain redundant data. The rules for normalisation are called normal forms.

**First normal form**

A relation is in first normal form if the domain of each attribute contains only atomic values and the value of each attribute contains only a single value form that domain.

All relations were found to be in first normal form except for the case relation. Initially I had modelled the case relation with a storage attribute that would contain a list of storage drives that were inserted in the case. To remedy this issue I added a case\_sn attribute to the storage relation instead. This way each storage drive is linked to case using a relationship and all attributes of the case relation are now atomic.

**Second normal form**

A relation is in second normal form if in addition to satisfying the criteria for first normal form, every non-key attribute is fully functionally dependent on the entire primary key.

All relations were found to comply with the rules of second normal form.

**Third normal form**

A relation is in third normal form if in addition to satisfying the criteria for second normal form, no non-key attributes are transitively dependent upon the primary key.

All relations were found to comply with the rules of second normal form.

**Boyce Codd Normal Form (3.5 normal from)**

A relation is in Boyce Codd normal form if “all attributes in a relation should be dependent on the key, the while key and nothing but the key”.

All relations were found to comply with the rules of Boyce Codd normal form.

Integrity Constraints

The relational model describes integrity constraints. Three types of integrity constraints considered part of the relational model are:

**Key, Entity integrity, Referential integrity**.

The dbms must enforce these constraints and we must ensure that our sql operations do not violate any of these constraints

**Key Constraints**

Key constraints specify that there may not be any duplicate entries in key attributes. Keys are used to uniquely identify a tuple. This will be handled by the dbms for us.

**Entity Integrity Constraints**

Entity constraints are specified on individual relations, one such constraint is that none of the primary key may be null. In my case, entity constraints are handled by the dbms.

**Referential Integrity Constraints**

Referential integrity constraints are specified between two relations – they maintain consistency among tuples in the two relations

A tuple in one relation that refers to another relation, must refer to an existing tuple in that relation. A foreign key formally specifies a referential integrity constrain between two relations, for this we must be careful with our sql operations and define when foreign keys may or may not be null. E.g. we can only insert a cpu into the cpu table if it is not linked to a motherboard that already exists.

Semantic Constraints

Semantic constraints are vital to ensure the integrity of the database and avoid the accidental corruption of information. These constraints can be defined when creating the tables in the database and in the form of checks, triggers and assertions.

Security Constraints

Security constraints are concerned with deliberate corruption of data in the database. We can ensure database security using security policies and access control.

For security I have implemented different access control levels. Staff have full access to all the data but only at a read level, Administration and engineers have the ability read and write to the database and finally the public have access to only a view, that contains only currently released products, if a pc has not been released yet it will not be placed in the view. Pc components with no names can be viewed as not released, this analogous with real world semantics as product names are often decided much closed to release time.