



# Genesis Create Instructions

- [1. Create](#)
- [2. Post-Create and loading into IntelliJ](#)
- [3. Pro-code Enhancements](#)
  - [3.1 Multi-directional Trade Calculations](#)
  - [3.2 Pivot Positions](#)
  - [3.3 Notifications](#)
  - [3.4 Reporting](#)

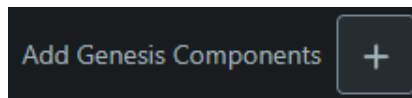
## 1. Create

The following instructions will give you a step-by-step guide to build the fundamentals of an FX Trading platform using Genesis Create. This application will have the following functionality:

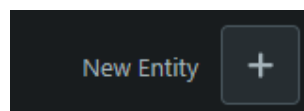
- Manual trade entry, update and cancel.
- Trade blotter - live trade feed showing version history.
- Positions view - show currently fluctuations and newly calculated rates.
- Notifications - alert the user on changes to currencies based on totals and thresholds.

1. Login to <https://create.genesis.global/> with the userid and password provided in the welcome email.
2. Provide a **valid project name** and project description - we recommend naming the project 'fx-trading' but feel free to give it your own custom branding.
3. The next stage will be adding your Genesis components. For now, select the following:
  - a. Real Time Queries - Dataservers will give us the real time trade feed
  - b. Real Time Aggregation - Consolidators will allow up to work out positions and calculate values based on fluctuating rates
  - c. Notifications
  - d. Reporting

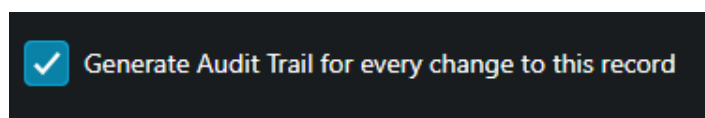
You can do this by selecting **Add Genesis Components** , top right



4. Click on Next to take you to the **Entity Models** page, where you will start to populate entities (tables) and their respective attributes (fields) within your application.
5. Create the entity by clicking **New Entity** button, again in the top right and provide **TRADE** as the Entity Name, and click save. This will provide the base trade fact data for our application.



6. Check the **Generate Audit Trail** button.



7. We can start adding attributes you'll need to use the **New Attributes** screen on the right hand side. To start, we'll only need to update the **Name** , **Type** and **Allow Null Values** fields for most attributes. For the **ENUM** type attributes, you'll also have to provide some values, of which the first in the list can be set to the **Default Value** .

### New Attribute

Primitive

▼

Name

▼

Type

STRING

▼

☐ Allow Null Values

Max




☐ Sensitive Data (display avoided where possible)

Add Attribute

Field Name	Type	Allow Null Values?	Value
TRADE_ID	string		
TRADE_DATETIME	datetime		
TRADE_STATUS	enum		New, Amended, Cancelled
TRADER_NAME	string	Y	
SALES_NAME	string	Y	
BROKER_CODE	string	Y	
ENTITY	string	Y	
BOOK	string	Y	
CLIENT_NAME	string	Y	
SOURCE_CURRENCY	string		
TARGET_CURRENCY	string		
SETTLEMENT_DATE	date		

Field Name	Type	Allow Null Values?	Value
NOTIONAL	double		
SIDE	enum		BUY, SELL
RATE	double		

8. Once you have added all these attributes, we'll want to set `TRADE_ID` to be the Primary Key field. We'll also want to make sure this value is a generated attribute, and the sequence is set to `TR`. This means that `TRADE_ID` field will auto generate values for us.

