Surveybe Freedom: Populating Rosters in Version 5

1. Population Rules

1.1. Introduction

If you configure your questionnaire properly, when you open a new interview, the Implementer is able to automatically create a pre-populated table or tables for your questionnaire. In version 5 the pre-populated table can be a roster table at any level in the table structure or the questionnaire base table itself.

Before you read this document it will help to have an understanding of how in version 5 questionnaire and table identifiers cascade down through the questionnaires table structure. If you haven’t already, please read the Cascading IDs Guide which shows how Cascading ID’s work and explains why they’ve been introduced.

To pre-populate your interview the Implementer needs you to tell it how to find the correct data to populate this table.

This is done by adding a SQL data select statement to the appropriate table in the questionnaire using the Surveybe Designer. The Surveybe Freedom Designer Manual will show you how to create a table and enter its properties. This guide will help you understand what the population SQL does and how to write it.

The Implementer needs to know:

- Which columns from your reference table to use to select the correct rows for the questionnaire (and any parent rosters) from your reference table and to populate your questionnaire and table identifiers
- Which columns from your reference table use to pre-populate which answers in your interview
- Which reference table(s) you want to use
- Any additional filtering criteria you want to apply to the reference data

This seems quite complicated, but the good news is you can either

- Make your SQL simple by carefully selecting and naming your reference data columns or
- Write more sophisticated SQL that will work with non-matching reference data

This document will first explain in some details the general principles you need to understand to construct reference tables and SQL for pre-population. If you are finding the principles difficult to follow you may find reviewing and experimenting with some of the examples found in the subsequent sections helpful.

Section 2. STANDARD PRE-PopULATION SQL below deals with the most common pre-population scenarios. It first explains how to prepopulate the questionnaire base table, first level and sub-
It then explains using the explicit syntax, which makes the full SQL syntax available to the questionnaire designer.

Section 3. ADVANCED PRE-POPULATION SQL covers some additional ways you can use the explicit syntax. It explains how to prepopulate an item list roster and how populate a roster which is the same for every interview.

It’s not possible to document every possible SQL statement that could be used to populate a table, so once you understand the principles you can experiment to see what’s possible!

Section 4. HINTS AND TIPS includes some pointers that will help you use SQL effectively. It’s worth checking this section even if you skip some of the more detailed syntax advice in the previous sections.

### 1.2. Tables You Can Pre-populate
- You can pre-populate any table, including the questionnaire base table
- You can also pre-populate Item list roster tables

### 1.3. Sources You Can Use To Pre-populate
- You can pre-populate any table, from a reference table (or combination of reference tables)
- You can pre-populate any roster table from an item list (but not the questionnaire base table)
- You can use reference table column values directly or use any valid SQL to manipulate column values and/or introduce literals
- You can use roster tables above the table you’re currently populating

### 1.4. When Population Happens
Population happens when you create an interview for the first time. This means a single pre-population statement for each table must return all the rows you need.

The questionnaire base table will be populated first, followed by the first level tables, then the second level tables and so on until all the tables are populated.

If a table is linked to an item list and also has a population statement, the item list will be used to initially populate the roster.

The order of population of tables on the same level can’t be predicted, the database will determine this. This means it’s not safe to write population SQL that refers to other questionnaire tables at the same level or below in the table hierarchy. You can rely on parent tables being populated. SQL that runs after population (for routing, dynamic text and dynamic answers) isn’t affected by this and can address tables at any level.

**You can’t dynamically populate a child table if it doesn’t have a populated parent**, so if you put a dynamic roster below an empty one both will be empty! Similarly if you add rows to parent roster during an interview, new rows in the dynamic child roster won’t be created or populated. You can, of course, still use an item list roster below an empty roster.
1.5. **Using the current. Prefix**

You can’t use `current.` in table population SQL. The `current.` prefix only works where a statement is executed for a single row, whereas the population statement retrieves a set of rows to populate your roster table.

1.6. **Other Pre-population Rules**

The pre-population select statement must provide column values for all\(^1\) the table id columns, and as many questionnaire identifier columns as are required to correctly filter the returned rows (at least one\(^2\)).

The pre-population select statement may return more rows than are required for population, as the selected rows will be filtered by the current questionnaire identifiers. You don’t need to use a WHERE statement.

The pre-population select statement may return columns that don’t exist in the target table. Such columns will be ignored at insert time. For example an extra column may be used to order the rows returned to select a subset\(^3\).

The pre-population select statement for the questionnaire level table must only return a single row after filtering.

1.7. **SQL Conventions**

In the following examples, these conventions are followed to illustrate how to assemble SQL statements:

- Where text appears in italics, like this: \texttt{ReferenceTableName}, you’ll need to substitute the appropriate table, column or variable names from your questionnaire design.
- Clauses in \{brackets\} are optional
- The symbol ..... `following` brackets surrounding a term or clause indicates that the preceding term or clause can be repeated as required. \{, \texttt{IdentifierVariableName3}\} ..... indicates that you may, or may not include a variable name, or several separated by commas.
- The symbol ..... after a term or clause `inside` brackets indicates that the preceding term is the start of a SQL statement that is incomplete in the example given. \{\texttt{WHERE}......\} indicates that you have the option to add a complete WHERE condition to the statement.

