ACADEMY

Crosser Online Training

ADVANCED SESSION

C1 – Databases

Use databases as sources and destinations for your data

Session C1

Agenda

- Databases
  - SQL (relational databases)
  - NoSQL databases
  - Time series databases
- Modules
  - Ms SQL Insert/Select
  - Redis Set/Get
- Exercises

crosser

SQL Modules

- Wide support for common SQL databases, both self-hosted and cloud-based
- Work with data without SQL knowledge
- Or, use any custom SQL statements
- Common interface both for configuration and data
  → Easily switch between different databases

crosser

Using SQL Databases without SQL

- Use Select modules to get data from a table
  - Choose a table
  - Select columns of interest, or get all
  - Add filters to get the right data
  - Use message data to make dynamic queries
  - The output is always an array of row(s)
- Use Insert modules to write rows to the database
  - Choose a table
  - Insert one row at a time
  - Or batch load multiple rows

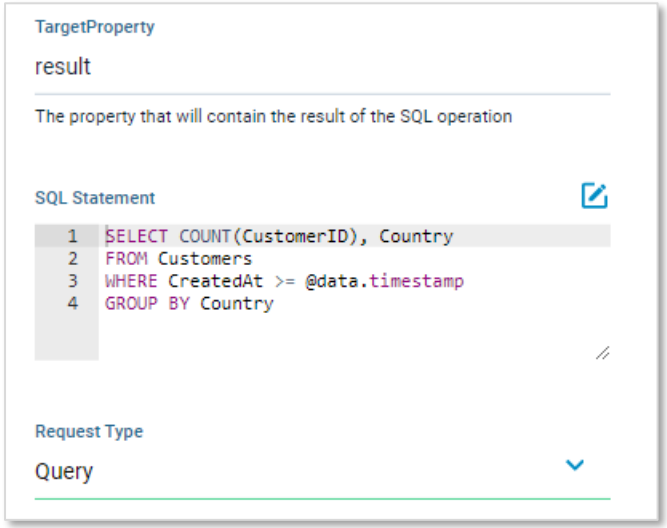
Common message format

```
{
    “col1”: value1,
    “col2”: value2,
    “col3”: value3,
    …
}
```

Property names match column names

Using Custom SQL

- Use Executer modules to run any queries against your database
  - Combine data from multiple tables
  - Take actions: e.g. add columns and tables, or delete records
  - Use stored procedures
  - Use message data as parameters in the SQL statements, for dynamic queries



