Kathy He
Professor Heller
CSCI 331
4/28/24

As described in the video, User-Defined Datatypes provide a layer of abstraction to define a database datatype when modeling a database. UDTs can be classified hierarchically in a system called a UDT Taxonomy, where UDTs are organized based on their properties/characteristics within a domain parent. UDTs are given fully-qualified names consisting of a physical name and a domain parent, which can be represented by a domain or subdomain or database schema name.

The concept of a UDT Metadata Taxonomy comes from Object-Oriented Programming and SOLID principles. Since databases can be viewed as containers of objects, the video points out that it makes sense to apply OOP principles to UDTs. UDT Metadata Taxonomies follow the SOLID principles by having each UDT physical name datatype have a Single Responsibility (e.g. the UDT Physical Name Address clearly is only responsible for keeping track of one type of thing: addresses), swapping out dependency on concrete implementations for abstractions instead (e.g. defining EmployeeFirstName as a UDT FirstName rather than the built-in concrete datatype nvarchar(10)), and following the Liskov substitution principle by having UDT Physical Name datatypes function separate but still having a connection to domain parent categorizations.

This leads to the next point, because domain parent categorizations allow database designs to expand use of already-existing UDT Physical Name datatypes to other entities, while still producing unique fully-qualified domain names and grouping those objects under those entities, which adheres to the Open-Closed principle, allowing for further modification of the design while keeping the existing structures stable. By adhering to the SOLID principles, incorporating UDT Metadata Taxonomies into a database design promotes flexibility, understandability and maintainability.

There are a couple of benefits to this. One is reusability. Businesses and organizations will likely have use for a large volume of attributes in their databases and some of these attributes might share the same abstract datatypes, like FirstName, LastName, Telephone, etc. By defining a UDT Physical Name datatype once, it can be reused multiple times to define these attributes in turn, saving a lot of time. Additionally, the benefit of this design is that UDTs can encapsulate or be loosely coupled together with Domain Constraints. This allows for higher quality of data being entered into the system at the application level, since the check and default constraints will ensure that incoming data will be more uniform, and therefore be usable by organizations.

The video also provides a summary of a currently-existing standard for gathering and defining domain objects as described in Harris and Hoberman's book *Data Modeling Made Simple With Erwin Data Modeler*. The steps outlined in the video begin by examining the data for patterns across entities or processes, which are then documented in a spreadsheet by domain name, description, and related attributes/entities. Before proceeding to create a logical data model, input from stakeholders is used to further refine the domain objects and their definitions. This is an important step that repeats when we move from the logical data model to the physical. What can be seen here is that in this already-existing standard, the

use of Fully-Qualified User-Defined Datatype Taxonomies in developing these domain objects slots in very easily in the use of a spreadsheet to draft ideas for domain objects. In the example that contrasts the way a spreadsheet is set up for the already-existing standard versus the way it is set up to define a fully-qualified domain name for a UDT, the use of a Domain Parent can provide more context to the UDT that stakeholders can understand more easily.