\(^1\) Unlike earlier Surveybe versions (versions released before version 5), a value for the target table’s table identifier \texttt{must} be provided. If you don’t provide this, the population SQL will fail at run time in the Implementer. If you don’t have this in your reference data it may be possible to create them in SQL.

\(^2\) The questionnaire identifier can also be generated in SQL if it’s not in your reference data.

\(^3\) For example, if the reference table contains a list of fields, and you only want to return the 3 largest for each household, you can order by field size and select the TOP 3. See section 3.3 \texttt{Dynamic Population Using ‘Top’ And ‘OrderBy’}. 

Where clauses need to be separated by commas you’ll find the comma and the space are part of and precede the clause.

Note that the population SQL, like all of the SQL in Surveybe, is pre-processed by the Surveybe Implementer before it runs; it doesn’t run directly against the referenced tables. This means if you are experienced with using SQL outside the Surveybe environment you may find the SQL required is not quite what you expect!

The examples through this document will get progressively more complicated as the document progresses. The first time you use the document, work through from the beginning so you understand the simpler examples before you move on to the more complicated ones.

You’ll find additional help with the SQL conventions where needed in the Advanced section.
2. **Standard Pre-Population SQL**

2.1. **Pre-Populating With Implicit Syntax:**

Implicit, in this context, means that the SQL population statement does not explicitly reference the identifiers and column required to pre-populate. The Surveybe Implementer automatically identifies the required variables and populates the table. You must make all the required information available in the reference data for this to work.

2.1.1. **Questionnaire Base Table:**

The questionnaire base table is at the top of the table hierarchy and holds all the answers at the questionnaire level, in other words for questions that aren’t on a roster. The syntax to pre-populate the questionnaire base table is:

```
SELECT * FROM ReferenceTableName (WHERE......)
```

Example: SELECT * from Population_RT

<table>
<thead>
<tr>
<th>Population Elements</th>
<th>Element In Example SQL</th>
<th>Identifier/Column Name Used</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>Questionnaire Identifiers</td>
<td>*</td>
<td>RT_Code</td>
</tr>
<tr>
<td>Population</td>
<td>Question Variable Names</td>
<td>*</td>
<td>RT_Population RT_Males</td>
</tr>
<tr>
<td>From</td>
<td>Table Names</td>
<td>FROM Population_RT</td>
<td>Population_RT</td>
</tr>
<tr>
<td>Filter</td>
<td>WHERE criteria</td>
<td>-</td>
<td>(Optional)</td>
</tr>
</tbody>
</table>

The questionnaire base table is populated using this syntax in v5 Population Questionnaire 0.

You need to include all the questionnaire identifiers necessary to identify a single row to be selected from the reference table. You’ll get a runtime error in the Implementer if more than one row is returned.

You can miss out any questionnaire identifiers not needed for this matching, for example an auto-generated identifier, or perhaps an interviewer ID.

2.1.2. **Top Level Roster:**

A top level roster is a roster directly beneath the questionnaire base table, in other words there is no roster table above it in the hierarchy. The syntax is:

```
SELECT * FROM ReferenceTableName (WHERE......)
```

Example: SELECT * FROM City_RT

---

*If you want to populate all interviews without using at least one questionnaire identifier to select the rows you’ll need to use the explicit syntax. See 3.2 Dynamic Population Without Questionnaire Identifiers in the Advanced Pre-Population SQL section below.*
### Population Elements

<table>
<thead>
<tr>
<th>Elements</th>
<th>Element In Example SQL</th>
<th>Identifier/Column Name Used</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>Questionnaire</td>
<td>*</td>
<td>TCountryNum</td>
</tr>
<tr>
<td></td>
<td>Identifiers</td>
<td></td>
<td>Reference table column name(s) must match at least one questionnaire</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>identifier³ (There must be sufficient identifiers to filter rows</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>correctly).</td>
</tr>
<tr>
<td></td>
<td>Table Identifier</td>
<td>*</td>
<td>TCityNum</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A reference table column name must match the target table identifier.</td>
</tr>
<tr>
<td>Population</td>
<td>Question Variable Names</td>
<td>*</td>
<td>TCityName</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Column names must match required Question Variable Names.</td>
</tr>
<tr>
<td>From</td>
<td>Table Names</td>
<td>FROM City_RT</td>
<td>City_RT</td>
</tr>
<tr>
<td>Filter</td>
<td>WHERE criteria</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

In **v5 Population Questionnaire 1** the roster on screen one is populated in using this syntax.

