Critical manufacturing
an ASM PT company

Database Integration Tutorial

12.0.1

November 2026

DOCUMENT ACCESS

Public

DISCLAIMER

The contents of this document are under copyright of Critical Manufacturing S.A. it is released on condition that it shall not be copied in whole, in part or otherwise reproduced (whether by photographic, or any other method) and the contents therefore shall not be divulged to any person other than that of the addressee (save to other authorized offices of his organization having need to know such contents, for the purpose for which disclosure is made) without prior written consent of submitting company.

12.0

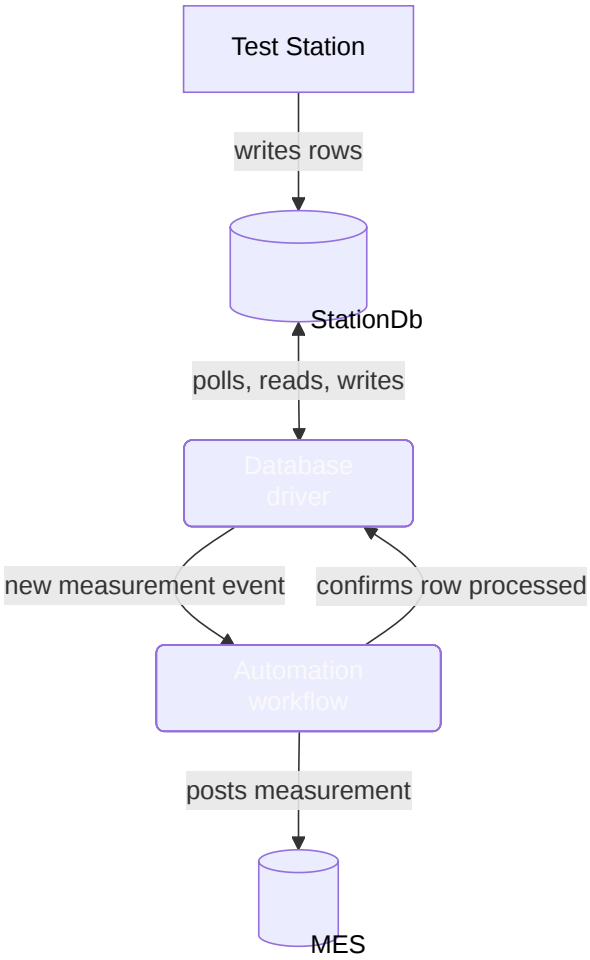
Copy Page

Database Integration Tutorial

Estimated time to read: 13 minutes

This tutorial integrates a legacy test station with MES using the Connect IoT **Database driver**. Many stations on a shop floor never speak an equipment protocol. They write their results into a relational database, and that table is the only integration point available.

The Database driver turns such a table into an equipment interface. It polls the table for you, builds the SQL from what you configure on the **Automation Driver Definition**, and raises an Automation Event for each row it finds. It also runs commands, so a workflow can read reference data or write back into the database.



The test station never talks to MES directly. The driver watches the table on its behalf, and the workflow is what decides what a new row means and where it goes.

By the end you will have an Automation Controller that:

- Detects each new measurement row written by the test station.
- Posts the measurement to MES, one row at a time, without losing rows or reprocessing them.
- Writes the processing result back into the station database.
- Reads the recipe limits for a material on demand.

Database Overview

Why a database integration

A database integration fits when the equipment, or the software that drives it, already persists what you need:

- Legacy testers and inspection stations that log results to a local SQL Server or MySQL instance.
- Third-party line software with a documented database schema but no API.
- Intermediate staging tables written by a supplier gateway.

You do not change the station. You read what it already writes.

Key concepts

- Trigger query - the query the driver runs on every polling cycle to detect that something changed. It is built from the event configuration, or supplied as a raw query.
- Result query - the query the driver runs after the trigger detects a change, to fetch the columns the event carries.
- Processing mode - decides whether the driver keeps polling immediately after raising an event, or waits for the workflow to confirm that the row was handled.
- Command - a query the workflow runs on demand: a select, an insert, an update, a delete, a stored procedure, or a raw statement.

> **Note**
>
> The driver polls. It does not receive notifications from the database, so the polling interval sets how quickly a new row is noticed, and how much load the driver puts on the database server.

