Datasets QuestDB Example

Read, downsample, and write QuestDB time-series data from a Dataset.

How-to **Examples** **Feature** **Application** DatasetDB

This example shows how to read, downsample, and write to a QuestDB table using the existing PostgreSQL Dataset Provider. QuestDB speaks the PostgreSQL wire protocol natively, so no separate QuestDB-branded Dataset Provider is required in 10.1.5.

Prerequisites:

- A running QuestDB 7.0+ on `localhost:8812` (PostgreSQL wire) and `localhost:9000` (ILP /HTTP and Web Console).
- A database named `qdb` with a `sensors` table. The QuestDB Time-Series Database Connector reference page covers the install walkthrough and the seed schema.
- Default credentials on a fresh local install are `admin` / `quest`.

Summary

This example walks through three time-series operations against a QuestDB table from a FrameworX solution: a basic latest-rows read, a server-side downsampling query using QuestDB's `SAMPLE BY` stage, and a row insert through a Dataset Table. The work is done entirely from FrameworX, using the existing PostgreSQL Dataset Provider with one mandatory connection-string flag.

For other surfaces of the QuestDB connector (Device Protocol, UNS TagProvider, Historian StorageLocation), see the QuestDB Time-Series Database Connector reference page. For high-volume time-series appends at runtime, the Historian StorageLocation surface using ILP/HTTP is the recommended write path; this example uses a Dataset Table insert for demonstration parity with the MongoDB Datasets example.

Technical Information

The example creates one Dataset DB connection, three Queries, one Table, three Script Tasks, and a small set of UNS Tags. Each piece is configured in the Designer.

Configure the Dataset DB

In **Datasets / DBs**, create a connection:

- Name: `questdb_local`
- Provider: `PostgreSQL`
- Connection String:

```
Server=localhost;Port=8812;Database=qdb;Username=admin;Password=quest;Server Compatibility Mode=NoTypeLoading;
```

The `Server Compatibility Mode=NoTypeLoading` flag is required. Without it, the Npgsql client tries to load PostgreSQL system catalog tables that QuestDB does not expose, and the connection fails. Click **Test** to verify the connection works against the running QuestDB server.

Configure the Queries

In **Datasets / Queries**, create three queries against the `questdb_local` connection. All three are native QuestDB SQL.

QueryLatestReadings. Returns the latest 10 readings for Plant01.

```
SELECT timestamp, plant, temperature, quality
FROM sensors
WHERE plant = 'Plant01'
ORDER BY timestamp DESC
LIMIT 10
```

QuerySampleByHourly. Server-side hourly downsampling of temperature readings for Plant01, using QuestDB's signature `SAMPLE BY` stage. The aggregation runs inside QuestDB - the FrameworX runtime receives one row per hour bucket, not raw samples.

```
SELECT timestamp, avg(temperature) avg_temp, min(temperature) min_temp, max(temperature) max_temp
FROM sensors
WHERE plant = 'Plant01'
SAMPLE BY 1h FILL(NULL)
ORDER BY timestamp DESC
LIMIT 24
```

QueryCountGoodQuality. Counts samples whose `quality` field is Good (FrameworX quality code 192).

```
SELECT count() AS good_count
FROM sensors
WHERE quality = 192
```

Configure the Dataset Table

In **Datasets / Tables**, create one table named `TableSensors` bound to the `sensors` table on the `questdb_local` connection. The Dataset Table primary key is the designated timestamp column. `AddRow()` + `Save()` dispatches an `INSERT` over the PostgreSQL wire to QuestDB. For high-volume telemetry appends at runtime, configure the Historian StorageLocation surface instead - it routes through ILP/HTTP and is optimized for batched append.

Configure the UNS Tags

In **Unified Namespace / Tags**, create the tags used by the tasks:

- `QueryResult`. Receives the output of the Latest and SampleBy queries.
- `GoodSampleCount`. Receives the result of the Count query.
- `TriggerLatest`, `TriggerSampleBy`, `TriggerInsert`. Script task triggers.
- `LatestPlantCode`, `LatestTemperature`. Values written back to QuestDB by the Insert task.

Configure the Script Tasks

In **Scripts / Tasks**, create three tasks, each triggered by the matching trigger tag.

Latest task.

```
@Tag.QueryResult = @Dataset.Query.QueryLatestReadings.SelectCommand();
@Info.Trace("Latest OK: " + @Tag.QueryResult);
```

SampleBy task.

```
@Tag.QueryResult = @Dataset.Query.QuerySampleByHourly.SelectCommand();
@Info.Trace("SampleBy OK: " + @Tag.QueryResult);
```

Insert task (via the Dataset Table).

```
@Dataset.Table.TableSensors.AddRow();
@Dataset.Table.TableSensors.Row["plant"] = @Tag.LatestPlantCode;
@Dataset.Table.TableSensors.Row["temperature"] = @Tag.LatestTemperature;
@Dataset.Table.TableSensors.Row["quality"] = (short)192;
@Dataset.Table.TableSensors.Row["timestamp"] = DateTime.UtcNow;
int i = @Dataset.Table.TableSensors.Save();
@Info.Trace("Insert OK: " + i);
```

Run the example

After you finish the configuration and create the scripts, run the solution and trigger each task. Values arrive in the QuestDB **sensors** table and the Latest and SampleBy results populate the `QueryResult` tag. The Count query populates `GoodSampleCount`. Trace output appears in the runtime log.

QuestDB time-series extensions

The PostgreSQL Dataset Provider passes native QuestDB SQL through unchanged. Beyond `SAMPLE BY` shown above, QuestDB exposes several time-series-specific features that work directly from a Dataset Query:

- **`LATEST ON`** - returns the most recent row per group. Example: `SELECT * FROM sensors LATEST ON timestamp PARTITION BY plant;` returns the latest reading for each plant.
- **`ASOF JOIN`** - time-series alignment with optional `TOLERANCE`. Joins two tables on the closest preceding timestamp.
- **Materialized views** - persisted aggregations refreshed on a calendar or interval. Read them from a Dataset Query the same way you read a regular table.
- **Array types** - QuestDB supports N-dimensional arrays for AI and financial workloads. Returned as native arrays through the PostgreSQL wire protocol.

Caveats

- Cursor support is forward-only. Scrollable cursors (`DECLARE CURSOR ... SCROLL`) are not available.
- QuestDB transaction semantics differ from standard PostgreSQL. Single-statement implicit transactions are the default.
- PostgreSQL extensions (PostGIS, pg_stat_statements, etc.) are not available - QuestDB exposes only the wire protocol, not the full PostgreSQL ecosystem.
- For high-volume time-series appends at runtime, prefer the Historian StorageLocation surface (ILP/HTTP) over Dataset Table inserts (PostgreSQL wire). See the QuestDB Time-Series Database Connector reference page for the Historian StorageLocation configuration.

In this section...
