MEMORANDUM

To: Wade Fergus DBM/380

From: Nicholas Evans

Date: November 9, 2015

Subject: Restaurant POS database design process

Database design, like any creative process, if facilitated and improved by following a process. The first step in designing a database to support a Restaurant POS was to perform a requirements analysis. Outlining the necessary functions provided initial direction. Functions were addressed in last week’s memo but here is a more concise list.

* Produce checks, invoices for the items purchased.
* Produce orders, communications of required items to production areas.
* As there are multiple production areas, orders must be sent to the appropriate area.
* Track Items with limited inventory
* Allow users to add modifiers to ordered items.
* Employee’s total sales and credit card tips need to be calculated for a given shift.
* System serves as employee time clock.

I took each function and sketched a Chen’s model style diagram by hand. Drawing connections, adding attributes and scratching out others as I worked towards workable models for each function.

* With the creation of checks and orders being so closely related I grouped those together. My approach uses of a LINE entity which contains foreign keys to CHECK, ORDER, and PRODUCT. The LINE entity can be queried for CHECK\_ID to produce all the items order on a single check. It can also be queried by ORDER\_ID when an order is sent. The PRODUCT entity instance contains an assigned PROD\_ID (PK), a plain English description, a display name for the POS, and a price. The CHECK entity contains the EMP\_ID who made the check, the total/tip amount, and when it was opened/closed. These attributes also allow tracking a user’s sales and tips. The ORDER entity only needed the EMP\_ID and the time it was placed.
* Sending items to multiple production areas needed a PRINTER\_Code attribute to be added to the PRODUCT entity. This foreign key connects to a PRINTER entity containing a printer name and network location for each PRINTER\_CODE.
* Tracking limited inventory just needed a PROD\_INSTOCK attribute added to the PRODUCT entity. This value is set and managed by the application with a N/A value used for items without a limited stock.
* Allowing modifiers to be added to items stumped me a bit. The solution I came up with uses a PRODUCTMOD and LINEMOD entity to store mandatory modifications on specific products and custom modifiers made by users respectively. Each instance has either the PROD\_ID or LINE\_ID the modifier is associated with and a MOD\_CODE. This code is a foreign key to a MODLIBRARY entity which attaches specific modifiers to MOD\_CODES. If all modifications were to be custom then a MODLIBRARY would be unnecessary but users need the speed of predefined modifiers in their operation of the system.
* A SHIFT entity was created to enable an employee timeclock. Each shift is given an SHIFT\_ID (the primary key) and is attached to an EMP\_ID. It also records the clock in and clock out time/date. Time and date are needed in case a shift ends past midnight.

Once I had completed the individual function models I began adding the entities, attributes, and relationships into a Crow’s Foot diagram. The majority of primary keys are arbitrarily assigned ID numbers, so much so that I’ll be looking closely at some entities to look for better options before finalizing the design. As I mapped the relationships I also determined cardinality which is indicated by the connections in the diagram. The diagram is on the following page.

