Unit 8

Hands-on with Database Design (Reflection)

The focus in the week was on an introduction to developing entity-relationship diagrams (ERD) that help drive the structure of a (relational) database. As part of the consideration of developing database structures, there was an engagement in a collaborative discussion to consider alternates to traditional relational SQL structures. The main alternative considered is NoSQL databases.

Graph databases are interesting NoSQL databases, and I have had the fortune to work with them for several years. The most significant difference between graph databases and relational databases is the idea that relationships are first-class citizens. This concept is essential in graph databases. In relational databases, we focus solely on the data sets (the schema) and then consider *how* to traverse each group using relevant joins. However, with graph databases, we think in terms of *relations*. Graph databases, therefore, allow us to model our data using Subject-Verb-Object sentence structures. Not an easy task with relational databases because relational databases do not have a matching paradigm for a Verb.

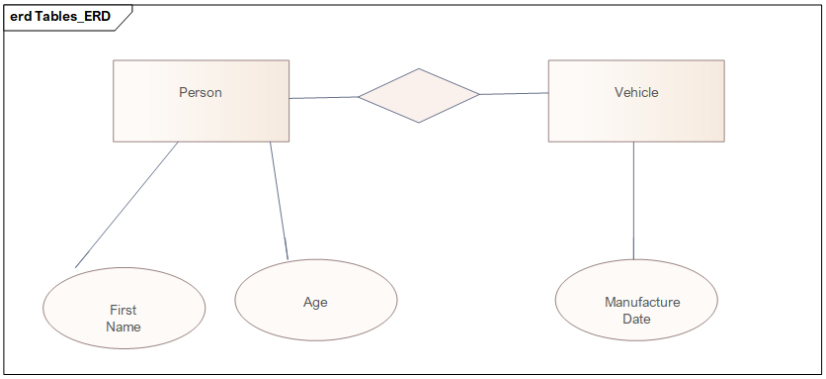
Two database concepts, ACID and BASE, refer to models for how relational (ACID) and NoSQL (BASE) databases provide consistency and reliability.

Modelling ERD

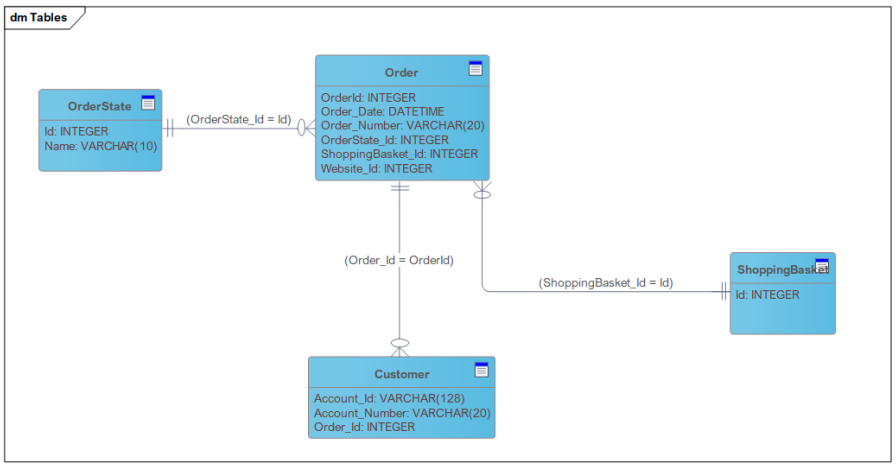
Modelling *entities* concerns itself with three topics, namely:

- ***Entities.*** Items about which information is collected and stored. These usually represent real-world concepts, tangible elements or even intangible ideas or concepts. The entities always has one or more attributes connected to them.
- ***Attributes.*** These are the properties linked with entities and provide descriptive information about them. Attributes can be considered either as *descriptors* or *identifiers* (the terms “primary key” or “foreign key” are used).
- ***Relationships.*** Structures that link two or more entities together. While relational databases do not manifest relationships as physical entities, NoSQL Graph databases do.

While there is no standard notation for an ER model, Peter Chen developed the original notation of entity-relationship diagrams as shown below:



However, I have hardly ever encountered this notation in my working career. While it is suitable to represent the *nuances* of data, it is very clunky and leads to bloated diagrams. Instead, diagrams similar to below are plentiful and make use of the familiar “crow’s feet” connectors to represent the cardinality of relations between two entities, with simple boxes representing entities:



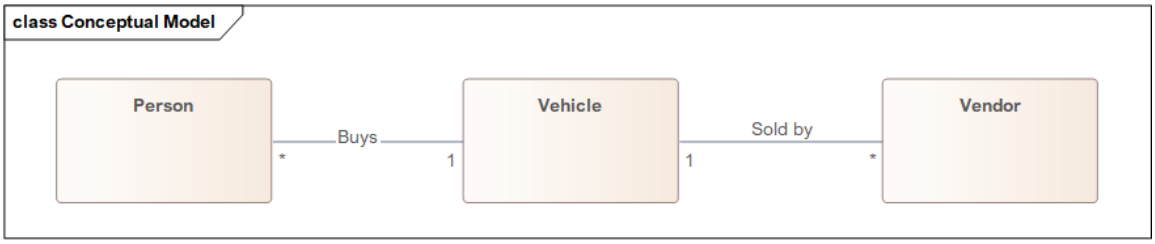
I prefer using crow’s feet notation because the attributes of each entity table are grouped into a single visual block. Whereas in Chen’s notation, each attribute becomes a single bubble. Such explosion of attribute bubbles will, in my opinion, quickly create database models that are unwieldy and hard to read! I think back to unit 7, where students were challenged to practice their normalization skills and observed that all submitted diagrams did not utilize Chen’s notation. I believe this is due in part to two reasons: path-of-least resistance and prior exposure to crow’s feet notation.

Moving from a UML class diagram to an ERD is relatively straightforward, except in a specific scenario where a class **composes** another class. Modelling a database for the UML model in unit7 challenged me to consider why I often use many-to-many tables. The standard approach

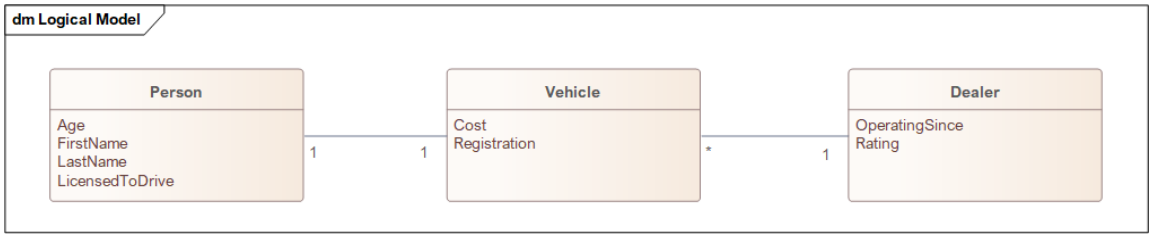
uses the parent table's primary key in the child table as a foreign key; however, this is not always the case as experienced in this unit. *Please see the artefact for Unit 8's DB design.*

I find interesting the close correlation between UML class diagrams and an ultimate database model. This is because both diagram types have *entities, attributes* and *relationships.* It then follows that it is possible to provide a UML class diagram to skilled DB admins. They will turn the UML models into valid database structures—the mapping between both diagram types is straightforward. Along the lines of this analysis, database diagrams *evolve* from conceptual to logical and then to physical; each evolution adds detail to the model. For example:

- ***Conceptual Design.***
  At this stage, before a full ERD, we merely link ideas or concepts together.

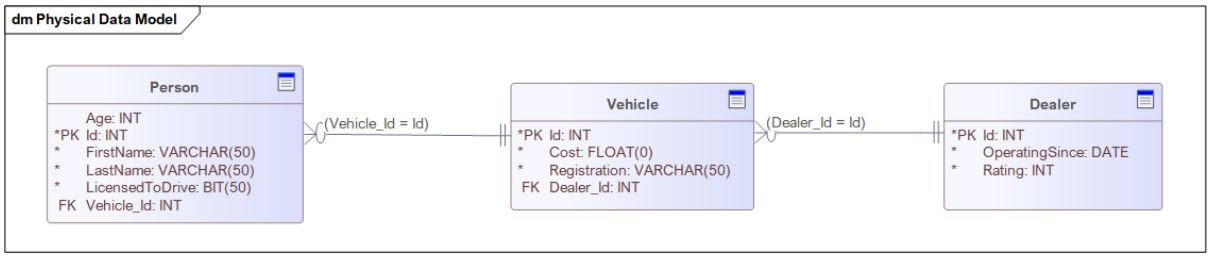


- ***Logical Design.***
  At this stage of the design, we add additional details to our concepts. We are not yet at a full-fledged ERD diagram, but this stage sets the stage for how our data model will look.



  Note: I consider that Chen's notation is mainly similar to this stage. However, the notation does have the concept of multivalued attributes and weak entities.

- ***Physical Design.***
  At this stage, we fill out our logical design with the actual data types required to honour the UML class diagram.



This unit also gave a task to practice database normalization. Database normalization is a valuable part of designing databases and ensures data duplication is minimized. However, from researching NoSQL databases as part of the collaborative discussion, NoSQL databases are growing in popularity primarily because businesses need to handle large volumes of data far quicker than relational databases allow.