Entity Attributes					
Attribute	Primary Key?	Entity Ref	Generated	Sequence	Type
TRADE_ID	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	TR	STRING 
BUY_SELL	<input type="checkbox"/>				ENUM 
TRADE_DATE	<input type="checkbox"/>				DATE 

9. Repeat steps 5-8, but for the following two tables:

`EXCHANGE_RATES` (keep track of exchange rate data)

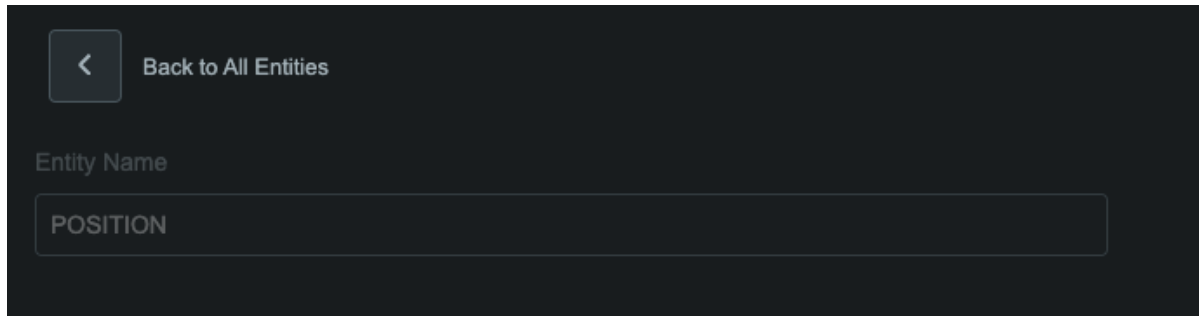
Field Name	Type	Allow Null Values?	Primary Key?	Value
SOURCE_CURRENCY	string		Y	
TARGET_CURRENCY	string		Y	
RATE_DATE	date		Y	
RATE	double			

`POSITION` (fed by the consolidator, a calculated table to keep track of positions)

Field Name	Type	Allow Null Values?	Primary Key?	Value
SOURCE_CURRENCY	STRING		Y	
TARGET_CURRENCY	string			
SETTLEMENT_DATE	date		Y	

Field Name	Type	Allow Null Values?	Primary Key?	Value
AMOUNT	double			

10. Go Back to All Entities page by clicking button at top



The screenshot shows a dark-themed interface. At the top left, there is a button with a left-pointing arrow and the text 'Back to All Entities'. Below this, there is a label 'Entity Name' followed by a text input field containing the word 'POSITION'.

11. Repeat these steps until all of the entities are added to the project
12. We'll skip the views section for now. Please hit Next and move on to Queries.
13. (OPTIONAL) - Click on `New Query` to configure real time datasever that will stream the data from the entities and views. Select `Entity or View` from dropdown and provides query name, followed by clicking `Add Query` button. You can decide to include or not include the fields that you wants to include. After that click on `Back to Queries` button to go back to Query page.
14. (OPTIONAL) - Repeat the previous step to add Additional Queries if you wish. Click on `Next` to move onto define further components.
15. Next we'll be defining a consolidator to calculate the position by currency.
16. Provide the `Consolidator Name` , `Input Entity` and `Output Entity` , Define the `Group By` and respective `Input Field` → `Aggregated Method` → `Output field` .
  - a. Name: POSITION\_CALC
  - b. Input Entity: TRADE
  - c. Output Entity: POSITION
  - d. Group By Clause should Group By `SETTLEMENT_DATE` and `SOURCE_CURRENCY`
  - e. Select the following aggregation function which should be defined for you

Aggregation Configuration			
Enabled?	Input Attribute	Aggregation Function	Output Attribute
<input checked="" type="checkbox"/>	NOTIONAL	SUM	AMOUNT