crosser

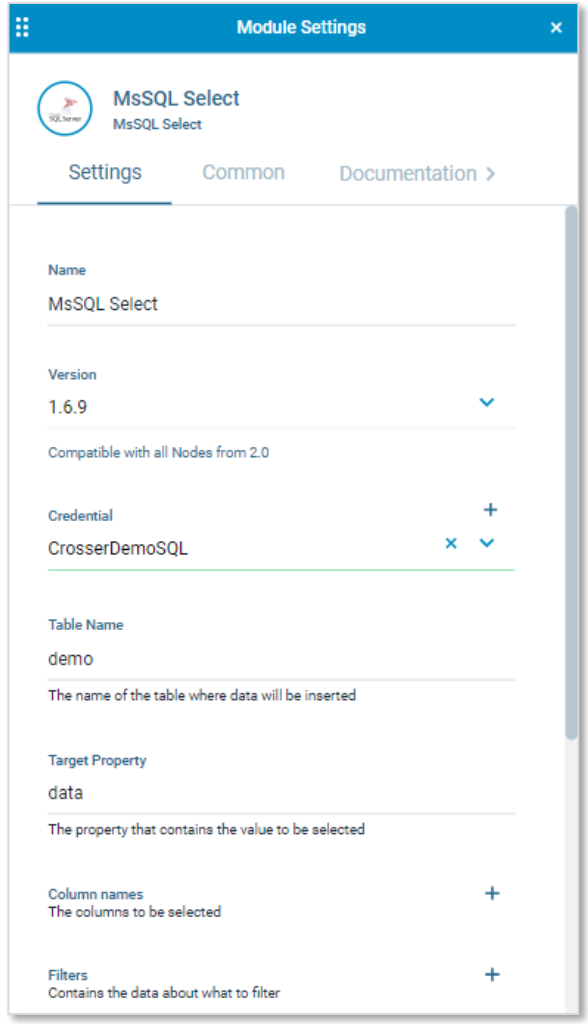
Database Credentials

- Most of the database modules use connection strings to connect to the databases. Connection strings typically contain at least:
  - Server address (host name or IP address)
  - Database name
  - Username and password
  - Other, optional, settings may be available depending on the database used
  - Page for connection string examples: connectionstrings.com
- Example Ms SQL connection string:
  - Server=myServerAddress;Database=myDataBase;User Id=myUsername;Password=myPassword;
  - Note that the database is specified in the connection string, ie using another database will require a new credential
- Database connection strings are added on the Credentials page or from the module settings

Module

Ms SQL Select

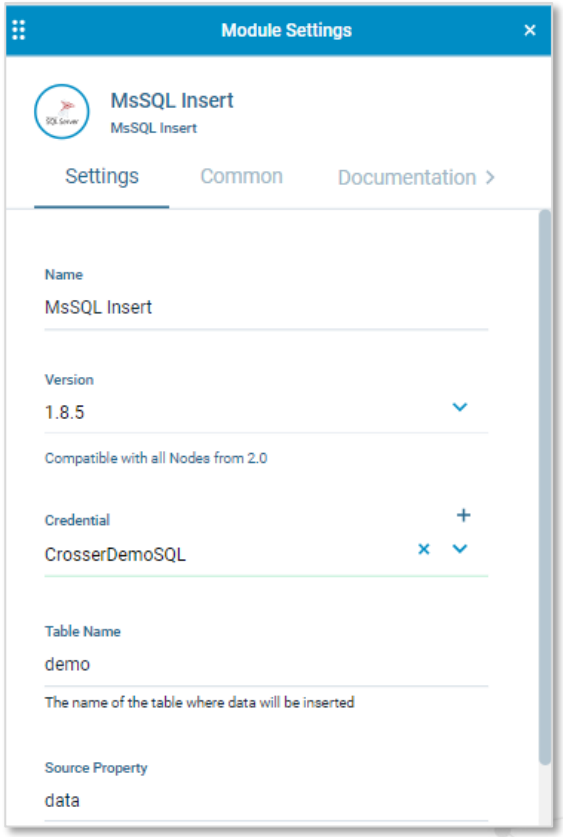
- The *Ms SQL Select* module retrieves rows from a Microsoft SQL Server database table
- Runs a query and returns data each time a message is received
- Will get data from one configured Table
- Get all, or a specified subset of, Columns
- Possible to set how many rows to skip or read
- Order By and Sort Order can be specified
- Filters - A list of conditions that are translated into a SQL WHERE statement
  - Behavior - possible to filter using static data or use data from a Property on the incoming messages
  - If many filters are used each row must match every one of them (AND)
- Results are delivered as arrays [1:n] of objects, with one row per object



Module

Ms SQL Insert

- The *Ms SQL Insert* module inserts flow messages into rows of a table in a Microsoft SQL Server database
- Credential – Connection string to the DB added in Credential page
- Table name – Name of the table on the Ms SQL server
- Property names and datatypes must be same as in the database table