You can use this approach to populate the *questionnaire base table* as well – it’s used on the **v5 Sample Questionnaire** to populate the head teachers name (*HTName*) on the questionnaire base table (*Household*) using the SQL: `SELECT * from r_household`.

#### 2.1.3. **Child Roster**:  
A child roster is a roster beneath another roster, in other words there is at least one parent roster table above it in the hierarchy. The parent table may be a top level roster table, or another child roster. The syntax is:

```
SELECT * FROM ReferenceTableName {WHERE......}
```

Example: `SELECT * FROM Team_RT`

In **v5 Population Questionnaire 2** the additional child roster on screen one is populated in using this syntax.

³ If you want to populate all interviews without using at least one questionnaire identifier to select the rows you’ll need to use the explicit syntax. See 3.2 Dynamic Population Without Questionnaire Identifiers in the **Advanced Pre-Population SQL section below**.
2.2. PRE-POPULATING WITH EXPLICIT SYNTAX

Explicit, in this context, means that the SQL population statement explicitly references all the identifiers and column names required to pre-populate.

You need to be able to use the explicit syntax so that you can pre-populate interviews:

- Where the implicit syntax won’t work
- Where you want to pre-populate with values that don’t come directly from a single reference table
- Where it’s inconvenient to redesign your reference tables

The implicit syntax might not work because:

- Your reference table populates two or more tables
- You want to populate two or more answers from the same reference table column
- Your reference data doesn’t include a questionnaire identifier

If your variable names don’t match, you choose to select data from several tables or you include complex logic, your SQL will need to be longer and more sophisticated. The rest of this document explains what you will need to do if any of the data names don’t match. You can’t mix the two approaches.

The explicit syntax is:

```sql
SELECT ColumnIdentifier1 AS QuestionnaireIdentifier1 {, ColumnIdentifier2 AS Questionnaire Identifier2}......
, ColumnIdentifier1 AS TableIdentifier1 {, ColumnIdentifier2 AS TableIdentifier2}...... ,
, ColumnIdentifier1 AS QuestionVariableName1 {, ColumnIdentifier2 AS QuestionVariableName2}......
FROM ReferenceTableName
{WHERE......}
```

2.2.1. QUESTIONNAIRE BASE TABLE:

If you are populating the questionnaire base table there won’t be a table identifier, so this row would be omitted from the syntax.

To populate our questionnaire base table example using the explicit syntax our SQL would be:

```sql
```
<table>
<thead>
<tr>
<th>Population Elements</th>
<th>Element In Example SQL</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>Questionnaire Identifiers</td>
<td>RT_Code AS Q_Code</td>
</tr>
<tr>
<td>Population</td>
<td>Question Variable Names</td>
<td>RT_Population AS Q_Population, RT_Males AS Q_Males</td>
</tr>
<tr>
<td>From</td>
<td>Table Names</td>
<td>FROM Population_RT</td>
</tr>
<tr>
<td>Filter</td>
<td>WHERE criteria</td>
<td>-</td>
</tr>
</tbody>
</table>

In **v5 Population Questionnaire 0 ES** the roster on screen one is populated in using this syntax.

### 2.2.2. Top Level Roster:

To populate our top level roster example using the explicit syntax our SQL would be:

```sql
SELECT TCountryNum AS TCountryNum, TCityNum AS TCityNum, TCityName AS TCityName FROM City_RT
```

<table>
<thead>
<tr>
<th>Population Elements</th>
<th>Element In Example SQL</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>Questionnaire Identifiers</td>
<td>TCountryNum AS TCountryNum</td>
</tr>
<tr>
<td></td>
<td>Table Identifiers</td>
<td>TCityNum AS TCityNum</td>
</tr>
<tr>
<td>Population</td>
<td>Question Variable Names</td>
<td>TCityName AS TCityName</td>
</tr>
<tr>
<td>From</td>
<td>Table Names</td>
<td>FROM City_RT</td>
</tr>
<tr>
<td>Filter</td>
<td>WHERE criteria</td>
<td>-</td>
</tr>
</tbody>
</table>

You can see this used in **v5 Population Questionnaire 1 ES**

### 2.2.3. Child Roster:

The explicit syntax to populate our child roster example would be:

```sql
SELECT TCountryNum AS TCountryNum, TCityNum AS TCityNum, TNumber AS TNumber, TName AS TName FROM Team_RT
```

<table>
<thead>
<tr>
<th>Population Elements</th>
<th>Element In Example SQL</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>Questionnaire Identifiers</td>
<td>TCountryNum AS TCountryNum</td>
</tr>
<tr>
<td></td>
<td>Table Identifier</td>
<td>TCityNum AS TCityNum, TNumber AS TNumber</td>
</tr>
<tr>
<td>Population</td>
<td>Question Variable Names</td>
<td>TName AS TName</td>
</tr>
<tr>
<td>From</td>
<td>Table Names</td>
<td>FROM Team_RT</td>
</tr>
<tr>
<td>Filter</td>
<td>WHERE criteria</td>
<td>-</td>
</tr>
</tbody>
</table>