17. (OPTIONAL) - Repeat the pervious step if you wish to create any more consolidators, for example you could add consolidation logic of exchange rate change. Once complete, click **Next** to define user interface.
18. Click **Next** to move past the business components section - we'll come back later to configure these components!
19. Depending on your user interface needs, you can add one of more pages either by clicking **+** button next to **HOME** or Click on **\*** icon near Manage Pages. Here, we'll ask you to create a basic front end - this will allow you to keep track of your trades, being able to book new trades, amend or cancel and a positions view to see book consolidation in real time.
  - a. First, let's create the trade blotter. You can rename 'HOME' to 'Blotter' in the Manage Pages config.
  - b. Add a tile 'Trade Blotter' - **Real-Time Data Grid + Data Modifications** . This gives you the real time dataserver element but also the ability to create new trades and amend existing trades. This will create necessary events in the back end event handlers.
  - c. Set the data source to be **TRADE**
  - d. Ensure the allowed actions are 'Insert', 'Update' and 'Delete' to generate that code
  - e. Now we'll need to define the table structure. TRADE\_ID is generated by the system, so hide that, but feel free to play with all the other fields and set data types.
  - f. Add a new tile 'Positions' - this will just be a read only grid, reading from the positions table that is being updated automatically by the consolidator
  - g. Datasource will be **POSITION**
  - h. We'll tidy this table up in pro-code so it provides the more well known 'pivot' look and feel of a positions ladder.

- i. Finally, you should add the `EXCHANGE_RATE` table similar to the Trade Blotter ( `Real-Time Data Grid + Data Modifications` ) so you're able to add/update rate data
20. Click next to download the project by clicking on `Generate Project`
21. Extract the zip file and open in IntelliJ - there is a video embedded in the final step of Genesis Create which demonstrates this process, but we recommend looking through the section 2, **Post-Create and loading into IntelliJ**, to get you up and running and make sure your local development environment is correctly configured.

At the end of day one, you should have the basis of an FX platform generated from Genesis Create with a number of core Genesis components:

- Authentication and authorisation - have the ability to add field level permissions, create users, rights and profiles
- Dataservers - have a live feed of trade data as users create new and amend existing trades
- Events - basic CRUD events added for each entity created. This will be where you'll start your 'pro-code' experience and build application logic, connectors and API integrations
- Consolidators - the backbone behind data manipulation in Genesis. Listen to table updates and run real time calculations

and have a seed project to continue your build. You will also have deployed the project locally, learning more about:

- The Genesis plugin - build, deploy and run applications on your local machine
- Genesis tooling - deploy data to your local database and query directly
- Intellisense - experience how integrations with IntelliJ and the Genesis DSL, GPAL, autocompletes and aids your Genesis development

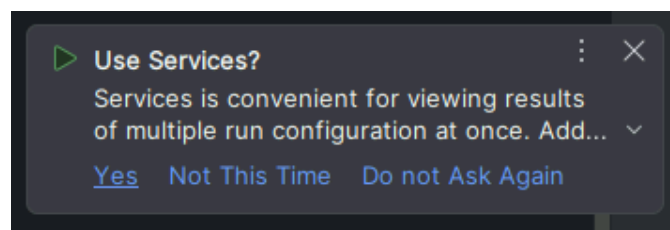
Once you've opened the app in IntelliJ and running in your development environment, we recommend transitioning to the [Genesis Academy](#) (Day 2), where you'll learn about components and the Genesis stack to a point where you can return to the application and build enhancements to finalise the FX trading platform.

On completion of the Academy training, come back and visit section 3, **Pro-code Enhancements**, to run through how to begin working directly with the platform and build world class financial applications.

## 2. Post-Create and loading into IntelliJ

Some important points on install:

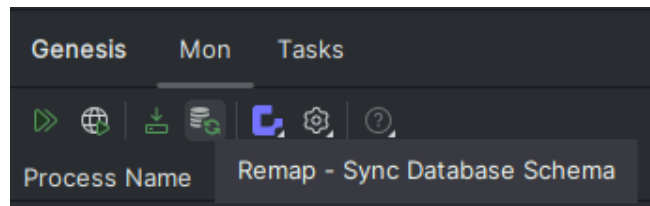
- Please make sure you open the project at the root in IntelliJ. The Genesis plugin may not detect your Genesis project if you open from an external a folder (i.e. you unzip and then open `fx-trading/fx-trading` )
- Make sure your SDK is set up in IntelliJ - go to `File` → `Project Structure` → `Project` and make sure your **Java version is set to 17**
- **Make sure Gradle is using the same SDK - go to `File` → `Settings` and search for `Gradle` . Under `Build, Execution, Deployment` , open `Gradle` and select the `Gradle JVM` to be the Project SDK (17)**
- When prompted, please click `Load Gradle Project` in the bottom right of your IntelliJ window. Please ensure the SDK is set to Java 17 before running this step. if you've loaded the project with the wrong SDK, you'll need to restart IntelliJ and may experience crashes
- When running the project, if you see a pop up in the bottom right asking if you'd like to `Use Services` , it is worth clicking 'Yes' for a better UX when running your application.



