

Databases, Data Widgets and Forms

- [Basic Database Concepts](#)
 - [What is a Database?](#)
 - [Primary Keys and Foreign Keys](#)
 - [One-to-Many Relationships and One-to-One Relationships](#)
 - [Database Views and Queries](#)
- [Importing Data Models Into WaveMaker](#)
 - [Import Sample Data Model](#)
 - [Install MySQL and Import Data Model](#)
 - [Importing Existing Data Model](#)
 - [Basic Settings For Database Import](#)
 - [Advanced Options](#)
- [Creating New Data Models in WaveMaker](#)
 - [Changing Database Connection Settings](#)
 - [Exporting a Data Model](#)
 - [Deleting a DataModel](#)
- [Changing Database Schema](#)
 - [Re-Importing a Database](#)
 - [Changing the WaveMaker Data Schema Manually](#)
- [Editing Tables In WaveMaker](#)
 - [Table Editor: General Properties](#)
 - [Table Editor: Editing Columns](#)
 - [Primary Key Generation](#)
 - [Relationships and Foreign Keys](#)
- [Views In WaveMaker](#)
 - [Creating a View](#)
 - [Using a View](#)
- [Queries in WaveMaker](#)
 - [Defining a Query](#)
 - [Using List Inputs](#)
 - [Testing the Query](#)
 - [Using a Query](#)
- [Forms In WaveMaker](#)
 - [Using Data Form Widgets](#)
 - [Using Live Form Widgets](#)
 - [Using the EditPanel Widget to Create, Update and Delete Data](#)
- [Forms and Relationships](#)
 - [Using Related Editor Widgets for One-To-One Relationships](#)
 - [Using Related Editor Widgets for One-To-Many Relationships](#)
 - [Using Data Navigator Widgets in Forms](#)
- [Setting Form Primary and Foreign Keys Automatically](#)
 - [Insert operation clears all field values](#)
 - [Multi-column primary keys](#)
 - [Adding related editor to set foreign key values](#)
 - [Setting lookup editor for foreign key](#)
- [See Also](#)

In order to access a database from your WaveMaker Studio application, you must first create a new data model or import an existing data model into the project for that application. When you import the database, you specify database connection information and then WaveMaker Studio creates a Data Model for you. Both the database connection and the Data Model itself are editable after the import.

WaveMaker Studio does not automatically synchronize the Data Model with the database on which it is based. If the database changes (for example, if you add a new table to the database) you must re-import the database. Similarly, if you make changes to the Data Model and you want your database to include those changes, then you must re-export the Data Model back to the database.

Basic Database Concepts

WaveMaker makes it easy to create web-based forms that are connected to an underlying database.

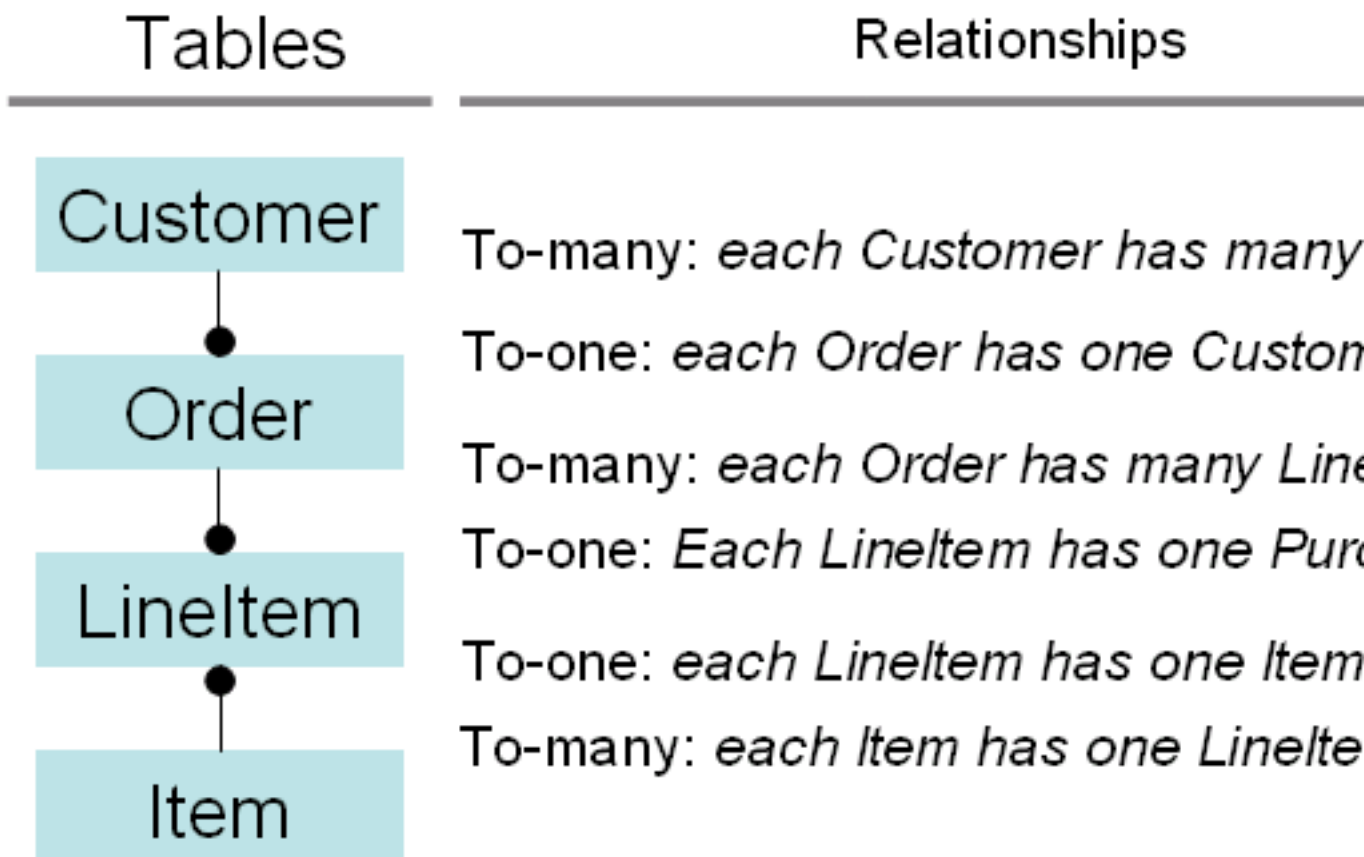
What is a Database?

A database stores information used in an application as a set of tables. Each table is similar to Excel spreadsheet and consists of rows and columns.

Create a new data model or import an existing data model in WaveMaker by selecting the Database Service option from the Services menu. This brings up the New Data Model window, which gives a choice of creating a new data model or importing an existing data model.

Sample Data Model: Customers and Orders

The Customer/Order data model will be used throughout this section to illustrate basic database concepts. The following example shows two tables, customers and orders:



Database Field

A field in a database is the same as a column in Excel. When the data model is created, each field is defined to contain a certain kind of data, such as characters, integers or dates. In addition a field may be allowed to be

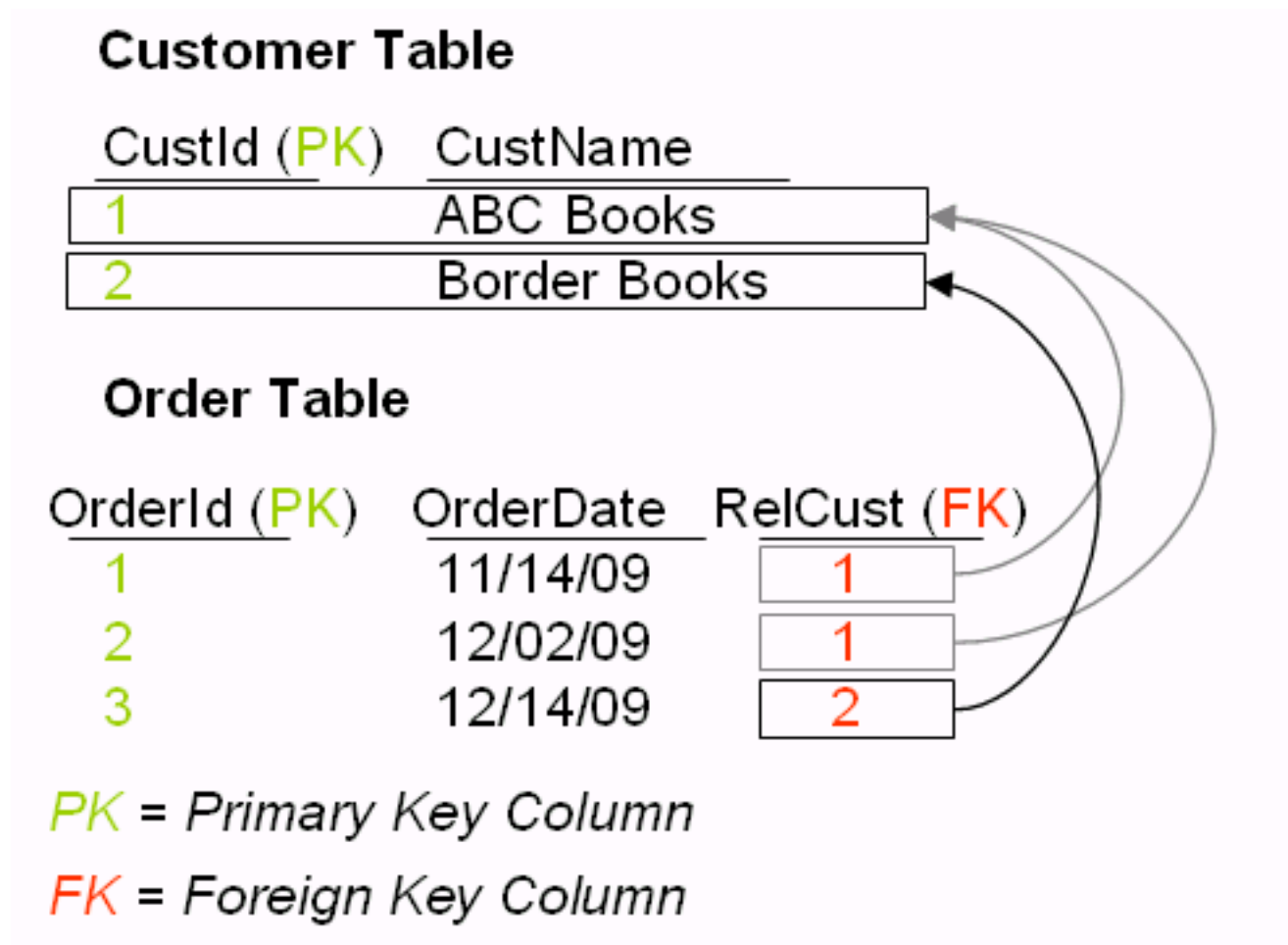
blank (NULL allowed) or must always have a value (NULL not allowed). Finally, a field may be defined to have a default value.

To create a new table in WaveMaker, select the Database icon in the Model palette and then press the green plus icon, then enter a name for the new table.

Each table that you import from the database is represented in the Data Model as a *LiveTable*. You can edit LiveTables in the LiveTables editor ([LiveTables ?](#)). If you choose, you can export your changes back to the database. If the database changes (for example, a new table is added) then you need to re-import ([Importing a Database ?](#)).