Scenario

Test station **TEST-01** writes one row per measured unit into the table `dbo.TestResults` of the `StationDb` database:

Column	Type	Meaning
`Id`	`INT`	Identity, incremented per row

Column	Type	Meaning
`MaterialName`	`NVARCHAR(50)`	The unit measured
`RecipeName`	`NVARCHAR(50)`	The recipe used
`Value`	`DECIMAL(18,3)`	The measured value
`MeasuredAt`	`DATETIME`	When the measurement was taken
`ProcessedStatus`	`NVARCHAR(20)`	Empty for a new row. The integration writes `Processed` or `Failed`

Table: Structure of the TestResults table

A second table, `dbo.RecipeLimits`, holds the acceptance limits per recipe:

Column	Type	Meaning
`RecipeName`	`NVARCHAR(50)`	The recipe
`LowerLimit`	`DECIMAL(18,3)`	Lowest accepted value
`UpperLimit`	`DECIMAL(18,3)`	Highest accepted value

Table: Structure of the RecipeLimits table

The integration goals are:

- Flow A - detect each new row of `TestResults`, post the measurement to MES, and mark the row as processed.
- Flow B - read the limits of a recipe from `RecipeLimits` on demand, so the workflow can classify the measurement before posting it.

Prerequisites

- A running Automation Manager and access to the GUI. If you are new to controllers and protocols, complete the Connect IoT - Basic Configuration Tutorial

first.

- A reachable Microsoft SQL Server or MySQL server, and a database account that can read the two tables and update `TestResults`.
- The two tables created, with a few rows in `RecipeLimits`.
- A **Resource** in MES named `TEST-01`, marked as Connect IoT enabled. See How To: Mark an Entity Type as Connect IoT Enabled.

Demo environment (containers)

For a self-contained trial, the following `docker-compose.yml` starts a single Microsoft SQL Server instance. The Automation Manager connects to it on `localhost:1435`. Host port `1435` is used instead of the default `1433` so the demo server does not clash with an existing SQL Server already using `1433`.

```yaml
services:
  sqlserver-station:
    image: mcr.microsoft.com/mssql/server:2022-latest
    container_name: sqlserver-station
    ports:
      - "1435:1433"                     # host 1435 -> container 1433
    environment:
      ACCEPT_EULA: "Y"
      MSSQL_SA_PASSWORD: "Station#Passw0rd"
      MSSQL_PID: Developer
```

Start it with `docker compose up -d`, then create the database and the tables:

```sql
CREATE DATABASE StationDb;
GO
USE StationDb;
GO

CREATE TABLE dbo.TestResults (
    Id              INT IDENTITY(1,1) PRIMARY KEY,
    MaterialName    NVARCHAR(50)   NOT NULL,
    RecipeName      NVARCHAR(50)   NOT NULL,
    Value           DECIMAL(18,3)  NOT NULL,
    MeasuredAt      DATETIME       NOT NULL DEFAULT GETDATE(),
    ProcessedStatus NVARCHAR(20)   NULL
);

CREATE TABLE dbo.RecipeLimits (
    RecipeName NVARCHAR(50)  NOT NULL PRIMARY KEY,
    LowerLimit DECIMAL(18,3) NOT NULL,
    UpperLimit DECIMAL(18,3) NOT NULL
);

INSERT INTO dbo.RecipeLimits (RecipeName, LowerLimit, UpperLimit)
VALUES ('RECIPE-A', 9.500, 10.500);
```

Simulate the station by inserting a row whenever you want to trigger the integration:

```
INSERT INTO dbo.TestResults (MaterialName, RecipeName, Value)
VALUES ('LOT-4471', 'RECIPE-A', 10.120);
```

Automation Protocol

Create an **Automation Protocol** and select the package `@criticalmanufacturing/connect-iot-driver-database`. Configure the connection to the station database:

Parameter	Value in this tutorial	Notes
`dataBase`	`MSSQL`	Use `MYSQL` for a MySQL server.
`server`	`localhost`	Hostname or address of the database server.
`port`	`1435`	Use `1435` if you started the demo container, and `1433` otherwise.
`databaseName`	`StationDb`	The database holding the two tables.
`user`	`sa`	Use a dedicated account with the least privileges the integration needs.
`password`	`Station#Passw0rd`	The password of that account.
`encrypt`	`true`	Keep it enabled on any server outside a trial.
`requestTimeout`	`15000`	Raise it if the result query is slow on a large table.
`applicationName`	`TEST-01 Integration`	Shown in the database server logs, which makes the integration easy to identify.

Table: Database protocol parameters used in this tutorial

> **Warning**
>
> The account the driver uses needs `SELECT` on both tables and `UPDATE` on `TestResults`. Do not give it broader rights. The driver builds the statements it runs from the Automation Driver Definition, so the account is the boundary that keeps the integration from touching anything else.

Automation Driver Definition

Create an **Automation Driver Definition** for the protocol above, and associate it with the **Resource** `TEST-01`.

Configure the new measurement event

Add an event that detects each new row of `TestResults`.

