Software Engineering + Data Wrangling with SQL

Combined Assessment

Scope statement:
In the Data Wrangling with SQL course, we have seen how we could write stored procedure/functions to build dynamic SQL pivot survey answers data in usable format for analysis in the toy database "`SurveySample_A19`".
After a few iterations, we ended up with the following design:
1. A stored function `dbo.fn_GetAllSurveyDataSQL()` which generates and returns a dynamic SQL query string for extracting the pivoted survey answer data.

2. A trigger `dbo.trg_refreshSurveyView` a. firing on `INSERT, DELETE and UPDATE` upon the table `dbo.SurveyStructure`

b. executing a `CREATE OR ALTER VIEW vw_AllSurveyData AS` + the string returned by `dbo.fn_GetAllSurveyDataSQL`

With this design, we have enforced an *"always fresh"* data policy in the view `vw_AllSurveyData`.
As discussed, this solution is "ideal" as it respects the principle of data locality. But it requires to have privileges for creating stored procedures/functions and triggers. If the former may be rare, the latter is often heavily restricted.
You are now in a scenario where **the only databases operations allowed** are:
1. to select data from tables.
2. to create/alter views.

You can use programmatic access to the database server via an ODBC library and you have to develop in Python 3.
Your Python 3 application must accommodate the following requirements:
1. Gracefully handle the connection to the database server.

2. Replicate the algorithm of the `dbo.fn_GetAllSurveyDataSQL` stored function.

3. Replicate the algorithm of the trigger `dbo.trg_refreshSurveyView` for creating/altering the view `vw_AllSurveyData` whenever applicable.

4. For achieving (3) above, a persistence component *(in any format you like: CSV, XML, JSON, etc.)*, storing the last known surveys' structures should be in place. It is not acceptable to just recreate the view every time: the trigger behaviour must be replicated.

5. Of course, extract the *"always-fresh"* pivoted survey data, in a CSV file, adequately named.