Normalized Functional Dependencies

These are in the format of:

1.)The relational schema.

2.)The functional dependencies.

3.)The normal form in which the entites are in.

Vendor(URL, company)

FD: URL→Company

BCNF

Warehouse(Warehouse_ID, URL, Location)

FD: WarehouseID, URL→Location

BCNF

Stock(Item_ID, Warehouse_ID, Count, Current_price, Historical_Low, Historical_High, Sale_status, Shipping_price)

FD: Item_ID, Warehouse_ID →Count, Current_price, Historical_Low, Historical_High, Sale_Status, Shipping_price

BCNF

Item(Item_ID, Item_name, Category, Manufacturer, Series, Release_date, Time_since_release, Model_numbe

FD: Item_ID→Item_name, Category, Manufacturer, Series, Release_date, Time_since_release, Model_Number

BCNF

CPU(CPU_ID, Chipset, Integrated_graphics, Wattage)

FD: CPU_ID →Chipset, Integrated_graphics, Wattage

BCNF

Memory(Memory_ID, Memory_capacity)

FD: Memory_ID →Memory_capacity

BCNF

Storage(Storage_ID, Capacity, Storage_type, Storage_standard, Form_factor, Wattage)

FD: Storage_ID →Capacity, Storage_type, Storage_standard, Form_factor, Wattage

BCNF

Motherboard(Motherboard_ID, Chipset, Num_Usbports, Network, Form_Factor)

FD: Motherboard_ID→Chipset, Num_Usbports, Network, Form_Factor

BCNF

Monitor(Monitor _ID, Screen_size, Resolution, Refresh_rate, Type, Audio, Hdmi, Displayport, Dvi, Color)

FD: Monitor_ID→Screen_size, Resolution, Refresh_rate, Type, Audio, Hdmi, Displayport, Dvi, Color

BCNF

Keyboard(Keyboard_ID, Color, Backlight_color, Numpad, Wireless)

FD: Keyboard_ID→Color, Backlight_color, Numpad, Wireless

BCNF

Phone(Phone_ID, Resolution, Screen_type, IP_rating, Storage, Ram, CPU, OS, Carrier, 5G, Battery, Size)

FD: Phone_ID→Resolution, Screen_type, IP_rating, Storage, Ram, CPU, OS, Carrier, 5G, Battery, Size

BCNF

Identifier(Warehouse_ID, URL, Item_ID)

No functional dependencies.

BCNF

Normalization Justification

Due to the nature of our database, decomposition was straightforward. The first hurdle was determining what would and would not be considered entities, but since the database focuses primarily on finding the best options for computer components, it was simple to split up entities into each of the invidual tech components. The entities were then converted into their respective schemas via referencing the ER diagram and adding the attributes listed from each entity. The super keys and foreign keys were already listed on the ER diagram, so it was just a matter of underlining the already defined attribute keys when translating them to the schemas. The relationships were a bit more complex to define, as there is only one major relationship holding the ERD together, which is "contains."

As this database is used for finding the best price for technology along with other qualities, the only real relationship is to check what items a warehouse contains. A warehouse containing a specific item means we are capable of checking the qualities of each item, alongside whatever data the warehouse tracks about those items. For instance, if I want a 16 GB RAM stick for under 60 dollars, the warehouse's data will be check the RAM category, then if there are any 16 GB RAM in stock, and finally determine if there are any in the warehouse for under 60 dollars. This leads to contains being a one-to-many relationship, as warehouses can have many different items, alongside products that are available in other warehouses. However, each instance of a product can only exist one warehouse at any given momet. Furthermore, all of the traits for each item can be narrowed down by referring to the "item" schema and then whatever particular component the database is checking for. All entities branching off of "item" will have the attributes of item name, category, manufacturer, series, and so forth, but not all entities will have the same qualities to check for.

Because of these very streamlined and simplistic relations, BCNF came naturally. The functional dependencies start off relatively close to already being in BCNF, as with a warehouse ID and a location, checking the inventory of the warehouse and then pinpointing what component you are looking for is just a matter of knowing the item_ID. The item ID will functionally determine all of the specs of a component, as the item IDs are unique. Therefore, if an item ID 425 is in warehouse 500 from the company Amazon, then there is only one instance of that item throughout the entire database. From these three attributes alone, the rest of the attributes can be functionally determined, with no overlap from the other entities. This lack of overlapping from other entities makes it very easy to determine the normal form the database design without much decomposition.