- Once the project is open, and recognised by the Genesis plugin, you can then start running Gradle tasks to build and start you application. To simplify the sequence, run the following (you can learn more about these steps during Genesis Academy and our training documentation):

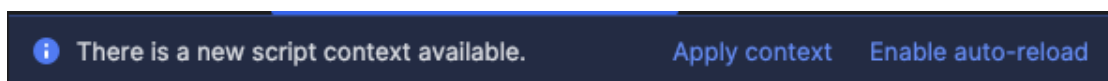


- `prepare local environment` - click on prepare local environment to generate all configuration files.
- `remap` - executed from `Mon` section of the plugin. This also runs `genesisInstall` and gives you the option to import data!



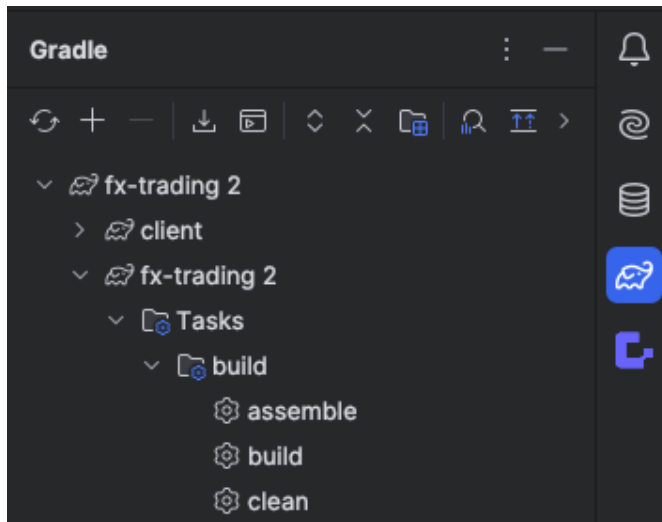
**⚠ Note**, make sure you say yes to importing the data! this process will loop through all the 'data' folders in your project and import crucial user, rights and profile data. Check out more information in our documents are via the Academy

- **OPTIONAL** - If you want to add a new user for yourself, you can update `USER.csv` here: `fx-tradingserver\fx-trading-app\src\main\genesis\data\USER.csv`  
The password is a concatenation of your username and your password, then sha512 encrypted  
Right click the csv from the project plane and select `Import CSV(s) to Genesis`
- Once you've started the GUI, you'll be able to log using User / Pass: `admin` / `genesis`
- If you want to add/remove/amend a field and/or a table definition, make sure you run your `genesisInstall` to 'deploy' the change to the environment before running a remap
- When opening a GPAL file for the first time, make sure you click 'Enable auto-reload' to make the most of Genesis' intellisense. You may need to close and re-open IntelliJ if this is not appearing