---

6 If you want to populate all interviews without using at least one questionnaire identifier to select the rows you’ll need to use the explicit syntax. See 3.2 Dynamic Population Without Questionnaire Identifiers in section 3 Advanced Pre-Population SQL below.
You can see this used in v5 Population Questionnaire 1 ES

2.3. Concise Explicit Syntax

Where the source and target variable names are the same it’s possible to abbreviate the AS clauses to make them easier to type. Doing this to the child roster syntax population statement results in this SQL:

```
SELECT TCountryNum, TCityNum, TNumber, TName FROM Team_RT
```

In our examples above the column identifiers and question variable names are the same. If they are different (say the reference table names are prefixed with R rather than T the full explicit syntax to populate our first top level roster example would look like this:

```
SELECT RCountryNum AS TCountryNum, RCityNum AS TCityNum, RCityName AS TCityName FROM City_RT
```

You can mix abbreviated and full AS clauses in the same statement. In this case the syntax to populate our child roster example might look like this:

```
SELECT RCountryNum AS TCountryNum, RCityNum AS TCityNum, TNumber, TName FROM Team_RT
```

You can see this used in v5 Population Questionnaire 2 ES
3. **Advanced Pre-Population SQL**

You can make your population statement as sophisticated as you wish, exploiting the full power of SQL. This section looks at some straightforward examples of using more advanced SQL to accommodate the common circumstances where the standard approach won’t work.

This document can’t cover all the possible ways in which pre-populating SQL can be exploited, so please post any novel approaches you discover to the [Questionnaire Design](#) section of the Surveybe forum or share them with us. If we think they may be useful to the wider user community we’ll adapt and de-personalise your examples so we can share them on the forum or incorporate them in future versions of this guide.

### 3.1. Pre-Populating An Item List Roster

Version 5 makes it possible to pre-populate an item list roster.

This can be useful where you have a fixed number of rows in your questionnaire that you want to populate from incomplete reference data.

In the example questionnaire [v5 Population Questionnaire 5](#) the household member’s assets table has both an item list and a SQL data Select statement:

![Image of item list and SQL data Select statement](image)

The item list creates a roster for each household member containing 6 rows, one for each of the assets we are interested in.

These 6 rows are then populated from this reference table where suitable corresponding rows can be found:

<table>
<thead>
<tr>
<th>KEYONE</th>
<th>HHMEMID</th>
<th>HHMEMASSETID</th>
<th>ASSETOWNED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Y</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>Y</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>Y</td>
</tr>
</tbody>
</table>
In this case the previously collected reference data contains only data for assets of types 1, 2 and 3, and only has a row if the asset is recorded as owned by the household member. As a result the remaining answers are unticked.

You can use the explicit syntax to achieve the same purpose if you need to. For example:

```
SELECT KeyOne, HHMemID, HHMemAssetID, AssetOwned FROM RefAssets
```

### 3.2. Dynamic Population Without Questionnaire Identifiers

You might want to populate a roster table without filtering the content using a questionnaire identifier as a selection criterion. This would be similar to creating an item list roster.

You’d use this approach instead of an item list to allow you to pre-populate values in the roster or where you want to easily access reference data (for example to input into dynamic text or enablements in the sub-roster).

#### 3.2.1. Top Level Roster Tables

The implementer demands that at least one questionnaire identifier exists in the Pre-population statement, so you can’t use the `SELECT(*)`... syntax. The constraint can be overcome by defining the identifier as itself. Use this syntax to achieve this:

```
SELECT (SELECT QuestionnaireIdentifier1 FROM BaseTableName) AS QuestionnaireIdentifier1
```

```
, ColumnIdentifier1 AS TableIdentifier1 (, ColumnIdentifier2 AS TableIdentifier2)......
```

```
, ColumnIdentifier1 AS QuestionVariableName1 (, ColumnIdentifier2 AS QuestionVariableName2)......
```

```
FROM ReferenceTableName
```

```
{WHERE......}
```

Here’s the example syntax used in **v5 Population Questionnaire 3**:

```
SELECT (SELECT QID FROM BaseTable) AS QID, Shirt AS TTRT, FamilyName AS Player_Name FROM RefTable
```

<table>
<thead>
<tr>
<th>Population Elements</th>
<th>Element In Example SQL</th>
<th>Constraints/Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td></td>
<td>Reference table column name(s) must match at least one questionnaire identifier. <strong>As our reference data doesn't include a questionnaire identifier we've used one from the interview.</strong></td>
</tr>
<tr>
<td><strong>Table Identifier</strong></td>
<td></td>
<td>Reference table column names must be defined for all of the parent table identifiers at levels above the target table. <strong>We are populating a top level roster table so this isn’t required.</strong> A reference table column name must be defined as the target table identifier.</td>
</tr>
<tr>
<td>Population</td>
<td></td>
<td>Column names must be defined as all the required Question Variable Names</td>
</tr>
<tr>
<td>From</td>
<td>FROM RefTable</td>
<td>(Optional)</td>
</tr>
<tr>
<td>Filter</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 3.2.2. **Sub-Roster Tables**

If a parent roster is populated, you can populate a sub-roster with non-interview specific rows.