Primary Keys and Foreign Keys

In this Customer/Order data model, each Customer is identified by a unique CustId and each Order is identified by a unique OrderId. These unique identifiers are called Primary Keys.



In WaveMaker, to identify a column as being part of the primary key, select the Primary Key check box for that column.

Every table must have a primary key and each primary key must be unique. One examples of primary keys from the real world include using a social security numbers to uniquely identify a person.

Because primary keys are so common, data models often use primary key values that are automatically generated by the database.

Foreign Keys

Looking at the Customer/Order data model, notice that each order is related to a corresponding customer by the column, RelCust. For example, the first and second Orders were both placed by Customer with CustId = 1, and the third order was placed by Customer 2.

Columns that indicate a relationship between two tables are called foreign keys. The RelCust column of the Order table is a foreign key to the CustId column of the Customer table.

To create a foreign key in WaveMaker, add the appropriate foreign key column to a table, then select the foreign key column and press the plus icon near the relationships editor.

One-to-Many Relationships and One-to-One Relationships

In the Customer/Order data model, the customer table has a one-to-many relationship with the order table. This means that for each customer, there can be many related orders.

In the Customer/Order data model, the order table has a one-to-one relationship with the customer table. This means that for each order, there is exactly one related customer.

WaveMaker automatically understands data model relationship, making it easy to create forms that show related data. For example, dragging a relatedEditor widget into a Customer LiveForm makes it easy to display a grid of related orders for a particular customer. Similarly, dragging a relatedEditor widget into an Orders LiveForm make it easy to display editable customer data along with an order form.

Database Views and Queries

A view can join information from several tables together, for example adding the CustomerName field to the the Order information.

Create views in WaveMaker by selecting the LiveView option from the Services menu. LiveViews are collections of related tables. They are similar to views in databases. Use LiveViews when you want to display columns from related tables in a single widget, such as a LiveForm. LiveViews cannot be exported back to the database. ([LiveViews ?](#))

A query returns information from a table or set of tables that matches particular criteria. For example, a query could return all Orders made before 12/10/09. Queries can also join information from several tables together (these are called Join queries).

Create queries in WaveMaker by selecting the Query option from the Services menu. In WaveMaker Studio database queries are treated as objects in the Data Model. You can create, edit and delete queries in the Queries editor. ([Queries ?](#))

Importing Data Models Into WaveMaker

The fastest way to start working with WaveMaker is to import an existing data model into WaveMaker. Once the data model is imported, WaveMaker makes it very easy to create forms based on this data model to create, read, update and delete information from the database.

This section provides three approaches to importing a data model into WaveMaker:

- Import one of the sample data models that ship with WaveMaker
- Install the open source MySQL database, create a sample data model and then import it into WaveMaker
- Import an existing data model from another database

Import Sample Data Model

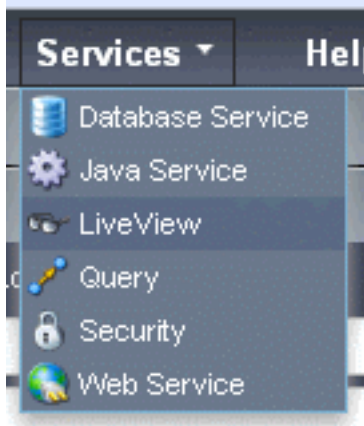
The easiest way to get started with WaveMaker is to use the bundled HSQLDB database and one of the sample data models: cmdb, hrdb and userdb.

HSQLDB is a single user database that is included primarily for demo and testing purposes. [For more information on HSQLDB, click here](#)

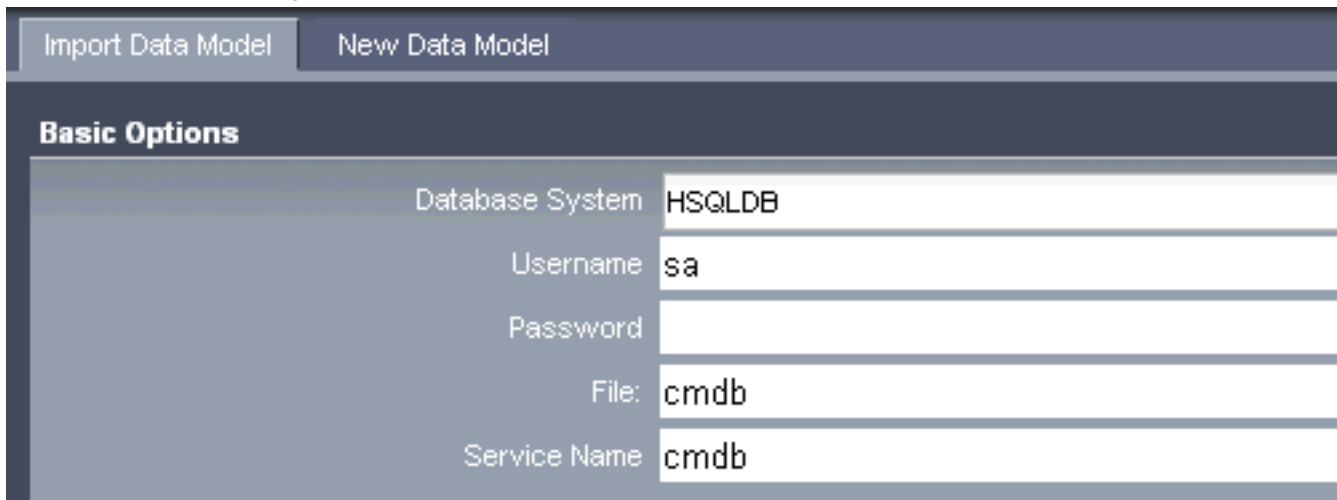
The benefit of HSQLDB is that it stores its data as part of a WaveMaker project. Using HSQLDB for a project makes it very easy to export that project to a zip file, then send that zip file to someone else. All the recipient has to do is unzip the project and they will automatically have all the data they need to run the project.

To implement the sample cmdb database into a project, follow these steps:

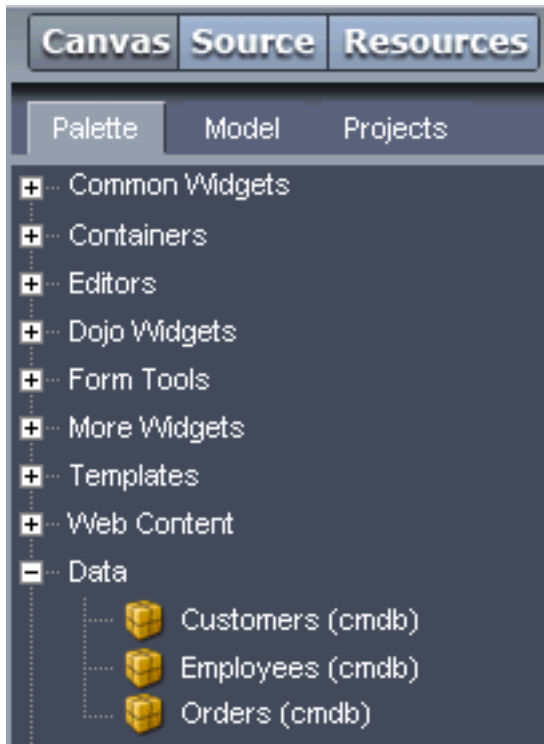
1. Create a new WaveMaker project
2. Choose the **Database Service** option in the **Services** menu



3. The New Data Model dialog appears with the Import Data Model tab selected.



4. Set the Database System parameter to "HSQLDB"
5. Set the File parameter to "cmdb"
6. Leave the username = "sa" and the password blank
7. Press the import button to import the data model
8. Go to the palette and look under the Data section to see the Data Form Widgets that were created as part of the database import process.



9. Drag the Customers widgets onto the canvas to see a master-detail form: a datagrid listing all the Customers in the database and associated form that shows detailed information for the currently selected Customer as well as the ability to create a new customer, update a customer or delete a customer.

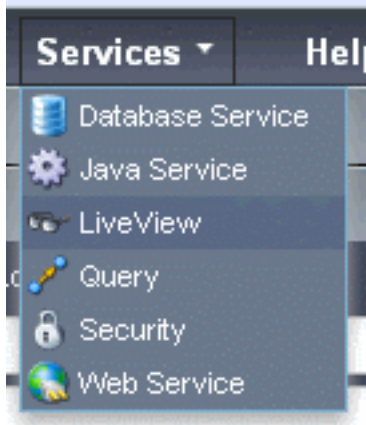
Note: HSQLDB is meant primarily for development and testing purposes. It is not appropriate for multi-user or high transaction web applications.

Install MySQL and Import Data Model

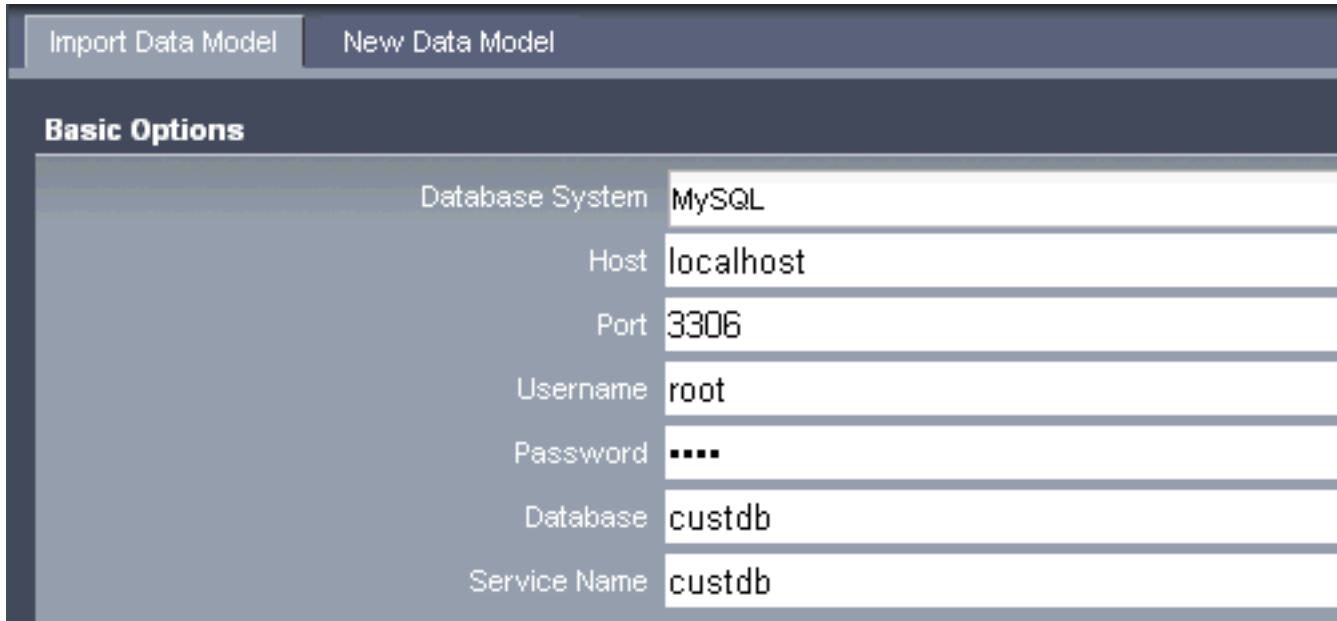
For novice developers who want a more scalable database than HSQLDB, MySQL is a good choice.