Event configuration:

Setting	Value	Why
`table`	`[TestResults]`	The table the driver polls.
`argument`	`Id`	The column the trigger watches.
`pollingInterval`	`2000`	Detect a new row within two seconds.
`triggerType`	`OnNewRows`	Raise the event when rows are added.
`trigger`	`RowNumber`	Detect new rows by their row number, which suits an identity column.
`resultDataFormat`	`SingleRow`	Deliver one row per event, so the workflow handles one measurement at a time.
`orderByColumn`	`Id`	Process the rows in the order the station wrote them.
`orderByType`	`ASC`	Oldest first.

Setting	Value	Why
`processingMode`	`ContinueOnProcessingConfirmation`	Do not raise the next event until the workflow confirms this one.
`processingTimeout`	`30000`	Wait 30 seconds for the confirmation.
`maxRetries`	`3`	Raise the same event up to three times before discarding it.
`ignoreInitialValues`	`true`	Do not replay the rows that already exist when the driver starts.

Table: New measurement event configuration

Add one event property per column the workflow needs. The `columnName` extended data binds the property to the database column:

Property name	Data type	Protocol data type	`columnName`
`RowId`	`Integer`	`INT`	`Id`
`MaterialName`	`String`	`NVARCHAR`	`MaterialName`
`RecipeName`	`String`	`NVARCHAR`	`RecipeName`
`Value`	`Decimal`	`FLOAT`	`Value`
`MeasuredAt`	`DateTime`	`DATETIME`	`MeasuredAt`

Table: New measurement event properties

> **Warning**
>
> The protocol data type of `Value` is `FLOAT`, not `DECIMAL`, even though the column is `DECIMAL(18,3)`. Binding a decimal column as `DECIMAL` rounds the value to a whole number. See Handling Decimal Precision in Database.

Configure the confirmation command

Add a command that marks the row as handled and confirms the event, so the driver resumes polling.

Command configuration:

Setting	Value
`type`	`Update`
`argument`	`[TestResults]`
`operation`	`AND`
`resultType`	`SingleResult`

Table: Confirmation command configuration

Command parameters:

Parameter name	`type`	`columnName`	`operation`	Meaning
`Status`	`ParameterValue`	`ProcessedStatus`	`None`	The value written to the column.
`RowId`	`ParameterQuery`	`Id`	`Equal`	Restricts the update to the row that raised the event.

Table: Confirmation command parameters

With `Status` set to `Processed` and `RowId` set to `18`, the driver runs:

```
Update [TestResults] Set ProcessedStatus = 'Processed' Where Id = 18
```

Add a second command of type `ConfirmationDataProcessing`, with a single parameter of type `EventReply` holding the event name. This is the command that tells the driver the row was handled and that it can raise the next event.

Configure the recipe limits command

Add a command that reads the limits of one recipe.

Command configuration:

Setting	Value
`type`	`Select`
`argument`	`[RecipeLimits]`
`operation`	`None`
`resultType`	`SingleResult`

Table: Recipe limits command configuration

Command parameters:

Parameter name	`type`	`columnName`	`operation`
`Lower`	`ParameterValue`	`LowerLimit`	`None`
`Upper`	`ParameterValue`	`UpperLimit`	`None`
`Recipe`	`ParameterQuery`	`RecipeName`	`Equal`

Table: Recipe limits command parameters

With `Recipe` set to `RECIPE-A`, the driver runs:

```
Select LowerLimit, UpperLimit From [RecipeLimits] Where RecipeName = 'RECIPE-A'
```

and returns a single object:

```
{ "LowerLimit": 9.5, "UpperLimit": 10.5 }
```

Automation Controller

Create an **Automation Controller** for the **Resource** `TEST-01`, associate the Automation Driver Definition with it, and create the setup workflow so the driver connects on start. See Create the Setup Workflow.

Flow A - Post each measurement to MES

Overview

A Data Flow workflow that reacts to the new measurement event, classifies the value against the recipe limits, posts it to MES, and confirms the row back to the driver.

When to use

Use this pattern whenever the station database is the source of truth and every row must be processed exactly once, in order.

Build the workflow

1. Add an **On Equipment Event** task and select the new measurement event. The event properties appear as outputs of the task.
2. Add an **Execute Equipment Command** task for the recipe limits command. Connect `RecipeName` from the event to its `Recipe` parameter.
3. Add a **Condition** or an **Expression Evaluator** task that compares `Value` against `LowerLimit` and `UpperLimit`, and produces a `Passed` or `Failed` result.
4. Add a **Data Collection** task, or an **Execute Service Call** task, that posts the measurement to MES for the **Material** named in `MaterialName`.
5. Add an **Execute Equipment Command** task for the confirmation command. Connect `RowId` from the event to its `RowId` parameter, and the classification result to its `Status` parameter.
6. Add an **Execute Equipment Command** task for the `ConfirmationDataProcessing` command, and run it after the update succeeds.