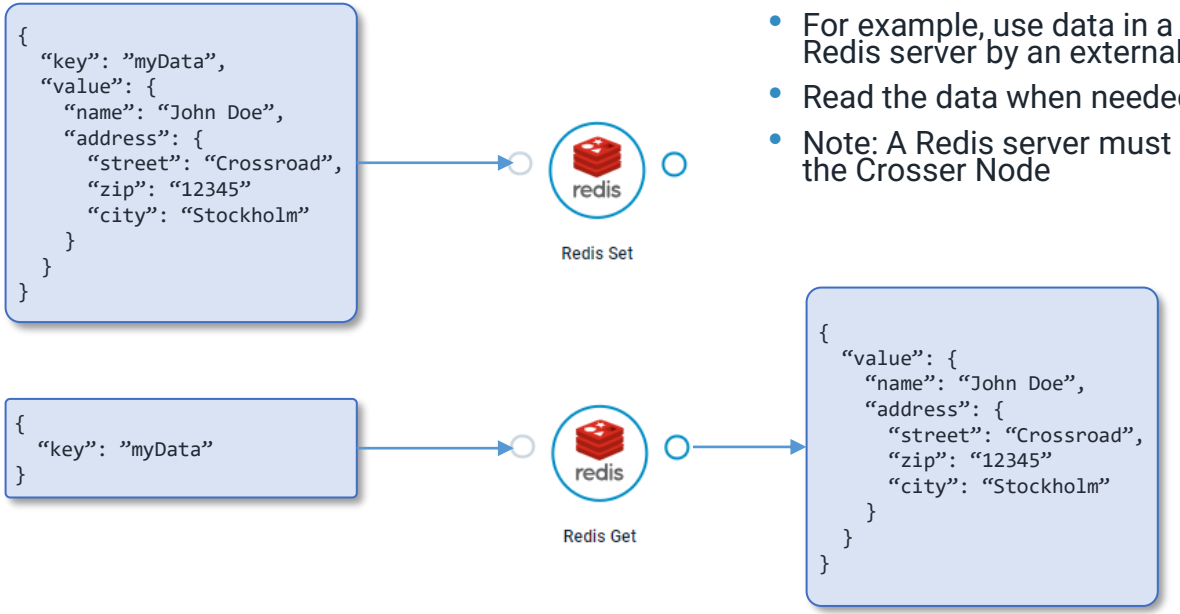


NoSQL Databases

- Redis (key/value and pub/sub database)
  - Data is written to a specified Key or Topic
- MongoDB (document database)
  - Data is written to a Collection
- Internal (key/value)
  - Data is written to a key
  - Data can be stored on disk or in memory
  - Always available

- These databases have no specific requirements on the data inserted, any message (object) can be used

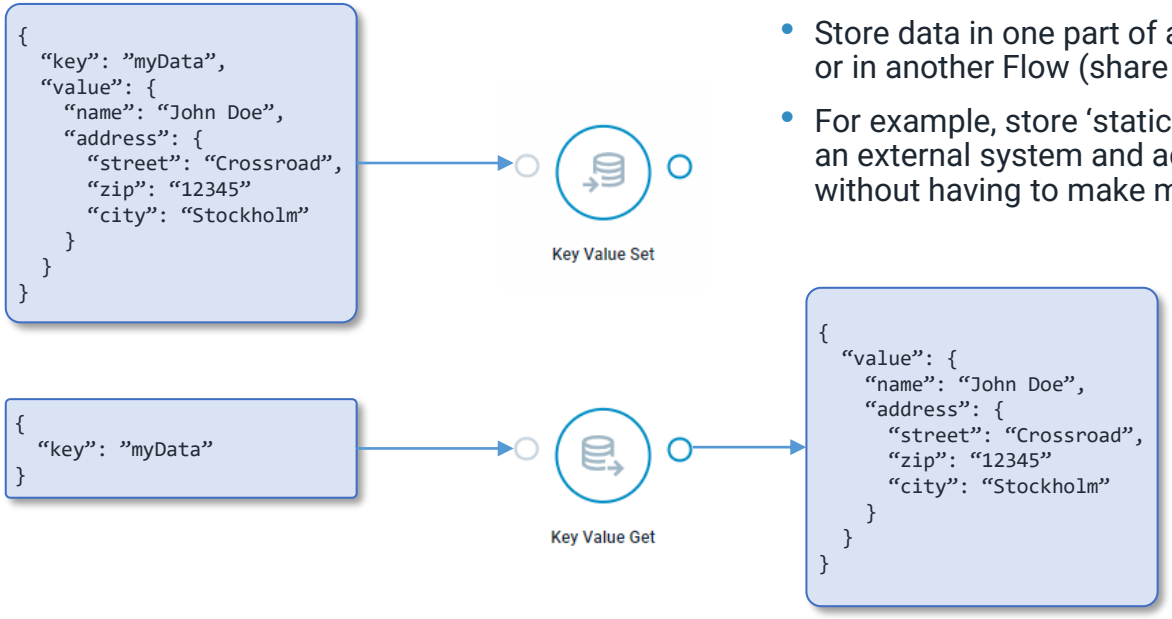
Key/Value Stores - Redis



- Decouple storage and usage of data
- For example, use data in a Flow that has been stored in the Redis server by an external system, or another Flow
- Read the data when needed
- Note: A Redis server must be available and accessible from the Crosser Node