Follow these steps to start using MySQL with WaveMaker:

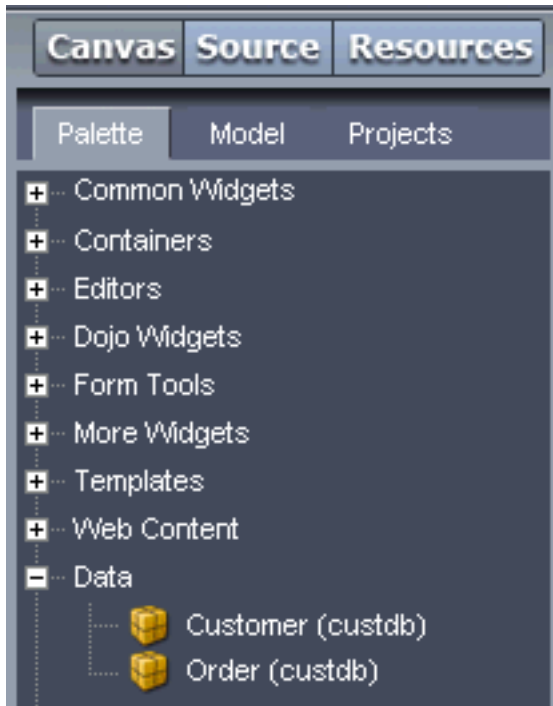
1. [Download the MySQL database here](#) , choose the latest generally available (GA) release.
2. Install the MySQL database on your computer
3. The default login for the MySQL is username = root, password = root
4. Run the MySQL Command Line Client application (this is included as part of the MySQL installation). At the Enter Password prompt, enter "root"
5. Cut and paste the Sample MySQL Database DDL from below into the MySQL Command Line Client (right click in the MySQL Command Line Client window, then select the paste option). Press return. MySQL should respond with the following message: "Query OK, 3 rows affected"
6. Now start WaveMaker and create a new project.
7. Choose the **Database Service** option in the **Services** menu



8. The New Data Model dialog appears with the Import Data Model tab selected.



9. Set the Database System parameter to "MySQL"
10. Set the Username parameter to "root" and Password parameter to "root"
11. Set the Database parameter to "custdb"
12. Press the import button to import the data model
13. Go to the palette and look under the Data section to see the Data Form Widgets that were created as part of the database import process.



14. Drag the Customer widgets onto the canvas to see a master-detail form: a datagrid listing all the Customers in the database and associated form that shows detailed information for the currently selected Customer as well as the ability to create a new customer, update a customer or delete a customer.

Sample MySQL Database DDL Copy and paste this code into the MySQL Command Line Client to create a sample database called "custdb".

```
create database

if not exists `custdb`;
USE `custdb`;
/*Table structure

for table `customer` */
DROP TABLE IF EXISTS `customer`;
CREATE TABLE `customer` (
  `CustId`

int(11) NOT NULL auto_increment,
  `CustName` varchar(30)

default NULL,
  PRIMARY KEY (`CustId`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
/*Data

for the table `customer` */
insert into `customer`(`CustId`,`CustName`)
values (1,'ABC Books'),(2,'Border Books');
/*Table structure

for table `order` */
DROP TABLE IF EXISTS `order`;
CREATE TABLE `order` (
  `OrderId`

int(11) NOT NULL auto_increment,
  `OrderDate` date NOT NULL,
  `RelCust`

int(11)

default NULL,
  PRIMARY KEY (`OrderId`),
  KEY `FK_order` (`RelCust`),
  CONSTRAINT `FK_order` FOREIGN KEY (`RelCust`)
REFERENCES `customer` (`CustId`)
```

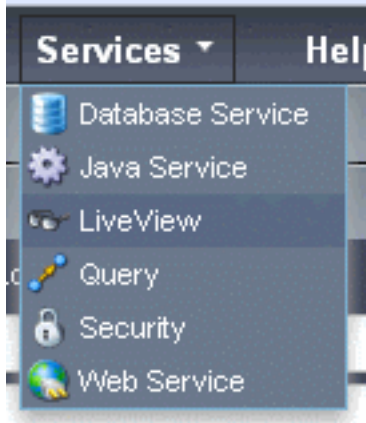
```
ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
/*Data

for the table `order` */
insert into `order`(`OrderId`,`OrderDate`,`RelCust`)
values (1,'2009-11-14',1),(2,'2009-12-02',1),(3,'2009-12-14',2);
/* End of DDL */
```

Importing Existing Data Model

To use a database in WaveMaker, you first need to import that data model for that database into the WaveMaker project. To import a database, follow these steps:

1. Choose the **Database Service** option in the **Services** menu



2. The New Data Model dialog appears with the Import Data Model tab selected.

New Data Model

Import Data Model New Data Model

Basic Options

Database System	MySQL
Host	localhost
Port	3306
Username	root
Password
Database	custpurchase
Service Name	custpurchase

Advanced Options

Driver Class

Dialect

Naming Strategy

Test Connection

Alert http://localhost:8094/ X

Connection successful

3. In the "Basic Options" section, select the Database System property from a list of options, such as MySQL and PostgreSQL.
4. If the database is located on your computer, leave the Host property as "localhost". Otherwise, enter the hostname for the computer running the database.
5. Do not change the Port number unless you are a database expert - the port number is usually set automatically when you select the Database System property.
6. Enter the Username and Password properties to login to your database
7. Enter the Database name - this is the name of the data model defined in your database
8. In most cases, you should leave the Service Name property the same as the Database name
9. Press the **Test Connection** button to test whether the parameters were entered correctly.
10. Press the **Import** button to import the data schema. When the import is complete, the Data Model Editor shows you the tables defined in the schema that you have just created.
11. Save the project.

Note: In most cases you will not need to worry about Advanced Settings. If you need them, see [Advanced Options](#).

When you import the database, WaveMaker Studio creates a Data Model for you ([What is in the Data Model](#)). Once the database has been imported, here are some things that you can do within WaveMaker:

- **Use Data Form Widgets:** for every table in the imported data model, there is a corresponding widget added to the Data section of the palette. Drag this widget onto the WaveMaker canvas to create custom grids and forms to create, read, update and delete database information.
- **Create data model views:** use the WaveMaker View Editor to combine information from several related tables into a data view that can be displayed in datagrids and forms.
- **Create queries:** use the WaveMaker Query Editor to build custom database queries for custom reports and dashboards.
- **Create LiveData variables:** LiveData variables hold data from views and queries so that they can be connected to datagrids and forms.

WaveMaker Studio does not automatically synchronize the Data Model and the database. If the database changes (for example, a new table is added) then you need to re-import ([Re-Importing a Database](#)). Similarly, if you make changes to the Data Model that you want to see reflected in the database, then you need to export the Data Model ([Exporting a Database](#)).

Basic Settings For Database Import

The "Settings" area of the Import Database screen allows you to choose what type of database you want to import and configure the basic required settings.

The most important properties

- **Host:** The network host for the database. The default value is localhost, meaning that the database is located on the same computer that WaveMaker studio is running on.
- **Port:** The port number for the database. Typically this is set automatically when the database type is selected and should not be changed.
- **Database:** The name of the database. Within a database system, a database name is similar to a directory name in the operating system - it is a way of organizing a set of tables.

When you choose your database type the Database drop-down menu, the configuration for that type of database appear.

Some of the configuration fields are different for the different database types. Here are the specific differences:

1. **HSQLDB:** HSQLDB is a file-based database. WaveMaker expects the HSQLDB file to be located in the project directory `./webapproot~data~` with the name `"dbname.script"`
 - File property: enter the name of the HSQLDB file located in the `./webapproot~data~` directory. If the name of the file is `foo.script`, just enter the file name `"foo"`
2. **SQL Server:** SQL Server has a special property to specify the database instance. Normally, this should be set to the value `"SQLExpress"` if you are using SQL Server Express Edition.
 - Instance name property: by default, this should be set to `"SQLExpress"`
3. **Oracle:** Oracle has a System ID property that is one of the connection parameters. By default, the SID and the database name are the same.
 - SID: set this to the System ID for the Oracle database. By default, this is the same as the database name.

Advanced Options

For advanced users, WaveMaker Studio provides some advanced configuration options. The following options are available:

- **Connection URL:** When you import a database using the standard configuration options on the Import Database screen, WaveMaker Studio creates a JDBC URL for you. This URL is shown here in the Connection URL field. If you know what you are doing and want to change this JDBC URL, you can do it here.

- **Java Package:** WaveMaker Studio generates Java classes for you when you import a database. If you want a different Java package name than the default name we generate, type it in here.
- **Table Filter:** By default WaveMaker Studio imports all the tables in the database. If you want to import only a subset of the tables, type in a comma-delimited list of regular expressions here.
- **Schema Filter:** When you are importing a database that supports schemas, WaveMaker Studio imports only the tables for the default schema. If you want to import tables from other schemas, add the schema names in this field.
- **Driver Class:** JDBC driver class name. WaveMaker Studio creates a JAR file for each database that uses a specified driver. If you want to use a different JAR file for connecting, specify it here (the JAR file must be in the Application Server's class path).

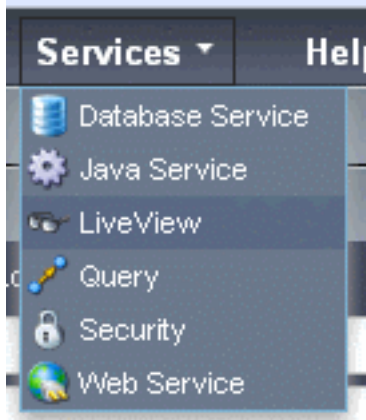
Creating New Data Models in WaveMaker

WaveMaker studio can be used to create a new data model. **Note:** this is an advanced feature. For novice users, it is best to create data models directly in the database using tools like [SQLYog](#) and then import the data model into WaveMaker.

