**SQLSerializer Documentation**

**Overview**

SQLSerializer is a Kotlin utility designed to facilitate the serialization of data into SQL INSERT statements, suitable for various SQL databases, particularly MySQL. This utility efficiently converts Java/Kotlin objects to SQL-compatible string formats, handling a wide range of data types with specific considerations for length, precision, and format.

**Features**

* **Support for Multiple Data Types**: Handles standard SQL data types including string, numeric, date/time, binary, and text.
* **Customizable Formatting**: Offers precision handling for numeric types and supports custom lengths for string and binary types.
* **Date and Time Support**: Formats date and time values according to SQL standards.
* **Binary and Blob Data Handling**: Converts binary data into hexadecimal format for direct SQL usage.
* **Type Safety**: Ensures values are appropriately cast to the expected data type.

**Usage**

**Initialization**

First, initialize the SQLSerializer with the target table name and a map defining the data types of each column:

kotlin

val columnDataTypes: Map<String, SQLDataType> = mapOf(

"columnName1" to SQLStringType(255),

"columnName2" to SQLDecimalType(10, 2),

// ... other column types

)

val serializer = SQLSerializer("your\_table\_name", columnDataTypes)

**Serialization**

To serialize your data, provide a graphState (representing the current state of the data to be serialized) and a schemaEntities object (defining the schema of the data):

kotlin

val serializedSQL = serializer.serialize(graphState, schemaEntities)

**Handling Output**

The serialize method returns a string containing the INSERT INTO SQL statements. This output can be written to a file, logged, or executed directly against a database, depending on your application's needs.

**Data Types Handling**

SQLSerializer uses a set of data classes to represent different SQL data types. Each data type class is tailored to handle specific aspects of the corresponding SQL type:

* SQLStringType(length: Int): For VARCHAR and similar types, with specified length.
* SQLDecimalType(precision: Int, scale: Int): For DECIMAL types, defining precision and scale.
* SQLFloatType(precision: Int, scale: Int): For FLOAT and DOUBLE types.
* SQLBinaryType(length: Int): For binary data with a specified length.
* SQLDateType, SQLTimeType(fsp: Int), etc.: For various date and time types.

**Error Handling**

SQLSerializer includes robust error handling for type mismatches, data validation, and formatting issues. Ensure to handle any exceptions or errors based on your application’s error-handling strategy.

**Extensibility**

SQLSerializer is designed to be extensible. Additional data type classes can be added to handle more SQL types or custom requirements.