If the parent roster is empty you’ll have to use an Item List Roster, and use reference tables and dynamic text to display any values as you would have done in earlier versions of Surveybe.

In order to populate the table for the sub-roster the Implementer needs:

- at least one questionnaire identifier
- the parent roster row identifiers

This can be achieved by creating a temporary SQL table containing the required population data. The temporary table will provide a set of child rows for each parent row. The following syntax illustrates how this would be achieved at the first level of sub-rosters.

This SQL uses a cross join to create the temporary table:

```sql
SELECT p.ParentTableIdentifier,
       r.ColumnIdentifier1, {r.ColumnIdentifier2}......
FROM ReferenceTableName r JOIN ParentTableName p
{WHERE......}
```

This table contains every possible combination of city and player.

This full syntax, populates the sub roster table from the temporary table:

```sql
SELECT (SELECT QuestionnaireIdentifier1 FROM BaseTableName) AS QuestionnaireIdentifier1,
       ParentTableIdentifier,
       ColumnIdentifier1 AS ChildTableIdentifier,
       TempColumnIdentifier2 AS QuestionVariableName1, {ColumnIdentifier3 AS QuestionVariableName2}......
FROM (SELECT p.ParentTableIdentifier,
       r.ColumnIdentifier1, {r.ColumnIdentifier2}......
FROM ReferenceTableName r JOIN ParentTableName p
{WHERE......})
{WHERE......}
```

The second `SELECT` creates an intermediate table by joining the parent table and the reference table. The first `SELECT` reads this to generate the rows to populate your child roster.
The use or \textit{r} and \textit{p} after the reference and parent table names indicates the use of a single lower case character that can then be used to substitute for the preceding table name. So \(p\text{.ParentTableName}\) expands to become \textit{ParentTableName.ParentTableIdentifier}. This makes it clear to SQL (and to anyone reading the statement) the table from which the column was taken. It’s good practice to use these prefixes even if they’re not always strictly necessary.

Here’s the example syntax used in \texttt{v5 Population Questionnaire 4}:

\begin{verbatim}
SELECT (SELECT QID FROM BaseTable) AS QID, CT, Shirt AS TT, FamilyName AS FName
FROM (SELECT c.CT, t.Shirt, t.FamilyName FROM Team_RT t JOIN CityTable c)
\end{verbatim}

<table>
<thead>
<tr>
<th>Population Elements</th>
<th>Element In Example SQL</th>
<th>Constraints/Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection Questionnaire Identifiers</td>
<td>(\text{SELECT (SELECT QID FROM BaseTable) AS QID})</td>
<td>Reference table column name(s) must match at least one questionnaire identifier. \textit{As our reference data doesn’t include a questionnaire identifier we’ve used one from the interview.}</td>
</tr>
<tr>
<td>Parent Table Identifier</td>
<td>(\text{CT})</td>
<td>Reference table column names must be defined for all of the parent table identifiers at levels above the target table.</td>
</tr>
<tr>
<td>Table Identifier</td>
<td>(\text{Shirt AS TT})</td>
<td>A reference table column name must be defined as the target table identifier.</td>
</tr>
<tr>
<td>Population Question Variable Names</td>
<td>(\text{FamilyName AS FName})</td>
<td>Column names must be defined as all the required Question Variable Names.</td>
</tr>
<tr>
<td>From Table Names</td>
<td>(\text{FROM (SELECT c.CT, t.Shirt, t.FamilyName FROM Team_RT t JOIN CityTable c)})</td>
<td>This join creates a temporary table from which the outer SELECT statement populates the roster table.</td>
</tr>
<tr>
<td>Filter</td>
<td>(\text{WHERE criteria})</td>
<td>(Optional)</td>
</tr>
</tbody>
</table>

Please note that this SQL won’t work if you use it in the data browser because it uses the parent roster table, \texttt{CityTable}. This table will be empty in the Designer (unless you’ve imported an interview). To create a similar temporary table try using this SQL, this uses the reference table \texttt{City_RT} instead:

\begin{verbatim}
SELECT c.CityCode, c.CityName, t.Shirt, t.FamilyName FROM Team_RT t JOIN City_RT c
\end{verbatim}

You’ll need to adapt the outer part of the SQL as well to run the whole statement as the variable names have changed as well!

You can populate rosters lower down in the hierarchy by joining more than one table if you need to.

When using joins, particularly Cross joins (sometimes called Cartesian joins) like the one above, very large tables can be created – these can take time to process, so please try to avoid using these joins where they are not needed.
3.3. Dynamic Population Using ‘TOP’ and ‘ORDER BY’

This example shows you how you can use SQL to manipulate the selected pre-population data by relatively complex criteria. It also shows how using the Data Browser can be really helpful when constructing more sophisticated SQL. Refer to the example questionnaire: v5 Population Questionnaire 6.