crosser

Key/Value Stores - Crosser

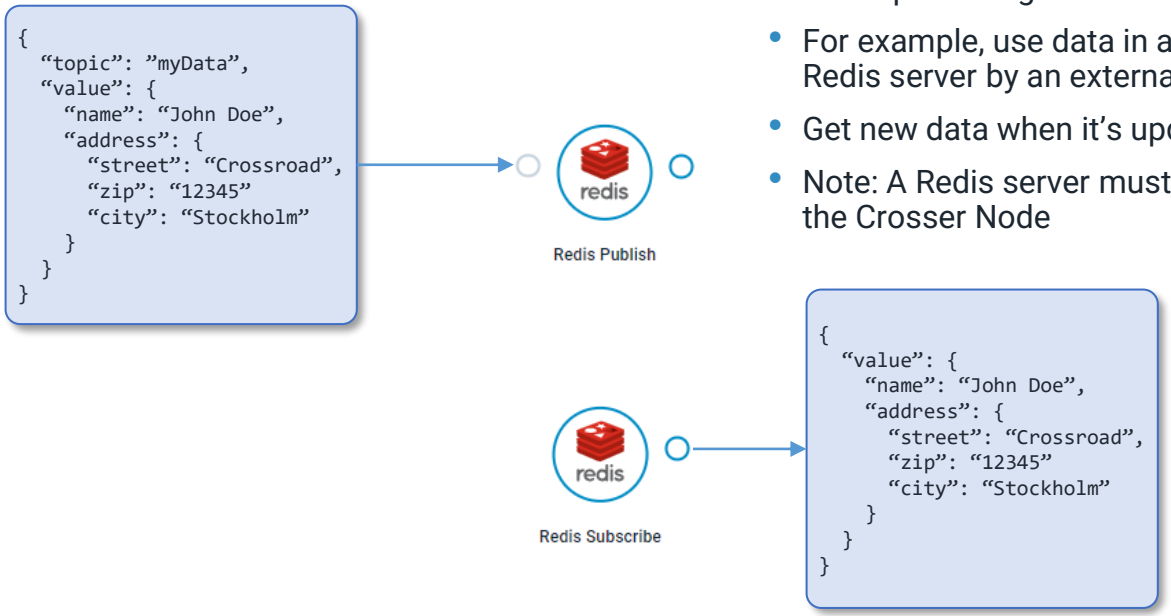


- Store data in one part of a Flow and use it in another part, or in another Flow (share data between Flows)
- For example, store 'static' data that has been pulled in from an external system and add it to streaming data messages without having to make multiple external requests

Each Crosser Node has a built-in key/value store!
Store data in memory or on disk

crosser

Pub/Sub Stores - Redis



- Decouple storage and usage of data
- For example, use data in a Flow that has been stored in the Redis server by an external system, or another Flow
- Get new data when it's updated
- Note: A Redis server must be available and accessible from the Crosser Node

crosser

Time Series Databases

- A time series database is a database optimized for time series data. Time series data are simply measurements or events that are tracked, monitored and aggregated over time.
- All data is associated with a timestamp, either provided explicitly, or added by the database when ingesting data
- Example of use cases where a time series database make sense:
  - Monitoring physical systems (sensor data): Equipment, machinery, connected devices, the environment
  - Monitoring software systems (performance metrics): Virtual machines, containers, services, applications
  - Asset tracking (positional data): Vehicles, trucks, physical containers, pallets
- Crosser supports the following time series databases:
  - Influx DB
  - TimescaleDB

Timescale

crosser

EXERCISES

You need docker and docker-compose on your local machine!