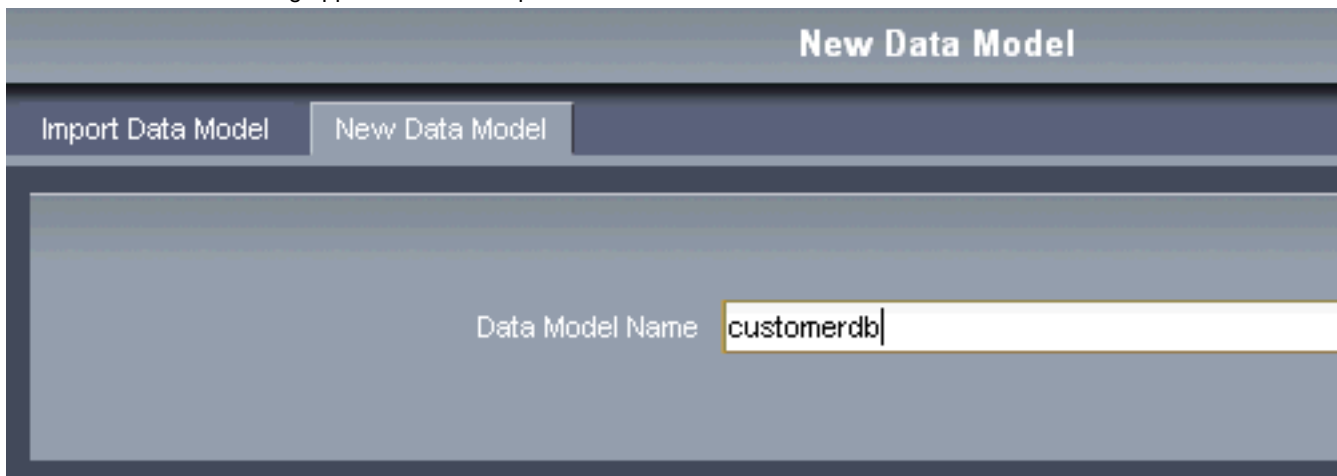
Create a new data model in WaveMaker by selecting the Database Service option from the Services menu. This brings up the New Data Model window, which gives a choice of creating a new data model or importing an existing data model.

In creating a new data model, the steps are:

1. Choose the **Database Service** option in the **Services** menu



2. The New Data Model dialog appears with the Import Data Model tab selected.



3. **Create tables:** for each table add columns and identify the primary key
4. **Create relationships:** for each relationship, add foreign keys and define the relationship
5. **Save the data model:** press the diskette icon from the data model editor
6. **Export the data model:** press the wrench icon from the data model editor to bring up the Database Connection Settings window and press the "Export" button.



Changing Database Connection Settings

Whenever you import a new WaveMaker project, such as a sample application, it is important to update the database connection settings to work with your local database. To do this, follow these steps:

1. Select the database to export from the Model palette. Next, select the wrench icon to bring up the Database Connection Settings window.



2. Set the Database Connection properties required for your database, such as username and password.
3. Press the "Test Connection" button at the bottom of the dialog to test that the database connection properties are valid.
4. Press the "Save" button to save the new connection settings

Exporting a Data Model

Note: exporting a database deletes any existing data in the database tables. Do not use WaveMaker to update the data model if you want to preserve the data in the tables.

The Table Editor allows you to change table definitions within the WaveMaker project. To export these changes to the database, follow these steps:

1. Select the database to export from the Model palette. Next, select the wrench icon to bring up the Database Connection Settings window.



2. Set the Database Connection properties required for your database, such as username and password.
3. Press the "Test Connection" button at the bottom of the dialog to test that the database connection properties are valid.
4. Click the Export button at the bottom of the dialog. An export confirmation dialog appears that displays the SQL syntax for this export.
5. Click **OK**.
6. Any errors encountered in creating the database will be reported.
7. Click **Close** in the Database Connections dialog.

Exporting to MySQL

When exporting to MySQL, the Database property in the Database Connections dialog translates to the MySQL schema name. Enter a name for the database that will contain all the tables defined in the schema.

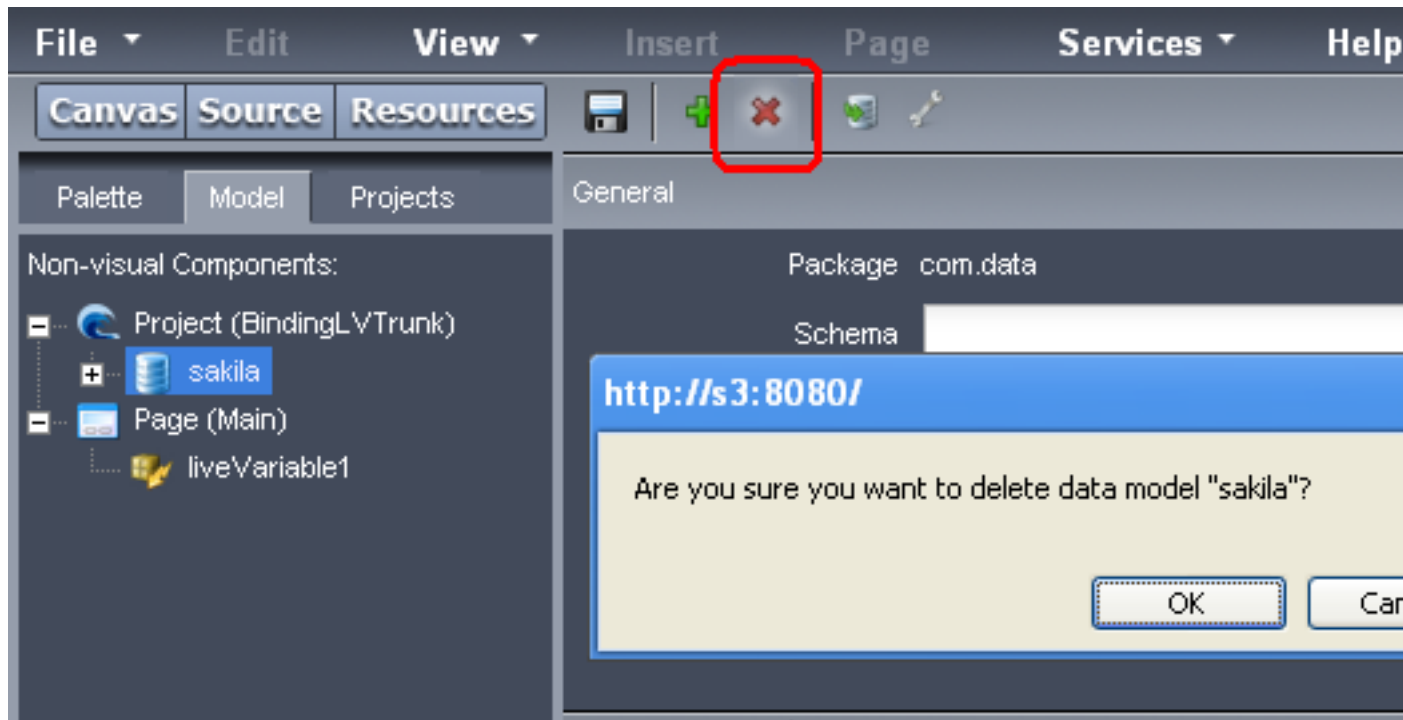
Exporting to Postgres

When exporting to Postgres, the Database property in the Database Connections dialog translates to the Postgres schema. A good default choice is to set the Database property to "public".

Deleting a DataModel

To delete a data model from a project:

1. Select the data model to be deleted
2. Use the delete button in the toolbar



Changing Database Schema

It is common to change the underlying database schema during a project. There are two approaches to handle data schema changes: re-importing the data model and manually changing the schema in WaveMaker.

Re-Importing a Database

The recommended approach for managing data model changes with WaveMaker is to use a database management tool such as PHPMyAdmin or SQLYog.

If a data model changes after it has been imported, then it should be re-imported. Re-importing a data model updates all of the table definitions in the project. Existing views are not changed, however if they reference tables or columns that have changed, you might get errors at runtime.

Note that WaveMaker does not support re-importing a data model once you have defined custom HQL queries. This is a limitation of the Hibernate DDL import utility. The work-around is to copy your queries to a text editor and then delete them, re-import the data schema and re-create the queries.

To re-import, follow these steps:

1. Make changes to the database schema using your choice of database administration tool
2. If you have defined custom HQL queries, copy them into a text editor and delete them from the project (the Hibernate re-import facility will not work if you have queries defined)
3. Select the database to re-import from the Model palette. Next, select the wrench icon to bring up the Database Connection Settings window.



4. Set the Database Connection properties required for your database, such as username and password.
5. Press the Test button at the bottom of the dialog to test that the database connection properties are valid.
6. Click the Re-import button at the bottom of the dialog.
7. Click the "OK" button.
8. Recreate any custom HQL queries needed

Changing the WaveMaker Data Schema Manually

It is often easier to manage data model changes by applying the changes twice: first using a database administration tool and next making the same changes within WaveMaker using the data model editor.

1. Make the db schema changes using a db tool like SQLYog or PHPMYAdmin
2. Now open the WaveMaker project and use WaveMaker's data model editor to make the same changes to each table. For relationship constraints, make the changes to the relationship definitions to the table which contains the foreign key - Wavemaker will automatically update the other side of the relationship.
3. When you are done making the changes to the table, press the save button to save changes for that table. This will automatically regenerate the required Hibernate files.

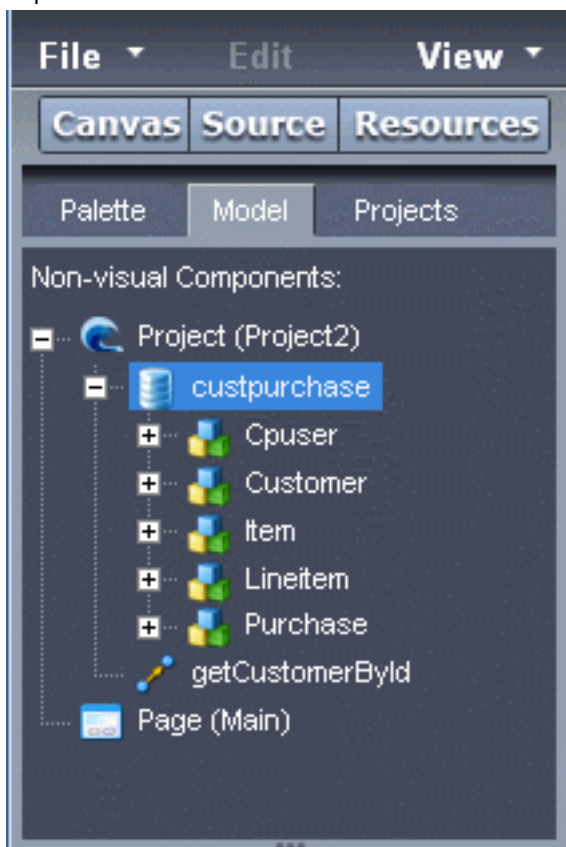
Editing Tables In WaveMaker

The Table Editor in WaveMaker can be used to customize a data model in WaveMaker or to create an entirely new database schema.