The requirement is to ask questions about the 3 types of livestock on the farm that have the greatest total value. We have previous round data which shows counts of the animals we’re interested in:

Table 1: Livestock table (RT_LIVESTOCK)

<table>
<thead>
<tr>
<th>Farm_ID</th>
<th>Livestock_Code</th>
<th>Livestock_Type</th>
<th>Livestock_Owned</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>1</td>
<td>Goats</td>
<td>2</td>
</tr>
<tr>
<td>101</td>
<td>2</td>
<td>Sheep</td>
<td>0</td>
</tr>
<tr>
<td>101</td>
<td>3</td>
<td>Chickens</td>
<td>6</td>
</tr>
<tr>
<td>101</td>
<td>4</td>
<td>Cows</td>
<td>1</td>
</tr>
<tr>
<td>101</td>
<td>5</td>
<td>Horses</td>
<td>20</td>
</tr>
<tr>
<td>102</td>
<td>1</td>
<td>Goats</td>
<td>25</td>
</tr>
<tr>
<td>102</td>
<td>2</td>
<td>Sheep</td>
<td>0</td>
</tr>
<tr>
<td>102</td>
<td>3</td>
<td>Chickens</td>
<td>6</td>
</tr>
<tr>
<td>102</td>
<td>4</td>
<td>Cows</td>
<td>1</td>
</tr>
<tr>
<td>102</td>
<td>5</td>
<td>Horses</td>
<td>0</td>
</tr>
<tr>
<td>103</td>
<td>1</td>
<td>Goats</td>
<td>0</td>
</tr>
<tr>
<td>103</td>
<td>2</td>
<td>Sheep</td>
<td>50</td>
</tr>
<tr>
<td>103</td>
<td>3</td>
<td>Chickens</td>
<td>12</td>
</tr>
<tr>
<td>103</td>
<td>4</td>
<td>Cows</td>
<td>1</td>
</tr>
<tr>
<td>103</td>
<td>5</td>
<td>Horses</td>
<td>1</td>
</tr>
<tr>
<td>104</td>
<td>1</td>
<td>Goats</td>
<td>2</td>
</tr>
<tr>
<td>104</td>
<td>2</td>
<td>Sheep</td>
<td>0</td>
</tr>
<tr>
<td>104</td>
<td>3</td>
<td>Chickens</td>
<td>0</td>
</tr>
<tr>
<td>104</td>
<td>4</td>
<td>Cows</td>
<td>0</td>
</tr>
<tr>
<td>104</td>
<td>5</td>
<td>Horses</td>
<td>0</td>
</tr>
</tbody>
</table>

We also have reference data showing the base value of each type of animal:

Table 2: Livestock Values (RT_Livestock_Value).

<table>
<thead>
<tr>
<th>Livestock_Code</th>
<th>Livestock_Type</th>
<th>Livestock_Unit_Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Goats</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>Sheep</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>Chickens</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>Cows</td>
<td>250</td>
</tr>
<tr>
<td>5</td>
<td>Horses</td>
<td>200</td>
</tr>
</tbody>
</table>

To build the fastest possible SQL, we’d combine this data outside Surveybe, but the volume of data is very small and Surveybe will cope with large reference tables so we can build SQL to manage this all for us.

Import v5 population Questionnaire 6 into the Designer and use the Data Browser to view the SQL as we go along. Don’t forget to use SET SCHEMA MATERIALIZED before you start entering other SQL statements.
First we’ll assemble the set of columns we need to pre-populate the roster. When we’ve done that we can look at how to select the rows we want for the roster.

First, we’ll need to calculate the total value of each type of animal. We’ll use this statement to multiply the number of animals by the unit value, creating a column called Livestock_Type_Value:

```
SELECT (SELECT Livestock_Unit_Value FROM RT_Livestock_Value WHERE RT_Livestock_Value.Livestock_Code = RT_Livestock.Livestock_Code) * Livestock_Owned
AS Livestock_Type_Value FROM RT_LIVESTOCK
```

You can run the preceding statement in the data browser. The first part:

```
(SELECT Livestock_Unit_Value FROM RT_Livestock_Value WHERE RT_Livestock_Value.Livestock_Code = RT_Livestock.Livestock_Code)
```

Finds the correct unit value (Livestock_Unit_Value) for the type of livestock (Livestock_Code) from the reference table holding the livestock values (RT_Livestock_Value).

This is then multiplied by the number of animals (Livestock_Owned) calculating a total value for each row in the Livestock table (RT_LIVESTOCK).

