Research on Fully Qualified Domains, Fully Qualified Table Names, and Taxonomies

By Giorgos Stefanis, Hasan Husseini, Ridhwan Ibrahim, Michael Cipriani, Seeram Govindan, Youlun Wang

The Need For Data Governance

- Companies often face inconsistent data naming across systems
- Different teams use conflicting table names and data types
- Results in errors, confusion and unreliable reporting
- Ensures all data is consistent accurate and secure. And easy to manage.

How FQDS, FQTNS, and Taxonomies solve it

- FQDS: Keeps data definitions consistent across systems
- FQTNs: Standardize table name
- Make Data easy to find – Ex employee's name or email
- Taxonomies: Organize FQDs & FQTNS in structured hierarchies

Together they deliver order, reliability and data consistency.

What Are FQDs

- FQD stands for **Fully Qualified Domain**
- Fully Qualified Domains are used to address the complete name of an object with all its parts.
- Are completely specified domains with a name, data type, and restrictions that can be easily referenced in SQL statements.
- Mainly used to ensure data reusability and uniformity between tables.

Example:

AdventureWorks2019.HumanResources.Employee.BusinessEntityID

- **AdventureWorks2019** is the database
- **HumanResources** is the schema
- **Employee** is the table
- **BusinessEntityID** is the column

By using it, we know precisely the item we are referring to

Advantages and Disadvantages:

Advantages:

- Clear representation on which server, database, schema you are referring to.
- Allows different databases from other servers to be joined together.
- Able to resolve any problems with schema or database faster since you already know where they are referring.

Disadvantages:

- When a schema or database change, all the queries must be updated.
- Databases and schemas are tied together, making it harder to move to a different environment.

CREATE DOMAIN (ANSI SQL)

- CREATE DOMAIN is an ANSI SQL statement.
- Used to create a user-defined data type on an existing data type.
- It has optional restrictions like NOT NULL, NULL, DEFAULT.
- Easier to understand what Is the data type instead of VARCHAR(50).
- Part of the SQL standard.

Example:

```
CREATE DOMAIN HomeAddress as VARCHAR(50)
CREATE TABLE Persons(
    ...
    ...
    HomeAddress HomeAdrress
)
```

This example allows you to create HomeAddress data type in the schema, and it can be used later when writing a query.

CREATE TYPE

- Defines a user-defined scalar or table type.
- It is used to create a data type in an existing(current) database.
- Fewer constraints than CREATE DOMAIN. CREATE TYPE has NULL as a default but, also, supports NOT NULL.
- SQL server specific (T-SQL).

Example:

```
CREATE TYPE HomeAddress FROM VARCHAR(50) NOT NULL
CREATE TABLE Persons(
   ...
   ...
   HomeAddress(column name) HomeAdrress(type name)
);
```

This example allows you to create HomeAddress reusable used-defined data type in the schema, and it can be used later when writing a query.

FQDs – Practical Benefits

- Promotes data consistency across multiple databases and schemas
- Allow reusable, standardized data types instead of redefining fields in each individual table
- It simplifies updates, because if you change once in the domain, it applies everywhere
- Reduces user confusion by enforcing uniform naming and constraints
- Improve governance and auditing since all data types are centrally defined

FDQs in Use

- Example ensuring address format consistency across all tables:

```
CREATE DOMAIN HomeAddress AS VARCHAR(100) NOT NULL;
CREATE TABLE Customer (
   CustID INT PRIMARY KEY,
   Address HomeAddress
);
CREATE TABLE Employee (
   EmpID INT PRIMARY KEY,
   Address HomeAddress
);
```

- Both tables reuse the HomeAddress domain, so consistent type and rule
- Simplifies integration across departments such as Sales, HR, etc.
- Reduces redundancy and enforces organization wide standards