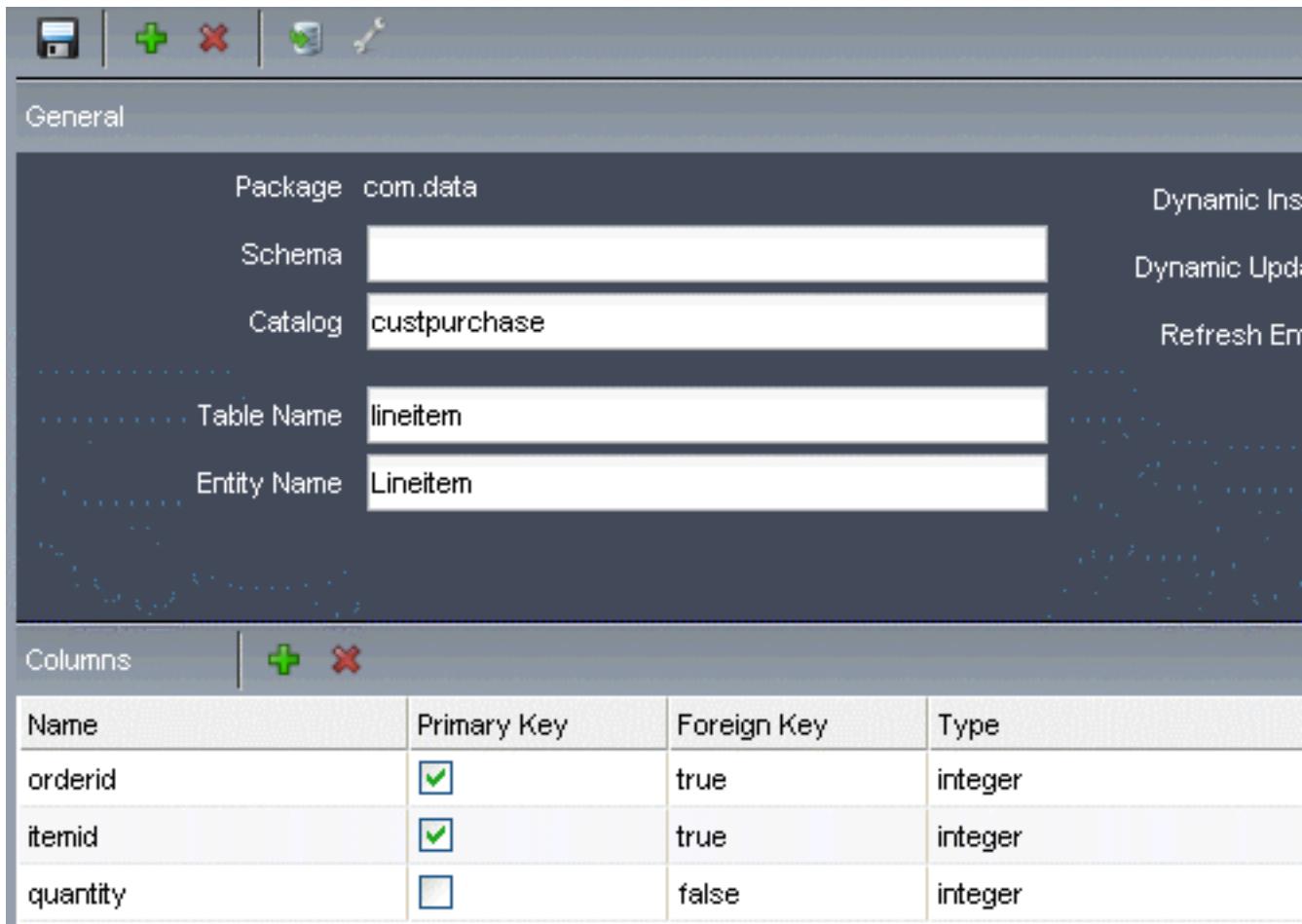
- **Customizing an existing data model:** the data model editor can be used to create more understandable aliases for legacy table and column names (not that these changes are overwritten when the database is re-imported, so Views maybe a better solution).
- **Creating a new data model:** the data model editor can create new tables, columns and relationships. The new data model must be saved and then exported for the changes to take effect in the database.

When a database is imported into a project, WaveMaker Studio creates a Data Model tree that represents that database. To edit a table in the Data Model, follow these steps:

1. Expand a table in the tree to see the columns for that table.



2. Select a table from the model Data Model tree brings up the table editor.



The right side of the Table editor has three main areas: "General", "Columns" and "Relationships". Each area allows the developer to view and edit a different aspect of the Table.

Table Editor: General Properties

The "General" section of the WaveMaker Table Editor allows you to view and configure main properties of the LiveTable. In general, only the Entity Name property should be changed.

The four fields on the left of the screen are:

- **Package:** This shows the java package name for the Data Model and cannot be changed.
- **Schema:** this is the database schema property from the Database Connection Settings.
- **Catalog:** this is the database name property from the Database Connection Settings.
- **Table Name:** The value in this field is the name of the table in the database that this LiveTable references.
- **Entity Name:** The value in this field is the name of the Table in WaveMaker. By default it is set to the same as the table name in the database but it can be changed to create an more "user friendly" name when working with legacy data schemas. Note that changes made to the Entity Name will be overwritten when you re-import the data schema, so it is often better to use Views to create project aliases for tables and column names.

The three check boxes on the right of the screen are all for retrieving values that were set by the database.

- **Dynamic Insert:** when the Dynamic Insert parameter is checked, the INSERT SQL statement that WaveMaker sends to the the database does not include empty fields.
- **Dynamic Update:** when Dynamic Update parameter is checked, the UPDATE SQL statement that WaveMaker sends to the database does not include empty fields.

- **Refresh Entity:** if checked, a SQL select statement is run to fetch the row just inserted or updated to include any fields that are set by the underlying database (this is useful for TIMESTAMP columns set by the database)

Note: Dynamic Insert and Dynamic Update map directly to the Hibernate settings dynamic-insert and dynamic-update.

Table Editor: Editing Columns

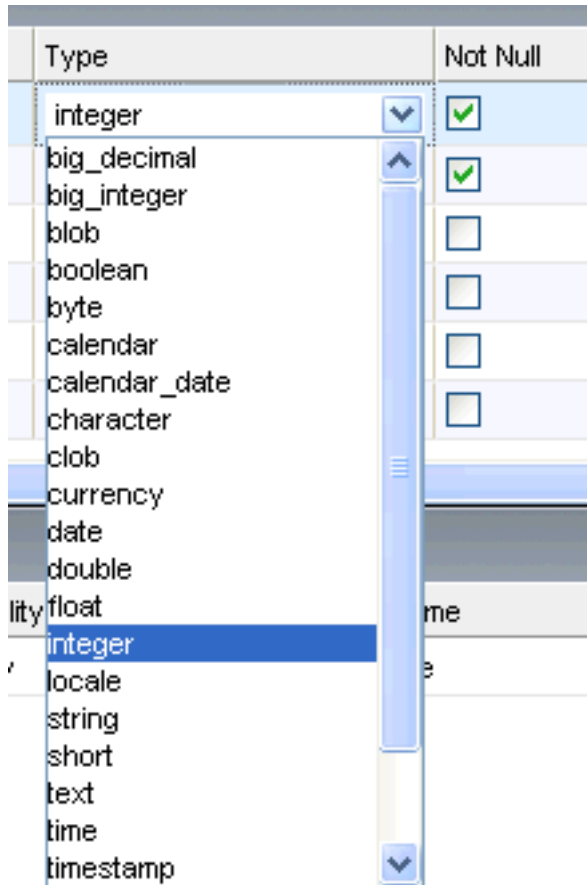
The "Columns" section of the WaveMaker Table Editor shows all the columns in the Table and the properties for each column. Add and delete columns using the add (+) and delete (-) buttons.

Changes made in this editor are not automatically made in the database that it represents. In order to change your database to reflect these changes, first save the changes by pressing the diskette icon, then export the data model Data Model ([Exporting a Data Model](#)).

The following table explains the information that is displayed about each column. You can change any of these values and save your changes to the Data Model.

Type	Description
Primary Key	A check box. Checked if the column is a primary key.
Foreign Key	True or False value. True if the column is a foreign key, false if not. This column is set automatically when a relationship is defined.
Type	Data type of the column.
Not Null	A check box. Checked if the column can NOT hold a null value. If unchecked, the column value can be null.
Length	Maximum column length - used primarily for string values.
Precision	Refers to the maximum number of digits used by the selected (numeric) data type and the digits past the decimal point. For example, a precision of "10,2" specifies a data type with a maximum of 10 digits, 2 of which are to the right of the decimal point.
Generator	If the column is a primary key, specifies the method of key generation (Key Generation).
Params	If a generator is selected, specifies the parameters for that generator, if any.

WaveMaker column data types



Note: for multi-column primary keys, it is sometimes useful to uncheck the primary key parameter for all but one of the primary key columns. This enables the table to be used in a LiveForm and also allows access to primary key columns that would otherwise not be accessible (hibernate does not allow access to composite primary key values).

Primary Key Generation

For primary keys, it is common to use the database itself to generate unique key values.

- **For primary keys where the user will input a value:** set the generator parameter to "assigned." For example, this would work well if the primary key for a table were a user assigned value like social security number.
- **For primary keys where the database will automatically assign a value:** set the generator field to "identity." This is the default option for new tables and ensures that the database will take care of making each row have a unique identifier.



Here is a detailed description of the Generator parameter options:



Generator	Description
assigned	lets the application assign an identifier to the object before save() is called. This is the default strategy if no element is specified.
identity	supports identity columns in DB2, MySQL, MS SQL Server, Sybase and HypersonicSQL. The returned identifier is of type long, short or int.
sequence	uses a sequence in DB2, PostgreSQL, Oracle, SAP DB, McKoi or agenerator in Interbase. The returned identifier is of type long, short or int
native	picks identity, sequence or hilo depending upon the capabilities of the underlying database

If your Generator type requires an input parameter, enter it in the **Params** column next to the **Generator** column.

Relationships and Foreign Keys

Relationships provide a way to connect data in one table with data in a nother table. The WaveMaker Table Editor shows the relationships for the selected Table. You can add and delete relationships here.

Columns  				
Name	Primary Key	Foreign Key	Type	Not
orderid	<input checked="" type="checkbox"/>	true	integer	<input checked="" type="checkbox"/>
itemid	<input checked="" type="checkbox"/>	true	integer	<input checked="" type="checkbox"/>
quantity	<input type="checkbox"/>	false	integer	<input checked="" type="checkbox"/>

Relationships  			
Name	Related Type	Cardinality	Table Name
purchase	Purchase	to-one	lineitem
item	Item	to-one	lineitem

Changes you make in this editor are not automatically made in the database that it represents. In order to change your database to reflect these changes, you need to export your Data Model ([Exporting a Data Model](#)).

To add a relationship, first create the two tables that will be related. Next determine which table will contain the foreign key.

If two tables are related, such as Customers and Orders, the foreign key will need to go in the table that is on the "to-one" side of a relationship. Each Customer can have many related Orders, but each Order can have only one related Customer. Thus the foreign key should be added to the Order table and must have the same data type as the primary key of the Customer table.

The steps to create a relationship follow:

1. Create the two related tables, each with a primary key
2. In the table with the "to-one" side of the relationship, add a "foreign key" column of the same datatype as the primary key of the table with the "to-many" side of the relationship.
3. Now select the foreign key column and click the "+" icon next to the Relationships section of the Table Editor
4. Give the relationship a name such as relatedOrders
5. Specify the table name of the related table, such as "Customer"
6. Set the cascade actions parameter to "none" (the other options are for advanced users only).
7. Save the data model by pressing the diskette icon after editing each table(remember to export the data model when all changes are complete).

Views In WaveMaker

A View (also called a LiveView in WaveMaker) is a collection of related Tables and their columns. Use Views when to combine data from several related tables. For example, a view could combine data from the customer table and the order table to show the customer name associated with each order.