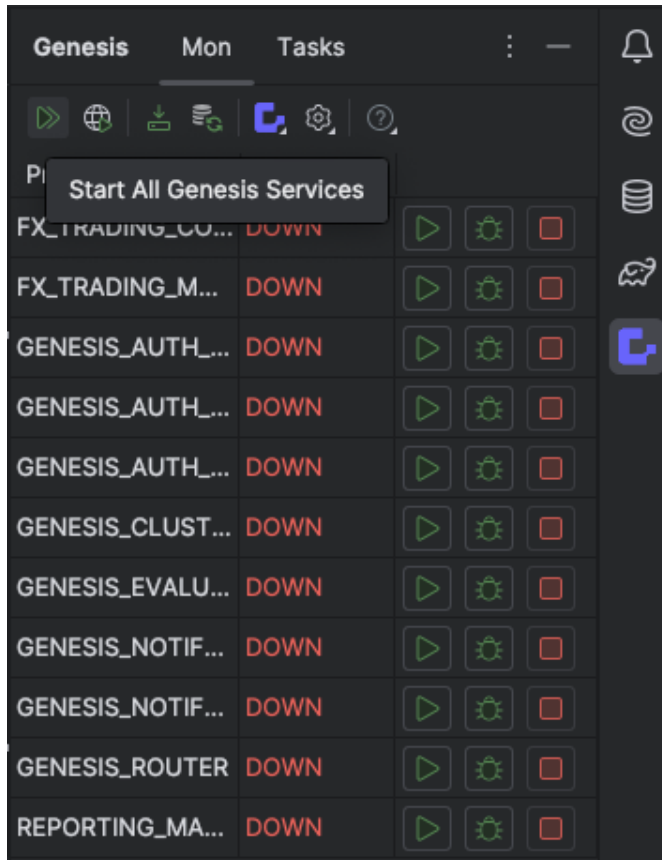


Now it's time to start your application!

- First you'll want to make sure your application builds - go to the Gradle plugin on the right hand side of your IntelliJ window, open up the root node and hit assemble:



- If you get any errors at this point, attempt to debug yourself but feel free to contact [dev.support](mailto:dev.support) to guide you through.
- Now, under within the Genesis plugin, click start - this will bring up all the backend services



- Once they are marked as **HEALTHY**, start the UI - this will launch the application in your local browser. you can log in using the the users mentioned previously.



## 3. Pro-code Enhancements

There are few things we'll need in 'pro-code' to do to make the app a fully functional trading view.

### 3.1 Multi-directional Trade Calculations

Update the **TRADE\_INSERT** event handler in your **fx-trading-eventhandler.kts** file so that it will insert two records into database. One on the **SELL** side and other one to **BUY** side. This will ensure the proper calculation for the **Position**.

```

eventHandler<global.genesis.gen.dao.Trade>("TRADE_INSERT") {
    onCommit { event ->
        var details = event.details
        LOG.info("original trade {}", details)
        var otherSide: Trade = details.copy()
        otherSide.targetCurrency = details.sourceCurrency
        otherSide.sourceCurrency = details.targetCurrency
        if(details.side == Side.BUY){
            otherSide.side = Side.SELL
            otherSide.notional = - details.notional
        }else{
            otherSide.side = Side.BUY
            details.notional = - details.notional
        }
        LOG.info("original trade after assignment {}", details)
        LOG.info("other side trade after assignment {}", otherSide)
        val firstLeg = entityDb.insert(details)
        val secondLeg = entityDb.insert(otherSide)
        // return an ack response which contains a list of record IDs
        ack(listOf(mapOf(
            "TRADE_ID" to firstLeg.record.tradeId,
            "TRADE_ID" to secondLeg.record.tradeId,
        )))
    }
}

```

## 3.2 Pivot Positions

Let's create a consolidated view of your FX trades. To do this, we need to create a new custom element and replace the existing grid pro. This new element will subscribe to a stream from the consolidator output table and transform the output.

First update your `fx-trading-dataserver.kts` file and add a query called ALL\_POSITIONS which will stream updates from the positions table.

```

dataServer {

  query("ALL_POSITIONS", POSITIONS)
  ...
}

```

Next create your new custom element in a new file - `client/src/components/positions-grid.ts`. This class will subscribe to the new ALL\_POSITIONS stream. It will transform the array and group trades by date and create an aggregate daily amount for each currency. It will also dynamically update the column definitions on the grid so that there is a column for each currency.

```

import { customElement, FASTElement, html, ref } from '@microsoft/fast';
import { Connect } from '@genesislcap/foundation-comms';
import { GridPro } from '@genesislcap/foundation-zero-grid-pro';
import { getDateFormatter, getNumberFormatter } from '../utils';

@customElement({
  name: 'positions-grid',
  template: html<PositionsGrid>`
    <zero-grid-pro
      ${ref('grid')}
      @onGridReady=${(x, e) => x.onGridReady(e)}
    >
  </zero-grid-pro>
  `
})
export class PositionsGrid extends FASTElement {

  @Connect connect!: Connect;

  grid: GridPro;

  private rowData = [];