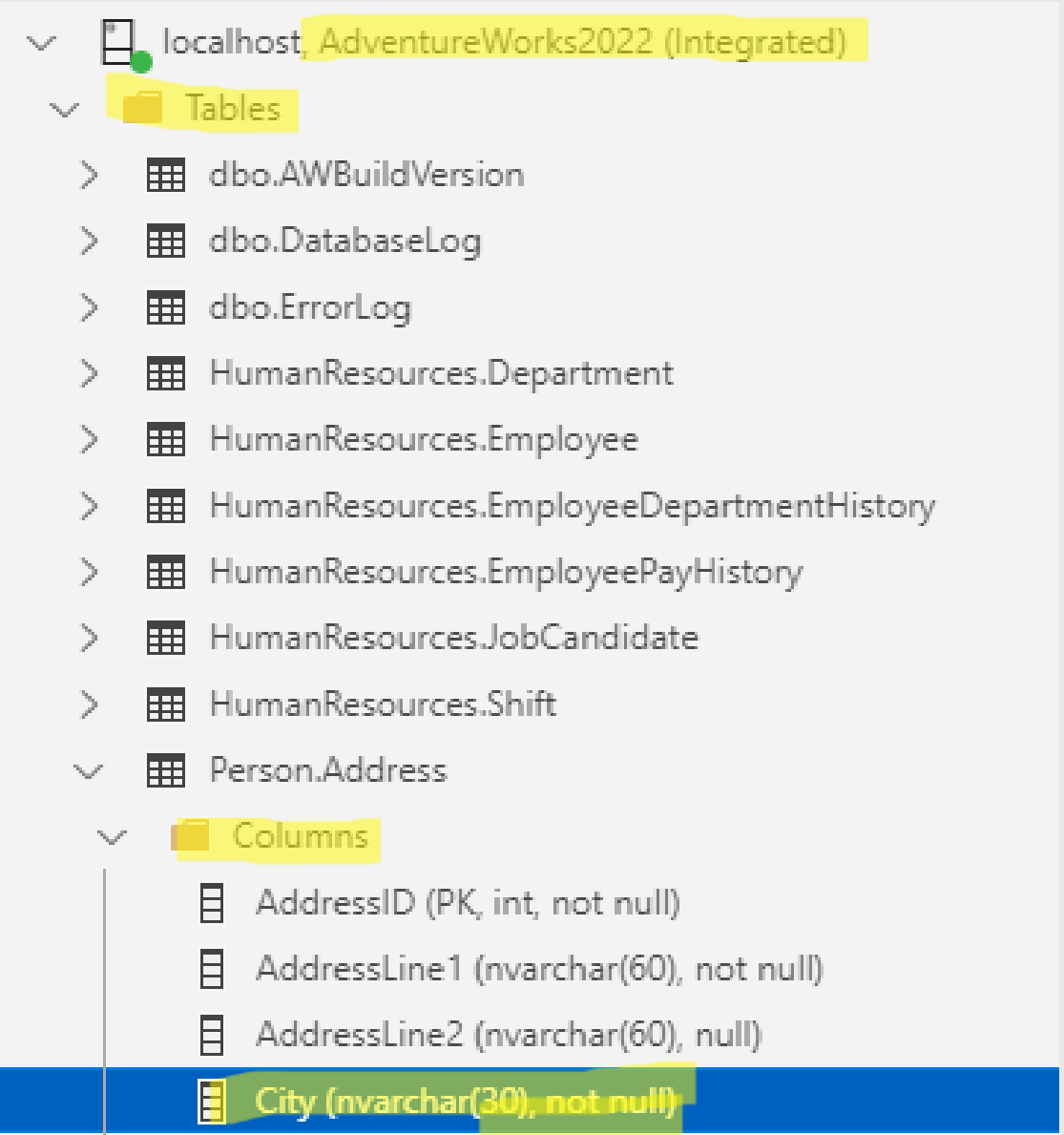
FQDs and Data Governance Impact

- Having centralized definitions makes data more reliable and easier to trace when an issue comes up
- Makes database maintenance and schema updates a lot simpler over time
- Helps teams collaborate better by sharing the same data definitions
- Build foundation for Taxonomies and FQTNs

Taxonomy in Data Governance - Definition

- Taxonomies are hierarchical naming systems used for categorization of objects with similarities or shared traits.
- This is used in data governance to map different forms of data, like tables, schema, datatypes and columns, into structured categories.