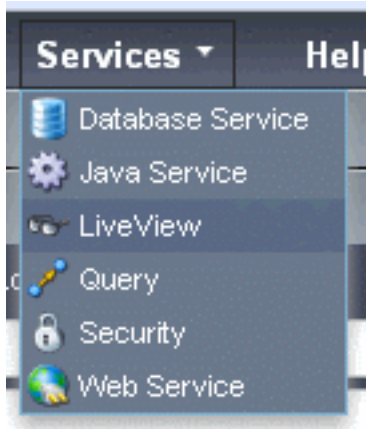
Creating a View

Create Views using the WaveMaker Views Editor. Views are implemented using WaveMaker's underlying Hibernate component - they are not exported to the database.

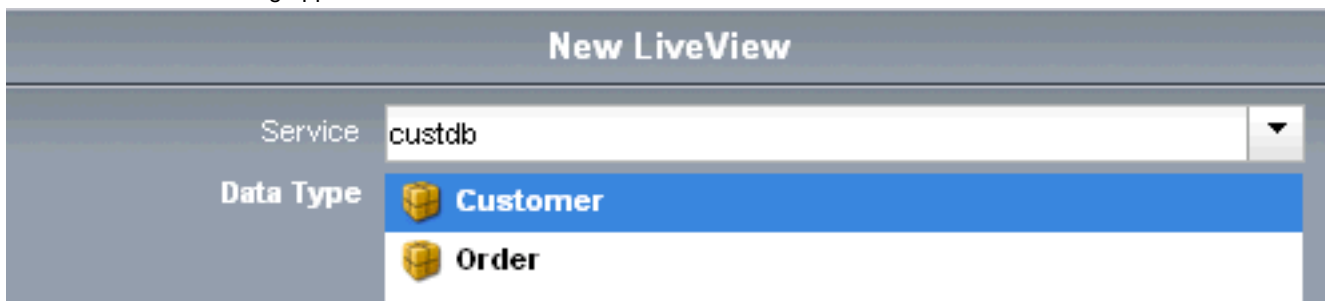
A LiveView creates a new data type that can be used as the data type for a Live Variable and then bound to a datagrid or LiveForm. Each time a Data Form object is dragged off the palette and onto the canvas, a LiveView and LiveVariable are automatically created for that Data Form object.

To create a new View, follow these steps:

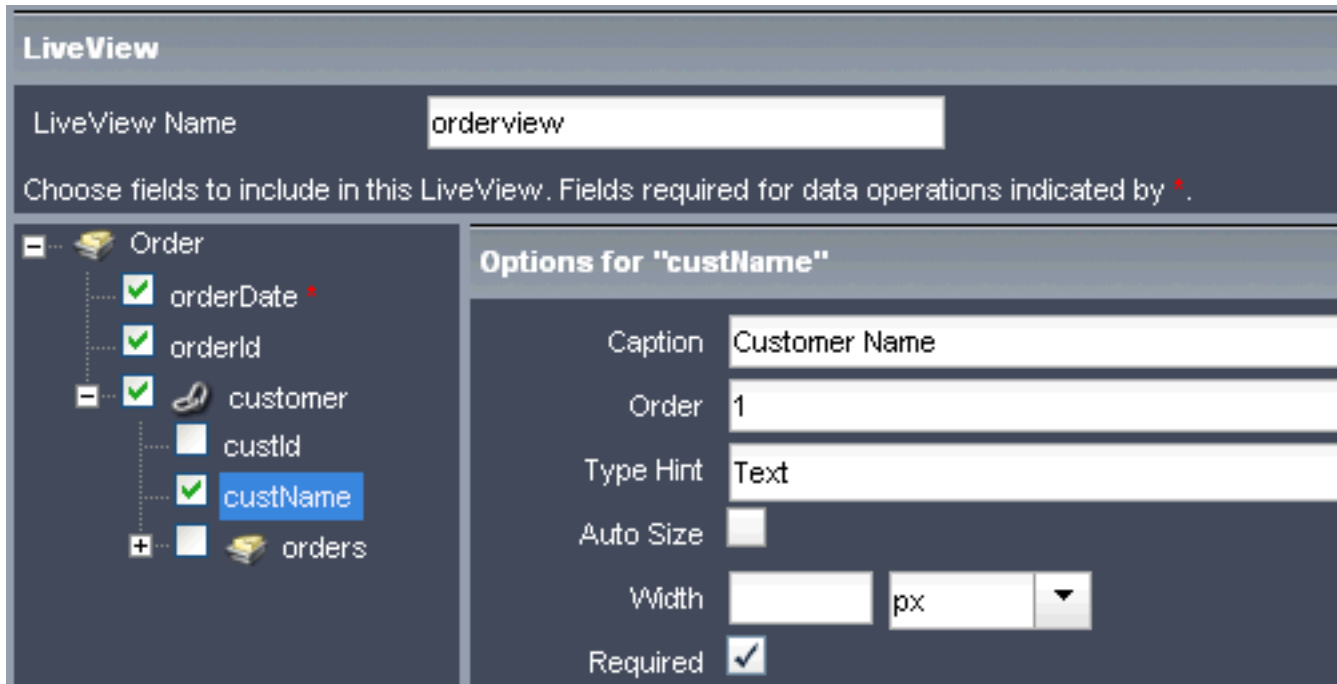
1. Choose the **LiveView** option in the **Services** menu



2. The New Live View dialog appears.



3. Select the Data Type (or Table) that will be the basis for this Live View. Always choose the Data Type which is on the to-one side of the relationship. For example, for a view that will include data from the Customer and Order tables, choose the Order table as the base table.
4. The LiveView editor will appear. Expand the tree and select that attributes to include in the view. If you plan to use the LiveView in a LiveForm widget (e.g., to create and update database information), then you should select all the required fields.



5. When you select a column, configuration options appear on the right side of the LiveViews editor. The following configuration options allow you to affect the default configuration for LiveForm and DataGrid widgets:
- **Caption:** Specify a default value for the caption property. Applies both LiveForms and DataGrids.
 - **Order:** Specify the order in which this field should appear in the LiveForm or DataGrid. For example, if you type in a one (1) then the Editor for this field would be the first one in the LiveForm. The column for this field would be the first one in the DataGrid.
 - **Type Hint:** Specify the default value of the display property for this Editor. The display property specifies the type of value you're displaying. Applies to LiveForms and DataGrids. Note that some of the types in this menu are valid only for DataGrids and others are valid only for LiveForms. If you choose a Type Hint that is not valid for the field, your Type Hint is ignored.
 - **AutoSize:** Specify that this column should be automatically sized. Applies only to DataGrids.
 - **Width :** Specify a default value for the column's width property. The width property defines the width of a column in a DataGrid. Applies only to DataGrids.
 - **Required:** Specifies that a value is required in this field in order for the LiveForm to be submitted. Applies only to Editors in LiveForms.
6. Save the LiveView by pressing the diskette icon.

The LiveView is now complete. Click the Preview checkbox to see a preview of the LiveView in a DataGrid.

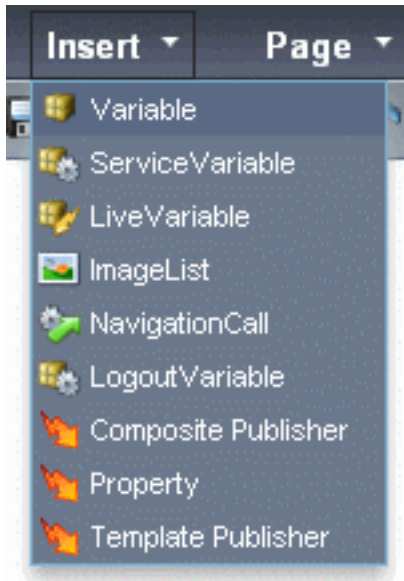
Note: LiveViews cannot be exported back to the database. They reside only in the Data Model.

Using a View

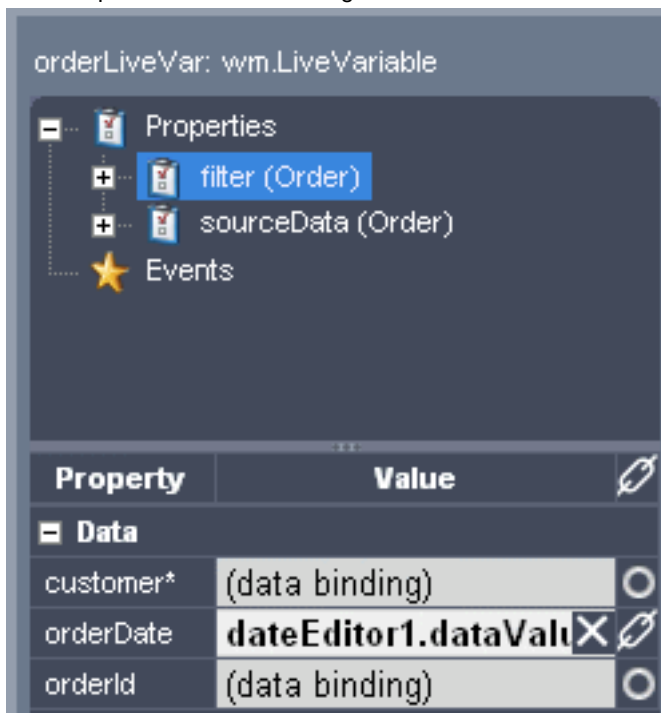
A LiveView defines a data type in WaveMaker. A LiveVariable is required to use that data type in an application either in a datagrid or a liveform.

Follow these steps to use a LiveView in an application:

1. **Define a LiveView**
2. **Create a LiveVariable:** choose the **LiveVariable** option in the **Insert** menu



3. **Set the LiveSource property:** the LiveSource is the data type for this variable, which must either be a table or LiveView. Select the appropriate data type from the drop down menu.
4. **Set filter options:** each live variable can be filtered to only show results which match a certain data value. The following screenshot shows a live variable that will filter to only show orders for a particular date based on the input to a date editor widget.



5. **Set sort options:** each live variable can be sorted. The following screenshot shows a live variable that will sort the data in ascending order for the customer.custName field.

orderLiveVar: wm.LiveVariable

Properties

- filter (Order)
- sourceData (Order)
- Events

Property	Value	
Common		
name	orderLiveVar	?
owner	Page	?
autoUpdate	<input checked="" type="checkbox"/>	?
startUpdate	<input checked="" type="checkbox"/>	?
Data		
operation	read (default)	?
liveSource	orderview (Order)	?
sourceData	(data binding)	?
filter	(data binding)	?
matchMode	start (default)	?
firstRow	0	?
maxResults	0	?
designMaxResu	50	?
orderBy	asc:customer.custName	?
ignoreCase	<input type="checkbox"/>	?
Operations		

- Show LiveVariable data in a datagrid:** drag a datagrid onto the canvas. Set its **dataSet** parameter to the name of the new LiveVariable. The datagrid will show the specified data using the captions and column ordering set in the LiveView as well as the filtering and sorting specified in the LiveVariable.
- Connect the datagrid to a LiveForm:** drag a LiveForm widget onto the canvas. Set its **dataSet** parameter to the datagrid.selected variable. Now the LiveForm will show the detailed information for the currently selected datagrid item.
- Show LiveVariable data in a LiveForm:** drag a LiveForm widget onto the canvas. Set its **dataSet** parameter to the name of the new LiveVariable. The LiveForm will show detailed information for the first record from the database view.
- Add data navigation to the LiveForm:** drag a DataNavigator widget into the LiveForm (note: the DataNavigator must be contained inside the LiveForm container to work). Now a user can navigate through a LiveView dataset record by record.