We can then assemble the other columns we’ll need to populate the roster:

```
SELECT Farm_ID, Livestock_Code AS LivestockID, Livestock_type, Livestock_Owned,
(SELECT Livestock_Unit_Value FROM RT_Livestock_Value WHERE RT_Livestock_Value.Livestock_Code = RT_Livestock.Livestock_Code) AS L_Unit_Value,
AS Livestock_Type_Value FROM RT_LIVESTOCK
```

This too will run in the data browser, the first part of output looks like this:

<table>
<thead>
<tr>
<th>FARM_ID</th>
<th>LIVESTOCK_ID</th>
<th>LIVESTOCK_TYPE</th>
<th>LIVESTOCK_OWNED</th>
<th>L_UNIT_VALUE</th>
<th>LIVESTOCK_TYPE_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>1</td>
<td>Goats</td>
<td>2</td>
<td>25</td>
<td>50</td>
</tr>
<tr>
<td>101</td>
<td>2</td>
<td>Sheep</td>
<td>0</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>101</td>
<td>3</td>
<td>Chickens</td>
<td>6</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>101</td>
<td>4</td>
<td>Cows</td>
<td>1</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>101</td>
<td>5</td>
<td>Horses</td>
<td>20</td>
<td>200</td>
<td>4000</td>
</tr>
<tr>
<td>102</td>
<td>1</td>
<td>Goats</td>
<td>25</td>
<td>25</td>
<td>625</td>
</tr>
<tr>
<td>102</td>
<td>2</td>
<td>Sheep</td>
<td>0</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>102</td>
<td>3</td>
<td>Chickens</td>
<td>6</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>102</td>
<td>4</td>
<td>Cows</td>
<td>1</td>
<td>250</td>
<td>250</td>
</tr>
</tbody>
</table>

We don’t need all of these columns to populate the roster, but their presence makes it easier to see what’s going on. They can be (and have been) left in the population statement.
To select the 3 most valuable livestock for each farm we need to sort our result set using:

```
ORDER BY Livestock_Type_Value DESC NULLS LAST
```

Change the `SELECT` at the beginning of the statement to:

```
SELECT TOP 3
```

And eliminate any rows where the farm doesn’t have any animals by including:

```
WHERE Livestock_Owned > 0
```

**IMPORTANT:** When this SQL is running in an interview, we’ll only want to see the rows for an individual farm. This would normally be filtered automatically based on the Questionnaire Identifier. Restricting the selection using `TOP` means this doesn’t work, so this will need to be added explicitly to the `WHERE` statement when it’s in the questionnaire:

```
AND Farm_ID = (SELECT FarmerNum FROM BaseTable)
```

Try appending this to the statement in the data browser to do the same thing:

```
WHERE Farm_ID = 103
```

Change the `Farm_ID` to see results for other farms.

This gives us a complete statement like this:

```
SELECT TOP 3 Farm_ID, Livestock_Code AS LivestockID, Livestock_type, Livestock_Owned,
(SELECT Livestock_Unit_Value FROM RT_Livestock_Value WHERE
RT_Livestock_Value.Livestock_Code = RT_Livestock.Livestock_Code) AS L_Unit_Value,
(SELECT Livestock_Unit_Value FROM RT_Livestock_Value WHERE
AS Livestock_Type_Value FROM RT_LIVESTOCK WHERE
Farm_ID = 101 AND
Livestock_Owned > 0 ORDER BY Livestock_Type_Value DESC NULLS LAST
```

This statement will run in the Data Browser and produces this output:

<table>
<thead>
<tr>
<th>FARM_ID</th>
<th>LIVESTOCKID</th>
<th>LIVESTOCK_TYPE</th>
<th>LIVESTOCK_OWNED</th>
<th>L_UNIT_VALUE</th>
<th>LIVESTOCK_TYPE_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>6</td>
<td>Horse</td>
<td>20</td>
<td>200</td>
<td>1000</td>
</tr>
<tr>
<td>101</td>
<td>4</td>
<td>Cows</td>
<td>1</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
<td>Goats</td>
<td>2</td>
<td>25</td>
<td>50</td>
</tr>
</tbody>
</table>

Try running it again for Farm 104 – you’ll see only one row is returned as this farm has only goats – on the other rows `Livestock_Owned` is zero.

Note that while the rows appear in the data browser in order of `Livestock_Type_Value`, in the Implementer they will appear in `LivestockID` order, like this:
This table below breaks down this SQL (the statement actually used in the questionnaire) in the same way as previous examples:

```
SELECT TOP 3 Farm_ID, Livestock_Code AS LivestockID, Livestock_type, Livestock_Owned,
(SELECT Livestock_Unit_Value FROM RT_Livestock_Value WHERE RT_Livestock_Value.Livestock_Code = RT_Livestock.Livestock_Code) AS L_Unit_Value,
(SELECT Livestock_Unit_Value FROM RT_Livestock_Value WHERE RT_Livestock_Value.Livestock_Code = RT_Livestock.Livestock_Code) * Livestock_Owned AS Livestock_Type_Value FROM RT_LIVESTOCK WHERE Livestock_Owned > 0 AND Farm_ID = (SELECT FarmerNum FROM BaseTable) ORDER BY Livestock_Type_Value DESC NULLS LAST
```