```

```

connectedCallback() {
  super.connectedCallback();
  this.grid.gridOptions = {};
}

onGridReady(e): void {
  this.connect.stream('ALL_POSITIONS', message => {
    const currencies = this.getAllCurrenciesFromStream(message.ROW);
    const colDefs = this.createColDefs(message.ROW, currencies);
    this.rowData = [...this.rowData, ...this.mapRowData(message.ROW)];
    this.grid.gridApi.setColumnDefs(colDefs);
    this.grid.gridApi.setRowData(this.rowData);
  }, console.error);
}

private mapRowData(message: any[], currencies: string[]): any[] {
  if (!message) {
    return [];
  }

  const rowMap: { [key: string]: any } = message.reduce((acc, row) => {
    const { SETTLEMENT_DATE, AMOUNT, SOURCE_CURRENCY } = row;

    if (!acc[SETTLEMENT_DATE]) {
      acc[SETTLEMENT_DATE] = {
        settlementDate: SETTLEMENT_DATE,
        [SOURCE_CURRENCY]: AMOUNT
      }
    } else {
      const existingValue = acc[SETTLEMENT_DATE][SOURCE_CURRENCY];
      acc[SETTLEMENT_DATE][SOURCE_CURRENCY] = existingValue + AMOUNT;
    }

    return acc;
  }, {});
}

```

```

    return Object.keys(rowMap).map(key => rowMap[key]).sort((a,b) =>
}

private getAllCurrenciesFromStream(message: any[]): string[] {

    if (!message) {
        return [];
    }

    return message.reduce((currencies: string[], row) => {
        const { SOURCE_CURRENCY } = row;
        if (!currencies.includes(SOURCE_CURRENCY)) {
            currencies.push(SOURCE_CURRENCY)
        }

        return currencies;
    }, []);
}

private createColDefs(message: any[], currencies: string[]) {

    const colDefs: any[] = [
        {
            field: "settlementDate",
            headerName: "Settlement Date",
            hide: false,
            valueFormatter: getDateFormatter("en-GB", {
                year: 'numeric',
                month: 'long',
                day: 'numeric',
            })
        }
    ]

    if (!message || message.length) {
        currencies.forEach(c => {
            colDefs.push({

```

```

        field: c,
        headerName: c,
        hide: false,
        valueFormatter: getNumberFormatter("0,0.00", null)
    })
});

return colDefs;
}

}
}

```

Next, add your new custom element to the `client/src/components/components.ts` file so the browser knows how to render it.

```

import { EntityManagement } from '@genesislcap/foundation-entity-man
import { Form } from '@genesislcap/foundation-forms';
import { Navigation } from '@genesislcap/foundation-header';
import { foundationLayoutComponents } from '@genesislcap/foundation-
import { getApp } from '@genesislcap/foundation-shell/app';
import { FoundationRouter } from '@genesislcap/foundation-ui';
import {
    assureDesignSystem,
    DesignSystemModule,
    ResourceType,
} from '@genesislcap/foundation-utils';
import { zeroGridComponents } from '@genesislcap/foundation-zero-gri
import { g2plotChartsComponents } from '@genesislcap/g2plot-chart';
import { logger } from '../utils';
import { PositionsGrid } from '../positions-grid';

/**
 * Ensure tree shaking doesn't remove these.
 */