Queries in WaveMaker

WaveMaker queries provide a way to retrieve selected information from the database for reporting and dashboards. WaveMaker has a query editor that provides a visual way to create and test queries.

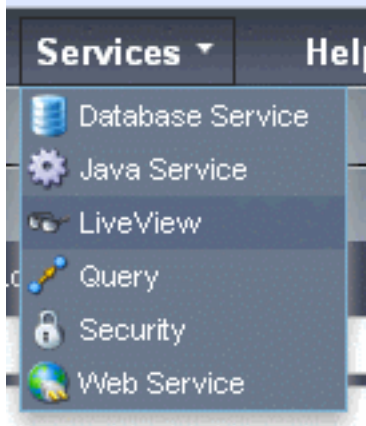
WaveMaker queries are implemented using the Hibernate Query Language (HQL), so they are stored as part of a WaveMaker project. This also means they do not need to be exported to the database.

[For an in-depth Hibernate tutorial using WaveMaker, click here](#)

Defining a Query

To create a new View, follow these steps:

1. Choose the **Query** option in the **Services** menu.



2. The Query Editor appears.

The screenshot shows a database query editor with the following sections:

- Data Model:** custdb
- Name:** getOrdersByRange
- Comment:** (empty)
- Usage Notes:**
 - 1. Remember
 - 2. After you use it
 - [Details on qu](#)
- Query Definition:**

```
SELECT customer.custName AS custname, orderDate AS orderdate
FROM Order
WHERE (orderDate >= :firstdate) AND (orderDate <= :lastdate)
```
- Returns single result:**
- Query Parameters:**

Name	Type
firstdate	Date
lastdate	Date

Add bind parameter: Name Type
- Test Query:**

Test Parameter Values: 1259625600000,1262217600000

```
ABC Books,1259740800000
Border Books,1260777600000
```

3. Enter a name for the query.
4. In the query definition, enter an HQL query.
5. In the query parameters section, enter a parameter for each named parameter in the query:
 - Enter a name which matches the parameter in the query, for example firstdate.
 - Enter a data type which matches the type for the parameter in the query, for example date
 - Press the '+' button to add the parameter to the parameter list.
 - Preface the input parameter in the query with a colon, for example :firstdate.
6. Save the query by pressing the diskette icon. A Hibernate query will now be generated and any errors displayed in a popup window.

Using List Inputs

Inputs can be a list of values. Check the list check box for the bind parameter to use this option.

For example if our select is:

```
SELECT name
FROM customer
WHERE name IN (:param)
```

Specifying the Test parameter value as [john, jane] will return all customers where the name is john OR jane.

Testing the Query

To test a query, follow these steps:

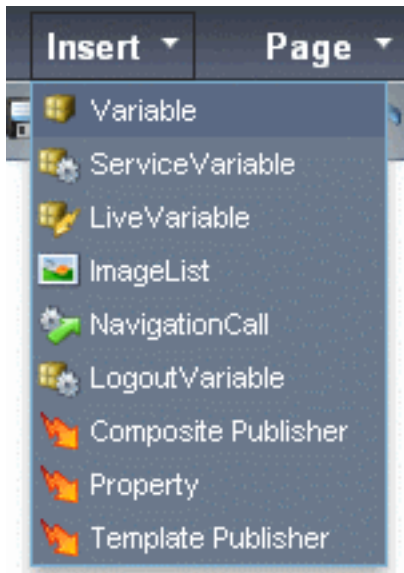
1. Create and save a query
2. From the Query editor, enter values for each query parameter
3. In the screenshot example, two parameters are required: firstdate and lastdate. Note that date parameters must be entered using a unix time format with three added trailing zeros. [Convert a date to a unix time here](#) then add three trailing zeros to the resulting number. This is only required by the query test editor, not in the application itself!
4. Press the lightning bolt icon to run the query and see the results, which are shown in the screenshot above.

Using a Query

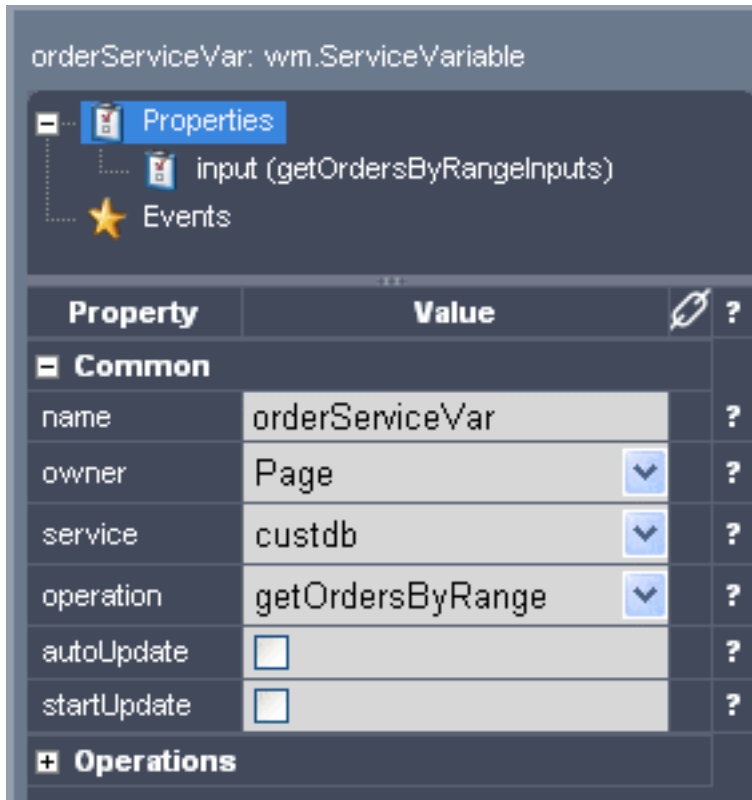
A Query defines a data service in WaveMaker. A Service Variable is required to use that query in an application either in a datagrid or a chart widget.

Follow these steps to use a Query in an application:

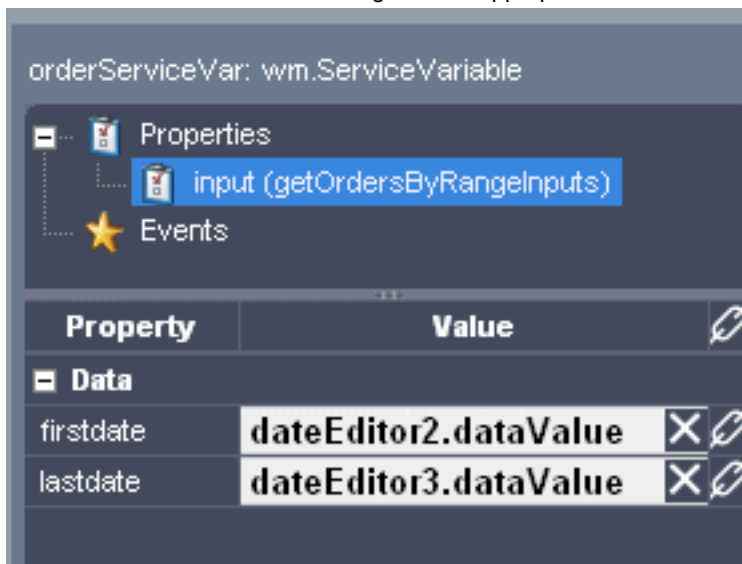
1. **Define a Query**
2. **Create a Service Variable:** choose the **ServiceVariable** option in the **Insert** menu



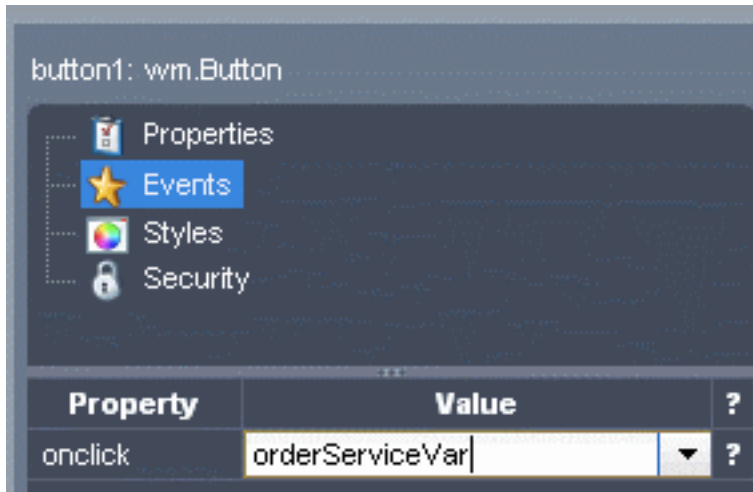
3. **Set the Service property:** the Service is the data service for this variable, which is the name of the imported data service.



4. **Set the Operation property:** the operation drop down shows the queries that have been defined for a particular data service. In this case, select the operation named getOrdersByRange.
5. **Set the input parameters:** for each input parameter in the query, specify where that input will come from. In the screenshot example, the two parameters required for the getOrdersByRange query are coming from two date editor widgets: dateEditor2 and dateEditor3. Select the **input** properties for the service variable and then connect or bind each widget to the appropriate service variable input.



6. **Set the startupdate checkbox:** if the startupdate property is checked, the query will automatically run when the application starts. For this query, it does not make sense to run the query before the two input parameters are entered.
7. **Set the autoupdate checkbox:** if the autoupdate property is checked, the query will automatically run whenever one of the input values changes.
8. **Add a button to call the query:** drag a button onto the canvas. Connect



9. **Show Service Variable data in a datagrid:** drag a datagrid onto the canvas. Set its **dataSet** parameter to the name of the new Service Variable.

Forms In WaveMaker

WaveMaker makes it easy to create Forms by simply dragging data elements from the palette. Once a data model has been imported, each table from that data model is available in the Data section of the WaveMaker palette. WaveMaker includes the following widgets for working with forms:

- **Data Form Widget:** these widgets are automatically created whenever a data model is imported. Each table in the data model has a corresponding Data Form widget in the Data section of the widget palette. Each Data Form is a composite widget includes a dataGrid widget that shows a master list of all the rows in a table, coupled with a LiveForm widget that shows the detailed information about the currently selected row in the dataGrid.
- **Live Form Widget:** a Live Form widget is similar to a Data Form widget but is not automatically bound to a particular table, giving the developer more control over building a form to create, read, update and delete data.
- **Related Editor Widget:** dragging a RelatedEditor widget inside a LiveForm allows the developer to display related data from both one-to-one and one-to-many relationships within the LiveForm.
- **Data Navigator Widget:** dragging a DataNavigator widget inside a LiveForm provides controls that allow the user to navigate from one data row to another within the LiveForm.

Using Data Form Widgets

These widgets are automatically created whenever a data model is imported. Each table in the data model has a corresponding Data Form widget in the Data section of the widget palette. Each Data Form is a composite widget includes a DataGrid widget that shows a master list of all the rows in a table, coupled with a LiveForm widget that shows the detailed information about the currently selected row in the DataGrid.