This SQL is virtually the same as that used in the Data Browser, the difference is that it uses the questionnaire identifier to select the farm number like this:

```
Farm_ID = (SELECT FarmerNum FROM BaseTable)
```

Instead of explicitly referencing it, like this:

```
Farm_ID = 101
```
<table>
<thead>
<tr>
<th>Population Elements</th>
<th>Element In Example SQL</th>
<th>Constraints/Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>Qualifier</td>
<td></td>
</tr>
<tr>
<td>TOP 3</td>
<td></td>
<td>Limits selection to first three rows in results table.</td>
</tr>
<tr>
<td>Questionnaire</td>
<td>Identifiers</td>
<td>Reference table column name(s) must match at least one questionnaire identifier.</td>
</tr>
<tr>
<td>TOP 3</td>
<td></td>
<td>Reference table column names must be defined for all of the parent table identifiers at levels above the target table. <strong>We are populating a top level roster table so this isn’t required.</strong> A reference table column name must be defined as the target table identifier.</td>
</tr>
<tr>
<td>Table Identifier</td>
<td>Livestock_Code AS LivestockID</td>
<td></td>
</tr>
<tr>
<td>Population</td>
<td>Question Variable Names</td>
<td></td>
</tr>
<tr>
<td>Livestock_type,</td>
<td></td>
<td>Column names match the required Question Variable Names.</td>
</tr>
<tr>
<td>Livestock_Owned,</td>
<td></td>
<td>Populates L_Unit_Value for display in the Data Browser, column not used in questionnaire.</td>
</tr>
<tr>
<td>(SELECT Livestock_Unit_Value FROM RT_Livestock_Value WHERE RT_Livestock_Value.Livestock_Code = RT_Livestock.Livestock_Code) AS L_Unit_Value,</td>
<td></td>
<td>Calculates total value for livestock type. Populates hidden question in questionnaire (i.e. value appears in exported data, but not visible to interviewer).</td>
</tr>
<tr>
<td>(SELECT Livestock_Unit_Value FROM RT_Livestock_Value WHERE RT_Livestock_Value.Livestock_Code = RT_Livestock.Livestock_Code) * Livestock_Owned AS Livestock_Type_Value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>From</td>
<td>Table Names</td>
<td>RT_LIVESTOCK</td>
</tr>
<tr>
<td>Filter</td>
<td>WHERE criteria</td>
<td>Livestock_Owned &gt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Farm_ID = (SELECT FarmerNum FROM BaseTable)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ORDER BY Livestock_Type_Value DESC NULLS LAST</td>
</tr>
</tbody>
</table>

7 Omitting this clause shouldn’t change the way the SQL works as the column it generates isn’t used in the pre-population.  
8 I’ve chosen to include the total value in the exported data using a hidden question, but this question could be omitted. The clause has to remain in the statement as the value it generates is needed for the ORDER BY.
4. **Hints and Tips**

4.1. **Exploit SQL’s Flexibility**

You can substitute any valid SQL expression as the left hand side of an AS clause to pre-populate an answer in your questionnaire. This can be a SELECT statement or a literal (string or integer as appropriate), and can address any of the tables available to the questionnaire at the time this table is being built (reference tables and any questionnaire tables above the one currently being populated).

4.2. **Keep it Simple ... If You Can**

Although you can make your SQL very sophisticated, we would advise users to keep pre-population statements as simple as possible. Complex SQL running on large reference tables can increase the time taken when an interview is first created. If it’s practical, adapt your reference tables to minimise this overhead where you can.

4.3. **Take Care When Joining**

If you want to select values from more than one table, and your tables are large, try to avoid accidentally making Cartesian Joins which will increase the time taken to create an interview, select each field individually as in example A rather than using syntax B.

<table>
<thead>
<tr>
<th>A</th>
<th>SELECT IdentifierVariableName1, IdentifierVariableName2, (SELECT ColumnIdentifierB FROM ReferenceTableNameB WHERE ReferenceTableNameB.ColumnIdentifierA = ReferenceTableNameA.ColumnIdentifier3) AS IdentifierVariableName3 FROM ReferenceTableNameA</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>SELECT IdentifierVariableName1, IdentifierVariableName2, ReferenceTableNameB. ColumnIdentifierB AS IdentifierVariableName3 FROM ReferenceTableNameA, ReferenceTableNameB WHERE ReferenceTableNameB.ColumnIdentifierA = ReferenceTableNameA.ColumnIdentifier3</td>
</tr>
</tbody>
</table>

Although you may need to adapt the syntax, you can test your joins (or any of your SQL) using the Data Browser - take a look at the [Introduction To The Data Browser](#) document. Remember the browser uses ‘pure’ SQL, so you may need to adapt your statements to the different environments. The introduction document provides advice on this.