```



```

FoundationRouter;
Navigation;
EntityManagement;
Form;
PositionsGrid; // IMPORTANT - YOUR NEW ELEMENT

/**
 * zeroDesignSystemImport.
 * @remarks
 * Attempts to use a module federation version of zero before fallin
 * @internal
 */
async function zeroDesignSystemImport(): Promise<DesignSystemModule>
    let module: DesignSystemModule;
    let type: ResourceType = ResourceType.remote;
    try {
        module = await import(
            /* webpackChunkName: "foundation-zero" */
            'foundationZero/ZeroDesignSystem'
        );
        return assureDesignSystem(module);
    } catch (e) {
        logger.info(
            `Please note remoteEntry.js load errors are expected if module
        );
        type = ResourceType.local;
        module = await import(
            /* webpackChunkName: "foundation-zero" */
            '@genesislcap/foundation-zero'
        );
        return assureDesignSystem(module);
    } finally {
        logger.debug(`Using '${type}' version of foundation-zero`);
    }
}

/**
 * registerComponents.

```

```

* @public
*/
export async function registerComponents() {
  const designSystem = await zeroDesignSystemImport();
  const { provideDesignSystem, baseComponents } = designSystem;

  /**
   * Register any PBC components with the design system
   */
  getApp().registerComponents({
    designSystem,
  });

  provideDesignSystem().register(
    baseComponents,
    zeroGridComponents,
    g2plotChartsComponents,
    foundationLayoutComponents,
  );
}

```

Then update your route template in `client/src/routes/blotter/blotter.template.ts` and add the custom component.

```

<zero-layout-item title="Positions">
  <positions-grid></positions-grid>
</zero-layout-item>

```

**And now we can add new functionality!**

### 3.3 Notifications

We've already selected the Notifications PBC when creating the application so the foundations have been laid. Firstly, let's make sure your user can access the configuration:

1. You'll see that the notification module comes with build in rights and profiles - you can check this under the `genesis-notify` section of the build:

```
fx-trading-app/build/genesis-home/genesis-notify/data
```

2. All csv files under 'data' folders will get sent to the DB when you agree to sending the data during a remap. But you can double check this using `DbMon` . For example;

```
DbMon>table RIGHT
DbMon:RIGHT>search CODE=="NotificationRuleCreate"
=====
RIGHT
=====
Field Name                                Value
=====
TIMESTAMP                                2024-05-21 10:03:06.647(n:6
CODE                                     NotificationRuleCreate
DESCRIPTION                             Create a notification rule
-----
-----
```

3. If this data is not present, run a remap and import all data
3. Your user must have a profile and rights to access notification configuration. Make sure this is correct in the `RIGHT_SUMMARY` table using `DbMon` . If it's not present, find and run the `ConsolidateRights` script in the Genesis Plugin.
4. You're now free to create notifications! Make sure you enter details for every field

The screenshot shows a 'Create Rule' dialog box in a dark-themed application. The dialog is titled 'Create Rule' and has a close button (X) in the top right corner. It contains several input fields and dropdown menus. The 'Name' field is 'New Trade Threshold' and the 'Description' is 'Notify when a new trade is entered above 10mUSD'. The 'Resource' dropdown is 'TRADE', 'Topic' is 'Everyone's Screen', 'Severity' is 'SERIOUS', and 'Update Type' is 'INSERT'. The 'Header' field is 'TRADE >\$10M USD' and the 'Message' is 'A large trade has been entered manually'. The 'Condition(s)' section shows a single condition: 'NOTIONAL' > 'GREATER\_THAN' > 'VALUE' > '10000000'. There is a '+ Condition' button below the condition list. A 'Submit' button is at the bottom right. The background shows a sidebar with 'Rules', 'Templates', and 'Routes' tabs, and a top navigation bar with 'Blotter', 'Profiles', 'User Management', 'Notifications', and 'Reporting' tabs. A user profile 'admin' is in the top right corner.

7. Try adding a rule on the positions table such that you'd get a notification when a currency total, on a date, exceeds a threshold

### 3.4 Reporting

As you've already configured the reporting module during Create, and imported your users, profiles and rights when setting up notifications, you're free to build reports!

You can configure reports on multiple entities and trigger the download of data adhoc.