Taxonomy in Data Governance – Example



This is an example of a taxonomical hierarchy, using the AdventureWorks2022 database. Highlighted in yellow are the representations of **[Database]->[Table]->[Column](->[Row])**
Note: Rows are not displayed but are implied.
Each highlighted element describes a layer of the hierarchy, where rows are instances of columns, columns belong to tables, and there exist some tables within a database. This type of taxonomy should be familiar to anyone working in databases

Taxonomy in Data Governance – Application & Significance

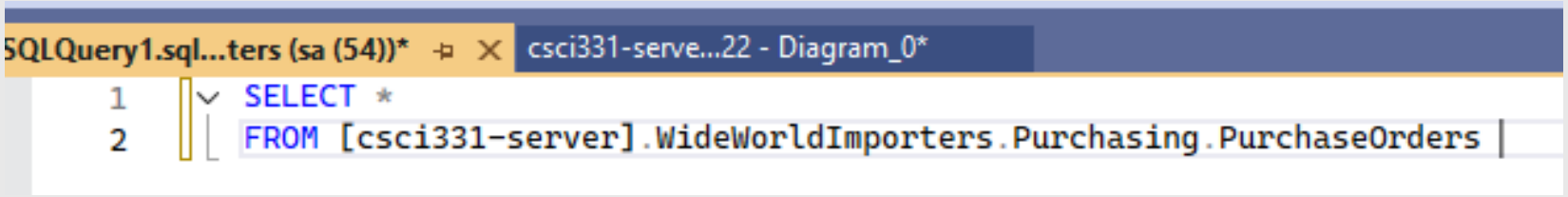
- Taxonomical systems ensure consistency for everyone in an organization throughout an entire database.
- Promotes data discoverability, reducing user search and interpretation times
- Defines structure and accountability, ensuring data governance.
- Builds a foundation to implement the aforementioned FQDs and FQTNs into the structure, as they exist within the framework.
- Avoids the event in which data is duplicated.

Taxonomy in Data Governance – Sources

- RudderStack "What is Data Taxonomy? Organizing Your Information Chaos." https://rudderstack.com/learn/data/what-is-data-taxonomy/
- Amplitude "What is Data Taxonomy?"
- Dovetail "What are Taxonomies?" https://dovetail.com/research/what-are-taxonomies/
- Microsoft AdventureWorks Sample Database
- https://learn.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver17&tabs=ssms

What are FQTN's?

- FQTN stands for **Fully Qualified Table Name**
- It defines the exact table being referred to in a Database
  - `[ServerName].[DatabaseName].[SchemaName].[TableName]`
- It can be thought of as an address to where the table lives on the DB.
- Ensures every table reference is unique



Clarity and Referential Integrity

- Using FQTNs make queries easier to read and understand
- Different schemas or databases can have the same table names
  - Example: HR.Employees, IT.Employees, Finance.Employees
- FQTNs specify exactly which table we're referring to
- Preserves referential integrity by specifying the exact table when using a JOIN especially when identical tables exist in multiple schemas or databases

Scalability across Heterogenous environments

- Large organizations rarely use one form of a database
- At the enterprise level an organization can have multiple DB platforms, different DB servers for specific teams, or integrate cloud-based DB services.
- FQTNs become necessary when creating cross-database queries
- Improves searchability across all databases because you know where the table resides
- Using FQTNs as a standard allows for organizations to scale safely across multiple platforms because all data is fully qualified and explicitly referenced

Integration of FQDs,FQTNs, and Taxonomy

- The triage of FQDs, FQTNs, and Taxonomy go hand in hand. With each defining a stage or level of long term scalable systems for the business to keep organized with their database needs.
- The FQD ensures exactly how each areas/business terms are defined in a SQL database context.
- FQTN ensures the location is consistent across systems
- Taxonomy ensures the business logic/meaning is applied to the database model and organized in a consistent/scalable manner.

Real-World Impact

- Long term growth and scalability of the business.
- In the real world, especially large organizations with complex business needs, allows businesses to grow while maintaining strong data governance and accountability
- Faster data discovery and reuse
- Better alignment between IT and business teams, where everyone speaks the same language about the data.
- Providing a strong framework connecting technical database (tables, domains) objects to business concepts (customer, product, order)