Setting up the training environment

- The exercises require a local set up with a Crosser Node and databases running in Docker
- Use the docker-compose.yml file that can be downloaded from the Help Center on the page for this session
- Register a Node in Control Center and add the credentials to the docker-compose.yml file
- Install the software by running the following command in the same directory as your docker-compose.yml file:

```
docker-compose up –d
```

- All exercises must run on the Node you registered above

Exercise C1.1

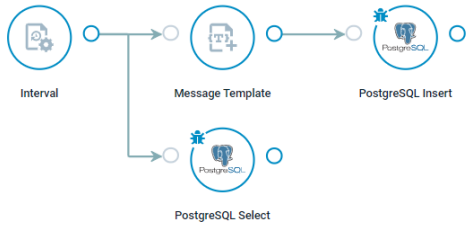
Overview

- In this exercise you will use a PostgreSQL database to store calibration values for sensor data.
- We will start by writing the calibration data and then use this data to adjust our sensor data.
- Finally, we will write the calibrated data to the database
- To access the PostgreSQL database you need the following connection string: `Server=postgres;Database=crosser-dev;User ID=crosser;Password=CrosserDev2023;`

  You can add this from within one of the *PostgreSQL* modules, or on the Credentials page
- The database has two empty tables: *calibration* (name, value) and *data* (timestamp, name, value)

Exercise C1.1

Add calibration data

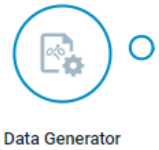


name	value
machine-1	1.5
machine-2	2
machine-3	2.5

1. Use a *PostgreSQL Insert* module to insert data as shown above, into the calibration table (the *Message Template* module can come in handy here)
2. Run your Flow once to insert the data and then disable the insert module. Use a *PostgreSQL Select* module to verify that the data has been written to the database.

Exercise C1.1

Generate sensor data



1. Use a *Data Generator* module to produce some 'sensor' data. Use the default example ('ADD EXAMPLE') and then make the following changes:
   - Number of Samples: 3
   - Data Rules→name→Behavior: Identifier
2. Run the Flow and verify that you get messages with random temp/pressure data and where 'name' is one of: machine-1, machine-2 or machine-3

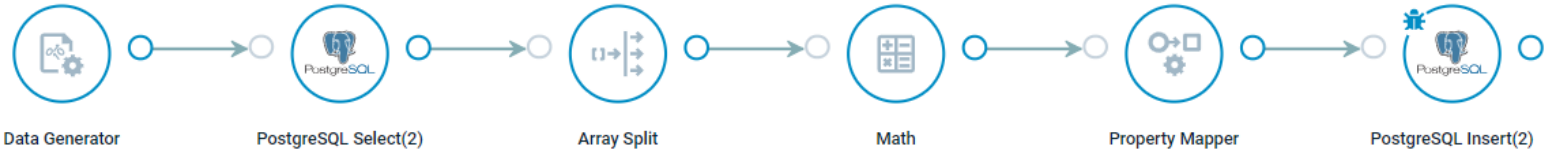
Exercise C1.1

Fetch calibration data

1. Add a *PostgreSQL Select* module to fetch the calibration data corresponding to each sensor value:
   - Add a filter on the name column and check if it equals the name from the incoming message. Set 'Behavior' to Property to indicate that the data should be taken from the message property.
   - Set the 'Target Property' to calibrationData.
2. Run the Flow and verify that your messages now also contain calibration data corresponding to the name of the sensor (compare with the calibration table)

Exercise C1.1

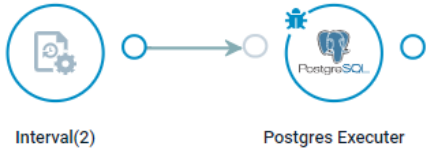
Write adjusted data to the database



1. Use a *Math* module to adjust the *temp* values by multiplying with the corresponding calibration data
2. Use a *Property Mapper* to create a message with properties matching the columns in the 'data' table, with data from the sensor name, the calibrated temp value and a timestamp from the *Data Generator* module.
3. Run the Flow and insert some values
4. Change the *PostgreSQL Select* module you used to check the calibration data to show you the data inserted into the 'data' table.