How it works at runtime

- Every two seconds the driver runs the trigger query, which detects that the highest row number changed.
- The driver runs the result query for the next row and raises one event with the five configured properties.
- Because the processing mode is `ContinueOnProcessingConfirmation`, the driver stops raising further events for this table and waits.

- The workflow classifies the value, posts it to MES, updates `ProcessedStatus`, and runs the `ConfirmationDataProcessing` command.
- The driver resumes polling and raises the next event.

Expected outcomes

- One MES record per row of `TestResults`, in the order the station wrote them.
- `ProcessedStatus` filled with `Processed` or `Failed` for every processed row.
- No row processed twice, and no row skipped while the workflow is busy.

Troubleshooting

Symptom	Likely cause
Events stop after the first one.	The `ConfirmationDataProcessing` command never ran. Check that its `EventReply` parameter carries the event name.
The same measurement is posted several times.	The confirmation took longer than `processingTimeout`, so the driver raised the event again. Raise the timeout, or make the MES call idempotent.
All existing rows are processed when the controller starts.	`ignoreInitialValues` is set to `false`.
The measured value arrives rounded to a whole number.	The protocol data type of the property is `DECIMAL`. Set it to `FLOAT`.
No event is ever raised.	The account cannot read the table, or the trigger column in `argument` does not exist. Check the driver log.

Table: Flow A troubleshooting

Flow B - Read reference data on demand

Overview

The recipe limits command is also useful on its own, in a Control Flow that answers a question from MES rather than reacting to the station.

When to use

Use it when a workflow, or an operator action in MES, needs a value that only the station database holds.

Build the workflow

1. Add a trigger that starts the flow, for example an **On Timer** task or a **Message Bus Listener** task.
2. Add an **Execute Equipment Command** task for the recipe limits command, and set its `Recipe` parameter.
3. Use the returned object in the rest of the flow.

How it works at runtime

The command is a plain query. It is executed when the task runs, and nothing is polled. The `resultType` of `SingleResult` returns one object, so a recipe that does not exist returns an empty result rather than an error. Handle that case explicitly in the workflow.

Tuning the polling

The polling interval and the batch size decide the load the integration puts on the database:

- Raise `pollingInterval` when the station writes rows infrequently. Every cycle is a query against the table.
- Set `maxRowsToProcess` when the station can write a burst of rows, so each event carries a bounded batch instead of everything found at once. It only applies when `resultDataFormat` is `AllNew`.
- Set `databasePollingInterval` on the protocol to control how often the driver checks the database for structural changes.
- Use `rawTriggerQuery` and `rawResultQuery` when the generated queries cannot express what you need, for example when the trigger needs a join or an index hint. A raw query replaces the generated one entirely.

> **Warning**
>
> Point the trigger at an indexed column. A trigger query that scans a large table on every polling cycle is the most common cause of a slow station database after a Connect IoT integration goes live.

Summary

You configured a Database Automation Protocol, described a station table as an Automation Driver Definition, and built two workflows on top of it.

The pattern generalizes. Any table a machine writes to can become an Automation Event, and any query that machine's database can answer can become a command. The parts worth carrying over are the confirmation loop, which is what keeps rows from being lost or reprocessed, the write back into the station database, which makes the integration state visible to whoever maintains the station, and the polling settings, which is where a database integration succeeds or fails in production.

Legal Information

Disclaimer

The information contained in this document represents the current view of Critical Manufacturing on the issues discussed as of the date of publication. Because Critical Manufacturing must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Critical Manufacturing, and Critical Manufacturing cannot guarantee the accuracy of any information presented after the date of publication. This document is for informational purposes only.

Critical Manufacturing makes no warranties, express, implied or statutory, as to the information herein contained.

Confidentiality Notice

All materials and information included herein are being provided by Critical Manufacturing to its Customer solely for Customer internal use for its business purposes. Critical Manufacturing retains all rights, titles, interests in and copyrights to the materials and information herein. The materials and information contained herein constitute confidential information of Critical Manufacturing and the Customer must not disclose or transfer by any means any of these materials or information, whether total or partial, to any third party without the prior explicit consent by Critical Manufacturing.

Copyright Information

All title and copyrights in and to the Software (including but not limited to any source code, binaries, designs, specifications, models, documents, layouts, images, photographs, animations, video, audio, music, text incorporated into the Software), the accompanying printed materials, and any copies of the Software, and any trademarks or service marks of Critical Manufacturing are owned by Critical Manufacturing unless explicitly stated otherwise. All title and intellectual property rights in and to the content that may be accessed through use of the Software is the property of the respective content owner and is protected by applicable copyright or other intellectual property laws and treaties.

Trademark Information

Critical Manufacturing is a registered trademark of Critical Manufacturing.

All other trademarks are property of their respective owners.

Copyright © 2023 Critical Manufacturing. All rights reserved.