When a Data Form Widget is dragged onto the canvas, WaveMaker performs the following actions:

1. Create a DataGrid to display a master list of all rows from an underlying database table
2. Create a LiveForm to display a detailed form for the currently selected DataGrid row
3. Create a View data type that can specify which related data from other tables should be included in this Data Form. The developer can edit the View data type to adjust what data is available to display.
4. Create a LiveVariable to hold the data displayed in the DataGrid. The developer can edit the LiveVariable to filter and sort this data.
5. Bind the LiveVariable to the DataGrid.
6. Bind the LiveForm to the selected row in the DataGrid.

Using Live Form Widgets

The LiveForm widget makes it easy to build forms that create, update and delete data in an underlying data table.

The LiveForm widget has built-in functionality for displaying, creating, deleting or updating a record. The LiveForm widget has an associated DataSet object that represents its data source. It also has a DataOutput object, which you can use to get the data out of the LiveForm.

When you specify the DataSet (Getting the DataSet), the LiveForm widget automatically generates Editor widgets for every item in the DataSet (except lists and substructures). The binding on these Editor widgets is already done for you, so this can save you a huge amount of time.

The Editor widgets that LiveForm generates for you are ordinary Editor widgets. You can configure them as you would any other widget and you are free to delete them. However, do not move these Editor widgets outside the LiveForm that generated them.

If you use a LiveVariable for the dataSet then LiveForm can automatically generate Create/Read/Update/Delete functionality for you. A LiveVariable can be based either on a LiveTable (LiveTables) or a LiveView (LiveViews). However, for LiveForms, use a LiveVariable that references a LiveView.

Note: If a table has composite keys, the developer must edit the key in the WaveMaker data model to have only one column value. This does not affect the underlying database schema, but allows the data model to be handled properly in the Live Form.

[For more information on LiveForm widgets](#)

Using the EditPanel Widget to Create, Update and Delete Data

The EditPanel Widget is a specialized container must be placed inside a LiveForm and holds button widgets to perform create, update and delete operations.

[For more information on Edit Panel widgets](#)

Forms and Relationships

Forms can be used to display and update data from related tables. For example, the employee table may be related to the timesheet table such that each employee has many timesheets. WaveMaker uses Related Editor Widgets to create composite forms, for example showing a list of timesheets within an editor form.

For more information, see the [Tutorial on Composite Forms and Relationships](#)

Using Related Editor Widgets for One-To-One Relationships

Dragging a RelatedEditor widget inside a LiveForm allows the developer to display related data from both one-to-one and one-to-many relationships within the LiveForm. For a one-to-one relationship, the data is represented using a sub-form or lookup editor (similar to a select editor).

[For more information on Related Editor widgets](#)

Using Related Editor Widgets for One-To-Many Relationships

Dragging a RelatedEditor widget inside a LiveForm allows the developer to display related data from both one-to-one and one-to-many relationships within the LiveForm. For a one-to-many relationship, the data is represented in a grid.

[For more information on Related Editor widgets](#)

Using Data Navigator Widgets in Forms

Dragging a DataNavigator widget inside a LiveForm provides controls that allow the user to navigate from one data row to another within the LiveForm.

This is a specialized widget for use with LiveForms that allows the user to move through a list of records, similar to the record navigation feature of MS Access and FileMaker.

There are two ways to use the dataNavigator widget:

- Record navigation in a LiveForm: this allow the user to show data in a LiveForm record by record, similar to a database cursor.
- Paging in a Grid: this allows the user to page through data in a grid. This requires setting the byPage property checkbox in the dataNavigator and also setting the maxResults property of the LiveVariable bound to both the grid and the dataNavigator.

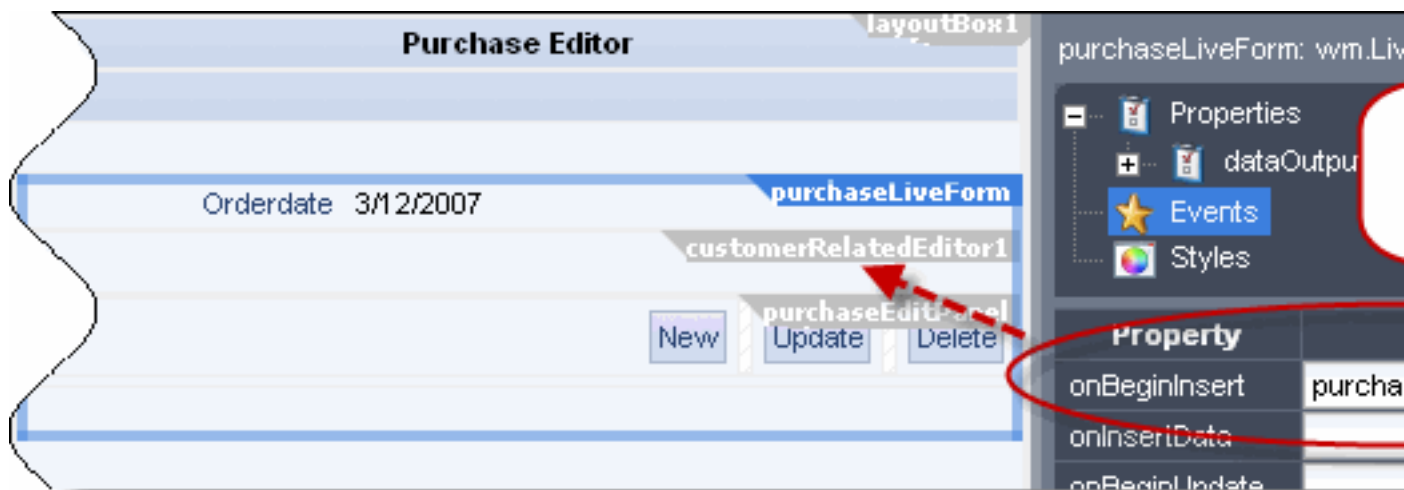
[For more information on Data Navigator widgets](#)

Setting Form Primary and Foreign Keys Automatically

In building an application, it is frequently useful to be able to set the primary or foreign key value of a new data object automatically. Usually, the primary or foreign key editor widget would be hidden on the form so that this happens transparently to the user.

Insert operation clears all field values

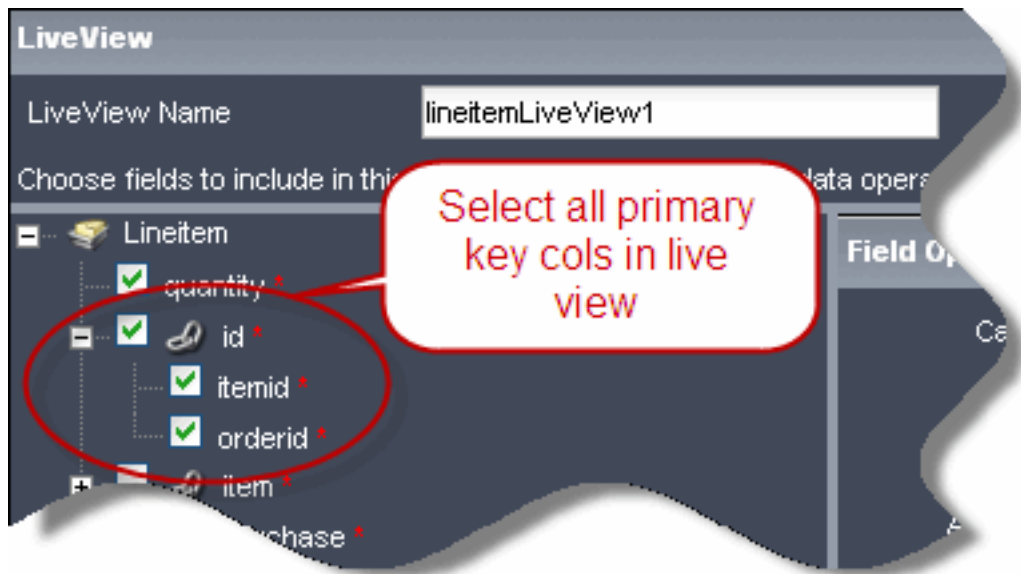
The insert operation on a Live Form clears all the editor values, so it is not possible to use a normal binding to "preset" the value of a primary or foreign key editor. Instead, set the onBeginInsert() event for a form to call a Javascript function, and in that function manually setting the primary or foreign key editor value.



Multi-column primary keys

To work with a multi-column primary key, make sure that the LiveView that is bound to your form is set to display each column of the primary key separately. This will ensure that there is a separate editor in the live form for

each primary key value. Use the onBeginInsert() event to set these editors. The example below shows setting the Lineitem live view to show the two primary key columns, itemid and orderid:



Adding related editor to set foreign key values

To set a foreign key relationship, first add a RelatedEditor widget to the form. For example, add a related editor widget to the Purchase form and set the formField property to "Customer". Next, remove all editors but the primary key editor for Customer, which is "custid".



The steps to set a foreign key value using Javascript are as follows:

1. Set the onBeginInsert event for a live form widget to call a Javascript function. This function will be called before the data is inserted.
2. In the Javascript function that is created, set the appropriate foreign key value. Note that the required foreign key

```
// Called by the onBeginInsert event
for the purchaseLiveForm widget
// Set foreign key

for purchase to currently selected Customer
purchaseLiveFormBeginInsert: function(inSender) {
    // Get primary key value of currently selected customer
```

```

var custid = app.selectedCustomerLiveVar.getValue(
"custid");
    // Set related customer

for
this purchase to currently selected customer

this.custidEditor1.setValue(
"dataValue", custid);
    },

```

Setting lookup editor for foreign key

It is common on inserting a new object with several relationships to allow the user to set those relationships using a lookup editor. To do this, follow these steps:

1. **Add RelatedEditor widget:** add a RelatedEditor widget to the form. For example, add a related editor widget to the Purchase form and set the formField property to "Customer".
2. **Change editingMode parameter to lookup:** with the RelatedEditor widget selected, change the editingMode parameter to lookup.



3. **Remove un-needed editors from RelatedEditor widget:** by default, the RelatedEditor widget shows all editors for the related table. Remove the ones which are not needed (e.g., all the editors except for the lookup editor)
4. **Choose column to display in lookup editor:** the RelatedEditor widget should now just contain a single lookup widget (in this example, customerLookup1). Select the lookup widget, then select the editor property. Now clear the value in the displayExpression property and select the column to display from the displayField property.

Orderdate 4/3/2007

Orderid 1

Customer (lookup) 1

New Update Delete

editor

Events

Styles

caption

Property	Value
autoDataSet	<input checked="" type="checkbox"/>
startUpdate	<input checked="" type="checkbox"/>
displayField	company
displayExpression	
pageSize	20
allowNone	<input type="checkbox"/>

Set the displayField property for the lookup widget and clear the displayExpression

5. **Add onBeginInsert event to form:** select the Purchase form and set the onBeginInsert event to call a Javascript function.
6. **Set default lookup value:** in the onBeginInsert Javascript function, set the lookupEditor's displayValue property to a default value, as shown in the following code:

```
this.addressLookupEditor1.setValue (
    "displayValue",
    "47 MySakila Drive");
```

See Also

- [Developer's Documentation - Databases and Data Models](#)
- [Hibernate Query Language \(HQL\)](#)
- [HQL Tutorial](#)
- [Filters](#)
- [Reimporting a Data Model](#)