Exercise C1.1

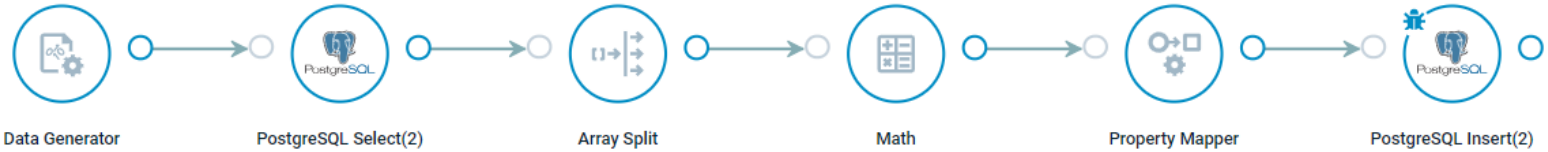
Extra: Use the Executer module to run generic SQL statements



1. Use a *Postgres Executer* module to run some generic SQL statements. For example:
    1. Remove all data from a table:
       DELETE FROM data
    2. Check what tables that are available:
       SELECT * FROM pg_catalog.pg_tables WHERE schemaname != 'pg_catalog' AND schemaname != 'information_schema'
    3. Check what columns that are available:
       SELECT * FROM information_schema.columns WHERE table_schema = 'public' AND table_name = 'data';

Exercise C1.1

Wrap-up



In this exercise you have tried som of the SQL modules, both to read data from a database and write data back to the database.

Some things to consider:

- How do you select which columns to write data to?
- How do the messages differ when writing a single row, versus writing multiple rows?
- Why did you have to add an *Array Split* module after the *Select* module?
- What happens if you change the behavior on the filter in the *Select* module to *Static*?

crosser

Exercise C1.2

Overview

- In this exercise you will rebuild the previous exercise by replacing the PostgreSQL database with a Redis key/value store for the calibration lookup.
- As before we will start by writing the calibration data into Redis and then use this data to adjust our sensor data. The name of the sensor will be the *key* and the calibration data the *value*.
- Finally, we will write the calibrated data to the database, just like before.
- To access the Redis database you need the following connection string: redis
  You can add this from within one of the *Redis* modules, or on the Credentials page

crosser

Exercise C1.2

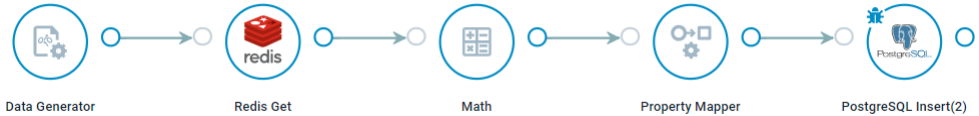
Add calibration data

name	value
machine-1	1.5
machine-2	2
machine-3	2.5

1. Make a copy of the Flow you built in the previous exercise (use 'New Flow from draft' in the Flow Studio)
2. Start by writing the calibration values to the database. Replace the *PostgreSQL Insert* module with a *Redis Set* module. This time we cannot write all values at once, they need to be written one at a time.
3. Use the *data.name* as 'Source Property' and *data.value* as 'Value Property'.

Exercise C1.2

Adjust the sensor data



1. Replace the *PostgreSQL Select* module with a *Redis Get* module in the sensor data update Flow.
2. Use *name* as 'Source Property' and set the 'Target Property' to *calibrationData*. You need to update the expression in the *Math* module correspondigly.
3. Enable 'Keep Properties' in the *Redis Get* module.
4. Run the Flow and verify that it's working like before.

Exercise C1.2

Wrap-up

In this exercise you have used a key/value database (Redis) to lookup the calibration values.

Some things to consider:

- What is the difference between a key/value database and a SQL database? Why could both be used for this use case.
- Why did we have to enable 'Keep Properties' on the *Redis Get* module?

Extra: We could have used the built-in key/value store instead of Redis. Try modifying your flow by replacing the Redis modules with the *Key Value Set/Get* modules. With these modules you get the option to use an in-memory database ('Persistant storage' false), which could be advantageous when you need fast lookups.

crosser

SESSION – C1 END